10 Algoritmalar ve Sonlu Durumlu Makinalar

10.1 Algoritmalar ve Karmaşıklık

Ayrık matematikte karşılaşılan birçok problem sınıfı mevcuttur. Örneğin verilen tamsayı grubu içindeki en büyük olanının bulunması, verilen bir kümenin bütün alt kümelerinin listelenmesi, verilen bir tamsayı kümesinin artan sırada sıraya dizilmesi, verilen bir ağ'da iki kenar arasındaki en kısa yol'un bulunması gibi. Böyle problemler ile karşılaşıldığında, ilk olarak problemin matematiksel yapısını içeren bir modelinin bulunması gerekir. Böyle modellerde, permutasyonlar, bağıntılar, graflar, ağaçlar ve sonlu durumlu makinalar gibi ayrık yapılar sıkça kullanılırlar.

Uygun matematiksel modelin kurulması çözümün ilk adımıdır. Modeli kullanarak çözümü gerçekleştiren bir yöntem gerekli olacaktır. Cevabı bulmak için sıralı adımları takip eden bir yordam olacaktır. Böyle sıralı işlem adımlarına Algoritma denir.

Tanım: ***Algoritma;*** *bir hesaplamayı gerçekleştirmek veya bir problemi çözmek için kesin işlemlerin sonlu bir kümesine algoritma denir*

Algoritma kelimesi Arap matematikçisi olan ve 9. yüzyılda yaşamış olan Al-Khowarizmi 'nin isminden türetilmiştir.

Örnek: Sonlu sayıdaki tamsayılar kümesinin en büyük elemanının bulunması için bir algoritma kurulmaya çalışılırsa:

Problemin uygulaması çeşitli olabilir. Örneğin üniversite öğrencileri arasında derecesi en yüksek olanın belirlenmesi, bir spor organizasyonunda en yüksek dereceli olan sporcunun belirlenmesi gibi.

Problemin birçok çözümü olabilir. Bir çözüm aşağıda verilmiştir.

1. Sayılarin ilkini geçici en büyük olarak belirle.
2. Bir sonraki sayı ile geçici en büyük tamsayıyı karşılaştır. Eğer yeni sayı geçiçi enbüyükten büyük ise yeni sayıyı geçici enbüyük olarak belirle.
3. Eğer daha sayı var ise bir önceki adımı tekrarla.
4. Dizide başka eleman kalmamış ise dur. Bu noktada en büyük sayı geçici enbüyük olarak belirlenmiş olan sayı olacaktır.

Algoritma bir bilgisayar dili yardımı ile gösterilebilir. Ancak bunu anlamak çoğu zaman zor olur. Bunun yerine ortak olarak kullanılan ***pseudokod*** ile ifade edilir. Bu kod ingilizce olarak ifade edilen işlem adımlarını gösterir.

Psudokod temel bilgileri: Bu bölümde psudokod ile ilgili temel bilgiler(Algoritmaları anlatmak için kullanılan) kısa olarak verilecektir**.**
Ayıraçlar : begin ,end , ; dir.

```
Begin
   Program
End
```

Bildiriler : procedure, begin, integer,boolean, real, array, string
İşlemler

End.
Procedure'ların arasında arşiv fonksiyonlarını belirtmeye gerek yok.

Atama : =, :=
örnek b:=2 , c:=3, a:=b+c gibi
Psodokod aşağıdaki gibi blok yapısındadır.

```
begin
        real temp;
        temp:=a;
        a:=b;
        b:=temp;
end
```

