BÖLÜM 1

1. ALGORİTMALAR

1.1. Algoritmanın Tanımı, Matematikteki Yeri ve Önemi

Endüstri ve hizmet sektörü organizasyonlarının karmaşıklığının artan bir yapıda olması, büyük ölçekli optimizasyon problemleri için çözümleri ve mevcut verimliliği devam ettirmede yeni alternatiflerin belirlenmesini gerektirir. Bu büyük ölçekli optimizasyon problemlerinin formülasyonu, kompleks matematik modellere yol açar ve bu modellerin çözümü yalnızca çok güçlü hesaplama kaynakları kullanılarak elde edilebilir.

Böyle bir matematik uygulamanın kesikli bir yapıda olması gerekir. Bundan dolayı, algoritma fikri bu tip uygulamalarda önemli bir rol oynar. Algoritmanın kesikli bir yapıya sahip uygulamalardaki rolü, matematiğin klasik branşlarında bir fonksiyon bilgisinin önemi gibidir. Matematikte yeri ve önemi böylesine net olan algoritmalara genel anlamda bir prosedür gözü ile bakılabilir. Bu anlamda, bir algoritma için yaygın olarak kullanılan tanımlardan bazıları şunlardır:

- Algoritmalar, problemleri çözmek için adım adım prosedürlerdir.
- Algoritma, bilgisayarda problemlerin bir sınıfını çözmek için bir metoddur.
- Algoritma, bir mekanik kural veya otomatik metod veya bazı matematiksel işlemlerin düzenlenmesi için programdır.
- Algoritma, soruların herhangi verilen bir sınıfına cevaplar bulmakta kullanılabilen bir hesaplama prosedürü (nümerik olması gerekmeyen) için etkili komutların kümesidir.
- Algoritma, açık olarak tanımlanmış olan ve herhangi bir bilgisayara icra edilen bir prosedürdür.

Algoritma için bu tanımları verdikten sonra, algoritmanın, matematikteki yeri ve önemini şöyle özetleyebiliriz:

Her algoritmanın bünyesinde aritmetik ve mantıki adımları bulundurması, tersine matematiksel çözüm isteyen ve analitik yapıda olmayan bütün problemlerin bir algoritma ile ifade edilmesi ve algoritmaların çalışma sürelerinin fonksiyonel bir yapıda olması, algoritmaların matematikteki yeri ve önemini belirtir .

1.2.Algoritmalar Sistemi

Bir algoritma, belirli bir problem için tatmin edici bir çözüme yakınsamayı sağlayan bir prosedürdür. Endüstrideki problemlerin bazıları çok kompleks olduğu için, Yöneylem Araştırmasının standart modelleri, bu tip problemler için uygun değildir.

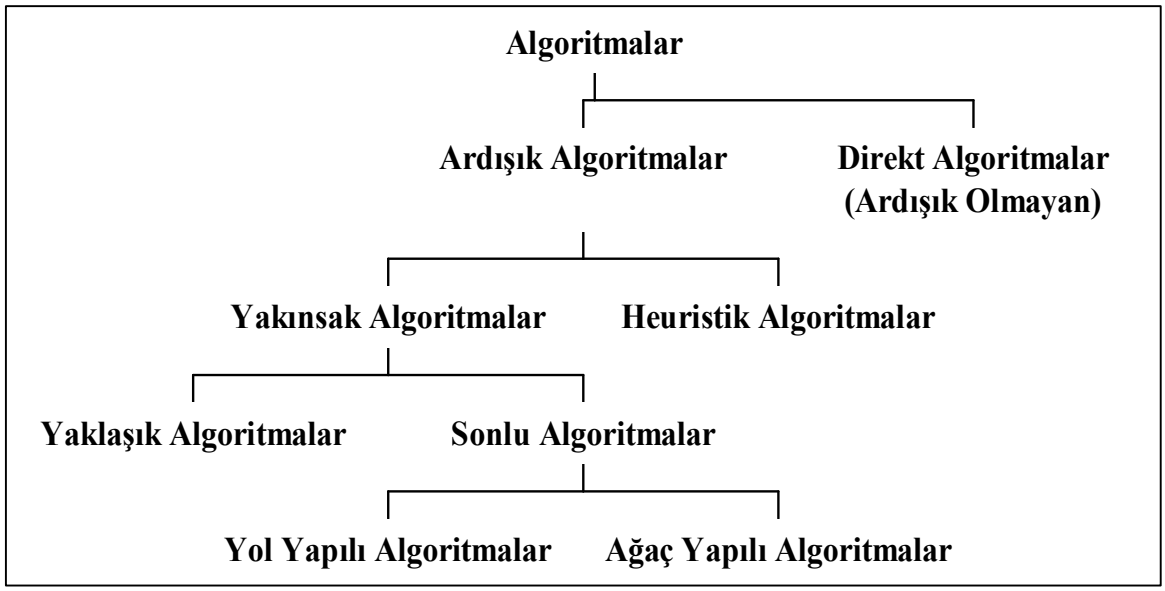
Bu durum;

- Problemi tarif etmek için gereken verinin çok fazla olduğu
- Problem için doğru bilgi toplamanın zor olduğu durumlarda ortaya çıkar.

Buna karşılık algoritmalar, matematik terimlerle kurulan bir problemin çözümü için prosedürlerdir. Değişik algoritmalar arasında bazı temel farklılıklar vardır.

1.2.1. Algoritma Tipleri

Algoritmalar, gerek yapıları, gerekse çalışma durumlarına göre farklılıklar gösterirler ve şekildeki gibi altı tipe ayrılırlar.



Şekil 1.1. Algoritma Sisteminin Ağaç Diyagramla Gösterimi

a) Direkt (Ardışık Olmayan) Algoritmalar

İterasyonlarda çalışmayan algoritmalar, direkt algoritmalar olarak adlandırılır. $y= ax^b$, $(a>0)$ polinomunun türevi $y' = abx^{b-1}$ bunun basit bir örneğidir.

b) Ardışık Algoritmalar

Belirli alt prosedürlerin pek çok kez tekrarlandığı algoritmalardır ve ardışık olarak çalışırlar. Algoritmaların çoğu ardışık olarak çalışır.

c) Yakınsak Algoritmalar

Aranılan çözüme doğru yakınsayan ardışık algoritmalardır.

d) Yaklaşık Algoritmalar

Bazı yakınsak algoritmalar kesin çözümü elde edemezler, fakat bu çözüme yaklaşık bir değeri kesin çözüm alırlar. Yaklaşık algoritmalar sonlu değillerdir; fakat herbir ileri iterasyon onları kesin çözüme biraz daha yaklaştırır. Yaklaşık algoritmalara Değişken Kesen Metodu, Arama Teknikleri vb. çok bilinen birkaç örnek verilebilir.

e) Sonlu Algoritmalar

Bu algoritmalar, iterasyonların sonlu bir sayısında kesin çözümü garanti eden yakınsak algoritmalardır ve kendi arasında yol yapılı ve ağaç yapılı olmak üzere ikiye ayrılırlar.

1. Yol Yapılı Algoritmalar: Sonlu algoritmaların pekçoğu yol yapısına sahiptir; bu yol yapısında bir iterasyon bir önceki iterasyonu iterasyon dizilerinde farklı dallar üretmeksizin takip eder.

Böyle algoritmaların örnekleri, Tamsayı Programlamada Kesme Düzlem Algoritmalarının pekçoğu, Şebekelerde Maksimum Akış Algoritmaları, pekçok En Kısa Yol Algoritmaları ve Lineer Programlamanın Simpleks Algoritmaları ve herhangi pivot tekniği ile matris tersleridir.

2. Ağaç Yapılı Algoritmalar: Diğer sonlu algoritmalarda iterasyon dizileri, pekçok paralel dalları içeren bir ağaç şeklindedir. Bir çok ağaç arama algoritmaları bu sınıfa aittir. Bu arama algoritmalarının bazıları, Dinamik Programlama, Dal ve Sınır, Sınırlı Sayım, Kapalı Sayım, Dal ve Çıkarma, Dal ve Atma vb. olarak sıralanabilir.

Yaklaşık algoritmalar bu yol yapısına doğru yönelir. Diğer yandan heuristikler ise bir yol yapısı ile son bulabilen indirgenmiş bir ağaç yapısı olarak gözönüne alınabilir.

1.3. Algoritmanın Yapısı

Bir problemi çözmek için adım adım uygulanacak bir prosedür olarak tanımlanan algoritmanın tipik adımları;

i. Atama adımları, (Bir değişkene bazı değerlerin atanması gibi)

ii. Aritmetik adımlar, (Toplama, bölme, çıkarma, çarpma gibi)

iii. Mantıki adımlardır. (İki sayının karşılaştırılması gibi)

Genel yapısı bu şekilde kısaca özetlenen algoritmalar, belirli bir hata kabülü içinde yaklaşık cevap verirler; uygun programlama dili seçildiğinde genellikle hızlı çalışırlar.

Bir algoritmanın sahip olduğu adımların sayısı (ki bu, algoritmanın gereksinimi olan zamanı büyük ölçüde belirler) problemin bir örneğinden diğerine farklılık gösterir. Problemin bazı "iyi" örnekleri, bu algoritma yardımıyla hızlı olarak çözülebilirse de, problemin bazı "kötü" örneklerini bu algoritma yardımıyla çözmek çok uzun zaman alabilir. Bu, ilgili algoritmanın performansına bağlı bir faktördür. Bu performansın nasıl ölçüleceği Bölüm 1.6 da incelenecektir.

1.4.Algoritmaların Dili

Kodlama :

1) Procedure = Bir algoritmanın kodlanmasına başlanan ilk ifadedir .Bu ifadede algoritmanın ad Örnek: Procedure max(L = list of integers)

Burada algoritmanın adı max iken tamsayıların L listesinin maksimumunu bulur.

2) Assignments=Atamalar ve ifadelerin diğer tipleri

Assigments ifadesi değişkenlere değer atamada kullanılır .Bu ifadede sol taraf değerin adını alırken sağ taraf ise prosedürlerle tanımlanan fonksiyonları , değerleri atanan değişkenleri , sabitleri içeren bir ifade yada deyimdir .Sağ tarafta ayrıca aritmetik işlemlerin herhangi biride bulunabilir.

: = atamalar için semboldür.

Böylece ;

Variable : = expression

Max: = a (a'nın değeri max değişkenine atanır.)

Ayrıca, x : = Largest integer in the list L şeklinde bir ifade de kullanılır.

x ' i L'de en büyük tamsayı olarak atar.

Güncel bir programlama diline bu ifadeyi dönüştürmek için birden fazla ifade kullanmalıyız . Bunun için ; interchange a and b kullanılır. Karmaşık prosedürler için ifade blokları kullanılır . Bu begin ile başlar ve end ile sona erer.

Begin

Statement 1

Statement 2

.....

Statement n

end.

Şartlı Yapılar

1) **if condition then statement**

veya

if condition then

begin

blok of statements

end

Eğer durum doğru ise verilen ifade gerçekleştirilir

2) if condition then statement1 else statement 2

Genel form: İf condition 1 then statement 1 else if condition 2 then statement 2 else if condition 3 then Statement 3……..

Else if condition n then statement n else statement n+1

Eğer durum 1 doğruysa ifade 1 gerçekleştirilir v e programdan çıkılır .Eğer, durum1 yanlışsa program durum2 'nin doğruluğunu kontrol eder .Eğer durum2 doğruysa ifade2

gerçekleştirilir . Böylece devam edilir. Sonuç olarak ,eğer durum1 ile durum – n'nin hiçbiri doğru değilse (n+1).ifade yerine getirilir.

Döngü Yapıları

1) for

2) while

1) **for variable : =** initial value to final value

statement

veya

for variable : = initial value to final value

begin

block of statements

end.

Eğer başlangıç değer sonuç değerden küçük yada eşitse değişken olarak başlangıç değer atanır ve sonuç değer değişken atanana kadar bu döngü gerçekleştirilir.Eğer, başlangıç değer sonuç değeri aşarsa döngü olmaz .Örneğin;

sum : = 0

for i : = 1 to n

sum : = sum+i

2) while condition statement

veya while condition

begin

block of statements

end şeklindedir. Burada durum doğruysa ifade gerçekleştirilir ve durum yanlış olana kadar bu döngü sürdürülür .

1.5. Çeşitli Algoritma Örnekleri

1) Algoritma: Sonlu bir dizideki en büyük elemanı bulan Algoritma

```
procedure max (a1 , a2 , ..... , an : integers )
max : = a1
for i : = 2 to n
  if max < ai then max : = ai
{ max en büyük eleman }
```

2) Algoritma: Euclidean Algoritma

```
procedure gcd(a , b : positive integers )
x : = a
y : = b
while y ≠ 0
begin
  r : = x mod y
  x : = y
  y : = r
end { gcd ( a ,b ) is x }
```

3) Algoritma : Constructing Base *b* Expansions

```
procedure base b expansion (n : positive integer )
q : = n
k: = 0
while q ≠ 0
begin
  ak : = q mod k
  q  : = [ q / b ]
  k  : = k + 1
end { the base b expansion of n is ( ak-1 ..... a1 a0)b }
```

4) Algoritma:Tamsayıların toplamı

```
procedure add(a ,b : positive integer )
{ the binary expansions of a and b are ( an-1 , an-2 , ..... a1a0 )2
        and (bn-1 , bn-2 ..... b1b0 )2 , respectively }
c : = 0
for j : = 0 to n - 1
begin
  d  : =  [ ( aj + bj + c) / 2 ]
```

```
  s_j : =  a_j + b_j + c – 2d
  c  : =  d
end
s_n  : =  c
{ the binary expansion of the sum is ( s_n s_{n-1} .....  s_0 )_2 }
```

5) Algoritma:Tamsayıların çarpımı

```
procedure multiply(a ,b : positive integer )
{ the binary expansions of a and b are ( a_{n-1} , a_{n-2} , ..... a_1a_0 )_2
          and (b_{n-1} , b_{n-2} ..... b_1b_0 )_2 , respectively }
for j : = 0 to n - 1
begin
  if b_j : =  1 then c_j : = a shifted  j places
  else c_j  : = 0
end
{ c_0 , c_1 , ..... , c_{n-1} are the partial products }
p : = 0
{ p is the value of a.b }
```

6) Algoritma: Matris çarpımı

```
procedure matrix multiplication(A ,B : matrices )
for i : = 1 to m
begin
   for j : = 1 to n
   begin
      c_ij : = 0
      for q : 1 to k
         c_ij : = c_ij + a_iq b_qj
   end
end { C = [ c_ij ] is the product of A and B }
```

7) Algoritma: Boolean çarpımı

```
procedure Boolean Product (A ,B : ziro-one matrices )
for i : = 1 to m
begin
   for j : = 1 to n
   begin
      c_ij : = 0
      for q : 1 to k
         c_ij : = c_ij ∨ a_iq ∧ b_qj
   end
end { C = [ c_ij ] is the Boolean product of A and B }
```

1.5. Algoritmaların Karmaşıklığı

Bir algoritmanın karmaşıklığı, bu algoritmanın çalışma zamanı ile ilgili bir kavramdır. Bir algoritmanın çalışma zamanı olarak da isimlendirilen algoritmanın gereksinimi olan zaman, verinin hem yapısına hem de boyutuna bağlıdır.

Büyük problemler çok daha fazla çözüm zamanı gerektirirken; verideki farklılıklara bağlı olarak aynı boyuttaki değişik problemler belirgin olarak farklı çözüm zamanları gerektirirler. Bir algoritmanın karmaşıklık fonksiyonu, problem

uzayı boyutunun bir fonksiyonudur ve verilen boyutun zamanının herhangi bir problem örneğini çözmek için algoritma tarafından ihtiyaç duyulan en geniş zamanı belirtir. Bir başka deyişle, zaman karmaşıklığı fonksiyonu, problemin boyutu arttıkça çözüm zamanının gelişme hızını ölçer.

Burada dikkat edilmesi gereken konu, zaman karmaşıklığı fonksiyonunun, verilen bir problem uzayı boyutunda herhangi bir problem örneğini çözmek için gereken çalışma zamanını, çözüm için gerekli en büyük çalışma zamanını ölçerek hesapladığıdır. Karmaşıklık fonksiyonu, algoritmanın performansının ölçümünde, problemin giriş verisinin uygun ölçülmesine bağlı olarak bir performans garantisi sağlar.

Bu nedenle zaman karmaşıklığı fonksiyonu aynı zamanda bir algoritmanın en kötü durum karmaşıklığı veya sadece karmaşıklığı olarak bilinir. Bir algoritmanın en kötü durum analizinde bu detaylı bir şekilde anlatılmıştır .

1.6. Polinom ve Üstel Zaman Algoritmaları

Bir algoritmanın etkinliği, en kötü durumda algoritma için gereken adımların sayısını sayarak ve bunu problem uzayı boyutu olan n'nin bir fonksiyonu olarak, n büyüdükçe çalışma zamanının davranışını saptamaktır. Algoritma için gereken adımların sayısı n'nin bir polinom fonksiyonu olduğunda, "*O*" sembolu kullanılır ve algoritmanın çalışma zamanı $O(n^k)$'olarak gösterilir. Burada, *k*, büyük *n*'ler için diğer terimler anlamsız olduğundan, polinomun en büyük derecesidir.

Bir başka deyişle, eğer bir algoritmanın zaman karmaşıklığı fonksiyonu $O(n^k),(k \in N)$ (fonksiyon *k*.nci derecedendir) ise bu algoritmaya **Polinom Zaman Algoritması** denir ve **algoritma $O(n^k)$**'dır denir. Örneğin, $k=1$ ise algoritmaya *Lineer*, $k=2$ ise algoritmaya *karesel* algoritma denir.

Giriş verisi bir polinom fonksiyon ile sınırlandırılan bir çalışma zamanına sahip olan bir algoritma genellikle *"iyi"* olarak adlandırılır. Bu algoritma ile çözülen problemlere de *"kolay"* denir. Diğer yandan, çalışma zamanı bazı $c>1$ için en az c^n kadar hızla artan bir algoritmaya "**Üstel Zaman Algoritması**" denir.

Nlogn gibi ne polinom ne de üstel bir fonksiyon ile sınırlandırılmamış yarı üstel fonksiyonlar olmasına karşılık en kötü durum kriteri bir polinom fonksiyon ile sınırlandırılmamış herhangi bir algoritmaya kolaylık için "*Üstel Zaman Algoritması*" denir. **$O(2n), O(n!), O(n^{\log n})$** üstel zaman algoritmalarına örnek olarak gösterilebilir. Polinom ve üstel zaman algoritmaları arasındaki farklılık, saniyede 10^6 işlem

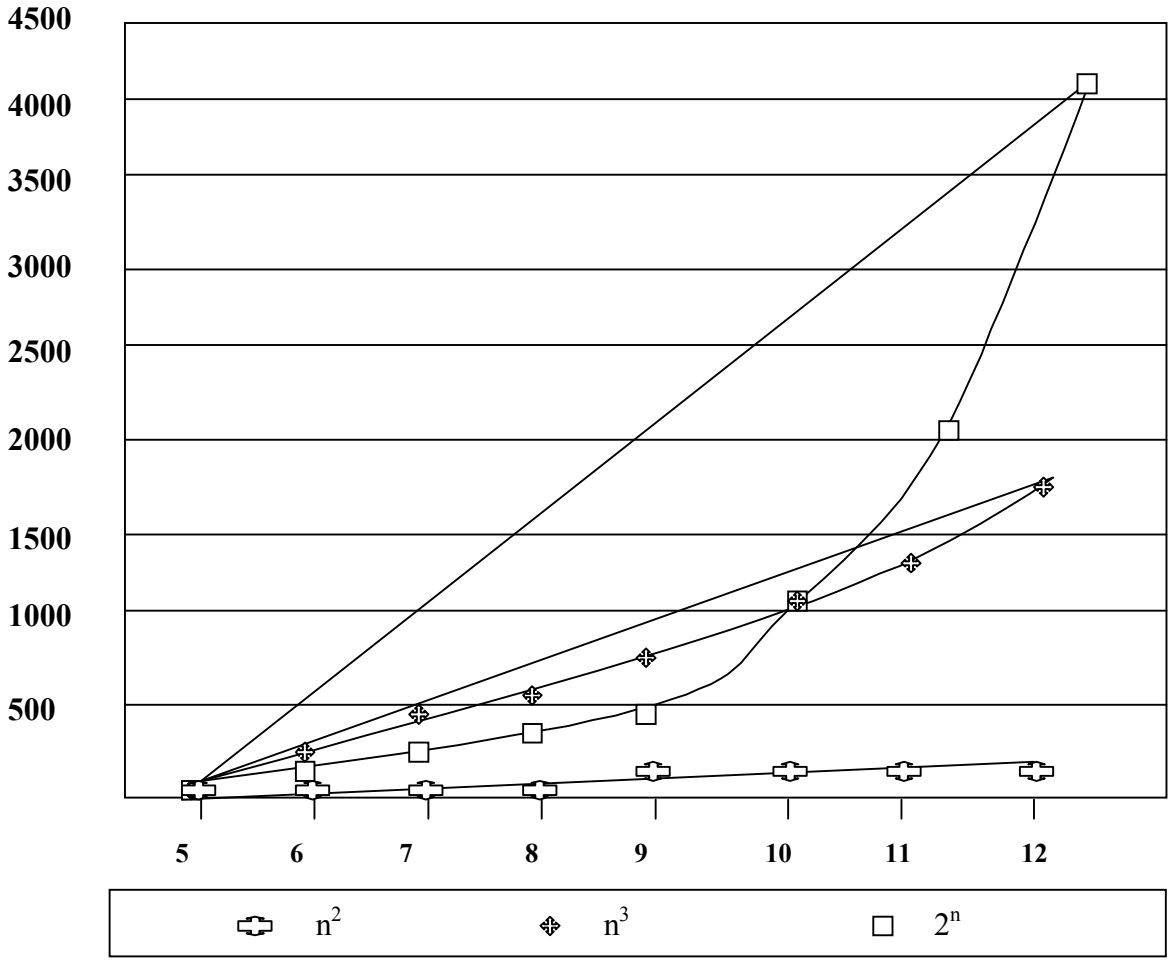
düzenleme kapasitesine sahip bir bilgisayarın farklı algoritmaları icra etme zamanı ile hesaplama gücü arttığında en büyük çözülebilir problemin problem uzayı boyutunun karşılaştırılmasıyla ortaya çıkar.

Aşağıdaki tabloda bazı polinom ve üstel fonksiyonların problem uzayının boyutunun *n*'ye bağlı büyüme oranları verilmiştir.

Tablo 1.1. Bazı üstel ve polinom fonksiyonların büyüme oranları

n	log n	$n^{0.5}$	N^2	n^3	2^n	n!
10	332	316	10^2	10^3	10^3	$3,6 \times 10^6$
100	664	10	10^4	10^6	$1,27 \times 10^{30}$	$9,33 \times 10^{157}$
1000	997	31.62	10^6	10^9	$1,07 \times 10^{301}$	$4,02 \times 10^{2,567}$
10000	1329	100.00	10^8	10^{12}	$0,99 \times 10^{3010}$	$2,85 \times 10^{35,659}$

Aşağıdaki şekilde de problem uzayının boyutu olan n'ye göre f(n) hesaplama adımlarının sayısı, $O(n^2), O(n^3)$ ve $O(2^n)$ algoritmalar için gösterilmiştir.



Şekil 1.2. Problem uzayının boyutu n'ye göre bir algoritmayı gerçekleştirmede f(n) hesaplama adımlarının sayısı.

Gerek Tablo 1.1 gerekse Şekil 1.1 den de anlaşılacağı gibi genelde polinom zaman algoritmaları üstel zaman algoritmalarına göre daha küçük derecelere sahip olduklarından daha iyi düzenlenirler.

2.BÖLÜM. Algoritmaların Performans Ölçümü

2. Algoritmaların Performans Ölçümü

Bir algoritmanın performansı, o algoritmanın performansının nasıl ölçüleceği sorusundan doğar. Bu sorun, bir problemin çözümü için ortaya konulan algoritmalar arasından "en iyi" algoritmayı seçme ile ortaya çıkar. Bir algoritmanın performans ölçümü için üç temel yaklaşım yaygın olarak kullanılır: Bunlar *deneysel analiz, olasılık (ortalama durum) analizi* ve *en kötü durum analizidir*. Şimdi bu analizleri kısaca açıklayalım:

2.1. Deneysel Analiz

Deneysel analiz, örnek problemlerde denenmiş bir algoritmada hesaplama deneyiminde dayanır. Bu analizin amacı, pratikte algoritmanın nasıl davrandığını tahmin etmektir. Bu analizde, algoritma için bir bilgisayar programı yazılır ve problem örneklerinin bazı sınıfları üzerinde programın performansı test edilir .

Bu nedenle deneysel analiz, araştırmacı ve uygulamacıların en çok güvendiği analizlerden biridir ve bilimsel yaklaşımdan çok, uygulamaya yöneliktir .

Deneysel analizin başlıca dezavantajları şunlardır:

i. Bir algoritmanın performansının, programı yazan programcının tekniği kadar kullanılan bilgisayara, derleyiciye ve programlama diline bağlı olması.

ii. Bu analizin çok zaman alması ve düzenlenmesinin pahalıya malolması

iii. Algoritmaların karşılaştırılması; farklı algoritmaların problem örneklerinin farklı sınırlarını daha iyi düzenlemesi ve farklı deneysel çalışmaların birbirini tutmayan sonuçlar vermesi anlamında sıkça kesinlik taşımamasıdır. Bu analizin başlıca avantajı, güncel problemler için kesin ve geçerli olmasıdır .

2.2. Olasılık (Ortalama Durum) Analizi

Bu analiz son zamanlarda yoğun bir şekilde kullanılmaya başlanmıştır. Bu analizin amacı, algoritmanın alması beklenen adımlarının sayısını tahmin etmektir. Olasılık analizinde, problem örnekleri için bir olasılık dağılımı seçilir ve algoritma için beklenen asimptotik çalışma zamanlarını üreten istatistik analiz kullanılır .

Olasılık analizinin başlıca dezavantajları şunlardır:

i. Belirli analizin, problem örneklerini temsil etmek için seçilmiş olasılık dağılımına kesin olarak bağlı olması ve olasılık dağılımındaki farklı seçimlerin algoritmaların yapısından kaynaklanan avantajlar nedeni ile ilgili farklı değerlendirmelere yol açabilmesi.

ii. Pratikte karşılaşılan problemler için uygun olasılık dağılımlarını belirlemenin genellikle zor olması.

iii. Olasılık analizinin, algoritmanın en basit tipini değerlendirmek için bile oldukça yoğun matematik alt yapı gerektirmesi ve bu analizin tipik olarak çok kompleks algoritmalar için başarılmasının oldukça zor olması.

Bir algoritmanın performansının önceden tahmini, o algoritmanın olasılık analizine dayanmasına rağmen, analistin problem örneklerinin büyük bir çoğunluğunu çözmesini gerektirmesi ve sonuçların dağılımı hakkında bilgi sağlamaması, bu analizin rağbet görmemesine yol açar.

2.3. En Kötü Durum Analizi

En kötü durum analizi, özel bir *H* heuristiği ile bir *P* probleminin örneklerine uygulandığında ortaya çıkan optimalden sapma ile ilgili analizdir. Bu analiz, verilen bir algoritmanın herhangi bir problem örneği üzerinde alabileceği adımların sayısına bir üst sınır getirir.

En kötü durum analizinde diğer iki analizde bulunan dezavantajların çoğu yoktur. Bu analizin iki dezavantajı, bir algoritmanın performansını belirlemek için pratikte son derece nadiren ortaya çıkan anlamsız örneklere izin vermesi, ve bir algoritmanın en kötü durum ortalama performansının önceden bilinmemesidir. En kötü durum analizi, hesaplama çevresinden bağımsızdır ve düzenlenmesi diğer analizlere nisbeten kolaydır. Ayrıca bu analiz, bir algoritmanın adımları (çalışma zamanı) üzerinde bir üst sınır sağlar.

Bu nedenlerden dolayı bir algoritmanın performans ölçümü için bu üç analizden bilimsel literatürde en popüler olanı en kötü durum analizidir. Performans garantisi, en kötü durum oranı olan *r* ile ifade edilir. Buna göre, bazı $r>1$ için $C_h(I)$ ve $C(I)$ sırasıyla bir $I \in P$ minimizasyon problemi örneğinin heuristiği ve optimal çözümü olmak üzere;

$$C_h(I) = r \,.\, C(I) \qquad (\exists\, r>1) \qquad \textbf{(1)}$$

ile ifade edilir. Ayrıca heuristiğin performansı en kötü durum bağıl hatası $\varepsilon=r-1$ ile degerlendirilir. ε ve r ikilisi çok kullanılır ve duruma göre bunlardan önemli olanı seçilir [4].

2.4. "O", "Ω", "Φ" Sembolleri

O, Ω, Φ sembolleri, araştırmacıların algoritmaların analizinde kullandığı sembollerdir. Şimdi sırasıyla bunları tanıyalım:

"O" : Eğer c ve n_0 sayıları için, bir algoritmanın gereksinimi olan zaman en fazla her $n \geq n_0$ için $c.f(n)$ ise bu algoritma $O(f(n))$ çalışma zamanına sahiptir denir.

Yani, *O* sembolü algoritmanın performansı üzerinde bir üst sınırı belirtir. Problem uzayı boyutu olan n'nin yeterince büyük değerleri için algoritmanın çalışma zamanında bir üst sınır belirlendiğinde $O(f(n))$ karmaşıklık ölçümü çalışma zamanının asimptotik bir büyüme oranına sahip olmasına yol açar .

"Ω" : Ω sembolü, algoritmanın çalışma zamanı üzerinde bir alt sınırı belirtir. Eğer bazı c' ve n_0 sayıları ve her $n \geq n_0$ için bir algoritmanın çalışma zamanı bazı problem örneklerinde en az $c'.f(n)$ ise bu algoritma $\Omega(f(n))$ olarak isimlendirilir. *"O"* ve *"Ω"* sembollerini kısaca şu şekilde özetlenebilir :

Eğer bir algoritmanın çalışma zamanı, $O(f(n))$ ise sabit bir c sayısı için problemin herbir örneği en çok $c.f(n)$ zamanında çalışır. Benzer şekilde, eğer bir algoritma $\Omega(f(n))$ zamanında çalışırsa, problemin bazı örnekleri c' bir sabit olmak üzere en az $c'.f(n)$ zamanında çalışır.

