

**DİZİ İFADELERİNE DAYANAN YENİ BİR OPTİKSEL KARAKTER
TANIMA ALGORİTMASI**

Derya YILMAZ

85921

**YÜKSEK LİSANS TEZİ
ELEKTRONİK-BİLGİSAYAR EĞİTİMİ**

**TC. YÜKSEKÖĞRETİM ENSTİTÜSÜ
DOKÜMANTASYON MERKEZİ**

**GAZİ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ**

Eylül 1999

ANKARA

Derya YILMAZ tarafından hazırlanan DİZİ İFADELERİNE DAYANAN YENİ BİR OPTİKSEL KARAKTER TANIMA ALGORİTMASI adlı bu tezin Yüksek Lisans tezi olarak uygun olduğunu onaylarım.



Doç.Dr. Nihal Fatma GÜLER
Tez Yöneticisi

Bu çalışma, jürimiz tarafından Elektronik Eğitimi Anabilim Dalında Yüksek Lisans tezi olarak kabul edilmiştir.

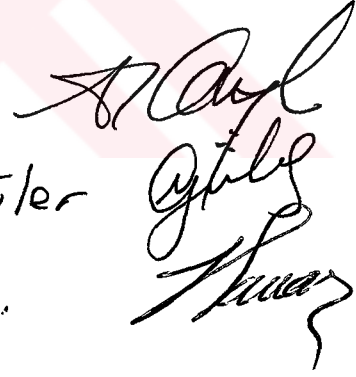
Başkan : Doç. Dr. Abdullah ÇAVUŞOĞLU

Üye : Doç.Dr. Nihal Fatma Güler

Üye : Yrd.Doç.Dr. Yılmaz Savaş

Üye : _____

Üye : _____



Bu tez, Gazi Üniversitesi Fen Bilimleri Enstitüsü tez yazım kurallarına uygundur.



İÇİNDEKİLER

	Sayfa
ÖZET	i
ABSTRACT	ii
TEŞEKKÜR	iii
ÇİZELGELERİN LİSTESİ	iv
ŞEKİLLERİN LİSTESİ	vi
SİMGELER VE KISALTMALAR	viii
1. GİRİŞ	1
2. KARAKTER TANIMA ALGORİTMALARINA GENEL BİR BAKIŞ	3
3. KARAKTER ZAYIFLATMA VE KULLANILAN KARAKTER ZAYIFLATMA ALGORİTMASI	8
4. DERYA KARAKTER TANIMA ALGORİTMASI (DECRA) ...	11
4.1. Karakterin Yeniden Yapılandırılması	12
4.2. Karakterin Diziler ve Diziler Arası Bağlantılarla Tanımlanması	16
4.2.1. Pixellerin numaralandırılması ve dizilerin oluşturulması mantığı	19
4.2.2. Dizilerin diğer dizilerle olan ikili bağlantılarının çıkarılması	22
4.2.3. Piramit yapılı hatların düzgünleştirilmesi	24
4.2.4. Karakter bilgisi taşımayan üç elemanlı gereksiz dizilerin silinmesi	29
4.2.4.1. Çapraz(/backslash) diziler üzerindeki işlemler	29

	Sayfa
4.2.4.2. Düz hatlı(vertical/horizontal) diziler üzerindeki işlemler	30
4.2.5. Üç ve iki elemanlı diziler üzerindeki işlemler	32
4.2.5.1. Çıkıntı durumundaki üç elemanlı gereksiz dizilerin silinmesi	32
4.2.5.2. Bağlantı olarak mevcut olan gereksiz ikili dizilerin silinmesi	33
4.2.6. Aynı yöne sahip dizilerin iki elemanlı dizi yardımıyla birleştirilmesi	34
4.2.7. Kırılması gereken dizilerin kırılması	35
4.2.8. Çıkıntı durumundaki iki elemanlı gereksiz dizilerin silinmesi	36
4.2.9. Karakterden çıkarılan tüm dizilerin sıralanması	37
4.2.10. Diziler arası bağlantıların ve bağlantı konumlarının tespit edilmesi	37
4.3. Database ve Karakter Tanıma	40
4.3.1. Database ve database yapısı	42
4.3.2. Segmentasyon ve tanıma	46
5. SONUÇ VE ÖNERİLER	48
KAYNAKLAR	51
EKLER	53
ÖZGEÇMİŞ	121

**DİZİ İFADELERİNE DAYANAN YENİ BİR OPTİKSEL KARAKTER
TANIMA ALGORİTMASI
(Yüksek Lisans Tezi)**

Derya YILMAZ

**GAZİ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
Eylül 1999**

ÖZET

Bu çalışmada karakter tanıma (OCR) uygulamaları için yeni bir tanıma algoritması ve database yapısı geliştirilmiştir. DECRA adı verilen tanıma algoritması, iskelet halindeki karakteri, “dizi” adı verilen pixel grupları ve bu diziler arasındaki bağlantı tanımları ile ifade eder. DECRA algoritması, karmaşık matematiksel ifadeler kullanmadan basit şekilde istendik karakter bilgileri üretir. Database’in yapısı ise burada ortaya çıkan karakter bilgilerine göre oluşturulmuştur. Database’de karakterler “segment” denilen parçalar ve segmentlere ait özelliklerle tanımlanmışlardır. DECRA algoritmasının çıkışından alınan karakter bilgileri segmente edilir. Bu bilgiler database’de tanımlı karakter bilgileri ile karşılaştırılır ve tanıma işlemi gerçekleşir. Algoritmanın doğruluğu teorik ve uygulamalı olarak test edilmiş ve hatalı tanımayla karşılaşılmamıştır. Teorik olarak çok çeşitli gürültü seviyelerindeki bozuk karakterler incelenmiş ve algoritma bu tür karakterleri tanıyacak şekilde geliştirilmiştir. Uygulamada ise doküman tanınması yapılmadığından, düzgün karakterler üzerinde çalışılmıştır.

Bilim Kodu :646.04.01
Anahtar Kelimeler :Karakter tanıma, zayıflatma
Sayfa Adedi :121
Tez Yöneticisi :Doç.Dr.F.Nihal GÜLER

A NEW OPTICAL CHARACTER RECOGNITION ALGORITHM BASED ON STRING EXPRESSIONS

(M.Sc. Thesis)

Derya YILMAZ

**GAZİ UNIVERSITY
INSTITUTE OF SCIENCE AND TECHNOLOGY
September 1999**

ABSTRACT

In this thesis, a new character recognition algorithm and database structure was developed for recognition of character's applications (OCR-Optical Character Recognition). The new algorithm; called DECRA, expresses character in skeleton form, pixel groups called "string" and relationships among the strings. DECRA algorithm produces the requested character information in a simple way that doesn't require complicated mathematical expressions. Structure of the database was built according to the character informations that arised from DECRA. The characters in the database were defined by segments and their properties. The character information from the output of DECRA were segmented. This information is compared with the characters defined in the database and recognition is realized. Performance of the algorithm was tested theoriectically and practically. The algorithm has worked correctly. Various characters with theoriectically different noise level were studied and the algorithm was developed to recognize these kinds of characters. However, since no definition was made for documents, characters without noise were studied in practice.

Bilim Kodu : 646.04.01
Anahtar Kelimeler : Character recognition, skeletion
Sayfa Adedi : 121
Tez Yöneticisi : Doç.Dr.F.Nihal GÜLER

TEŞEKKÜR

Bu çalışma sırasında yardım ve önerilerinden faydalandığım tez danışmanım Sayın Doç.Dr. Nihal Fatma GÜLER'e, uygulama alanında destek olan İbrahim GÜMÜŞ'e, manevi destekleriyle beni hiçbir zaman yalnız bırakmayan eşim E.Hakan YILMAZ'a, aileme ve arkadaşlarıma teşekkür ederim.



ÇİZELGELERİN LİSTESİ

Çizelge	Sayfa
Çizelge 4.1.	Şekil 4.8’de gösterilen yön numaralarına karşılık gelen yönlerin isimleri 20
Çizelge 4.2.	Şekil 4.9’daki “A” karakterinden çıkarılan ilk diziler 22
Çizelge 4.3.	Çizelge 4.2’de gösterilen “A” karakterinden çıkarılan ilk dizilerin bağlantılı oldukları diziler (Co_Scons) 23
Çizelge 4.4.	“+” karakterinden çıkarılan ilk diziler 28
Çizelge 4.5.	Piramit yapılı hatların düzgünleştirilmesi aşamasından sonra “+” karakterine ait diziler 29
Çizelge 4.6.	Şekil 4.12’deki yıldız “*” karakterinden çıkarılan tüm diziler 32
Çizelge 4.7.	Şekil 4.12’de gösterilen yıldız “*” karakterinin silinecek gereksiz iki elemanlı dizileri 34
Çizelge 4.8.	Yıldız “*” karakterinin dört dizisinin orta noktalarından kırılmasıyla oluşan sekiz dizi 36
Çizelge 4.9.	Bir dizinin diğer dizilerle bağlantılarını gösterir FF,FL,LF,LL dizileri 38
Çizelge 4.10.	“X” karakterinden çıkarılan dizilerin sıralanmış son durumu 38
Çizelge 4.11.	“X” karakterine ait dizilerin diğer dizilerle olan bağlantıları ve bağlantı konumları 39
Çizelge 4.12.	Şekil 4.16’da görülen beş segmente ayrılmış “A” karakterinin her bir segmentinin database’e yerleştirilen özellikleri 43

Çizelge	Sayfa
Çizelge 4.13.	Şekil 4.17’de görülen dört segmente ayrılmış “X” karakterinin her bir segmentinin database’e yerleştirilen özellikleri 45
Çizelge 4.14.	Şekil 4.18’de görülen beş segmente ayrılmış “X” karakterinin her bir segmentinin database’e yerleştirilen özellikleri 46



ŞEKİLLERİN LİSTESİ

Şekil	Sayfa
Şekil 2.1. Çeşitli gürültü seviyelerindeki tanıma sonuçları (Bose and Kou'dan, 1994)	4
Şekil 2.2. Dört giriş dokümanı ve onların tanıma sonuçları (Kim and Park'dan, 1995)	5
Şekil 2.3. FRSOM sınıflayıcı tarafından doğru olarak sınıflandırılmış el yazısı rakamların örnekleri (Chi et al.'dan, 1994)	7
Şekil 3.1. Tanıma sisteminin genel yapısı	9
Şekil 3.2. Bir "D" karakterinin BLISS ile zayıflatılmış şekli	10
Şekil 4.1. Derya karakter tanıma algoritmasının genel blok Diyagramı (DECRA)	11
Şekil 4.2. "□" sembolü ile gösterilen boşluğun doldurulmadığı pixel yerleşimi	13
Şekil 4.3. Pixel yerleşim durumları-1	13
Şekil 4.4. Pixel yerleşim durumları-2	14
Şekil 4.5. Pixel yerleşim durumları-3	15
Şekil 4.6. Karakterin yeniden yapılandırılması aşamasına ait bir "M" karakteri örneği	16
Şekil 4.7. Karakterin diziler ve diziler arası bağlantılarla tanımlanması aşamasının işlem sırasını gösterir blok diyagram	18
Şekil 4.8. Dolu bir "X" pixeli ve sekiz yöndeki komşulukları	19
Şekil 4.9. Zayıflatılmış bir "A" karakterinin numaralandırılarak dizilere atılması	21
Şekil 4.10. Piramitsel pixel yapıları	24

Şekil	Sayfa
Şekil 4.11. Piramitsel yapılı hatların düzgünleştirilmesi aşamasının “+” karakterine uygulanması	28
Şekil 4.12. Düzgün bir yıldız “*” karakterinden çıkarılan üç elemanlı gereksiz dizi örneği	31
Şekil 4.13. Yıldız “*” karakterinden elde edilen dizilere göre karakterin tanımlı son durumu	35
Şekil 4.14. Bir “X” karakterinin tüm aşamalar sonrası örneklenmesi	39
Şekil 4.15. Database’in genel yapısı	41
Şekil 4.16. Bir “A” karakterinin segmentlere ayrılmış hali	42
Şekil 4.17. Bir “X” karakterinin segmentlere ayrılmış hali	44
Şekil 4.18. Değişik bir “X” karakterinin segmentlere ayrılmış hali	44

SİMGELER VE KISALTMALAR

Bu çalışmada kullanılan simgeler ve kısaltmalar, açıklamaları ile birlikte aşağıda sunulmuştur.

Simgeler

Açıklama

F	First, başlangıç noktası
FF	First-First bağlantı
FL	First-Last bağlantı
L	Last, bitiş noktası
LF	Last-First bağlantı
LL	Last-Last bağlantı
Nil	Bağılantısızlık
x	Dolu bir pixel

Kısaltmalar

BLISS	Boundary Lookout Information and Soft Sweep
DECRA	Derya Character Recognition Algorithm
FRSOM	Fuzzy Rulers and Self Organizing Maps
OCR	Optical Character Recognition
SOM	Self Organizing Maps

1. GİRİŞ

Karakter Tanıma, karakterlerin belirli özelliklerinin ve temel yapısının; database'deki karakter bilgileri ile karşılaştırılması esasına dayanır. Database'de; karakterlere ait özellik bildiren referans bilgiler vardır. Bunun için karakteri diğerlerinden ayıran belirli özelliklere ve bu özelliklerin korunduğu temel karakter yapısına gerek vardır. Çünkü bir karakterin, dokümandan alındığı şekliyle işlenmesi çok zor ve pixel yerleşimleri de oldukça karışıktır.

Bugün karakter tanıma sistemleri elle yapılan yoğun tuşlamalar olmaksızın bilgisayar sistemine, bilgi transferini kolaylaştır (Liang et al.,1994). Karakter tanıma uygulamaları, çok çeşitli alanlar üzerinde yapılmıştır. Bunlar arasında arap, çin karakterleri, el yazısı, matbaa harfleri, rakamlar, semboller ve karakter sayılmayan diğer simgeler sayılabilir. Algoritmaların özellikleri tanıma yapılacak alana göre değişim gösterir. Tanıma uygulamalarında, algoritmanın doğruluk yüzdesini yükseltmeye çalışmanın verdiği zorluklarla karşılaşılır. En büyük problem bazı karakterlerin zayıflatılmış şekillerinin birbirine çok benzemesidir. Örneğin zayıflatma sonrasında “8” rakamı ile “B” harfi birbirine çok benzer. Bu nedenle tanıma algoritmasında karakter özelliklerinin dikkatli bir şekilde tespit edilmesi gerekir. Tanımadaki diğer bir problem de nokta ve çizgili karakterlerin özelliklerini belirten bu tür parçalarının tarama ya da zayıflatma sırasında kaybolmasından dolayı ortaya çıkar ki bu tipteki hataların tanıma algoritması ile önlenmesi mümkün değildir.

Karakter Tanıma, bir başka yaklaşım tarzıyla şekil tanımı değişik alanlarda kullanılmaktadır. Örneğin parmak izi, sayfa, şekil, v.b. tanımalarda kullanılan prensip aynıdır. Biraz daha genişletilerek değişik alanlarda; sinyal izleme ve resim tanımda da aynı prensipden faydalanılır.

Karakter tanınması deęişik şekillerde yapılabilmektedir. Kullanılan bazı algoritmalar sınır gözetlemeli olup bu tür algoritmalarda karakterin zayıflatılmasına gerek yoktur. Diğerlerinde ise karakterin doğru bir iskeletine ihtiyaç duyulmaktadır. Derya Algoritmasında da karakterin zayıflatılması gerekmektedir. Bu nedenle BLISS (İ.Gümüş, 1998) adı verilen yeni bir zayıflatma algoritması kullanılmıştır. Zayıflatılmış karaktere, üç aşamadan oluşan Derya Algoritması uygulanarak karakterin tanınması sağlanır. Bu çalışmada amaç daktilo yada bilgisayar çıktısı karakterlerin, doğru olarak tanınmasıdır. Bu amaçla 0..9 arası rakamlar, a..z ve A..Z arası harfler tanınacaktır.



2. KARAKTER TANIMA ALGORİTMALARINA GENEL BİR BAKIŞ

Karakter tanıma, değişik karakterler üzerinde ve değişik şekillerde yapılabilir. Tanınacak karakterlerin zorluğu, özellikleri, kalitesi, dokümantasyon durumu bu algoritmaların doğruluk oranlarını değiştirmektedir. Tanınacak karakterler çok düzgün bir şekilde gelmiş olsa bile, bazı karakterler arasındaki benzerlikler doğru tanıma oranını etkilemektedir. Kaldı ki karakterler her zaman temiz bir şekilde gelmeyebilir. Bu nedenlerle tanıma algoritmaları çok çeşitli sınıflarda ve tiplerde olabilmekte ve farklı problemlere çözüm üretmeye çalışmaktadır.

İyi şekilli ve iyi şekilde print edilmiş karakterleri tanıyan bir karakter tanıma algoritması geliştirmek zor değilse de, giriş dokümanlarının ve karakterlerin kalitesine bakmayarak, yüksek bir tanıma oranına sahip tanıma algoritması geliştirmek uğraştırıcıdır (Liang et al.,1994). Liang et al.,(1994), birbirine temas eden karakterlere sahip bir dokümanın segmentasyonu üzerinde çalışmışlar ve birbirine temas eden karakterleri parçalamak için, karakter kesim noktalarını dikkate alan dinamik, dönüşümlü bir parçalama algoritması geliştirmişlerdir. Bu çalışmada Liang et al.,(1994), karakter kesim noktaları ayırma fonksiyonları ile hesaplanmış ve sonra, parçalama algoritmaları tarafından üretilen parçalar üzerinde işlem yapan bir karakter sınıflayıcının çıkışlarına dayanan; bir ileri yön parçalama ve geri yön sınır prosedürü tekrarlı olarak işlenmiştir. Anlaşılacağı gibi Liang et al.,(1994) tanıma işleminin daha en başında karşılaşılabilecek bir probleme çözüm aramıştır. Bose and Kuo, (1994); bağlantılı text dokümanlar üzerinde yine Liang et al.'un, (1994) çalışmasına benzer bir çalışma yapmışlar ve bu çalışmada Hidden Markov Modellerini kullanmışlardır (Rabiner,1989). Bose and Kuo, (1994) çalışmalarında; kelime işleme, karakter segmentasyonu ve özellik çıkarma aşamaları üzerinde durmuşlardır. Bu çalışmadan alınan ve çeşitli gürültü

seviyelerindeki tanıma çıktılarını gösteren bir örnek Şekil 2.1’de görülmektedir.

Sunulan Yazı	Tanınan Yazı	Gerçek Yazı
yes	yes	yes
then	then	then
box	box	box
fix	fix	fix
fly	fly	fly
allow	aflow	allow
cod	ood	cod
fix	lix	fix

Şekil 2.1. Çeşitli gürültü seviyelerindeki tanıma sonuçları
(Bose and Kuo’dan, 1994)

Kim and Park, (1995), Hidden Markov Modellerini (HMM) ve fuzzy kurallarını (Zadeh,1965) kullanarak, elyazısı harf ve rakamları tanımaya çalışmışlardır. Rabiner, (1989), HMM’yi otomatik ses tanımda geniş bir şekilde kullanmış ve makalesinde detaylıca açıklamıştır. Kim and Park’ın, (1995) çalışmalarının sonuçlarına bakacak olursak “a”, “u”, “v”, “o” vb... harflerin sırasıyla; “o” veya “d”, “v”, “u”, “a” olarak tanındıkları

görülmektedir. Bu makaleden alınan bir tanıma örneği Şekil 2.2'de görülmektedir.

Hangeul shapes are classified
into 6 types by fuzzy inference
and their recognition based on
quantized features is performed
by optimally arranging features

(a)

Hangeul shapes are classified
into 6 types by fuzzy inference
and their recognition based on
quantized features is performed
by optimally arranging features

(b)

Şekil 2.2. Dört giriş dokümanı ve onların tanıma sonuçları

(a) Giriş dokümanları

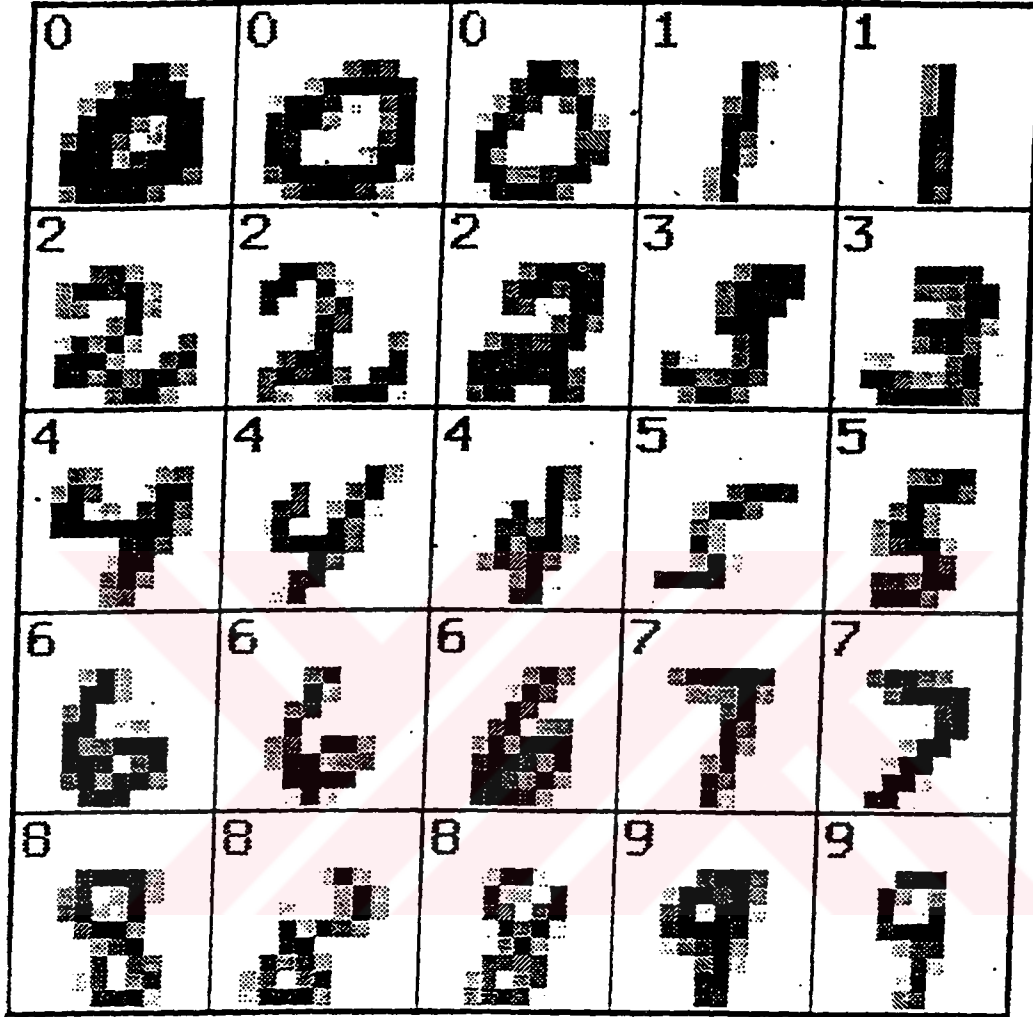
(b) Onların tanıma sonuçları (Kim and Park'dan, 1995)

Chi et al., (1994) el ile yazılmış rakam tanıma üzerinde bir çalışma yapmışlar ve kendi kendine organize olan haritaları (SOM) ve fuzzy kurallarını (Zadeh, 1965) kullanmışlardır. SOM ve fuzzy kurallarının kombinasyonu olan bir mantık geliştirmişler ve buna da FRSOM adını vermişlerdir. Fuzzy mantığı, Zadeh, (1965) tarafından tanıtıldığından bu yana

çeşitli pattern tanıma işlemlerinde kullanılmaktadır. Bu mantık pattern tanımanın ilk aşamasında yani özellik çıkarma ve sınıflandırma aşamasında kullanılmaktadır. Fuzzy kuralları fuzzy sisteminin basit bir versiyonudur ve kontrol içeren, karar veren birçok alanda ve pattern tanıma uygulamaları vardır (Chi et al.,1994). Chi et al.'un, (1994) çalışmasından alınan, FRSOM sınıflayıcı kullanılarak tanınmış el yazısı rakam örnekleri Şekil 2.3'de gösterilmiştir.

Hu and Yan, (1998), çalışmalarına el ile yazılmış rakam tanınması için yapısal temel özellik çıkarma ve kodlama üzerinde bir metod sunmuşlardır. Rakamlar ilk önce düzgünleştirilmiş ve iskeleti çıkarılmıştır. Hu and Yan, (1998), bu çalışmalarının düzgünleştirme aşamasında, yaptığım çalışmanın “Karakterin Yeniden Yapılandırılması” aşamasındaki işlemlere benzer pixel-yerleştirme ve silme yöntemi farklı şekillerde kullanılmıştır. Daha sonra iskelet basit bir şekilde ayrıştırılarak bir özellik noktaları kümesi ortaya çıkarılmıştır. Her iskelete ait bilgiler basit bir kod ile belirlenmiş ve iskeletin topolojik yapısını anlatmak için bu basit kodlardan global bir kod üretilmiştir. Global kodlara göre bütün rakamlar 26 alt sınıfta sınıflandırılmıştır ve basit kodları temel olarak geliştirilmiş iki tanıma algoritması ile tanıma sağlanmıştır. Bu çalışma işleyiş sırası ve işlemlerin temel özellikleri açısından burada yapılan çalışma ile benzerlik göstermektedir. Ayrıca Hu and Yan, (1998), tanıma algoritmalarının performansının, daha çok geliştirilmiş bir zayıflatma algoritmasının kullanılması durumunda geliştirilebileceğinden bahsetmişlerdir.

Yukarda açıklanan bu çalışmalar ve burada bahsedilmeyen diğer OCR uygulamaları sonuçlarına genel olarak bakacak olursak, tanıma oranlarının algoritmalara göre farklılık gösterdiğini; tanıma yapılan alanların, karakterlerin zorluk derecesinin, doküman ve karakter kalitesinin ve kullanılan algoritmaların bu oranı etkilediği görülmektedir.



Şekil 2.3. FRSOM sınıflayıcı tarafından doğru olarak sınıflandırılmış el yazısı rakamların örnekleri (Chi et al.'dan, 1994)

3. KARAKTER ZAYIFLATMA VE KULLANILAN KARAKTER ZAYIFLATMA ALGORİTMASI

Dijitalize edilmiş iki boyutlu bir karakterin iskeletini çıkarma, karakter tanımanın temel bir adımıdır. Karakter tanıma, karakterin belirli özelliklerinin ve temel yapısının, ana karaktere ait özellik bildiren referans bilgileri ile karşılaştırılması esasına dayandığından, karakterin özelliklerinin korunduğu, temel yapısının elde edilmesi gereklidir. Bu da iyi bir zayıflatma algoritması ile mümkündür.

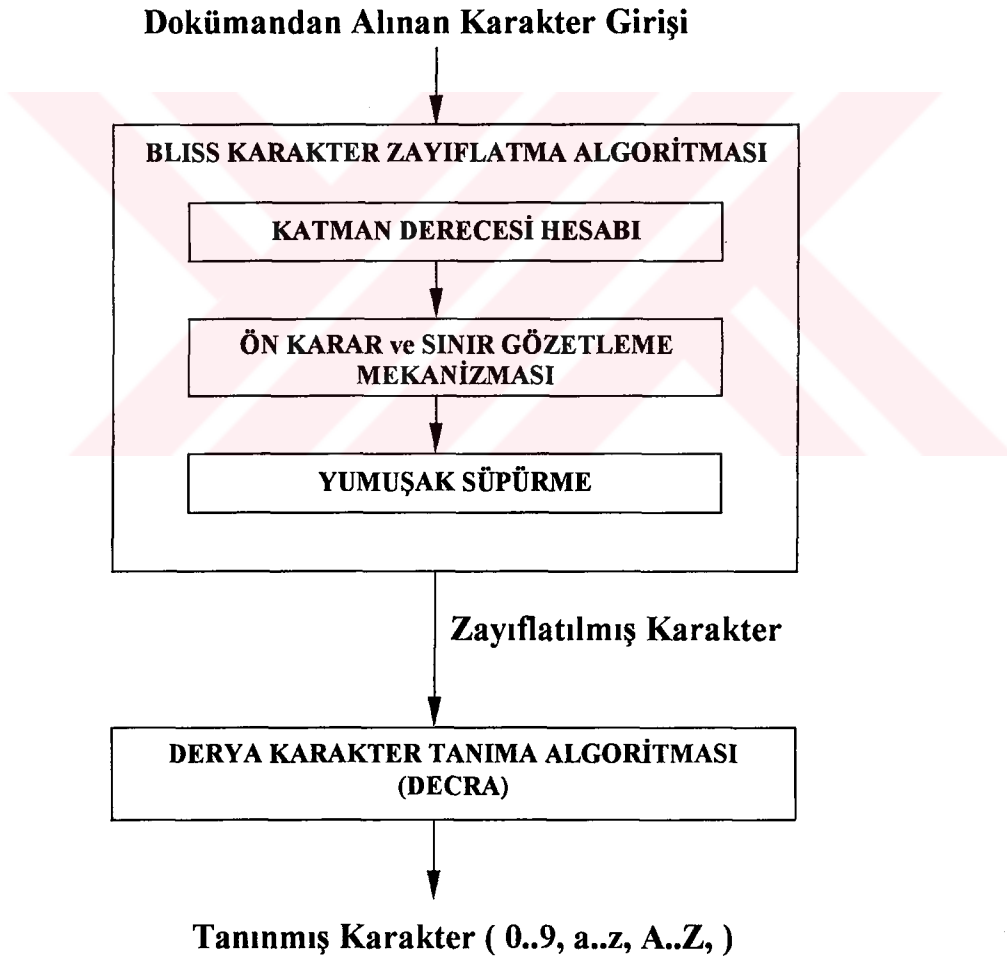
Karakterin bir iskelet olarak temsil edilmesi, şeklin ana parçalarının yapısal özelliklerini ifade etmek için iyi bir yoldur. İskeletin avantajları; karakterin temel yapısal bilgilerini depolamak için gerekli hafıza alanındaki azalma, karakter işlemlerinde gerekli bilgi yapılarının sadeleşmesi ve gerekli işlem zamanındaki azalmadır. İskeleti çıkarma işlemini etkileyen en önemli faktör ise karakter üzerindeki parazitlerdir. Zayıflatma işleminden önce bu tür gürültülerin giderilmesine çalışılır. Gürültüler giderilirken, şeklin özellik bildiren parçalarının korunmasına dikkat edilir. Bununla birlikte kullanılan zayıflatma algoritmasının da gürültü oluşturmaması gerekir.

Karakter tanıma karşılaşılan problemleri en aza indirgeyen yeni bir zayıflatma algoritması kullanılmıştır. BLISS (Boundary Lookout Information and Soft Sweep) adı verilen bu algoritma zayıflatma işlemini üç ana aşamada gerçekleştirmektedir:

- 1-Katman Derecesi Hesabı,
- 2-Ön Karar ve Sınır Gözetleme Mekanizması,
- 3-Yumuşak Süpürme.

Katman derecesi hesabında, ana karakter üzerinde yer alan her bir pixeli dıştan-içe doğru numaralandırır. Böylece her bir numara, ait olduğu pixelin karakter üzerindeki katman derecesini gösterir.

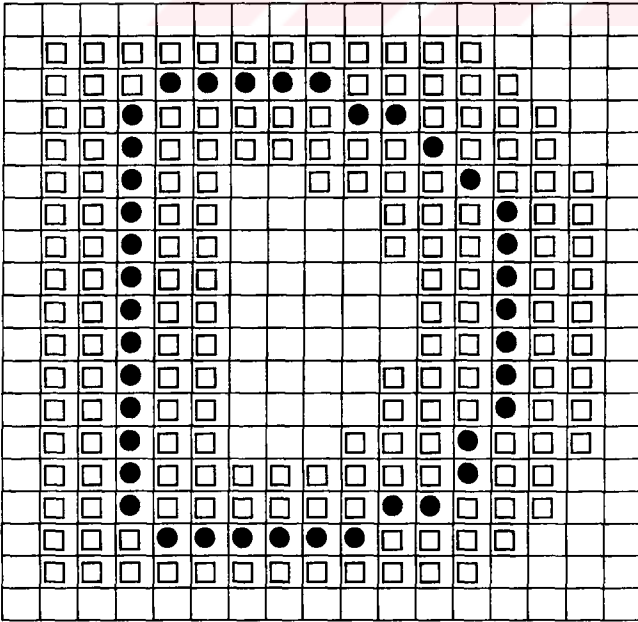
Ön karar mekanizması modüllerden oluşmaktadır. Burada her bir modül katman derecesi hesabına ve kendi içinde o ana kadar işlenmiş olan modülün/modüllerin sonuçlarına bakarak, iskeletin temel bağlantılarını ortaya çıkarır. Temel bağlantılarda ortaya çıkacak bazı pürüzleri gidermek için de sınır gözetleme mekanizması geliştirilmiştir.



Şekil 3.1.Tanım sistemini genel yapısı

BLISS algoritmasının temeli bir takım seri işlemlere dayanır. Seri işlemler yapan algoritmalar daha önceki işlem sonuçlarını kullanırlar. Ayrıca bu işlemler, her işlem basamağının gerektirdiği kadar, yani iskelet verilerinde değişim olduğu sürece tekrarlanır.