Kontrol yapıları :

if p **then** s_1 **else** s_2; { eğer p önermesi doğru ise s_1 'i değil ise s_2'yi yap.}

```
if a>b then
        begin
        ...
        end
else
        begin
        .......
        end
```

```
for j :=1 to n do
        begin
        .....
        end {j nin değerini 1denbaşlatarak arttır. j=n oluncays kadar begin end bloğunu icra et}
```

while p **do** s {p önermesi doğru olduğu sürece s'i icra et. İşlem sonunda p'nin değişmesi gerekir}

```
örnek   j:=1;
        toplam:=j;
        while j <10 do
        begin
                j:=j+1;
                toplam:= toplam+j;
        end
```

do s **until** p {p önermesi sağlanıncaya kadar s işlemini yap. s işlemi p önermesine etkili olmalıdır}

```
örnek :
j:=1;
toplam:=j;
do
begin
        j:= j+1;
        toplam:= toplam+1;
end
until j = 10;
```

Örnek . Enbüyük tam sayıyı bulma algoritması

Procedure *enbuyuk*($a_1, a_2,,....., a_n$: integers)
enbuyuk := a_1
for *i* := 2 **to** *n*
    if *enbuyuk* < a_i **then** *enbuyuk* :=a_i
{işlem sonunda enbuyuk tamsayı bulunmuş olur}

Başka bir algoritma da bu tamsayıları büyükten küçüğe doğru azalan şekilde sıralayıp ilk elemenı enbuyuk olarak almak olabilir.

Algoritmalarda aşağıdaki özellikler bulunur ve bu özellikleri akıldan çıkartmamak gereklidir.

Giriş: Belirlenen veri kümesinden algoritma giriş değerleri alır.

Çıkış: Algoritma herbir giriş kümesinde çıkış değerleri üretir. Bu değerler problemin çözümüdür.

Açıklık : Algoritmanın adımları açık olarak tanımlanmalıdır.

Doğruluk: Algoritma herbir giriş kümesi için doğru çıkış üretmelidir.

Sonluluk : Algoritma, herbir giriş kümesi için amaçlanan çıkışı, sonlu işlem adımı(büyük olabilir) sonunda üretmelidir

Verimlilik : Algoritmanın herbir adımı tam ve sonlu bir zaman diilimnde gerçeklenmelidir.

Genellik : Yordam formdaki her probleme uygulanabilecek şekilde genel olmalıdır.

Enbüyük tamsayıyı bulma algoritması bu açıdan değerlendirilirse;

Giriş : Sonlu sayıda tamsayı kümesi
Çıkış : kümedeki en büyük tamsayı
Açık olarak adımlar tanımlanmıştır, ve doğru sonuç üretir.
Algoritma sonlu işlem adımı kullanır(n ad)
Algoritma herbir adımda bir karşılaştırma işlemi yapar.(verimlilik)
Algoritma bu tür kümelerdeki enbüyük tamsayıyı bulacak şekilde geneldir.

Arama Algoritmaları:

Sıralı listedeki bir elemanın yerinin bulunması çok değişik olarak karşılaşılan bir problemdir. Örneğin sözlükten bir kelime aranması gibi problemlere arama problemleri denir.

Genel arama problemi aşağıdaki şekilde açıklanabilir.: Farklı elemanları $a_1, a_2,,....., a_n$ olan bir listede bir *x* elemanın yerinin öğrenilmesi veya listede olup olmadığının öğrenilmesi şeklinde olabilir. Bu arama probleminin çözümü, x elemanına eşit olan a_i elemanın yerinin bulunmasıdır. (eğer $x = a_i$ ise x i. Elemandır.)

Problemin çözümü için ilk algoritma doğrusal veya ardışıl aramadır. Doğrusal arama algoritması, x ve a_1'i karşılaştırarak işleme başlar. Eğer $x = a_1$ ise aranan eleman 1. elemandır. $x \neq a_1$ ise, x ile a_2 karşılaştırılır. Eğer $x = a_2$ ise çözüm a_2'nin konumudur. Eğer $x \neq a_2$ ise, x , a_3 ile karşılaştırılır.Bu işlem bir uyuşma bulununcaya kadar devam eder.Uyuşma olmadıkça işlem devam eder. Eğer bir uyuşma bulunamaz ise sonuç sıfır olarak elde edilir. Doğrusal arama algoritmasının pseudokod'u aşağıda verilmiştir.