"Φ" : *Φ* sembolu bir algoritmanın performansı üzerinde hem alt hem de üst sınırı belirtir. Yani eğer bir algoritma hem $O(f(n))$ hem de $\Omega(f(n))$ ise bu algoritma $\Phi(f(n))$ olarak isimlendirilir.

Bir algoritmanın performansının alt veya üst sınırı algoritmanın çalışma zamanında bir üst sınır veya bir alt sınır belirleme anlamındadır .

2.5.Fonksiyonların Büyüme Oranları

Bu bölümde algoritmaların performans ölçüsünde bize yardımcı olacak fonksiyonların büyüme oranları, modüler aritmetik ve buna ait temel kavramlar incelenmiştir.Önceki bölümde kısaca bahsetdiğimiz "*O*" notasyonu fonksiyonlarının gelişiminde en yaygın kullanılan notasyon oldugundan burada bu notasyonu daha iyi anlamak için tanım ve örnekleri ele alınacaktır. "*O*" sembolü ilk olarak Alman

matematikçi Paul Gustav Heinrich Bachmann (1837-120) tarafından 1892 de sayı teorisi üzerine önemli bir kitapta kullanılmış olmasına rağmen çalışmasının her aşamasında bu sembolü kullanan Edmund Landau (1877-138) sayesinde "*O*"sembolü "Landau Sembolü" olarakda adlandırılır.

Tanım 1.1: f ve g fonksiyonları reel sayılardan reel sayılara veya tam sayılardan reel sayılara tanımlı iki fonksiyon olsun. *C* ve *k* sabitleri için ;

$$|f(x)| \leq C\,|g(x)| \qquad (x > k)$$

sağlanıyorsa ; $\boldsymbol{f(x) = O\,(g\,(x))}$ denir.

Örnek 1.1 : $f(x) = x^2 + 2x + 1$ için $O(x^2)$ olduğunu gösterelim.

Çözüm 1.1 : $x > 1$ için ;

$0 \leq x^2 + 2x + 1 \leq x^2 + 2x^2 + x^2 = 4x^2$ olduğundan $C = 4$ ve $k = 1$ için $f(x) = O(x^2)$ olur.

$x > 2$ için ;

$2x \leq x^2$ olduğundan

$0 \leq x^2 + 2x + 1 \leq x^2 + x^2 + x^2 = 3x^2$ burada da $C = 3$ ve $k = 2$ için $f(x) = O(x^2)$ olduğu görülür.

Böylece , $f(x) = x^2 + 2x + 1$ ve $g(x) = x^2$ için $f(x) = O(g(x))$ dir.

Genelleyecek olursak geçişme özelliğinden aşağıdaki sonucu yazabiliriz.

$f(x) = O(g(x))$ ve x' in yeterince büyük değerleri için $h(x)$ fonksiyonu $g(x)$'den daha büyük mutlak değerlere sahip olsun.

Bu durumda ;

$|f(x)| \leq C\,|g(x)|$ $\qquad x > k$,

ve eğer bütün $x > k$ 'lar için $|h(x)| > |g(x)|$ oluyorsa;

$|f(x)| \leq C\,|h(x)|$ $\qquad x > k$ olur.Bu nedenle ;

$f(x) \leq O(h(x))$ elde edilir.

2.5.1.Fonksiyonların Kombinasyonunun Büyüme Oranı

Algoritmaların çoğu iki veya daha fazla alt prosedürden oluşur.Böyle bir algoritmayı kullanarak belirli ölçüdeki bir problemi çözmek için bilgisayar tarafından kullanılan adımların sayısı bu alt prosedürler tarafından kullanılan adımların sayıları toplamına eşittir. Kullanılan adımların sayısının tahmini için 'O' notasyonu kullanılır.Bunun için iki fonksiyonun toplamı ve çarpımı için 'O' notasyonu nasıl bulunur onu inceleyelim.

$f_1(x) = O(g_1(x))$ ve $f_2(x) = O(g_2(x))$ olsun. C_1, C_2 , k_1 ve k_2 sabitleri için;

$|f_1(x)| \le C_1 |g_1(x)| \qquad x > k_1$,

$|f_2(x)| \le C_2 |g_2(x)| \qquad x > k_2$,

olur.Bu iki fonksiyonun toplamı ise;

$|(f_1 + f_2)(x)| = |f_1(x) + f_2(x)|$

$\le |f_1(x) + f_2(x)|$ (üçgen eşitsizliği $|a + b| \le |a| + |b|$)

$x > k_1$, $x > k_2$ durumu için;

$|f_1(x)| + |f_2(x)| < C_1 |g_1(x)| + C_2 |g_2(x)|$

$\le C_1 |g(x)| + C_2 |g(x)|$

$= (C_1 + C_2) |g(x)|$

$= C |g(x)|$,

burada, $C = C_1 + C_2$ ve $g(x) = \max(|g_1(x)|, |g_2(x)|)$ alındığından

$k=\max(k_1,k_2)$ iken

$$|(f_1 + f_2)(x)| \le C |g(x)| \qquad x > k$$

olarak elde edilir.

Teorem.1.1.

$f_1(x) = O(g_1(x))$ ve $f_2(x) = O(g_2(x))$ olsun.Bu durumda;

$(f_1 + f_2)(x) = O(\max(g_1(x)), g_2(x)))$ .Burada, $\max(g(x), g(x)) = g(x)$ olduğundan

$(f_1 + f_2)(x)$ yine $O(g(x))$'dir.

Sonuç.1.1. $f_1(x) = O(g(x))$ ve $f_2(x) = O(g(x))$ olduğunu varsayalım.Böylece;

$(f_1 + f_2)(x) = O(g(x))$'dir.

Teorem.1.2: $f_1(x) = O(g_1(x))$ ve $f_2(x) = O(g_2(x))$ olsun.Buna göre; $x > k_1$, $x > k_2$ ve

$C= C_1 C_2$ olmak üzere;

$|(f_1 f_2)(x)| = |(f_1(x)| . |(f_2(x)|$

$\le C_1 |g_1(x)| \; C_2 |g_2(x)|$

$\le C_1 C_2 |(g_1 g_2)(x)|$

$\le C |(g_1 g_2)(x)|$

$|(f_1 f_2)(x)| \le C |(g_1(x) g_2(x)|$ $\qquad x > k$

Teorem.1.3: $f_1(x) = O(g_1(x))$ ve $f_2(x) = O(g_2(x))$ olsun. Buna göre;

$(f_1 f_2)(x) = O(g_1(x)(g_2(x))$ elde edilir.

2.5.2. Asimptotik Notasyonlar

i) (asimptotik üst sınır) Her $n \geq n_0$ için. $0 \leq f(n) \leq c.g(n)$ olacak şekilde c pozitif sabiti ile n_0 pozitif tamsayısı bulunursa $f(n) = O(g(n))$'dir.

ii) (asimptotik alt sınır) Her $n \geq n_0$ için. $0 \leq c.g(n)) \leq f(n)$ olacak şekilde c pozitif sabiti ile n_0 pozitif tamsayısı bulunursa $f(n) = \Omega(g(n))$'dir.

iii) (asimptotik sıkı sınır) Her $n \geq n_0$ için. $c_1.g(n) \leq f(n) \leq c_2.g(n))$ olacak şekilde c_1 ve c_2 pozitif sabitleri ile n_0 pozitif tamsayısı bulunursa $f(n) = \theta(g(n))$ 'dir.

iv) (O-notasyonu) Her $n \geq n_0$ için. $0 \leq f(n) \leq cg(n)$ olacak şekilde c pozitif sabiti ile n_0 pozitif tamsayısı bulunursa $f(n) = O(g(n))$'dir

2.5.2.1. Asimptotik Notasyonların Özellikleri

$f(n)$,$g(n)$,$h(n)$, and $l(n)$, herhangi fonksiyonlar olsun.Buna göre, aşağıdaki özellikler doğrudur.

i) Eğer, $g(n) = \Omega(f(n))$ ise $f(n) = O(g(n))$.

ii) Eğer, $f(n) = O(g(n))$ ve $f(n) = \Omega(g(n))$.ise $f(n) = \theta(g(n))$

iii) Eğer, $f(n) = O(h(n))$ ve $g(n) = O(h(n))$ iken $(f + g)(n) = O(h(n))$

iv) Eğer, $f(n) = O(h(n))$ ve $g(n) = O(l(n))$ iken $(f . g)(n) = O(h(n)l(n))$

v) (Yansıma) $f(n) = O(f(n))$.

vi) (Geçişkenlik) Eğer, $f(n) = O(g(n))$ ve $g(n) = O(h(n))$ iken $f(n) = O(h(n))$

2.5.2.2.Bazı Fonksiyonların Karmaşıklık Değerleri

i) (Polinom fonksiyonlar) Eğer, $f(n)$, k. dereceden bir polinom fonksiyon ise $\boldsymbol{f(n) = \theta(n^k)}$.

ii) Herhangi bir $c > 0$ sabiti için , $\boldsymbol{\log_c n = \theta(\lg n)}$.

iii)(Stirling's formülü) $n \geq 1$ tamsayıları için,

$$\sqrt{2\pi n}\left(\frac{n}{e}\right)^n \leq n! \leq \sqrt{2\pi n}\left(\frac{n}{e}\right)^{n+(1/12n)}$$

ve böylece; $\boldsymbol{n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n (1 + \theta(1/n))}$. Aynı zamanda, $\boldsymbol{O(n^n)}$ **and** $\boldsymbol{n! = \Omega(2^n)}$.

iv) $\boldsymbol{\lg(n!) = \theta(n \lg n)}$.

Örnek.1.2 : ***e*** ve ***c*** $0 < e < 1 < c$ özelliğindeki keyfi sabitler olsunlar.Bu durumda,

$$1 < \ln\ln n < \ln n < \exp(\sqrt{\ln n \ln\ln n}) < n^e < n^c < n^{\ln n} < c^n < n^n < c^c .$$

2.6.Algoritmaların Karmaşıklık Hesabı (Performans Analizi)

Bu bölümde Arama,Sıralama ve İndirgeme algoritmaları ele alınarak bunların performans analizi incelenmiştir.Algoritmanın performans analizi, algoritmanın karmaşıklığı ile ilgili bir kavram olduğundan, bu konu ilk olarak açıklanmıştır.

2.6.1.Algoritmaların Karmaşıklığı

Bir algoritmanın bir problem için ne zaman tatminkar bir çözüm sağladığı önemli bir sorudur. Bu sorunun cevabı şudur. İlk olarak, bir algoritma her zaman doğru cevaplar ortaya koymalıdır. İkinci olarak, etkili ve etkin çalışmalıdır. Bu bölümde algoritmaların etkinliği konusu üzerinde duracağız.

Bir algoritmanın etkinliği nasıl incelenir. Etkinliğin ölçümlerinden birisi, giriş değerleri belirtildiğinde, bu algoritmayı kullanarak belirli problemi çözmek için bilgisayar tarafından kullanılan zamandır. İkinci bir ölçüm, giriş değerleri özel olarak belirtildiğinde algoritmayı tamamlamak için gereken bilgisayar hafıza miktarının ne kadar olduğudur.

Böyle sorunlar **algoritmaların hesaplama karmaşıklığı** ile ilgilidir**.** Boyutu belirli bir problemi çözmek için gereken zamanın analizi, algoritmanın zaman karmaşıklığını içerir. Gerekli olan bilgisayar hafızasının analizi de algoritmanın uzay karmaşıklığını içerir. Bir algoritmanın zaman ve uzay karmaşıklığının ele alınması algoritmaları tamamlayabilmekte önemlidir. Bir algoritmanın bir cevabı bir mikrosaniye, bir dakika yada bir milyar yılda üretip üretemeyeceğini bilmek çok önemlidir. Ayrıca bir problemin çözümünde gerekli hafızanın mevcut olma şartı vardır.

Uzay karmaşıklığının ele alınması algoritmanın tamamlanmasında kullanılan kısmi veri yapıları ile bağlantılıdır. Veri yapıları bu çalışmada yer almayacağından, uzay karmaşıklığı dikkate alınmayacaktır. Özellikle zaman karmaşıklığı üzerinde durulacaktır. Bir algoritmanın zaman karmaşıklığı, giriş özel bir ölçüye sahip olduğunda algoritmayla kullanılan işlem sayısına göre ifade edilebilir. Algoritmaların zaman karmaşıklığının ölçümünde kullanılan işlemler, tamsayıların karşılaştırılması, toplanması, çarpılması, bölünmesi yada diğer temel işlemler olabilir. Karmaşıklık, temel işlemleri düzenlemek için farklı bilgisayarlarda farklı zamanlara ihtiyaç olduğundan sahip olunan bilgisayara göre gereken işlemlerin sayısına bağlı olarak tanımlanır. Ayrıca, karmaşık bit işlemleri yapılacağı zaman bilgisayar kullanılır. En hızlı bilgisayarlar temel işlemleri(toplama,çarpma vb..)1 nano saniye, kişisel

bilgisayarlar 1 mikrosaniye de yaparlar.Yani, aynı işlemi kişisel bilgisayardan 1000 kez daha hızlı icra ederler.

2.6.2.Algoritmaların Oluşturulması: Sonlu bir tamsayı dizisinin en büyük elemanını bulmak için bir örnek algoritmayı ele alalım.

Örnek.1.7: Bir dizideki en büyük elemanı bulma problemi oldukça açık olmasına rağmen , algoritma kavramının anlaşılması açısından güzel bir örnek oluşturur . Ayrıca, sonlu bir tamsayı dizisindeki en büyük elemanı bulmayı gerektiren birçok algoritma örneği vardır . Örnek olarak bir üniversitede binlerce öğrencinin katıldığı rekabete dayalı bir yarışmada en yüksek skorun bulunmasına ihtiyaç duyulabilir .veya bir spor organizasyonunda her ay en yüksek reyting alan üyenin tanıtılması istenebilir .

Biz sonlu bir tamsayı dizisindeki en büyük elemanı bulmakta kullanılacak bir algoritma geliştirmek istiyoruz .

Bu problemi birçok yolla çözmek için prosedür belirtebiliriz. Şöyle bir çözüm sağlayabiliriz .

Çözüm.1.7 : Aşağıdaki adımları yerine getiriyoruz .

1 . Maksimum değerini geçici olarak dizinin ilk elemanına eşitliyoruz .(Geçici maksimum değeri prosedürün herhangi bir adımında sınanmış en büyük tamsayıdır)

2 . Dizinin bir sonraki elemanıyla geçici maksimum değerini karşılaştırıyoruz, eğer dizinin bir sonraki elemanı geçici maximum değerinden büyükse, geçici maximum değerine bu sayıyı atıyoruz .

3 . Eğer dizide başka sayılar varsa önceki adımları tekrarlıyoruz .

4 . Dizide karşılaştıracak tamsayı kalmadığında duruyoruz.Bu noktada geçici maksimum dizideki en büyük tamsayıdır.

Ayrıca bilgisayar dili kullanılarak algoritma tanımlanabilir. Böyle bir şey yapıldığında sadece o dildeki komutların kullanılmasına izin verilir. Bu da algoritma tanımını karmaşık ve anlaşılması zor bir hale getirir . Üstelik birçok programlama dilinin genel kullanımında , özel bir dil seçmek istenmeyen bir durumdur . Bu yüzden algoritmaları belirtirken özel bir bilgisayar dili yerine pseudocode kullanılır.(Ayrıca bütün algoritmalar ingilizce tanımlanır)

Pseudocode bir algoritmanın ingilizce dil tanımı ve bu algoritmanın programlama diline dönüştürülmesi arasında bir ara basamak oluşturur .

Programlama dilinde kullanılan komutlar bu noktada geçici maximum değeri dizinin en büyük elemanıdır.İyi tanımlanmış işlemleri ve durumları içerir . Herhangi bir bilgisayar dilinde başlangıç noktasında psedeucode tanımlanarak bir bilgisayar programı üretilebilir .

Bu çalışmada pascal programlama dili temel alınarak pseudocode kullanılmıştır . Bununla birlikte pascal ve diger programlama dillerinin söz dizimi kullnılmayacaktır .Daha ilerde iyi tanımlı komutlar bu pseudocotta kullanılabilir . Aşağıda sonlu bir dizideki maximum elemanı bulan pseudocode algoritması tanımlanmıştır .

Algoritma .1.1 : Sonlu bir dizideki maksimum elemanı bulan algoritma

Procedure max(a_1,a_2,.....a_n: integers)

max : =a_1

for i:=2 to n

if max<a_i then max:=a_i

{maksimum dizinin en büyük elemanıdır}

Bu algoritma önce dizinin ilk elemanını maksimum değişkenine atar. For döngüsü dizinin elemanlarını ardışık bir şekilde incelemekte kullanılır. Eğer bir terim max'ın geçerli değerinden daha büyük ise maksimumun yeni değeri olarak atanır.Algoritmalarda genel olarak kullanılan bir çok özellik vardır . Algoritmalar tanımlanırken bunları akılda tutmak yararlıdır .

2.7.Algoritmaların Özellikleri :

Giriş : Bir algoritma açıkça belirtilen bir kümeden giriş değerlere sahiptir .

Çıkış : Bir algoritmanın her bir giriş değerinin kümesinden, çıkış değerlerinin kümesi üretilir . Çıkış değerleri problemin sonucunu içerir

Tanımlılık : Algoritmanın adımları tam olarak tanımlanmalıdır.

Sonluluk: Bir algoritma herhangi bir giriş kümesi için sonlu sayıdaki adımlardan sonra istenilen sonucu üretmelidir .

Etkinlik: Algoritmanın her bir adımı tam olarak ve sınırlı bir zamanda gerçekleşebilmelidir.

Genellik : Prosedür, sadece belli giriş değerleri için değil istenilen formdaki bütün problemler için uygulanabilir olmalıdır.

Sonlu bir tamsayı dizisindeki maksimum elemanı bulan algoritmanın özellikleri şöyle ifade edebiliriz.Algoritmadaki girdi tamsayı dizisidir.Çıktı dizideki en büyük tamsayıdır. Algoritmadaki her adım tam olarak tanımlanmıştır , atamalar sonlu döngü ve şarta bağlı durumlar yer almaktadır. Algoritma sınırlı bir zamanda karşılaştırma ve atama adımlarının herbirini gerçekleştirmektedir. Sonuç olarak algoritma geneldir ve herhangi bir sınırlı tamsayı dizisindeki maximum sayıyı bulmak için kullanılabilir .

2.8.Arama algoritmaları ve performans analizi

Bu bölümde ve ikinci bölümde değinildiği gibi bir algoritmanın performans analizi o algoritmanın karmaşıklık hesabı anlamındadır. Şimdi sırasıyla Lineer ve İkili arama algoritmalarının performans analizini ele alalım. Karmaşıklık analizinin ikinci bölümde belirtildiği gibi üç çeşidi vardır. Bunlar, En Kötü Durum Analizi, Ortalama Durum Analizi ve Deneysel analizdir.Burada sadece En Kötü Durum Analizi ve Ortalama Durum Analizi ele alınmıştır.Ortalama durum analizinde işlemlerin ortalama sayısı kullanılarak verilen tüm girişlerden problem çözülür. Ortalama durum zaman karmaşıklığı analizi genellikle en kötü durum analizinden daha karmaşıktır. En Kötü Durum Analizi en yaygın kullanılan analizdir.

Sıralanmış bir listedeki bir elemanın yerini bulma problemi pekçok konuda karşımıza çıkar.Sıralanmış kelimelerin listesinin bulunduğu bir sözlükte, kelimelerin hecelenmesini kontrol eden bir program bunun en basit örneklerinden biridir.Bu tür problemler arama problemleri olarak adlandırılır. Bu bölümde arama için birkaç algoritma ele alınacaktır.

Genel olarak arama problemleri şöyle tanımlanır. Farklı $a_1,a_2,.....a_n$ elemanlarının listesinden x'in yerini bulur, ya da listede olmadığını belirler. Bu arama problemi için çözüm dizi içerisinde x'e eşit olan sayının yeridir. Eğer $x=a_i$ ise çözüm 1'dir. x=0 ise dizide yoktur. İlk algoritma sıralı arama algoritmadır . Sıralı arama algoritması x ve a_1'i karşılaştırarak başlar . $x=a_1$ ise çözüm a_1'in yeridir . Eğer, x a_1'e eşit olmazsa x , a_2 ile karşılaştırılır. Çözüm , a_2'nin yeridir. Eğer, x, a_2'ye eşit olmazsa x,a_3 ile karşılaştırılır. Bir eşleme oluncaya kadar listedeki her bir eleman ile x'i karşılaştırma işlemine devam edilir. Çözüm x'e eşit olan terimin yeridir. Eğer, tüm liste arandığında x'in yeri yoksa sonuç 0'dır.

Algoritma.1.2. Sıralı Arama Algoritması

```
Procedure lineer search(x:integer , a1,a2,....,an: distinct integer)
i :=1
while (i≤n and x≠ai)
i:=i+1
if i≤n then location :=i
else location :=0
```

{location x'e eşit olan terim , eğer "x" 0 ise bulunamadı}

Yeni bir arama algoritması incelenecek . Bu algoritma listedeki terimlerin artan bir şekilde sıralı olması halinde kullanılabilir . (Terimler sayı ise küçükten büyüğe , eğer kelime ise alfabetik olarak sıralamalıdır) . Bu ikinci algoritma ikili arama algoritması olarak adlandırılır. Eleman listenin ortasına yerleştirilmiş terimler karşılaştırılarak ilerler . Liste aynı boyutta iki küçük listeye ayrılır veya küçük listelerden birindeki terim sayısı diğerinden bir tane az olabilir. Diğer bölümde ikili arama algoritmasının sıralı arama algoritmasından daha etkili olduğu gösterilecektir . İkili aramanın nasıl çalıştığı aşağıdaki örnekte gösterilmiştir.

1 sayısı için arama yapılması :

1 2 3 5 6 7 8 10 12 13 15 16 18 1 20 22

16 terimden oluşan listeyi 8 terimli iki listeye ayırıyoruz.

1 2 3 5 6 7 8 10 12 13 15 16 18 1 20 22

İlk listedeki en büyük terimle 1 sayısı karşılaştırılıyor. 10<1 olduğundan arama orijinal listedeki 9. Ve 16. Terime sınırlandırılıyor. Daha sonra bu liste 4 terimden oluşan iki listeye ayrılıyor.

12 13 15 16 18 1 20 22

ilk listedeki en büyük terimle 1 karşılaştırılıyor. 16<1 olduğundan arama orijinal listedeki 13. ve 16. terim arasına sınırlandırılıyor. Liste iki parçaya ayrılıyor.

18 1 20 22

İlk listenin en büyük terimi olan 1 sayısı 1'dan büyük olmadığından dolayı arama ilk listedeki 18 ile 1 sayıları ile sınırlandırılıyor. (Orijinal listedeki 13. Ve 14.terim). Sonra iki terimin listesi tek bir terim olarak ayrılıyor. 18<1 olduğundan arama ikinci listeye sınırlanıyor. Liste 14.terimi içeren 1 sayısıdır. Şu an arama bir terime indiriliyor , karşılaştırma yapılıyor ve 1 sayısı orijinal listenin 14. terimine yerleştiriliyor.

2.8.1.Lineer arama algoritması

Algoritma.1.3 : The linear search algorithm.

procedure linear search (x **:** integer , a_1 , a_2 , **.....** , a_n **:** distinct integers)

i **:** 1

while ($i \geq n$ ve $x \neq a_i$)

$i := i + 1$

if $i \leq n$ **then** location **: =** i

{ location is the subscript of term that equals x , or is 0 if x is not found }

2.8.1.1: Lineer arama algoritmasının zaman karmaşıklığı(en kötü durum analizi)

Zaman karmaşıklığının ölçümü karşılaştırma sayısı kullanılarak yapılacaktır. Algoritmada çevrimin her adımında, iki karşılaştırma yapılacaktır. Birincide listenin sonuna gelindiğinde ve listenin birinci terimiyle ***x*** elemanı karşılaştırılacaktır. Sonuçta bir dahaki karşılaştırmada çevrimden çıkılır. Dolayısıyla, eğer ***x= a_i , 2i+1*** karşılaştırması kullanıldı. En çok karşılaştırma ***2n***, listedeki olmayan eleman olduğu zamandır. Bu bölümde ***2n*** karşılaştırması ***x*** hesabında kullanıldıysa ***a_i*** değildir ve sonraki karşılaştırmalarda çevrimden çıkılır. ***x*** listede değilse ***2n+2*** karşılaştırmaları toplamı kullanılır. Böylece ***O(n)*** karşılaştırmaları. Bir lineer seçim olmuştur. Bu karmaşıklık analizi **en kötü durum analizidir**. Bu analiz bize bir algoritmada kaç tane işlem yapılırsa çözüme ulaşılacağının garantisini verir.

2.8.1.2. Lineer arama algoritmanın ortalama durum performansının hesabı

Listede x bilindiği zaman mümkün n tip giriş vardır. Listede x ilk terimse, 3 karşılaştırmaya ihtiyaç duyulur. Birinci hesap liste sonuna gelinip gelinmediğini, birinci karşılaştırma x ve ilk terim ve ilk çevrim dışına çıkılır. Eğer listede x ikinci terimse, 2' den fazla karşılaştırmaya ihtiyaç duyulur. Bu yüzden toplam 5 karşılaştırma yapılır. Genelde eğer x i' ninci terimse çevrimden i adımın herbirinde iki karşılaştırma kullanılır ve bir çevrim dışına çıkılır. Bu yüzden 2i+1 karşılaştırmaya ihtiyaç duyulur. Böylece karşılaştırmaların ortalama sayısı :

$$\frac{3+5+7+.....+(2n+1)}{n} = \frac{2(1+2+3.....+n)+n}{n}$$ 'dir.

$$1+2+3+.....+n = \frac{n(n+1)}{2}$$

Böylece lineer arama algoritmada kullanılan karşılaştırmaların ortalama sayısı:

$$\frac{2(n(n+1)/2)}{n}+1 = n+2 = O(n)$$

2.8.2.İkili arama algoritması

Algoritma.1.4 : The Binary search algoritma.

procedure binary search (x **:** integer , a_1 , a_2 , **.....** , a_n **:** increasing integers)

i **:** 1 {***i*** is left endpoint of search interval}

j **:** *n* {***j*** is right endpoint of search interval}

while (***i*** < ***j***)

m **:** = [(***i*** + ***j***) / 2]

if *x* > a_m **then** ***i*** **:** = *m* + 1

else ***j*** **:** = *m*

end

if *x* = a_i **then** location **:** = ***i***

else locations **:** = 0

{ location is the subscript of term that equals x , or is 0 if x is not found **}**

2.8.2.1.İkili arama algoritmanın zaman karmaşıklığının hesabı

Kolaylık için listede $n=2^k$ eleman olduğunu farz edelim. ***k*** pozitif tamsayı listede ***k=log n*** geçmekte, 2' nin kuvveti değildir. 2^{k+1} elemanla daha büyük bir liste dikkate alınabilir. Burada $2^k<n<2^{k+1}$ 'dir.

Algoritmanın her bölümünde , i ve j ilk terimin yerinde ve en son terimle sınırlandırılmıştır. Karşılaştırmada bir terimden daha fazla terim sınırlandırılamayacağı gözlenir. Eğer i<j , sınırlandırılmış listenin orta teriminden, x' in daha büyük olup olamayacağı hesaplanır.

İlk kısımda, seçim 2^{k-1} terimle sınırlandırılarak yapılır. Şimdiye kadar 2 karşılaştırma kullanıldı. Bu prosedüre devam edersek, bir listede her bölgenin sınırlandırılmasında iki karşılaştırma kullanarak birçok terimin yarısıyla bir liste seçiminde.

Diğer durumlarda, liste 2^k elemana sahip olduğu zaman algoritmanın ilk kısmında iki karşılaştırma kullanılır. İkiden fazla seçim yapılacağı zaman 2^{k-1} elemanla liste yapılır. Sonuç olarak listede bir terim ayrılacağı zaman, birinci karşılaştırmada takip eden terimler atılır ve bir daha karşılaştırmada eğer terim x ise hesaplamada kullanılır.

Böylece,en çok 2k+2=2logn+2 karşılaştırmayla, 2^k elemana sahip seçilen bit listede ikili bir ikili seçimi gerçekleştirdik. Sonuç olarak bir ikili seçim yapılırken

O(logn) karşılaştırma yapılır. Bu analiz bize göstermiştir ki algoritmada ikili seçim lineer seçimden daha etkindir.

2.9. Sıralama Algoritmaları ve Karmaşıklığı (Ağaç Yapılar)

2.9.1.Giriş: Bir kümedeki elemanların sıralanması problemi çok çeşitli konularda ortaya çıkar.Örneğin,telefon idaresi bir rehber bastırmak istese abonelerinin isimlerini alfabetik olarak sıralaması gerekir.

Bir kümedeki elemanların hepsinin sıralı olduğunu varsayalım.Sıralama ,bu elemanların artan bir sırada bir liste içinde yeniden sıralaması olarak tanımlanır. Örneğin,7,2,1,4,5,9 elemanlarından oluşan bir listeden 1,2,4,5,7,9 şeklinde sıralanmış yeni bir liste üretebiliriz.Başka bir örnek olarak,d,h,k,m,n,a harflerinden oluşan listeden alfabetik sıra gözönüne alınarak a,d,h,k,m,n şeklinde yeni bir liste elde ederiz.

Bilgisayar kullanımının büyük bir oranı bir şeyi veya diğer bir şeyi sıralamaya bağlıdır.Bu nedenle,etkili sıralama algoritmaları geliştirmek için daha çok çaba sarfetmemiz gerekir.Bunun için,burada bazı sıralama algoritmaları ve onların karmaşıklık analizi incelenmiştir.Bu inceleme için ağaç yapıları (Trees) kullanılmıştır.