Zayıflatma algoritması olarak başka zayıflatma algoritmaları da kullanılabilir. Zayıflatma algoritmasının çıktıları ne kadar düzgün olursa, uygulama o kadar başarılı olur. Şekil 3.1’de ifade edildiği gibi Derya Algoritması (DECRA), BLISS zayıflatma algoritmasının çıktılarını, yani iskeleti çıkarılmış karakterleri kullanmaktadır. Bu uygulamada dokümantasyon tanıma uygulaması kullanılmadığından, direkt doküman üzerinden alınan veriler değil düzgün şekilli “Arial Black” fontundan yararlanılmıştır. Dolayısıyla BLISS’de bu karakterler üzerinde zayıflatma yapmaktadır. Şekil 3.2’de bir “D” karakterinin BLISS algoritması ile zayıflatıldıktan sonraki şekli görülmektedir. Şekilde karakterin zayıflatılmış halini, yani iskeletini “●” sembolü yerleştirilen hat ifade etmektedir.

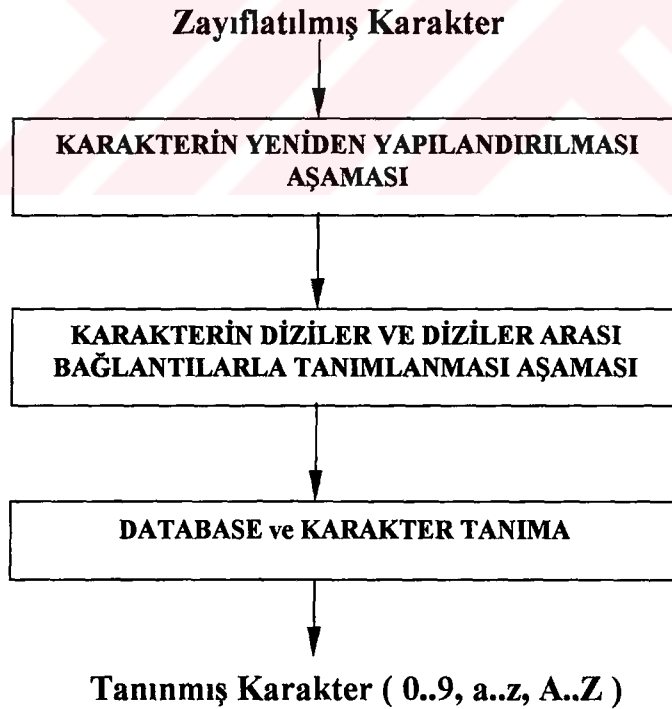


Şekil 3.2. Bir “D” karakterinin BLISS ile zayıflatılmış şekli

4. DERYA KARAKTER TANIMA ALGORİTMASI (DECRA)

Derya Algoritması, zayıflatılmış karakteri mevcut pixel yerleşimine göre, aynı doğrultudaki pixellerin birarada tutulduğu dizilerle ve diziler arası bağlantılarla ifade eder. Bu bilgileri database’de mevcut olan karakter bilgileriyle karşılaştırır ve karakteri tanır.

DECRA (Derya Character Recognition Algorithm) algoritması, Şekil 4.1’de görüldüğü gibi, temel olarak üç bölümden oluşur. Bunlardan birincisi, zayıflatılmış olarak gelen karakter üzerindeki problemleri giderildiği, Karakterin Yeniden Yapılandırılması aşamasıdır. Bu aşamada gereksiz pixel çıkıntıları giderilerek, kesişim bölgelerindeki pixel yerleşimleri düzeltilerek; algoritmanın ileriki aşamalarında karşılaşılabilecek zorlukların önlenmesine çalışılmaktadır.



Şekil 4.1. Derya karakter tanıma algoritmasının genel blok diyagramı (DECRA)

İkinci aşama, karaktere ait dizilerin ve bu dizilerle diğer diziler arasındaki bağlantıların çıkarıldığı, Karakterin Diziler ve Diziler Arası Bağlantılarla Tanımlanması aşamasıdır. Dizi, karakter üzerinde aynı doğrultuda yer alan pixellerin bir araya toplandığı yerdir. Burada karakterin sahip olduğu pixeller numaralandırılır ve diziler oluşturulur. Bu aşamanın ilerleyen evrelerinde ise, karakterden ilk olarak çıkarılan diziler üzerinde yapılan işlemlerle, gereksiz diziler silinir. Bu aşama sonunda karakter; yalnızca gerekli dizilerle ve diziler arası bağlantılarla temsil edilir. Buradan elde edilen bilgiler bir sonraki aşama için gerekli olan verilerdir.

Son aşama, karakterin tanındığı Database ve Karakter Tanıma aşamasıdır. Database, her bir karaktere ait özellik bilgilerinin saklandığı yerdir. Karakterin Diziler ve Diziler Arası Bağlantılarla Tanımlanması aşamasından gelen bilgiler, Database'deki bilgilerle karşılaştırılır ve bu bilgilerin hangi karaktere ait olduğu tespit edilerek, karakter tanınmış olur.

4.1. Karakterin Yeniden Yapılandırılması

Karakter tanıma uygulamalarında, dokümandan alınan ve zayıflatılmış karakterin yani iskeletin; karakterin temel ve gerekli özelliklerini taşıması, gereksiz parça ve çıkıntıları da barındırmaması istenir. Ancak tanıma algoritmalarının da bu özelliklere sahip, kusursuz iskeletler üretmeleri çok zordur. Bu nedenle bu çalışmada iskelet olarak gelen karakter üzerinde, zayıflatma algoritmalarının yok edemediği ya da düzgünleştiremediği kısımlar düzeltilmeye çalışılmıştır. Karakterin düzgün bir şekilde tanımlanabilmesi için, DECRA algoritmasının yapısına ve işlevine uygun olarak, zayıflatılmış karakter üzerinde düzeltilmesi gereken kısımlar ele alınır ve aşağıda ifade edildiği şekilde düzgünleştirilir. Problemlerli durumlar üç kısım dahilinde giderilmeye çalışılmıştır. Bu aşamada yer alan Şekil 4.2, Şekil 4.3 ve Şekil

4.4’de (a) şekillerinde problemlı pixel yerleşimleri, (b) şekillerinde bu problemlerin düzeltilmiş durumları gösterilmiştir.

I. Şekil 4.2’de gösterilen durum dışında aşağıda ifade edilen durumlarda, koyu renkle gösterilen pixel silinerek, □ sembolü ile gösterilen boşluğa pixel yerleştirilir. Burada karakter üzerinde aynı doğrultuda yer alan tek pixel yerleşimi çıkıntılar giderilmeye çalışılır.

	X	
X	□	X
	X	

Şekil 4.2. “□” sembolü ile gösterilen boşluğun doldurulmadığı pixel yerleşimi

	X	
X	□	X

X	X	X

	X	
X	□	
	X	

	X	
	X	
	X	

	X	
	□	X
	X	

	X	
	X	
	X	

X	□	X
	X	

X	X	X

(a)

(b)

Şekil 4.3. Pixel yerleşim durumları-1 (a) “□” sembolü ile gösterilen boşluğun doldurulacağı pixel yerleşimleri (b) “□” sembolü ile gösterilen boşluğun doldurulma durumu

II. Şekil 4.4'deki şekillerle ifade edilen durumlarda da koyu renkle gösterilen pixel silinerek, □ sembolü ile gösterilen boşluklara pixel yerleştirilir. Bunun için burada küçük "x" harfi ile gösterilen pixelin, şekilde görülen tek pixel komşuluğu dışında; başka dolu pixel komşuluklarının da olması gerekir. Burada gösterilen pixel durumlarının iyileştirilmesi ile çapraz hatlar üzerindeki problemleri durumlarda ve bu hatlar üzerindeki kesişim noktalarında, algoritmanın ileriki aşamalarında, diziler oluşturulurken, çıkabilecek problemler önlenmektedir.

X	□	x
	X	□
		X

X	X	X
		X
		X

X		
□	X	
x	□	X

X		
X		
X	X	X

x	□	X
□	X	
X		

X	X	X
X		
X		

		X
	X	□
X	□	x

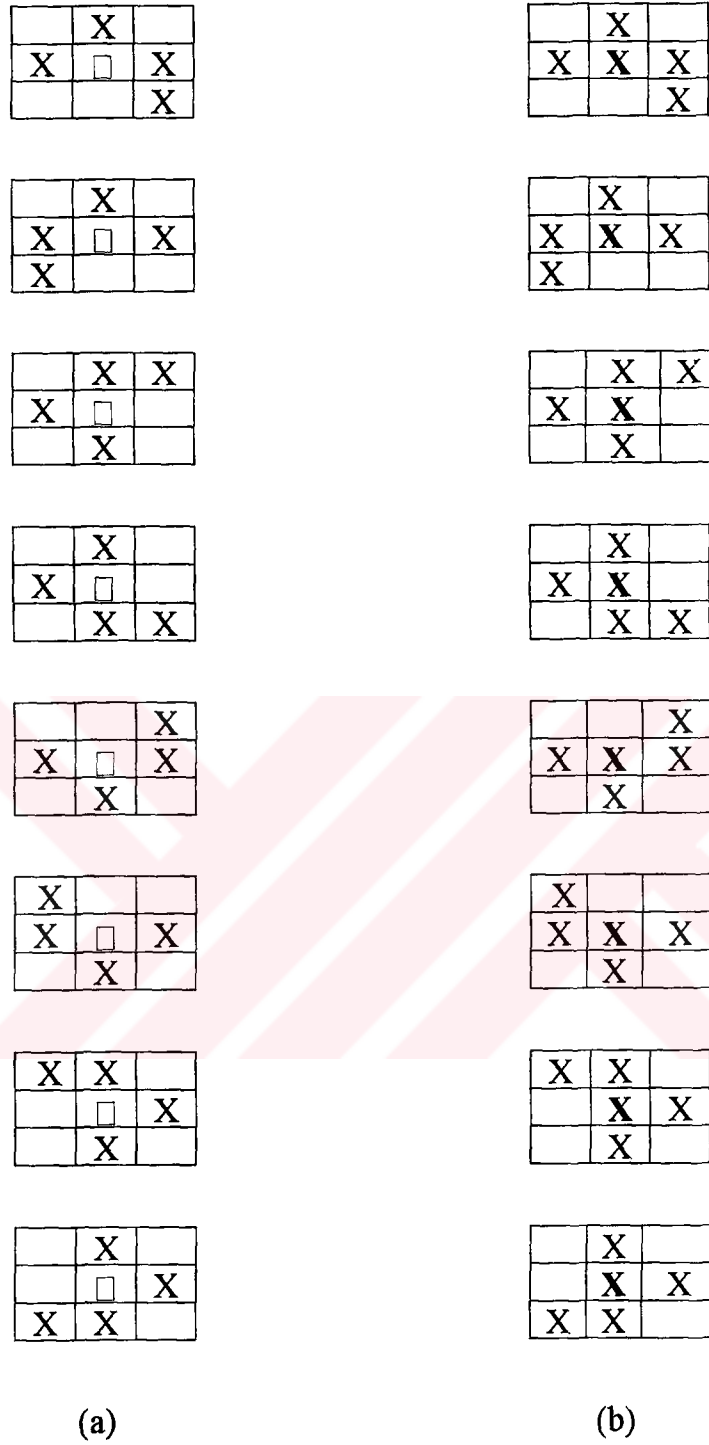
		X
		X
X	X	X

(a)

(b)

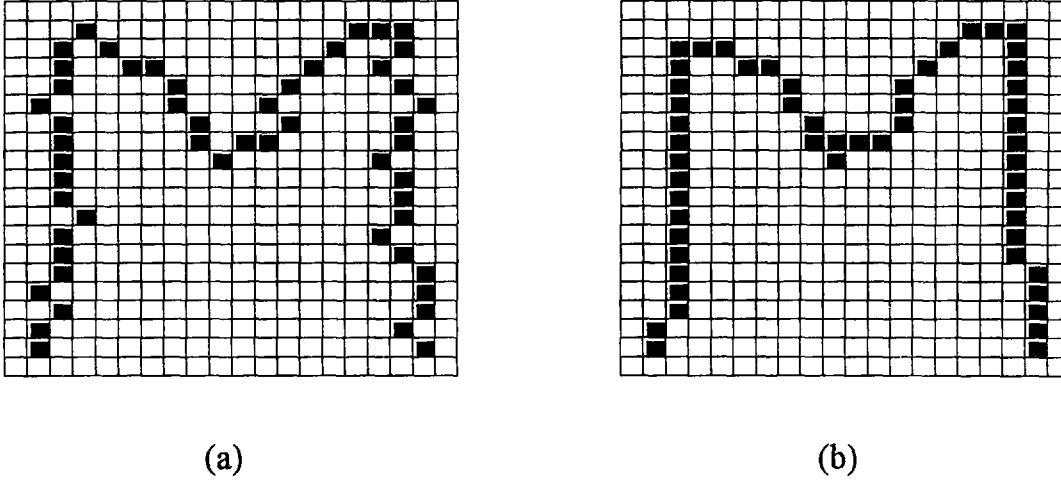
Şekil 4.4. Pixel yerleşim durumları-2 (a) "□" sembolü ile gösterilen boşluğun doldurulacağı ve gereksiz pixelin silineceği pixel yerleşimleri (b) "□" sembolü ile gösterilen boşluğun doldurulma ve gereksiz pixelin silinme durumu

III. Şekil 4.5'de gösterilen durumlarda da "□" sembolü ile gösterilen boşluklara pixel yerleştirilir. Böylece özellikle kesişim noktalarındaki gereksiz pixellerin dizilere alınması ve gereksiz dizi oluşma problemi giderilir.



Şekil 4.5. Pixel yerleşim durumları-3 (a) “□” sembolü ile gösterilen boşluğun doldurulacağı pixel yerleşimleri (b) “□” sembolü ile gösterilen boşluğun doldurulma durumu

Bu aşamada yer alan üç maddenin nasıl işlendiğine ilişkin bir örnek aşağıda verilmiştir. Şekil 4.6’da bir “M” karakteri örneği görülmektedir.



Şekil 4.6. Karakterin yeniden yapılandırılması aşamasına ait bir “M” karakteri örneği (a) Zayıflatılmış ve problemleri olan bir “M” karakteri (b) “M” karakterinin, karakterin yeniden yapılandırılması aşamasından sonraki durumu

4.2. Karakterin Diziler ve Diziler Arası Bağlantılarla Tanımlanması

DECRA algoritmasının en önemli aşaması bu aşamadır. Çünkü, karaktere ait bilgiler burada yer alan bir dizi işlemlerle çıkarılmakta ve tanıma aşamasına bu bilgiler gönderilmektedir. Burada karakter, aynı yöndeki pixellerin tarama yönüne uygun olarak bir araya getirilmesiyle oluşturulan, gösterilen dört yönden (Bkz. Şekil 4.8, Çizelge 4.1) birine sahip diziler ve bu diziler arası bağlantılarla tanımlanır.

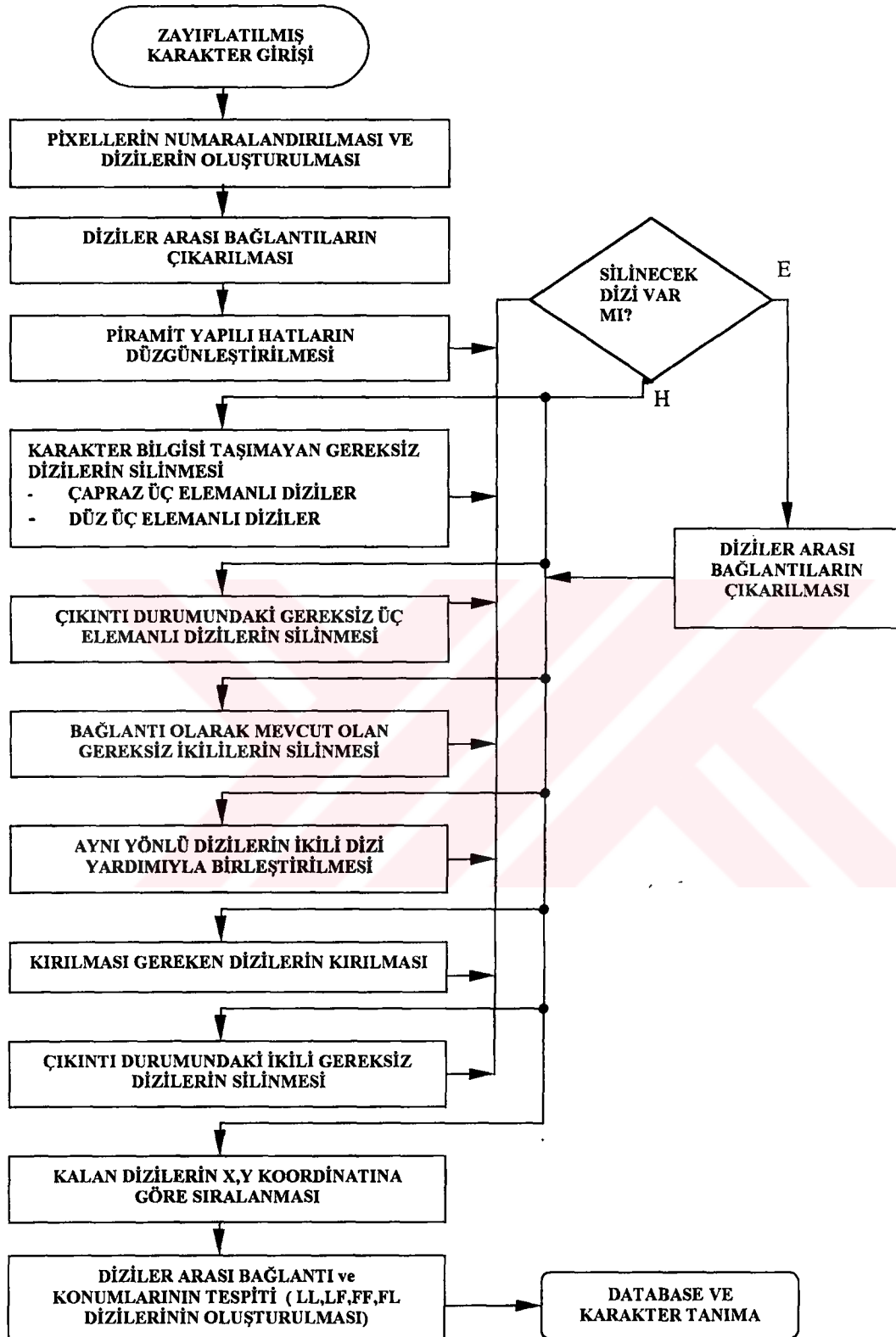
Şekil 4.8’de gösterilen sekiz yön Freeman Chain (zincir) kodu (Fu, K.S., 1982) olarak gösterilen yönlerden esinlenerek çıkarılmıştır. Freeman’ın kodunu Mahmoud, S.A., (1994) çalışmasında kullanmış, Kim, H.J. and Kim, P.K., (1996) ise bu koddan esinlenerek çalışmaları için kendilerine sekiz yön tespit etmişlerdir. DECRA’da kullanılan yön numaraları ve bu yönlerin kullanım şekli de farklıdır (Bkz. Şekil 4.8, Çizelge 4.1). Çünkü bu kodun kullanıldığı algoritmalarda bu

yönlere ait basit şekiller geliştirilmiş ve her basit şekil bir harf ile isimlendirilmiştir. Zayıflatılmış karakter bu basit şekilleri simgeleyen harflerin ard arda yazılması ile (zincir gibi) ifade edilmiştir (Fu, K.S., 1982). Fan, K.-C., et al., (1995) çalışmalarında kullandıkları yön sayısını onaltıya çıkarmışlar ve her yönü de sayısal olarak isimlendirerek, zayıflatılmış karakteri bu yön isimlerini ard arda yazarak ifade etmişler ve sonra “And-Or Araştırma Ağacı” dedikleri mantığı kullanmışlardır.

Karakter tanıma uygulamalarında kullanılmış olan birçok metod Fu, K.S., (1982) tarafından; kullanılan çalışmalar referans verilerek açıklanmıştır. Bunlar arasında, demetleme analizi (Cluster Analysis), ki bu analizde tanıma ağacı, ağaçlar arasındaki farklar, gramer özellikleri kullanılır, yakın komşu tanıma kuralı, ikili (binary) ağaç formu, syntax analizi vb. gibi, kullanılmış diğer yöntemler bulunmaktadır.

DECRA algoritmasında kullanılan sekiz yön, dört ana yön adı altında, ikişer ikişer gruplanırlar. Yani birbirine çapraz durumda bulunan iki yön aslında tek bir yönü ifade eder. Ancak pixeller dizilere atılmak üzere işlenirken, bu pixelin etrafında yer alan pixeller, en küçük yön numarasına sahip pixelden başlanarak ele alınır.

Burada yer alan işlemler oldukça dikkatle hesaplanmıştır. Çünkü yetersiz ya da eksik bilgi tanımlaması karakterin tanınmasını engeller. İşlemler bir sıra dahilinde birbirini izler ve bazıları da her işlem sonrası tekrar edilir. İstendik ve doğru sonuçlar için buradaki işlem sırasına uyulması gerekir. Bu aşamaya ait, Şekil 4.7’de verilen blok diyagram, işlem sırasını ve programlama mantığını göstermektedir.



Şekil 4.7. Karakterin diziler ve diziler arası bağlantılarla tanımlanması aşamasının işlem sırasını gösterir blok diyagram

4.2.1. Pixellerin numaralandırılması ve dizilerin oluşturulması mantığı

Zayıflatılmış karakter üzerindeki dolu pixellerin, numaralandırılması ve aynı doğrultudaki pixellerin dizilere atılması ile diziler oluşturulur. Bu işlem yapılırken, oluşan dizinin yönü de belirlenir. Buna göre;

1. Soldan-sağa, yukarıdan-aşağıya taramada karşılaşılan ilk dolu pixel, ilk olarak numaralandırılır.
2. Numaralandırılan pixelin sekiz yöndeki komşuluklarına bakılır. Şekil 4.8’de gösterilen yönler göre ilk olarak “1” numaralı yönden başlanır ve “8” nolu yöne doğru sırayla gidilir.

7	6	5
8	X	4
1	2	3

Şekil 4.8. Dolu bir “X” pixeli ve sekiz yöndeki komşulukları

3. Bu ilgili yöndeki tüm dolu pixeller numaralandırılır ve bu yönün yukarı çaprazında yer alan diğer yöndeki pixellere bakılır. Bu çapraz yönde de pixel varsa yön doğrultusuna dolu olan tüm pixeller numaralandırılır (numaralandırılmış pixel varsa tekrar numara verilmez).
4. İlgili ve çapraz yönlerin her ikisinde de pixel var ve numaralandırılmışsa, artık bu yöndeki tüm pixeller en yukarıdaki pixelden başlanarak bir diziye atılır.
5. İlgili yönün çapraz yönde pixeli yoksa, diziye atma işlemi komşuluklarına bakılan ana (merkez) pixelden başlar ve ilgili yöndeki pixellerin tamamı alınır.

Çizelge 4.1. Şekil 4.8’de gösterilen yön numaralarına karşılık gelen yönlerin isimleri

Yön Numarası	Yönün İsmi
1 veya 5	SLASH (/)
2 veya 6	VERTICAL ()
3 veya 7	BACKSLASH (\)
4 veya 8	HORIZONTAL (-)

6. Oluşturulan bu dizinin yönü tespit edilir. Dizinin yönü, ilgili yöndür. Çizelge 4.1’de, Şekil 4.8’de numaralanmış olan yönlerin karşılıkları gösterilmiştir. Tüm dizilerde elemanların(pixellerin) dizilere alınma yönü, yukarıdan-aşağıya ve soldan-sağadır.

7. Oluşturulan bu dizinin büyüklüğü tespit edilir. Dizinin büyüklüğü eleman sayısına eşittir.

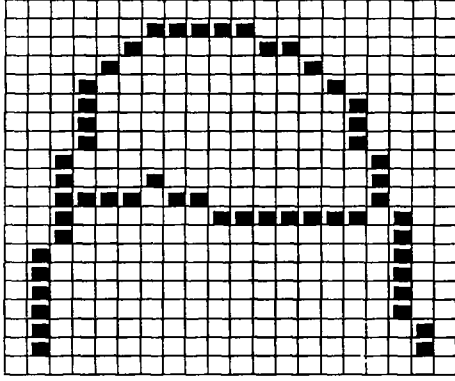
8. İlgili yönde pixel yoksa, çapraz yöndeki pixele bakılmaksızın, sıradaki ilgili yöne geçilir.

9. Madde 3...8 arası tüm maddeler, bu yön içinde işlenir.

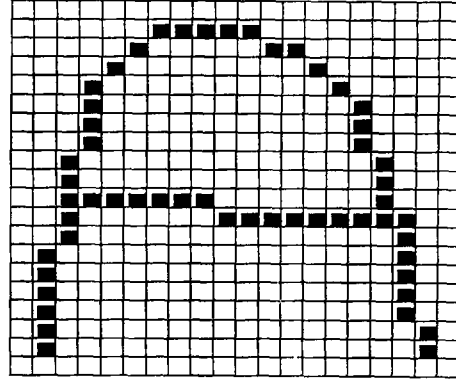
10. Pixel sekiz yönde incelendikten sonra, sıradaki numaralı pixele geçilir. Yukarıdaki 2...9 maddeleri, sıradaki pixel için de tekrar edilir.

11. Numaralandırılmış pixeller bitti ise karakter kaldığı yerden tekrar taranarak başka numaralandırılmamış dolu pixel var mı? Bakılır. Varsa en son kalan numaradan itibaren numaralandırılır ve tüm işlemler tekrar edilir.

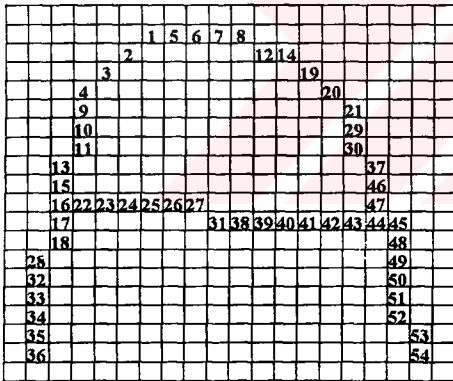
Aşağıda zayıflatılmış bir karaktere ait pixellerin, numaralandırılarak dizilere atılmasına ilişkin bir örnek verilmiştir.



(a)



(b)



(c)

Şekil 4.9. Zayıflatılmış bir “A” karakterinin numaralandırılarak dizilere

atılması (a) Zayıflatılmış bir “A” karakteri

(b) Karakterin yeniden yapılandırılması aşamasından sonraki durum, burada karakter algoritmaya uygun olarak düzeltilmiştir

(c) Karakterin numaralandırılmış hali

Çizelge 4.2. Şekil 4.9'daki "A" karakterinden çıkarılan ilk diziler

Dizi		
NUMARASI	DİZİNİN ELEMANLARI	DİZİNİN YÖNÜ
1	1,2,3,4	SLASH
2	1,5,6,7,8	HORIZONTAL
3	4,9,10,11	VERTICAL
4	8,12	BACKSLASH
5	11,13	SLASH
6	12,14	HORIZONTAL
7	13,15,16,17,18	VERTICAL
8	14,19,20,21	BACKSLASH
9	15,22	BACKSLASH
10	16,22,23,24,25,26,27	HORIZONTAL
11	22,17	SLASH
12	18,28	SLASH
13	21,29,30	VERTICAL
14	27,31	BACKSLASH
15	28,32,33,34,35,36	VERTICAL
16	30,37	BACKSLASH
17	31,38,39,40,41,42,43,44,45	HORIZONTAL
18	37,46,47,44	VERTICAL
19	47,43	SLASH
20	44,48	BACKSLASH
21	45,48,49,50,51,52	VERTICAL
22	47,45	BACKSLASH
23	52,53	BACKSLASH
24	53,54	VERTICAL

4.2.2. Dizilerin diğer dizilerle olan ikili bağlantılarının çıkarılması

İncelenen karaktere ait diziler incelenerek, ortak elemana sahip diziler arasındaki ikili bağlantılar tespit edilir. Buna göre; aynı numaralı bir pixel birden çok dizide yer alıyorsa, bu diziler kendi aralarında ikili bağlantıya sahiptirler. Bir dizinin bağlantılı olduğu dizilere; dizinin Co_Scon'ları¹, bir pixelin yer aldığı dizilere; o pixelin Pre_Scon'ları² adı verilir. Dizilere ait ikili bağlantılar çıkarılırken (Co_Scon), ilgili dizi elemanlarının (pixellerin) Pre_Scon'larından faydalanılır. İkili bağlantılar çıkarılırken aşağıdaki kurallar uygulanır.

¹ Co_Scon, yardımcı ikili bağlantı anlamındadır. Co, Secondary, Connection kelimelerinin ön harflerinin bir araya getirilmesi ile oluşturulmuş bir kısaltmadır.

² Pre_Scon, ön ikili bağlantı anlamındadır. Pre, Secondary, Connection kelimelerinin ön harflerinin bir araya getirilmesi ile oluşturulmuş bir kısaltmadır.

1. Her bir dizi için ikili bağlantılar çıkarılırken, ilk bulunan diziden başlanır ve incelenen diziden sonraki diziler göz önüne alınır. Yani incelenen dizi numarasından daha büyük numaralı dizilerde ikili bağlantı aranır.
2. Bunun için incelenen dizinin elemanları diğer dizilerde aranır. Yani elemanların Pre_Scon'larına bakılır. Elemanın bulunduğu dizi(ler) incelenen dizinin ikili bağlantılı olduğu dizilerdir ve Co_Scon'larına işlenir.
3. Daha önce yazılmış olan bir ikili bağlantı tekrar yazılmaz.

Çizelge 4.3. Çizelge 4.2’de gösterilen “A” karakterinden çıkarılan ilk dizilerin bağlantılı oldukları diziler (Co_Scons)

DİZİ NUMARASI	DİZİNİN BAĞLANTILI OLDUĞU DİZİ(LER) (CO SCONS)
1	2,3
2	1,4
3	1,5
4	2,6
5	3,7
6	4,8
7	5,9,10,11,12
8	6,13
9	7,10,11
10	7,9,11,14
11	9,10,7
12	7,15
13	8,16
14	10,17
15	12
16	13,18
17	14,19,18,20,21,22
18	16,19,22,17,20
19	18,22,17
20	17,18,21
21	17,22,20,23
22	18,19,17,21
23	21,24
24	23

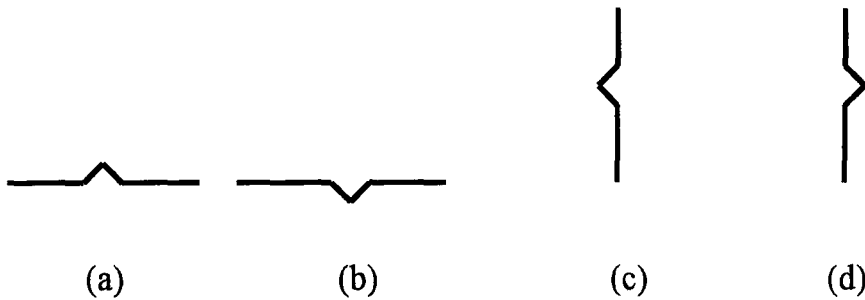
Yukarıda bir “A” karakteri için verilen örnekte (Bkz. Şekil 4.9, Çizelge 4.2) ilk olarak çıkarılan yirmidört dizinin yine ilk olarak çıkarılan bağlantıları (Co_Scons) Çizelge 4.3’de görülmektedir.

Bu aşama, daha sonra yer alan aşamaların birçoğunun sonunda; tekrar işlenmektedir. Böylelikle karakterden ilk olarak çıkarılan diziler üzerindeki her işlem sonucunda, diziler arası bağlantılar yenilenmiş olacaktır. Çünkü herhangi bir dizinin silinmesi, kırılması (bölünmesi), diziye yeni bir elemanın eklenmesi ya da diziden eleman silinmesi bu dizinin diğer dizi ya da dizilerle olan bağlantılarını değiştirmektedir.

4.2.3. Piramit yapılı hatların düzgünleştirilmesi

Karakterin düzgün hatları üzerinde yer alan ve hattı bozan, piramitsel pixel yerleşimi giderilerek hat düzgünleştirilir. Bunun için diziler en küçük numaralı diziden başlamak üzere incelenir ve piramitsel pixel yapısını ifade eden diziler bulunur. Piramidin düz hattının, piramitsel yerleşimin olduğu hattın yönünü gösterdiği düşünülürse, piramidin tepe noktasında birleşen hatlar; birbirlerine göre ters çapraz yöne sahip, iki tane, iki elemanlı dizi olmalıdır. Bu iki dizi, üzerinde yer aldıkları düzgün hat veya hatlar ile birleştirilerek, bu düzgün hattı ifade eden tek bir dizi haline getirilecektir. Oluşabilecek piramit yapılı hatlar Şekil 4.10'da gösterilmiştir. Buna göre iki elemanlı diziler incelenerek piramitsel yapı tespit edilir. Bulunması gereken dizi;

1. İki elemanlı olmalıdır,
2. Çapraz bir hat ifade etmelidir,



Şekil 4.10. Piramitsel pixel yapıları

3. Kendisine göre ters çapraz yönlü ve iki elemanlı en az bir dizi ile ikili bağlantısı olmalıdır.

Bu özelliklere sahip dizi ve bu dizinin istenen özellikte ikili bağlantısı olan diğer ikili dizi bulunduktan sonra ki bunlar piramidin tepe noktasında birleşen hatlarıdır, bunların üzerinde bulunduğu düz hattın, piramidin düz hattı (tabanı), yönü tespit edilir. Tespit edilen bu yön yeni oluşturulacak dizinin yönü olacaktır. Bu işlem için oluşabilecek durumlar aşağıda ifade edilmiştir.