Procedure dogrusalarama(x: integer, $a_1, a_2,,....., a_n$: farklı tamsayılar)
i := 1;
while($i \leq n$ and $x \neq a_i$)
    i := i+1;
if $i \leq n$ **then** konum := i
else konum := 0; {konum, x'e eşit olan terimin indisidir.Eğer x bulunamamış ise değeri sıfırdır}

Şimdi başka bir algoritma düşüneceğiz.

Bu algoritmada verilen veriler artan şekilde sıralanmıştır. Veriler tamsayı ise en küçükten en büyüğe doğru sıralanmış, eğer kelime iseler alfabetik olarak sıralı şekildedir. Böyle bir veri kümesinde bir eleman aranması için kullanılacak algoritma, ikili aramadır. İkili arama algoritmasının mantığı sıralı veri kümesi ortadan iki kümeye ayrılarak bulunması istenen veri bu alt kümelerden hangisinin içerisinde olabileceğine bakılır. Arama işlemi alt kümelerde tekrarlanarak bulunması gerekli olan veri bulunmaya çalışılır. Aşağıdaki örnek ikili aramayı gösterir.

Örnek: 1,2,3,5,6,7,8,10,12,13,15,16,18,19,20,22 sıralı dizisi içerisinde 19 tamsayısı aransın.

Dizide 16 eleman bulunduğundan 1,2,3,5,6,7,8,10 12,13,15,16,18,19,20,22 şeklinde 8'li iki alt kümeye ayrılır. 19 tamsayısı birinci alt kümenin enbüyük elemanı ile karşılaştırılır. 10 < 19 olduğundan aranan sayı ikinci alt kümededir. Bundan sonra ikinci alt küme 12,13,15,16 18,19,20,22 olmak üzere 4 elemanlı iki alt kümeye ayrılır.

10 < 19 olduğundan aranan tamsayı sağ alt kümede olabilecektir. Bu nedenle sağ alt küme yine 18,19 ve 20,22 olmak üzere iki elemanlı iki alt kümeye ayrılır.

Şimdi 19 tamsayısı, son ikili kümenin en büyük elemanından büyük olmadığı için arama ilk kümenin 13. ve 14. elemanını içeren kümeyle sınırlanır. Böylece son kümede 18 ve 19 tamsayılı ve birer elemanlı iki alt kümeye ayrılır. 18 < 19 olduğundan arama 19 tamsayısından oluşan son kümeye sınırlanır. ve kümenin 14. elemanı olarak bulunur.

Algoritmanın pseudokod'u aşağıda verilmiştir.

Procedure ikiliarama(x: integer, $a_1, a_2,,....., a_n$: artan tamsayılar)
i := 1; { i ,arama aralığının sol bitiş noktasını gösterir}
j := n; { j ,arama aralığının sağ bitiş noktasını gösterir}

while i < j **do**
begin
m := [(i+j)/2];
if x > a_m **then** i:= m+1;
else j := m;
end
if x = a_i **then** konum := i
else konum := 0;
{ konum, x'e eşit olan terimin indisidir. Veya eğer x bulunamamış ise değeri sıfırdır}

Algoritmaların karmaşıklığı

Algoritmaların özellikleri içerisinde verimlilik olması gerektiği açıklanmıştı. Algoritmanın verimliliği ne demektir? Bunun analizi nasıl yapılır? Verimliliğin bir ölçütü, algoritmanın belirli bir giriş verisine karşın, problemin çözümü için bilgisayarın harcadığı zamanın ölçülmesidir. Diğer bir ölçü ise belirli giriş verisine karşı bilgisayarın kullandığı bellek miktarıdır. Böyle sorular algoritmanın bir hesaplama karmaşıklığının geliştirilmesini gerektirir. Problemi çözmek için algoritmanın harcadığı zamanın analizi zaman karmaşıklığı'nı, gerekli belleğin analizi ise yer(space) karmaşıklığının hesabını gerektirir.