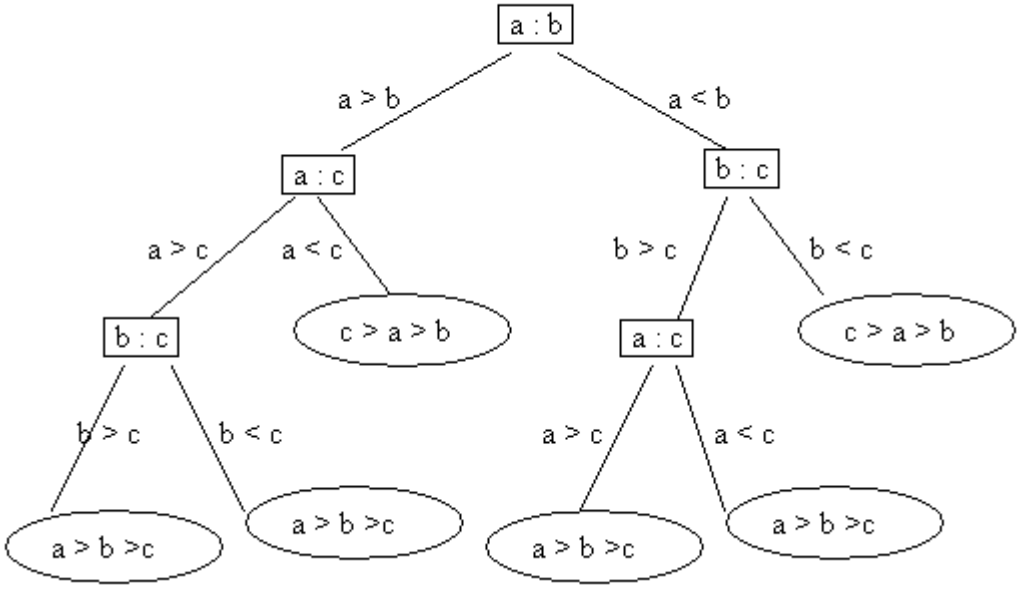
2.9.2.Sıralama Algoritmalarının Karmaşıklığı

Literatürde pekçok sıralama algoritması geliştirilmiştir.Belirli bir sıralama algoritmasının etkili olup olmadığına karar vermek için onun karmaşıklığının belirlenmesi gerekir. Ağaç Yapılı Algoritmalar**,** diğer sonlu algoritmalarda olduğu gibi iterasyon dizileri, pekçok paralel dalları içeren bir ağaç şeklindedir. Bir çok ağaç arama algoritmaları bu sınıfa aittir. Bu arama algoritmalarının bazıları, Dinamik Programlama, Dal ve Sınır, Sınırlı Sayım, Kapalı Sayım, Dal ve Çıkarma, Dal ve Atma vb. olarak sıralanabilir.

Modeller olarak ağaçları kullanarak sıralama algoritmalarının en kötü durum karmaşıklığı için bir alt sınır bulunabilir. n elemanlı bir kümenin permütasyon sayısı ,aynı zamanda n elemanın mümkün olan sıralama sayısı olup n!'dir.Sıralama algoritmalarının karmaşıklık hesabı için ikili karşılaştırmalara dayalı bir yol izlenecektir. Yani,bir zamanda iki eleman karşılaştırılacaktır. Buna göre,böyle karşılaştırmaların herbirinin sonucu olarak aşağıya doğru daralan mümkün sıralama kümeleri oluşur.

Böylece, sıralama algoritması ikili karşılaştırmalara dayalı ikili bir karar ağacı yardımıyla tarif edilebilir. İkili karar ağacı herbir iç tepe noktası iki elemanın karşılaştırılması esasını içerir.Her bir yaprak, n elemanlı bir kümenin n! permütasyonunun birini temsil eder.

Teorem.1.17:Aşağıdaki şekilde de ifade edilen üç elemanlı bir kümenin ikili karşılaştırmalarına dayalı bir sıralama algoritması için en az **(logn!)** karşılaştırmaya ihtiyaç vardır.(Şekil.1.5)



Şekil.1.5.Üç farklı elemanın sıralaması için Karar Ağacı

(logn!)= O(nlogn) olduğu hatırlanırsa karşılaştırmaları kullanarak O(nlogn)'den daha iyi en kötü durum zaman karmaşıklığına sahip bir sıralama algoritması bulunamaz.Sonuç olarak, O(nlogn) zaman karmaşıklığına sahip böyle etkili bir algoritma bulmak zordur.

2.10.Algoritmaların Karmaşıklığı için Kullanılan Terminoloji ve İşlem Zamanı

Tablo.1.6.algoritmaların zaman karmaşıklığının tanımında kullanılan terminolojiyi gösterir. Örnek olarak, $O(n^b)$ zaman karmaşıklığıyla belirtilen bir

algoritma dendiğinde polinom karmaşıklıktan bahsedilir. Lineer arama algoritması, lineer (en kötü yada ortalama durumu) karmaşıklığa ve ikili arama algoritması logaritmik (en kötü durum) karmaşıklığa sahiptir. 'O' sembolü bir algoritmanın zaman karmaşıklığının tahminini ifade eder. Bununla beraber bilgi zamanın etkin kullanımında 'O' zaman karmaşıklığının tahmini direk yapılamaz. Birinci neden bir büyük 'O' tahmini f(n)=O(g(n)), burada f(n) bir algoritmanın zaman karmaşıklığını ve g(n) bir referans fonksiyondur. n>k olduğu zaman f(n)<Cg(n) dir. C ve k sabitlerdir. Eşitsizlikte C ve k bilinmeksizin bu tahmin kullanılan işlemlerin sayısında bir üst sınır hesaplanamaz.Bununla beraber eğer tüm işlemler bilgisayar kullanılarak bit işlemler şeklinde yapılacaksa problem çözümünde kullanılacak algoritmanın zaman hesabı yapılabilir.

Tablo1.6.algoritmaların zaman karmaşıklığının tanımında kullanılan terminoloji

Karmaşıklık	Terminoloji
O(1)	*Sabit Karmaşıklık*
O(log n)	*Logaritmik Karmaşıklık*
O(n)	*Lineer Karmaşıklık*
O(n logn)	*nlogn*
$O(n^b)$	*Polinom Karmaşıklık*
$O(b^n)$, $b>1$	*Üstel Karmaşıklık*
O(n!)	*Faktöriyel Karmaşıklık*

Tablo 1.7 bit işlem sayısını göstermektedir. 10^{100} yıldan büyük zamanlar asterisk (*) ile gösterilmiştir. Bu tablo yapılırken her bit işlemi 10^{-9} saniye alınmıştır. Gelecekte bit işlem hızı gelişen bilgisayar teknolojisiyle daha artacaktır.

Problem çözümünde nasıl bilgisayara ihtiyaç duyduğumuz önemli bir noktadır. Örnek olarak bir algoritma 10 saat ise bu problemin çözümü için bilgisayar zamanına ve para harcamaya değer. Fakat eğer algoritma 10 milyar yıl ise bu algoritmanın sonuçlanması olanaksızdır. Gelişen bilgisayar teknolojisi hızları ve hafıza kapasitelerini artırmıştır. Özellikle paralel işlem özelliği algoritmaların ve problem çözümlerini kolaylaştırmıştır.

Tablo.1.7.Algoritmalar tarafından kullanılan zaman

Problem Uzunluğu	Kullanılan bit işlem					
n	*logn*	*n*	*nlogn*	n^2	2^n	*n!*

10	$3x10^{-9}$	10^{-8}	$3x10^{-8}$	10^{-7} s	10^{-6}	$3x10^{-3}$s
10^2	$7x10^{-9}$	10^{-7}	$7x10^{-7}$	10^{-5} s	$4x10^{13}$yıl	*
10^3	$1x10^{-8}$	10^{-6}	$1x10^{-5}$	10^{-3} s	*	*
10^4	$1.3x10^{-8}$	10^{-5}	$1x10^{-4}$	10^{-1} s	*	*
10^5	$1.7x10^{-8}$	10^{-4}	$2x10^{-3}$	10 s	*	*
10^6	$2x10^{-8}$	10^{-3}	$2x10^{-2}$	17 dakika	*	*

Fonksiyonların büyüme oranlarında düşük düzen adlandırmalarına aldırmayabiliriz
-----$O(n^3+4n^2+3n)$ bir algoritmaysa bu algoritma aynı zamanda $O(n^3)$'tür

-----biz algoritmanın fonksiyon büyüme oranındaki gibi yüksek düzende adlandırmaları kullanabiliriz.

Fonksiyonun büyüme oranının yüksek düzenli adlandırmasındaki sabit çarpıma aldırmayabiliriz.
-----$O(5N^3)$ bir algoritmaysa bu aynı zamanda $O(n^3)$'tür.

$O(f(n))+O(g(n))=O(f(n)+g(n))$
----Fonksiyonların büyüme oranlarını sabitleyebiliriz
----Bir algoritma şu şekildeyse:$O(n^3) + O(4n^2)$.
Bu ayrıca bu şekildedir: $O(n^3 +4n^2)$ ➔ dolayısıyla bu şekildedir: $O(n^3)$.
----Benzer yöneticiler çeşitliliği sağlamak için saklarlar

Örnek 1:

```
                                                  değer          zaman
        i = 1;                                     d1              1
        sum = 0;                          d2              1
        while (i <= n) {                  d3              n+1
                i = i + 1;                d4              n
                sum = sum + i;            d5              n
        }
```

T(n) = d1 + d2 + (n+1)*d3 + n*d4 + n*d5
= (d3+d4+d5)*n + (d1+d2+d3)
= a*n + b
➔ fonksiyonun büyüme oranı **O(n)**'e bağlıdır

Örnek 2:

```
                                                  Cost            Times
        i=1;                                       d1              1
        sum = 0;                          d2              1
        while (i <= n) {                  d3              n+1
                j=1;                              d4              n
                while (j <= n) {          d5              n*(n+1)
                    sum = sum + i;        d6              n*n
                    j = j + 1;                    d7              n*n
          }
          i = i +1;                               d8              n
        }
```

T(n) = d1 + d2 + (n+1)*d3 + n*d4 + n*(n+1)*d5+n*n*d6+n*n*d7+n*d8
= (d5+d6+d7)*n2 + (d3+d4+d5+d8)*n + (d1+d2+d3)
= $a*n^2 + b*n + c$
➔ fonksiyonun büyüme oranı **$O(n^2)$**'ye bağlıdır.

3.MODÜLER ARİTMETİK

3.1.Modüler Aritmetiğe Ait Temel Kavramlar

Tanım.1.2: a = 0 olmak üzere a ve b tamsayılar ise ve b = ac şeklinde bir c tamsayısı varsa

a, b' yi böler diyebiliriz. a b' yi böldüğünde b'nin faktörü a' dır ve a' nın çarpanı b' dir. a/b şeklinde gösterilir. a / b şeklinde gösterildiğinde a b' yi bölemez anlamındadır.

Tanım.1.3: 1' den büyük pozitif p tamsayısının çarpanları sadece 1 ve kendisi ise böyle sayılara asal denir. 1' den büyük ve asal olmayan pozitif tamsayılara "composite" denir.

Tanım.1.4: Bölüm algoritmasında verilen eşitlikte d bölen, a bölünen, q bölüm ve r kalan olarak adlandırılır.

Tanım.1.5: a ve b, 0' dan farklı tamsayılar olsun. d / a ve d / b olmak üzere en büyük d tamsayısı a ve b' nin en büyük ortak bölen' i olarak adlandırılır. a ve b' nin en büyük ortak böleni gcd(a,b) şeklinde gösterilir.

Tanım.1.6: a ve b tamsayılarının en büyük ortak böleni 1 ise bu sayılara (nispeten) asal sayılar denir.

Tanım.1.7: a_1 , a_2 ,...., a_n 'nin $1 \leq i < j \leq n$ şartıyla $\gcd(a_i , a_j) = 1$ ise böyle sayılara ikişerli aralarında asal sayılar denir.

Tanım.1.8: a ve b pozitif tamsayılarının en küçük ortak çarpanı, a ve b'nin her ikisiyle bölünebilen en küçük pozitif tamsayılardır. a ve b' nin en küçük çarpanı lcm(a,b) şeklinde gösterilir.

Tanım.1.9: a, bir tamsayı ve m, bir pozitif tamsayı olsun. a, m' ye bölündüğünde kalanı a mod m şeklinde gösteririz.

Tanım.1.10: a ve b tamsayılar ve m bir pozitif tamsayı olsun. Bu durumda, eğer m a-b' yi bölerse, modul m' ye göre a, b' ye denkdir denir ve $a = b \pmod m$ ile gösterilir. Eğer a ve b modul m'ye göre denk değillerse $a \neq b \pmod m$ ile gösterilir.

Teorem.1.4: a, b ve c tamsayılar olsun.

1. Eğer a / b ve a / c ise a / (b+c);
2. Eğer a / b ise tüm sayılar için a / bc' dir.
3. Eğer a / b ve b / c ise a / c' dir.

Ispat: a / b ve a / c olduğunu dğüşünelim. Bölünebilirlik tanımından aşağıdaki b = as ve c = at ile s ve t tamsayıları vardır. Bununla beraber;

b + c = as + at = a(s + t)' dir.

Bundan başka; a, b + c' yi böler.

a tamsayısı 1' e ve kendine bölünebildiğinde 1' den büyük her pozitif tamsayı en küçük iki tamsayı ile bölünür. İki farklı pozitif çarpanları içeren tamsayılara asal denir.

Teorem.1.5: Bölüm Algoritması: a ve d pozitif tamsayılar olsun. a = dp + r olmak şartıyla $0 \leq r < d$ iken; q ve r tek tamsayılardır.

Teorem.1.6: a ve b pozitif tamsayılar olsun. ab = gcd(a,b)lcm(a,b)' dir.

Teorem.1.7: m bir pozitif tam sayı olsun. a = b + km olmak üzere sadece bir k tamsayısı varsa a ve b modul m' ye göre denkdir.

Ispat: $a \equiv b \pmod{m}$ ise, m / (a-b)' dir. Bunun anlamı; a – b = km iken bir k tamsayısı vardır. Böylece a = b + km' dir. a = b + km olmak üzere bir k tamsayısı varsa km = a – b' dir. bununla beraber m, a – b' yi böler, böylece $a \equiv b \pmod{m}$' dir.

Teorem.1.8: m bir pozitif tamsayı olsun. $a \equiv b \pmod{m}$ ve $c \equiv d \pmod{m}$ ise;
$a + c \equiv b + d \pmod{m}$ ve $ac \equiv bd \pmod{m}$' dir.

Ispat: $a \equiv b \pmod{m}$ ve $c \equiv d \pmod{m}$ iken, b = a + sm ve d = c + tm ile s ve t tamsayıları vardır. Bununla beraber ;

b + d = (a + sm) + (c + tm) = (a + c) + m(s + t) ve

db = (a + sm)(c + tm) = ac + m(at + cs + stm)' dir.

Ayrıca;

$a + c \equiv b + d \pmod{m}$ ve

$ac \equiv db \pmod{m}$' dir.

3.1.1.Bölünebilirliğin Özellikleri: Her a,b,c $\in$ Z için aşağıdakiler doğrudur.

i) a $\mid$ a

ii) Eğer, a $\mid$ b ve b $\mid$ *c* ise a $\mid$ *c* olur.

iii) Eğer, a $\mid$ b ve a $\mid$ *c* ise her x,y $\in$ Z için a $\mid$ (bx + cy) 'dir.

iv) Eğer, a $\mid$ b ve b $\mid$ a ise $a = \pm b$ olur.

Teorem.1.9: $\pi(x)$, x'den küçük asal sayıları göstersin. $x \geq 17$ için;

$$\pi(x) > x / \ln x$$

ve $x > 1$ için; $\pi(x) < 1{,}25506\,(x / \ln x)$

Teorem.1.10 : (Aritmetiğin Temel Teoremi) Her $n \geq 2$ tamsayısı için; p_i'ler farklı asallar ve e_i'ler pozitif tamsayılar olmak üzere;

$$n = p_1^{e_1} p_2^{e_2} \ldots\ldots p_k^{e_k}$$

olup bu yazılış tektir.

Özellik.1.1: Her bir $e_i \geq 0$ ve $f_i \geq 0$ için, Eğer, $a = p_1^{e1} p_2^{e2} \ldots\ldots p_k^{ek}$, $b = p_1^{f1} p_2^{f2} \ldots\ldots p_k^{fk}$ ise; *Ebob* (a,b) $= p_1^{\min(e_1,f_1)} p_2^{\min(e_2,f_2)} \ldots\ldots p_k^{\min(e_k,f_k)}$ ve

Ekok (a,b) $= p_1^{\max(e_1,f_1)} p_2^{\max(e_2,f_2)} \ldots\ldots p_k^{\max(e_k,f_k)}$.

Örnek.1.3: $a=4864=2^8.1$ ve $b=3458=2.7.13.1$ olsun.Buna göre,

gcd(4864,3458)=2.1=38 ve lcm(4864,3458)= 2^8.17.13=442624

1.11.2.Euler phi fonksiyonunun özellikleri

i) Eğer, p asal sayı ise **ø (*p*) = *p* – 1**

ii) Eğer, gcd(*m*,*n*) = 1 ise ø (*m*.*n*) = ø (*m*) **.** ø (*n*)

iii)Eğer, $n= p_1^{e1} p_2^{e2} \ldots\ldots p_k^{ek}$ n'in asal çarpanları ise;

ø (*n*) = *n* (1- (1 /p_1)) (1- (1 /p_2)) ..... (1- (1 /p_k))

Özellik.1.2: Her $n \geq 5$ tamsayısı için,

ø (*n*) > (*n* /6 ln ln *n*)

3.2.Z 'de Algoritmalar

a ve b negatif olmayan tamsayılar olsun ve her biri *n* 'den küçük veya eşit olsun.

Tablo.1.2.Z'de dört işlemin karmaşıklığı

İşlem		Bit Karmaşıklığı
Toplama	a + b	O (lg a + lg b) = O (lg n)
Çıkarma	a – b	O (lg a + lg b) = O (lg n)
Çarpma	a **.** b	O((lg a) (lg b)) = O ((lg n)^2)
Bölme	a = qb + r	O((lg a) (lg b)) = O ((lg n)^2)

Özellik.1.3 : Eğer, a ve b, a > b şeklindeki pozitif tamsayılar ise gcd(a,b) = gcd(b,a mod b)

Algoritma.1.1: İki tamsayının en büyük ortak bölenini bulan Euclidean algoritması

İnput : a ve b, $a \geq b$ özelliğindeki iki pozitif tamsayı olsun.

Output: a ve b'nin en büyük ortak böleni;

1. b ≠ 0 iken aşağıdakini gerçekleştir.

   1.1 set r←a mod b, a←b, b←r .

2. return(a)

Sonuç.1.2: Eucledian algoritmasının çalışma zamanı $O((\lg n)^2)$ bit işlemdir.

Örnek.1.4: Eucledian algoritmasından 4854 ve 3458 sayılarının en büyük ortak bölenini bulun. gcd(4864,3458) = 38:

$$4864 = 1 \cdot 3458 + 1406$$
$$3458 = 2 \cdot 1406 + 646$$
$$1406 = 2 \cdot 646 + 114$$
$$646 = 5 \cdot 114 + 76$$
$$114 = 1 \cdot 76 + 38$$
$$76 = 2 \cdot 38 + 0$$

Eucledian algoritması sadece iki tamsayının en büyük ortak bölenini bulmanın yanısıra x ve y tamsayıları için ax + by = d eşitliği bulmak istendiğinde genişletilebilir.

Algoritma.1.2: Genişletilmiş Euclidean Algoritması

İnput : a ve b, a ≥ b özelliğindeki iki pozitif tamsayı olsun.

Output: d = gcd(a,b) olan x,y tamsayıları için ax + by = d sağlanır.

1. if b = 0 then set d←a, x←1, y←0, and return(d,x,y).
2. set x_2←1, x_1←0, y_2←0, y_1←1.
3. while b > 0 do
   - 3.1 q←[a / b] , r←a – qb, x←x_2 – qx_1, y←y_2 – qy_1
   - 3.2 a←b, b←r, x_2←x_1, x_1←x, y_2←y_1 , and y_1←y
4. set d←a, x←x_2 , y←y_2 , and return (d,x,y)

Genişletilmiş Euclidean Algoritmasının çalışma zamanı $O((\lg n)^2)$ bit işlemdir.

Örnek.1.5: Tablo.1.2 'de a = 4864 ve b = 3458 girişleri için Genişletilmiş Euclidean Algoritması kullanılarak gcd(4864,3458) = 38 ve (4864)(32) + (3458) – (-45) = 38 olarak bulunur. İnput: a = 4864 ve b = 3458.

Böylece, gcd(4864,3458) = 38 ve (4864)(32) + (3458) – (-45) = 38

Tablo.1.3. a = 4864 ve b = 3458 girişleri için Genişletilmiş Euclidean Algoritması

Q	r	x	y	a	b	x_2	x_1	y_2	y_1

-	-	-	-	4864	3458	1	0	0	1
1	1406	1	-1	3458	1406	0	1	1	-1
2	646	-2	3	1406	646	1	-2	-1	3
2	114	5	-7	646	114	-2	5	3	-7
5	76	-27	38	114	76	5	-27	-7	38
1	38	32	-45	76	38	-27	32	38	-45
2	0	-91	128	38	0	32	-91	-45	128

1.12.1. Denkliğin özellikleri: Her $a,a_1,b,b_1,c \in Z$ için aşağıdakiler doğrudur.

i) $a \equiv b \pmod{n}$ sadece n 'ye bölündüklerinde aynı kalanı verdiklerinde doğrudur.

ii) (yansıma özelliği) $a \equiv a \pmod{n}$

iii) (simetri özelliği) $a \equiv b \pmod{n}$ ise $b \equiv a \pmod{n}$

3.3. Z_n^* Üretecinin Özellikleri

i) Yalnız ve yalnız, $n = 2,4\ p^k$ veya $2p^k$ ve p tek asal sayı ($k \geq 1$) iken Z_n^* üreteçtir.Kısmen de , p asalsa Z_p^* üreteçtir.

ii) Eğer, α, Z_n^* in üreteci ise $Z_n^* = \{ \alpha^i \bmod n \mid 0 \leq i \leq \phi(n) - 1 \}$

iii)Varsayalım ki, α, Z_n^* in üreteci olsun.Eğer, gcd(i,ø(*n*)) = 1 ise b= α^imodn ' de

Z_n^* in üreteci olur.Böylece, Z_n^* periyodiktir üretilenlerin sayısı ø(ø(*n*))'dir.

iv) Yalnız ve yalnız, ø(*n*)'in herbir asal böleni için $\alpha^{\phi(n)/p} \equiv 1 \pmod{n}$ olursa $\alpha \in Z_n^*$

Z_n^* 'ın bir üretecidir.

3.3.1.Z_n 'de Algoritmalar: n pozitif bir tamsayı olsun. Buna göre,

Z_n={ 0,1,2,... *n* – 1 }kümesi ile tanımlanır.Eğer, **a,b** $\in$ **Z_n** ise**;**

$$(a + b) \bmod n = \begin{cases} a+b, & a+b < n \\ a+b-n, & a+b \geq n \end{cases} \text{ olur.}$$

Böylece, Modüler toplama veya çıkarma bölme gerektirmeden düzenlenebilir.

Algoritma.1.3. Z_n'de Çarpmanın Terslerinin Hesabı

Giriş: $a \in Z_n$

Çıkış: $a^{-1} \bmod n$ 'nin bulunması.

1. $ax + ny = d$ ve $d = \gcd(a,n)$ olan x , y tamsayıları Genişletilmiş Euclidean Algoritması kullanılarak bulunur.
2. Eğer, d > 1 ise $a^{-1} \bmod n$ mevcut değildir.Aksi halde,x'e geri dön.

Not.1.1 : $a^k = \prod_{i=0}^{t} a^{k_i 2^i} = \left(a^{2^0}\right)^{k_0} \left(a^{2^1}\right)^{k_1} \ldots\ldots\ldots \left(a^{2^t}\right)^{k_t}$ şeklindeki modüler üs alma işlemi aşağıdaki algoritma ile etkili olarak düzenlenir ve bu özellik kriptografi protokollerinin çoğu için çok önemli bir araçtır.

Algoritma.1.4: Z_n'de üs alma işlemi için tekrarlanan kare alma ve çarpma algoritmaları

Input : $a \in Z_n$ ve 2'li gösterimi $k = \sum_{i=0}^{t} k_i .2^i$ olan $0 \geq k_n$ olsun

Output : $a^k \bmod n$

1. Set b←1.if k = 0 then return(b).
2. Set A←a .
3. if $k_0 = 1$ then set b←a

4.For i = 1 to t do

4.1 Set A←A

4.2 if $k_i = 1$ then set b←A . b mod *n*

5. return (b)

Örnek.1.6:Aşağıdaki tabloda 5^{596} mod 1234 = 1013 'ın hesaplanmasındaki adımlar gösterilmiştir.

Tablo 1.4. 5^{596} mod 1234 = 1013'ün hesaplanması

i	0	1	2	3	4	5	6	7	8	9
k_i	0	0	1	0	1	0	1	0	0	1
A	5	25	625	681	1011	369	421	947	947	925
b	1	1	625	625	67	67	1059	1059	1059	1013

3.3.2:Z_n'de temel işlemlerin karmaşıklığı: Tablo 1.5 ile Z_n'de toplama , çıkarma , çarpma, tersini bulma ve üs alma işlemlerine ait bit karmaşıklığı verilmiştir.

Tablo.1.5. Z_n'de temel işlemlerin bit karmaşıklığı

İşlem		Bit Karmaşıklığı
Modüler toplama	(a + b) mod *n*	$O(\lg n)$
Modüler çıkarma	(a – b) mod *n*	$O(\lg n)$
Modüler çarpma	(a . b) mod *n*	$O((\lg n)^2)$
Modüler ters	a^{-1} mod *n*	$O((\lg n)^2)$
Modüler üs alma	a^k mod *n*,*k* < *n*	$O((\lg n)^3)$

3.4.Denkliğin Uygulamaları

Sayı teoresi kavramı içine giren **Denklik** pek çok alanda kullanılır. Bunların en önemlileri :

1.Hafıza bölgelerini bilgisayar dosyalarına atamak için benzerlik kullanımı .

2.Rasgele sayıların üretimi .

3.Modüler aritmetiğe dayalı şifreleme sistemi .

3.4.1.Hashing Fonksiyonları

Okuldaki her bir öğrenci için kayıtları devam ettiren merkezi bir bilgisayarımız olduğunu düşümelim.Bu öğrencilerin kayıtlarına hızlı bir şekilde yeniden ulaşmak için ne kadarlık bir hafıza bölgesi atanmalıdır.Böyle bir problemi çözmek için uygun bir hashing fonksiyonu seçilmiştir. Her bir öğrencinin kaydı bir

anahtar kullanılarak tanımlansın. Örneğin bu anahtaröğrencinin sosyal güvenlik numarası olarak seçilebilir. Buna göre "h" hashing fonksiyonu "k" anahtarına sahip olan h (k) hafıza bölgesine atar.

Pratikte pek çok farklı hashing fonksiyonları kullanılır. Bunun en yaygın kullanımı h(k)=k(modm)olanıdır. Burada m elde edilebilir hafıza bölgesinin sayısıdır. Hashing fonksiyonları kolayca değerlendirilebilir. Bu sayede verilere hızlı bir şekilde ulaşılabilir. h(k)=k mod m hashing fonksiyonları bu gereksinimleri karşılar. H(k)'yı bulmak için k'nın m'ye bölümünden kalanı hesaplamamız yeterli olacaktır. Ayrıca bütün hafıza bölgelerine ulaşmak bu fonksiyonla mümkündür.

Örnek.1.8: 111 hafıza bölgesine sahip olalım. Sosyal güvenlik numarası 064212848 olan öğrenci h(064212848)=064212848 mod 111=14 hashing fonksiyonu yardımıyla 14.hafıza bölgesine atanır.Sosyal güvenlik numarası 037149212 olan öğrenci ise benzer yolla

h(037149212)=037149212 mod 111=65

hashing fonksiyonu yardımıyla 65.hafıza bölgesine atanır.Hashing fonksiyonu birebir olmadığında (hafıza bölgesinden daha çok anahtar olduğunda) birden fazla dosya bir hafıza bölgesine atanır. Bu olduğunda çakışma oluştu denir. Hashing fonksiyonları yardımıyla atanan son hafıza bölgesinde ilk boş alanına atanır. Örneğin önceden 2 atama yapıldıktan sonra sosyal güvenlik numarası 107405723 olan öğrencinin kaydını 15.hafıza bölgesine atar.

h(107405723)=107405723 mod 111=14 olduğu gözönüne alınır ve 14 nolu hafıza bölgesinin 064212848 nolu öğrenciye ait olduğu bilindiğinden müteakip ilk boş hafıza bölgeside 15 olduğundan buradaki çakışma önlenmiş olur.

3.4.2.Yarı Rastgele Sayılar

Rasgele seçilen sayılar bilgisayarların simulasyonunda gereklidir. Rasgele seçilen sayıların özelliklerine sahip üretilmiş sayılar için farklı metodlar mevcuttur. Sistematik metodlar yardımıyla üretilen ayılar gerçekten rasgele olmadığından olayı bu sayılara yalancı rasgele sayılar denir.

Yalancı sayıların üretiminde kullanılan en yaygın işlem lineer benzerlik metodudur. Bu metodta 4 tamsayı seçilir. Bunlardan m modülü , a çarpanı , c aralığı ,(x0)? Asıl sayıyı göstermek üzere $2 \leq a < m$, $0 \leq c < m$ ve $0 \leq xn < m$? şeklindedir.Bütün n'ler için $0 \leq xn < m$ aralığında yalancı rasgele sayıların {xn} dizisini üretmek için (xn+1=(axn+c)) mod m benzerliği ardışık olarak kullanılır.

Bu bağıntı aynı zamanda indirgeme bağıntının tanımıdır. İlerki bölümlerde bu bağıntıdan bahsedilecektir. Birçok bilgisayar uzmanı 0-1 arasındaki yalancı rasgele sayıların üretimine ihityaç duyar.Bu sayıları modüller yardımıyla bir lineer benzerlik bağıntısı kurup üretilen sayıları bölerek elde ederiz. Yani xn/m sayılarını kullanırız.Örnek olarak m=9 , a=7 , c=4 ve x_0=3 seçilerek yarı rastgele sayıların kümesi aşağıdaki gibi üretilir.

Örnek.1.9. $x_9=x_0$ ve her bir terim bir önceki terime bağlı olduğundan 3 ,7 ,8 ,6 ,1 ,2 ,0 , 4, 5 ,3 , 7 , 8 , 6 ,1 , 2 , 0 , 4 , 5 , 3..... dizisi üretilir. Bu dizi 9 tane farklı sayının sonsuz tekrarından ibarettir. Pek çok bilgisayar bunu kullanır. En çok kullanılan * ifadesinde c'nin 0 olduğu bağıntıdır. Buna pure multiplicative genarator denir. Örneğin 231-2 ? modülü ve 7^5 =16807 çarpanı ile pure multiplicative generator yaygın olarak kullanılır. Bu değerler ile 2^31 −2 tane sayı tekrarsız olarak üretilir.