SLASH ve BACKSLASH olarak bulunan iki hattın birleştiği noktadaki pixel;

1. Birinin ilk ve diğerinin son pixeli ise hat VERTICAL'dır,
2. Her ikisinin de ilk veya son pixeli ise hat HORIZONTAL'dır.

Yönün tespiti ile birlikte bu iki çapraz dizinin, soldan-sağa ve yukarıdan-aşağıya yazılış sırası göz önüne alınarak, yeni oluşturulacak diziye hangisinin birinci, hangisinin ikinci sırada yazılacağı da tespit edilmelidir. Bu tespit için aşağıdaki yol izlenir:

Tespit edilen yön VERTICAL iken;

1. Bulunan ilk çapraz dizinin yönü SLASH ve bu dizinin ilk pixeli, BACKSLASH yönlü ikinci çapraz dizi ile ortak olan pixelse sıra BACKSLASH-SLASH, ikinci pixeli ortak pixel ise sıra SLASH-BACKSLASH olacaktır.

2. Bulunan ilk çapraz dizinin yönü BACKSLASH ve bu dizinin ilk pixeli, SLASH yönlü ikinci çapraz dizi ile ortak olan pixelse sıra SLASH-BACKSLASH, ikinci pixeli ortak pixel ise sıra BACKSLASH-SLASH olacaktır.

Tespit edilen yön HORIZONTAL iken;

1. Bulunan ilk çapraz dizinin yönü SLASH ve bu dizinin ilk pixeli, BACKSLASH yönlü ikinci çapraz dizi ile ortak olan pixelse sıra SLASH-BACKSLASH, ikinci pixeli ortak pixel ise sıra BACKSLASH-SLASH olacaktır.

2. Bulunan ilk çapraz dizinin yönü BACKSLASH ve bu dizinin ilk pixeli, SLASH yönlü ikinci çapraz dizi ile ortak olan pixelse sıra SLASH-BACKSLASH, ikinci pixeli ortak pixel ise sıra BACKSLASH-SLASH olacaktır.

Oluşacak olan yeni dizinin yönü bu şekilde tespit edildikten sonra, bu iki çapraz hattın tespit edilen yönde bağlantılı oldukları diziler araştırılır. Oluşturulacak yeni diziye önce yazılacak çapraz dizinin, varsa tespit edilen yönde bağlantılı olduğu diziye “İlk Dizi”, oluşturulacak yeni diziye sonra yazılacak çapraz dizinin, varsa tespit edilen yönde bağlantılı olduğu diziye “Son Dizi” adı verilir. Bulunan diziler de yeni oluşacak diziye alınacaklardır. Dizi oluşturulurken soldan-sağa ve yukarıdan-aşağıya yazılış sırası dikkate alınarak yeni bir dizi numarası altında birleştirilirler. Birleşen dizilerin “Birleşim-Dizisi” ne yeni oluşan dizinin numarası kaydedilir ve karakterin tamamı işlendikten sonra, birleşen tüm diziler silinir. Bu birleştirmede aşağıdaki hususlara dikkat edilir:

1. “İlk Dizi” ve “Son Dizi” varsa, yeni diziye sırasıyla “İlk Dizi”, önce yazılacak çapraz dizi, sonra yazılacak çapraz dizi, “Son Dizi” elemanları alınır. Ortak olan elemanlar ikinci kez alınmaz. Birleşen tüm diziler silinmek üzere işaretlenir.

2. “İlk Dizi” ve “Son Dizi” yoksa ve iki çapraz dizinin ortak olan elemanının bu iki çapraz dizinin dışında herhangi bir dizi ile bağlantısı varsa, yeni dizi oluşturulmaz ve bu iki çapraz dizi silinmek üzere işaretlenir. İki çapraz dizinin ortak olan elemanının bu iki çapraz dizinin dışında herhangi bir dizi ile bağlantısı yoksa, sadece bu iki çapraz diziyi içeren yeni bir dizi oluşturulur ve bu iki çapraz dizi silinmek üzere işaretlenir.

3. “İlk Dizi” veya “Son Dizi” aynı numaralı dizilerse veya farklı numaralı diziler iken daha önce yeni bir dizi numarası altında birleştirilmişlerse; yeni dizi oluşturulmaz, iki çapraz dizi silinmek üzere işaretlenir. Bu noktada bu dizinin bu iki çapraz dizi dışında başka bir dizi ile de bağlantısı varsa dizi bu noktadan kırılır ve iki yeni dizi oluşturulur. Yine iki çapraz dizi ve kırılan dizi silinmek üzere işaretlenir.

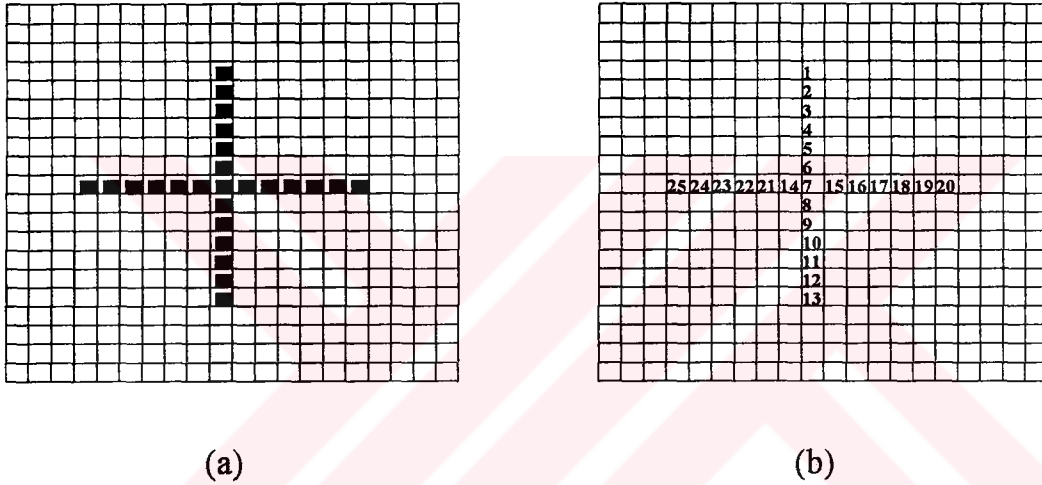
4. “İlk Dizi” veya “Son Dizi”den herhangi biri varsa; olan çapraz dizi ve bu çapraz ile komşu olan ilk veya son dizi birleştirilir ve sadece birleşen diziler silinmek üzere işaretlenir. Buna göre “İlk Dizi” ve önce yazılacak çapraz dizi veya sonra yazılacak çapraz dizi ve “Son Dizi” yeni bir dizide birleşebilir.

5. Oluşan yeni dizi numarası en son oluşturulan dizi numarasını takip eder ve yeni dizilerde birleşim işlemine katılırlar. Ancak birleşecek diziler daha önce farklı yeni dizilerde, farklı dizilerle birleşmişlerse tekrar birleştirilebilirler.

6. Karakter üzerindeki tüm dizi işlemleri bittikten sonra silinmek üzere işaretlenen diziler silinir ve diziler arası ikili bağlantılar (PreScon ve Co_Scon) tekrar çıkarılır.

Bu aşama için aşağıda basit bir örnek sunulmuştur. Şekil 4.11’de gösterilen “+”, karakteri için ilk olarak altı tane dizi çıkarılmıştır. Buna göre karakter iki

ana dizi ve bu dizilerin orta noktadan bağlantısını gösteren çapraz yönlü, dört tane iki elemanlı dizi ile ifade edilmiştir (Bkz.Çizelge 4.4). Piramit Yapılı Hatların Düzgünleştirilmesi aşamasında, orta noktada karşılaşılan piramitsel durumlar üzerinde yukarıda açıklanan işlemler gerçekleştirilmiş ve bu dört çapraz yönlü iki elemanlı dizi silinmiş ve iki ana dizi de orta noktalarından kırılmıştır (Bkz. Çizelge 4.5). Çizelge 4.4’ de “+” karakterinden çıkarılan ilk diziler, Çizelge 4.5’de ise piramitsel yapılı hatların düzgünleştirilmesi aşaması sonrasında oluşan diziler görülmektedir.



Şekil 4.11. Piramitsel yapılı hatların düzgünleştirilmesi aşamasının “+” karakterine uygulanması
 (a) Zayıflatılmış düzgün bir “+” karakteri
 (b) “+” karakterinin numaralandırılmış hali

Çizelge 4.4. “+” karakterinden çıkarılan ilk diziler

DİZİ NUMARASI	DİZİNİN ELEMANLARI	DİZİNİN YÖNÜ
1	1,2,3,4,5,6,7,8,9,10,11,12,13	VERTICAL
2	6,14	SLASH
3	6,15	BACKSLASH
4	25,24,23,22,21,14,7,15,16,17,18,19,20	VERTICAL
5	15,8	SLASH
6	14,8	BACKSLASH

Çizelge 4.5. Piramit yapılı hatların düzgünleştirilmesi aşamasından sonra “+” karakterine ait diziler

DİZİ		
NUMARASI	DİZİNİN ELEMANLARI	DİZİNİN YÖNÜ
1	1,2,3,4,5,6,7	VERTICAL
2	7,8,9,10,11,12,13	VERTICAL
3	25,24,23,22,21,14,7	HORIZONTAL
4	7,15,16,17,18,19,20	HORIZONTAL

4.2.4. Karakter bilgisi taşımayan üç elemanlı gereksiz dizilerin silinmesi

4.2.4.1. Çapraz (slash/backslash) diziler üzerindeki işlemler

Kalan tüm diziler içinde, birden fazla ikili bağlantı içeren diziler incelenir. Böyle bir dizinin varsa üç elemanlı ve çapraz (SLASH veya BACKSLASH) yönlü dizisi(leri) bulunarak, sırayla analiz edilir. Bulunan çapraz dizi aşağıdaki özelliklere sahipse silinir.

1. Bu üç elemanlı, çapraz dizinin ikili bağlantılı olduğu diziler (Co_Scon’lar), ilgili dizinin (ilk bulunan dizi) ikili bağlantıları (Co_Scon’ları) içinde aranır.

2. Bulunan ikili bağlantılar “Direct Source” olarak işaretlenir.

3. Bulunamayan ikili bağlantılar, ilgili dizinin, bu çapraz dizi dışındaki, ikili bağlantı içerdiği dizilerin ikili bağlantıları içinde aranır. Aranan ikili hangi dizinin ikili bağlantısı ise, bu dizi yada diziler de “Indirect Source” olarak işaretlenir. Yani “Indirect Source” olarak birden fazla dizi bulunabilir.

4. “Direct” ve “Indirect Source” olarak işaretlenen tüm diziler bir tampon belleğe kaydedilir. Tampon belleğe daha önce yazılan dizi tekrar yazılmaz.

5. Üç elemanlı, çapraz dizinin “Direct Source” veya “Indirect Source” olarak işaretlenmemiş ikili bağlantısı yoksa, tampon belleğe kayıtlı diziler incelenir.

6. Bu incelemede çapraz dizinin, ilgili dizi ile ortak elemanı dışındaki tüm elemanları, tampon bellekteki dizilerde aranır. Bu işlem elemanların PreScon’larında, tampon belleğe kayıtlı diziler aranarak gerçekleştirilir.

7. Çapraz dizinin diğer elemanları tampon bellekteki dizilerin herhangi birinde bulunmuşsa bu üç elemanlı çapraz dizi silinir.

8. Bu üç elemanlı çapraz dizinin ikili bağlantısı yoksa dizinin ilgili dizi ile ortak olmayan elemanları, ilgili dizinin, bu çapraz dizi dışındaki, ikili bağlantı içerdiği dizilerde aranır. Yine bu elemanları PreScon’larından yararlanılır.

9. Çapraz dizinin diğer elemanları bu dizilerin herhangi birinde bulunmuşsa bu üç elemanlı çapraz dizi silinir.

10. Karakterdeki tüm çapraz diziler işlendikten sonra, tüm bağlantılar (PreScon ve Co_Scon) yeniden çıkarılır.

4.2.4.2. Düz hatlı (vertical/horizontal) diziler üzerindeki işlemler

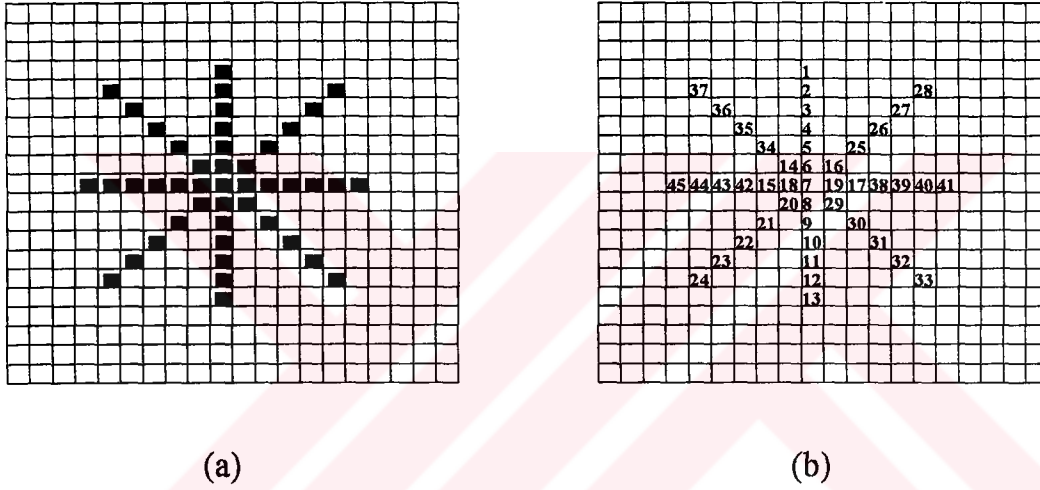
Kalan tüm diziler içinde, birden fazla ikili bağlantı içeren diziler incelenir. Böyle bir dizinin varsa üç elemanlı ve düz hatlı (VERTICAL veya HORIZONTAL) yönlü dizisi(leri) bulunarak, sırayla analiz edilir. Bulunan düz hatlı dizi aşağıdaki özelliklere sahipse silinir.

1. Bu üç elemanlı düz hatlı dizinin ikili bağlantısının olup olmadığına bakılmaksızın, bu dizinin ilgili dizi ile ortak olmayan elemanları, ilgili dizinin,

bu düz dizi dışındaki, ikili bağlantı içerdiği dizilerde aranır. Bu işlemde de elemanların PreScon'larındaki diziler kullanılır.

2. Düz hatlı dizinin diğer elemanları bu dizilerin herhangi birinde bulunmuşsa bu üç elemanlı düz hatlı dizi silinir.

3. Karakterdeki tüm düz diziler işlendikten sonra, tüm bağlantılar (PreScon ve Co_Scon) yeniden çıkarılır.



Şekil 4.12. Düzgün bir yıldız “*” karakterinden çıkarılan üç elemanlı gereksiz dizi örneği

(a) Zayıflatılmış bir “*” karakteri

(b) Karakterin numaralandırılmış hali

Karakter bilgisi taşımayan üç elemanlı gereksiz dizilerin silinmesi aşamasına en iyi örnek yıldız “*” karakteridir. Bu karakter üzerinde dört tanesi çapraz, dört tanesi düz hatlı olmak üzere toplam sekiz tane gereksiz üçlü dizi çıkar ve bunlar bu aşamada silinirler. Çizelge 4.6’da silinecek olan gereksiz üç elemanlı diziler koyu renkle belirtilmiştir. Şekil 4.12’de düzgün bir şekilde zayıflatılarak geldiği düşünülen “*” karakteri ve dizileri gösterilmiştir. Karakterden çıkarılan toplam dizi sayısının onaltı olduğu düşünülürse, silinen bu sekiz gereksiz dizi sayesinde toplam dizi sayısı sekize indirilmiş olur.

Çizelge 4.6. Şekil 4.12’deki yıldız “*” karakterinden çıkarılan tüm diziler

Dizi		
NUMARASI	DİZİNİN ELEMANLARI	DİZİNİN YÖNÜ
1	1,2,3,4,5,6,7,8,9,10,11,12,13	VERTICAL
2	5,14,15	SLASH
3	5,16,17	BACKSLASH
4	6,18	SLASH
5	6,19	BACKSLASH
6	14,6,16	HORIZONTAL
7	28,27,26,25,16,7,20,21,22,23,24	SLASH
8	37,36,35,34,14,7,29,30,31,32,33	BACKSLASH
9	45,44,43,42,15,18,7,19,17,38,39,40,41	HORIZONTAL
10	20,8,29	HORIZONTAL
11	19,8	SLASH
12	18,8	BACKSLASH
13	17,29,9	SLASH
14	15,20,9	BACKSLASH
15	14,18,20	VERTICAL
16	16,19,29	VERTICAL

4.2.5. Üç ve iki elemanlı diziler üzerindeki işlemler

4.2.5.1. Çıkıntı durumundaki üç elemanlı gereksiz dizilerin silinmesi

Karakterin herhangi bir yerinde, karaktere ait olmayan üç elemanlı dizi olarak ortaya çıkan dizilerin silinmesi gereklidir. Birden çok dizinin bağlantısını ifade eden fakat bir pixeli herhangi bir dizi ile bağlantı içermeyen üçlü dizilerin ikili dizi haline dönüştürülmesi lazımdır.

1. Tüm üçlü diziler incelenir.
2. İncelenen üçlü dizinin ençok bir elemanının başka herhangi bir dizi ile bağlantısı yoksa ve bu eleman dizinin başında ya da sonunda yer alıyorsa; bu eleman üçlü diziden silinir. Böylece üç elemanlı dizi iki elemanlı dizi haline gelir.
3. İncelenen üçlü dizinin ilk ve son elemanının başka herhangi bir dizi ile bağlantısı yoksa, bu üçlü dizi silinir.

4. Tüm dizi bağlantıları (PreScon ve Co_Scon) yeniden çıkarılır.

4.2.5.2. Bağlantı olarak mevcut olan gereksiz ikili dizilerin silinmesi

İki yada daha fazla dizinin uç uca değilde başka şekilde kesiştiği durumlarda, bu bağlantıyı gösteren ortak pixel dışında,yine birleşen bu diziler arasındaki bağlantıyı ifade eden, gereksiz iki elemanlı diziler ortaya çıkar. Halbuki bu gibi durumlarda ortak olan pixel sayesinde birleşim yapan diziler Co_Scon'lardaki dizi numaraları ile tespit edilmişlerdir. Bu nedenle bu türdeki iki elemanlı dizilere gerek yoktur ve aşağıdaki şekilde bu diziler silinir.

1. Tüm iki elemanlı diziler bulunarak, tek tek incelemeye alınır.
2. İncelenen iki elemanlı dizinin ilk elemanının bulunduğu dizi(ler) tespit edilir.
3. İncelenen iki elemanlı dizinin son elemanının bulunduğu dizi(ler) tespit edilir.
4. İlk elemanın bulunduğu dizilerden en az birinin, ikinci elemanın bulunduğu herhangi bir dizi ile ikili bağlantısı varsa bu iki elemanlı dizi silinir.
5. Bu durumdaki tüm iki elemanlı diziler silindikten sonra, tüm bağlantılar (PreScon ve Co_Scon) yeniden çıkarılır.

Bu aşama için yukarıda verilen yıldız “*”, örneğine devam edebiliriz (Bkz. Şekil 4.12, Çizelge 4.6). Çizelge 4.7’de görülen diziler içindeki dört tane iki elemanlı dizi de burada silinecektir. Silinecek olan gereksiz iki elemanlı

diziler koyu renkle belirtilmiştir. Böylece bu aşamanın sonunda bu karakter için dört tane dizi kalacaktır.

Çizelge 4.7. Şekil 4.12’de gösterilen yıldız “*” karakterinin silinecek gereksiz iki elemanlı dizileri

DİZİ NUMARASI	DİZİNİN ELEMANLARI	DİZİNİN YÖNÜ
1	1,2,3,4,5,6,7,8,9,10,11,12,13	VERTICAL
4	6,18	SLASH
5	6,19	BACKSLASH
7	28,27,26,25,16,7,20,21,22,23,24	SLASH
8	37,36,35,34,14,7,29,30,31,32,33	BACKSLASH
9	45,44,43,42,15,18,7,19,17,38,39,40,41	HORIZONTAL
11	19,8	SLASH
12	18,8	BACKSLASH

4.2.6. Aynı yöne sahip dizilerin iki elemanlı dizi yardımıyla birleştirilmesi

Diğer iki elemanlı diziler “Link” oluşturmak üzere incelenirler. İki elemanlı dizi, aynı yönlü iki dizinin uç uca bağlantılı olduğunu gösteriyorsa bu iki aynı yönlü dizi tek bir dizi haline getirilir. Link durumu için;

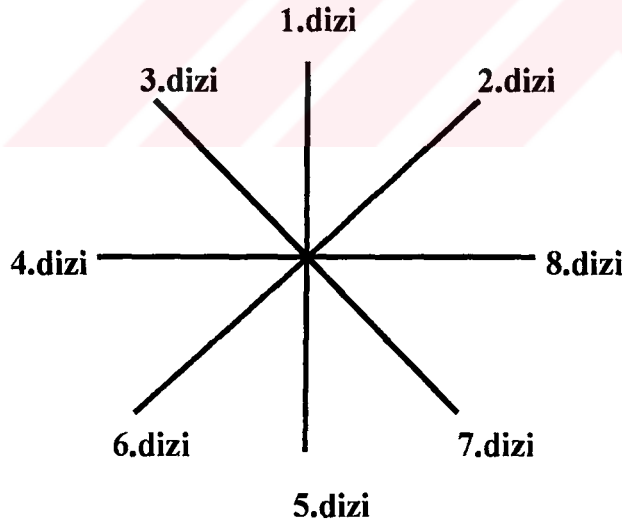
1. Kalan tüm iki elemanlı diziler incelenir.
2. Dizinin elemanlarının bulunduğu aynı yönlü diziler bulunur.
3. Bulunan aynı yönlü dizilerde, iki elemanlı dizinin elemanları başta ya da sonda yer alıyorsa; bu iki dizinin elemanları, yukarıdan-aşağıya, soldan-sağa yazılış sırası dikkate alınarak yeni bir dizi altında birleştirilirler.
4. Bir iki elemanlı dizi sadece bir link oluşturabilir ve bir dizi birden fazla linke dahil edilemez.

5. Bu linkin oluşmasını sağlayan iki elemanlı dizi, link adı altında birleştirilen diziler ve tüm bağlantıları iptal edilir.

6. Link durumunu sağlayan tüm diziler birleştirdikten sonra, tüm bağlantılar (PreScon ve Co_Scon) yeniden çıkarılır.

4.2.7. Kırılması gereken dizilerin kırılması

Uçuca bağlantı dışında herhangi bir orta noktasından başka dizi ya da dizilerle bağlantısı olan diziler kırılarak iki yeni dizi haline getirilir. Amaç karakter üzerindeki tüm dizilerin uçuca bağlantılı hale getirilmesidir. Bu, karakterin tanınması aşamasında diziler arasındaki bağlantıların bulunmasını kolaylaştıracaktır. Böylelikle mevcut tüm diziler birbirleri ile yalnızca uç uca bağlantı içereceklerdir.



Şekil 4.13. Yıldız “*” karakterinden elde edilen dizilere göre karakterin tanımlı son durumu

Bu aşama içinde yukarıda verilmiş olan yıldız “*” karakteri örneğine devam edelim (Bkz. Şekil 4.12, Çizelge 4.7). Hatırlayacağımız gibi karakterin son

olarak dört tane dizisi kalmıştı ve bu dizilerin hepsi birbiri ile orta noktalarından bağlantılı durumdaydı. Bu dizileri uçuca bağlantı hale getirmek için, kırmak gereklidir. Dolayısıyla bu dizilerin hepsi orta noktalarından kırılacak ve bu aşama sonunda herbiri uçuca bağlantı içeren sekiz tane dizi ortaya çıkacaktır (Çizelge 4.8). Bu aşamadan sonra ortaya çıkan bu dizilere göre yıldız “*” karakterinin tanımlı son durumu da Şekil 4.13’da ifade edilmeye çalışılmıştır.

Çizelge 4.8. Yıldız “*” karakterinin dört dizisinin orta noktalarından kırılmasıyla oluşan sekiz dizi

DİZİ	DİZİNİN ELEMANLARI	DİZİNİN YÖNÜ
1	1,2,3,4,5,6,7	VERTICAL
2	28,27,26,25,16,7	SLASH
3	37,36,35,34,14,7	BACKSLASH
4	45,44,43,42,15,18,7	HORIZONTAL
5	7,8,9,10,11,12,13	VERTICAL
6	7,20,21,22,23,24	SLASH
7	7,29,30,31,32,33	BACKSLASH
8	7,19,17,38,39,40,41	HORIZONTAL

4.2.8. Çıkıntı durumundaki iki elemanlı gereksiz dizilerin silinmesi

Karakterin herhangi bir yerinde, karakterin özelliğini taşımayan, çıkıntı şeklindeki iki elemanlı dizilerin silinmesi gereklidir. Bunların dışında herhangi iki dizinin uç uca olan bağlantısını gösteren iki elemanlı diziler de bağlantı kaybedilmemek şartıyla silinirler. Amaç karakteri ifade eden tüm dizilerin, minimum üç elemanlı dizi olarak ortaya çıkarılmasıdır. Çünkü karakterin özelliklerini gösteren iki elemanlı diziler, yalnızca bağlantı ifade ederler ve ifade ettikleri bağlantılar korunarak silinirler.

1. İki elemanlı diziler incelenir.

2. İncelenen ikili dizinin elemanlarından herhangi biri başka herhangi bir dizinin elemanı değilse bu ikili dizi silinir.

3. İncelenen ikili dizinin elemanlarının, ilk veya son eleman olarak bulundukları diğer diziler tespit edilir. Bu dizilerden elemanı az olan diziye(lere), ikili dizideki kendisi ile ortak olmayan eleman, yazılış sırası dikkate alınarak eklenir. Ve bu ikili dizi silinir.

4. Tüm iki elemanlı diziler bittikten sonra bağlantılar (PreScon ve Co_Scon) tekrar çıkarılır.

4.2.9. Karakterden çıkarılan tüm dizilerin sıralanması

Yukarıdaki tüm işlemlerden geçen zayıflatılmış karaktere ait tüm diziler, soldan_sağa, yukarıdan_aşağıya taramaya uygun olarak, başlangıç pixellerinin x,y koordinatına göre sıralanırlar. Diziler tanıma aşamasında bu sıraya uygun olarak incelenirler.

4.2.10. Diziler arası bağlantıların ve bağlantı konumlarının tespit edilmesi

Algoritmanın bu aşamasında, karakterin database’de tanımlanan özellikleri dikkate alınarak, tanınması için gerekli, sahip olduğu dizilere ait bağlantılar ve bu bağlantıların konumları çıkarılacaktır. Karakter tanıma aşamasında, bu bağlantıların database’deki karakter tanımına uygunluğu kontrol edilerek tanınacaktır.

Karaktere ait tüm diziler sıralamaya uygun olarak ele alınarak, diğer dizilerle olan bağlantıları ve bu bağlantıların konumları tespit edilir. Artık tüm diziler uçuca bağlantı içermektedir. Burada bağlantıyı gösteren pixel, dizinin ilk ve

bir sonraki veya son ve sondan bir önceki olabilir. Bağlantılar çıkarılırken bu duruma dikkat edilmelidir. Bir dizinin diğer dizi yada dizilerle olan bağlantısı ya başlangıç noktasından ya da bitiş noktasındandır. Başlangıç noktası “F”(First), bitiş noktası “L”(Last) olarak gösterilir ve bir dizi başka dizi yada dizilerle, dört konumda bağlantı içerebilir. Bunlar FF, FL, LF, LL bağlantılardır. Burada ilk harf ilgili dizinin bağlantı noktasını, ikinci harf ise bağlantılı olduğu dizinin bağlantı noktasını gösterir. Bunların herbiri birer dizi olarak ifade edilir ve içlerinde bağlantısını gösterdikleri dizi numara(larını) tutarlar. Aşağıdaki örnek bu durumu açıklamaktadır.

Çizelge 4.9. Bir dizinin diğer dizilerle bağlantılarını gösteren FF, FL, LF, LL dizileri

Dizi No	LF Dizileri	LL Dizileri	FF Dizileri	FL Dizileri
2	3	4,6	1	Null

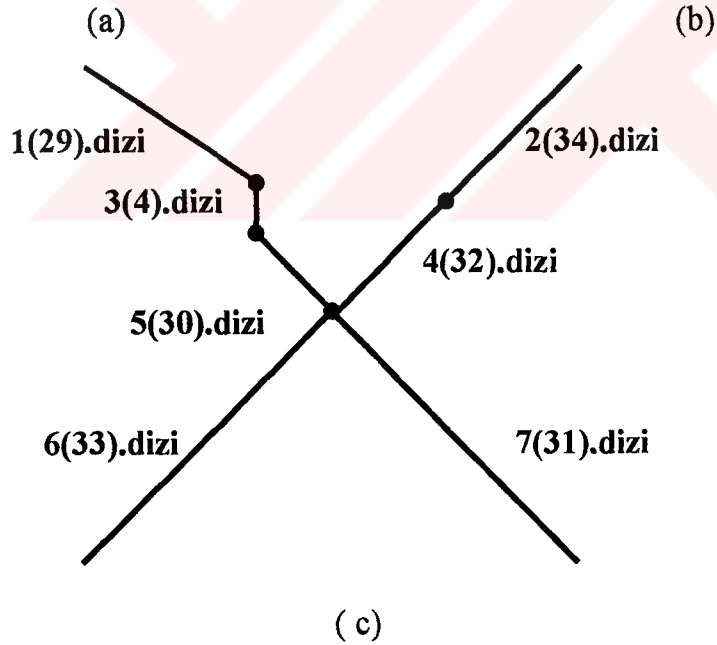
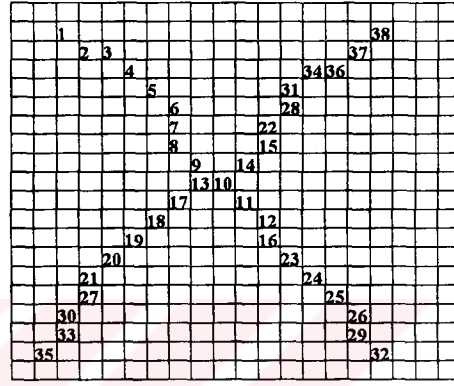
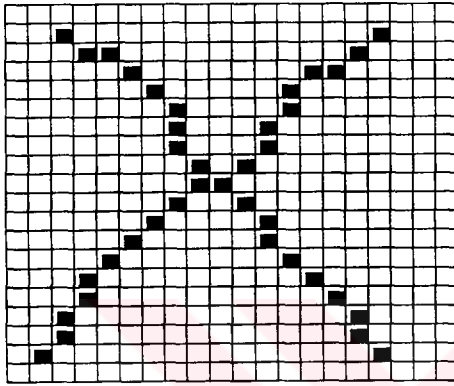
Yukarıdaki örneğe göre; 2 nolu dizi başlangıç noktasından(F) 1 nolu dizinin başlangıç(F) noktası ile bağlantılıdır ve “FF” dizisine yazılır, “FL” konumunda bağlantılı olduğu dizi yoktur. Bitiş noktasından(L) ise; 3 nolu dizinin başlangıç(F) noktasından, 4 ve 6 nolu dizilerin de bitiş(L) noktasından bağlantılıdır ve sırasıyla “LF” ve “LL” dizilerine yazılırlar. Daha iyi anlaşılması için bir “X” karakterine ait dizileri ve dizi bağlantılarını şekil 4.14’de ve çizelge 4.10, 4.11’de sunulmuştur.