Yer karmaşıklığı probleminin çözümü, algoritmayı gerçeklerken kullanılan veri yapıları ile bağlantılıdır. Ancak bu konular içerisinde yer karmaşıklığından bahsedilmeyecektir.

Algoritmanın zaman karmaşıklığı ise, belirli miktardaki giriş verisine karşılık, yapılan karşılaştırma, tamsayı toplama, tamsayı çıkartma, tamsayı çarpma ve bölme işlemleri ile diğer basit işlemlerin sayısı olarak hesaplanır.

Örnek(Zaman karmaşıklığı için) : Bir A dizisinin en küçük elemanını bulan algoritmanın zaman karmaşıklığının hesabı:

Procedure *enkucuk*(*A real array,*, *enkucuk*:*real*)
enkucuk := A[1];
for *i* := 2 **to** *n*
    begin
      if A[i] < *enkucuk* **then** *enkucuk :=* A[i] { en kötü durumda ***n-1*** defa icra edilir.}
    end
{işlem sonunda enküçük sayı bulunmuş olur}

Bu algortimanın zaman karmaşıklığı en kötü durumda dizinin büyüklüğü mertebesindedir. Bu yordamın işlem sayısını hesaplamaya çalışalım. Mertebesi n-1dir.

Karşılaştırma işlemlerinin sayısı : n-1

Atama işlemlerinin sayısı: n-1 ; Algoritmanın karmaşıklığı(zaman) O(n) dir.

Karmaşıklığı ifade etmek için O(n) notasyonu kullanılır. O mertebe işareti , (n) in sonlu bir çarpanla çarpımından daha küçüktür.

İşlemlerin sayısı $\leq$ kn { **k** : sınırlı sabit ,**n** : problemin büyüklüğü olarak tanımlanır}

Örnekler,

En küçük sayıyı bulma algoritması : n-1 O(n)

Hem en küçük hemde en büyüğü bulma : n-1 + n-2 = 2n-3 :O(n)

Sıralama yapan algoritma(En küçükten büyüğe): (n-1) +(n-2) +(n-3) +...= :O(n^2)

Algoritma	Zaman Karmaşıklığı	Çözülebilen En büyük problem			Örnek algoritma
		1 sn.	1 dk.	1 saat	
A1	n	1000	$6x10^4$	$3.6x10^6$	En küçüğü bulma
A2	nlogn	140	4893	$2x10^5$	Sıralama(Quicksort)
A3	n^2	31	244	1897	Geleneksel sıralama
A4	n^3	10	39	153	Matris Çarpımı
A5	2^n	9	15	21	Torba doldurma problemi

Burada ilginç nokta, mertebe arttıkça çözebildiğimiz problem sayısı çok çabuk düşer.

Örnek: İkili Arama algoritmasının karmaşıklık hesabının yapılması:

Çözüm : Basitlik için $a_1, a_2,,....., a_n$ listesinde n = 2^k eleman olduğunu varsayalım.(k > 0) Burada k =logn olacaktır.Eğer kümedeki elemanların sayısı 2'nin katı şeklinde değil ise, liste

2^{k+1} elemanlı daha büyük bir liste olacaktır(burada . $2^k < n < 2^{k+1}$ dir.)

Aranan sayı bulununcaya kadar, i ve j sayıları birbirine yaklaşır. İlk adımda liste 2^{k-1} e sınırlanır. İkinci adımda liste 2^{k-2} 'ye sınırlanır. En sonunda liste $2^1 = 2$ elemanlı olarak kalır. Listede tek eleman kalınca karşılaştırma başka eleman olmadığını gösterir işlem biter. İkili arama algoritmasını icra etmek için toplam 2k+2= 2logn+2 karşılaştırma yapılır Buradan ikili arama algoritmasının karmaşıklığının en kötü durumda O(logn) olduğu söylenebilir. Diğer bir karmaşıklık analizi ortalama durum analizidir. En kötü durum analizinden daha karmaşık olan bu analiz doğrusal arama algoritmasının karmaşıklık hesabında kullanılmıştır.