BÖLÜM 4

Şifreleme Algoritmaları (Kriptoloji)

Temel Terminoloji:

Özellikle internet teknolojisinin yaygınlaşmasıyla adını sıkça duymaya başladığımız kriptoloji gitgide önem kazanmaktadır.

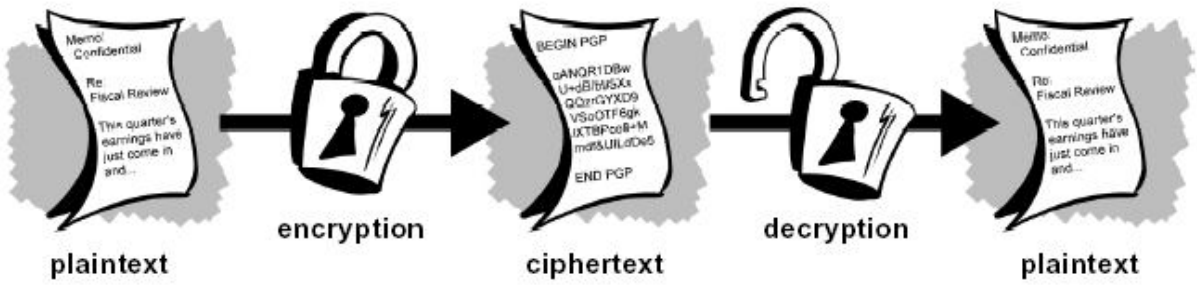
Bu bilim dalı üç ana kısımdan meydana gelir

Kriptografi, Kriptoloji, Kriptoanaliz

Kriptografi gizli mesajlaşma, onaylama, dijital imzalar, elektronik para ve diğer uygulamaların tüm yönleriyle ilgilidir. Kriptoloji ise kriptografik metotların matematiksel temelleriyle ilgilenen bir matematik dalıdır

Kriptografik algoritmaların açıklarını bulup ortaya çıkartmaya ise kriptoanaliz denilmektedir.

Şifreleme (Encryption), Veriyi bir anahtarla şifrelemeye verilen addır. Hedef, veriyi gerekli anahtar olmadan çözülebilmesi imkansıza mümkün olduğunca yakın şekilde kodlamaktır. Şifre Çözme(Decryption) ise şifrelenmiş veriyi çözüp eski haline getirme işlemidir.



Farz edelim ki bir kişi diğer bir kimseye bir mesaj göndermek istiyor ve başka hiçbir kimsenin mesajı okumadığından emin olmak istiyor. Ancak, bir başka kimsenin mektubu açması veya elektronik iletişimi duyması olasılığı vardır. Kriptografik terminolojide mesaj plaintext (düz-metin) veya cleartext (temiz-metin) olarak adlandırılır. Mesajın içeriğini diğer kişilerden saklamak için kodlamaya encryption (şifreleme) adı verilir. Şifrelenmiş mesaja ciphertext (şifreli-mesaj) denir. Ciphertext'ten düz-metni elde etme işlemine decryption (çözme) adı verilir. Veriyi şifrelerken ve çözerken kullanılan matematiksel metoda is eşifreleme algoritması denilmektedir. Şifreleme ve çözme genelde bir anahtar(Key) kullanılarak yapılır ve şifreleme algoritması öyledir ki çözme işlemi ancak doğru anahtarın bilinmesiyle gerçekleştirilebilir.

1.17.3.Kriptoloji (Şifreleme)

Benzerlikler kesikli matematik ve bilgisayar biliminde çok yaygın kullanılır. Kongrüansın en önemli uygulamalarından biri gizli mesajları içeren kriptolojilerdir. Kriptoloji' yi kullananlardan biri Julius Caesor'dır. Caesor alfabedeki (ingiliz alfabesi) her bir harfi üç harf ileri kaydırarak gizli mesajlar oluşturmuştur. Bu şifrelemenin bir örneğidir. Yani gizli bir mesaj yapma işlemidir. Caesor'ın bu şifreleme işlemini matematiksel olarak ifade etmek için alfabedeki pozisyonuna bağlı olarak her bir harfi 0'dan 25'e bir tamsayıya eşliyoruz. Yani A→0 , K→10 , Z→25 şeklinde. Böylece Caesor'ın şifreleme metodu $p \leq 25$ olmak üzere negatif olmayan bir p tamsayısı yardımıyla $f(p) = (p+3) \bmod 26$ şeklinde tanımlanır. Burada f(p) bir tamsayı olup {0, 1, 2,3, ...,25} kümesinden değer alır.

Örnek.1.10: "MEET YOU IN THE PARK" mesajından Caesar'ın şifreleme yöntemini kullanarak gizli mesaj üretebiliriz.

Çözüm.1.10: İlk olarak mesajın harflerini numaralara çeviriz.

12 4 4 1 24 14 20 8 23 1 7 4 15 0 17 10

Her numarayı $f(p)=(p+3) \bmod 26$ fonksiyonuyla değiştiririz.

15 77 22 1 17 23 11 16 22 10 7 18 3 20 13

Numaraları harflere çevirerek şifrelenmiş mesaj PHHW BRX LQ WKH SDUN olarak üretilir. Caesar'ın şifreleme yöntemiyle şifrelenmiş gizli mesajdan orijinal mesajı elde etmek için f^-1 fonksiyonu kullanılır. F^-1 fonksiyonu {0,1,2.....,25} sayılarından bir p tamsayısı gönderir.

$F^{-1}(p)=(p-3) \bmod 26$

Orijinal mesajı bulmak için alfabedeki her bir harf üç harf geriye kaydırılır. Şifrelenmiş mesajdan orijinal mesaj belirleme işleminede şifre çözme(description) denir. Caesor'ın şifreleme yöntemini genelleştirebiliriz. Bunun basit örneği ;

$f(p)=(p+k) \bmod 26$ olup bu tip şifreye kaydırma şifre denir. Böyle bir şifreyi çözmek içinde $f^{-1}(p)=(p-k) \bmod 26$ kullanılır. (Burada k alfabedeki ilerletilecek sayıdır) Bununla birlikte Caesor'ın bu şifreleme yöntemi çok güvenli değildir. Bunun güvenirliliğini arttırmak için $f(p)=(ap+b) \bmod 26$ fonksiyonu seçilir. Burada a ve b tamsayılardır.

Örnek.1.11: f(p)=(7p+3) mod 26 fonksiyonunda k harfi hangi harfe karşılık gelir.

Çözüm.1.11: k=10.harf ise f(10)=(7.10+3) mod 26 =21=v

Böylece k'ya karşılık gelen v harfi bulunur.

Bilinen ilk şifreleme algoritması sezar şifresi adı verilen algoritmadır. Sezar adı verilen general komutanlarına mesaj göndermek için kullanmıştır.

Bu algoritmada şifreleme her bir harfi belirli bir miktar kaydırarak gerçekleştirilir. Çözme işleminde ise bu işlemin tersi yapılır. Bundan dolayı bu algoritmaya N Kaydırma algoritması da denilmektedir

Sezar Şifresi:

```
0123456789012345678901234 5
ABCDEFGHIJKLMNOPQRSTUVWXYZ
NOPQRSTUVWXYZABCDEFGHIJKLM
          |
         n=13
```

0 Açık Metin: "SEZAR SIFRESI"

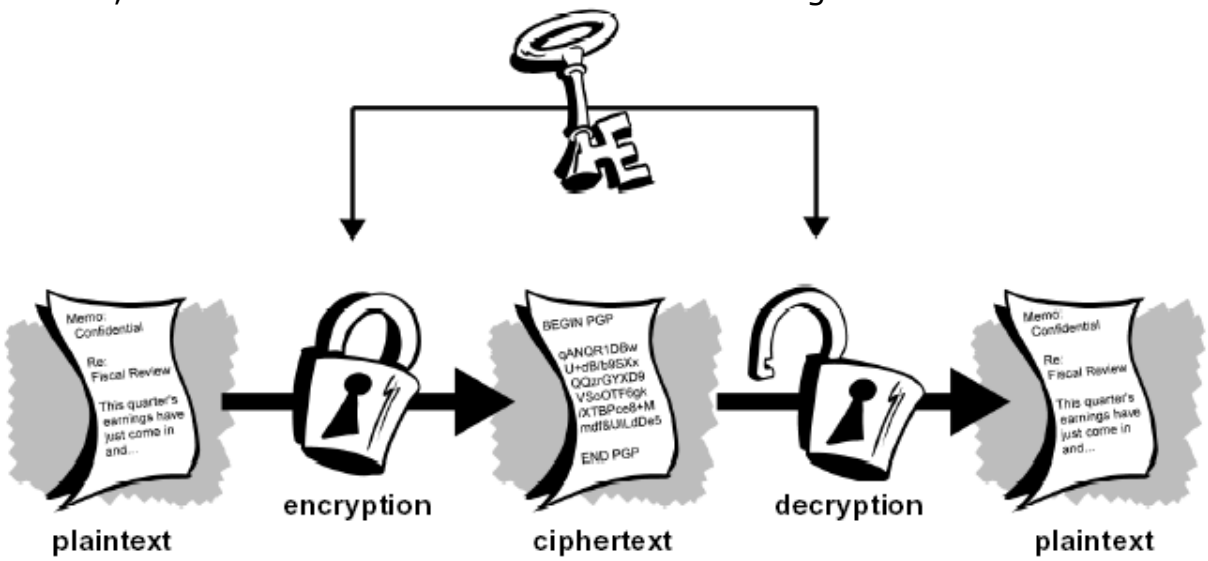
1 Şifrelenmiş Metin: "FRMNE FVSERFV"

Bu tür basit şifreleme metodlarını kırmak için (kriptoanaliz) fazla çaba göstermeye gerek yoktur, yeterince uzun bir şifreli metin elde olduğu zaman metinde geçen harfler sayılarak hangi harfin ne miktarda çıktığı hesaplanır, daha sonra her dilde kendine has olan harf kullanım oranları ile karşılaştırılarak kelimeler deşifre edilir. Edgar Allen Poe'nin Golden Bug adlı hikayesinde de aynı şekilde hikayenin kahramanı korsan hazinesinin yerini anlatan şifreli mesajı çözmek için bu metodu kullanmıştı.

Geçmişte ilgilenilen kriptografik algoritmalar; algoritmanın gizliliğine dayanmaktaydı. Günümüzde kullanılmakta olan modern ve güçlü şifreleme algoritmaları ise artık gizli değildir. Bu algoritmalar güvenliklerini kullandıkları farklı uzunluk ve yapılardaki anahtarlarla sağlarlar. Bütün modern algoritmalar şifrelemeyi ve şifre çözmeyi kontrol için anahtarları kullanır. Bir anahtar ile şifrelenmiş bilgi, kullanılan algoritmaya bağlı olarak, ilgili anahtar ile çözülebilir. Anahtar yapısı bakımdan, şifreleme algoritmaları iki grupta incelenebilir:

6.4.1. Simetrik (Gizli) Anahtar Algoritmaları

Simetrik kripto-algoritmalar; şifreleme ve çözmede aynı anahtarı kullanma prensibine dayalı olarak çalıştıklarından "**simetrik**" olarak nitelendirilirler. Bu tip algoritmalarda tek bir gizli anahtar kullanıldığından **gizli anahtar algoritması** diye de adlandırılırlar. Bu algoritmalarda şifreli iletinin çözülebilmesi için alıcının, kullanılan anahtarı daha önceden bilmesi gerekir.



Bu sistemlerin avantajı hızı, dezavantajı ise ortak anahtarların belirlenmesi ve taraflara iletilmesindeki zorluktur.

Şekil.6.1 : Gizli Anahtar Algoritma

0 Çok Kullanılan Simetrik Şifreleme Algoritmaları

1. XOR Şifreleme

Oldukça sık kullanılan basit bir şifrelemedir. Buradaki işlem aslında tamamen bir baytı başka bir bayt ile XOR işlemine tabi tutmaktan ibaret. Şifrelenmiş yazıyı çözmek de çok basit; işlemi tekrarlamak yeterli (Tabi doğru anahtar biliniyorsa). Eğer anahtar bilinmiyorsa, şifreyi kırmak zordur.

Karakter	**A**
	1000010
Anahtar	**P**
	1010000
Sonuç	**¶**
	0010010

XOR Doğruluk Tablosu

A	B	A XOR B
1	1	0
1	0	1
0	1	1
0	0	0

XOR şifreleme tabiki bu kadar değil. Bir mesajın herbir baytını tek bir baytlık anahtar ile XORlanması ile yeterli oranda bir şifreleme elde edilemez. Bu yüzden, bir anahtar sözcük ile XOR'lama yapmak çok daha iyidir. Mesela;

'SIFRELEME RUTINI' ni 'password' sözcüğünü kullanarak XORla şifrelersek;

Kaynak	Anahtar	Sonuç
S	P	#
I	a	(
F	s	5
R	s	!
E	w	2
L	o	#
E	r	7
M	d	)
E	p	5
	a	A
T	s	'
E	s	6
K	w	<
N	o	!
I	r	;
K	d	/
L	p	<
E	a	$
R	s	!
I	s	:

sonucunu elde etmiş oluruz. Sonuçta elde ettiğimiz karakterleri şifreleme sözcüğü ile bir kez daha xorlayarak şifreyi kaldırabiliriz.

XOR şifrelemenin en zayıf yönü ise sıfırlardan oluşan bir dizinin herhangi bir anahtar ile XOR'lanması sonucunda sonucun anahtarın kendisi olmasıdır.

DES – (Data Encryption Standard)

En bilinen ve yaygın biçimde kullanılan simetrik algoritma **DEA (Data Encryption Algorithm)** dir. Bu algoritma **DES** (veri şifreleme standardı) olarak da bilinir.

160'ların sonunda IBM'de çalışan bir araştırmacı olan Horst Feistel başkanlığındaki bir grup LUCIFER adı verilen bir şifreleme sistemi geliştirmiştir. 173 yılında ABD standartlar enstitüsü NIST sivil kullanım için bir standart saptamak için firmaları davet etti. Yapılan incelemeler sonucu amaca en yakın çözüm LUCFIFER bulundu. 128 bitlik bir şifre anahtarına sahip LUCIFER üzerinde çalışan ABD güvenlik teşkilatı (**NSA**) uzmanları bazı düzenlemeler yaptılar ve anahtar uzunluğunu 56 bit'e indirdiler. Bu yeni algoritma 177 yılında DES olarak yayınlandı ve kısa bir süre içinde başta finans endüstrisi olmak üzere birçok alanda da standart olarak kullanılmaya başlandı.
DES klasik şifreleme sistemleri içinde efsanevi bir yere sahiptir ve bugün bile VISA, MASTERCARD, BKM, v.s. tüm kart sistemlerinin şifreleme omurgasını oluşturmaktadır. DES te en küçük değişikliğin, çok büyük farklar yarattığı çığ etkisi (**avalanche effect**) bulunmaktadır. Yani tek bir bitlik bir değişiklik bile sonucu tamamen değiştirmekte ve değişiklikler önceden tahmin edilememektedir.

DES algoritması GİZLİ ANAHTAR yöntemini kullanan SİMETRİK bir kriptodur. Yani her iki tarafta aynı anahtarları bilmek zorundadır. Aynı anahtar hem şifrelemeye, hem de çözmeye yarar. BKM + 40 BANKA veya VISA + Binlerce Banka için zorda olsa uygulanabilir bir sistemdir. Ama anahtarları gizlemenin ve dağıtmanın zorluğu uygulamayı kısıtlamaktadır.

3DES (Triple DES)

Standart DES'in 112 veya 168 bitlik iki veya üç anahtar ile artarda çalıştırılması ile oluşturulan bir şifreleme tekniğidir. Anahtar alanı 2^{112} veya 2^{168} sayısına ulaşınca bugün için veya tahmin edilebilir bir gelecekte çözülmesi mümkün olmayan bir kod olmaktadır. Buna "**strong crypto**" denilir, en güçlü teknik güç olan ABD'nin bile çözmesi mümkün değildir.

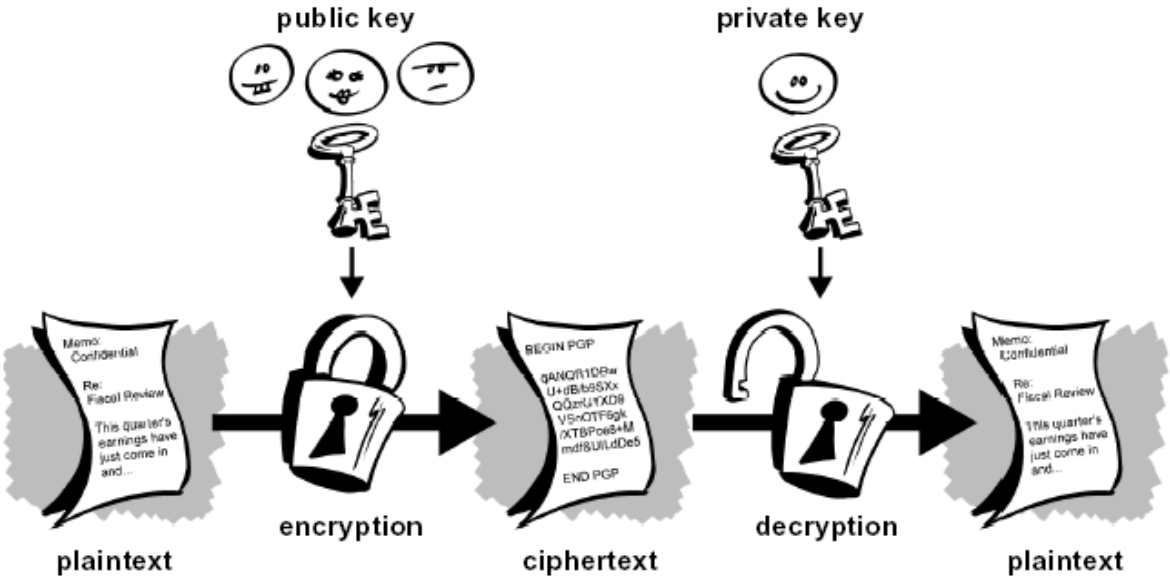
Bu nedenle kullanımı ve yayılması sınırlanmak istenmektedir. ABD ve birçok batı ülkesi 40 bit' ten daha güçlü kripto sistemlerin ihracını kısıtlamaktadır. 40 bit, birkaç saniyede ABD güvenlik kurumu NSA tarafından çözülebilmektedir ve böylece "**real-time**" dinlemeyi olanaklı kılmaktadır. 56-bit bile bugün için belirli bir süre gerektirmektedir. Şüphesiz anında çözebilecek güçte bazı sistemler vardır ama sayısı sınırlıdır ve maliyeti yüksektir. Bu nedenle bu gücün ciddi işlere tahsis edilebilmesi için 40 bit üzeri şifreler kısıtlanmaktadır. Türkiye'de bankalara verildiği söylenen 128 bit gücündeki sistemler, sözde SET ile çalışan siteler, v.s. hepsi bir aldatmacadır. Bu sistemlerdeki tüm şifrelerin anası firmalar tarafından NSA' ya teslim edilmektedir. NSA istediği bankanın bilgisayarına girerek kendi şifresiyle istediği şifre anahtarını çekebilmektedir. Başka türlü bir firmanın ABD'den ihracat izni alması mümkün değildir

Türkiye'de 128-bit şifreleme ile çalıştığını iddia eden veya SET olduğunu öne süren çoğu siteler 40-bit SSL ile çalışmakta olduğu bildirilmektedir. Bunu ilgili siteye bağlandığınızda çıkan kapalı anahtar üzerine tıklayarak görebilirsiniz. 128 bit görünen siteler ise sanki 128 bitmiş gibi gözüken ama aslında anahtar sahalarının belirli bölümü ABD tarafından bilinen veya tümü arkadan dolaşılarak öğrenilebilen sitelerdir.

6.4.2. Asimetrik Kripto-Algoritmalar

Whitfield Diffie ve Martin E.Hellman 176 da iki farklı anahtara dayalı algoritma geliştirilebileceğini gösterdiler. Anahtarlardan biri açık (**public**), diğeri ise gizli (**private key**) olmalıdır.

176 yılında Stanford Universitesinden Diffie ve Hellman adlı araştırmacılar iki farklı anahtara dayalı şifreleme sistemi önerdiler, bu sistemde bir tane şifreleme için(**public** key) ve bundan farklı olarak bir tanede şifre çözmek için(**private** key) anahtar bulunur ve **private** key, **public** key den elde edilemez.



Public Key: Public key otomatik olarak private keyden türetilir. private keyin tersine, adından anlaşıldığı gibi,geneldir. Dünyanın her yerindeki kişi ve anahtar sunucularına güvenle gönderilebilir. Mesajların sifrelenmesinde alıcının public keyi kullanılır. Örneğin herhangi biri size şifreli mesaj göndermek istediğinde,sizin public keyinize ihtiyacı vardır. Bu durumda Public keyinizi,anahtar sunuculardan (Eger eklemişseniz),yada direk sizden elde edebilir. Mesaj, public keyiniz ile beraber özel bir oturum anahtarı ile kodlanarak internet üzerinde size güvenli bir şekilde gönderilir. Tabi bu mesajı açabilecek tek kişi sizsiniz. Bunun için private key dosyanız bilgisayarınızda bulunmalı ve bu anahtarın şifresini bilmelisiniz.

Private key Kişi kendisine gönderilen şifrelenmiş mesajı açmak için private keyini kullanır. Bu kişiye özel olmalıdır,saklanmalıdır.

Farklı yerlerde bulunan iki kullanıcıdan birinin diğerine bilgi aktarmak istediğini düşünelim. Bu durumda şifrelemede ihtiyaç duyacağı anahtarlar, göndereceği kişinin public-key'idir. Bu anahtarı kullanarak şifreleme işlemini

yapar ve verileri karşı tarafa gönderir. Şifrelenmiş verileri almış olan kişi ise, şifreyi çözmek için, kendisinin private-key' ini kullanır.

Bu yöntem private-key' lerin karşılıklı aktarılmasını gerektirmez. Böylece ilk defa, gizli simetrik anahtar dağıtım problemi çözülmüş oldu ve herkes tarafından gerçeklenebilecek sayısal imza gibi yeni yöntemler mümkün oldu.

Burada da, private-key' lerin kullanıcılara nasıl dağıtılacağı konusu gündeme gelmektedir. Çünkü bu dağıtım işlemi esnasında da anahtarlar istenmeyen ellere geçebilir. Bu iş için e-posta 'dan yararlanılabileceği gibi (gerekli güvenlik önlemleri alınmak kaydıyla), telefonla da bu anahtarlar sahiplerine aktarılabilir. Tabi tüm bu işlemler sırasında güvenliğe azami derecede dikkat etmek gerekmektedir.

Şifreleri kişilere atamanın bir başka yolu da, bir sunucu vasıtasıyla dağıtma işlemini yapmaktır. Bu durumda, şifreleri ele geçirmek isteyenlere direkt bir hedef sunulmakta, burada sağlanacak yeterli bir güvenlik mekanizması ile, bu tehlike de önlenebilmektedir.

Her kullanıcıya iki ayrı anahtar atamanın temel sebebi, private-key ile yaşanan, verilerin karşılıklı olarak aktarılması esnasında zorunlu olan, private-key' lerin değiş-tokuş'u işleminden kurtulmak, dolayısıyla bunun getirdiği zorlukları bertaraf etmektir.

Görüldüğü gibi public-key yaklaşımı bize büyük bir avantaj getirmiştir. Ancak bunun da dezavantajları vardır. Bunların başında da, birbirlerine şifrelenmiş veri gönderecek bilgisayarların her defa yürütmeleri gereken el sıkışma protokolleri, ilgili anahtarları kullanarak bilgileri çözme işlemlerinin CPU zamanından çok fazla almasıdır. Bu zaman ileti uzunluğu ile üssel olarak artar. Bu nedenle anonim anahtarla şifreleme yöntemi birçok uygulamada gizli anahtarla şifreleme yöntemi ile birlikte kullanılır. Örneğin elektronik postaların şifrelenip gönderilmesinde en yaygın kullanılan yöntemler hem gizli hem de anonim anahtar algoritma yöntemlerini içerenlerdir. (PGP)

Şu an için private-key yaklaşımı ile ilgili olarak üzerinde çalışılan konuların başında, anahtar değiştirme işlemini otomatik yaparak, zaman kaybını en aza indirmektir.

RSA Algoritması

178 yılında, Ronald L.Rivest, Adi Shamir ve Leonard Adleman yukarıdaki kriterleri sağlayan bir algoritma önerdiler ve Metotları geniş çapta kabul edildi. RSA olarak bilinen ve dünyada en yaygın biçimde kullanılan asimetrik algoritma, ismini mucitlerinin baş harflerinden almıştır. Büyük sayıların aritmetiğine dayalı çok basit bir prensibi vardır.

Şifreleme : $y = x^e \bmod n$
Şifre Çözme : $x = y^d \bmod n$
$n = p*q$

Şekil 4.15 RSA Şifreleme ve çözme
x : açık metin e : açık anahtar
y : şifreli metin d : gizli anahtar
n : açık modül (public modulus)
p, q : gizli asal sayılar

Şekil.6.5: RSA

Şifreleme açık metnin exponansiyelinin (üssünün) alınması ve bunu takiben bir modül alma işleminden ibarettir. Bu işlem sonucunda şifreli metin elde edilir ve ancak gizli anahtarın bilinmesiyle benzer bir işlemle açılır.

Şifrelemeden önce, açık metnin uzunluğu uygun bir değere gelecek şekilde ek yapılmalıdır. Eklenecek alan kullanılan anahtar uzunluğuna bağlıdır.

Anahtarlar, iki büyük asal sayıdan üretilir

Dolayısıyla, algoritmanın güvenliği büyük sayı üretme problemine dayalıdır. İki asal sayının çarpımından hareketle açık modülü hesaplamak çok kolaydır, bununla birlikte açık modülü oluşturan asal çarpanları bulmak oldukça zordur. RSA algoritmasının anahtar üretim algoritması aşağıdaki basit örnekte görülmektedir.

RSA Anahtar Elde Etme Örneği

1. İki asal sayı seç (p ve q)	p = 3, q=11
2. *Açık modülü hesapla* (public modulus)	n = p*q =33
3. Anahtar üretimi için ara bir değişken hesapla (z)	z = (p-1)*(q-1)
4. *Açık anahtar (public key) "e"* yi aşağıdaki yöntemle hesapla: e < z ve gcd(z,e) = 1, yani z ve e nin en büyük ortak böleni 1. Bu özelliği sağlayan birden fazla sayı olabileceğinden biri seçilir.	= 2*10 = 20 e = 7
5. *Gizli anahtar "d" yi* hesapla: (d*e) mod z = 1	d = 3

RSA kullanım örneği

1. Açık metin "4" olsun	x = 4
2. Şifrele	$y = 4^7 \bmod 33 = 16$
3. Şifreyi çöz	$x = 16^3 \bmod 33 = 4$

Şekil6.6 : RSA Anahtar Elde Etme Örneği

Smartkartın RAM'ı büyük sayıların eksponansiyelinin hesaplanması durumunda yetersiz olacağından *"modülo-üsleme"* (modulo exponentiation)

denilen, ve hesaplanan değerlerin asla modülü geçmesine izin vermeyen bir yöntem kullanılır. Örneğin x^2 mod n değerinin hesaplanmasında yöntem çok büyük sayılarla uğraşılması gerekeceğinden (x*x) mod n işleminde biçiminde çalışmaz, bunun yerine aynı sonucu veren (((x mod n)*(x mod n)) mod n) yol tercih edilir. Bu daha küçük sayılarla uğraşılması demek olduğundan RSA için kullanılan bellek ve adım sayısı indirgenmiş olur.

Saldırıları zorlaştırmak için anahtar uzunluğu mümkün olduğunca büyük seçilmelidir. Açık ve gizli anahtar uzunlukları farklı olabilir, açık anahtarın uzunluğunun az olduğu durumlarda sayısal imzanın kontrolü için gereken zaman önemli ölçüde azaldığından genel olarak açık ve gizli anahtar uzunlukları farklı seçilir.

Bir saldırgan açık modülü (public modulus) asal çarpanlarına doğru biçimde ayırırsa anahtarı elde edebilir. Küçük bir sayı için bu kolay olmakla birlikte büyük bir sayı için bu oldukça zordur. RSA anahtarları bu yüzden yeterince büyük olarak seçilir. RSA de (512 bit=64 byte) anahtarlar yeterince büyük olarak kabul edilir. Bununla birlikte 768 bit ve 1024 bitlik anahtarlarda kullanılır. Anahtar uzunluğu şifreleme ve çözme zamanını doğrusal olarak değil eksponansiyel olarak arttırmaktadır.

RSA hesaplanmasını 8-bitlik mikroişlemcili bir smartkartta gerçeklemek bir kaç dakikadan uzun sürecektir. Bu yüzden RSA hesaplamayı destekleyici aritmetik birimler içermektedirler. Donanım destekli bir RSA algoritmasının program kodu 300 bayt uzunluğundadır. 768 bit ya da 1024 bitlik RSA için 1 kbyte lık bir assembler kodun kartta yer alması gerekir.

RSA bütün gücüne rağmen uzun işlem gerektirdiğinden veri şifreleme için ender kullanılır. Asıl kullanım alanı sayısal imzalardır.

6.4.4.1. RSA ile Kredi Kartı Şifreleme

Kredi Kartı şifrelemede M kart numaramız olsun ; M= 6882 3268 7966 6683,

Önce bu sayı küçük sayılara ayrılır. Örneğin;

m1=688 m2 =232 m3=687 m4=966 m5=668 m6=3

teker teker her biri şifrelenir.

$688^{79} \pmod{3337} = 1570 = c1$

Aynı işlemleri diğer m_i değerleri için yaptığımızda

C = 1570 2756 2714 2276 2423 158 değerini elde ederiz.

Bu mesaj ağ üzerindeki terminale gönderilir. Mesajın şifre çözümü terminalde yapılır. Terminal müşterinin bilmediği gizli anahtar değerini ("d") bilir. Böylece terminal yukarıdaki mesajın şifresini çözer.