Çizelge 4.10. “X” karakterinden çıkarılan dizilerin sıralanmış son durumu

DİZİ		
NUMARASI	DİZİNİN ELEMANLARI	DİZİNİN YÖNÜ
1(29)	1,2,3,4,5,6	BACKSLASH
2(34)	38,37,36,34,31,28	SLASH
3(4)	6,7,8	VERTICAL
4(32)	28,22,15,14,10,13	SLASH
5(30)	8,9,10,13	BACKSLASH
6(33)	13,17,18,19,20,21,27,30,33	SLASH
7(31)	13,10,11,12,16,23,24,25,26,29	BACKSLASH

Çizelge 4.11. “X” karakterine ait dizilerin, diğer dizilerle olan bağlantıları ve bağlantı konumları

DİZİ NUMARASI	LF DİZİLERİ	LL DİZİLERİ	FF DİZİLERİ	FL DİZİLERİ
1(29)	4	Nİİİ	Nİİİ	Nİİİ
2(34)	32	Nİİİ	Nİİİ	Nİİİ
3(4)	30	Nİİİ	Nİİİ	29
4(32)	31,33	30	Nİİİ	34
5(30)	31,33	32	Nİİİ	4
6(33)	Nİİİ	Nİİİ	31	30,32
7(31)	Nİİİ	Nİİİ	33	30,32



Şekil 4.14. Bir “X” karakterinin tüm aşamalar sonrası örneklenmesi

(a) Karakterin zayıflatılmış hali

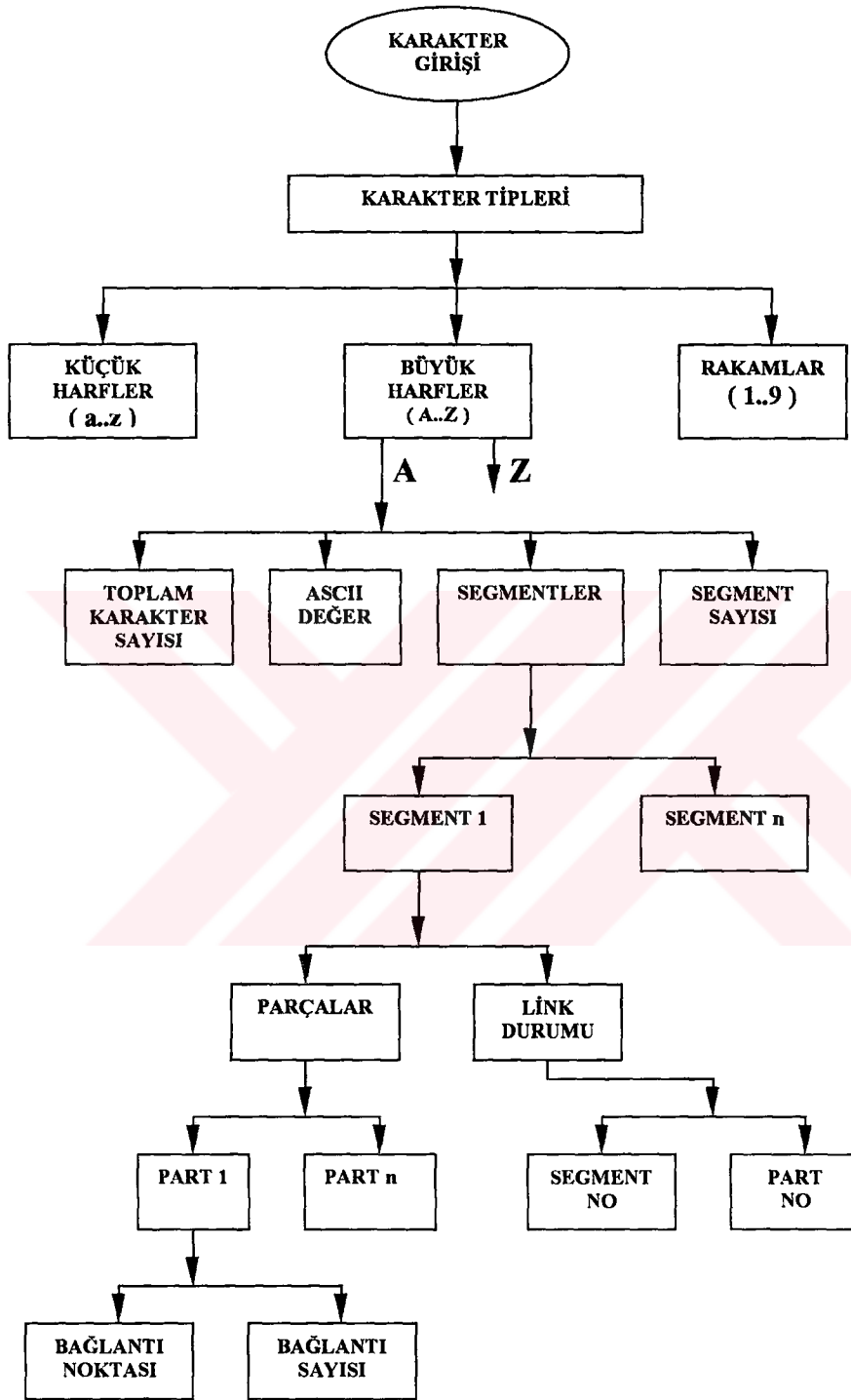
(b) Karakterin numaralandırılmış hali

(c) “X” karakterinin mevcut dizi ve bağlantılarına göre tanımlı durumu

4.3. Database ve Karakter Tanıma

Bu aşama karakterin tanındığı aşamadır. Karaktere ait diziler ve diziler arası bağlantılar bu aşamada, karakterin özelliklerini bildiren veriler olarak kullanılır. Bu bilgiler database'e yerleştirilen aynı türden karakter bilgileri ile karşılaştırılır ve tanıma gerçekleştirilir. İstendik karakter bilgilerinin yerleştirilmesi; Database ve Yapısı, karakter üzerinde yapılan segmentasyon ve tanımda dikkat edilecek hususlar; Segmentasyon ve Tanıma başlıkları adı altında açıklanmıştır.

Şekil 4.15'de, database'in genel yapısı görülmektedir. Database'de her bir karakter için; segmentler, segment sayısı, ASCII değeri, her bir segment için; sahip olduğu parçalar (diziler, parts), diğer segmentlerle olan bağlantıları ve bağlantı konumları, "part 1" ve "part n" parçaları için de; bağlantılı olduğu segment(ler)deki "part 1" ve "part n" parçaları arasındaki bağlantı sayısı ve konumları yer almaktadır. Bir segmentin sahip olduğu dizilere parçalar (parts) adı verilmiş ve sadece segmentin başlangıcında ve bitiminde yer alan parçalar (part 1, part n) tanımlanmıştır. Bu yapıdan anlaşıldığı gibi, tanınacak karakter için, algoritmadan elde edilen diziler ve dizi bağlantıları kullanılarak segmentler oluşturulmakta ve aslında database öncesinde karakter segmentler ve segment bağlantıları şeklinde ifade edilmektedir. Segmentasyonun yapılabilmesi için database'deki segment bilgileri referans alınmaktadır. Bu nedenle karakter, kendi özellikleri ile database'de mevcut özellikler birbirine uyana dek, değişik karakterler için değişik şekillerde segmente edilir ve karşılaştırılır. Yani tanınacak olan karakterin, database'de her bir karakter için mevcut segment ve segment bağlantılarından hangisini en iyi biçimde içerdiğine bakılır. Birden fazla karakter ile uygunluk durumu oluştuğunda ise en fazla uygunluk gösterdiği karakter olarak tanınır.

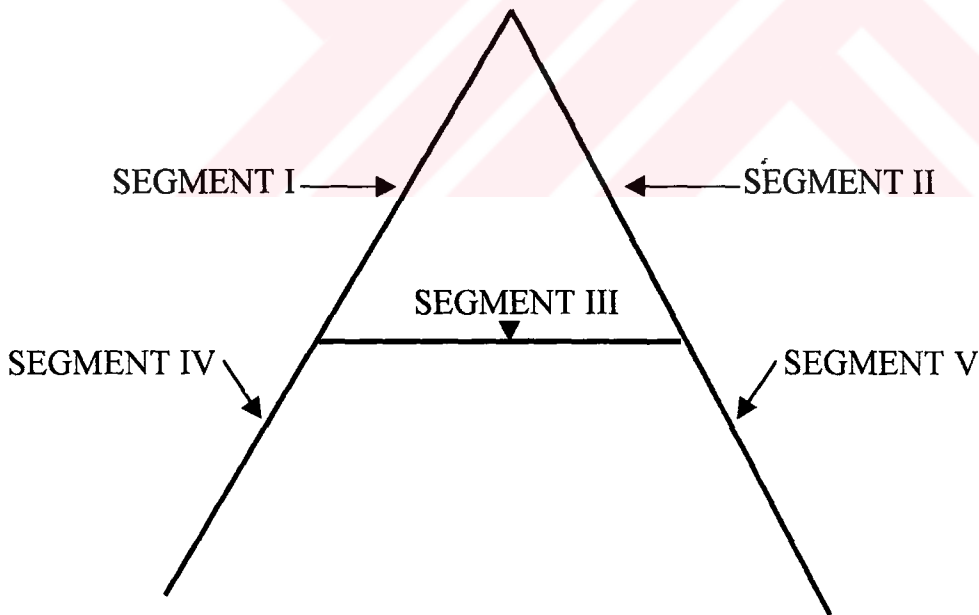


Şekil 4.15. Database'in genel yapısı

4.3.1. Database ve database yapısı

Database; karakterin temel yapısını ve istenen özelliklerini barındıran; tarafımızdan, algoritmanın özelliklerine göre oluşturulmuş, sabit bir bilgi tabanıdır. Bu yapıda karakterler parçalanmış, ki biz bu parçalara “Segment” diyoruz, ve bu parçaların özellikleri depolanmıştır. Her bir karakter parçası (segmenti) için, özellik olarak; bağlantı durumları ve konumları, karakter içindeki diğer segmentlerle olan bağlantıları ve gerekli görülen segmentler için de ek olarak segment yönleri (Slash, Backslash, Vertical, Horizontal) yer almaktadır. Buna göre bir karakter istendiği kadar parçalara ayrılabilir ya da bir karakter birden fazla tipte parçalanarak database’e yerleştirilebilir.

Aşağıda verilen örnekler datase’e yerleştirilen karakterlerin segmentlere nasıl ayrıldığını göstermektedir.

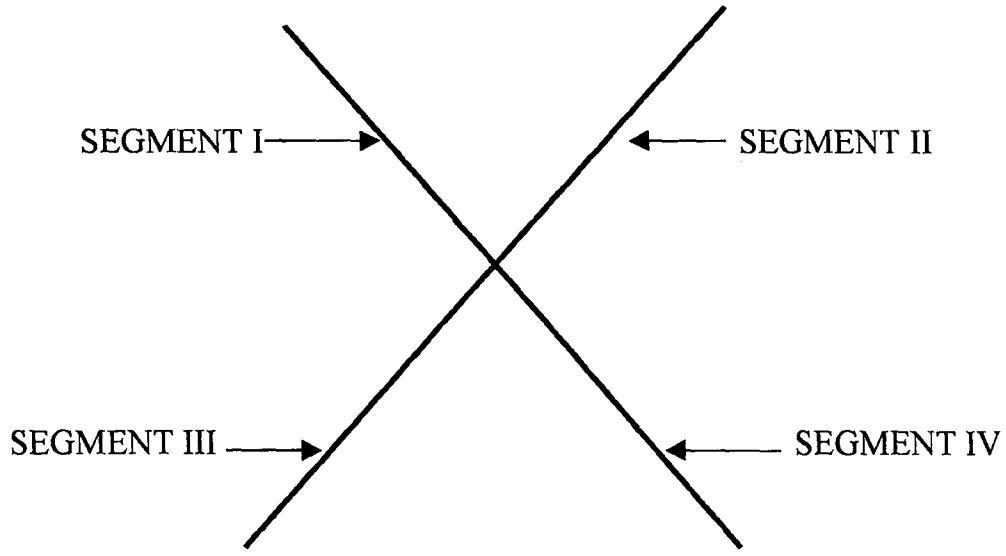


Şekil 4.16. Bir “A” karakterinin segmentlere ayrılmış hali

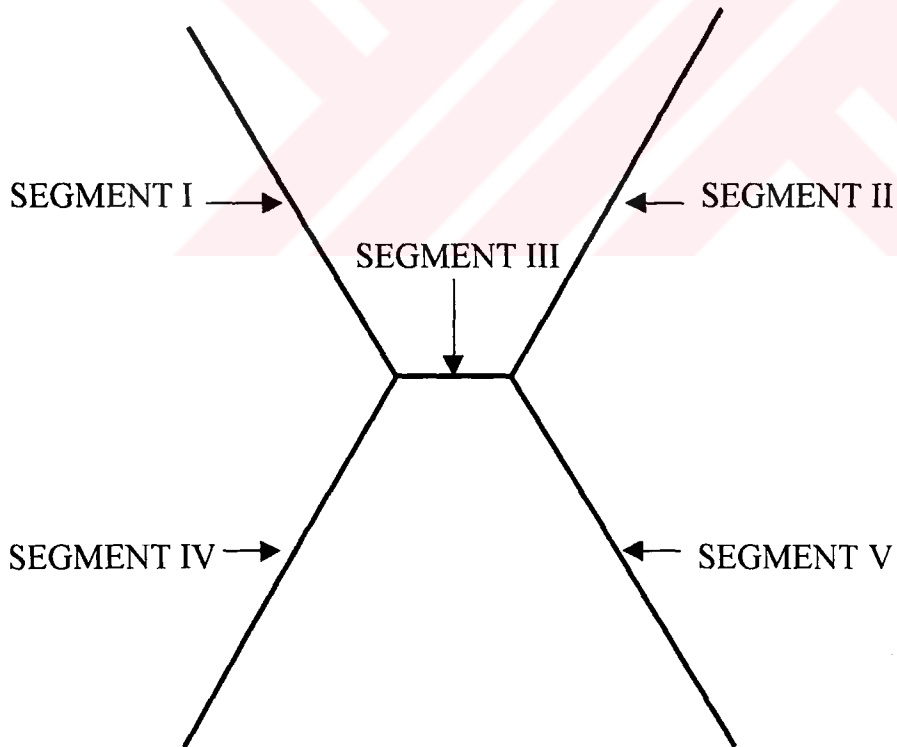
Çizelge 4.12. Şekil 4.16’da görülen beş segmente ayrılmış “A” karakterinin her bir segmentinin database’e yerleştirilen özellikleri

Segment Numarası	I	II	III	IV	V
Segmentin başlangıç noktası(F) bağlantı konumu(F,L)	F	F	F,L	F,L	L
Segmentin başlangıç noktası(F) bağlantı sayısı(F,L)	1, 0	1, 0	1, 1	1, 1	0, 2
Segmentin başlangıç noktasından(F) bağlantılı olduğu diğer segment(ler)(F,L)	II	I	IV, I	III, I	II, III
Segmentin bitiş noktası(L) bağlantı konumu(F,L)	F	F, L	F, L	Null	Null
Segmentin bitiş noktası(L) bağlantı sayısı(F,L)	2, 0	1, 1	1, 1	----	----
Segmentin bitiş noktasından(L) bağlantılı olduğu diğer segment(ler)(F,L)	III, IV	V, III	V, II	----	----

Her bir segment için, segmentin yukarıdan-aşağıya, soldan-sağa ilk karşılaşılan noktası başlangıç “F”, son noktasıda bitiş “L” noktasıdır. Segmentler başlangıç veya bitiş noktalarından, diğer segmentlerin yine başlangıç veya bitiş noktalarına bağlantılı olabilirler. Yukarıdaki çizelgeye bakarak; örneğin III nolu segmenti açıklarsak: bu segment başlangıç noktasından(F); I nolu segmentin bitiş(L), IV nolu segmentin de başlangıç(F) noktasına, bitiş noktasından da(L); II nolu segmentin bitiş(L), V nolu segmentin başlangıç noktasına(F) bağlantılıdır.



Şekil 4.17. Bir “X” karakterinin segmentlere ayrılmış hali



Şekil 4.18. Değişik bir “X” karakterinin segmentlere ayrılmış hali

Database’de “X” karakteri için birden fazla tanımlama yapılmıştır. Çünkü değişik “X” karakterlerinde mevcut segmentlerden daha fazla segment olabilir. Bu gibi durumlarda tanınacak olan “X” karakteri Şekil 4.17’deki gibi değilse, Şekil 4.18’deki database tanımı sayesinde tanınacaktır. Bir karakter için, istenilen sayıda database tanımlaması yapılabilir.

Çizelge 4.13. Şekil 4.17’de görülen dört segmente ayrılmış “X” karakterinin her bir segmentinin database’e yerleştirilen özellikleri

Segment Numarası	I	II	III	IV
Segmentin başlangıç noktası(F) bağlantı konumu(F,L)	Null	Null	F,L	F,L
Segmentin başlangıç noktası(F) bağlantı sayısı(F,L)	----	----	1, 2	1, 2
Segmentin başlangıç noktasından(F) bağlantılı olduğu diğer segment(ler)(F,L)	----	----	IV, I, II	III, I, II
Segmentin bitiş noktası(L) bağlantı konumu(F,L)	F,L	F, L	Null	Null
Segmentin bitiş noktası(L) bağlantı sayısı(F,L)	2, 1	2, 1	----	----
Segmentin bitiş noktasından(L) bağlantılı olduğu diğer segment(ler)(F,L)	III, IV, II	III, IV, I	----	----

Çizelge 4.14. Şekil 4.18’de görülen beş segmente ayrılmış “X” karakterinin her bir segmentinin database’e yerleştirilen özellikleri

Segment Numarası	I	II	III	IV	V
Segmentin başlangıç noktası(F) bağlantı konumu(F,L)	Null	Null	F, L	F,L	L
Segmentin başlangıç noktası(F) bağlantı sayısı(F,L)	----	----	1, 1	1, 1	0, 2
Segmentin başlangıç noktasından(F) bağlantılı olduğu diğer segment(ler)(F,L)	----	----	I, IV	III, I	II,V
Segmentin bitiş noktası(L) bağlantı konumu(F,L)	F	F, L	F,L	Null	Null
Segmentin bitiş noktası(L) bağlantı sayısı(F,L)	2, 0	1, 1	1, 1	----	----
Segmentin bitiş noktasından(L) bağlantılı olduğu diğer segment(ler)(F,L)	III, IV	V, III	V, II	----	----

4.3.2. Segmentasyon ve tanıma

Segmentasyon, karakterden çıkarılan diziler üzerinde, dizi bağlantıları yardımıyla yapılan bir grupta yada bir araya getirme işlemidir. Segmentasyon, OCR uygulamalarında “parçalara ayırma” anlamında kullanılan bir terimdir. Aslında burada da parçalama işlemi yapılmaktadır ancak, bu işlem mevcut bir yapının bölünmesi şeklinde değil, database’deki bilgilere göre aynı segmentte yer alan veya alabilecek dizilerin bir araya getirilmesi şeklindedir.

Segmentasyon, karakterin database'deki bilgiler türünden ifadesidir. Database'de her bir karakteri tanımlayan bilgiler, yukarıda açıklanmıştır (Bkz. Database ve yapısı). Burada her bir karakter segmentlerle ve segmentler arası bağlantılarla ifade edilmekte idi. Tanınacak karakter için önce database'de yerleşik ilk karakter bilgisi (segmentler ve bağlantıları) alınır ve alınan bu segment bilgilerine göre karakter segmente edilmeye çalışılır. Yani karaktere ait diziler bu bilgilere göre; bağlantılı oldukları diğer diziler takip edilerek segmentlere dahil edilirler. Database tarafından belirlenen özellikte segmentler oluştuktan sonra, bu segmentler arası ilişkiler (yani segment bağlantıları) yine database'den alınan bilgiler ışığında kontrol edilir. Bu şekilde yapılan tarama ve kontrollerde database bilgileri ile tanınacak karakter bilgileri birbirine uygunsa, herhangi bir eksiklik yoksa, tanınacak karakter; database'de bilgileri karşılaştırılan karakter olarak ortaya çıkarılır. Bu işlemlerde tam bir uyum söz konusu değilse ve birden fazla karakter üzerinde uyum sağlanıyorsa, bilinmeyen karakter en yüksek oranda uyum sağlanan karakter olarak tanınır.

5. SONUÇ VE ÖNERİLER

Karakter tanıma ya da diğer bir adıyla OCR uygulamaları oldukça zahmetli ve yorucu bir çalışma gerektirir. Çünkü bu tür uygulamalar geniş bir çalışma alanına yayılmıştır. Genel bir tanıma sistemi için karakter tanıma çalışması, tek başına yeterli olmaz. Tanıma çalışmasının; gürültü giderme, doküman tarama ve zayıflatma algoritmaları ile desteklenmesi gerekir ve bunların tamamının bir tanıma ya da OCR sistemi içinde bulunması gerekir. Fakat aynı zamanda bunların herbiri de başlıbaşına bir araştırma ve uygulama alanını oluşturur. Bu nedenle, bunlardan herhangi biri üzerinde yapılan bir çalışma diğer alanlarda yapılan çalışmalarla kombine edilir. Dolayısıyla oluşturulan kombinasyonun en iyiyi yakalaması istenir. Uygulamaların istenen sonuçları verebilmesi, her aşamanın dikkatle çalışılmasını gerektirir. Herhangi bir aşamada oluşabilecek en küçük bir istenmedik durum diğer aşamalar ne kadar da mükemmel olsa uygulamanın doğruluğunu ve performansını düşürür.

Bu uygulamada tanıma algoritması zayıflatılmış karakter üzerine kurulduğundan BLISS adı verilen yeni bir zayıflatma algoritması kullanılmıştır. Zhang and Suen, (1984) ve Li and Suen, (1991) tarafından üretilen zayıflatma algoritmaları ile karşılaştırıldığında kullanılan zayıflatma algoritmasının daha iyi iskeletler ürettiği görülmektedir. Ancak istenirse farklı zayıflatma algoritmaları da kullanılabilir.

Bu çalışmada karakter tanıma alanında DECRA adı verilen yeni bir algoritma ve yeni bir database yapısı üzerinde çalışılmıştır. DECRA algoritması zayıflatılmış olarak gelen karakteri, dizi adı verilen yapılarla ve bu diziler arası bağlantılarla, en sade şekilde ifade eder. Bu kısım tanıma için çok önemlidir. Çünkü zayıflatılarak gelen karakter tek başına yeterli olmaz. Karakter iskeletinin yani karakter bilgilerinin anlaşılır ve kullanılır duruma getirilmesi gereklidir. DECRA

algoritmasının yaptığı işlem üzerinde çalışan birçok başka algoritmalar da üretilmiştir. Ancak bunların çoğu ağır matematiksel işlemler ve fonksiyonlar gerektiren algoritmalar. Dolayısıyla bu algoritmalar üzerinde yapılan uygulamalar da zahmetli ve uzun süreli işlemler gerektirir. Yine çoğunluğunda da daha önce başka algoritmalarca kullanılmış bazı kural ve yapılar kullanılmış ve bu da bu algoritmaları bağımlı duruma getirmiştir (Bkz.Kaynaklar). DECRA ise mantıki işlemlere dayanan, karmaşık işlemler ve karışık matematiksel sonuçlar içermeyen, karakterin özelliklerini sergileyen anlaşılır ve kullanılması kolay sonuçlar üreten, geliştirilmesi de kolay olan bir algoritmadır.

Kullanılan database de DECRA gibi yeni bir yapı sergiler. Database yapısı basitçe çıkarılabilecek şekilde oluşturulmuştur. Bir karakter için birden fazla tanımlamaya olanak sağlaması da tanıma için bir avantajdır. Çünkü mevcut database'lerdeki en büyük problem, bir karakterin tek bir şekilde tanımlanabiliyor olmasından ileri gelmektedir. Bu tür database yapılarında karakterin olabilecek diğer durumları parçalar şeklinde aynı yere yerleştirilmekte, bu da işlemleri zorlaştırmakta ve eksik tanımlamalara sebep olmaktadır. Burada kullanılan database yapısı herne kadar basit, anlaşılır ve kolay kurulur bir yapıda da olsa bazı karakterlerin tanımlanması için daha fazla bilgi girişine ihtiyaç olabilmektedir. Bazı karakterler için gerekli olan bu fazla bilgilerin, database öncesinde yer alan algoritmaya dahil edilmesi bu durumu giderebilir.

Bu çalışmada doküman taraması yapılmamıştır. Çünkü doküman taraması, doküman tanınması problemini de beraberinde getirir. Doküman tanınması da en az burada sunulan çalışma kadar zahmetlidir ve kendi başına bir araştırma ve çalışma alanıdır. Bu nedenle, bu çalışmada üzerinde tanıma yapılan karakterler düzgün şekilli karakterlerdir. Tanıma alanında "Arial Black" fontu kullanılmıştır. Ancak el ile teorik olarak yapılan testlerde, olabildiğince zor ve problemli

karakterler seçilmiş ve algoritma bu zorluklara çözüm üretecek şekilde geliştirilmiştir. Dolayısıyla algoritmanın düzgün şekle sahip olmayan karakterler üzerinde de doğru tanımlar yaptığı, yapılan bu test sonuçlarında görülmektedir.

DECRA algoritmasının performansı daha iyi bir zayıflatma algoritması ve iyi bir doküman tanıma algoritması kullanılarak arttırılabilir. Database yapısı da istenilen şekilde geliştirilmeye müsaittir.



KAYNAKLAR

- Bose, C. B. and Kuo, S.-S., 1994, Connected and Degraded Text Recognition Using Hidden Markov Model, **Pattern Recognition Society**, Vol.27, No.10, 1345-1363.
- Calvert, C., 1999, Delphi 4 Unleashed, **Sams Borland Press**, 197-283.
- Chi, Z., Wu, J. and Yan, H., 1995, Handwritten Numeral Recognition Using Self-Organizing Maps and Fuzzy Rules, **Pattern Recognition Society**, Vol.28, No.1, 59-66.
- Fan, K.-C., Lin, C.-K., Chou, K.-S., 1995, Confusion Set Recognition of On-Line Chinese Characters By Artificial Intelligence Technique, **Pattern Recognition Society**, Vol.28, No.3, 303-313.
- Fu, K.S., 1982, Syntactic Pattern Recognition and Applications, **Prentice-Hall, Inc.**, 1-441, USA.
- Gümüş, İ., 1998, sözlü görüşme, Aydın Yazılım Elektronik Sanayi A.Ş. Beysukent-Ankara.
- Hu, J., Yan, H., 1998, Structural Primitive Extraction and Coding for Handwritten Numeral Recognition, **Pattern Recognition Society**, Vol.31, No.5, 493-509.
- Kim, H.J. and Kim. P.K., 1996, Recognition of Off-Line Handwritten Korean Characters, **Pattern Recognition Society**, Vol.29, No.2, 245-254.
- Kim, W.S., Park, R.-H., 1996, Off-Line Recognition of Handwritten Korean and Alphanumeric Characters Using Hidden Markov Models, **Pattern Recognition Society**, Vol.29, No.5, 845-858.
- Li, B. and Suen, C.Y., 1991, A Knowledge-Based Thinning Algorithm, **Pattern Recognition Society**, Vol.24, No.12, 1211-1221.
- Liang, S., Shridhar, M., Ahmadi, M., 1994, Segmentation of Touching Characters In Printed Document Recognition, **Pattern Recognition Society**, Vol.27, No.6, 825-840.

- Lu, S.Y. and Fu, K.S., 1978, A Sentence-to-Sentence Clustering Procedure for Pattern Analysis, **IEEE Transactions On Systems, Man, and Cybernetics**, Vol.SMC.8, No.5, 382-389.
- Mahmoud, S.A., 1994, Arabic Character Recognition Using Fourier Descriptors and Character Contour Encoding, **Pattern Recognition Society**, Vol.27, No.6, 815-824.
- Manber, U., 1989, Introduction to Algorithms A Creative Approach, **Addison-Wesley Publishing Comp.Inc.**, 61-86.
- Rabiner, L.R., 1989, A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition, **Proc. IEEE**, 77, 257-286.
- Reisdorph, K., 1998, Sams Teach Yourself Borland Delphi 4 in 21 Days, **Sams Publishing**, 659-747.
- Yanık, M., 1998, Borland Delphi 3 ile Görsel Programlama, **Beta Bas. Yay. Dağ. A.Ş.**, 559-760.
- Zadeh, L.A., 1965, Fuzzy Sets, **Inform.Contr.**, 8, 338-353.
- Zhang, T.Y. and Suen, C.Y., 1984, A Fast Parallel Algorithm for Thinning Digital Patterns, **Comm.ACM.**, Vol.27, No.3, 236-239.

EKLER

Ek-1 : DECRA algoritmasının program kodu

Ek-2 : DECRA için hazırlanan database yapısı



EK-1

DECRA ALGORİTMASININ PROGRAM KODU

```

(*****)
(*          AN OPTICAL CHARACTER RECOGNITION ALGORITHM          *)
(*          BASED ON STRING EXPRESSIONS                          *)
(*****)
(* Programmer   : DERYA YILMAZ                                   *)
(* Birth-Date   : January, in the second weekend in 99          *)
(* Coding-Date  : 03/13/1999                                     *)
(* Last Updated : 08/28/1999                                     *)
(*****)

unit DeryaAlg;

interface

(*****)
(*          ==>  CONSTANTS  <==                                *)
(*****)
Const

  C_NUMBER_OF_ARRAY      = 200;
  C_SKELETON_BUFFER_WIDE = 20;
  C_NUMBER_OF_DIRECTION = 8;
  C_NUMBER_OF_MEMBERS    = 50;
  C_NUMBER_OF_DARK_PIXELS = C_SKELETON_BUFFER_WIDE *
                           C_SKELETON_BUFFER_WIDE;
  { Steps in X and Y coordinates for each direction }
  Dir : array [ 1..C_NUMBER_OF_DIRECTION ] of record
    x, y : smallint;
  end =
  (
    ( x: - 1; y: + 1 ), { South-West }
    ( x:   0; y: + 1 ), { South }
    ( x: + 1; y: + 1 ), { South-East }
    ( x: + 1; y:   0 ), { East }
    ( x: + 1; y: - 1 ), { North-East }
    ( x:   0; y: - 1 ), { North }
    ( x: - 1; y: - 1 ), { North-West }
    ( x: - 1; y:   0 ) { West }
  );

(*****)
(*          ==>  TYPE DECLERATIONS  <==                          *)
(*****)
type

  { to give a direction, for array }
  TDIRECTION = ( NONE, VERTICAL, HORIZONTAL, SLASH, BACKSLASH );
  SCON_STRUCTURE =

```

```

    record
        Scon : Array [ 1..C_NUMBER_OF_MEMBERS ] of boolean;
        Scon_Count : smallint;
        Co_Scon : Array [ 1..C_NUMBER_OF_MEMBERS ] of smallint;
        Co_Scon_Count : smallint;
    end;

    { An Array Structure }
    BUFFER_STRUCTURE =
    record
        Not_Clean : boolean;
        Enable : boolean;
        { for faster algorithm and linking }
        Cleaned : boolean;
        { for break }
        Broken_Last : boolean;
        Broken_First : boolean;
        TTA_First : boolean;
        TTA_Second : boolean;
        Members : array [ 1..C_NUMBER_OF_MEMBERS ] of smallint;
        Size : smallint;
        Dir : TDIRECTION;
        Scons : SCON_STRUCTURE;
    end;

    {A buffer to find pairs, relationship between two different
    existing arrays}
    PRIMARY_SCON = record
        Array_Numbers : array [ 1..C_NUMBER_OF_DARK_PIXELS ] of smallint;
        Enable : array [ 1..C_NUMBER_OF_DARK_PIXELS ] of boolean;
        Count : smallint;
    end;

    { need its own value of each pixel due to our Algorithm }
    PROPERTIES_OF_CALCULATED_PIXEL = record
        Around : array [ 1..C_NUMBER_OF_DIRECTION ] of boolean;
        Co_x : smallint;
        Co_y : smallint;
        Enable : boolean;
    end;

    end;

    TXY = record
        x , y : smallint;
    end;

    TARRAYS = record
        Array_No : smallint;
        Array_Type : TDirection;
        Enable : boolean;
    end;

    TADVARRAYS = record
        Ordered_Arrays : Array [ 1..10 ] of
            record
                FArrays : Array [ 1..4 ] of smallint;
            end;
    end;

```

```

        end;

TARRAYLINK = record

    Inuse    : boolean;
    Point    : Array [ 1..50 ] of boolean;
    FFArray , FLArray ,
    LFArray , LLArray : record
        Link    : Array [ 1..10 ] of smallint;
        Count    : smallint;
        Pointer : Array [ 1..10 ] of boolean;
        end;
    end;

type
TDUMMY_SEGMENT_PARTS = record
    Back : smallint;
    end;

(*****
(*                               ==>  VARIABLES  <==                               *)
(*****)
var
    Array_Number      : smallint;

    Main_Buffers      : array [ 1..C_NUMBER_OF_ARRAY ] of
        BUFFER_STRUCTURE;

    Skeleton          : array [ 1..C_SKELETON_BUFFER_WIDE,
        1..C_SKELETON_BUFFER_WIDE ] of
        boolean;

    Pixels_Calculated : array [ 1..C_SKELETON_BUFFER_WIDE,
        1..C_SKELETON_BUFFER_WIDE ] of
        smallint;

    PreSCON           : array [ 1..C_NUMBER_OF_DARK_PIXELS ] of
        PRIMARY_SCN;

    Pixel_Number      : smallint;

    Pixel_Around      : array [ 1..C_NUMBER_OF_DARK_PIXELS ] of
        PROPERTIES_OF_CALCULATED_PIXEL;

    Pixel_Around_Count : smallint;

    ASMResult         : boolean;

    Dummy_Pixel_Number : smallint;

    Coordinated_Pixels : Array [ 1..C_NUMBER_OF_DARK_PIXELS ] of TXY;

    Typed_Array       : Array [ 1..50 ] of TArrays;

    No_TArray         : smallint;