10.2Sonlu Durumlu Makinalar ve Turing Makinaları

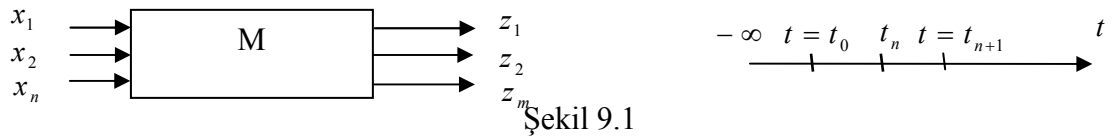
10.2.1 Sonlu durumlu Makina:

(a) Bir başlangıç durumu olan ve Sonlu sayıda duruma sahip $\{Q = q_0,q_1,.....q_n\}$ olan,

(b) : Giriş$\{G=x_1,x_2,.....,x_n\}$, ve Çıkış$\{Ç=z_1,z_2,.....,z_m\}$, olmak üzere sonlu Alfabe(A) vardır.

(c) : Bu parametreler ile bir geçiş fonksiyonu tanımlanır: { QxG → ÇxQ }

Z(t+1) çıkışı temelde x(t)'ye bağlıdır Q(t)'ye , o andaki durum veya makinaların başından geçen olaylar(History) denir.



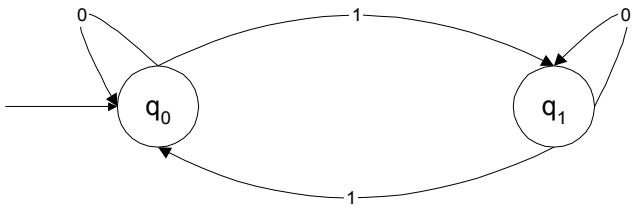
Şekil 9.1

10.2.2 Akseptör(Sonlu) : Bir sonlu akseptör aşağıdakilerden oluşur.

(a) Başlangıç durumu q_0, Son durumlar alt kümesi olmak üzere bir (sonlu) durum kümesi:
(b) : Bir A alfabesi(sonlu)
(c) : g : QxA → Q fonksiyonu
Örnek:
Örnek:

g/ç	q_0	q_1
0	q_0	q_1
1	q_1	q_0

Q ={ q_0, q_1 } ; A{0,1}

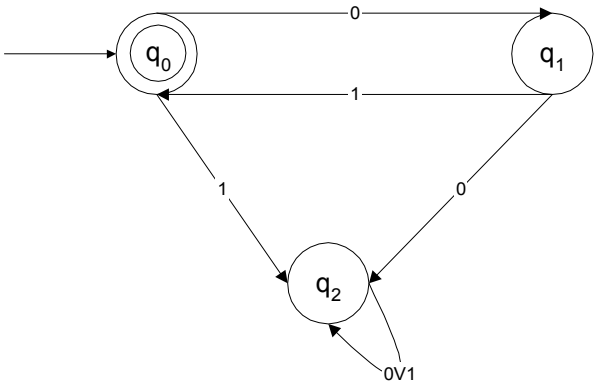


Şekil 9.2

Çıkışta bir işaret yok. q_1'i son durum alsak, tek sayıda 1 vererek bunu yine q_1'e getirmek mümkün. Bu akseptör tek sayıda bir bulunan bir katarı kabul eder.(10101110001'i kabul etmez, 6 ad .1 var)

Sonuç : Sonlu akseptör verilen bir katarın verilen bir gramere uygun olup olmadığını kontrol eder. Uygunluk son duruma erişip erişmeme ile anlaşılıyor.