C1 : 1570^{101} (mod 3337) = 688 = m1

Bu yolla mesajın arta kalan kısımları geri alınabilir.

6.4.4.2. Veri Alanı Geniş Rakamlarla RSA Örneği

Aralarında asal iki küçük rakam seçilir (p,q) p=5 q=17

Açık modül hesaplanır (public modulus) n=p*q

Ara bir değişken hesaplanır (phi(n)) phi(n)= (p-1)*(q-1)=4*16=64

Bir sonraki adım şifreleme anahtarı seçmek .Bu örnekte e=13 değeri kullanıldı.

Mesaj olarak M=3 değerini kullanmayı öne sürüyoruz. Fakat istediğiniz mesaj değerini seçebilirsiniz.5 'ten küçük bir değer olması hesap makinesinden kontrolü mümkün kılar.

Örneğin; Eğer 120 'nin 13 ile bölümünü bilmek istiyorsak hesap makinemizi

120/13=9.23076923077 değerini elde etmek için kullanırız.

Sonra tamsayı kısmını ondalık kısımdan (9) ayırmak için çıkarılır. Sonunda sonuç 13 ile çarpılır. 13*23076923077=3.0000000001 sonucu elde edilir.

Birinci adım :Mesajın üssü alınır. e=12 olsun

M^e =3^13=1 594 323

Şimdi n=85 ile bölümünden kalan kısım bulunur. Sonuç 63 olur.

M^e =3 ^13 =63 (mod n)

İkinci adım :Mesajı çözmek.

Sonuç ^5=63^5 =992 436 543

Ve hesaplama ;

63^5=992 436 543 =3 (mod n)

Eğer bu gerçel sayılarla yapılacaksa hesaplamayı kolaylaştırmak için birkaç teknik kullanmalı. Açık anahtar (e,n) ve gizli anahtar (d,n)

RSA Şifreleme Örneği

p = 61 <= first prime number (destroy this after computing e and d)

q = 53 <= second prime number (destroy this after computing e and d)
p.q = 3233 <= modulus (give this to others)
e = 17 <= public exponent (give this to others)
d = 2753 <= private exponent (keep this secret!)
Your public key is **(p.q,e)**
Your private key is **d**
The encryption function is: **encrypt(T) = (T^e) mod(p.q)**
= (t^17) mod 3233
The decryption function is: **decrypt(C) = (C^d) mod(p.q)**
= (C^2753) mod 3233
To encrypt the plaintext value 123, do this:

encrypt(123) = (123^17) mod 3233
= 337587917446653715596592958817679803 mod 3233
= 855
To decrypt the ciphertext value 855, do this:

decrypt(855) = (855^2753) mod 3233 =

504328889584160687344228991273944663145387836003550931555496756
4501
055628612082559978744245428110054383498654289336384930246451441 5
0785
172091796654782635307099638035387326500896686074771829745822950 3
4295
040790358184594095637793858659893688380836028401325097686207669 7
7396
675332505428260934757351379880632564826393344530925943855624292 3
3017
517710016924916912809150596011787601713497254392792156967017899 0
2
134307146468971279610277181378394586967728986934236524031169321 7
0892
696176437265213156658331587124597598030425031440068378832461017 8
4830
717585474547252069688925995892544366701432205469543174002285500 9
2386
369424448559733330630516073853028632130291350374547146757776713 5
79
549652029179050578153287155839207030315958593749366328354860209 0
830
635507044556588963131801134122017826923344101330116480696334024 0
75
046952588669876586690062240241020884665075302639538705266313358 4
734
810948761562271260373275973603752373883641480889484380961577570 4
5380
081079469800667348777958837582899851327930703533551275090439948 1
7897
905489933812173294585354474132680569810872633482854638168850488 2
4346
588978393334662544540066164521876669479552802308841246594823927 5
105
770491133290256843065052292561427303898320890070515110552506189 9
4171
231777951579794297117954752963018378438629139778776612982073890 7
2796

767202350113992715816427307640741898910486860748124549315795374377
124416014387650691458681640227602776686953090395131496831097324505
452345944772565878876926933539186923548185185424209230649964068221 84
490111357108854244285211207737122383110545543126530739407592789082 2
606043171133395752266034451645259763161842774590432011345289329932 1
613074405322274705728948121435868317841559727649635709090121513130 4
157569209798518321041155969357848833665315951327344675243940875769 977
789084901269153228420809496307929724713044221424390659030814289393 0
291584830873687450789770869218452967411463211556678655283381648067 795
455941891006950916589908545679807239237084630255354568691235546299
571573587906227458615721721110788286575638597094107763205097832395
713464110250047020848560408217509491077165531176529747380317676582 0
587673140288910328834318508844721164427139037404131556498699591373 6
516210845113740224335185995766577539693628125425390068552624545614 1
258809437402128886669744109721845342218171808991153707545542033911
964539366461792968165342652234639936742330970183533904623677693670 38
053426448217358238421251590438148524738896864244370318665419615377
913969649003039587606549152449450436001359392771339521012512857209 2
597887511601596296156902711643189463734265002363100455571800369358 6
055264910000907245183786689564417164907278356281009708545241354694 660
844811613387806548545151761673086051080657829365241087232636672280 54
003879410864348226750090778265121013728158316531396983090887317417 4
745359886842985598071851221597004650810606844559536480892249440542 7
663296745923088984848684358654798505115428440164623526969317993778 44
302178570117098751629654665130278009966580052178208139317232379013
232494682609200819810376848471678749891369499791482471634506093712
565412250153795166897601855087599313367797793952782227323337529580 2
631226653589482055665152894663690320832876804323906115493509545909 34

0667640225867084833760536998679410262047090571567447056531112428
6290
7354888492989983560999636092141128497745861469604028702967070147
8179
4902482829074841600836804586668550760461225209434980471574526881
813
1850859150148527635965034581536416565493160130613304074344579651
083
8030406224027889804282518909471629226689801668448096364518090510
905
7965130757037924595807447975237126676101147387874214414915481359
1743
9279949695641565386688389171544630561180536972834347021206348999
531
9176401611039249043917980339897549176539592360851180765318470647
3318
0157820741276478759273908749295571685366518591266637383123594589
1267
8709583800022451509424457564874484086877530845395521730636693891
7023
9403718478036277464317147085583049159895146776294392143100245613
061
1142993700055775133971728254911005600894089841671317091181655429
08
7610900832499783133824078696157849234186299168008677495934077593
066
0220781494380785499679894539936406368572269742236185841142504837
2451
2446558027085917979559108652309975651838277952945756996574245578
688
3835444236857223681399021261363744082131478483203563615611346287
018
5142390184290974163862023205103971218498335528630868518428263461
5027
4418735863950404228151239950599598365379222728584742207167783667
9451
3436380708657977421853595393166279988789721695963455346336497949
221
1301766131620747726611310701232140371388227022172323308547267953
3015
0799806225383545894802482004314472611596105260340690613093929072
4
1028494870016717296951770346790997944097506376492963567555800711
6218
2772760318292179035029048609097626628539662702439253689025633710
1471
6832740450458306022867631421581599007916426277000546123229121299
71
6990769016902594646810414121420447240266165827568052416686147339
3322
6595912700645630447416085291672187007045144649793226668732146346
7490
4118588676083684030610695786990096521390675205017440767765104388
51
5141613184799113492438815282203846472926944608491529995881859885
5

151490663073117772381322675169458825936387861072430256598091490103
2
783848214011365567849341024315124828645291703141004001201636482929853
251663490560537945850894244038552524554777922401046148907527451634253
139921637383568141490479320374263373018782540569961163520138969825
544786313097737491544784276345325939987417001381631811664537720894
4
002854850002696859826445621837941167021518477210933923218508777579
0
959332676311413129613984959261389879016697108810276638623167694057
2
959325380786434441005121380250817976227237972103521677326844146486
1640296105989902771053257045701633261343107641770004323715247462639
3
990118997278453629493036369149008810605312316300090101508393318801
16
682151638931046666595137827498923745560511004016477716822716267270
78
370122424655126487845492350418521674263831897333234346744490397800017
846897264054621480241241258338435017048853206014756878623180940900
12
632416909225202267988011340807301221626440413388739260052309607238
6
158554965158001034746117921307672245438036718832537086067133113258
1
992279755227718486484753261243028041779430909389923709380536520464462
551472678849615277732741126570911661358008414542148768731039444105
4
796393085308968803656085047721445921725001265007170689694281546275563
704588389042117739810648731080148287390581594622278672774186101111
027632479729041222119411738820452633570175909067862815928151982214
576527968538925172187200900703891385628400073322585075904853480465564
543498370732876259358914278543182665872946080723896522915990217388887
957736477387265746104008225511241827200961681888284938946788104688847
312655417262097890567845810965179753008730631546490302112133528180084
761229904095764278573163641248809309497707395675884229631711584645694
842024551090298823985179536841258914463527918973076838340736961314091
745229856386682726910433575176771288945278813686239650666540898943944
951611200216077789887686473648183782532484669916830728122031079135

64666840159148582699993374427677252275403853322168522985908515481
10
40229657916338257385513314823459591633281445818436145963060249936
17
53097925561238039014690665163673718859582772525683119899846460272
16
46279764077057074816406450769779869955106180046471378082232501489
34
07851137833251073753823403466269553292608813843895784099804170410
417
77608463062862610614059615207066695243018438575031762939543026312
673
77406936404705896083462601885911184367532529845888040849710922999
15
65539701111111883273086037667753396077224556321135065721106758 7
51186812786344175723921526333385653838824005710102564949233944516
65959203992392217400247234147109709645621082995477461322898118128
6
05556588093851898811812905614274085809168765711112247632886587127
55
38928438126611913792462411263299073986785455875665245305617509891
1457811473577128360755400177426866096509330517210272306663573946
2334
13638045914237759965220309418558880039496755829711258361621890140
359
54234930424749053693992776114261796407100127643280428706083531594
582
30594632682786127020335698034614324569702148437 5 mod 3233 = **123**

6.4.3. Algoritması açık Kriptografik Algoritmaların Gücü

Teorik olarak, bir anahtar kullanan Algoritması açık kriptografik algoritmalar, olası bütün anahtarları sırasıyla denemek yoluyla kırılabilir. Olası anahtarların denenmesi işlemi de, anahtarın uzunluğu arttıkça güçleşecektir. Örneğin 5 bitlik (5 adet 0 veya 1'den oluşan) bir anahtarın, en fazla 2^5=32 farklı anahtarı olabileceğinden, bu anahtarların sırayla denenmesi, şifrenin çözülmesine yeterli olacaktır. 32 bitlik bir anahtar için 2^32 yani 10^9 deneme yapılması gerekir. Bu bir amatörün kendi ev bilgisayarıyla deneyebileceği bir şeydir. 40 bitlik anahtara sahip bir sistem için 2^40 adim gerekir ki, bu da birçok üniversitenin ve hatta bazı küçük şirketlerin bile sahip olabileceği bir güçle mümkündür. 56 bitlik anahtarlı bir sistem (DES gibi) oldukça fazla bir çaba gerektirir fakat özel bir donanım yardımıyla bu da kırılabilir. Bu özel donanımın maliyeti oldukça fazla olmasına rağmen,organize suçluların, büyük şirketlerin ve hükümetlerin sahip olabilecekleri bir güçtür. 64 bitlik anahtarlı sistemler, büyük hükümetlerce kırılabilmektedir ve yakın zamanda organize suçlular, büyük firmalar ve daha küçük hükümetler tarafından da kırılabilecektir. Saniyede 3 trilyon anahtarı deneyebilen bir makine ile (yaklaşık 1 milyon dolar yatırımla elde edilebilecek bir makine), 56 bitlik DES algoritmasını 3,5 saat içinde kırılabilir. Aynı makine 80 bitlik anahtar kullanan bir algoritmayı 6.655 yılda, 128 bitlik anahtar kullanan bir algoritmayı ise 2.000.000.000.000.000.000 yılda kırabilir. Gelişen teknoloji ve işlemci hızları ile bu tip işlemlerin daha hızlı gerçekleştirilebileceği açıktır. 250 bitlik anahtar kullanan bir algoritmanın kırılması için gereken çabayı hesaplamak için termodinamik yasaları kullanılabilir. Buna göre, bir bit işlemi için gerekli minimum enerji kullanılarak yapılan hesaplamada, 250 bitlik tek bir anahtarı kırmak için güneşin tahmin edilen ömrü boyunca dışarı verdiği tüm enerjinin gerekli olduğu ortaya çıkar. Güneşin sadece bir yıl boyunca dışarıya verdiği enerji ile yetinecek olursak, en fazla 12 bitlik bir anahtarı kırmamız mümkün olur. Görüldüğü gibi, teorik olarak kırılabilir kabul edilen algoritmaların pratikte kırılması, anahtar büyüdükçe imkansızlaşmaktadır.

Anahtar Uzunluğu	Sayı Değeri	10^6 şifre/s	10^9 şifre/s	10^{12} şifre/s
32 bit	$\sim 4 \times 10^9$	36 dak	2.16 s	2.16 ms
40 bit	$\sim 10^{12}$	6 gün	9 dak	1 s
56 bit	$\sim 7.2 \times 10^{16}$	1142 yıl	1 yıl 2 ay	10 saat
64 bit	1.8×10^1	292 000 yıl	292 yıl	3.5 ay

128 bit	$1.7x10^{38}$	$5.4x10^{24}$ yıl	$5.4x10^{21}$ yıl	$5.4x10^{18}$ yıl

Çift anahtarlı kriptografide kullanılan anahtarların uzunlukları simetrik anahtarlı kriptografide kullanılanlardan genellikle çok daha büyüktür. Açık anahtar kriptografisini kırabilmek için, doğru anahtarı tahmin etmek değil, açık anahtardan ona uygun gizli anahtarı elde etmek gerekmektedir. Örneğin RSA için bu, iki büyük asal çarpanı olan büyük bir sayının çarpanlarına ayrılması anlamına gelmektedir. Bu işlem de teorik olarak yapılabilir olmasına karşın, pratik olarak gerçekleştirilmesi günümüz teknolojisiyle imkansızdır.

Bir kripto sistemin gücü genellikle en zayıf noktasına eşittir. Dolayısıyla, algoritma seçiminden, anahtar dağıtımına ve kullanım koşullarına kadar hiç bir şey göz ardı edilmemelidir.

Kriptanaliz ve Kripto sistemlere saldırılar

Kriptoanaliz

- Sadece Şifreli Metin Saldırısı
- Bilinen Düz Metin Saldırısı
- Seçilen Düz Metin Saldırısı
- Ortadaki Adam Saldırısı

Kriptoanaliz ve Kriptosistemlere Saldırılar:

Kriptanaliz uygun anahtarların bilinmeden şifrelenmiş iletişimlerin çözülmesi sanatıdır. En önemli kriptanaliz tekniklerinden bazıları aşağıda verilmiştir:

Cipher-text only (Sadece Şifreli Metin) Saldırısı:

Bu saldırı tipinde, saldırıyı yapan kişi mesajın içeriği hakkında hiç bir şey bilmemektedir, ve sadece şifreli-metni kullanarak çalışmalıdır. Uygulamada düz metne (plain text) - pek çok mesaj türü sabit başlık formatlarına olduğundan - ilişkin tahminler yapmak genelde olasıdır. Sıradan mektuplar ve belgeler bile kestirilebilir bir şekilde başlamaktadır. Örneğin, pek çok klasik saldırıda ciphertext'in frekans analizi kullanılmaktadır, ancak modern cipherlara karşı bu yöntem iyi çalışmamaktadır. Buna rağmen mesajlar bazen istatistiksel bir yanlılık içermektedirler.

Bilinen Düz-Metin Saldırısı:

Saldırgan ciphertext'in bazı kısımlarından düz metni tahmin edebilir veya bölebilir. Geriye kalan iş bu bilgiyi kullanarak ciphertext bloklarını çözmektir. Bu işlem, veriyi şifrelemek için kullanılan anahtarın belirlenmesi ile yapılabilir. En sık kullanılan Bilinen Düz Metin Saldırısı, blok cipher'lara karşı lineer kriptanaliz saldırısıdır.

Seçilmiş Düz Metin Saldırısı:

Saldırgan bilinmeyen anahtar ile şifrelenmiş istediği her metni elde edebilmektedir. Buradaki iş, şifreleme için kullanılan anahtarı belirlemektir. Bu saldırı için iyi bir örnek, blok cipherlara karşı (ve bazı durumlarda hash fonksiyonlarına karşı) uygulanabilen diferensiyel kriptanalizdir. Bazı kriptosistemler, özellikle RSA, seçilmiş-düz metin saldırılarına karşı açıktır. Bu tip algoritmalar kullanıldığında, uygulama (veya protokol) öyle tasarlanmalıdır ki saldırgan istediği düz metni şifrelenmiş olarak elde etmeMElidir.

Ortadaki Adam Saldırısı:

Bu saldırı, kriptografik iletişim ve anahtar değişimi protokolleri ile ilgilidir. Fikir şudur; iki kişi güvenli iletişim için anahtarlarını değiş-tokuş ederken (örneğin Diffie-hellman kullanarak), bir düşman kendisini iletişim hattındaki iki kişi arasına yerleştirir. Sonra bu düşman her iki kişi ile ayrı bir anahtar değiş-tokuşu gerçekleştirir. Her iki kişi farklı bir anahtar kullanarak işlerini tamamlayacaklardır ki bu anahtarlar düşman tarafından bilinmektedirler. Bu noktadan sonra saldırgan uygun anahtar ile herhangi bir iletişimi deşifre edebilecek ve bunları diğer kişiye iletmek için diğer anahtar ile şifreleyecektir. Her iki tarafta güvenli bir şekilde konuştuklarını sanacaklardır, ancak gerçekte saldırgan konuşulan herşeyi duymaktadır.

Ortadaki-adam-saldırısını engellemenin bir yolu dijital imzaları kullanabilen bir açık anahtar kriptosistemi kullanmaktır. Kurulum için her iki tarafta karşı tarafın açık anahtarını bilmelidir (ki bu bazen açık anahtar kriptosisteminin esas avantajını baltalamaktadır). Paylaşılan gizlilik oluşturulduktan sonra, taraflarkendi dijital imzalarını karşı tarafa göndermelidir. Ortadaki-Adam bu imzaları taklit etmeye çalışacak, fakat imzaların sahtesini yapamayacağı için başarısız olacaktır.

Bu çözüm, açık anahtarların güvenli bir biçimde dağıtımı için bir yolun varlığı halinde yeterlidir. Böyle bir yol X.509 gibi bir sertifika hiyerarşisidir. Bu, örneğin, IPSec (Internet Protocol Security) 'de kullanılmaktadır.

Gizili anahtar ile kriptosistemin çıktısı arasındaki korelasyon kriptanaliz için ana bilgi kaynağıdır. En kolay durumda, gizli anahtar hakkkındaki bilgi direkt olarak kriptosistemden sızdırılır. Daha karmaşık durumlar, kriptosistem hakkında gözlenen (ölçülen) bilgi ile tahmin edilen anahtar bilgisi arasındaki korelasyon üzerinde çalışmayı gerektirir. Bu sıkça çok ileri düzey saldırı durumlarında kullanılır. Örneğin, blok cipherlara karşı lineer (ve diferansiyel) saldırılarda kriptanalist bilinen (seçilmiş) düz metin ve gözlenen şifreli-metin üzerinde çalışır. 180 lerin sonuna doğru Adi Shamir ve Eli Biham tarafından tanıtılan direfansiyel kriptanaliz, blok cipherlara (özellikle DES 'e) karşı bu fikri (korelasyon fikrini) tam anlamıyla uygulayan ilk saldırıdır. Sonraları Mitsuru Matsui DES 'e karşı daha etkili olan lineer kriptanalizi ileri sürmüştür. Sonraqdan benzer fikirlere sahip saldırı çeşitleri geliştirilmiştir.
Bu konuya ilişkin belki de en iyi tanıtım EUROCRYPT ve CRYPTO 'nun 190 lı yıllar boyuncaki sayılarında verilmiştir.Çalışmaya, Mitsuru Matsui'nin DES'in lineer kriptanalizi hakkındaki tartışması ile, veya Lars Knudsen 'in budanmış diferansiyeller fikri ile (örneğin, IDEA kriptanalizi) başlayabilirsiniz. Ayrıca Eli Biham ve Adi Shamir'in DES'İn kriptanalizi hakkındaki kitabı bu konu üzerine "klasik" bir çalışma olarak görülebilir.
Korelasyon fikri kriptografide temeldir ve pek çok araştırmacı bu tip saldırılar karşısında güvenliği sağlamaya çalışmışlardır. örneğin, Knoudsen ve Nyberg diferansiyel kriptanalize karşı güvenliği sağlamak üzerine çalışmışlardır.

ŞİFRELEME ALGORİTMALARININ ANALİZİ

Şifreleme algoritmalarının analizi iki analiz yaklaşımından meydana gelir. Bunların birincisi performans analizidir ki bir algoritmanın zaman karmaşıklığının ölçülmesi olarak belirlenmiştir. Diğeri ise şifreleme algoritmalarının saldırılara karşı dayanıklılığının araştırılması bilimi olan kriptanalizdir.

IV.1. ŞİFRELEME ALGORİTMALARININ PERFORMANS ANALİZLERİ

Şifreleme algoritmalarının performansı çok tartışılan bir konudur. Bu tartışmaların çoğu C ve assembler yürütmeleri üzerine odaklanmıştır [36]. Bu çalışmada ise kodlar delphi içerisine gömülü assembler satırlarıyla yapılmıştır.

Bu bölümde DES, RC5, MD5, RSA şifreleme ve hash algoritmalarını delphide geliştirilmiş bağımsız fonksiyonlarla yürüterek analiz ettik. Farklı platformlar üzerindeki performans ölçüm sonuçlarını verdik.

IV.1.1. Performans ölçüm yaklaşımı

Burada tartışılan ve değerlendirilen bütün şifreleme algoritmaları Delphi'de ayrı ayrı fonksiyonlar halinde kodlanmıştır. Performans değerlendirmeleri ise daha

çok algoritmaların şifreleme, çözme ve hash özeti hesaplama hızları yani zaman karmaşıklıkları üzerine yoğunlaşmıştır. Hafıza karmaşıklığı değerlendirilmemiştir.

Burada kullanılan performans analizi yöntemi ise daha önceki bölümlerde açıklanan deneysel analiz yöntemidir. Ki bu yöntem karmaşık algoritmaların değerlendirilmesinde tercih edilen daha güvenilir bir yöntemdir[11].

IV.1.2. Performans değerlerinin bağımlılıkları

Algoritmaların performansı ölçülürken algoritmanın harcadığı zaman yerine bu algoritmayı işlemcinin kaç cycle da gerçekleştirdiği ölçülerek işlemci frekansından bağımsız bir değer elde edilmiştir. Fakat işlemcilerin mimarisine göre bu cycle sayısı değişebilir. Bunun için algoritmalar yaygın kullanılan işlemcilere sahip bilgisayarlarda çalıştırılarak ayrı ayrı değerler elde edilmiştir.

IV.1.3. Değerlendirme parametreleri

Yapılan ölçmelerde şu üç kriter değerlendirilmiştir. Şifreleme hızı, çözme hızı ve hash değeri hesaplama hızı.

IV.1.4. Değerlendirme platformları

Performans ölçüm sonuçları yaygın kullanılan üç farklı platformda yapılmıştır. Bunlar Celeron, Pentium IV ve AMD işlemcili 256 MB Rame sahip Windows XP işletim sistemi kurulu bilgisayarlardır.

IV.1.5. Performans analizi

Şifreleme algoritmalarının performans analizlerini yapmadan önce performans analizi yöntemi hakkında bazı genel konuları belirtmeye ihtiyaç vardır.

IV.1.5.a. Performans ölçme

Performansı ölçülen bütün algoritmaları Delphi'de kodlanmıştır. Hatta bir çok matematiksel dönüşümler de pascal kodları arasında assembler satırları kullanarak kodlanmıştır. Bu kodlar bir yapılan simulator içerisinde kullanılarak algoritmaların istenilen verileri şifrelemesi ve çözmesi sağlanmıştır. Bir algoritmanın performansını ölçmek için algoritmanın şifreleme ile çözme işlemlerinden önce ve sonra işlemcinin saat sayacının değerleri alınır. Performans sonucu bu iki değerin farkını hesaplayarak elde edilir. Bu sonuç işlemcinin şifreleme ve çözme işlemi için harcadığı cycle

sayısıdır. Algoritmanın hızı, cycle sayısını işlenen bilgi miktarına bölerek elde edilir. (cycle / KByte)

İşlemcinin cycle sayısını elde etmek için Windows API fonksiyonlarından PerfCounter adlı DLL fonksiyonu kullanılmıştır. Bu fonksiyon kernel32.dll dosyasından alınmıştır.

IV.1.5.b. Pentium III Mimarisinde Performans değerleri

Aşağıdaki tablodaki algoritmaların performans değerleri Pentium III 1 GHZ işlemcisine sahip bir bilgisayarla elde edilmiştir. Performans, her algoritmayla 1 KB lık dosya 30 defa şifrelenip çözülerek ve 32 KB lık dosyanın 30 defa özeti çıkarılarak ölçülmüştür. Bulunan sonuçlar 30 iterasyonun ortalamasıdır.

Tablo IV. 1 Hash Algoritmalarının Pentium III Mimarisindeki Hesaplama Hızları

Algoritma	Hash hızı Cycle / Kb
Message Digest 4	32,86
Message Digest 5	43,99
Secure Hash Algorithm	326,09
Secure Hash Algorithm 1	131,22
Ripe Message Digest 128	126,70
Ripe Message Digest 160	123,00
Ripe Message Digest 256	114,76
Ripe Message Digest 320	226,99
Haval-128	115,37
Haval-160	72,13
Haval-12	104,60
Haval-224	104,47
Haval-256	127,58
Sapphire II-128	181,97
Sapphire II-160	182,07
Sapphire II-12	274,17
Sapphire II-224	251,67
Sapphire II-256	314,92
Sapphire II-288	181,97
Sapphire II-320	182,73
Snefru-256	1.209,57
Square	377,50
Tiger	116,99
XOR-16	38,89
XOR-32	20,15
CRC-16 CCITT	42,34
CRC-16 Standard	44,72
CRC-32	24,09
Sample Hash	14,79

Tablo IV. 2 Şifreleme Algoritmalarının Pentium III Mimarisindeki Şifreleme / Çözme Hızları

Mod	ECB			OFB			CFB			CBC			CTR		
Algoritma	Şifreleme Cycle / Kb	Çözme Cycle / Kb	Ø Cycle / Kb	Şifreleme Cycle / Kb	Çözme Cycle / Kb	Ø Cycle / Kb	Şifreleme Cycle / Kb	Çözme Cycle / Kb	Ø Cycle / Kb	Şifreleme Cycle / Kb	Çözme Cycle / Kb	Ø Cycle / Kb	Şifreleme Cycle / Kb	Çözme Cycle / Kb	Ø Cycle / Kb
3Way	334,83	365,65	349,60	4.424,87	4.424,87	4.41,28	4.424,87	4.424,87	4.424,87	343,72	396,78	368,27	350,62	382,04	365,54
Blowfish	117,1	117,38	117,27	1.426,80	1.409,54	1.417,54	1.403,88	1.409,54	1.408,40	129,04	156,97	141,63	139,88	138,88	139,40
Gost	256,84	241,91	249,23	2.410,79	2.394,28	2.400,86	2.444,51	2.410,79	2.424,17	271,40	281,68	276,40	278,98	264,02	271,30
IDEA	371,88	377,50	374,67	3.495,65	4.424,87	3.910,12	3.495,65	3.566,99	3.534,53	383,72	426,30	403,84	393,65	398,14	395,88
Q128	127,72	69,93	90,38	2.240,80	2.212,44	2.227,95	2.226,53	2.212,44	2.220,87	137,1	95,69	112,75	142,22	84,93	106,35
SAFER-K40	294,25	299,29	296,85	2.987,73	2.987,73	2.985,18	3.177,86	3.177,86	3.180,75	305,83	338,40	321,32	375,88	386,69	381,20
SAFER-SK40	294,25	297,25	295,69	2.962,41	2.937,52	2.949,92	3.121,12	3.121,12	3.121,12	305,56	336,12	320,20	375,47	382,88	379,18
SAFER-K64	358,53	360,01	359,1	3.495,65	3.461,04	3.474,80	3.603,76	3.603,76	3.592,65	364,89	395,88	379,76	445,87	455,76	450,70
SAFER-SK64	352,38	362,24	357,25	3.603,76	3.603,76	3.618,68	4.161,49	3.641,30	3.875,44	374,27	400,88	387,20	445,87	455,76	450,64
SAFER-K128	565,64	575,89	570,90	5.296,44	5.296,44	5.312,54	5.548,65	5.548,65	5.548,65	592,48	616,52	604,26	707,62	723,74	715,88
SAFER-SK128	566,56	575,89	571,00	5.377,92	5.296,44	5.336,87	6.595,56	5.730,57	6.100,61	580,67	610,06	595,11	707,62	723,74	715,73
SCOP	33,60	30,10	31,75	4.262,99	4.262,99	4.273,41	4.262,99	4.262,99	4.257,79	38,73	39,37	39,05	41,93	38,70	40,25
Shark	187,94	254,04	216,06	1.680,60	1.672,56	1.678,18	1.705,1	1.618,36	1.658,28	201,25	290,10	237,67	207,70	713,40	321,71
Square	102,09	118,98	109,89	1.899,81	1.869,33	1.883,43	1.879,38	1.859,39	1.870,33	112,44	144,03	126,29	117,30	136,02	125,97
TEA	178,99	142,16	158,46	1.705,1	1.664,59	1.684,65	1.680,60	1.672,56	1.676,57	162,97	189,26	175,10	171,27	168,14	169,72
TEA extended	159,91	153,59	156,69	1.774,44	1.811,22	1.792,64	1.792,64	1.820,65	1.805,60	172,37	14,64	182,84	183,88	175,57	179,63
Twofish	152,18	155,64	153,91	2.865,29	2.962,41	2.915,47	2.841,99	2.937,52	2.898,55	160,06	181,22	170,00	167,34	170,85	169,07
Cast 128	147,1	121,29	132,99	1.506,75	1.468,76	1.487,51	1.456,52	1.438,54	1.448,07	160,50	163,04	161,79	167,34	144,57	155,13
Cast 256	225,96	223,51	224,74	4.017,99	4.017,99	4.013,37	4.017,99	4.017,99	4.022,61	233,51	248,98	240,95	240,09	241,25	240,66
DES Single 8byte	308,80	259,51	282,04	2.608,69	2.608,69	2.610,64	2.628,31	2.628,31	2.628,31	279,88	301,87	290,46	290,82	284,43	287,49
DES Double 8byte	4.923,45	1.116,82	1.822,55	6.722,40	6.595,56	6.620,55	6.595,56	6.473,42	6.546,16	773,37	796,28	783,95	783,78	814,84	798,64
DES Double 16byte	801,75	751,75	776,29	12.484,46	12.484,46	12.61,67	12.484,46	12.484,46	12.574,28	766,59	773,37	770,14	773,37	766,59	769,46
DES Triple 8byte	755,00	748,53	752,08	6.595,56	6.473,42	6.533,92	6.473,42	6.595,56	6.521,73	769,97	792,66	780,98	929,69	1.072,29	995,34
DES Triple 16byte	755,00	755,00	755,16	12.484,46	12.484,46	12.61,67	12.946,85	12.484,46	12.665,40	764,91	778,54	771,50	769,97	769,97	769,29
DES Triple 24byte	768,27	764,91	766,25	1.420,27	1.420,27	18.895,40	1.420,27	1.420,27	18.998,09	776,81	787,31	782,55	783,78	783,78	783,78
DESX	270,56	266,23	268,36	2.668,43	2.628,31	2.644,21	2.648,22	2.668,43	2.660,31	281,45	313,23	296,44	290,58	289,14	289,85
Diamond II	584,56	595,51	589,98	9.987,57	9.987,57	9.874,72	9.710,14	10.281,32	9.959,12	592,48	61,80	605,83	600,63	610,06	605,62