```

```

Sorted_Arrays      : Array [ 1..50 ] of TAdvArrays;
Array_Links        : Array [ 1..50 ] of TArrayLink;
Sort_Typed_Arrays  : Array [ 1..50 ] of smallint;
index_Module       : smallint;
index_Order_No     : smallint;
Segment_Start      : Array [ 1..20 ] of smallint;
Segment_Finish     : Array [ 1..20 ] of smallint;
Segment_Start_Pointer : Array [ 1..20 ] of boolean;
Segment_Finish_Pointer : Array [ 1..20 ] of boolean;
Locked_Arrays      : Array [ 1..100 ] of boolean;
InUse              : Array [ 1..100 ] of boolean;
Dummy_Segment_Parts : Array [ 1..10, 1..10, 1..10 ] of smallint;
Found_Segment_Parts : Array [ 1..10 ] of record
    Records : Array [ 1..10 ] of boolean;
    Size    : smallint;
end;
Dummy_Segment      : Array [ 1..50 ] of TDummy_Segment_Parts;

procedure Algorithm_of_Derya ( Width, Height : smallint );
Function Calculating_Founding_Extracting ( Width, Height : smallint
):boolean;
    procedure Cleanup_ThreeTuple_Arrays;
    procedure Removing_Pyramids_to_Land;
    procedure Removing_Triple_SBS_Array;
    procedure Removing_Triple_HV_Array;
    procedure Cleanup_Unneccassary_TwoTuple_Arrays_Which_Confirm_Before;
    procedure Join_The_Same_Dir_Arrays;
    procedure Cleanup_TwoTuple_Arrays;
    Procedure Reconstruction_Given_Character(Width,Height : smallint);
    procedure Catching_Scons;
    procedure Breaking_The_UnBreaking_Arrays;
    procedure Transferring_Arrays;
    procedure Sorting_X_Via_Y (Width, Height : smallint);
    procedure Defines_Arrays_Link;
    procedure Sorted_Typed_Arrays;
    procedure Module_Founder ( Start, LF, LL, FF, FL : smallint;
                                Other_Links          : boolean );
Function Find_Relevant_Segment ( Rec_Segment : TSegment;
                                Module_No    : smallint
                                ): boolean;

```

implementation

uses Dialogs, Unit1 , sysutils, DB;

```

(*****
(*)          ==>  TYPE DECLERATIONS <==          (*)
(*****
type

TCon_State = ( Nil1, First, Last );
TTypeofChar = ( All,LowerCase,UpperCase );

TPART  = record
    Con_Point : Array [ 1..2 ] of TCon_State;
    No_Con : Array [ 1..2 ] of smallint;
    Con_Point_Dir : Array [ 1..2 ] of TDirection;
end;

TLINKED  = record
    Segment_No : Array [ 1..4 ] of smallint;
    Part : Array [ 1..4 ] of TCon_State;
end;

TSEGMENT  = record
    Part1      : TPart;
    PartN      : TPart;
    Linked     : TLinked;
    Seg_Dir    : TDirection;
end;

TCHARACTER_INFO  = record
    ASCII_Value : smallint;
    No_Segment   : smallint;
    Segments     : Array [ 1..10 ] of TSegment;
end;

TCHARACTER_TYPES  = record
    Lower_Cases   : Array [ 1..9 ] of TCharacter_Info;
    Upper_Cases   : Array [ 1..9 ] of TCharacter_Info;
    Numeric_Char   : Array [ 1..10] of TCharacter_Info;
    Total_No_Chars : Array [ 1..3 ] of smallint;
end;

(*****
(*)          ==>  VARIABLES <==          (*)
(*****
var
    DBOCR : Array [ 1..62 ] of TCharacter_Info;

```

```

(*****)
(*          -->    FINDING PIXEL AROUND    <--          *)
(*****)
(* Programmer    : Derya Yilmaz                      *)
(* Coding-Date   : 03/13/1999                        *)
(* Last Updated  : 03/17/1999                        *)
(*****)
Procedure Finding_Pixel_Around ( Current_x, Current_y,
                                Dummy_Pixel_Number : smallint );

```

```

var
    way : smallint;

begin
    for way := 1 to C_NUMBER_OF_DIRECTION do
        if Skeleton [ Current_x + Dir [ way ].
            x, Current_y + Dir [ way ].y ] = false then
            Pixel_Around [ Dummy_Pixel_Number ].Around [ way ] := false;
    end;

```

```

(*****)
(*          -->    RESET ALL BUFFERS    <--          *)
(*****)
(* Programmer    : Derya Yilmaz                      *)
(* Coding-Date   : 03/13/1999                        *)
(* Last Updated  : 03/17/1999                        *)
(*****)

```

```

procedure Reset_All_Buffers;

```

```

var
    index : smallint;

begin
    fillchar ( Main_Buffers, sizeof ( Main_Buffers ), 0 );
    fillchar ( PreScon, sizeof ( PreScon ), 0 );
    fillchar ( Pixels_Calculated, sizeof ( Pixels_Calculated ), 0 );
    fillchar ( Pixel_Around, sizeof ( Pixel_Around ), 1 );
    for index := 1 to C_NUMBER_OF_DARK_PIXELS do
        begin
            Pixel_Around [ index ].Enable := false;
            Pixel_Around [ index ].Co_x := 0;
            Pixel_Around [ index ].Co_y := 0;
        end;
    end;
end;

```

```

(*****)
(*          -->    UPDATE PRESCONS    <--          *)
(*****)
(* Programmer    : Derya Yilmaz                      *)
(* Coding-Date   : 04/03/1999                        *)
(* Last Updated  : 04/08/1999                        *)
(*****)

```

```

procedure Update_Prescons;

```

```

var
    index_Of_Array_Number : smallint;

```

```

Members_Count      : smallint;
Co_Array_Number    : smallint;
begin
  index_of_Array_Number := 1;
  fillchar ( Prescon , sizeof ( Prescon ) , 0 );
  repeat
    Members_Count := 1;
    if Main_Buffers [ index_of_Array_Number ].Enable = true then
      begin
        repeat
          Co_Array_Number := Main_Buffers [ index_of_Array_Number ].
            Members [ Members_Count ];
          inc ( Prescon [ Co_Array_Number ].Count );
          Prescon [ Co_Array_Number ].
            Array_Numbers [ Prescon [ Co_Array_Number ].
              Count ] := index_of_Array_Number;
          PreSCON [ Co_Array_Number ].
            Enable [ index_of_Array_Number ] := true;
          inc ( Members_Count );
        until Members_Count > Main_Buffers [ index_of_Array_Number ].
          Size;
      end;
      inc ( index_of_Array_Number );
    until index_of_Array_Number = Array_Number;
  end;

```

```

(*****)
(*          -->  CATCHING SCONS  <--          *)
(*****)
(* Programmer   : Derya Yilmaz                  *)
(* Coding-Date  : 03/19/1999                    *)
(* Last Updated : 04/10/1999                    *)
(*****)

```

```

procedure Catching_Scons;
var
  Scon_Members      : smallint;
  Index_of_Array_Number : smallint;
  Index_of_Array_Size : smallint;
  Index_of_PreScon_Count : smallint;
  index             : smallint;
  index_of_Scon      : smallint;

begin
  for index := 1 to Array_Number - 1 do
    begin
      Main_Buffers [ index ].Scons.Scon_Count := 0;
      Main_Buffers [ index ].Scons.Co_Scon_Count := 0;
    end;
    for index := 1 to Array_Number - 1 do
      for index_of_Scon := 1 to C_NUMBER_OF_MEMBERS do
        Main_Buffers [ index ].Scons.Scon [ index_of_Scon ] := false;
      end;
    end;
  end;
  Index_of_Array_Number := 1;
  repeat
    Index_of_Array_Size := 1;

```

```

if Main_Buffers [ index_of_Array_Number ].Enable = true then
begin
  repeat
    Index_of_PreScon_Count := 1;
    repeat
      Scon_Members :=
        PreScon [ Main_Buffers [ Index_of_Array_Number ].
          Members [ Index_of_Array_Size ] ].
          Array_Numbers [ Index_of_PreScon_Count ];
      if ( Scon_Members <> Index_of_Array_Number ) and
        ( Main_Buffers [ index_of_Array_Number ].Scons.
          Scon [ Scon_Members ] = false ) then
      begin
        inc ( Main_Buffers [ Index_of_Array_Number ].Scons.
          Scon_Count );
        Main_Buffers [ Index_of_Array_Number ].Scons.
          Scon [ Scon_Members ] := true;
        inc ( Main_Buffers [ Index_of_Array_Number ].
          Scons.Co_Scon_Count );
        Main_Buffers [ Index_of_Array_Number ].Scons.
          Co_Scon [ Main_Buffers [ Index_of_Array_Number ].
            Scons.Co_Scon_Count ] := Scon_Members;
      end;
      inc ( Index_of_PreScon_Count );
    until Index_of_PreScon_Count >
      PreScon [ Main_Buffers [ Index_of_Array_Number ].
        Members [ Index_of_Array_Size ] ].Count;
    inc ( Index_of_Array_Size );
  until Index_of_Array_Size =
    Main_Buffers [ Index_of_Array_Number ].Size + 1;
end;
inc ( Index_of_Array_Number );
until Index_of_Array_Number = Array_Number;
end;

```

```

(*****)
(*      --> NECESSARY VALUES FOR RECONSTRUCTION STATE  <--      *)
(*****)
(* Programmer   : Derya Yilmaz                                     *)
(* Coding-Date  : 06/02/1999                                       *)
(* Last Updated : 06/02/1999                                       *)
(*****)
Function Necessary_Values_for_Recon_State( x,y:smallint ):smallint;
var
  Res : smallint;

begin
  Res := 0;
  if Skeleton [ x - 1, y - 1 ] then Res := Res or 1;
  if Skeleton [ x     , y - 1 ] then Res := Res or 2;
  if Skeleton [ x + 1, y - 1 ] then Res := Res or 4;
  if Skeleton [ x + 1, y     ] then Res := Res or 8;
  if Skeleton [ x + 1, y + 1 ] then Res := Res or 16;
  if Skeleton [ x     , y + 1 ] then Res := Res or 32;
  if Skeleton [ x - 1, y + 1 ] then Res := Res or 64;

```

```

if Skeleton [ x - 1, y ] then Res := Res or 128;
Necessary_Values_for_Recon_State := Res;
end;

```

```

(*****
(*          --> RECONSTRUCTION GIVEN CHARACTER  <--          *)
(*****
(* Programmer   : Derya Yilmaz                               *)
(* Coding-Date  : 06/02/1999                                 *)
(* Last Updated : 06/02/1999                                 *)
(*****
Procedure Reconstruction_Given_Character( Width,Height:smallint );
var
  x, y : smallint;

begin
  for y := 1 to Height do
    for x := 1 to Width do
      begin
        if skeleton [ x, y ] then
          begin
            Case Necessary_Values_for_Recon_State ( x, y ) of
              80 : begin
                if Necessary_Values_for_Recon_State( x, y + 1 )
                  <> 170 then
                  skeleton [ x, y ] := false;
                  skeleton [ x, y + 1 ]:= true;
                end;
              20 : begin
                if Necessary_Values_for_Recon_State( x + 1, y )
                  <> 170 then
                  skeleton [ x, y ] := false;
                  skeleton [ x + 1, y ]:= true;
                end;
              65 : begin
                if Necessary_Values_for_Recon_State( x - 1, y )
                  <> 170 then
                  skeleton [ x, y ] := false;
                  skeleton [ x - 1, y ]:= true;
                end;
              5 : begin
                if Necessary_Values_for_Recon_State( x, y - 1 )
                  <> 170 then
                  skeleton [ x, y ] := false;
                  skeleton [ x, y - 1 ]:= true;
                end;
              21 : begin
                if Necessary_Values_for_Recon_State( x + 1, y - 1 )
                  <> 64 then
                  begin
                    skeleton [ x, y ] := false;
                    skeleton [ x + 1, y ]:= true;
                    skeleton [ x, y - 1 ]:= true;
                  end;
                end;
            end;
          end;
        end;
      end;
    end;
  end;
end;

```

```

69 : begin
    if Necessary_Values_for_Recon_State( x - 1, y - 1 )
        <> 16 then
        begin
            skeleton [ x, y ] := false;
            skeleton [ x - 1, y ]:= true;
            skeleton [ x, y - 1 ]:= true;
        end;
    end;
84 : begin
    if Necessary_Values_for_Recon_State( x + 1, y + 1 )
        <> 1 then
        begin
            skeleton [ x, y ] := false;
            skeleton [ x + 1, y ]:= true;
            skeleton [ x, y + 1 ]:= true;
        end;
    end;
81 : begin
    if Necessary_Values_for_Recon_State( x - 1, y + 1 )
        <> 4 then
        begin
            skeleton [ x, y ] := false;
            skeleton [ x - 1, y ]:= true;
            skeleton [ x, y + 1 ]:= true;
        end;
    end;
end;
end else
begin
    if Necessary_Values_for_Recon_State ( x, y ) in [ 170,
        169, 172, 154,202, 178, 166, 43, 106 ] then
        skeleton [ x, y ] := true;
    end;
end;
end;
end;

```

```

(*****)
(*      -->  CALCULATING & FOUNDING & EXTRACTING  <--      *)
(*****)
(* Programmer   : Derya Yilmaz                               *)
(* Coding-Date  : 03/10/1999                                  *)
(* Last Updated : 05/01/1999                                  *)
(*****)

```

Function

Calculating_Founding_Extracting(Width,Height:smallint):boolean;

var

```

Cur_x, Cur_y           : smallint;
Dummy_cur_x, Dummy_cur_y : smallint;
x, y, dx, dy, Count    : smallint;
Opposite_Way           : smallint;
Aborted_Way            : smallint;
Second_Times           : boolean;

```

begin

```

FillChar ( DBOCR, Sizeof ( DBOCR ), 0 );
Pixel_Number := 0;
Array_Number := 1;
Reset_All_Buffers;
x := 0;
y := 0;

Repeat
  asm
  {
    - find a non-valued pixel
    - a found pixel will be given the last value of pixel number
    - it ll be given a value ( Co_x, Co_y )
    - Create boolean ASMResult whether to go on
  }

    { Saving the current used regs' values to stack }
    push ax
    push bx
    push dx
    push esi
    push edi

    xor esi, esi
    xor edi, edi
    mov si, y
    mov di, x
@loop2: push di
    imul di, C_SKELETON_BUFFER_WIDE
@loop6: mov al, byte ptr [ esi + edi + Skeleton ]
    pop di
    cmp al, 0
    jne @loop1
@loop5: inc di
    cmp di, Width
    jne @loop2
    inc si
    cmp si, Height
    je @loop3
    mov edi, 0
    jmp @loop2
@loop1: push di
    imul di, 2
    imul di, C_SKELETON_BUFFER_WIDE
    push si
    imul si, 2
    mov ax, word ptr [ Pixels_Calculated + esi + edi ]
    pop si
    pop di
    cmp ax, 0
    je @loop4
    jmp @loop5

    { No more dark-pixel to calculate }
@loop3: mov ASMResult, 0
    jmp @exit

```

```

        { found a new dark-pixel hasnt got value, up to now }
@loop4: mov  ax, Pixel_Number
        inc  ax
        mov  word ptr Pixel_Number , ax
        push di
        imul di, 2
        imul di, C_SKELETON_BUFFER_WIDE
        push si
        imul si, 2
        mov  word ptr [ Pixels_Calculated + esi + edi ], ax
        pop  si
        pop  di
        inc  di
        mov  word ptr Cur_x, di
        mov  x, di
        mov  word ptr Cur_y, si
        mov  y, si
        inc  word ptr Cur_y
        mov  ASMResult, 1

@exit:  { Getting original regs' values from stack }
        pop  edi
        pop  esi
        pop  dx
        pop  bx
        pop  ax

end;

{ start a new pixel }
if ASMResult = True then
begin
    Pixel_Around [ Pixel_Number ].Enable := true;
    Pixel_Around [ Pixel_Number ].Co_x := Cur_x;
    Pixel_Around [ Pixel_Number ].Co_y := Cur_y;
    Coordinated_Pixels [ Pixel_Number ].x := Cur_x;
    Coordinated_Pixels [ Pixel_Number ].y := Cur_y;
    Dummy_Pixel_Number := Pixel_Number;

    { executing phase for new pixel(s) }
    While Pixel_Around [ Dummy_Pixel_Number ].Enable = true do
    begin
        { initializing properties of the dark pixel }
        Pixel_Around_Count := 1;
        Cur_x := Pixel_Around [ Dummy_Pixel_Number ].Co_x;
        Cur_y := Pixel_Around [ Dummy_Pixel_Number ].Co_y;
        Opposite_Way := 0;
        Finding_Pixel_Around ( Cur_x, Cur_y, Dummy_Pixel_Number );

        { searhing all eight ways of pixel to find the arrays }
        While ( Pixel_Around_Count <= C_NUMBER_OF_DIRECTION ) do
        begin
            if ( Pixel_Around [ Dummy_Pixel_Number ].
                Around [ Pixel_Around_Count + Opposite_Way ] ) then
            begin
                dx := Dir [ Pixel_Around_Count ].x;

```

```

dy := Dir [ Pixel_Around_Count ].y;
Dummy_cur_x := Cur_x + dx;
Dummy_cur_y := Cur_y + dy;
Second_Times := true;

While Second_Times do
begin
  Repeat
    if Pixels_Calculated [ Dummy_cur_x, Dummy_cur_y ] = 0 then
    begin
      inc ( Pixel_Number );

      Pixels_Calculated [ Dummy_cur_x, Dummy_cur_y ] :=
        Pixel_number;

      Coordinated_Pixels [ Pixel_Number ].x :=
        Dummy_Cur_x;

      Coordinated_Pixels [ Pixel_Number ].y :=
        Dummy_Cur_y;
      Pixel_Around [ Pixel_Number ].Enable := true;

      Pixel_Around [ Pixel_Number ].Co_x := Dummy_cur_x;
      Pixel_Around [ Pixel_Number ].Co_y := Dummy_cur_y;
    end;

    if Pixel_Around_Count in [ 1..4 ] then
      Aborted_Way := 4
    else Aborted_Way := -4 ;

    Pixel_Around [ Pixels_Calculated
      [ Dummy_cur_x, Dummy_cur_y ] ].
      Around [ Pixel_Around_Count + Aborted_Way ] := false;

    Pixel_Around [ Pixels_Calculated
      [ Dummy_cur_x, Dummy_cur_y ] ].
      Around [ Pixel_Around_Count ] := false;

    inc ( Dummy_cur_x, dx );
    inc ( Dummy_cur_y, dy );
  Until Skeleton [ Dummy_cur_x, Dummy_cur_y ] = false;

  { Look for Opposite-way to go OK }
  if ( Pixel_Around_Count + 4 < 9 ) and
    ( Pixel_Around [ Dummy_Pixel_Number ].
      Around [ Pixel_Around_Count + 4 ] ) then
  begin
    Opposite_Way := 4;
    dx := Dir [ Pixel_Around_Count + Opposite_Way ].x ;
    dy := Dir [ Pixel_Around_Count + Opposite_Way ].y ;
    Dummy_cur_x := Cur_x;
    Dummy_cur_y := Cur_y;

    Pixel_Around [ Dummy_Pixel_Number ].
      Around [ Pixel_Around_Count + 4 ] := false;

```

```

        inc ( Dummy_cur_x, dx );
        inc ( Dummy_cur_y, dy );
    end else Second_Times := false;
end;

{ catching array in relevant way }
case Pixel_Around_Count of
    1..4 : begin
        dx := Dir [ Pixel_Around_Count ].x;
        dy := Dir [ Pixel_Around_Count ].y;
        if Skeleton[Cur_x-dx, Cur_y-dy]=false then
            begin
                Dummy_cur_x := Cur_x - dx;
                Dummy_cur_y := Cur_y - dy;
            end;
        end;
    5..8 : begin
        dx := Dir [ Pixel_Around_Count - 4 ].x;
        dy := Dir [ Pixel_Around_Count - 4 ].y;
        end;
end;

Count := 0;
inc ( Dummy_cur_x, dx );
inc ( Dummy_cur_y, dy );
if Skeleton [ Dummy_cur_x, Dummy_cur_y ] then
    begin
        while Skeleton [ Dummy_cur_x, Dummy_cur_y ] do
            begin
                inc ( count );
            end;
        { Controlling to find large Skeleton not to be damaging }
        if Count >= C_NUMBER_OF_MEMBERS then
            begin
                MessageDlg('Out of C_NUMBER_OF_MEMBERS',
                    mtInformation, [mbOk], 0);
                Calculating_Founding_Extracting := false;
                exit;
            end;
        Main_Buffers [ Array_Number ].Members [ count ] :=
            Pixels_Calculated [ Dummy_cur_x, Dummy_cur_y ];

        inc ( PreSCON [ Pixels_Calculated [ Dummy_cur_x,
            Dummy_cur_y ] ].count );

        PreSCON [ Pixels_Calculated
            [ Dummy_cur_x, Dummy_cur_y ] ].
            Array_Numbers [ PreSCON [ Pixels_Calculated
            [ Dummy_cur_x, Dummy_cur_y ] ].count ] := Array_Number;
        PreSCON [ Pixels_Calculated
            [ Dummy_cur_x, Dummy_cur_y ] ].
            Enable [ Array_Number ] := true;
        inc ( Dummy_cur_x, dx );
        inc ( Dummy_cur_y, dy );
    end;

    { Setting up the size and way of array and others}

```

```

Main_Buffers [ Array_Number ].Size := count;
Main_Buffers [ Array_Number ].Enable := true;

Case Pixel_Around_Count of
  1,5 : Main_Buffers[ Array_Number ].Dir:= SLASH;
  2,6 : Main_Buffers[ Array_Number ].Dir:= VERTICAL;
  3,7 : Main_Buffers[ Array_Number ].Dir:= BACKSLASH;
  4,8 : Main_Buffers[ Array_Number ].Dir:= HORIZONTAL;
end;
inc ( Array_Number );
{ Controlling to find large Skeleton not to damage }
if Count >= C_NUMBER_OF_ARRAY then
begin
  MessageDlg('Out of C_NUMBER_OF_ARRAY',mtInformation,
    [mbOk], 0);
  Calculating_Founding_Extracting := false;
  exit;
end;
end;
inc ( Pixel_Around_Count );
Opposite_Way := 0;
end;
inc ( Dummy_Pixel_Number );
end;
end;
Until not ASMResult;
if Array_Number > 1 then
  Calculating_Founding_Extracting := true
else
  Calculating_Founding_Extracting := false;
end;

(*****)
(*          --> CLEAR UNNECESSARY ARRAYS  <--          *)
(*****)
(* Programmer   : Derya Yilmaz                          *)
(* Coding-Date   : 04/03/1999                            *)
(* Last Updated  : 06/02/1999                            *)
(*****)

procedure Clear_Unnecassary_Arrays;
var
  index_Of_Array_Number : smallint;
begin
  index_Of_Array_Number := 1;
  repeat
    if ( Main_Buffers [ index_Of_Array_Number ].Cleaned ) then
      Main_Buffers [ index_Of_Array_Number ].Enable := false;
    inc ( index_Of_Array_Number );
  until index_Of_Array_Number = Array_Number;
  Update_PreScons;
  Catching_Scons;
end;

```

```

(*****)
(*      -->  REMOVING PYRAMIDS TO LAND  <--      *)
(*****)
(* Programmer   : Derya Yilmaz                      *)
(* Coding-Date  : 03/24/1999                        *)
(* Last Updated : 06/02/1999                        *)
(*****)
procedure Removing_Pyramids_to_Land;
var
  Index_of_Array_Number      : smallint;
  First_Array, Second_Array  : smallint;
  First_Secondary_Array,
    Second_Secondary_Array   : smallint;
  Way_to_Write               : boolean; {true=LR+RR;false=RR+LR}
  Way_to_Found               : boolean;
  Common_Member              : smallint;
  First_Other_Member,
    Second_Other_Member      : smallint;
  First_Found, Second_Found  : boolean;
  Index                      : smallint; {Count for Gen. Purpose}
  Scon_Members, Scon_Mem_Count : smallint;
  Mainway_to_Found           : TDirection;
  index_second               : smallint;
  Co_Array_Number            : smallint;
  Dummy_Array_Number         : smallint;
  Real_First, Real_Second    : smallint;
  Dummy_Join_Array           : smallint;
  Value                      : smallint;
  Exit_From_One              : boolean;
  index_For_Equal            : smallint;
  Exit_Process               : boolean;
  Not_Broken                 : boolean;
  Cur_Array                  : smallint;
  Really                     : boolean;
  index_dec                  : smallint;

begin
  Index_of_Array_Number := 1;
  Co_Array_Number := Array_Number;
  Dummy_Array_Number := Array_Number;
  Repeat
    if ( Main_Buffers [ Index_of_Array_Number ].Size = 2 )
      AND
      (
        ( Main_Buffers [ Index_of_Array_Number ].Dir = SLASH ) or
        ( Main_Buffers [ Index_of_Array_Number ].Dir = BACKSLASH )
      )
      AND
      (Main_Buffers[Index_of_Array_Number].Scons.Co_Scon_Count > 0)
    then
      begin
        First_Secondary_Array := Index_of_Array_Number;
        Scon_Mem_Count := 1;
        repeat
          Scon_Members:= Main_Buffers [ Index_of_Array_Number ].Scons.
            Co_Scon [ Scon_Mem_Count ];

```

```

if (
    ( Main_Buffers [ Scon_Members ].Dir = SLASH ) or
    ( Main_Buffers [ Scon_Members ].Dir = BACKSLASH )
)
    AND
    ( Main_Buffers [ Scon_Members ].Size = 2 ) then
begin
    Second_Secondary_Array := Scon_Members;
    if ( Main_Buffers [First_Secondary_Array].Members [ 2 ] =
        Main_Buffers [Second_Secondary_Array.Members [ 1 ] ]
        or
        ( Main_Buffers [First_Secondary_Array].Members [ 1 ] =
          Main_Buffers [Second_Secondary_Array].Members [ 2 ] )
        then
        Way_to_Found := true { VL }
    else
        Way_to_Found := false; { HL }
    if Main_Buffers [First_Secondary_Array].Dir = BACKSLASH
    then
    begin
        asm
            { First_Secondary_Array is now SLASH }
            push ax
            mov ax, First_Secondary_Array
            xchg ax, Second_Secondary_Array
            mov First_Secondary_Array, ax
            pop ax
        end;
    end;
    if Way_to_Found then { VL }
    begin
        if Main_Buffers [First_Secondary_Array].Members [ 1 ] =
            Main_Buffers [Second_Secondary_Array].Members [ 2 ]
        then
        begin
            Way_to_Write := true; { LR + RR }
            Common_Member := Main_Buffers [First_Secondary_Array].
                Members [ 1 ];
            First_Other_Member := Main_Buffers
                [Second_Secondary_Array].Members [ 1 ];
            Second_Other_Member := Main_Buffers
                [First_Secondary_Array].Members [ 2 ];
            Real_First := Second_Secondary_Array;
            Real_Second := First_Secondary_Array;
        end else
        begin
            Way_to_Write := false; { RR + LR }
            Common_Member := Main_Buffers
                [ First_Secondary_Array ].Members [ 2 ];
            First_Other_Member := Main_Buffers
                [ First_Secondary_Array ].Members [ 1 ];
            Second_Other_Member := Main_Buffers
                [ Second_Secondary_Array ].Members [ 2 ];
            Real_First := First_Secondary_Array;
            Real_Second := Second_Secondary_Array;
        end;
    end;
end;

```

```

    Mainway_to_Found := VERTICAL;
end else
begin
    if Main_Buffers [First_Secondary_Array].Members [ 1 ] =
        Main_Buffers [ Second_Secondary_Array ].
            Members [ 1 ] then
    begin
        Way_to_Write := false; { RR + LR }
        Common_Member := Main_Buffers
            [ First_Secondary_Array ].Members [ 1 ];
        First_Other_Member := Main_Buffers
            [ First_Secondary_Array ].Members [ 2 ];
        Second_Other_Member := Main_Buffers
            [ Second_Secondary_Array ].Members [ 2 ];
        Real_First := First_Secondary_Array;
        Real_Second := Second_Secondary_Array;
    end else
    begin
        Way_to_Write := true; { LR + RR }
        Common_Member := Main_Buffers
            [ First_Secondary_Array ].Members [ 2 ];
        First_Other_Member := Main_Buffers
            [ Second_Secondary_Array ].Members [ 1 ];
        Second_Other_Member := Main_Buffers
            [ First_Secondary_Array ].Members [ 1 ];
        Real_First := Second_Secondary_Array;
        Real_Second := First_Secondary_Array;
    end;
    Mainway_to_Found := HORIZONTAL;
end;
index := 1;
First_Found := false;
repeat
    if Main_Buffers [ PreScon [ First_Other_Member ].
        Array_Numbers[ index ] ].Dir = Mainway_to_Found
    then
    begin
        First_Array := PreScon [ First_Other_Member ].
            Array_Numbers [ index ];
        First_Found := true;
    end else
        inc ( index );
until ( First_Found ) or
    ( index > PreScon [ First_Other_Member ].Count );

index := 1;
Second_Found := false;
repeat
    if Main_Buffers [ PreScon [ Second_Other_Member ].
        Array_Numbers [ index ] ].Dir = Mainway_to_Found
    then
    begin
        Second_Array := PreScon [ Second_Other_Member ].
            Array_Numbers [ index ];
        Second_Found := true;
    end else

```

```

    inc ( index );
until ( Second_Found ) or
    ( index > PreScon [ Second_Other_Member ].Count );

Really := false;
if ( PreScon [ common_Member ].Count <> 2 ) then
begin
    index := 1;
    repeat
        Cur_Array := PreScon [ common_Member ].
            Array_Numbers [ index ];
        if ( Cur_Array <> Real_First ) and
            ( Cur_Array <> Real_Second ) and
            ( Main_Buffers [ Cur_Array ].Size > 2 ) then
            Really := true;
        inc ( index );
    until ( index > PreScon [ common_Member ].Count ) or
        ( Really );
end;

if ( ( First_Found = true ) or ( Second_Found = true ) )
    and
    ( Really ) and ( First_Array = Second_Array ) then
begin
    index := 1;
    Exit_Process := false;
    { Discover the pixel whether is break-point }
    repeat
        value := Main_Buffers [First_Array].Members [index];
        if ( index > 1 ) and ( Main_Buffers [ First_Array ].
            Members [ index - 1 ] = First_Other_Member ) then
            begin
                if ( ( value = Main_Buffers [ First_Array ].
                    Members [ 1 ] ) and
                    ( PreScon [ Value ].Count = 2 ) )
                    or
                    ( ( value = Main_Buffers [ First_Array ].
                        Members [ 2 ] ) and
                    ( PreScon [ Main_Buffers [ First_Array ].
                        Members [ 1 ] ].Count = 2 ) )
                    or
                    ( ( value = Main_Buffers [ First_Array ].
                        Members [ Main_Buffers [First_Array ].Size ] )
                    and
                    ( PreScon [ Value ].Count = 2 ) )
                    or
                    ( ( value = Main_Buffers [ First_Array ].
                        Members [ Main_Buffers [ First_Array ].
                            Size - 1 ] ) and
                    ( PreScon [ Main_Buffers [ First_Array ].
                        Members [ Main_Buffers [ First_Array ].
                            Size ] ].Count = 2 ) )
                    then
                    Not_Broken := true
                else
                    Not_Broken := false;
            end
        else
            Not_Broken := false;
    until ( index > Main_Buffers [ First_Array ].Members.Count );
end;

```

```

        Exit_Process := true;
    end;
    inc ( index );
until Exit_Process;
if ( Not_Broken = false ) and
    (not Main_Buffers [ First_Array ].Cleaned and
     not Main_Buffers [Second_Array ].Cleaned )
    then
begin
    index_for_Equal := 0;
    Exit_from_one := false;
    index := 1;
    { The first array is being created until break-point }
    repeat
        value := Main_Buffers [First_Array].Members [index];
        Main_Buffers[Co_Array_Number].Members[index] := value;
        if ( index > 1 ) and ( Main_Buffers [ First_Array ].
            Members [ index - 1 ] = First_Other_Member ) then
            begin
                Main_Buffers [ Co_Array_Number ].Enable := true;
                Main_Buffers [ Co_Array_Number ].Size := index ;
                if Mainway_to_Found = VERTICAL then
                    Main_Buffers [Co_Array_Number].Dir:= VERTICAL
                else
                    Main_Buffers [Co_Array_Number].Dir:= HORIZONTAL;
                Main_Buffers [Co_Array_Number].Not_Clean := true;
                Main_Buffers [ Real_First ].Cleaned := true;

                if Main_Buffers [Co_Array_Number].Size > 2 then
                    Main_Buffers [Co_Array_Number].Broken_Last :=
                        true;
                index_for_Equal := index;
                Exit_from_one := true;
                inc ( Co_Array_Number );
            end;
        inc ( index );
    until ( Exit_from_One ) or
        ( index > Main_Buffers [ First_Array ].Size );
    index := 1;
    { The other array is also being created until last point }
    if Exit_from_One then
        begin
            repeat
                Main_Buffers [Co_Array_Number].Members [index]:=
                    Main_Buffers [ First_Array ].
                        Members [ index_for_Equal ];
                inc ( index );
                inc ( index_for_Equal );
            until index_for_Equal > Main_Buffers [ First_Array ].
                Size;
            Main_Buffers [ Co_Array_Number ].Enable := true;
            Main_Buffers [ Co_Array_Number ].Size := index - 1;
            if Mainway_to_Found = VERTICAL then
                Main_Buffers [ Co_Array_Number ].Dir := VERTICAL
            else
                Main_Buffers [ Co_Array_Number ].Dir := HORIZONTAL;
        end
    end
end