Örnek : A{0,1} , Q{ q_0,q_1,q_2} , q_2 : dipsiz kuyu giren çıkamaz}

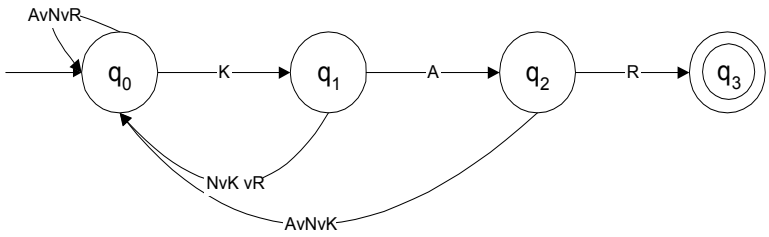


Şekil 9.3

Bu akseptör boş katar, 01,0101, 0101001 gibi katarları kabul eder. Türkçedeki bazı heceler sesli sessiz harflarden oluşur. Bunlar düzenlenebilir.

Örnek : a 0 ; at 01 ; yat 101 , dört 1011 gibi

Örnek : ANKARA'da KAR varmı? Akseptörü



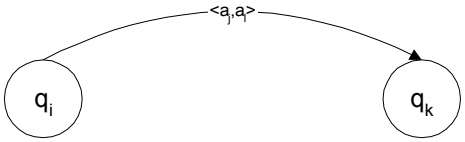
Şekil 9.4.

KAR bulduğu zaman son durumuna gelecektir. Buna Karakter uyuşturma{string matching} denilir. Uzun bir metnin içerisinde belirli bir harf dizisi varmı onu arıyoruz.

KAR Değilde başka bir şey aranırsa, mesela, aynı katar tekrarlanıyor, öyleki tekrardan sonra en bşa değil de daha ileri bir duruma geçilecek. Örnek KARAKAYA, Alt katarlar tekrarlanıyorsa akseptörü çizmek bir hayli zordur.

10.2.3 Sonlu Dönüştürücüler(Transduser) :

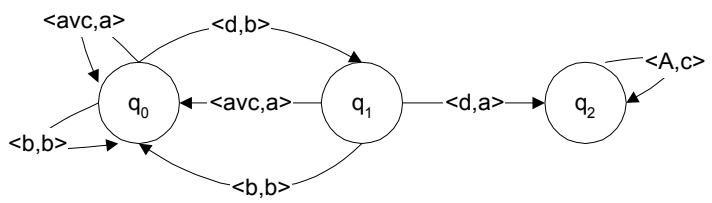
Bir Q kümesi (başı q_0)
Bir A alfabesi
g : QxA→ QxA
Bu makinalar yeni bir katarı alır, bundan yeni bir katar üretir
$q_i a_j \rightarrow q_k a_l$; $q_i \in Q$; $a_j \in A$



Şekil 9.5.

(q_i,a_j,q_k,a_l) dörtlüsü g fonksiyonunu tanımlar (q_i : durum , a_j , alfabe, q_k : donraki durum a_l: çıkış)

Örnek : A= {a,b,c,d} alfabesi üzerinde aşağıdaki dönüşüm işlemi yapılacaktır. Arka arkaya 2 d görülünceye kadar a ve b karakterleri aynı şekilde kopyalanacak, c'ler a'ya ; d'ler b'ye dönüştürülecektir. Arka arkaya gelen 2 d'den ikincisi a'ya dönüştürülecek. Daha sonra gelen karakterlerin yerine c koyulacaktır.



Şekil 9.6

Böylece dönüştürücünün durum geçiş diyagramı yukarıdaki gibi olacaktır. Bunlarda bellek yoktur. Eğer bellek eklenirse Turing makinaları elde edilir.