Diamond II Lite	696,34	685,42	690,84	6.026,98	5.924,83	5.975,47	6.026,98	5.924,83	5.944,98	706,1	725,24	715,88	716,32	716,32	716,62
FROG	480,83	334,51	394,45	15.889,32	16.645,95	16.035,09	15.889,32	15.889,32	15.675,56	486,18	353,45	409,42	498,67	352,03	412,81
Mars	206,84	215,51	211,06	3.718,78	3.718,78	3.726,71	3.718,78	3.718,78	3.706,95	215,25	239,10	226,56	223,65	231,65	227,60
Misty 1	417,64	411,25	414,27	3.841,37	3.841,37	3.824,56	3.841,37	3.799,62	3.820,38	429,44	451,63	440,42	444,17	438,60	441,54
NewDES	431,56	428,39	429,86	3.927,70	3.927,70	3.927,70	3.972,33	3.972,33	3.985,92	444,74	466,71	455,28	459,95	456,95	458,51
RC2	648,54	761,58	700,53	5.730,57	5.638,14	5.702,53	5.638,14	5.638,14	5.665,56	657,08	803,60	722,54	676,14	796,28	731,77
RC4	105,83	102,48	104,12	1.974,94	1.963,85	1.969,38	1.986,16	1.997,51	1.994,10	115,29	122,57	118,82	148,00	118,10	131,36
RC5	107,59	100,80	104,09	1.31,11	1.309,23	1.314,15	1.324,11	1.329,14	1.328,13	11,96	143,09	130,50	130,63	132,01	131,31
RC6	171,1	174,70	172,91	3.177,86	3.177,86	3.172,10	3.207,02	3.207,02	3.18,22	180,10	18,73	188,94	189,26	13,88	11,57
Rijndael	166,46	158,32	162,26	2.444,51	2.410,79	2.427,53	2.479,18	2.461,72	2.473,92	147,50	181,31	162,63	156,06	173,14	164,15
Sapphire II	185,54	174,96	180,08	6.242,23	6.242,23	6.253,40	6.242,23	6.473,42	6.321,25	17,27	12,07	14,65	18,17	188,34	13,16
Skipjack	532,87	537,79	535,32	4.788,56	4.788,56	4.788,56	4.788,56	4.788,56	4.762,47	541,96	578,75	559,84	559,30	560,20	559,39
Sample Cipher	61,77	53,30	57,22	7.768,11	7.768,11	7.837,78	7.768,11	7.768,11	7.837,78	67,17	62,27	64,62	70,96	62,74	66,60
RSA															

IV.1.5.c. AMD Athlon IV Mimarisinde Performans değerleri

Aşağıdaki tablodaki algoritmaların performans değerleri Amd Athlon IV 1 GHZ işlemcisine sahip bir bilgisayarla elde edilmiştir. Performans, her algoritmayla 1 KB lık dosya 30 defa şifrelenip çözülerek ve 32 KB lık dosyanın 30 defa özeti çıkarılarak ölçülmüştür. Bulunan sonuçlar 30 iterasyonun ortalamasıdır.

Tablo IV. 3 Hash Algoritmalarının AMD Athlon IV Mimarisindeki Hesaplama Hızları

Algoritma	Hash hızı Cycle / Kb
Message Digest 4	30,56
Message Digest 5	42,10
Secure Hash Algorithm	68,34
Secure Hash Algorithm 1	67,50
Ripe Message Digest 128	100,83
Ripe Message Digest 160	112,36
Ripe Message Digest 256	99,76
Ripe Message Digest 320	129,85
Haval-128	78,66
Haval-160	76,44
Haval-12	111,72
Haval-224	111,72
Haval-256	133,93
Sapphire II-128	309,90
Sapphire II-160	316,92
Sapphire II-12	312,11
Sapphire II-224	314,92
Sapphire II-256	314,36
Sapphire II-288	313,79
Sapphire II-320	316,63
Snefru-256	993,08
Square	325,78
Tiger	182,73
XOR-16	11,48
XOR-32	7,95
CRC-16 CCITT	24,90
CRC-16 Standard	22,91
CRC-32	24,56
Sample Hash	15,18

Tablo IV. 4 Şifreleme Algoritmalarının AMD Athlon IV Mimarisindeki Şifreleme / Çözme Hızları

Mod	ECB			OFB			CFB			CBC			CTR		
Algoritma	Şifreleme Cycle / Kb	Çözme Cycle / Kb	Ø Cycle / Kb	Şifreleme Cycle / Kb	Çözme Cycle / Kb	Ø Cycle / Kb	Şifreleme Cycle / Kb	Çözme Cycle / Kb	Ø Cycle / Kb	Şifreleme Cycle / Kb	Çözme Cycle / Kb	Ø Cycle / Kb	Şifreleme Cycle / Kb	Çözme Cycle / Kb	Ø Cycle / Kb
3Way	112,18	97,26	104,1	3.884,05	3.884,05	3.888,38	3.841,37	3.927,70	3.884,05	312,67	356,70	333,24	323,67	350,97	336,90
Blowfish	224,22	217,80	220,95	1.173,04	1.165,22	1.168,33	1.176,99	1.169,11	1.173,82	127,35	131,27	129,27	140,11	121,08	129,93
Gost	364,89	381,62	373,11	2.068,43	2.056,26	2.064,77	2.056,26	2.056,26	2.058,69	236,03	251,30	243,43	248,80	241,25	244,93
IDEA	116,21	66,50	84,59	3.461,04	3.566,99	3.523,84	3.236,71	3.297,78	3.273,08	368,35	41,65	392,37	386,69	407,89	396,92
Q128	562,00	537,79	549,98	2.032,35	1.997,51	2.014,78	1.997,51	1.997,51	2.000,94	127,35	88,21	104,21	135,28	83,97	103,62
SAFER-K40	535,32	545,34	539,87	4.315,62	4.369,56	4.326,30	4.539,80	4.481,60	4.504,70	548,77	586,52	567,29	561,10	560,20	560,47
SAFER-SK40	639,06	809,18	714,27	4.369,56	4.315,62	4.353,24	4.539,80	4.481,60	4.516,34	556,63	577,79	567,02	588,49	564,73	576,17
SAFER-K64	633,27	636,73	635,23	5.140,66	5.140,66	5.140,66	5.296,44	5.377,92	5.328,73	650,96	669,66	659,81	662,05	663,31	662,81
SAFER-SK64	1.040,37	1.046,60	1.043,48	5.140,66	5.140,66	5.133,11	5.461,95	5.548,65	5.470,50	647,34	672,24	659,93	660,80	659,56	660,18
SAFER-K128	1.040,37	1.049,74	1.044,41	8.129,42	7.944,66	8.054,49	8.525,97	8.525,97	8.546,82	1.059,29	1.075,58	1.067,70	1.069,01	1.069,01	1.068,68
SAFER-SK128	30,73	1,87	24,13	8.129,42	8.322,97	8.225,06	8.525,97	8.525,97	8.484,59	1.059,29	1.075,58	1.067,70	1.072,29	1.069,01	1.070,32
SCOP	177,90	229,37	200,39	2.377,99	2.377,99	2.381,23	2.444,51	2.394,28	2.415,79	42,40	27,73	33,53	49,93	30,50	37,87
Shark	112,18	113,38	112,78	1.450,48	1.409,54	1.430,30	1.444,48	1.421,00	1.433,82	177,62	261,85	211,63	189,47	252,39	216,41
Square	128,99	124,84	126,88	1.899,81	1.879,38	1.889,54	1.879,38	1.899,81	1.887,50	120,58	134,66	127,23	130,14	129,28	129,73
TEA	156,76	146,57	151,48	1.324,11	1.309,23	1.317,62	1.314,15	1.309,23	1.309,72	140,05	158,39	148,65	157,25	148,18	152,57
TEA extended	203,47	14,20	18,74	1.618,36	1.546,75	1.583,17	1.546,75	1.546,75	1.548,12	168,38	180,37	174,18	178,35	170,1	174,16
Twofish	137,30	117,86	126,84	3.427,11	3.495,65	3.457,62	3.461,04	3.495,65	3.464,47	214,06	217,12	215,59	220,96	211,35	216,07
Cast 128	212,76	16,16	204,16	1.299,50	1.339,33	1.317,62	1.354,90	1.324,11	1.339,33	148,50	151,33	149,89	160,13	141,41	150,18
Cast 256	243,77	254,04	248,78	3.461,04	3.461,04	3.461,04	3.530,96	3.461,04	3.488,67	214,59	216,05	215,32	225,38	211,22	218,07
DES Single 8byte	672,24	657,08	664,07	2.18,52	2.18,52	2.19,91	2.184,78	2.18,52	2.11,63	251,12	273,31	261,79	260,48	259,71	260,09
DES Double 8byte	668,38	655,84	661,80	5.730,57	5.730,57	5.739,98	5.730,57	5.826,08	5.787,50	686,77	689,48	687,58	697,73	690,84	694,27
DES Double 16byte	668,38	657,08	663,06	10.923,90	10.923,90	11.027,29	11.276,29	10.923,90	11.062,18	678,77	696,34	687,31	692,21	739,04	715,00
DES Triple 8byte	668,38	662,05	665,20	5.826,08	5.730,57	5.797,10	5.924,83	5.826,08	5.884,93	686,77	709,06	698,01	699,13	696,34	697,87
DES Triple 16byte	776,81	664,57	715,88	10.923,90	11.276,29	11.062,18	11.276,29	10.923,90	11.062,18	682,74	696,34	689,61	692,21	681,41	687,04
DES Triple 24byte	246,35	236,83	241,55	16.645,95	16.645,95	16.334,81	16.645,95	16.645,95	16.411,50	684,08	697,73	690,70	693,58	685,42	689,75

DESX	743,76	731,31	737,48	2.269,90	2.240,80	2.252,35	2.269,90	2.284,74	2.281,76	257,79	270,14	263,82	289,85	258,36	273,23
Diamond II	694,96	722,24	708,77	12.053,96	12.053,96	12.053,96	12.053,96	12.053,96	12.137,67	755,00	785,54	769,80	763,24	750,14	756,47
Diamond II Lite	617,61	358,90	453,98	5.826,08	5.826,08	5.855,36	5.924,83	5.826,08	5.865,18	711,94	846,40	773,03	723,74	743,76	733,76
FROG	223,65	211,99	217,69	1.420,27	1.420,27	1.528,77	1.420,27	1.420,27	1.528,77	620,90	375,07	467,65	701,94	377,09	490,69
Mars	416,64	410,29	413,54	3.841,37	3.758,76	3.799,62	3.758,76	3.718,78	3.746,68	232,11	233,98	233,06	253,68	229,67	241,05
Misty 1	323,07	318,37	320,73	3.641,30	3.679,63	3.652,72	3.679,63	3.603,76	3.641,30	440,81	440,81	440,76	449,89	434,78	442,26
NewDES	463,61	410,29	435,27	2.841,99	2.81,07	2.828,20	2.81,07	2.841,99	2.835,08	326,70	355,61	340,64	338,40	342,71	340,51
RC2	179,72	178,81	179,26	4.064,71	4.112,53	4.078,94	4.064,71	4.017,99	4.050,58	467,33	431,56	448,62	477,55	421,16	447,76
RC4	113,75	91,20	101,24	3.149,23	3.121,12	3.135,11	3.093,50	3.093,50	3.101,73	188,44	200,09	14,07	19,98	14,96	17,45
RC5	13,02	173,14	182,56	1.201,25	1.189,00	1.14,69	1.201,25	1.189,00	1.13,46	124,40	124,18	124,29	134,97	123,65	129,07
RC6	161,84	172,80	167,15	3.266,96	3.361,20	3.316,56	3.236,71	3.329,1	3.276,15	200,67	14,64	17,61	211,47	189,26	19,79
Rijndael	349,91	330,09	339,71	2.730,98	2.709,81	2.718,23	2.730,98	2.774,32	2.754,65	179,82	16,27	187,70	182,45	12,28	187,18
Sapphire II	508,83	493,04	500,88	10.923,90	10.923,90	11.062,18	11.276,29	11.276,29	11.204,00	357,79	346,45	351,96	361,87	344,74	353,20
Skipjack	63,13	42,60	50,87	4.424,87	4.369,56	4.386,01	4.424,87	4.369,56	4.408,13	521,74	536,14	528,84	549,63	521,74	535,24
Sample Cipher	62,40	48,44	54,54	6.991,30	6.854,21	6.881,20	6.854,21	6.854,21	6.867,68	73,87	51,00	60,34	77,41	57,20	65,78
RSA															

IV.1.5.d. Celeron Mimarisinde Performans değerleri

Aşağıdaki tablodaki algoritmaların performans değerleri Celeron 1.7 GHZ işlemcisine sahip bir bilgisayarla elde edilmiştir. Performans, her algoritmayla 1 KB lık dosya 30 defa şifrelenip çözülerek ve 32 KB lık dosyanın 30 defa özeti çıkarılarak ölçülmüştür. Bulunan sonuçlar 30 iterasyonun ortalamasıdır.

Tablo IV. 5 Hash Algoritmalarının Celeron Mimarisindeki Hesaplama Hızları

Algoritma	Hash hızı Cycle / Kb
Message Digest 4	28,88
Message Digest 5	35,66
Secure Hash Algorithm	71,31
Secure Hash Algorithm 1	63,22
Ripe Message Digest 128	84,50
Ripe Message Digest 160	80,1
Ripe Message Digest 256	74,60
Ripe Message Digest 320	102,57
Haval-128	118,18
Haval-160	51,63
Haval-12	68,95
Haval-224	71,22
Haval-256	83,71
Sapphire II-128	114,35
Sapphire II-160	113,90
Sapphire II-12	112,58
Sapphire II-224	112,80
Sapphire II-256	118,78
Sapphire II-288	112,26
Sapphire II-320	113,68
Snefru-256	654,62
Square	218,21
Tiger	82,23
XOR-16	31,07
XOR-32	20,49
CRC-16 CCITT	34,08
CRC-16 Standard	21,88
CRC-32	23,65
Sample Hash	17,68

Tablo IV. 6 Şifreleme Algoritmalarının Celeron Mimarisindeki Şifreleme / Çözme Hızları

Mod	ECB			OFB			CFB			CBC			CTR		
Algoritma	Şifreleme Cycle / Kb	Çözme Cycle / Kb	Ø Cycle / Kb	Şifreleme Cycle / Kb	Çözme Cycle / Kb	Ø Cycle / Kb	Şifreleme Cycle / Kb	Çözme Cycle / Kb	Ø Cycle / Kb	Şifreleme Cycle / Kb	Çözme Cycle / Kb	Ø Cycle / Kb	Şifreleme Cycle / Kb	Çözme Cycle / Kb	Ø Cycle / Kb
3Way	211,09	222,37	216,62	2.589,37	2.533,08	2.560,92	2.589,37	2.533,08	2.560,92	210,71	243,26	225,90	217,39	233,51	225,16
Blowfish	88,01	74,55	80,74	882,74	838,29	861,21	876,10	850,52	862,48	96,94	96,86	96,88	105,54	88,41	96,24
Gost	182,35	148,81	163,82	1.474,96	1.421,00	1.442,10	1.474,96	1.456,52	1.465,68	209,20	168,63	186,67	19,18	161,16	178,22
IDEA	233,04	223,65	228,29	1.974,94	1.974,94	1.985,04	1.942,03	1.942,03	1.948,52	239,76	246,87	243,11	248,45	236,35	242,25
Q128	104,41	45,97	63,84	1.324,11	1.280,46	1.301,92	2.913,04	1.294,69	1.787,14	115,60	62,14	80,83	11,63	57,15	77,33
SAFER-K40	223,22	216,58	21,69	1.942,03	1.849,55	1.903,95	1.849,55	1.820,65	1.840,78	201,25	208,45	204,68	207,70	15,83	201,59
SAFER-SK40	221,10	216,58	218,86	1.879,38	1.849,55	1.858,40	1.910,1	1.849,55	1.885,46	19,18	205,87	202,51	205,87	15,51	200,62
SAFER-K64	263,03	267,25	265,18	2.118,58	2.080,74	2.099,49	2.157,81	2.157,81	2.165,83	233,98	239,76	236,83	241,25	230,28	235,64
SAFER-SK64	262,44	259,51	260,91	2.18,52	2.157,81	2.177,98	2.240,80	2.18,52	2.223,70	240,25	245,31	242,96	246,87	232,11	239,36
SAFER-K128	410,29	416,15	413,64	3.236,71	3.236,71	3.245,73	3.427,11	3.329,1	3.367,68	359,63	368,74	364,13	364,13	356,34	359,97
SAFER-SK128	410,29	429,97	41,60	3.427,11	3.329,1	3.367,68	3.427,11	3.329,1	3.367,68	363,00	379,55	370,97	363,00	364,13	363,56
SCOP	41,44	23,06	29,63	2.533,08	2.533,08	2.516,67	2.533,08	2.479,18	2.527,59	47,83	29,90	36,80	52,68	29,60	37,90
Shark	222,79	274,82	246,03	1.040,37	947,33	993,36	1.031,17	995,91	1.011,47	11,65	300,31	233,93	205,14	317,50	249,03
Square	98,83	109,82	104,07	1.153,68	1.109,73	1.129,09	1.153,68	1.099,26	1.126,90	107,39	126,93	116,35	111,29	120,25	115,60
TEA	110,24	88,81	98,38	1.013,23	995,91	1.003,63	987,47	987,47	984,97	116,76	113,13	114,90	123,70	107,69	115,16
TEA extended	118,54	94,20	104,98	1.069,01	1.069,01	1.067,05	1.059,29	1.040,37	1.052,59	126,24	118,18	122,10	136,76	110,03	121,99
Twofish	108,09	96,30	101,85	1.713,55	1.739,13	1.723,69	1.688,72	1.765,48	1.728,81	114,80	112,04	113,41	121,50	106,61	113,55
Cast 128	123,43	77,53	95,23	896,32	832,30	861,85	917,49	844,36	876,76	132,26	101,15	114,65	142,62	94,35	113,57
Cast 256	158,75	142,45	150,12	2.377,99	2.377,99	2.387,74	2.427,53	2.427,53	2.422,49	184,96	156,62	169,61	170,85	150,16	159,90
DES Single 8byte	235,87	163,20	12,92	1.533,18	1.513,27	1.517,21	1.533,18	1.493,87	1.513,27	188,55	200,90	14,59	15,51	173,91	184,05
DES Double 8byte	462,39	448,16	455,52	3.884,06	3.884,06	3.845,60	3.884,06	3.884,06	3.845,60	471,75	471,75	471,75	487,54	477,55	482,09
DES Double 16byte	460,56	448,16	454,28	7.282,60	7.282,60	7.374,79	7.282,60	7.282,60	7.328,41	471,75	473,67	472,13	485,51	513,31	499,24
DES Triple 8byte	460,56	444,74	452,34	3.884,06	3.884,06	3.871,15	3.884,06	3.758,76	3.820,38	477,55	471,75	474,63	479,51	462,39	471,18
DES Triple 16byte	504,42	441,37	470,22	7.282,60	7.282,60	7.421,76	7.282,60	7.282,60	7.374,79	520,1	487,54	503,77	475,60	466,09	471,56
DES Triple	464,23	473,67	468,33	10.592,88	10.592,88	10.889,88	10.592,88	10.592,88	10.889,88	477,55	462,39	469,85	511,06	462,39	485,51

24byte															
DESX	184,96	174,43	179,51	1.596,1	1.574,62	1.581,03	1.574,62	1.533,18	1.555,70	13,24	188,24	10,71	202,65	232,11	216,38
Diamond II	406,00	41,14	412,76	6.132,72	6.132,72	6.165,17	6.132,72	6.132,72	6.132,72	417,64	438,05	427,13	443,05	436,41	439,21
Diamond II Lite	448,16	444,74	446,61	3.641,30	3.641,30	3.687,39	3.758,76	3.641,30	3.699,10	460,56	460,56	460,20	467,96	467,96	467,96
FROG	295,74	211,86	246,82	8.963,21	8.963,21	9.174,94	8.963,21	8.963,21	9.174,94	309,08	226,70	261,67	311,56	220,27	257,96
Mars	145,47	134,24	139,63	2.240,80	2.18,52	2.21,46	2.240,80	2.18,52	2.223,70	153,72	153,52	153,64	159,84	146,38	152,82
Misty 1	272,25	248,45	259,80	2.284,74	2.284,74	2.293,73	2.284,74	2.330,43	2.311,94	282,13	272,25	276,84	288,42	264,82	276,18
NewDES	274,17	256,09	264,88	2.330,43	2.284,74	2.316,53	2.330,43	2.330,43	2.339,79	286,29	284,20	285,59	294,99	288,42	291,81
RC2	374,67	467,96	416,60	3.149,23	3.066,36	3.107,24	3.149,23	3.066,36	3.098,98	385,83	491,65	431,88	399,05	491,65	440,20
RC4	79,43	66,28	72,26	1.226,54	1.201,25	1.211,24	1.252,92	1.213,77	1.230,43	89,84	79,92	84,60	96,22	75,37	84,52
RC5	87,61	64,70	74,43	826,39	820,58	824,06	826,39	863,12	844,97	96,54	88,95	92,58	103,48	83,65	92,53
RC6	129,47	109,72	118,79	1.974,94	1.974,94	1.978,30	1.974,94	1.910,1	1.948,52	136,76	128,05	132,23	139,55	120,37	129,30
Rijndael	113,24	126,79	11,61	1.493,87	1.438,54	1.465,68	1.533,18	1.438,54	1.484,35	121,50	145,47	132,40	125,43	140,22	132,40
Sapphire II	127,35	112,47	11,46	3.641,30	3.641,30	3.618,69	3.641,30	3.641,30	3.618,69	133,32	125,29	129,22	130,48	121,25	125,72
Skipjack	330,09	315,78	322,77	2.841,99	2.841,99	2.848,94	2.774,33	2.774,33	2.774,33	339,71	339,71	339,42	344,74	329,16	337,06
Sample Cipher	53,04	36,84	43,47	4.660,87	4.660,87	4.660,87	4.660,87	4.660,87	4.642,30	59,33	43,43	50,15	64,52	43,30	51,83
RSA															

IV.1.6. Sonuçların Yorumlanması

Bu çalışmada şifreleme algoritmalarının ve hash fonksiyonlarının zaman karmaşıklığını analiz ettik. Bu sonuçlar işlem hızı performans sıralamasının büyükten küçüğe doğru hash algoritmaları, stream şifreleme algoritmaları, blok şifreleme algoritmaları, asimetrik şifreleme algoritmaları şeklinde olduğunu gösterir. Burada gösterilen performans sonuçlarının algoritmaların güvenirliğiyle bir ilgisi yoktur.

IV.2. KRİPTOĞRAFİK ALGORİTMALARIN KRİPTANALİZLERİ

Kriptanaliz daha önce belirtildiği gibi kriptoğrafik tekniklerin güvenirliğini ölçmek üzere onların açıklarını araştıran matematik ağırlıklı bir bilim dalıdır. Burada, bölüm IV'teki şifreleme algoritmalarının kriptanalizleri yapılarak bir şifreleme algoritmasının kriptanalizinin nasıl yapıldığı hakkında bir fikir verilecektir.

IV.2.1. Sezar Şifresinin Kriptanalizi

Tablo IV. 7 Sezar Şifresinin Kriptanalizinde Brute Force Tablosu

N	Ç	Ö	J	K	C	D	T	P	C	Ö	Ç	T
1	C	O	İ	J	B	Ç	Ş	Ö	B	O	C	Ş
2	B	N	I	İ	A	C	S	O	A	N	B	S
3	A	M	H	I	Z	B	R	N	Z	M	A	R
4	Z	L	Ğ	H	Y	A	Q	M	Y	L	Z	Q
5	Y	K	G	Ğ	X	Z	P	L	X	K	Y	P
6	X	J	F	G	W	Y	Ö	K	W	J	X	Ö
7	W	İ	E	F	V	X	O	J	V	İ	W	O
8	V	I	D	E	Ü	W	N	İ	Ü	I	V	N
9	Ü	H	Ç	D	U	V	M	I	U	H	Ü	M
10	U	Ğ	C	Ç	T	Ü	L	H	T	Ğ	U	L
11	T	G	B	C	Ş	U	K	Ğ	Ş	G	T	K
12	Ş	F	A	B	S	T	J	G	S	F	Ş	J
13	S	E	Z	A	R	Ş	İ	F	R	E	S	İ
14	R	D	Y	Z	Q	S	I	E	Q	D	R	I
15	Q	Ç	X	Y	P	R	H	D	P	Ç	Q	H
16	P	C	W	X	Ö	Q	Ğ	Ç	Ö	C	P	Ğ
17	Ö	B	V	W	O	P	G	C	O	B	Ö	G
18	O	A	Ü	V	N	Ö	F	B	N	A	O	F
1	N	Z	U	Ü	M	O	E	A	M	Z	N	E
20	M	Y	T	U	L	N	D	Z	L	Y	M	D
21	L	X	Ş	T	K	M	Ç	Y	K	X	L	Ç
22	K	W	S	Ş	J	L	C	X	J	W	K	C
23	J	V	R	S	İ	K	B	W	İ	V	J	B
24	İ	Ü	Q	R	I	J	A	V	I	Ü	İ	A
25	I	U	P	Q	H	İ	Z	Ü	H	U	I	Z
26	H	T	O	P	Ğ	I	Y	U	Ğ	T	H	Y
27	Ğ	Ş	O	Ö	G	H	X	T	G	Ş	Ğ	X
28	G	S	N	O	F	Ğ	W	Ş	F	S	G	W
29	F	R	M	N	E	G	V	S	E	R	F	V
30	E	Q	L	M	D	F	Ü	R	D	Q	E	Ü
31	D	P	K	L	Ç	E	U	Q	Ç	P	D	U

Sezar şifresi brute force adı verilen deneme yanılma yöntemiyle kolayca kırılabilir. Bu yöntemde şifreli metinden alınan bir örnek metni sırayla 1den 26'ya kadar kaydırarak 26 farklı açık metin elde edilir. Bunlardan doğru olanı bulunarak kaydırma miktarı tespit edilir ve tüm metin çözülür.

IV.2.2. XOR Şifrelemenin kriptanalizi

XOR şifrelemenin kriptanalizi ise; sıfırlardan oluşan bir diziyi herhangi bir şifreli metin ile XOR'ladığımız zaman sonucun anahtarın kendisi olmasıdır.

IV.2.3. DES in Kriptanalizi

DES üzerine yaygın birkaç kriptanalitik saldırı vardır;

- BRUTE FORCE saldırısı
- Diferansiyel kriptanaliz
- Lineer Kriptanaliz

1) BRUTE FORCE Saldırısı

Bir DES anahtarını çözmek üzerine en basit saldırı Brute Force saldırısıdır. DES algoritmasına yapılan Brute Force saldırısı, 56 bit gibi küçük anahtar uzunluğu ve hesaplama gücü artan bilgisayarlardan dolayı etkilidir. 190'ların ortalarına kadar yüksek hızlı bilgisayarların fiyatlarının çok aşırı ve satın almaya güç yetmeyecek derecede olmasından dolayı hacker'ların kapasitelerinin üzerindeydi. Hesaplama alanındaki çok büyük gelişmeler ve yüksek performanslı bilgisayarların daha ucuz ve satın alınabilir olmasından dolayı, bugün genel amaçlı PC'ler başarılı bir şekilde Brute Force saldırısını kullanabilirler. Çoğu hackerlar, kırma işlemini ucuzlatmak ve hızlandırmak için bugün FPGA (Field Programmable Gate Array) ve ASIC (Applicadion-Spasific Integrated Circuits) teknolojileri gibi güçlü kripto işlemci teknolojileri kullanmaktadırlar.

Brute Force saldırısıyla, bir şifreyi muhtemel bütün anahtarları deneyerek kırabilirsiniz. Fakat, bir şifreyi kırmak için geçen süre, anahtar uzunluğuyla doğru orantılıdır. Brute Force saldırısında anahtarlar rastgele üretilip, doğru anahtarı buluncaya kadar şifreli metine uygulanabilir. Dolayısıyla şifreleme anahtarı uzunluğu,bir anahtar seçerken göz önüne alınması gereken temel bir faktördür. Örneğin, 32-bit uzunluğunda bir anahtar durumunla şifreyi kırmak için gereken adımların sayısı yaklaşık 2^{32}dir. Benzer şekilde 40 bit anahtar 2^{40} adım gerektirir. Bu kişisel bilgisayarı karşısında oturan bir kişinin bir haftada üstesinden gelebileceği bir şeydir. 56-bit anahtarı profesyoneller, devletler tarafından sağlanan özel

donanımlar kullanarak birkaç ayda kırılabildiği bilinmektedir. Bugün 128-bit şifreleme, mesajları şifrelemenin en emniyetli ve en güvenilir yolu olarak göz önüne alınabilir.