```

```

Main_Buffers [ Co_Array_Number ].Not_Clean := true;
Main_Buffers [ Real_Second ].Cleaned := true;
if Main_Buffers [ Co_Array_Number ].Size > 2 then
    Main_Buffers[Co_Array_Number].Broken_First:=true;
    inc ( Co_Array_Number );
Main_Buffers [ First_Array ].Enable := false;
Array_Number := Co_Array_Number;
Update_Prescons;
Catching_Scons;
end;
end else
    { Otherwise clear only Two-tuples array }
begin
    if Main_Buffers [ First_Array ].Members [ 1 ] =
        First_Other_Member then
        begin
            for index_dec := 1 to
                Main_Buffers[ First_Array ].Size - 1 do
                Main_Buffers[First_Array].Members[index_dec] :=
                    Main_Buffers[First_Array].Members[index_dec+1];
                dec ( Main_Buffers [ First_Array ]. Size );
            end else
                if Main_Buffers [First_Array].Members [Main_Buffers
                    [First_Array].Size] = Second_Other_Member then
                    dec ( Main_Buffers [ First_Array ]. Size );
                Main_Buffers [ Real_First ].Cleaned := true;
                Main_Buffers [ Real_Second ].Cleaned := true;
            end;
        end else
            if ( PreScon [ common_Member ].Count <> 2 ) then
            begin
                if ( First_Found and Second_Found = true ) then
                begin
                    Main_Buffers [ Real_First ].Cleaned := true;
                    Main_Buffers [ Real_Second ].Cleaned := true;
                end else
                begin
                    if ( PreScon [ First_Other_Member ].count = 1 ) then
                    begin
                        Main_Buffers [ Real_First ].Cleaned := true;
                    end;
                    if ( PreScon [ Second_Other_Member ].count = 1 ) then
                    begin
                        Main_Buffers [ Real_Second ].Cleaned := true;
                    end;
                end;
            end else
            begin
                if ( PreScon [ common_Member ].Count = 2 ) then
                begin
                    Main_Buffers [ Real_First ].Cleaned := true;
                    Main_Buffers [ Real_Second ].Cleaned := true;
                end;
            end;
        end;
    First_Secondary_Array := Index_of_Array_Number;
    inc ( Scon_Mem_Count );
until Scon_Mem_Count >

```

```

        Main_Buffers [ Index_of_Array_Number ].Scons.Co_Scon_Count;
    end;
    inc ( Index_of_Array_Number );
until Dummy_Array_Number = Index_of_Array_Number;
Clear_Unnecessary_Arrays;
end;

(*****
(*          -->  REMOVING TRIPLEX SBS ARRAY  <--          *)
(*****
(* Programmer   : Derya Yilmaz                               *)
(* Coding-Date  : 04/06/1999                               *)
(* Last Updated : 06/02/1999                               *)
(*****

```

Procedure Removing_Triple_SBS_Array;

type

```

    DI_Array_Structure = record
        Member : Array [ 1..C_NUMBER_OF_MEMBERS ] of smallint;
        Size   : smallint;
        Enable  : Array [ 1..C_NUMBER_OF_MEMBERS ] of boolean;
    end;

```

var

```

    index_of_Array_Number : smallint;
    index_of_CoScon       : smallint;
    value_of_CoScon       : smallint;
    index_of_VOC          : smallint; { VOC : Value_of_CoScon }
    DI_Array              : DI_Array_Structure;
    DI_Array_Count        : smallint;
    index_of_Main_CoScon  : smallint;
    indirect_Search       : boolean;
    not_This_CoScon       : boolean;
    index_of_DI_Array     : smallint;
    Scan_Pixel_Count      : smallint;
    Pixel_Checked         : boolean;
    index                 : smallint;

```

begin

```

    index_of_Array_Number := 1;
    repeat
        if ( Main_Buffers [ index_of_Array_Number ].Enable = true ) and
            ( Main_Buffers [ index_of_Array_Number ].
              Scons.Co_Scon_Count > 1 ) then
            begin
                index_of_CoScon := 1;
                Not_This_CoScon := true;
                repeat
                    value_of_CoScon := Main_Buffers [ index_of_Array_Number ].
                                          Scons.Co_Scon [ index_of_CoScon ];
                    if ( Main_Buffers [ value_of_CoScon ].Size = 3 )
                        and
                        (
                            ( Main_Buffers [value_of_CoScon].Dir = BACKSLASH ) or
                            ( Main_Buffers [value_of_CoScon].Dir = SLASH )
                        ) then

```

```

begin
  Pixel_Checked := false;
  if Main_Buffers [ value_of_CoScon ].
    Scons.Co_Scon_Count <> 0 then
  begin
    for index := 1 to C_NUMBER_OF_MEMBERS do
    begin
      DI_Array.Enable [ index ] := false;
      DI_Array.Member [ index ] := 0;
    end;
    DI_Array.Size := 0;
    index_of_VOC := 1;
    DI_Array_Count := 1;
    Not_This_CoScon := false;
    repeat
      indirect_Search := true;
      if Main_Buffers [ index_of_Array_Number ].Scons.
        Scon [ Main_Buffers [ value_of_CoScon ].Scons.
          Co_Scon [ index_of_VOC ] ] = true then
      begin { direct search }
        if DI_Array.Enable [Main_Buffers [value_of_CoScon].
          Scons.Co_Scon [index_of_VOC ] ] = false then
        begin
          inc ( DI_Array.Size );
          DI_Array.Member[ DI_Array.Size ] :=
            Main_Buffers [ value_of_CoScon ].Scons.
              Co_Scon [ index_of_VOC ];
          DI_Array.Enable [ Main_Buffers [value_of_CoScon].
            Scons.Co_Scon [ index_of_VOC ] ] := true;
        end;
      end else
      { searching CoScon as indirectly }
      begin
        indirect_Search := false;
        index_of_Main_CoScon := 1;
        repeat
          if ( Main_Buffers [ index_of_Array_Number ].Scons.
            Co_Scon [ index_of_Main_CoScon ] <>
              value_of_CoScon )
            AND
            ( Main_Buffers [
              Main_Buffers [ index_of_Array_Number ].
                Scons.Co_Scon [ index_of_Main_CoScon ]
              ].Scons.
                Scon [
                  Main_Buffers [ value_of_CoScon ].Scons.
                    Co_Scon [ index_of_VOC ]
                ] = true
            ) then
          begin
            indirect_Search := true;
            if DI_Array.Enable [ Main_Buffers
              [index_of_Array_Number].Scons.
                Co_Scon [index_of_Main_CoScon] ] = false
            then
              begin

```

```

        inc ( DI_Array.Size );
        DI_Array.Member [ DI_Array.Size ] :=
            Main_Buffers [ index_of_Array_Number ].
            Scons.Co_Scon [ index_of_Main_CoScon ];
        DI_Array.Enable [ Main_Buffers
            [index_of_Array_Number].Scons.
            Co_Scon [index_of_Main_CoScon] ] := true;
    end;
end;
inc ( index_of_Main_CoScon );
until ( index_of_Main_CoScon >
    Main_Buffers [ index_of_Array_Number ].
    Scons.Co_Scon_Count );
if (not indirect_Search) then Not_This_CoScon:=true;
end;
inc ( index_of_VOC );
until ( not indirect_Search ) or
    ( index_of_VOC > Main_Buffers [ value_of_CoScon ].
        Scons.Co_Scon_Count );
if ( not Not_This_CoScon ) then
begin
    Pixel_Checked := false;
    Scan_Pixel_Count := 1;
    repeat
        if not PreScon [ Main_Buffers [ Value_of_CoScon ].
            Members [ Scan_Pixel_Count ] ].
            Enable [ index_of_Array_Number ] then
        begin
            Pixel_Checked := false;
            index_of_DI_Array := 1;
            repeat
                if PreScon [ Main_Buffers [ Value_of_CoScon ].
                    Members [ Scan_Pixel_Count ] ].
                    Enable [ DI_Array.
                        Member [ index_of_DI_Array ] ] then
                    Pixel_Checked := true;
                    inc ( index_of_DI_Array );
                until ( Pixel_Checked ) or
                    ( index_of_DI_Array > DI_Array.Size );
            end else
                Pixel_Checked := true;
            inc ( Scan_Pixel_Count );
        until ( not Pixel_Checked ) or
            ( Scan_Pixel_Count > Main_Buffers [ Value_of_CoScon ].
                Size );
    end;
end else
begin
    { No This Array's CoScons }
    Pixel_Checked := false;
    Scan_Pixel_Count := 1;
    repeat
        if not PreScon [ Main_Buffers [ Value_of_CoScon ].
            Members [ Scan_Pixel_Count ] ].
            Enable [ index_of_Array_Number ] then
        begin

```

```

Pixel_Checked := false;
index_of_Main_CoScon := 1;
repeat
  if PreScon [ Main_Buffers [ Value_of_CoScon ].
    Members [ Scan_Pixel_Count ] ].
    Enable [Main_Buffers[index_of_Array_Number].
      Scons.Co_Scon [index_of_Main_CoScon] ] then
    Pixel_Checked := true;
    inc ( index_of_Main_CoScon );
until ( Pixel_Checked ) or
  ( index_of_Main_CoScon >
    Main_Buffers [ index_of_Array_Number ].
      Scons.Co_Scon_Count );
end else
  Pixel_Checked := true;
  inc ( Scan_Pixel_Count );
until ( not Pixel_Checked ) or
  ( Scan_Pixel_Count >
    Main_Buffers [ Value_of_CoScon ].Size );
end;
{ This CoScon will be being removed }
if Pixel_Checked = true then
begin
  Main_Buffers [ value_of_CoScon ].Enable := false;
  { Start from head of point of current Array }
  index_of_CoScon := 0;
  Update_Prescons;
  Catching_Scons;
end;
end;
inc ( index_of_CoScon );
until ( index_of_CoScon >
  Main_Buffers [ index_of_Array_Number ].Scons.
    Co_Scon_Count );
end;
inc ( index_of_Array_Number );
until ( index_of_Array_Number = Array_Number );
end;

```

```

(*****)
(*          -->  REMOVING TRIPLEX HV ARRAY  <--          *)
(*****)
(* Programmer   : Derya Yilmaz                          *)
(* Coding-Date   : 04/06/1999                            *)
(* Last Updated  : 04/10/1999                            *)
(* Revision      : 1.0                                    *)
(*****)

```

Procedure Removing_Triple_HV_Array;

var

```

  index_of_Array_Number      : smallint;
  index_of_CoScon            : smallint;
  value_of_CoScon            : smallint;
  index_of_Main_CoScon       : smallint;
  Scan_Pixel_Count           : smallint;
  Pixel_Checked              : boolean;

```

```

begin
  index_of_Array_Number := 1;
  repeat
    if ( Main_Buffers [ index_of_Array_Number ].Enable = true ) and
      ( Main_Buffers [ index_of_Array_Number ].Scons.
        Co_Scon_Count > 1 ) then
      begin
        index_of_CoScon := 1;
        repeat
          value_of_CoScon := Main_Buffers [ index_of_Array_Number ].
            Scons.Co_Scon [ index_of_CoScon ];
          Pixel_Checked := false;
          if
            ( Main_Buffers [ value_of_CoScon ].Size = 3 )
            and
            (
              ( Main_Buffers [ value_of_CoScon ].Dir = HORIZONTAL ) or
              ( Main_Buffers [ value_of_CoScon ].Dir = VERTICAL )
            ) then
            begin
              Pixel_Checked := true;
              Scan_Pixel_Count := 1;
              repeat
                if ( not PreScon [ Main_Buffers [ Value_of_CoScon ].
                  Members [ Scan_Pixel_Count ] ].
                  Enable [ index_of_Array_Number ] ) then
                begin
                  Pixel_Checked := false;
                  index_of_Main_CoScon := 1;
                  repeat
                    if ( PreScon [ Main_Buffers [ Value_of_CoScon ].
                      Members [ Scan_Pixel_Count ] ].
                      Enable [ Main_Buffers[index_of_Array_Number].
                        Scons.Co_Scon [ index_of_Main_CoScon ] ] )
                    AND
                    ( Value_of_CoScon <>
                      Main_Buffers [ index_of_Array_Number ].
                        Scons.Co_Scon [ index_of_Main_CoScon ] )
                    then
                      Pixel_Checked := true;
                      inc ( index_of_Main_CoScon );
                    until ( Pixel_Checked ) or
                      ( index_of_Main_CoScon >
                        Main_Buffers [ index_of_Array_Number ].
                          Scons.Co_Scon_Count );
                  end;
                  inc ( Scan_Pixel_Count );
                until ( not Pixel_Checked ) or
                  ( Scan_Pixel_Count >
                    Main_Buffers [ Value_of_CoScon ].Size );
              end;
              if Pixel_Checked = true then
                begin
                  Main_Buffers [ value_of_CoScon ].Enable := false;
                  { Start from head of point of current Array }
                  index_of_CoScon := 0;
                end;
              end;
            end;
          end;
        end;
      end;
    end;
  end;
end;

```

```

        Update_Prescons;
        Catching_Scons;
    end;
    inc ( index_of_CoScon );
until index_of_CoScon >
    Main_Buffers [ index_of_Array_Number ].Scons.
        Co_Scon_Count;

end;
inc ( index_of_Array_Number );
until ( index_of_Array_Number = Array_Number );
end;

(*****
(* CLEANUP UNNECESSARY TWO-TUPLE ARRAYS WHICH CONFIRM BEFORE *)
(*****
(* Programmer   : Derya Yilmaz *)
(* Coding-Date  : 05/29/1999 *)
(* Last Updated : 06/03/1999 *)
(*****
procedure Cleanup_Unneccassary_TwoTuple_Arrays_Which_Confirm_Before;
var
    Index_Array_Number      : smallint;
    This_Array              : smallint;
    First_Mem_This_Array    : smallint;
    First_Mem_Index         : smallint;
    Second_Mem_This_Array   : smallint;
    Second_Mem_Index        : smallint;
    Array_To_Clean          : boolean;
    F_Prescon               : smallint;
    S_Prescon               : smallint;

begin
    index_Array_Number := 1;
    Array_to_Clean := false;
    repeat
        if ( Main_Buffers [ index_Array_Number ].Enable = true ) AND
            ( Main_Buffers [ index_Array_Number ].Size = 2 ) then
            begin
                This_Array := index_Array_Number;
                Array_to_Clean := false;
                First_Mem_This_Array := Main_Buffers[This_Array].Members[1];
                Second_Mem_This_Array := Main_Buffers[This_Array].Members[2];
                for First_Mem_index := 1 to
                    Prescon [ First_Mem_This_Array ].Count do
                    for Second_Mem_index := 1 to
                        Prescon [ Second_Mem_This_Array ].Count do
                        begin
                            F_Prescon := Prescon [ Second_Mem_This_Array ].
                                Array_Numbers [ Second_Mem_index ];
                            S_Prescon := Prescon [ First_Mem_This_Array ].
                                Array_Numbers [ First_Mem_index ];
                            if (
                                ( Main_Buffers [F_Prescon].Scons.Scon [S_Prescon]=true )
                                or
                                ( Main_Buffers [S_Prescon].Scons.Scon [F_Prescon]=true )
                            )

```

```

        and
        ( S_Prescon <> This_Array ) and
        ( F_Prescon <> This_Array ) then
    begin
        Array_to_Clean := true;
        Break;
    end;
end;
end;
if Array_to_Clean = true then
begin
    Main_Buffers [ This_Array ].Enable := false;
    Update_Prescons;
    Catching_Scons;
end;
end;
inc ( index_Array_Number );
Until index_Array_Number = Array_Number;
end;

(*****
(*      -->   JOIN THE SAME DIRECTION ARRAYS      <--      *)
(*****
(* Programmer   : Derya Yilmaz                      *)
(* Coding-Date   : 05/29/1999                      *)
(* Last Updated  : 06/03/1999                      *)
(*****

procedure Join_The_Same_Dir_Arrays;
var
    Index_Array_Number      : smallint;
    This_Array              : smallint;
    First_Mem_This_Array    : smallint;
    Second_Mem_This_Array   : smallint;
    First_Mem_Relation_Array : smallint;
    Second_Mem_Relation_Array : smallint;
    Co_Array_Number         : smallint;
    index                  : smallint;
    Dummy_Array             : smallint;
begin
    Co_Array_Number := Array_Number;
    index_Array_Number := 1;
    repeat
        if ( Main_Buffers [ index_Array_Number ]. Enable = true ) AND
            ( Main_Buffers [ index_Array_Number ]. Size = 2 ) then
            begin
                This_Array := index_Array_Number;
                First_Mem_This_Array := Main_Buffers [ This_Array ].
                    Members [ 1 ];
                Second_Mem_This_Array := Main_Buffers [ This_Array ].
                    Members [ 2 ];
                if ( PreScon [ First_Mem_This_Array ].Count = 2 ) and
                    ( PreScon [ Second_Mem_This_Array ].Count = 2 ) then
                    begin
                        if PreScon [ First_Mem_This_Array ].
                            Array_Numbers [ 1 ] = This_Array
                        then

```

```

First_Mem_Relation_Array :=
    PreScon [ First_Mem_This_Array ].
    Array_Numbers [ 2 ]
else
First_Mem_Relation_Array :=
    PreScon [ First_Mem_This_Array ].
    Array_Numbers [ 1 ];
if PreScon [ Second_Mem_This_Array ].
    Array_Numbers [ 1 ] = This_Array
then
Second_Mem_Relation_Array :=
    PreScon [ Second_Mem_This_Array ].
    Array_Numbers [ 2 ]
else
Second_Mem_Relation_Array :=
    PreScon [ Second_Mem_This_Array ].
    Array_Numbers [ 1 ];
if ( Main_Buffers [ First_Mem_Relation_Array ].Dir =
    Main_Buffers [ Second_Mem_Relation_Array ].Dir ) and
( Main_Buffers [ First_Mem_Relation_Array ].
    Cleaned = false ) and
( Main_Buffers [ Second_Mem_Relation_Array ].
    Cleaned = false ) then
begin
    if Main_Buffers [First_Mem_Relation_Array].Members [1] =
        First_Mem_This_Array then
    begin
        Dummy_Array := First_Mem_Relation_Array;
        First_Mem_Relation_Array := Second_Mem_Relation_Array;
        Second_Mem_Relation_Array := Dummy_Array;
    end;
    index := 1;
    repeat
        Main_Buffers [ Co_Array_Number ].Members [ index ] :=
            Main_Buffers [ First_Mem_Relation_Array ].
                Members [ index ];
        inc ( index );
    until index > Main_Buffers [ First_Mem_Relation_Array ].
        Size;
    Main_Buffers [ Co_Array_Number ].Size := index - 1;
    index := 1;
    repeat
        inc ( Main_Buffers [ Co_Array_Number ].Size );
        Main_Buffers [ Co_Array_Number ].
            Members [ Main_Buffers [ Co_Array_Number ].Size ] :=
                Main_Buffers [ Second_Mem_Relation_Array ].
                    Members [ index ];
        inc ( index );
    until index > Main_Buffers [ Second_Mem_Relation_Array ].
        Size;
    Main_Buffers [ First_Mem_Relation_Array ].Cleaned := true;
    Main_Buffers [ Second_Mem_Relation_Array ].
        Cleaned := true;
    Main_Buffers [ This_Array ].Cleaned := true;
    Main_Buffers [ Co_Array_Number ].Enable := true;
    Main_Buffers [ Co_Array_Number ].Dir :=

```

```

        Main_Buffers [ First_Mem_Relation_Array ].Dir;
        inc ( Co_Array_Number );
    end;
    end;
    end;
    inc ( index_Array_Number );
until index_Array_Number = Array_Number;
Array_Number := Co_Array_Number;
Clear_Unnecassary_Arrays;
end;

(*****
(*      -->  CLEANUP THREE-TUPLE ARRAYS  <--      *)
(*****
(* Programmer   : Derya Yilmaz                      *)
(* Coding-Date  : 05/29/1999                        *)
(* Last Updated : 06/02/1999                        *)
(*****

procedure Cleanup_ThreeTuple_Arrays;
var
    index_Array_Number : smallint;
    This_Array         : smallint;
    First_Mem          : smallint;
    Center_Mem         : smallint;
    Second_Mem         : smallint;
    Okay               : boolean;
    Okay_TA            : boolean;
    Okay_Size          : smallint;
    index              : smallint;

begin
    index_Array_Number := 1;
    repeat
        if ( Main_Buffers [ index_Array_Number ].Enable = true ) and
            ( Main_Buffers [ index_Array_Number ].Size = 3 ) and
            ( not Main_Buffers [ index_Array_Number ].Not_Clean ) then
            begin
                okay := false;
                index := 1;
                This_Array := index_Array_Number;
                First_Mem := Main_Buffers [ This_Array ].Members [ 1 ];
                Center_Mem := Main_Buffers [ This_Array ].Members [ 2 ];
                Second_Mem := Main_Buffers [ This_Array ].Members [ 3 ];
                if ( PreScon [ First_Mem ].Count = 3 ) and
                    ( PreScon [ Second_Mem ].Count = 3 ) then
                    begin
                        Okay_TA := false;
                        Okay_Size := 0;
                        for index := 1 to 3 do
                            begin
                                if PreScon [ First_Mem ].Array_Numbers [ index ] =
This_Array
                                    then
                                        Okay_TA := true
                                    else

```

```

    if Main_Buffers [ PreScon [ First_Mem ].
        Array_Numbers [ index ] ].Size = 2 then
        inc ( Okay_Size )
    else
        Break;
    end;
    if ( Okay_TA ) and ( Okay_Size = 2 ) then
    begin
        Okay_TA := false;
        Okay_Size := 0;
        for index := 1 to 3 do
        begin
            if PreScon [ Second_Mem ].Array_Numbers [ index ] =
                This_Array then
                Okay_TA := true
            else
                if Main_Buffers [ PreScon [ Second_Mem ].
                    Array_Numbers [ index ] ].Size = 2 then
                    inc ( Okay_Size )
                else
                    Break;
                end;
            if ( Okay_TA = true ) and
                ( Okay_Size = 2 ) then Okay := true;
        end;
    end;
    if (
        ( PreScon [ First_Mem ].Count = 1 ) and
        ( PreScon [ Second_Mem ].Count = 1 ) and
        ( PreScon [ Center_Mem ].Count > 1 )
    )
    or
        ( okay = true ) then
        Main_Buffers [ This_Array ].Enable := false
    else
        if ( ( PreScon [ First_Mem ].Count = 1 ) or
            ( PreScon [ Second_Mem ].Count = 1 ) ) and
            ( PreScon [ Center_Mem ].Count > 1 ) then
        begin
            if PreScon [ Second_Mem ].Count = 1 then
                dec( Main_Buffers [ This_Array ].Size )
            else
                begin
                    Main_Buffers [ This_Array ].Members [ 1 ] :=
                        Main_Buffers [ This_Array ].Members [ 2 ];
                    Main_Buffers [ This_Array ].Members [ 2 ] :=
                        Main_Buffers [ This_Array ].Members [ 3 ];
                    dec( Main_Buffers [ This_Array ].Size );
                end;
            end;
        end;
    end;
    inc (index_Array_Number );
    until index_Array_Number = Array_Number;
    Update_PreScons;
    Catching_Scons;
end;

```

```

(*****)
(*      -->  BREAKING THE UNBREAKING ARRAYS      <--      *)
(*****)
(* Programmer   : Derya Yilmaz                      *)
(* Coding-Date  : 05/29/1999                        *)
(* Last Updated : 06/05/1999                        *)
(*****)
procedure Breaking_The_UnBreaking_Arrays;
var
  index_Array_Number  : smallint;
  this_Array          : smallint;
  Co_Array_Number     : smallint;
  pos_index           : smallint;
  First_index         : smallint;
  index              : smallint;

begin
  index_Array_Number := 1;
  Co_Array_Number := Array_Number;
  repeat
    if ( Main_Buffers [ index_Array_Number ].Enable ) and
       ( Main_Buffers [ index_Array_Number ].Size >= 5 ) then
      begin
        This_Array := index_Array_Number;
        First_index := 1;
        Pos_index := 3;
        repeat
          if (
            ( PreScon [ Main_Buffers [ This_Array ].
              Members [ Pos_index ] ].Count > 2 ) and
            ( Pos_index - First_index + 1 >= 3 )
          )
          or
          (
            ( PreScon [ Main_Buffers [ This_Array ].
              Members [ Pos_index ] ].Count = 2 ) and
            ( Pos_index - First_index + 1 >= 3 ) and
            ( Main_Buffers [ index_Array_Number ].
              Size - Pos_index > 1 ) and
            ( Main_Buffers [ PreScon [ Main_Buffers [ This_Array ].
              Members [ Pos_index ] ].Array_Numbers [ 1 ] ].
              Size > 2 ) and
            ( Main_Buffers [ PreScon [ Main_Buffers [ This_Array ].
              Members [ Pos_index ] ].Array_Numbers [ 2 ] ].
              Size > 2 )
          )
          or
          (
            ( Main_Buffers [ This_Array ].Size = Pos_index ) and
            ( Pos_index - First_index + 1 >= 3 )
          )
        ) then
          begin
            for index := 1 to Pos_index - First_index + 1 do
              Main_Buffers [ Co_Array_Number ].Members [ index ] :=
                Main_Buffers [ This_Array ].
                  Members [ First_index + index - 1 ];
          end
        end
      end
    else
      index_Array_Number := index_Array_Number + 1;
    end
  until index_Array_Number >= Array_Number;
end

```

```

        Main_Buffers [ Co_Array_Number ].Size :=
            Pos_index - First_index + 1;
        Main_Buffers [ Co_Array_Number ].Dir :=
            Main_Buffers [ This_Array ].Dir;
        Main_Buffers [ Co_Array_Number ].Enable := true;
        Main_Buffers [ This_Array ].Enable := false;
        inc ( Co_Array_Number );
        First_index := Pos_index;
    end;
    inc ( pos_index );
    until Pos_index > Main_Buffers [ This_Array ].Size;
end;
inc ( index_Array_Number );
until index_Array_Number = Array_Number;
Array_Number := Co_Array_Number;
Update_PreScons;
Catching_Scons;
end;

(*****
*)          -->  CLEANUP TWO-TUPLE ARRAYS  <--          *)
(*****
*) Programmer   : Derya Yilmaz                               *)
*) Coding-Date  : 05/29/1999                                   *)
*) Last Updated : 06/03/1999                                   *)
(*****

procedure Cleanup_TwoTuple_Arrays;
var
    index_Array_Number    : smallint;
    This_Array            : smallint;
    First_Mem_This_Array  : smallint;
    Second_Mem_This_Array : smallint;
    Less_Value_F          : smallint;
    Less_Value_S          : smallint;
    index_PreScon         : smallint;
    Value_to_Transfer     : smallint;
    Not_Value_to_Transfer : smallint;
    Current_Array         : smallint;
    index_carry           : smallint;

begin
    index_Array_Number := 1;
    repeat
        if ( Main_Buffers [ index_Array_Number ].Enable = true ) and
            ( Main_Buffers [ index_Array_Number ].Size = 2 ) then
            begin
                This_Array := index_Array_Number;
                First_Mem_This_Array := Main_Buffers [ This_Array ].
                    Members [ 1 ];
                Second_Mem_This_Array := Main_Buffers [ This_Array ].
                    Members [ 2 ];
                if ( PreScon [ First_Mem_This_Array ].Count = 1 ) or
                    ( PreScon [ Second_Mem_This_Array ].Count = 1 ) then
                    Main_Buffers [ This_Array ].Enable := false
                else

```

```

begin
  index_PreScon := 1;
  if PreScon [ First_Mem_This_Array ].Count <>
    PreScon [ Second_Mem_This_Array ].Count then
    begin
      Less_Value_F := 100;
      repeat
        if ( PreScon [ First_Mem_This_Array ].
          Array_Numbers [ index_PreScon ] <> This_Array )
          and
          ( Main_Buffers [ PreScon [ First_Mem_This_Array ].
            Array_Numbers [ index_PreScon ] ].
            Size < Less_Value_F )
          then
            Less_Value_F := Main_Buffers [ PreScon
              First_Mem_This_Array ].
              Array_Numbers [ index_PreScon ] ].Size;
            inc ( index_PreScon );
          until index_PreScon > PreScon [ First_Mem_This_Array ].Count;
          index_PreScon := 1;
          Less_Value_S := 100;
          repeat
            if ( PreScon [ Second_Mem_This_Array ].
              Array_Numbers [ index_PreScon ] <> This_Array )
              and
              ( Main_Buffers [ PreScon [ Second_Mem_This_Array ].
                Array_Numbers [ index_PreScon ] ].Size <
                Less_Value_S )
              then
                Less_Value_S := Main_Buffers [ PreScon
                  [ Second_Mem_This_Array ].
                  Array_Numbers [ index_PreScon ] ].Size;
                inc ( index_PreScon );
              until index_PreScon >
                PreScon [ Second_Mem_This_Array ].Count;
            end else
              if PreScon [ First_Mem_This_Array ].Count >
                PreScon [ Second_Mem_This_Array ].Count then
                begin
                  Less_Value_F := 1;
                  Less_Value_S := 0;
                end else
                  begin
                    Less_Value_F := 0;
                    Less_Value_S := 1;
                  end;
                if Less_Value_F > Less_Value_S then
                  begin
                    Value_to_Transfer := First_Mem_This_Array;
                    Not_Value_to_Transfer := Second_Mem_This_Array;
                  end else
                    begin
                      Value_to_Transfer := Second_Mem_This_Array;
                      Not_Value_to_Transfer := First_Mem_This_Array;
                    end;
                index_PreScon := 1;

```

```

repeat
  Current_Array := PreScon [ Not_Value_to_Transfer ].
  Array_Numbers [ index_PreScon ];

  if ( Current_Array <> This_Array ) then
  begin
    if ( Not_Value_to_Transfer = Main_Buffers
      [ Current_Array ].Members [ 1 ] ) or
      ( Not_Value_to_Transfer = Main_Buffers
      [ Current_Array ].Members [ 2 ] ) then
    begin
      if ( Main_Buffers [ Current_Array ].
        TTA_First = false ) and
        ( Main_Buffers [ Current_Array ].
        Broken_First = false ) then
      begin
        for index_carry := Main_Buffers [ Current_Array ].
          Size downto 1 do
        begin
          Main_Buffers [ Current_Array ].
            Members [ index_carry + 1 ] :=
              Main_Buffers [ Current_Array ].
                Members [ index_carry ];
        end;
        Main_Buffers [ Current_Array ].Members [ 1 ] :=
          Value_to_Transfer;
        inc ( Main_Buffers [ Current_Array ].Size );
        Main_Buffers [ Current_Array ].TTA_First := true;
        Main_Buffers [ This_Array ].Enable := false;
      end;
    end else
    if ( Not_Value_to_Transfer=Main_Buffers[Current_Array .
      Members[Main_Buffers[Current_Array].Size ] ) or
      ( Not_Value_to_Transfer=Main_Buffers[Current_Array].
      Members[Main_Buffers[Current_Array].Size - 1]) then
    begin
      if ( Main_Buffers [ Current_Array ].
        TTA_Second = false ) and
        ( Main_Buffers [ Current_Array ].
        Broken_Last = false ) then
      begin
        Main_Buffers [ Current_Array ].
          Members [ Main_Buffers [ Current_Array ].
            Size + 1 ] := Value_to_Transfer;
        inc ( Main_Buffers [ Current_Array ].Size );
        Main_Buffers [ Current_Array ].TTA_Second := true;
        Main_Buffers [ This_Array ].Enable := false;
      end;
    end;
    end;
    inc ( index_PreScon );
  until index_PreScon > PreScon [Not_Value_to_Transfer].Count;
end;
Update_Prescons;
Catching_Scons;
end;

```

```

    inc ( index_Array_Number );
until index_Array_Number = Array_Number;
end;

```

```

(*****
*)          -->  TRANSFERING ARRAYS  <--          *)
(*****
*) Programmer   : Derya Yilmaz                      *)
*) Coding-Date  : 06/25/1999                        *)
*) Last Updated : 06/25/1999                        *)
*)*****

```

```

procedure Transferring_Arrays;

```

```

var

```

```

    index_of_Array : smallint;
    index_of_TArray : smallint;

```

```

begin

```

```

    index_of_Array := 1;

```

```

    index_of_TArray := 1;

```

```

    repeat

```

```

        if Main_Buffers [ index_of_Array ].Enable then

```

```

            begin

```

```

                Typed_Array [ index_of_TArray ].Array_No := index_of_Array;

```

```

                Typed_Array [ index_of_TArray ].Array_Type := Main_Buffers

```

```

                    [ index_of_Array ].Dir;

```

```

                Typed_Array [ index_of_TArray ].Enable := true;

```

```

                inc ( index_of_TArray );

```

```

            end;

```

```

            inc ( index_of_Array );

```

```

        until index_of_Array = Array_Number;

```

```

        No_TArray := index_of_TArray;

```

```

    end;

```

```

(*****
*)          -->  SORTING_X_VIA_Y <--          *)
(*****
*) Programmer   : Derya Yilmaz                      *)
*) Coding-Date  : 06/25/1999                        *)
*) Last Updated : 06/25/1999                        *)
*)*****

```

```

procedure Sorting_X_Via_Y (Width, Height : smallint);

```

```

var

```

```

    x,y          : smallint;

```

```

    index_of_TArray : smallint;

```

```

    index_of_SA     : smallint;

```

```

    index_of_ORA    : smallint;

```

```

    Array_No        : smallint;

```

```

    Pixel           : smallint;

```

```

    Status_SA       : boolean;

```

```

    Status_ORA      : boolean;

```

```

begin

```

```

    Fillchar ( Sorted_Arrays , Sizeof ( Sorted_Arrays ), 0 );