10.2.4 Turing Makinaları :

Bu makinalarda şerit şeklinde bellek vardır. Herbir bellek gözünde alfabenin sembollerinden biri olacaktır. Yine,

Bir Q kümesi (başı q_0)
Bir A alfabesi (b boşluk dahil)
g : QxA→ QxA {R,L} kümesi bu şeridi okuyup kafanın sağamı, yoksa solamı hareket ettiğini belirtiyor.
(q_i,a_j,q_k,a_l,Y) ile tanımlanır

q_i : Makinanın durumu

a_j : kafanın şeritten okuduğu sembol

q_k : Makinanın yeni durumu

a_l : Kafanın şerite yazdığı yeni karakter

Y : R veya L olarak sağa yada sola doğru kafanın hareketi(bir göz hareket edecek). Böyle bir bellek özelliği olan ilkel makine turing makinası olarak bilinir. Turing makinası programı belirli bir işlemi yapan 5'lilerden oluşur

Örnek: m,n tamsayıları birli sistemde şerit üzerinde temsil edilmiştir. Sıfırı belirtmek için m tamsayısı m+1 adet 1 ile temsil edilmiştir

Birli sistemde 1 ve 4'ü'i temsil etmek için aşağıdaki gösterilim kullanılır.

b	1	1	b	1	1	1	1	1	b		

Yazacağımız program m ile n'i toplayacaktır

b	1	1	1	b 1	1	1	1	1	1	1	B		

Burada kafanın konumunun nerede olduğu önemlidir. Kafa en soldaki 1'in üzerindedir

q_i	s_j	q_k	s_l	Y
0	1	0	1	R
0	b	1	1	L
1	1	1	1	L
1	b	2	b	R
2	1	3	b	R
3	1	4	b	R

Bunu yukarıdaki programla toplayabiliriz.

b	1	1	b	1	1	1	1	1	b		

Turing makinaları ile başka işlemler de yapılabilir.

Örnek : A = {0,1} ,başta ve sonda boşluk bulunsun.

b	0	1	1	0	0	1	b		

Bu sayıyı tek yada çift pariteli yapmak için gerekli karakteri en sağına ilave eden program.

b	0	1	1	0	0	1	b		

1'leri sayıp tek ise sona 1 koyar, çift sayıda 1 varsa boşluğu sıfır yapar

	b	b	0	1	1	0	0	1	b	b	

Başka bir örnekte ikili sistemdeki sayının değerini birli sistemde yazmaktır. Mesela 15'i ikili sistemde okuyup 16 tane bir koymak olabilir.

10.3Alıştırmalar

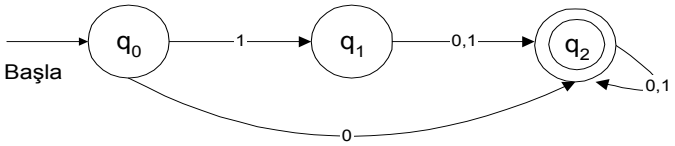
1. n uzunluğundaki bir listede bulunan sayılardan tamsayı(ondalık kısmı sıfır) olanların toplamını bulan bir algoritmayı psudokod ile yazınız.

2. Sadece atama deyimleri kullanarak x ve y değişkenlerinin değerlerini yer değiştiren algoritmayı psudokod ile yazınız.

3. Herbir işlemin 10^{-9} sn aldığı bir işlemde f(n) in aşağıdaki karmaşıklık değerleri olduğu algoritmalarda bir saniye içinde ne kadar büyüklükte problem çözülebileceğini hesaplayın.

   a)logn b)n c) n^2 d) 2^n

4. Aşağıdaki akseptör'ün hangi dizileri kabul ettiğini bulun..



5. {1,01,11} dizilerini kabul eden bir akseptör çizin.

Kaynaklar

1. Rowan Garnier, John Taylor, "Discrete mathematics for new technology ", Adam Hilger Publishing, 1992.
2. Sait Akkas, "Soyut matematik ", Gazi Üniversitesi, 1984.
3. Kenneth H .Rosen, éDiscrete Mathematics and its Applications", Mc Graw Hill ,1999.