1 Ocak 199 da dünyanın her yerinden bilgisayar hastalarından oluşan bir grup DES ile şifrelenmiş bir metnin şifresini kırmak için ortak bir çalışma başlattılar. Sonuç olarak 22 saat 15 dakikada anahtarı buldular. Bu koalisyon Distrubuted.Net olarak bilinmektedir[20]. Bunun için dünya çapındaki ağlardan internet üzerinde yaklaşık 100 bin PC çalıştı. DES kırma makinesi bu amaç için özel olarak tasarlanmıştı. Brute Force hakkında daha fazla bilgi için RFCs 2228 ve 2557[*]

2) Diferansiyel Kriptanaliz

Diferansiyel kriptanaliz saldırısı, açık metinlerinde bazı özel farklılıklara sahip şifreli metin çiftlerindeki ilişkiye bakar. Aynı anahtarla şifrelendiğinde DESin farklı çevrimlerinden yayılan bu farklılıkları analiz eder. Bu teknik sabit bir farka sahip açık metin çiftlerini seçer. 2 açık metin belirli fark şartlarını sağlayıncaya kadar rastgele seçilir ve şifreli metinlerin sonuçlarında farklı ihtimaller farklı anahtarlara tahsis edilir. Çok çok fazla açık metin çiftleri analiz edildiğinde aday anahtarlardan bir tanesi ihtimali en yüksek olarak ortaya çıkar.

Diferansiyel kriptanaliz hakkında daha fazla bilgi için RFCs 2144[*]

3) Doğrusal Kriptanaliz

Doğrusal kriptanaliz saldırısı 193 te Mitsuri Matsui tarafından icat edilmiştir[20]. Bu metot şu kavram üzerine dayalıdır. Eğer siz açık metnin bitlerinin bazılarını beraber XORlasanız, şifreli metnin bitlerinin bazılarını da XORlasanız, sonra sonuçları XORlasanız. Anahtarın bitlerinin bazılarını XORlayan tek bir bit elde edersiniz.

Büyük miktarda böyle şifreli metin - açık metin çiftleri anahtar bitlerinin değerini tahmin etmek için kullanılır. Temel veriler ne kadar çok olursa tahmin o kadar çok güvenilir olur.

IV.2.4. RSA'ya muhtemel saldırılar

RSA geniş bir şekilde yaygınlaşmasına rağmen bazı zayıf noktaları vardır. RSA'nın dayanabildiği bazı bilinen saldırılar şunlardır:

a) Açık anahtarın çarpanlarına ayrılması:

RSA halen güvenli görünmektedir. 20 yıldır zayıflıkları araştırılmasına rağmen hayatta kalmaktadır. Ve dünya çapında yaygın kullanılmaktadır. RSA için değerlendirilen bu saldırıda açık anahtar çarpanlarına ayrılır. Eğer bu başarılırsa açık anahtar ile şifrelenmiş bütün mesajlar çözülebilir.

b) Devir saldırısı:

Bu saldırıda şifreli metin orjinal metin görününceye kadar tekrar tekrar çözülür. Bu devrin tekrarlanması sayısının artması şifreli metni çözebilir. Bu metot çok yavaştır ve çok büyük anahtar için pratik saldırı değil. RSA'nın bütün bu zayıflıklarına rağmen şifreleme için hala bir endüstriyel de facto standart olarak sayılmaktadır. Özellikle internet üzerinde iletilen veriler için…

c) Zamanlama saldırısı:

RSA' nın çözme işleminin yüksek hesaplama maliyetini kendi çıkarı için kullanır. Saldıran, çözme işlemi için istatistiksel zaman analizleri gerçekleştirir.4 ayrı şifreli metnin çözülme işleminin zaman sapması tahmin için yeterlidir.

IV.2.5. RC4'ün kriptanalizi

RC4'e brute force saldırısı 256 bayta kadar değişen anahtar uzunluğundan dolayı pratik değildir. Fakat RC4'ün anahtar uzunluğunun küçük seçilmesi algoritmayı güvensiz yapacaktır.

RSA Data Security firması RC4'ün diferansiyel ve doğrusal kriptanalize karşı koyabileceğini iddia etmektedir[37]. Amerika RC4'ün 40 bit'e kadar anahtar uzunluğu ile ihracına izin vermektedir. 40 bit uzunluğu olan RC4 brute force saldırısına karşı dayanıksızdır. Bununla beraber geçmiş araştırmalar RC4'ün anahtar sıralama algoritmasında bazı zayıflıklar bulunduğunu haber vermektedir[38].

2001 yılının başlarında RC4'e dayalı bir algoritma olan WEP'e karşı bir saldırı keşfedildi[38].

4 ila 6 milyon paket elde edilerek yaklaşık 15 saniyede gizli anahtarın bulunması mümkün olmaktadır. Saldırı basit olarak bir bilinen açık metin saldırısıdır ve RC4'ün şu özelliklerini kullanır.

- Farklı başlangıç vektörü ve aynı anahtar ile üretilen anahtar akımı ile şifrelenmiş birçok mesaj
- Şifrelenmiş mesajlar için açık metnin ilk iki sekizlisi ile başlangıç vektörleri bilinmektedir.

IV.2.6. Affine Şifresinin Kriptanalizi

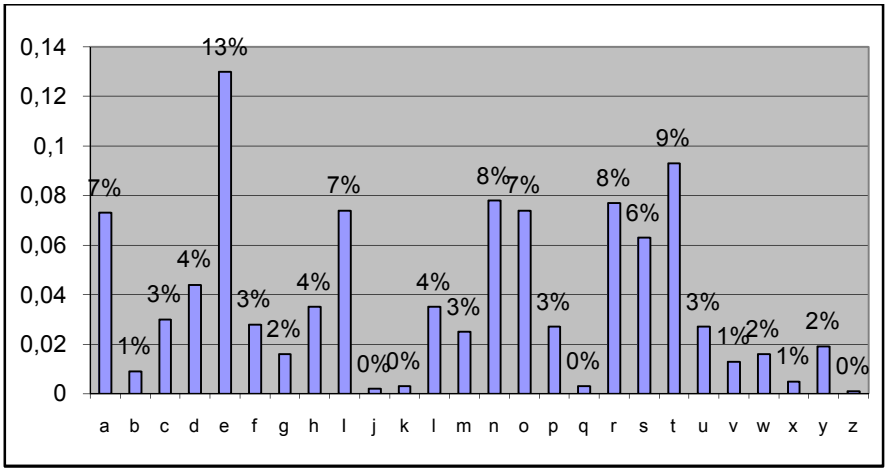
Affine şifresini kırmak için harflerin kullanım istatistiğiyle saldırı yöntemi kullanılır.

Bu yöntemde bir dildeki harflerin tekrar frekanslarıyla şifreli metindeki harflerin tekrar frekansları karşılaştırılır. Bazı istatistiksel analizlerle hangi harfin yerine hangi harf kullanıldığı tahmin edilebilir.

Aşağıdaki şifreli metnin affine yöntemiyle şifrelendiğini farz edelim.

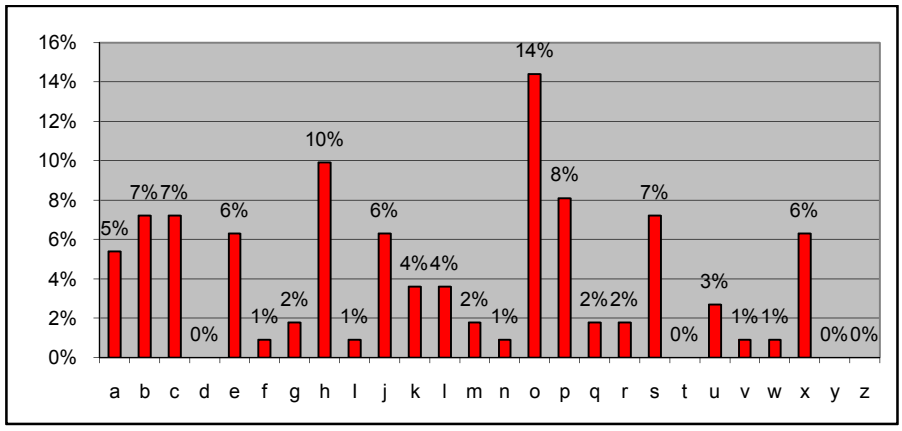
"hxo rboexmop fbapu c jahhjo gpsqjoluo ap cpl hxo eopasbe hcgo pspo skh es ah ciikmkjchoe hxbskux hxo wocbe jsqojj vboealoph sr xcbncbl"

Bu şifreli metindeki tekrar eden harf frekanslarını hesaplarsak Şekil IV.1'deki grafik elde edilir.



Şekil IV. 1 Şifreli Metnin Harf Kullanım Grafiği.

İngilizce bir metindeki harf kullanım frekansları ise Şekil IV.1'deki grafikte gösterilmiştir.



Şekil IV. 2 İngilizce Metinlerin Genel Harf Kullanım Grafiği.

İlk grafikte görüldüğü gibi şifreli metinde en çok tekrar eden harf o harfidir ve sıra değeri 14 tür. Bu iki grafik incelendiğinde İngilizcide en çok kullanılan e harfi şifrelendiğinde o harfi elde ediliyor. E'nin sıra değeri ise 4 tür.

C = a*P + b MOD 26 fonksiyonuna göre şifrelendiğini bildiğimizden bu değerlere göre fonksiyonumuzdan şu denklem elde edilir.

14 = a*4 + b MOD 26

Amacımız a ve b değerlerinin bulunmasıdır. Ve bunun iki yöntemi vardır.

a) I. Yöntem

Bütün a ve b değerlerini deneyerek bulmaktır. Bu yönteme brute force saldırısı denilir. Literatürde bu yöntem tercih edilmemektedir.

b) II. Yöntem

İki bilinmeyenli bir denklem gibi çözmek. Bu yöntemde iki tane denklem elde etmek gerektiğinden dolayı diğer harf frekanslarını kullanarak başka bir denklem daha elde etmemiz gerekir.

O harfinden sonra şifreli metinde en çok kullanılan harf h'dir ve İngilizce harf kullanım frekans grafiğinde, şifreli metindeki h harfini elde etmek için kullanılacak açık metin harf adayları t,n,o,r,i,a,s'dir. Bunların her birisi tek tek h'yi elde etmek için kullanıldığını farz ederek bir tane daha denklem elde edilir ve denklem sistemi çözülerek muhtemel a ve b değerleri bulunur. Bunlardan uygun olan a , b çifti deneme yanılma yöntemiyle bulunur.

T=1 dan h=7 elde edildiğini farz edelim.

14 = a*4 + b MOD 26

7 = a * 1 + b MOD

sistemi elde edilerek, a=3 ve b=2 bulunur. Bu değerlere göre şifreli metin çözüldüğünde başarılı olduğu görülecektir.

IV.2.7. Vigenère Şifresinin Kriptanalizi

Vigenère benzeri poli alfabetik yerine koyma şifreleme algoritmaları 300 yıl boyunca pratik bir şekilde kırılamaz gibi görüldü. 1863 te bir Prusya binbaşısı Kasiski Vigenère algoritmasını kırmak için bir metot önerdi[39]. Bu metot anahtar kelimenin uzunluğunu bulma ve mesajı bir çok basit kaydırma şifreli metinlerine bölme adımlarından oluşmakta yani dolaylı bir frekans analizi saldırısına dayanmaktadır.

Kasiski metodunda anahtar kelime uzunluğunu belirleme işlemi şifreli metindeki tekrar eden ikili harflerin arasındaki mesafeyi ölçme esasına dayalıdır. Yukarıdaki açık metindeki "TO" ikili harfleri 0 ve 9 konumlarında iki kez tekrarlanmıştır. Her iki durumda da anahtar kelimenin ilk iki harfiyle tam olarak alta alta gelmektedir. Bundan dolayı şifreli metinde aynı ikili olan "KS" yi üretir. Aynı şekilde "BE" ikilisi de iki kez 2 ve 11 konumlarında tekrarlanarak şifreli metinde "ME" ikilisinin üretilmesini sağlamıştır. Bunun gibi Vigenère algoritmasıyla şifrelenen bir açık metin bir çok böyle tekrar eden ikili üretecektir. Açık metindeki her ikili şifreli metinde aynı ikiliyi üretmeyebilir. Ama çoğunlukla aynı ikiliyi üretecektir. Kasisiki şifreli metindeki bu ikililerin arasındaki mesafeleri ölçerek ve çarpanlarına ayırarak anahtar uzunluğunu tahmin etmiştir.

Örnek:

Konum : 01234567890123456789012345678 9

Anahtar kelime : RELATIONSRELATIONSRELATIONSREL

Açık metin : TOBEORNOTTOBETHATISTHEQUESTION

Şifreli metin : KSMEHZBBLKSMEMPOGAJXSEJCSFLZSY

Şifreli metindeki ikililer incelendiğinde aşağıdaki tablo meydana gelir. Mesafeyi çarpanlara ayırma muhtemel anahtar uzunluklarını elde etmemizi sağlar. Bu örnekte ihtimali en yüksek anahtar uzunlukları 3 ve 9 dur. Şifreli metin ve ikili sayıları artınca bir anahtar uzunluğu ihtimali en yüksek olarak ortaya çıkacaktır. Bu ortaya çıkan değer anahtar uzunluğu olarak alınabilir.

Anahtar uzunluğu bulunduktan sonra şifreli metin anahtar uzunluğu kadar ayrı şifreli metinlere ayrılır. Yani anahtarın birinci harfine denk gelenler ilk şifreli metin ikincisine denk gelenler ikinci şifreli metin ve böylece bir çok parçaya ayrılır. Her şifreli metin parçası aynı alfabeyle basit Sezar kaydırması gibi şifrelendiğinden her şifreli metin basit frekans analizi ile kırılarak anahtar kelime bulunur.

Tablo IV. 8 Şifreli Metinde Tekraralana İkililerin Konum, Mesafe ve Çarpanları

Tekrarlanan ikili	Konum	Mesafe	Çarpanlar
KS	9	9	3, 9
SM	10	9	3, 9
ME	11	9	3, 9
...			

IV.2.8. Rail fence Algoritmasının Kriptanalizi

Rail Fence şifreleme algoritmasının kriptanalizi çok basittir. Muhtemel sütun uzunlukları tek tek denenerek şifreli metin kırılabilir.

1.17.4.Çin Kalan Teoremi

Bu sistemleri çok eski Çin ve Hindu matematikçilerinin yazdığı kelime bulmacalarında bulabiliriz. Böyle bir bulmaca örneği aşağıda verilmiştir:

1. yy.da Çinli matematikçi, Sun-Tsu soruyor:3' e bölündüğü zaman 2 kalanını veren, 5' e bölündüğü zaman 3 kalanını veren, 7' ye bölündüğü zaman 2 kalanını veren sayı kaçtır? Bu bulmacanın çözümü;

$x \equiv 2 \pmod 3$,

$x \equiv 3 \pmod 5$,

$x \equiv 2 \pmod 7$,

sistemin çözümü ile eşdeğer olup Çin kalan teoreminden sonra verilecektir.

Çin kalan teoremi, adını Çinlilerden miras kalan, lineer denklik sistemlerini içeren bulmaca problemlerden almıştır. Lineer denklik sistemleri modül alma işlemine dayanan ve ikişerli aralarında asal olan sayılardan oluşur. Böyle bir sistemin çözümü tektir.

Teorem.1.11: Çin Kalan Teoremi: $m_1,m_2,.....m_n$ ikişerli aralarında asal pozitif tamsayılar olsun. Buna göre;

$\mathbf{x \equiv a_1 \pmod{m_1}}$,

$\mathbf{x \equiv a_2 \pmod{m_2}}$,

.

$\mathbf{x \equiv a_n \pmod{m_n}}$

sistemi **modm= $m_1.m_2.....m_n$'** e göre tek bir çözüme sahiptir. $0 \leq x < m$ aralığında bir x çözümü vardır. Diğer tüm çözümler, bu tek çözümdeki m modülüne denkdir.

Ispat.1.11: Teoremi kanıtlamak için, bu çözümün varolduğunu ve **m** modülüne göre tek olduğunu göstermemiz gerekir. Bunun için;

k=1,2,....n için $\mathbf{M_k = m / m_k}$ alalım. Burada, $\mathbf{m_k}$ hariç modüllerin çarpımı $\mathbf{M_k}$' dır.

$\mathbf{m_i}$ ile $\mathbf{m_k}$, $\mathbf{i \neq k}$ iken 1' den büyük hiç ortak çarpana sahip değillerse $\mathbf{gcd(m_k,M_k)=1}$' dir. Sonuç olarak, modül m_k ' ya göre, M_k' nın tersi y_k ;

$\mathbf{M_k.y_k \equiv 1 \pmod{m_k}}$ olacak şekilde mevcuttur. Buna göre, sistemin tek çözümü

$\mathbf{x \equiv a_1.M_1.y_1 + a_2.M_2.y_2 + ...+a_n.M_n.y_n}$

şeklinde tanımlanır. x çözümünün similtane çözümü olduğunu gösterelim.

Bunun için, $j \neq k$ iken, $M_j \equiv 0 \pmod{m_k}$ olduğundan, bu toplamdaki k. terim hariç bütün terimler modül m_k ' ya göre 0' dır. $M_k.y_k \equiv 1 \pmod{m_k}$ olduğundan k=1,2,....n için

$\mathbf{x \equiv a_k.M_k.yk \equiv a_k \pmod{m_k}}$

olduğu görülür. Böylece x istenen tek çözümdür.

Aşağıdaki örnekler, teorem.4.1'in ispatındaki oluşumu denklik sistemlerin çözümlerinde nasıl kullanılacağını açıklar. Sun-Tsu bulmacasında ortaya çıkan sistemleri örnek.4.6'daki gibi çözeceğiz.

Örnek.1.12: Öncelikle m=3.5.7 = 105 oluşturulur.Sonra,

$M_1 = m/3 = 35$, $M_2 = m/5 = 21$, $M_3 = m/7 = 15$ eşitliklerini yazdıktan sonra modül m_k ' ya göre M_k ' nın tersi y_k ' yı bulmak için $M_k.y_k \equiv 1$ denkliğinden yararlanırız. Buna göre;

$M_1.y_1 \equiv 1 \pmod 3$ ise $35.\ y_1 \equiv 1 \pmod 3$ ve buradan $y_1 = 2$

$M_2.y_2 \equiv 1 \pmod 5$ ise $21.\ y_2 \equiv 1 \pmod 5$ ve buradan $y_2 = 1$

$M_3.y_3 \equiv 1 \pmod 7$ ise $15.\ y_3 \equiv 1 \pmod 7$ ve buradan $y_3 = 1$ bulunur.

Örneğin; birinci denklikte 35 ile öyle bir sayıyı çarpalım ki sonucun mod 3' e göre kalanı 1 olsun. Diğer denklikleri de bu şekilde ifade edebiliriz. y değerleri bulunduktan sonra sistemin aşağıdaki gibi x çözümüne ulaşırız:

$$x \equiv a_1.M_1.y_1 + a_2.M_2.y_2 + a_3.M_3.y_3 = 2.35.2 + 3.21.1 + 2.15.1 \pmod{105}$$

$$= 233 \equiv 23 \pmod{105}.$$

Sonuç olarak, 23 en küçük pozitif tam sayısı bir similtane çözümdür. 3' e bölündüğünde 2 kalanını veren, 5' e bölündüğü 3 kalanını veren, 7' e 2 kalanını veren en küçük pozitif tam sayı 23 olarak bulunur.

1.17.5.Büyük Sayılarla Bilgisayar Aritmetiği

Aralarında ikişerli asal ve 2' den büyük veya eşit tam sayılar $m = m_1, m_2, m_3,....m_n$ olsun. Çin kalan teoremine göre gösterebiliriz ki; m_i , $i=1,2,....n$ $0 \leq a < m$ aralığında olan bir a tamsayısı (a mod m_1, a mod m_2, ...a mod m_n) olacak şekilde ifade edilir.

Örnek.1.13: 12' den küçük pozitif tam sayıların 3 ve 4 ile bölümünden kalanları nokta çiftleri ile bulunuz:

Çözüm.1.13: 3 ve 4' e bölündüğü zaman her sayıdan kalanları bularak çözüme ulaşabiliriz.

0 = (0,0)	4 = (1,0)	8 = (2,0)
1 = (1,1)	5 = (2,1)	9 = (0,1)
2 = (2,2)	6 = (0,2)	10 = (1,2)
3 = (0,3)	7 = (1,3)	11 = (2,3)

Örneğin; 6 sayısını ele alalım. Bu sayı 3' e bölündüğünde 0 kalanını, 4' e bölündüğünde 2 kalanını verecektir. Bu şekilde 12' e kadar olan sayıların sırasıyla 3 ve 4' e bölümünden kalanları bulunarak, yukarıdaki gibi bir çözüm elde edilir.

Büyük tamsayılarla aritmetik uygulaması için, her m_i sayısı 2' den büyük olmak koşuluyla, $i \neq j$ iken $gcd(i,j) = 1$ olduğunda m_1, m_2,...m_n modüllerini seçeriz ve $m = m_1.m_2.....m_n$, gerçekleştirmek istediğimiz aritmetik işlemlerin sonuçlarından daha büyüktür.

İlk olarak, modülü seçeriz. m_i' ye, i=1,2,...n' e kadar sayıların bölümünden kalan n adet sayı bulunur. Sayıların işlem bileşenlerinde yerine getirilmesiyle büyük tamsayılarla aritmetik işlemleri gerçekleştirebiliriz. Sonuçta her bileşenin değerini hesaplarız. m_i, i = 1,2,....n modülünde denk bir n sistemini çözerek değerleri tekrar elde ederiz. Büyük tam sayılarla uygulanan aritmetik metodun önemli birkaç özelliği vardır. Birincisi, genellikle bilgisayarda uygulanan tamsayılardan daha geniş tamsayılarla aritmetik uygulaması kullanılabilir. İkincisi, farklı modüllerle ilgili olarak hesaplamalar yapılabilir, buna paralel olarak işlem hızı artar.

Örnek.1.14: Büyük tamsayılarla yapılan aritmetikten daha hızlı, kesin bir yöntem üzerinde 100' den küçük tamsayılarla bir işlem uygulaması varsayalım. 100' den küçük ardışık asal tamsayıların modülünden kalanların kullanıldığı sayıları gösterirsek, 100' den küçük tamsayılar için hemen hemen tüm hesaplamalarımızı sınırlayabiliriz. Örneğin; 99, 98, 97 ve 95' in modülünü kullanabiliriz. (Bu sayılar, 1' den başka ortak çarprnı bulunmayan ikişerli aralarında asal sayılardır.)

Çin kalan teoremi ile, 99.98.97.95 = 89,403,930' dan küçük tüm pozitif tamsayılar bu 4 modüle sırasıyla bölündüğünde, kalanları olarak tek gösterilebilir. Örneğin;

123684 mod 99 = 33

123684 mod 98 = 8

123684 mod 97 = 9

123684 mod 95 = 89

olduğu için 123684 sayısını (33,8,9,89) şeklinde de ifade edebiliriz. Buna denk, 413456 sayısını (32,92,42,16) gibi de ifade edebiliriz. 123684 ve 413456 sayılarının toplamını bulmak için 4 sayı yerine iki tamsayıyla çalışırız. 4 bileşen ekler ve her bileşenin, uygun modüllerini bularak indirgeriz.

(33,8,9,89) + (32,92,42,16)= (65 mod 99, 100 mod 98, 51 mod 97, 105 mod 95)

= (65,2,51,10).

Bu toplamı bulmak için;

$x \equiv 65 \pmod{99}$

$x \equiv 2 \pmod{98}$

$x \equiv 51 \pmod{97}$

$x \equiv 10 \pmod{95}$

sistemi Çin kalan teoremi yardımıyla bulunur.537140 , 89403930 sayısından küçük, bu sistemin tek pozitif çözümüdür. Sonuç olarak, 537140 toplamdır. Sayıları (65,2,51,10) olarak çevirdiğimiz zaman 100' den büyük sayılarla işlem yapmak zorundayız.

Büyük sayılarla işlem yapmak için, modülün iyi seçimleri, k pozitif tamsayı olmak şartıyla, 2^k -1 şeklindeki tam sayılar kümesidir. Çünkü, bu tür sayılarla ikili işlem modülü yapmak ve aralarında ikişerli asal sayılar kümesini bulmak kolaydır.

İkinci sebep ise $\gcd(2^a-1,2^b-1) = 2^{\gcd(a,b)} -1$ olan gerçek bir denkliktir. Örneğin; bilgisayarımızda 2^{35} ' ten küçük sayılarla işlem yapabileceğimizi varsayalım. Fakat, büyük sayılarla çalışmak özel prosedürler gerektirir. Büyük sayılarla işlem uygulaması için 2^{35}' ten küçük ikişerli aralarında asal sayı modülünü kullanabiliriz.

Mesela; $2^{35}-1$, $2^{34}-1$, $2^{33}-1$, $2^{31}-1$,$2^{29}-1$ ve $2^{23}-1$ sayıları ikişerli aralarında asal sayılardır. Bu altı modülün sonuçları 2^{184}' ü aşacağı için, 2^{184} kadar büyük sayılarla işlemi, bu altı modülden hiçbiri 2^{35}' i geçmeyecek şekilde aritmetik modülü yaparak düzenleyebiliriz.

Bir sayının asal olduğuna karar vermek için daha etkili yollar var mıdır? Eski Çin matematikçisi sadece $2^{n-1} \equiv 1 \pmod{n}$ ise n' nin asal olduğuna inanırdı.

Eğer bu doğruysa, etkili bir test uygulanacaktır. Neden bir sayının asal olduğuna karar vermek için kullanılabilen bu denkliğe inandılar? Birincisi, onlar n asal olduğu zaman denkliğin tuttuğunu gözlediler. Örneğin, 5 asaldır ve $2^{5-1} = 2^4 = 16 \equiv 1 \pmod{5}$.

İkincisi, denkliğin tuttuğu çok karmaşık olan bir n sayısı bulamadılar. Eski Çin matematikçileri kısmen haklıydı. n asal olduğu zaman denkliğin tuttuğunu doğru düşündüler. Fakat, denklik tutarsa n' in gerekli asal olduğu sonucuna varmaları yanlıştı.

Büyük Fransız matematikçisi Fermat, n asal olduğu zaman denkliğin tuttuğunu gösterdi. Bunu takiben daha genel sonuçlara da ulaştı.

Teorem.1.12.Fermat Teoremi: p asalsa ve a, p ile bölünmeyen bir tamsayı ise,

$a^{p-1} \equiv 1 \pmod{p}$

Bunun dışında, sahip olduğumuz her a sayısı için,

$a^p \equiv a \pmod{p}$

Maalesef, $2^{n-1} \equiv 1 \pmod{n}$ gibi karmaşık bir n tamsayısı vardır. Böyle tamsayılara yarı asallar denir.

(Pierre de Fermat (1601-1665). 17. yy.da önemli matematikçilerden biri olan Fermat profesyonel bir hukukçuydu. Tarihte çok ünlü amatör bir matematikçidir. Fermat kendi matematik buluşlarını yayınladı. Buna rağmen, diğer matematikçilerle çalışmaları da olduğunu biliyoruz. Fermat analitik geometriyi ve hesaplamanın birkaç temel mantığını geliştiren bir bilim adamıdır. Fermat, Pascalla temel matematik teorilerini vermiştir. Fermat matematikte çözülmeyen ünlü problemleri formüle etmiştir. n 2' den büyük bir tamsayı olduğunda, önemsiz pozitif tamsayı çözümlerine sahip olan $x^n+y^n = z^n$ eşitliğiyle yarar sağladı.)

Örnek.1.15: $2^{340} \equiv 1 \pmod{341}$ gibi gösterildiği ve (341=11.31) olarak tanımlandığı zaman 341 bir yarı asal sayıdır.

1.18.Bölüm ve Denklik Bağıntılarının Karmaşıklığı ile İlgili Teoremler ve Örnekleri

Varsayalım ki bir algoritma n- boyutlu bit problemi a tane alt probleme bölsün ve bu alt problemlerin herbirinin boyutu $^n/_b$ olsun. (n , b'yi böler varsayalım. Gerçekte , küçük problemler eşit boyutlu alt problemlere sahiptirler ve bu $^n/_b$ sayısı en yakın tamsayıya dönüştürülür.

n boyutlu problem küçük problemlere ayrıldığında gerekli olan ekstra işlemlerin sayısına **g(*n*)** diyelim. Böylece , ***f*(n)** bir problemi çözmek için gereken işlemlerim sayısını ifade ettiğinde;

$$f(n) = a.f(^n/_b) + g(n)$$

bağıntısı sağlanır ve bu bağıntı **bölüm ve denklik bağıntısı** olarak adlandırılır.

Örnek.1.16 : ikili arama algoritmasında;

$$f(n) = f\left(\frac{n}{2}\right) + 2 \qquad \text{(n çift)}$$

Örnek.1.17: n-elemanlı bir dizide bu dizinin maksimum ve minimum elemanını bulan algoritmada;

$$f(n) = 2.f\left(\frac{n}{2}\right) + 2 \qquad \text{(n çift)}$$

5. İNDIRGEME ALGORİTMALARI

5.1.İndirgeme Bağıntılarının Çözümü

k.dereceden sabit katsayılı Lineer homojen bir indirgeme bağıntısı;

$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \ldots\ldots\ldots\ldots + c_k a_{n-k}$

$c_1, c_2, \ldots\ldots\ldots, c_k$ reel sayılar ve $c_k \neq 0$ ' dır.

Başlangıç koşulları ;

$H_n = 2 H_{n-1} + 1$ homojen değil.

$B_n = n. B_{n-1}$ sabit katsayılı değil.

$a_{n-1} + a_{n-2}$ Lineer değil.

Bu bağıntılar sistematik olarak çözüldüğü ve problemlerin modellemesinde sıklıkla ortaya çıktığından dolayı seçilmiştir.

Çözümü : $a_n = r^n$, $a_n = c_1 a_{n-1} + c_2 a_{n-2} + \ldots\ldots\ldots\ldots + c_k a_{n-k}$

İndirgeme bağıntılarının çözümü ise ;

$$\mathbf{r^n = c_1.r^{n-1} + c_2.r^{n-2} + \ldots\ldots + c_k.r^{n-k}}$$

yazılır. Her iki taraf r^{n-1} 'ya bölünürse

$$\mathbf{r^k - c_1.r^{k-1} - c_2.r^{k-2} - \ldots\ldots - c_{k-1}.r - c_k = 0}$$ elde edilir.