```

```

    index_of_SA := 1;

```

```

    Status_SA := false;

```

```

Status_ORA := false;
for y := 1 to Height do
begin
  index_of_ORA := 1;
  for x := 1 to Width do
  begin
    index_of_TArray := 1;
    repeat
      Array_No := Typed_Array [ index_of_TArray ].Array_No;
      Pixel := Main_Buffers [ Array_No ].Members [ 1 ];
      if ( Coordinated_Pixels [ Pixel ].y = y ) and
        ( Coordinated_Pixels [ Pixel ].x = x ) then
      begin
        case Main_Buffers [ Array_No ].Dir of
          VERTICAL :begin
            Sorted_Arrays [ index_of_SA ].
              Ordered_Arrays[ index_of_ORA ].
                FArrays [ 2 ] := Array_No;
            Status_ORA := true;
            Status_SA := true;
          end;
          HORIZONTAL :begin
            Sorted_Arrays [ index_of_SA ].
              Ordered_Arrays[ index_of_ORA ].
                FArrays [ 4 ] := Array_No;
            Status_ORA := true;
            Status_SA := true;
          end;
          SLASH :begin
            Sorted_Arrays [ index_of_SA ].
              Ordered_Arrays[ index_of_ORA ].
                FArrays [ 1 ] := Array_No;
            Status_ORA := true;
            Status_SA := true;
          end;
          BACKSLASH :begin
            Sorted_Arrays [ index_of_SA ].
              Ordered_Arrays[ index_of_ORA ].
                FArrays [ 3 ] := Array_No;
            Status_ORA := true;
            Status_SA := true;
          end;
        end;
      end;
      inc ( index_of_TArray );
    until index_of_TArray = No_TArray;
    if Status_ORA then
    begin
      inc ( index_of_ORA );
      Status_ORA := false;
    end;
  end;
  if Status_SA then
  begin
    inc ( index_of_SA );
    Status_SA := false;
  end;
end;

```

```

    end;
  end;
end;

```

```

(*****
(*          -->  SORTED TYPED ARRAYS  <--          *)
(*****
(* Programmer   : Derya Yilmaz                      *)
(* Coding-Date  : 06/26/1999                        *)
(* Last Updated : 06/26/1999                        *)
(*****

```

```

procedure Sorted_Typed_Arrays;

```

```

var
  index_of_SOTA      : smallint;
  index_of_SA        : smallint;
  index_of_ORA       : smallint;
  index_of_FA        : smallint;
  Value              : smallint;

```

```

begin

```

```

  index_of_SOTA := 1;
  for index_of_SA := 1 to 50 do
    for index_of_ORA := 1 to 10 do
      for index_of_FA := 1 to 4 do
        begin
          Value := Sorted_Arrays [ index_of_SA ].Ordered_Arrays
                     [ index_of_ORA ].FArrays [ index_of_FA ];
          if Value <> 0 then
            begin
              Sort_Typed_Arrays [ index_of_SOTA ] := Value;
              inc ( index_of_SOTA );
            end;
          end;
        end;
      end;
    end;
  end;
end;

```

```

end;

```

```

(*****
(*          -->  DEFINES ARRAYS LINK  <--          *)
(*****
(* Programmer   : Derya Yilmaz                      *)
(* Coding-Date  : 06/25/1999                        *)
(* Last Updated : 06/25/1999                        *)
(*****

```

```

procedure Defines_Arrays_Link;

```

```

var

```

```

  index_of_TArray    : smallint;
  index_of_CoScon     : smallint;
  Array_No            : smallint;
  Slave_No            : smallint;
  Master_FP           : smallint;
  Master_FP_1         : smallint;
  Master_LP           : smallint;
  Master_LP_1         : smallint;
  Slave_FP            : smallint;
  Slave_FP_1          : smallint;
  Slave_LP            : smallint;
  Slave_LP_1         : smallint;

```

```

PreCount          : smallint;
Slave_FP_New      : smallint;
Slave_FP_1_New    : smallint;

begin
Fillchar ( Array_Links , Sizeof ( Array_Links ), #0 );
for index_of_TArray := 1 to NO_TArray - 1 do
begin
Array_No := Sort_Typed_Arrays [ index_of_TArray ];
Master_FP := Main_Buffers [ Array_No ].Members [ 1 ];
Master_FP_1 := Main_Buffers [ Array_No ].Members [ 2 ];
Master_LP := Main_Buffers [ Array_No ].
    Members [ Main_Buffers [ Array_No ].Size ];
Master_LP_1 := Main_Buffers [ Array_No ].
    Members [ Main_Buffers [ Array_No ].Size - 1 ];
if Main_Buffers [ Array_No ].Scons.Co_Scon_Count <> 0 then
begin
index_of_CoScon := 1;
repeat
Slave_No := Main_Buffers [ Array_No ].Scons.
    Co_Scon [ index_of_CoScon ];
Slave_FP := Main_Buffers [ Slave_No ].Members [ 1 ];
Slave_FP_1 := Main_Buffers [ Slave_No ].Members [ 2 ];
Slave_LP := Main_Buffers [ Slave_No ].
    Members [ Main_Buffers [ Slave_No ].Size ];
Slave_LP_1 := Main_Buffers [ Slave_No ].
    Members [ Main_Buffers [ Slave_No ].
        Size - 1 ];
if ( Master_FP = Slave_FP ) or ( Master_FP = Slave_FP_1 ) or
    ( Master_FP_1 = Slave_FP ) or ( Master_FP_1 = Slave_FP_1 )
then
begin
if Array_Links [Array_No].Point [Slave_No] = false then
begin
inc ( Array_Links [ Array_No ].FFArray.Count );
Array_Links [ Array_No ].FFArray.
    Link [ Array_Links [ Array_No ].
        FFArray.Count ] := Slave_No;
Array_Links [ Array_No ].FFArray.
    Pointer [ Array_Links [ Array_No ].
        FFArray.Count ] := true;
Array_Links [ Array_No ].Point [ Slave_No ] := true;
PreCount := 1;
repeat
if ( Array_Links [ Array_No ].Point [ PreScon
    [ Slave_FP_1 ].Array_Numbers [PreCount] ]=false )
and
    ( Array_No <> PreScon [ Slave_FP_1 ].
        Array_Numbers [ PreCount ] ) then
begin
Slave_No := PreScon [ Slave_FP_1 ].
    Array_Numbers [ PreCount ];
Slave_FP_New := Main_Buffers [ Slave_No ].
    Members [ 1 ];
Slave_FP_1_New := Main_Buffers [ Slave_No ].
    Members [ 2 ];

```

```

if ( Slave_FP_1 = Slave_FP_New ) or
  ( Slave_FP_1 = Slave_FP_1_New ) then
begin
  inc ( Array_Links [ Array_No ].FFArray.Count );
  Slave_No := PreScon [ Slave_FP_1 ].
    Array_Numbers [ PreCount ];
  Array_Links [ Array_No ].FFArray.
    Link [ Array_Links [ Array_No ].
      FFArray.Count ] := Slave_No;
  Array_Links [ Array_No ].FFArray.
    Pointer [ Array_Links [ Array_No ].
      FFArray.Count ] := true;
  Array_Links [ Array_No ].
    Point [ Slave_No ] := true;
end else
begin
  inc ( Array_Links [ Array_No ].FLArray.Count );
  Slave_No := PreScon [ Slave_FP_1 ].
    Array_Numbers [ PreCount ];
  Array_Links [ Array_No ].FLArray.
    Link [ Array_Links [ Array_No ].
      FLArray.Count ] := Slave_No;
  Array_Links [ Array_No ].FLArray.
    Pointer [ Array_Links [ Array_No ].
      FLArray.Count ] := true;
  Array_Links [ Array_No ].
    Point [ Slave_No ] := true;
end;
end;
inc ( PreCount );
until PreCount > PreScon [ Slave_FP_1 ].Count;
PreCount := 1;
repeat
  if ( Array_Links [ Array_No ].
    Point [ PreScon [ Slave_FP ].
      Array_Numbers [ PreCount ] ] = false ) and
    ( Array_No <> PreScon [ Slave_FP ].
      Array_Numbers [ PreCount ] ) then
begin
  Slave_No := PreScon [ Slave_FP ].
    Array_Numbers [ PreCount ];
  Slave_FP_New := Main_Buffers [ Slave_No ].
    Members [ 1 ];
  Slave_FP_1_New := Main_Buffers [ Slave_No ].
    Members [ 2 ];
  if ( Slave_FP = Slave_FP_New ) or
    ( Slave_FP = Slave_FP_1_New ) then
begin
  inc ( Array_Links [ Array_No ].FFArray.Count );
  Array_Links [ Array_No ].FFArray.
    Link [ Array_Links [ Array_No ].
      FFArray.Count ] := Slave_No;
  Array_Links [ Array_No ].FFArray.
    Pointer [ Array_Links [ Array_No ].
      FFArray.Count ] := true;
  Array_Links [ Array_No ].

```

```

        Point [ Slave_No ] := true;
    end else
    begin
        inc ( Array_Links [ Array_No ].FLArray.Count );
        Array_Links [ Array_No ].FLArray.
            Link [ Array_Links [ Array_No ].
                FLArray.Count ] := Slave_No;
        Array_Links [ Array_No ].FLArray.
            Pointer [ Array_Links [ Array_No ].
                FLArray.Count ] := true;
        Array_Links [ Array_No ].
            Point [ Slave_No ] := true;
    end;
end;
inc ( PreCount );
until PreCount > PreScon [ Slave_FP ].Count;
end else
begin
    PreCount := 1;
    repeat
        if ( Array_Links [ Array_No ].
            Point [ PreScon [ Slave_FP_1 ].
                Array_Numbers [ PreCount ] ] = false ) and
            ( Array_No <> PreScon [ Slave_FP_1 ].
                Array_Numbers [ PreCount ] ) then
        begin
            Slave_No := PreScon [ Slave_FP_1 ].
                Array_Numbers [ PreCount ];
            Slave_FP_New := Main_Buffers [ Slave_No ].
                Members [ 1 ];
            Slave_FP_1_New := Main_Buffers [ Slave_No ].
                Members [ 2 ];
            if ( Slave_FP_1 = Slave_FP_New ) or
                ( Slave_FP_1 = Slave_FP_1_New ) then
            begin
                inc ( Array_Links [ Array_No ].FFArray.Count );
                Slave_No := PreScon [ Slave_FP_1 ].
                    Array_Numbers [ PreCount ];
                Array_Links [ Array_No ].
                    FFArray.Link [ Array_Links [ Array_No ].
                        FFArray.Count ] := Slave_No;
                Array_Links [ Array_No ].FFArray.
                    Pointer [ Array_Links [ Array_No ].
                        FFArray.Count ] := true;
                Array_Links [ Array_No ].
                    Point [ Slave_No ] := true;
            end else
            begin
                inc ( Array_Links [ Array_No ].FLArray.Count );
                Slave_No := PreScon [ Slave_FP_1 ].
                    Array_Numbers [ PreCount ];
                Array_Links [ Array_No ].
                    FLArray.Link [ Array_Links [ Array_No ].
                        FLArray.Count ] := Slave_No;
                Array_Links [ Array_No ].FLArray.
                    Pointer [ Array_Links [ Array_No ].
                        FLArray.Count ] := true;
            end
        end
    until PreCount > PreScon [ Slave_FP ].Count;
end

```

```

        FLArray.Count ] := true;
        Array_Links [ Array_No ].
            Point [ Slave_No ] := true;
    end;
end;
inc ( PreCount );
until PreCount > PreScon [ Slave_FP_1 ].Count;
PreCount := 1;
repeat
    if ( Array_Links [ Array_No ].
        Point [ PreScon [ Slave_FP ].
            Array_Numbers [ PreCount ] ] = false ) and
        ( Array_No <> PreScon [ Slave_FP ].
            Array_Numbers [ PreCount ] ) then
    begin
        Slave_No := PreScon [ Slave_FP ].
            Array_Numbers [ PreCount ];
        Slave_FP_New := Main_Buffers [ Slave_No ].
            Members [ 1 ];
        Slave_FP_1_New := Main_Buffers [ Slave_No ].
            Members [ 2 ];
        if ( Slave_FP = Slave_FP_New ) or
            ( Slave_FP = Slave_FP_1_New ) then
        begin
            inc ( Array_Links [ Array_No ].FFArray.Count );
            Array_Links [ Array_No ].FFArray.
                Link [ Array_Links [ Array_No ].
                    FFArray.Count ] := Slave_No;
            Array_Links [ Array_No ].FFArray.
                Pointer [ Array_Links [ Array_No ].
                    FFArray.Count ] := true;
            Array_Links [ Array_No ].
                Point [ Slave_No ] := true;
        end else
        begin
            inc ( Array_Links [ Array_No ].FLArray.Count );
            Array_Links [ Array_No ].FLArray.
                Link [ Array_Links [ Array_No ].
                    FLArray.Count ] := Slave_No;
            Array_Links [ Array_No ].FLArray.
                Pointer [ Array_Links [ Array_No ].
                    FLArray.Count ] := true;
            Array_Links [ Array_No ].
                Point [ Slave_No ] := true;
        end;
    end;
    inc ( PreCount );
until PreCount > PreScon [ Slave_FP ].Count;
end;
end else
if ( Master_FP = Slave_LP ) or ( Master_FP = Slave_LP_1 ) or
    ( Master_FP_1 = Slave_LP ) or ( Master_FP_1 = Slave_LP_1 )
then
begin
    if Array_Links [Array_No].Point [Slave_No] = false then
    begin

```

```

inc ( Array_Links [ Array_No ].FLArray.Count );
Array_Links [ Array_No ].FLArray.
  Link [ Array_Links [ Array_No ].
    FLArray.Count ] := Slave_No;
Array_Links [ Array_No ].FLArray.
  Pointer [ Array_Links [ Array_No ].
    FLArray.Count ] := true;
Array_Links [ Array_No ].
  Point [ Slave_No ] := true;
PreCount := 1;
repeat
  if ( Array_Links [ Array_No ].
    Point [ PreScon [ Slave_LP_1 ].
      Array_Numbers [ PreCount ] ] = false ) and
    ( Array_No <> PreScon [ Slave_LP_1 ].
      Array_Numbers [ PreCount ] ) then
  begin
    Slave_No := PreScon [ Slave_LP_1 ].
      Array_Numbers [ PreCount ];
    Slave_FP_New := Main_Buffers [ Slave_No ].
      Members [ 1 ];
    Slave_FP_1_New := Main_Buffers [ Slave_No ].
      Members [ 2 ];
    if ( Slave_LP_1 = Slave_FP_New ) or
      ( Slave_LP_1 = Slave_FP_1_New ) then
    begin
      inc ( Array_Links [ Array_No ].FFArray.Count );
      Array_Links [ Array_No ].FFArray.
        Link [ Array_Links [ Array_No ].
          FFArray.Count ] := Slave_No;
      Array_Links [ Array_No ].FFArray.
        Pointer [ Array_Links [ Array_No ].
          FFArray.Count ] := true;
      Array_Links [ Array_No ].
        Point [ Slave_No ] := true;
    end else
    begin
      inc ( Array_Links [ Array_No ].FLArray.Count );
      Array_Links [ Array_No ].FLArray.
        Link [ Array_Links [ Array_No ].
          FLArray.Count ] := Slave_No;
      Array_Links [ Array_No ].FLArray.
        Pointer [ Array_Links [ Array_No ].
          FLArray.Count ] := true;
      Array_Links [ Array_No ].
        Point [ Slave_No ] := true;
    end;
  end;
  inc ( PreCount );
until PreCount > PreScon [ Slave_LP_1 ].Count;
PreCount := 1;
repeat
  if ( Array_Links [ Array_No ].
    Point [ PreScon [ Slave_LP ].
      Array_Numbers [ PreCount ] ] = false ) and
    ( Array_No <> PreScon [ Slave_LP ].

```

```

        Array_Numbers [ PreCount ] ) then
begin
    Slave_No := PreScon [ Slave_LP ].
        Array_Numbers [ PreCount ];
    Slave_FP_New := Main_Buffers [ Slave_No ].
        Members [ 1 ];
    Slave_FP_1_New := Main_Buffers [ Slave_No ].
        Members [ 2 ];
    if ( Slave_LP = Slave_FP_New ) or
        ( Slave_LP = Slave_FP_1_New ) then
begin
    inc ( Array_Links [ Array_No ].FFArray.Count );
    Array_Links [ Array_No ].FFArray.
        Link [ Array_Links [ Array_No ].
            FFArray.Count ] := Slave_No;
    Array_Links [ Array_No ].FFArray.
        Pointer [ Array_Links [ Array_No ].
            FFArray.Count ] := true;
    Array_Links [ Array_No ].
        Point [ Slave_No ]:= true;
    end else
begin
    inc ( Array_Links [ Array_No ].FLArray.Count );
    Array_Links [ Array_No ].FLArray.
        Link [ Array_Links [ Array_No ].
            FLArray.Count ] := Slave_No;
    Array_Links [ Array_No ].FLArray.
        Pointer [ Array_Links [ Array_No ].
            FLArray.Count ] := true;
    Array_Links [ Array_No ].
        Point [ Slave_No ]:= true;
    end;
end;
inc ( PreCount );
until PreCount > PreScon [ Slave_LP ].Count;
end else
begin
    PreCount := 1;
    repeat
        if ( Array_Links [ Array_No ].
            Point [ PreScon [ Slave_LP_1 ].
                Array_Numbers [ PreCount ] ] = false ) and
            ( Array_No <> PreScon [ Slave_LP_1 ].
                Array_Numbers [ PreCount ] ) then
begin
            Slave_No := PreScon [ Slave_LP_1 ].
                Array_Numbers [ PreCount ];
            Slave_FP_New := Main_Buffers [ Slave_No ].
                Members [ 1 ];
            Slave_FP_1_New := Main_Buffers [ Slave_No ].
                Members [ 2 ];
            if ( Slave_LP_1 = Slave_FP_New ) or
                ( Slave_LP_1 = Slave_FP_1_New ) then
begin
                inc ( Array_Links [ Array_No ].FFArray.Count );
                Array_Links [ Array_No ].FFArray.

```

```

        Link [ Array_Links [ Array_No ].
            FFArray.Count ] := Slave_No;
        Array_Links [ Array_No ].FFArray.
            Pointer [ Array_Links [ Array_No ].
                FFArray.Count ] := true;
        Array_Links [ Array_No ].
            Point [ Slave_No ] := true;
    end else
    begin
        inc ( Array_Links [ Array_No ].FLArray.Count );
        Array_Links [ Array_No ].FLArray.
            Link [ Array_Links [ Array_No ].
                FLArray.Count ] := Slave_No;
        Array_Links [ Array_No ].FLArray.
            Pointer [ Array_Links [ Array_No ].
                FLArray.Count ] := true;
        Array_Links [ Array_No ].
            Point [ Slave_No ] := true;
    end;
end;
inc ( PreCount );
until PreCount > PreScon [ Slave_LP_1 ].Count;
PreCount := 1;
repeat
    if ( Array_Links [ Array_No ].
        Point [ PreScon [ Slave_LP ].
            Array_Numbers [ PreCount ] ] = false ) and
        ( Array_No <> PreScon [ Slave_LP ].
            Array_Numbers [ PreCount ] ) then
    begin
        Slave_No := PreScon [ Slave_LP ].
            Array_Numbers [ PreCount ];
        Slave_FP_New := Main_Buffers [ Slave_No ].
            Members [ 1 ];
        Slave_FP_1_New := Main_Buffers [ Slave_No ].
            Members [ 2 ];
        if ( Slave_LP = Slave_FP_New ) or
            ( Slave_LP = Slave_FP_1_New ) then
        begin
            inc ( Array_Links [ Array_No ].FFArray.Count );
            Array_Links [ Array_No ].FFArray.
                Link [ Array_Links [ Array_No ].
                    FFArray.Count ] := Slave_No;
            Array_Links [ Array_No ].FFArray.
                Pointer [ Array_Links [ Array_No ].
                    FFArray.Count ] := true;
            Array_Links [ Array_No ].
                Point [ Slave_No ] := true;
        end else
        begin
            inc ( Array_Links [ Array_No ].FLArray.Count );
            Array_Links [ Array_No ].FLArray.
                Link [ Array_Links [ Array_No ].
                    FLArray.Count ] := Slave_No;
            Array_Links [ Array_No ].FLArray.
                Pointer [ Array_Links [ Array_No ].
                    FLArray.Count ] := true;
        end
    end
end repeat

```



```

        LLArray.Count ] := Slave_No;
        Array_Links [ Array_No ].LLArray.
        Pointer [ Array_Links [ Array_No ].
        LLArray.Count ] := true;
        Array_Links [ Array_No ].
        Point [ Slave_No ] := true;
    end;
end;
inc ( PreCount );
until PreCount > PreScon [ Slave_FP_1 ].Count;
PreCount := 1;
repeat
    if ( Array_Links [ Array_No ].
        Point [ PreScon [ Slave_FP ].
        Array_Numbers [ PreCount ] ] = false ) and
        ( Array_No <> PreScon [ Slave_FP ].
        Array_Numbers [ PreCount ] ) then
    begin
        Slave_No := PreScon [ Slave_FP ].
        Array_Numbers [ PreCount ];
        Slave_FP_New := Main_Buffers [ Slave_No ].
        Members [ 1 ];
        Slave_FP_1_New := Main_Buffers [ Slave_No ].
        Members [ 2 ];
        if ( Slave_FP = Slave_FP_New ) or
            ( Slave_FP_1 = Slave_FP_1_New ) then
        begin
            inc ( Array_Links [ Array_No ].LFArry.Count );
            Array_Links [ Array_No ].LFArry.
            Link [ Array_Links [ Array_No ].
            LFArry.Count ] := Slave_No;
            Array_Links [ Array_No ].LFArry.
            Pointer [ Array_Links [ Array_No ].
            LFArry.Count ] := true;
            Array_Links [ Array_No ].
            Point [ Slave_No ] := true;
        end else
        begin
            inc ( Array_Links [ Array_No ].LLArray.Count );
            Array_Links [ Array_No ].LLArray.
            Link [ Array_Links [ Array_No ].
            LLArray.Count ] := Slave_No;
            Array_Links [ Array_No ].LLArray.
            Pointer [ Array_Links [ Array_No ].
            LLArray.Count ] := true;
            Array_Links [ Array_No ].
            Point [ Slave_No ] := true;
        end;
    end;
    inc ( PreCount );
until PreCount > PreScon [ Slave_FP ].Count;
end else
begin
    PreCount := 1;
    repeat
        if ( Array_Links [ Array_No ].

```

```

        Point [ PreScon [ Slave_FP_1 ].
            Array_Numbers [ PreCount ] ] = false ) and
        ( Array_No <> PreScon [ Slave_FP_1 ].
            Array_Numbers [ PreCount ] ) then
begin
    Slave_No := PreScon [ Slave_FP_1 ].
        Array_Numbers [ PreCount ];
    Slave_FP_New := Main_Buffers [ Slave_No ].
        Members [ 1 ];
    Slave_FP_1_New := Main_Buffers [ Slave_No ].
        Members [ 2 ];
    if ( Slave_FP_1 = Slave_FP_New ) or
        ( Slave_FP_1 = Slave_FP_1_New ) then
    begin
        inc ( Array_Links [ Array_No ].LFArray.Count );
        Array_Links [ Array_No ].LFArray.
            Link [ Array_Links [ Array_No ].
                LFArray.Count ] := Slave_No;
        Array_Links [ Array_No ].LFArray.
            Pointer [ Array_Links [ Array_No ].
                LFArray.Count ] := true;
        Array_Links [ Array_No ].
            Point [ Slave_No ] := true;
    end else
    begin
        inc ( Array_Links [ Array_No ].LLArray.Count );
        Array_Links [ Array_No ].LLArray.
            Link [ Array_Links [ Array_No ].
                LLArray.Count ] := Slave_No;
        Array_Links [ Array_No ].LLArray.
            Pointer [ Array_Links [ Array_No ].
                LLArray.Count ] := true;
        Array_Links [ Array_No ].
            Point [ Slave_No ] := true;
    end;
end;
inc ( PreCount );
until PreCount > PreScon [ Slave_FP_1 ].Count;
PreCount := 1;
repeat
    if ( Array_Links [ Array_No ].
        Point [ PreScon [ Slave_FP ].
            Array_Numbers [ PreCount ] ] = false ) and
        ( Array_No <> PreScon [ Slave_FP ].
            Array_Numbers [ PreCount ] ) then
    begin
        Slave_No := PreScon [ Slave_FP ].
            Array_Numbers [ PreCount ];
        Slave_FP_New := Main_Buffers [ Slave_No ].
            Members [ 1 ];
        Slave_FP_1_New := Main_Buffers [ Slave_No ].
            Members [ 2 ];
        if ( Slave_FP = Slave_FP_New ) or
            ( Slave_FP_1 = Slave_FP_1_New ) then
        begin
            inc ( Array_Links [ Array_No ].LFArray.Count );

```

```

    Array_Links [ Array_No ].LFArrary.
    Link [ Array_Links [ Array_No ].
    LFArrary.Count ] := Slave_No;
    Array_Links [ Array_No ].LFArrary.
    Pointer [ Array_Links [ Array_No ].
    LFArrary.Count ] := true;
    Array_Links [ Array_No ].
    Point [ Slave_No ] := true;
end else
begin
    inc ( Array_Links [ Array_No ].LLArray.Count );
    Array_Links [ Array_No ].LLArray.
    Link [ Array_Links [ Array_No ].
    LLArray.Count ] := Slave_No;
    Array_Links [ Array_No ].LLArray.
    Pointer [ Array_Links [ Array_No ].
    LLArray.Count ] := true;
    Array_Links [ Array_No ].
    Point [ Slave_No ] := true;
end;
end;
inc ( PreCount );
until PreCount > PreScon [ Slave_FP ].Count;
end;
end else
begin
    if Array_Links [ Array_No ].
    Point [ Slave_No ] = false then
    begin
        inc ( Array_Links [ Array_No ].LLArray.Count );
        Array_Links [ Array_No ].LLArray.
        Link [ Array_Links [ Array_No ].
        LLArray.Count ] := Slave_No;
        Array_Links [ Array_No ].LLArray.
        Pointer [ Array_Links [ Array_No ].
        LLArray.Count ] := true;
        Array_Links [ Array_No ].
        Point [ Slave_No ] := true;
        PreCount := 1;
        repeat
            if ( Array_Links [ Array_No ].
            Point [ PreScon [ Slave_LP_1 ].
            Array_Numbers [ PreCount ] ] = false ) and
            ( Array_No <> PreScon [ Slave_LP_1 ].
            Array_Numbers [ PreCount ] ) then
            begin
                Slave_No := PreScon [ Slave_LP_1 ].
                Array_Numbers [ PreCount ];
                Slave_FP_New := Main_Buffers [ Slave_No ].
                Members [ 1 ];
                Slave_FP_1_New := Main_Buffers [ Slave_No ].
                Members [ 2 ];
                if ( Slave_LP_1 = Slave_FP_New ) or
                ( Slave_LP_1 = Slave_FP_1_New ) then
                begin
                    inc ( Array_Links [ Array_No ].LFArrary.Count );

```

```

    Array_Links [ Array_No ].LFArrary.
    Link [ Array_Links [ Array_No ].
    LFArrary.Count ] := Slave_No;
    Array_Links [ Array_No ].LFArrary.
    Pointer [ Array_Links [ Array_No ].
    LFArrary.Count ] := true;
    Array_Links [ Array_No ].
    Point [ Slave_No ] := true;
end else
begin
    inc ( Array_Links [ Array_No ].LLArray.Count );
    Array_Links [ Array_No ].LLArray.
    Link [ Array_Links [ Array_No ].
    LLArray.Count ] := Slave_No;
    Array_Links [ Array_No ].LLArray.
    Pointer [ Array_Links [ Array_No ].
    LLArray.Count ] := true;
    Array_Links [ Array_No ].
    Point [ Slave_No ] := true;
end;
end;
inc ( PreCount );
until PreCount > PreScon [ Slave_LP_1 ].Count;
PreCount := 1;
repeat
    if ( Array_Links [ Array_No ].
    Point [ PreScon [ Slave_LP ].
    Array_Numbers [ PreCount ] ] = false ) and
    ( Array_No <> PreScon [ Slave_LP ].
    Array_Numbers [ PreCount ] ) then
begin
    Slave_No := PreScon [ Slave_LP ].
    Array_Numbers [ PreCount ];
    Slave_FP_New := Main_Buffers [ Slave_No ].
    Members [ 1 ];
    Slave_FP_1_New := Main_Buffers [ Slave_No ].
    Members [ 2 ];
    if ( Slave_LP = Slave_FP_New ) or
    ( Slave_LP = Slave_FP_1_New ) then
begin
    inc ( Array_Links [ Array_No ].LFArrary.Count );
    Array_Links [ Array_No ].LFArrary.
    Link [ Array_Links [ Array_No ].
    LFArrary.Count ] := Slave_No;
    Array_Links [ Array_No ].LFArrary.
    Pointer [ Array_Links [ Array_No ].
    LFArrary.Count ] := true;
    Array_Links [ Array_No ].
    Point [ Slave_No ] := true;
end else
begin
    inc ( Array_Links [ Array_No ].LLArray.Count );
    Array_Links [ Array_No ].LLArray.
    Link [ Array_Links [ Array_No ].
    LLArray.Count ] := Slave_No;
    Array_Links [ Array_No ].LLArray.

```

```

        Pointer [ Array_Links [ Array_No ].
            LArray.Count ] := true;
        Array_Links [ Array_No ].
            Point [ Slave_No ] := true;
    end;
end;
inc ( PreCount );
until PreCount > PreScon [ Slave_LP ].Count;
end else
begin
    PreCount := 1;
    repeat
        if ( Array_Links [ Array_No ].
            Point [ PreScon [ Slave_LP_1 ].
                Array_Numbers [ PreCount ] ] = false ) and
            ( Array_No <> PreScon [ Slave_LP_1 ].
                Array_Numbers [ PreCount ] ) then
        begin
            Slave_No := PreScon [ Slave_LP_1 ].
                Array_Numbers [ PreCount ];
            Slave_FP_New := Main_Buffers [ Slave_No ].
                Members [ 1 ];
            Slave_FP_1_New := Main_Buffers [ Slave_No ].
                Members [ 2 ];
            if ( Slave_LP_1 = Slave_FP_New ) or
                ( Slave_LP_1 = Slave_FP_1_New ) then
            begin
                inc ( Array_Links [ Array_No ].LArray.Count );
                Array_Links [ Array_No ].LArray.
                    Link [ Array_Links [ Array_No ].
                        LArray.Count ] := Slave_No;
                Array_Links [ Array_No ].LArray.
                    Pointer [ Array_Links [ Array_No ].
                        LArray.Count ] := true;
                Array_Links [ Array_No ].
                    Point [ Slave_No ] := true;
            end else
            begin
                inc ( Array_Links [ Array_No ].LArray.Count );
                Array_Links [ Array_No ].LArray.
                    Link [ Array_Links [ Array_No ].
                        LArray.Count ] := Slave_No;
                Array_Links [ Array_No ].LArray.
                    Pointer [ Array_Links [ Array_No ].
                        LArray.Count ] := true;
                Array_Links [ Array_No ].
                    Point [ Slave_No ] := true;
            end;
        end;
    end;
    inc ( PreCount );
until PreCount > PreScon [ Slave_LP_1 ].Count;
PreCount := 1;
repeat
    if ( Array_Links [ Array_No ].Point[PreScon [ Slave_LP ].
        Array_Numbers [ PreCount ] ] = false ) and
        ( Array_No <> PreScon [ Slave_LP ].

```

```

        Array_Numbers [ PreCount ] ) then
begin
    Slave_No := PreScon [ Slave_LP ].
        Array_Numbers [ PreCount ];
    Slave_FP_New := Main_Buffers [ Slave_No ].
        Members [ 1 ];
    Slave_FP_1_New := Main_Buffers [ Slave_No ].
        Members [ 2 ];
    if ( Slave_LP = Slave_FP_New ) or
        ( Slave_LP = Slave_FP_1_New ) then
begin
        inc ( Array_Links [ Array_No ].LFArray.Count );
        Array_Links [ Array_No ].LFArray.
            Link [ Array_Links [ Array_No ].
                LFArray.Count ] := Slave_No;
        Array_Links [ Array_No ].LFArray.
            Pointer [ Array_Links [ Array_No ].
                LFArray.Count ] := true;
        Array_Links [ Array_No ].
            Point [ Slave_No ] := true;
    end else
begin
        inc ( Array_Links [ Array_No ].LLArray.Count );
        Array_Links [ Array_No ].LLArray.
            Link [ Array_Links [ Array_No ].
                LLArray.Count ] := Slave_No;
        Array_Links [ Array_No ].LLArray.
            Pointer [ Array_Links [ Array_No ].
                LLArray.Count ] := true;
        Array_Links [ Array_No ].
            Point [ Slave_No ] := true;
    end;
end;
inc ( PreCount );
until PreCount > PreScon [ Slave_LP ].Count;
end;
inc ( index_of_CoScon );
until index_of_CoScon > Main_Buffers [ Array_No ].
    Scons.Co_Scon_Count;
end;
end;
end;

(*****)
(*      --> MODULE FOUNDING  <--      *)
(*****)
(* Programmer   : Derya Yilmaz          *)
(* Coding-Date  : 06/27/1999            *)
(* Last Updated : 06/27/1999            *)
(*****)
procedure Module_Founder ( Start : smallint;
                           LF,LL,FF,FL : smallint;
                           Other_Links : boolean );

```

```

var
  FCount          : smallint;
{ to reconstruction the created array_sequence }
  FArray          : smallint; { stack to reconstruction }
  FOther          : boolean;
  index           : smallint;
  index_LF        : smallint;
  index_LL        : smallint;
  index_FF        : smallint;
  index_FL        : smallint;
  Desc_Order      : smallint;

{ Finds PartN and Creates Array_Sequence }
begin
  if ( Segment_Finish_Pointer [ Start ] = true ) then
    begin
      if ( Other_Links ) and
        ( Array_Links [ Dummy_Segment [ Start ].Back ].
          LArray.Pointer [ Start ] = true ) then
        begin
          Exit;
        end else
        begin
          index := 1;
          Dummy_Segment_Parts[index_Module,index_Order_No,index]:=Start;
          Desc_Order := Start;
          While ( Dummy_Segment [ Desc_Order ].Back <> 0 ) do
            begin
              inc ( index );
              Dummy_Segment_Parts[index_Module, index_Order_No,index]:=
                Dummy_Segment [ Desc_Order ].Back;
              Desc_Order := Dummy_Segment [ Desc_Order ].Back;
            end;
          FCount := 1;
          While ( FCount + 1 <= index ) do
            begin
              FArray := Dummy_Segment_Parts
                [ index_Module, index_Order_No, index ];
              Dummy_Segment_Parts[index_Module,index_Order_No,index]:=
                Dummy_Segment_Parts[index_Module,index_Order_No,FCount];
              Dummy_Segment_Parts[index_Module,index_Order_No,FCount]:=
                FArray;
              inc ( FCount );
              dec ( index );
            end;
            inc ( index_Order_No );
            Other_Links := true;
          end;
        end;
      if ( LF > 0 ) then
        begin
          if ( Array_Links [ Start ].LFArrary.Count <> 0 ) then
            begin
              index_LF := LF;
              if ( Other_Links ) then FOther := true

```

```

        else FOther := false;
    repeat
        if ( InUse [ Array_Links [ Start ].LFArrary.
            Link [ index_LF ] ] = false ) then
            begin
                Dummy_Segment [ Array_Links [ Start ].LFArrary.
                    Link [ index_LF ] ].Back := Start;
                InUse [ Array_Links [ Start ].LFArrary.
                    Link [ index_LF ] ] := true;
                InUse [ Start ] := true;
                Module_Founder ( Array_Links [ Start ].LFArrary.
                    Link [ index_LF ],1,0,0,0, FOther );
            end;
            inc ( index_LF );
        until ( index_LF > Array_Links [ Start ].
            LFArrary.Count );
    end;
end else
if ( LL > 0 ) and ( Not Other_Links ) then
begin
    if ( Array_Links [ Start ].LLArray.Count <> 0 ) then
        begin
            index_LL := LL;
            repeat
                if ( InUse [ Array_Links [ Start ].LLArray.
                    Link [ index_LL ] ] = false )
                    then
                        begin
                            Dummy_Segment [ Array_Links [ Start ].LLArray.
                                Link [ index_LL ] ].Back := Start;
                            InUse [ Array_Links [ Start ].LLArray.
                                Link [ index_LL ] ] := true;
                            InUse [ Start ] := true;
                            Module_Founder ( Array_Links [ Start ].LLArray.
                                Link [ index_LL ],1,0,0,0, False );
                        end;
                            inc ( index_LL );
            until ( index_LL > Array_Links [ Start ].
                LLArray.Count );
        end;
    end else
    if ( FF > 0 ) then
        begin
            if ( Array_Links [ Start ].FFArray.Count <> 0 ) then
                begin
                    index_FF := FF;
                    repeat
                        if ( InUse [ Array_Links [ Start ].FFArray.
                            Link [ index_FF ] ] = false )
                            and
                            ( Segment_Finish_Pointer [ Array_Links [ Start ].
                                FFArray.Link [ index_FF ] ] = false ) then
                            begin
                                Dummy_Segment [ Array_Links [ Start ].FFArray.
                                    Link [ index_FF ] ].Back := Start;
                                InUse [ Array_Links [ Start ].FFArray.

```

```

        Link [ index_FF ] := true;
        InUse [ Start ] := true;
        Module_Founder ( Array_Links [ Start ].FFArray.
            Link [ index_FF ], 1, 0, 0, 0, False );
    end;
    inc ( index_FF );
    until ( index_FF > Array_Links [ Start ].
        FFArray.Count );
end;
end else
if ( FL > 0 ) then
begin
    if ( Array_Links [ Start ].FLArray.Count <> 0 ) then
    begin
        index_FL := FL;
        repeat
            if ( InUse [ Array_Links [ Start ].FLArray.
                Link [ index_FL ] ] = false )
                and
                ( Segment_Finish_Pointer [ Array_Links [ Start ].
                    FLArray.Link [ index_FL ] ] = false ) then
            begin
                Dummy_Segment [ Array_Links [ Start ].FLArray.
                    Link [ index_FL ] ].Back := Start;
                InUse [ Array_Links [ Start ].FLArray.
                    Link [ index_FL ] ] := true;
                InUse [ Start ] := true; { Always running unfortunatly }
                Module_Founder ( Array_Links [ Start ].FLArray.
                    Link [ index_FL ], 1, 0, 0, 0, False );
            end;
            inc ( index_FL );
        until ( index_FL > Array_Links [ Start ].FLArray.Count );
    end;
end;
end;
end;

(*****)
(*          --> FIND RELEVANT SEGMENT  <--          *)
(*****)
(* Programmer   : Derya Yilmaz                      *)
(* Coding-Date   : 06/28/1999                        *)
(* Last Updated  : 06/28/1999                        *)
(*****)
Function Find_Relevant_Segment ( Rec_Segment : TSegment;
                                Module_No    : smallint
                                ): boolean;

var
    ACPO, ACPT      : TCon_State; { ACP = Alter_Con_Point }
    ANCO, ANCT      : smallint;   { ANC = Alter_No_Con }
    Start, Stop     : smallint;
    STA             : smallint;    { Sorted_Typed_Array_Number }
    index_Segment_Start : smallint;
    index_Segment_Finish : smallint;
    index_Array      : smallint;

begin

```

```

FillChar ( Segment_Start, Sizeof ( Segment_Start ), 0 );
FillChar ( Segment_Finish, Sizeof ( Segment_Finish ), 0 );
FillChar ( Segment_Start_Pointer,
          Sizeof ( Segment_Start_Pointer ), false );
FillChar ( Segment_Finish_Pointer,
          Sizeof ( Segment_Finish_Pointer ), false );
FillChar ( InUse, Sizeof ( InUse ), false );

index_Module := Module_No;
index_Order_No := 1;
index_Array := 1;
index_Segment_Start := 1;
index_Segment_Finish := 1;
repeat
  STA := Sort_Typed_Arrays [ index_Array ];
  if ( Locked_Arrays [ STA ] = false ) then
  begin
    ACPO := Rec_Segment.Part1.Con_Point [ 1 ];
    ACPT := Rec_Segment.Part1.Con_Point [ 2 ];
    ANCO := Rec_Segment.Part1.No_Con [ 1 ];
    ANCT := Rec_Segment.Part1.No_Con [ 2 ];
    case ACPO of
      First : begin
        if ( Array_Links [ STA ].FFArray.Count = ANCO )
        then
        begin
          if ( ( ACPT <> nil ) and
              ( Array_Links [ STA ].FLArray.Count = ANCT )
            )
          or
          ( ANCT = 0 ) then
          begin
            Segment_Start [ index_Segment_Start ] := STA;
            Segment_Start_Pointer [ STA ] := true;
            inc ( index_Segment_Start );
          end;
        end;
      end;
      Last : begin
        if ( Array_Links [ STA ].FLArray.Count = ANCO )
        then
        begin
          Segment_Start [ index_Segment_Start ] := STA;
          Segment_Start_Pointer [ STA ] := true;
          inc ( index_Segment_Start );
        end;
      end;
      Nil : begin
        if ( (Array_Links [ STA ].FFArray.Count = 0) and
            (Array_Links [ STA ].FLArray.Count = 0) or
            (Array_Links [ STA ].LFArray.Count = 0) and
            (Array_Links [ STA ].LLArray.Count = 0)) then
        begin
          Segment_Start [ index_Segment_Start ] := STA;
          Segment_Start_Pointer [ STA ] := true;
          inc ( index_Segment_Start );
        end;
      end;
    end;
  end;
  index_Array := index_Array + 1;
until index_Array = index_Segment_Finish;

```

```

        end;
    end;
    end; { for case }
    ACPO := Rec_Segment.PartN.Con_Point [ 1 ];
    ACPT := Rec_Segment.PartN.Con_Point [ 2 ];
    ANCO := Rec_Segment.PartN.No_Con [ 1 ];
    ANCT := Rec_Segment.PartN.No_Con [ 2 ];
    case ACPO of
        First : begin
            if ( Array_Links [ STA ].LFArray.Count = ANCO )
            then
                begin
                    if ( ( ACPT <> nill ) and
                        ( Array_Links [ STA ].LLArray.Count = ANCT )
                    )
                    or
                    ( ANCT = 0 ) then
                        begin
                            Segment_Finish [ index_Segment_Finish ] := STA;
                            Segment_Finish_Pointer [ STA ] := true;
                            inc ( index_Segment_Finish );
                        end;
                    end;
                end;
            end;
        Last : begin
            if ( Array_Links [ STA ].LLArray.Count = ANCO )
            then
                begin
                    Segment_Finish [ index_Segment_Finish ] := STA;
                    Segment_Finish_Pointer [ STA ] := true;
                    inc ( index_Segment_Finish );
                end;
            end;
        Nill : begin
            if ( (Array_Links [ STA ].FFArray.Count = 0) and
                (Array_Links [ STA ].FLArray.Count = 0) or
                (Array_Links [ STA ].LFArray.Count = 0) and
                (Array_Links [ STA ].LLArray.Count = 0) ) then
                begin
                    Segment_Finish [ index_Segment_Finish ] := STA;
                    Segment_Finish_Pointer [ STA ] := true;
                    inc ( index_Segment_Finish );
                end;
            end;
        end;
    end; { for case }
    end;
    inc ( index_Array );
until ( index_Array > No_TArray );
{ End of searching for part1 and partn }
For Start := 1 to index_Segment_Start - 1 do
    for Stop := 1 to index_Segment_Finish - 1 do
        begin
            if ( Segment_Start [ Start ] <> 0 ) and
                ( Segment_Finish [ Stop ] <> 0 )
            then Module_Founder ( Segment_Start [ Start ],
                                1,0,0,0, false );
        end;
    end;
end;

```

```

end;
end;

(*****)
(*      -->  STARTING DERYA ALGORITHM  <--      *)
(*****)
(* Programmer   : Derya Yilmaz                      *)
(* Coding-Date  : 03/19/1999                        *)
(* Last Updated : 07/26/1999                        *)
(*****)
procedure Algorithm_of_Derya (Width, Height : smallint) ;
var
  idx, Segment_No, Module_No : smallint;
begin
  Reconstruction_Given_Character ( Width, Height );
  if Calculating_Founding_Extracting ( Width, Height ) then
    begin
      Catching_Scons;
      Cleanup_ThreeTuple_Arrays;
      Removing_Pyramids_to_Land;
      Removing_Triple_SBS_Array;
      Removing_Triple_HV_Array;
      Cleanup_ThreeTuple_Arrays;
      Cleanup_Unneccassary_TwoTuple_Arrays_Which_Confirm_Before;
      Join_The_Same_Dir_Arrays;
      Breaking_The_UnBreaking_Arrays;
      Cleanup_TwoTuple_Arrays;
      Transferring_Arrays;
      Sorting_X_Via_Y (Width, Height );
      Sorted_Typed_Arrays;
      Defines_Arrays_Link;
      FillChar ( Dummy_Segment, Sizeof( Dummy_Segment ), 0);
      for idx := 1 to 62 do
        for Segment_No := 1 to 10 do
          for Module_No := 1 to 10 do
            begin
              Find_Relevant_Segment( DBOCR [ idx ].Segments[ Segment_No ],
                Module_No );
            end;
          end;
        end;
      end;
    end;
  end;
end.

```

EK-2

DECRA İÇİN HAZIRLANAN DATABASE YAPISI

```

(*****)
(*      --->  THE DATABASE STRUCTURE FOR DECRA <---      *)
(*****)

unit DB;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
  Dialogs,
  StdCtrls, ExtCtrls;

Type
  TDIRECTION = ( NONE, VERTICAL, HORIZONTAL, SLASH, BACKSLASH );
  TCon_State  = ( Nill, First, Last );
  TTypeofChar = ( All, LowerCase, UpperCase );

  TPART      = record
    Con_Point : Array [ 1..2 ] of TCon_State;
    No_Con    : Array [ 1..2 ] of smallint;
    Con_Point_Dir : Array [ 1..2 ] of TDIRECTION;
  end;

  TLINKED    = record
    Parts : Array [ 1..2 ] of record
      Segment_No : Array [ 1..4 ] of smallint;
      Part : Array [ 1..4 ] of TCon_State;
    end;
  end;

  TSEGMENT    = record
    Part1 : TPart;
    PartN : TPart;
    Linked : TLinked;
    Seg_Dir : TDIRECTION;
  end;

  TCHARACTER_INFO = record
    ASCII_Value : smallint;
    No_Segment : smallint;
    Segments : Array [ 1..10 ] of TSegment;
  end;

type
  TMYDB = class(TForm)
    Ascii_Code: TEdit;
    Label1: TLabel;
    Label2: TLabel;
  end;

```

```
No_Segments: TEdit;
Segments: TRadioGroup;
Label3: TLabel;
Label4: TLabel;
Label5: TLabel;
Seg_Dir: TComboBox;
Label6: TLabel;
Label7: TLabel;
Label8: TLabel;
Seg_P1Con1: TComboBox;
Label9: TLabel;
Seg_P1Con2: TComboBox;
Label10: TLabel;
Seg_P1NoCon1: TEdit;
Label11: TLabel;
Seg_P1NoCon2: TEdit;
Label12: TLabel;
Label13: TLabel;
Seg_P1Dir1: TComboBox;
Seg_P1Dir2: TComboBox;
Label14: TLabel;
Label15: TLabel;
Label16: TLabel;
Seg_PNCon1: TComboBox;
Label17: TLabel;
Seg_PNCon2: TComboBox;
Label18: TLabel;
Seg_PNNoCon1: TEdit;
Label19: TLabel;
Seg_PNNoCon2: TEdit;
Label20: TLabel;
Label21: TLabel;
Seg_PNDir1: TComboBox;
Seg_PNDir2: TComboBox;
Label22: TLabel;
Label23: TLabel;
Seg_Part1: TComboBox;
Label24: TLabel;
Seg_Part2: TComboBox;
Label25: TLabel;
Seg_Part3: TComboBox;
Label26: TLabel;
Seg_Part4: TComboBox;
Label27: TLabel;
Seg_No1: TEdit;
Seg_No2: TEdit;
Seg_No3: TEdit;
Seg_No4: TEdit;
Label28: TLabel;
Label29: TLabel;
Label30: TLabel;
Label31: TLabel;
Button1: TButton;
Label32: TLabel;
Edit1: TEdit;
Button2: TButton;
```

```

Edit2: TEdit;
Button3: TButton;
Button4: TButton;
Label33: TLabel;
Label34: TLabel;
Label35: TLabel;
Label36: TLabel;
Seg_No5: TEdit;
Seg_No6: TEdit;
Seg_No7: TEdit;
Seg_No8: TEdit;
Label37: TLabel;
Label38: TLabel;
Label39: TLabel;
Label40: TLabel;
Label41: TLabel;
Label42: TLabel;
Seg_Part5: TComboBox;
Seg_Part6: TComboBox;
Seg_Part7: TComboBox;
Seg_Part8: TComboBox;
procedure RefreshingScreen (idx, Seg_No : integer);
procedure FormShow(Sender: TObject);
procedure FormCreate(Sender: TObject);
procedure SegmentsClick(Sender: TObject);
procedure Seg_P1Con1Change(Sender: TObject);
procedure Seg_P1Con2Change(Sender: TObject);
procedure Seg_P1NoCon1Change(Sender: TObject);
procedure Seg_P1NoCon2Change(Sender: TObject);
procedure Seg_P1Dir1Change(Sender: TObject);
procedure Seg_P1Dir2Change(Sender: TObject);
procedure Seg_PNCon1Change(Sender: TObject);
procedure Seg_PNCon2Change(Sender: TObject);
procedure Seg_PNNoCon1Change(Sender: TObject);
procedure Seg_PNNoCon2Change(Sender: TObject);
procedure Seg_PNDir1Change(Sender: TObject);
procedure Seg_PNDir2Change(Sender: TObject);
procedure Seg_No1Change(Sender: TObject);
procedure Seg_No2Change(Sender: TObject);
procedure Seg_No3Change(Sender: TObject);
procedure Seg_No4Change(Sender: TObject);
procedure Seg_Part1Change(Sender: TObject);
procedure Seg_Part2Change(Sender: TObject);
procedure Seg_Part3Change(Sender: TObject);
procedure Seg_Part4Change(Sender: TObject);
procedure Ascii_CodeChange(Sender: TObject);
procedure No_SegmentsChange(Sender: TObject);
procedure Seg_DirChange(Sender: TObject);
procedure Button4Click(Sender: TObject);
procedure Button3Click(Sender: TObject);
procedure Button1Click(Sender: TObject);
procedure Button2Click(Sender: TObject);
procedure Seg_No5Change(Sender: TObject);
procedure Seg_No6Change(Sender: TObject);
procedure Seg_No7Change(Sender: TObject);
procedure Seg_No8Change(Sender: TObject);

```

```

    procedure Seg_Part5Change(Sender: TObject);
    procedure Seg_Part6Change(Sender: TObject);
    procedure Seg_Part7Change(Sender: TObject);
    procedure Seg_Part8Change(Sender: TObject);
private
    { Private declarations }
public
    { Public declarations }
end;

var
    MYDB: TMYDB;
    DBOCR : Array [1..62] of TCharacter_Info;
    index : integer = 1;
    Segment_No : integer = 1;

Implementation
Uses unit1, Deryaalg;

{$R *.DFM}

procedure TMYDB.RefreshingScreen (idx, Seg_No : integer);
begin
    Seg_P1Con1.ItemIndex :=
integer(DBOCR[idx].Segments[Seg_No].Part1.Con_Point[1]);
    Seg_P1Con2.ItemIndex :=
integer(DBOCR[idx].Segments[Seg_No].Part1.Con_Point[2]);
    Seg_P1NoCon1.text :=
inttostr(DBOCR[idx].Segments[Seg_No].Part1.No_Con[1]);
    Seg_P1NoCon2.text :=
inttostr(DBOCR[idx].Segments[Seg_No].Part1.No_Con[2]);
    Seg_P1Dir1.ItemIndex :=
integer(DBOCR[idx].Segments[Seg_No].Part1.Con_Point_Dir[1]);
    Seg_P1Dir2.ItemIndex :=
integer(DBOCR[idx].Segments[Seg_No].Part1.Con_Point_Dir[2]);

    Seg_PNCon1.ItemIndex :=
integer(DBOCR[idx].Segments[Seg_No].PartN.Con_Point[1]);
    Seg_PNCon2.ItemIndex :=
integer(DBOCR[idx].Segments[Seg_No].PartN.Con_Point[2]);
    Seg_PNNoCon1.text :=
inttostr(DBOCR[idx].Segments[Seg_No].PartN.No_Con[1]);
    Seg_PNNoCon2.text :=
inttostr(DBOCR[idx].Segments[Seg_No].PartN.No_Con[2]);
    Seg_PNDir1.ItemIndex :=
integer(DBOCR[idx].Segments[Seg_No].PartN.Con_Point_Dir[1]);
    Seg_PNDir2.ItemIndex :=
integer(DBOCR[idx].Segments[Seg_No].PartN.Con_Point_Dir[2]);

    Seg_No1.text :=
inttostr(DBOCR[idx].Segments[Seg_No].Linked.Parts[1].Segment_No[1]);
    Seg_No2.text :=
inttostr(DBOCR[idx].Segments[Seg_No].Linked.Parts[1].Segment_No[2]);
    Seg_No3.text :=
inttostr(DBOCR[idx].Segments[Seg_No].Linked.Parts[1].Segment_No[3]);

```

```

    Seg_No4.text :=
    inttostr(DBOCR[idx].Segments[Seg_No].Linked.Parts[1].Segment_No[4]);

    Seg_No5.text :=
    inttostr(DBOCR[idx].Segments[Seg_No].Linked.Parts[2].Segment_No[1]);
    Seg_No6.text :=
    inttostr(DBOCR[idx].Segments[Seg_No].Linked.Parts[2].Segment_No[2]);
    Seg_No7.text :=
    inttostr(DBOCR[idx].Segments[Seg_No].Linked.Parts[2].Segment_No[3]);
    Seg_No8.text :=
    inttostr(DBOCR[idx].Segments[Seg_No].Linked.Parts[2].Segment_No[4]);

```

```

    Seg_Part1.ItemIndex :=
    integer(DBOCR[idx].Segments[Seg_No].Linked.Parts[1].Part[1]);
    Seg_Part2.ItemIndex :=
    integer(DBOCR[idx].Segments[Seg_No].Linked.Parts[1].Part[2]);
    Seg_Part3.ItemIndex :=
    integer(DBOCR[idx].Segments[Seg_No].Linked.Parts[1].Part[3]);
    Seg_Part4.ItemIndex :=
    integer(DBOCR[idx].Segments[Seg_No].Linked.Parts[1].Part[4]);

```

```

    Seg_Part5.ItemIndex :=
    integer(DBOCR[idx].Segments[Seg_No].Linked.Parts[2].Part[1]);
    Seg_Part6.ItemIndex :=
    integer(DBOCR[idx].Segments[Seg_No].Linked.Parts[2].Part[2]);
    Seg_Part7.ItemIndex :=
    integer(DBOCR[idx].Segments[Seg_No].Linked.Parts[2].Part[3]);
    Seg_Part8.ItemIndex :=
    integer(DBOCR[idx].Segments[Seg_No].Linked.Parts[2].Part[4]);

```

```

    Seg_Dir.ItemIndex := integer(DBOCR[idx].Segments[Seg_No].Seg_Dir);

```

```

    Ascii_Code.text := inttostr(DBOCR[idx].Ascii_Value);

```

```

    No_Segments.text := inttostr(DBOCR[idx].No_Segment);
end;

```

```

procedure TMYDB.FormShow(Sender: TObject);

```

```

var

```

```

    r : TRect;

```

```

begin

```

```

    SystemParametersInfo(SPI_GETWORKAREA, 0, @r, 0);

```

```

    Left := r.right div 2 - width div 2;

```

```

    Top := r.Bottom div 2 - height div 2;

```

```

    RefreshingScreen (index, Segment_No);

```

```

end;

```

```

procedure TMYDB.FormCreate(Sender: TObject);

```

```

begin

```

```

    fillchar(DBOCR, sizeof(DBOCR), 0);

```

```

end;

```

```

procedure TMYDB.SegmentsClick(Sender: TObject);

```

```

begin
    Segment_No := Segments.ItemIndex + 1;
    RefreshingScreen (index, Segment_No);
    Label4.caption := inttostr(Segment_No);
end;

procedure TMYDB.Seg_P1Con1Change(Sender: TObject);
begin
    DBOCR[index].Segments[Segment_No].Part1.Con_Point[1] :=
    TCon_State(Seg_P1Con1.ItemIndex);
end;

procedure TMYDB.Seg_P1Con2Change(Sender: TObject);
begin
    DBOCR[index].Segments[Segment_No].Part1.Con_Point[2] :=
    TCon_State(Seg_P1Con2.ItemIndex);
end;

procedure TMYDB.Seg_P1NoCon1Change(Sender: TObject);
begin
    DBOCR[index].Segments[Segment_No].Part1.No_Con[1] :=
    strtoint(Seg_P1NoCon1.text);
end;

procedure TMYDB.Seg_P1NoCon2Change(Sender: TObject);
begin
    DBOCR[index].Segments[Segment_No].Part1.No_Con[2] :=
    strtoint(Seg_P1NoCon2.text);
end;

procedure TMYDB.Seg_P1Dir1Change(Sender: TObject);
begin
    DBOCR[index].Segments[Segment_No].Part1.Con_Point_Dir[1] :=
    TDirection(Seg_P1Dir1.ItemIndex);
end;

procedure TMYDB.Seg_P1Dir2Change(Sender: TObject);
begin
    DBOCR[index].Segments[Segment_No].Part1.Con_Point_Dir[2] :=
    TDirection(Seg_P1Dir2.ItemIndex);
end;

procedure TMYDB.Seg_PNCon1Change(Sender: TObject);
begin
    DBOCR[index].Segments[Segment_No].PartN.Con_Point[1] :=
    TCon_State(Seg_PNCon1.ItemIndex);
end;

procedure TMYDB.Seg_PNCon2Change(Sender: TObject);
begin
    DBOCR[index].Segments[Segment_No].PartN.Con_Point[2] :=
    TCon_State(Seg_PNCon2.ItemIndex);
end;

```

```

procedure TMYDB.Seg_PNNoCon1Change(Sender: TObject);
begin
  DBOCR[index].Segments[Segment_No].PartN.No_Con[1] :=
    strtoint(Seg_PNNoCon1.text);
end;

procedure TMYDB.Seg_PNNoCon2Change(Sender: TObject);
begin
  DBOCR[index].Segments[Segment_No].PartN.No_Con[2] :=
    strtoint(Seg_PNNoCon2.text);
end;

procedure TMYDB.Seg_PNDir1Change(Sender: TObject);
begin
  DBOCR[index].Segments[Segment_No].PartN.Con_Point_Dir[1] :=
    TDirection(Seg_PNDir1.ItemIndex);
end;

procedure TMYDB.Seg_PNDir2Change(Sender: TObject);
begin
  DBOCR[index].Segments[Segment_No].PartN.Con_Point_Dir[2] :=
    TDirection(Seg_PNDir2.ItemIndex);
end;

procedure TMYDB.Seg_No1Change(Sender: TObject);
begin
  DBOCR[index].Segments[Segment_No].Linked.Parts[1].Segment_No[1] :=
    strtoint(Seg_No1.text);
end;

procedure TMYDB.Seg_No2Change(Sender: TObject);
begin
  DBOCR[index].Segments[Segment_No].Linked.Parts[1].Segment_No[2] :=
    strtoint(Seg_No2.text);
end;

procedure TMYDB.Seg_No3Change(Sender: TObject);
begin
  DBOCR[index].Segments[Segment_No].Linked.Parts[1].Segment_No[3] :=
    strtoint(Seg_No3.text);
end;

procedure TMYDB.Seg_No4Change(Sender: TObject);
begin
  DBOCR[index].Segments[Segment_No].Linked.Parts[1].Segment_No[4] :=
    strtoint(Seg_No4.text);
end;

procedure TMYDB.Seg_Part1Change(Sender: TObject);
begin
  DBOCR[index].Segments[Segment_No].Linked.Parts[1].Part[1] :=
    TCon_State(Seg_Part1.ItemIndex);
end;

procedure TMYDB.Seg_Part2Change(Sender: TObject);
begin

```

```

    DBOCR[index].Segments[Segment_No].Linked.Parts[1].Part[2] :=
TCon_State(Seg_Part2.ItemIndex);
end;

```

```

procedure TMYDB.Seg_Part3Change(Sender: TObject);
begin
    DBOCR[index].Segments[Segment_No].Linked.Parts[1].Part[3] :=
TCon_State(Seg_Part3.ItemIndex);
end;

```

```

procedure TMYDB.Seg_Part4Change(Sender: TObject);
begin
    DBOCR[index].Segments[Segment_No].Linked.Parts[1].Part[4] :=
TCon_State(Seg_Part4.ItemIndex);
end;

```

```

procedure TMYDB.Ascii_CodeChange(Sender: TObject);
begin
    DBOCR[index].Ascii_Value := strtoint(Ascii_Code.text);
end;

```

```

procedure TMYDB.No_SegmentsChange(Sender: TObject);
begin
    DBOCR[index].No_Segment := strtoint(No_Segments.text);
end;

```

```

procedure TMYDB.Seg_DirChange(Sender: TObject);
begin
    DBOCR[index].Segments[Segment_No].Seg_Dir :=
TDirection(Seg_Dir.ItemIndex);
end;

```

```

procedure TMYDB.Button4Click(Sender: TObject);
var
    idx : integer;
begin
    idx := 1;
    while DBOCR[idx].Ascii_Value <> 0 do
        inc(idx);
    if idx < 50 then
        begin
            index := idx;
            Segment_No := 1;
            Segments.ItemIndex := 0;
            RefreshingScreen (index, Segment_No);
        end;
    end;
end;

```

```

procedure TMYDB.Button3Click(Sender: TObject);
var
    idx : integer;
    count : integer;
begin
    idx := 0;
    for count := 1 to 50 do
        begin

```

```

    if DBOCR[count].Ascii_Value = strtoint(edit2.text) then
    begin
        idx := count;
        break;
    end;
end;
if idx <> 0 then
begin
    index := idx;
    Segment_No := 1;
    Segments.ItemIndex := 0;
    RefreshingScreen (index, Segment_No);
end;
end;

procedure TMYDB.Button1Click(Sender: TObject);
var
    dosya : file;
begin
    try
        assignfile(dosya,edit1.text);
        rewrite(dosya,1);
        Blockwrite(dosya,DBOCR,sizeof(DBOCR));
        closefile(dosya);
    except
        MessageDlg('Hata olustu', mtInformation,[mbOk], 0);
    end;
end;

procedure TMYDB.Button2Click(Sender: TObject);
var
    dosya : file;
begin
    try
        assignfile(dosya,edit1.text);
        reset(dosya,1);
        Blockread(dosya,DBOCR,sizeof(DBOCR));
        closefile(dosya);
        index := 1;
        Segment_No := 1;
        Segments.ItemIndex := 0;
        RefreshingScreen (index, Segment_No);
    except
        MessageDlg('Hata olustu', mtInformation,[mbOk], 0);
    end;
end;

procedure TMYDB.Seg_No5Change(Sender: TObject);
begin
    DBOCR[index].Segments[Segment_No].Linked.Parts[2].Segment_No[1] :=
    strtoint(Seg_No5.text);
end;

procedure TMYDB.Seg_No6Change(Sender: TObject);
begin

```

```

    DBOCR[index].Segments[Segment_No].Linked.Parts[2].Segment_No[2] :=
    strtoint(Seg_No6.text);
end;

procedure TMYDB.Seg_No7Change(Sender: TObject);
begin
    DBOCR[index].Segments[Segment_No].Linked.Parts[2].Segment_No[3] :=
    strtoint(Seg_No7.text);
end;

procedure TMYDB.Seg_No8Change(Sender: TObject);
begin
    DBOCR[index].Segments[Segment_No].Linked.Parts[2].Segment_No[4] :=
    strtoint(Seg_No8.text);
end;

procedure TMYDB.Seg_Part5Change(Sender: TObject);
begin
    DBOCR[index].Segments[Segment_No].Linked.Parts[2].Part[1] :=
    TCon_State(Seg_Part5.ItemIndex);
end;

procedure TMYDB.Seg_Part6Change(Sender: TObject);
begin
    DBOCR[index].Segments[Segment_No].Linked.Parts[2].Part[2] :=
    TCon_State(Seg_Part6.ItemIndex);
end;

procedure TMYDB.Seg_Part7Change(Sender: TObject);
begin
    DBOCR[index].Segments[Segment_No].Linked.Parts[2].Part[3] :=
    TCon_State(Seg_Part7.ItemIndex);
end;

procedure TMYDB.Seg_Part8Change(Sender: TObject);
begin
    DBOCR[index].Segments[Segment_No].Linked.Parts[2].Part[4] :=
    TCon_State(Seg_Part8.ItemIndex);
end;

end.

```

ÖZGEÇMİŞ

1974 yılında Tokat'ta doğdu. İlk ve orta öğrenimini Ankara'da tamamladı. 1995 yılında Gazi Üniversitesi Teknik Eğitim Fakültesi, Elektronik-Bilgisayar Eğitimi Bölümü'nden mezun oldu. 1995-1997 yıllarında İzmir Bayındır Endüstri Meslek Lisesi'nde Elektronik Bölüm Şefi ve öğretmeni olarak görev yaptı. Şubat 1997'de istifa ederek Başkent Üniversitesi Teknik Bilimler Meslek Yüksekokulu'nda Öğretim Görevlisi olarak çalışmaya başladı. Aynı dönem Gazi Üniversitesi Fen Bilimleri Enstitüsü'ne, Elektronik-Bilgisayar Eğitimi Yüksek Lisans öğrencisi olarak girdi. T.S.E. ISO 9001 Temel Eğitim ve Tetkikçi belgelerini alarak Başkent Üniversitesi'nin kalite çalışmalarında; bölüm içinde Kalite Sorumlusu, üniversite içinde de Baş Tetkikçi olarak görev aldı. 1998'de uluslararası IVETA'98 konferansına katılarak, kalite sisitemi ile ilgili makalesini sundu. Halen yüksek lisansını tez aşamasında yürütmektedir.

T.C. YÜKSEKÖĞRETİM BAKANLIĞI
DOKÜMAN YÖNETİMİ MERKEZİ