Bu denkleme karakteristik denklem köklerinede karakteristik denklem kökleri denir.

Teorem.1.13 : c_1 ve c_2 reel sayılar olsun.

$r^2 - c_1 r - c_2 = 0$ 'ın r_1 ve r_2 iki farklı kökü olsun.

$a_n = c_1 a_{n-1} + c_2 a_{n-2}$

$a_n = \alpha_1 r_1^n + \alpha_2 r_2^n$ $\quad n = 0,1,2,\ldots..(\alpha_1, \alpha_2$ sabitler)

için $\{a_n\}$ dizisi çözümdür.

Teorem.1.14 : c_1 ve $c_2 \in |R$, $c_2 \neq 0$ olsun.

$r^2 - c_1 r - c_2 = 0$ $r_1 = r_2 = r_0$ olsun.

Eğer, $a_n = \alpha_1 r_0^n + \alpha_2 . n r_0^n$ $\quad n = 0,1,2,\ldots..(\alpha_1, \alpha_2$ sabitler)

İse $a_n = c_1 a_{n-1} + c_2 a_{n-2}$

İndirgeme bağıntılarının çözümü $\{a_n\}$ dizisidir.

Teorem.1.15: $\mathbf{c_1, c_2, \ldots\ldots c_k}$ reel sayılar olsun. ve

$\mathbf{r^k - c_1 r^{k-1} - \ldots\ldots - c_k = 0}$ karakteristik denklemi $r_1, r_2, \ldots\ldots, r_k$ farklı köklere sahip olsun.

Eğer, $a_n = \alpha_1 r_1^n + \alpha_2 . r_2^n + \ldots\ldots + \alpha_k . r_k^n$ $\quad n = 0,1,2,\ldots..(\alpha_1, \alpha_2, \alpha_k$ sabitler)

İse $a_n = c_1 a_{n-1} + c_2 a_{n-2} + \ldots\ldots + c_k a_{n-k}$

Bağıntısının çözümü $\{a_n\}$ dizisidir.

Genelleme : Eğer , n, b'yi böler ve $n = b^k$, $k \in Z^+$ ise

$$f(n) = a \cdot f(n/b) + g(n)$$

$$= a^k \cdot f(n/b^k) + \sum_{j=0}^{k-1} a^j \cdot g(n/b^j)$$

$$\frac{n}{b^k} = 1 \Rightarrow f(n) = a^k \cdot f(1) + \sum_{j=0}^{k-1} a^j \cdot g(n/b^j)$$

Teorem.1.16 : f artan bir fonksiyon , n , b' yi bölsün $a \geq 1$ ve b, 1'den büyük bir tamsayı ve c pozitif bir reel sayı olmak üzere ;

$f(n) = a \cdot f\left(\frac{n}{b}\right) + c$ sağlansın . Bu durumda ;

$$f(n) = \begin{cases} O\left(n^{\log_b a}\right) & , a > 1 \\ O(\log n) & , a = 1 \end{cases}$$

Örnek.1.18 : $f(n) = 5.f(n/2) + 3$ ve $f(1) = 7$ olsun. Buna göre, $f(2^k)$, $k \in Z^+$'yı bulunuz ve $f(n)$ için f artan bir fonksiyon iken bir tahmin yapınız?

Çözüm.1.18 : $\left.\begin{matrix} a = 5 \\ b = 2 \\ c = 3 \end{matrix}\right\}$ $n = 2^k$

$$f(n) = a^k \left[f(1) + \frac{c}{a-1} \right] - \frac{c}{a-1} = 5^k \left[7 + \frac{3}{4} \right] - \frac{3}{4}$$

$f(n) = 5^k \left(\frac{31}{4}\right) - \frac{3}{4}$ ve örtendir. $a > 1 \Rightarrow f(n) = O(n^{\log_b a}) = O(n^{\log 5})$

Örnek.1.1 : İkili arama algoritması: $f(n) = f(\frac{n}{2}) + 2$

a = 1 , b = 2 , c = 3 $\rightarrow$ **f (n) = O (Log n)**

Örnek.1.20 : $f(n) = 2.f(\frac{n}{2}) + 2$ a = 2, b = 2, c = 2 $\rightarrow$ a > 1

$f(n) = O(n^{\log_b a}) = O(n^{\log 2^2}) = O(n)$

5.2. İndirgeme Bağıntıları ile Modelleme

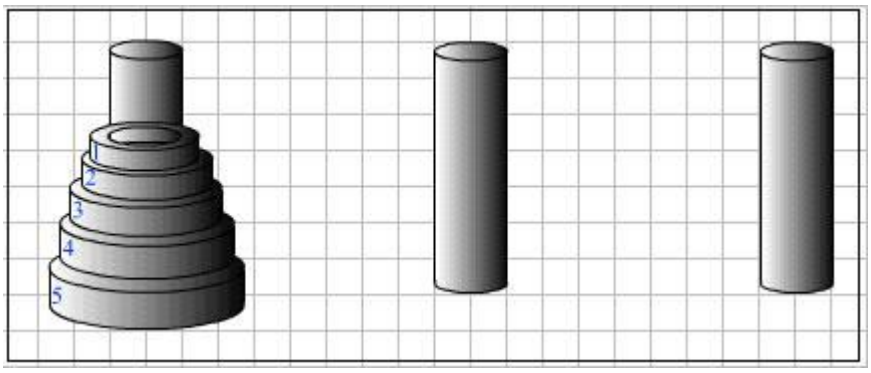
5.2.1.Hanoi Kulesi

Hanoi Kulesi 1–yy'ın meşhur bir puzzle'ı olup üç tane çubuktan oluşan bir düzlemde bir çubuktan diğer bir çubuğa tek bir hareketle farklı ölçülerdeki diskleri en küçük disk en üstte en büyük disk en altta olmak şartıyla sırayla dizmektir.

n tane diskten oluşan Hanoi Kulesi problemini ele alalım. n tane diskin hareket sayısını bir defada bir diski hareket ettirmek şartıyla H_n olarak tanımlayalım.Buna göre,

$\{ H_n \}$dizisi için bir indirgeme bağıntısı bulalım.

Aşağıdaki üç tane çubuğu göz önüne alalım.(Şekil.1.4)



Çubuk.1 Çubuk.2 Çubuk.3

Şekil1.4'de de görüldüğü gibi 1. çubukta n tane diskimiz var. Bu **n** tane diskten üsteki **(n – 1)** tanesini birer hareketle 2. çubuğa transfer ediyoruz. Bunun için $\mathbf{H_{n-1}}$ hareket yapıyoruz. Yani, $\mathbf{H_1 = 1}$ kabul ediyoruz. Böylece en büyük disk 1. çubukta kalıyor. Bundan sonra 1. çubuktaki en büyük diski $H_1 = 1$ olduğu hatırlanarak1 hareketle 3. çubuğa taşıyoruz ve 2. çubuktaki (n-1) diski yine H_{n-1} hareketle 3. çubuktaki en büyük diskin üstüne taşıyoruz.

Sonuç olarak n tane diski bir çubuktan diğer bir çubuğa en küçük disk en üstte en büyük disk en altta olmak ve bir hareketle bir disk taşınmak şartıyla;

$\mathbf{H_n = 2H_{n-1} + 1}$ şeklinde ifade edebiliriz.

Şimdi bu son ifadeden hanoi kulesi problemi için bir indirgeme bağıntısı bulmak için yine açıkladığımız mantığa göre H_{n-1}'den H_1'e kadar olan degerleri aşağıdaıki gibi yerlerine yazarsak ;

$$\mathbf{H_n = 2H_{n-1} + 1}$$
$$\mathbf{= 2(2H_{n-2} + 1) + 1 = 2^2.H_{n-2} + 2 + 1}$$
$$\mathbf{= 2^2(2H_{n-3} + 1) + 1 = 2^3.H_{n-3} + 2^2 + 2^2 + 2 + 1}$$

.

.

$= 2^{n-1}. H_1 + 2^{n-2} + 2^{n-3} + ... + 2 + 1$

$= 2^{n-1} + 2^{n-2} + ... + 2 + 1$

$= 2^{n-1} - 1$

5.2.2. FİBONACCİ SAYILARI

$a(1) = 1,\ a(2) = 1$ başlangıç değerleri ve

indirgeme bağıntısı ile tanımlı diziye Fibonacci dizisi denir.

İndirgeme bağıntısı,

$$a(n) - a(n-1) - a(n-2) = 0$$

ve karakteristik denklem

$$\lambda^2 - \lambda - 1 = 0$$

olmak üzere karakteristik denklemin kökleri,

$$\lambda_1 = \frac{1+\sqrt{5}}{2}$$

$$\lambda_2 = \frac{1-\sqrt{5}}{2}$$

dır. Dizinin genel terimi,

$$a(n) = A_1(\frac{1+\sqrt{5}}{2})^n + A_2(\frac{1-\sqrt{5}}{2})^n$$

biçimindedir. Başlangıç değerlerinden,

$$\begin{cases} A_1 \frac{1+\sqrt{5}}{2} + A_2 \frac{1-\sqrt{5}}{2} = 1 \\ A_1(\frac{1+\sqrt{5}}{2})^2 + A_2(\frac{1-\sqrt{5}}{2})^2 = 1 \end{cases}$$

denklem sistemi yazılır. Denklem sisteminin çözülmesiyle,

$$A_1 = \frac{1}{\sqrt{5}}$$

$$A_2 = -\frac{1}{\sqrt{5}}$$

elde edilir. Buna göre Fibonacci dizisinin genel terimi,

$$a(n) = \frac{1}{\sqrt{5}}\left[(\frac{1+\sqrt{5}}{2})^n - (\frac{1-\sqrt{5}}{2})^n \right]$$

dır.

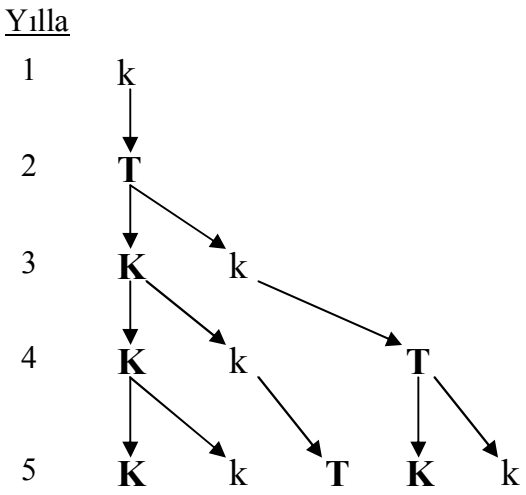
Fibonacci dizisinin ilk 20 terimi aşağıdadır.

n	$a(n)$	$a(n+1)/a(n)$	n	$a(n)$	$a(n+1)/a(n)$
1	1		11	89	1.618182
2	1	1	12	144	1.617977
3	2	2	13	233	1.618056
4	3	1.5	14	377	1.618026
5	5	1.666667	15	610	1.618037
6	8	1.6	16	987	1.618033
7	13	1.625	17	1597	1.618034
8	21	1.615385	18	2584	1.618034
9	34	1.61048	1	4181	1.618034
10	55	1.617647	20	6765	1.618034

Leonardo Fibonacci 12-13 üncü yüzyıllarda yaşamış bir İtalyan matematikçisidir Bu problem "tavşan problemi" dir. Ergin bir tavşan çiftinin her ay yeni bir yavru çifti verdikleri ve yeni doğan çiftin bir ay zarfında tam erginliğe eriştikleri varsayımıyla, yavru olan bir tavşan çiftinden başlayıp bir yılda (12 ayda) çiftlerin sayısı ne olur?

Aylar :	1	2	3	4	5	6	7	8	9	10	11	12
Çiftlerin sayısı:	1	1	2	3	5	8	13	21	34	55	89	144

Kitaplara tavşan problemi olarak geçen bu problem kuzu-toklu-koyun problemi olarak bilinmektedir. Bilindiği gibi ergin bir kuzu olan toklu genellikle iki yaşına geldiğinde yavrulamaktadır. Buna göre bir yavru dişi kuzu ile başlayıp her yavrulama sonucu bir dişi kuzunun doğduğu ve ölüm olmaması durumunda 12 yıl sonunda kaç baş hayvana sahip olunur? Bu gelişimin ilk beş yılını şekil üzerinde gözleyelim. Dişi kuzuyu k, tokluyu T, koyunu K harfi temsil etmek üzere gelişim aşağıdaki gibidir.



Belli bir yıldaki hayvan sayısı önceki iki yıldakilerin toplamı olmak üzere 12. yılda 144 baş hayvana ulaşılacaktır. Bunların yıllara göre yavru, toklu, koyun sayısı dağılışına gelince aşağıdaki gibi bir durum ortaya çıkmaktadır.

Yıllar	Kuzu sayısı	Toklu sayısı	Koyun sayısı	Toplam
1	1	-	-	1
2	0	1	-	1
3	1	0	1	2
4	1	1	1	3
5	2	1	2	5
6	3	2	3	8
7	5	3	5	13
8	8	5	8	21
9	13	8	13	34
10	21	13	21	55
11	34	21	34	89
12	55	34	55	144

Dikkat edilirse $b(n)$, n. yıldaki kuzuların sayısı olmak üzere, kuzular için

$$b(n) = b(n-1) + b(n-2) \quad , \quad b(1) = 1 \ , \ b(2) = 0$$

indirgeme bağıntısı sözkonusudur ve çözüm olan

1 , 0 , 1 , 1 , 2 , 3 , 5 , 8 , 13 , . . .

dizisinde $n \geq 3$ için Fibonacci dizisi ortaya çıkmaktadır, yani

$$b(n) = a(n-2) \quad , \quad n \geq 3$$

dır. Benzer bir durum yıllara göre toklu sayısı için de sözkonusudur. $c(n)$, n. yıldaki tokluların sayısı olmak üzere,

$$c(n) = a(n-3) \quad , \quad n \geq 3$$

dır. Yıllara göre koyunların sayısı ise iki yıl geçikmeli bir Fibacci dizisidir.

Fibonacci'nin kendisi,

1 , 1 , 2 , 3 , 5 , 8 , 13 , 21 , ...

dizisi üzerinde bir inceleme yapmamıştır. Hatta bu dizi üzerinde ondokuzuncu yüzyılın başlarına kadar ciddi bir araştırma yapılmadığı da belirtilmektedir. Ancak bundan sonra bu dizi üzerine yapılan araştırmaların sayısı Fibonacci'nin tavşanlarının sayısı gibi artmıştır.

Fibonacci dizisinde olduğu gibi,

$$a(n) = a(n-1) + a(n-2)$$

indirgeme bağıntılı ve herhangi $a(1), a(2)$ başlangıç değerli genelleştirilmiş Fibonacci dizileri,

$$a(n) = a(n-1) + a(n-2) + a(n-3)$$

indirgeme bağıntılı $a(1), a(2), a(3)$ başlanğıç değerli tribonacci dizileri gibi genellemeler yapılmıştır Fibonacci dizileri ile ilgili bilinen birkaç özellik üzerinde duralım.

Genelleştirilmiş Fibonacci dizilerinde de geçerli olmak üzere, $a(1) = 1$, $a(2) = 1$ ve

$$a(n) = a(n-1) + a(n-2)$$

olan Fibonacci dizisinin bir terimi öncekine bölündüğünde bölümün $n \to \infty$ için, "altın oran" denen ve irrasyonel bir sayı olan

$$\frac{1+\sqrt{5}}{2} = 1.61803398...$$

sayısına yakınsadığı görülmektedir

Örnek: Fibonacci dizisinin ilk 20 teriminin toplamı acaba nedir?

İlk n terimin toplamı

$$a(1) + a(2) + ... + a(n) = \sum_{k=1}^{n} \frac{1}{\sqrt{5}} \left[\left(\frac{1+\sqrt{5}}{2}\right)^k - \left(\frac{1-\sqrt{5}}{2}\right)^k \right]$$

$$= \frac{1}{\sqrt{5}} \left[\sum_{k=1}^{n} \left(\frac{1+\sqrt{5}}{2}\right)^k - \sum_{k=1}^{n} \left(\frac{1-\sqrt{5}}{2}\right)^k \right]$$

$$= \frac{1}{\sqrt{5}}\left[(\frac{1+\sqrt{5}}{2}) \frac{1-(\frac{1+\sqrt{5}}{2})^n}{1-(\frac{1+\sqrt{5}}{2})} - (\frac{1-\sqrt{5}}{2}) \frac{1-(\frac{1-\sqrt{5}}{2})^n}{1-(\frac{1-\sqrt{5}}{2})} \right]$$

$$= \frac{1}{\sqrt{5}}\left[(\frac{1+\sqrt{5}}{2})^{n+2} + (\frac{1-\sqrt{5}}{2})^{n+2} - \sqrt{5} \right]$$

$$= a(n+2) - 1$$

olmak üzere,

$$a(n+2) = a(1) + a(2) + \ldots + a(n) + 1$$

yazılabilir. Buna göre ilk 20 terimin toplamı,

$$a(1) + a(2) + \ldots + a(20) = a(22) - 1 = 17711 - 1 = 17710$$

dır.

$b(1) = \alpha, b(2) = \beta$ başlangıç değerli genelleştirilmiş Fibonacci dizisinin terimleri için,

$$b(3) = b(1) + b(2) = \alpha + \beta$$
$$b(4) = b(2) + b(3) = \alpha + 2\beta$$
$$b(5) = b(3) + b(4) = 2\alpha + 3\beta$$
$$b(6) = b(4) + b(5) = 3\alpha + 5\beta$$
$$b(7) = b(5) + b(6) = 5\alpha + 8\beta$$
$$b(8) = b(6) + b(7) = 8\alpha + 13\beta$$
$$. \; . \; .$$

yazılabilir. $(a(n))$ Fibonacci dizisi olmak üzere,

$$b(n) = \alpha.a(n-2) + \beta.a(n-1)$$

dır. Buna göre,

$$b(1) + b(2) + \ldots + b(n) = \alpha\left[1 + a(1) + a(2) + \ldots + a(n-2)\right] + \beta\left[1 + a(2) + a(3) + \ldots + a(n-1)\right]$$
$$= \alpha.a(n) + \beta\left[a(n+1) - 1\right]$$

elde edilir. $b(1) = 1, b(2) = 3$ başlangıç değerli

1 , 3 , 4 , 7 , 11 , 18 , 29 , 47 , 76 , . . .

genelleştirilmiş Fibonacci dizisinin ilk 15 teriminin toplamı,

$$b(1) + b(2) + \ldots + b(15) = 1.a(15) + 3.\left[a(16) - 1\right]$$

olmak üzere bu toplam 3568 dir.

Genelleştirilmiş bir Fibonacci dizisinde ilk 10 terimin toplamı 7. terimin 11 katıdır. Örneğin $b(1) = 1, b(2) = 3$ başlangıç değerli genelleştirilmiş Fibonacci dizisinde,

$$b(1) + b(2) + \ldots + b(10) = 1.a(10) + 3.(a(11) - 1)$$

$$= 55 + 3.(89 - 1) = 319$$

ve $b(7) = 29$ olmak üzere

$$b(1) + b(2) + ... + b(10) = 11.b(7)$$

dir. Genel olarak $b(1) = \alpha, b(2) = \beta$ başlangıç değerli bir genelleştirilmiş Fibonacci dizisinde,

$$b(7) = 5\alpha + 8\beta$$

olmak üzere

$$\begin{aligned} b(1) + b(2) + ... + b(10) &= \alpha.a(10) + \beta.(a(11) - 1) \\ &= \alpha.55 + \beta.(89 - 1) \\ &= 55\alpha + 88\beta \\ &= 11b(7) \end{aligned}$$

dir.

Ardışık iki Fibonacci sayısısının en büyük ortak böleni 1'dir.

Fibonacci dizisindeki her üçüncü terim 2'ye, her dördüncü terim 3'e, her beşinci terim 5'e, her altıncı terim 8'e, her yedinci terim 13'e, yani her *kj. (j=1,2,...)* terim $a(k)$'ya tam olarak bölünmektedir.

$a(1) = 1^2$ ve $a(12) = 12^2$ olmak üzere bunların dışında, kendi indisinin karesine eşit olan başka bir Fibonacci sayısının bulunmadığı ispatlanmıştır.

Fibonacci dizisinde asal sayı olan terimlerin sayısının ne olduğu sorusu çözülememiş problemlerden birisidir.

Fibonacci sayılarının üretimi bir başka hikaye ile şu şekilde de ifade edilir.

Bir adada bir genç tavşan çifti var. Bunlar ilk 2 aya kadar üremiyorlar. 2 ay sonra her bir çift başka bir çift olarak ürüyorlar. Hiçbir tavşanın ölmediği var sayılarak n ay sonra adadaki tavşan çiftlerinin sayısı için bir rekurans (indirgeme) bağıntısı bulalım.

AYLAR	ÜRETİLEN ÇİFT	GENÇ ÇİFT	TOPLAM ÇİFT
1	0	1	1
2	0	1	1
3	1	1	2
4	1	2	3
5	2	3	5
6	3	5	8

$f_1 = 1$, $f_2 = 1$, $f_3 = 2$, $f_4 = 3$ ise $f_3 = f_1 + f_2$ ve $f_4 = f_3 + f_2$ olarak bulunur. Buna göre;

$n \geq 3$ için **$f(n) = f_{n-2} + f_{n-1}$** olduğu kolayca görülebilir.

Fibonacci sayıları için bir indirgeme formülü bulalım.

$f_n = f_{n-1} + f_{n-2} \Rightarrow r^2 = r + 1 \Rightarrow r^2 - r - 1 = 0$ $f_0 = 1$ $f_1 = 1$

$r_1 = (1+5^{1/2})/2$ $r_2 = (1-5^{1/2})/2$ $f_n = \alpha_1 .(1+5^{1/2}/2)^n + \alpha_2 (1-5^{1/2}/2)^n$ olursa

$f_0 = 1 \Rightarrow \alpha_1 + \alpha_2 = 1$

$f_1 = 1 \Rightarrow \alpha_1 (1+5^{1/2}/2) + \alpha_2 (1-5^{1/2}/2) = 1$ olur. Buradan,

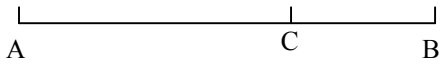
$\alpha_1 = 1/5^{1/2}$ ve $\alpha_2 = -1/5^{1/2}$ bulunur. Böylece,Fibonacci sayıları için bir indirgeme formülü ;

$$f_n = (1/5)^{1/2} .((1+5^{1/2})/2)^n - (1/5)^{1/2} ((1-5^{1/2})/2)^n$$

olarak bulunur.

ALTIN ORAN

Bilindiği gibi Altın Kesit bir AB doğru parçasının,



$$\frac{AB}{AC} = \frac{AC}{CB}$$

orantısına uygun olarak C noktası tarafından bölünmesidir.(AB gibi bir doğru parçasının uzunluğunu karışıklığa yol açmadığı takdirde yine AB ile göstereceğiz.)

$$AB = AC + CB$$

olduğu göz önüne alınırsa,

$$\frac{AC + CB}{AC} = \frac{AC}{CB}$$

$$1 + \frac{1}{\frac{AC}{CB}} = \frac{AC}{CB}$$

ve

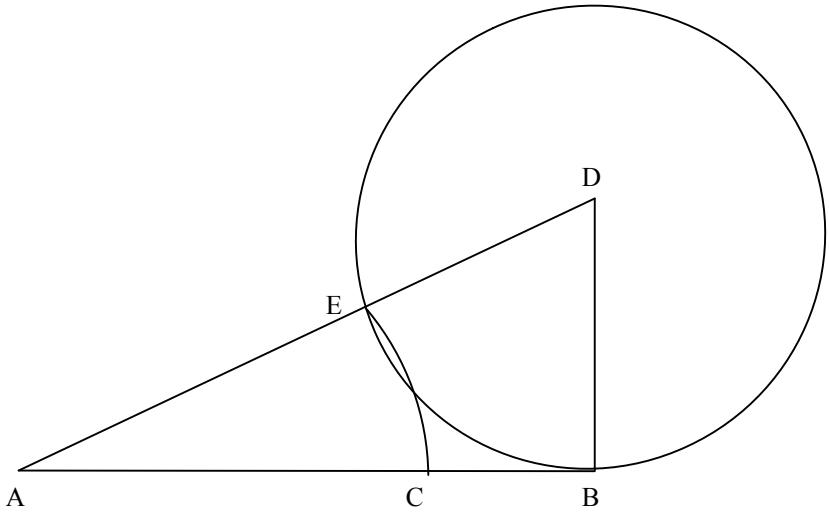
$$\frac{AC}{CB} = \frac{1+\sqrt{5}}{2}$$

elde edilir.

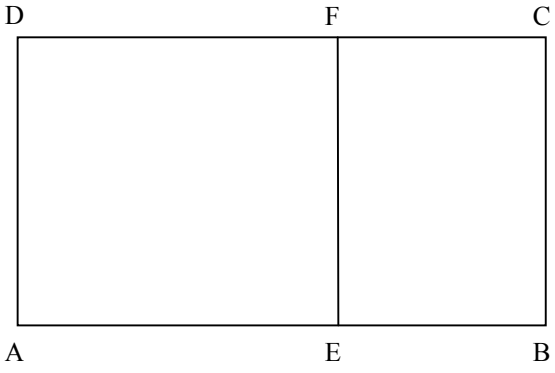
$$\frac{\mathbf{AB}}{\mathbf{AC}} = \frac{\mathbf{AC}}{\mathbf{CB}} = \frac{\mathbf{1}+\sqrt{\mathbf{5}}}{\mathbf{2}} = 1.618$$

oranına (sayısına) Altın Oran denir.

Bir AB doğru parçasının Altın Kesitini bulmak için AB ye dik olan ve uzunluğu BD=AB/2 olan BD doğru parçası çizilir. Kenar uzunlukları $1:2:\sqrt{5}$ oranında olan DBA dik üçgeninde D merkezli DB yarıçaplı çemberin AD hipotenüsü ile arakesiti (E noktası) bulunur. A merkezli AE yarıçaplı çemberin AB doğru parçası ile arakesiti (C noktası) Altın Kesiti vermektedir.

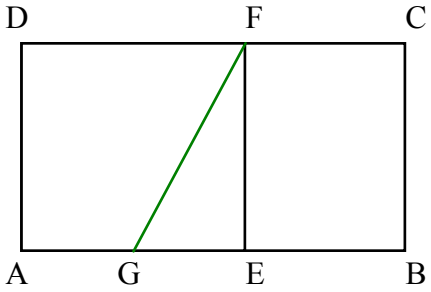


Euclide, Altın Kesit ile ilgili olarak başka bir problem daha vermiştir; Öyle bir dikdörtgen bulunsun ki, bundan bir kare çıkarıldığında geriye kalan küçük dikdörtgenin kenar uzunlukları oranı kendisininkiyle aynı olsun. Böyle bir dikdörtgene Altın Dikdörtgen denir.



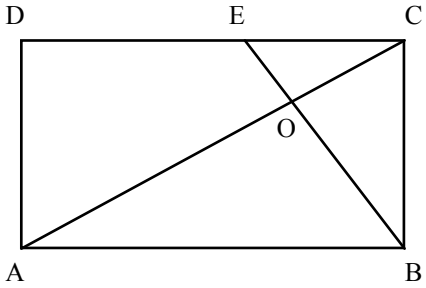
$$\frac{AB}{AD} = \frac{BC}{BE}$$

Uzun kenarından başlayıp bir Altın Dikdörtgen çizmek için AB kenarının Altın Kesiti olan E noktası bulunur ve sonra AD=AE olacak şekilde ABCD çizilir.

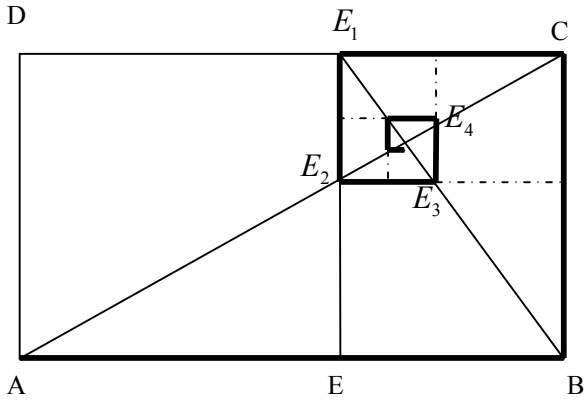


Kısa kenardan (AD) başlayıp bir Altın Didörtgen çizmek için önce AEFD karesi çizilip sonra AE nin orta noktası (G) den GF yarıçaplı çemberin AE doğrusu ile arakesiti olan B noktası bulunur.

Bir ABCD Altın Dikdörtgeninde AC köşegenine dik olan BO doğrusu DC kenarının (E) Altın Kesitinden geçer.



Yukarıdaki düşünceden istifade ederek bir ABCD Altın Dikdörtgeninde sırasıyla $E_1, E_2, E_3, \ldots$ noktalarının çizilmesiyle gitgide küçülen (alanları Φ^2 oranında küçülen) Altın Dikdörtgenler elde edilir.



AB kenarını kısa kenar kabul edip gittikçe büyüyen Altın Dikdörtgenler de çizilebilir. AB, BC, CE_1, E_1E_2, E_2E_3, ... doğru parçalarının oluşturduğu şekle dik çizgili sarmal (spiral) denir.

ABCD Altın Dikdörtgeninde,

$$\frac{AB}{BC}=\frac{BC}{CE_1}=\frac{CE_1}{E_1E_2}=\frac{E_1E_2}{E_2E_3}=\ldots=\Phi$$

dır.

Bilindiği gibi, eşit açılı logaritmik spiralin kutupsal koordinatlarda denklemi, k ve c pozitif reel sabitler olmak üzere,

$$\rho = k\mathrm{e}^{c\theta} \quad , \quad \theta \in [0,\infty($$

biçimindedir. ρ ışın vektörü ile ışın vektörünün spirali kestiği noktadaki teğet arasındaki açının tanjantı,

$$\mathrm{tg}\varphi=\frac{\rho}{\frac{d\rho}{d\theta}}=\frac{1}{c}$$

dır.

$A,B,C,E_1,E_2,E_3,\ldots$ noktalarından geçen eşit açılı logaritmik spiral için,

$$c=\frac{2}{\pi}\ln\Phi$$

ve

$$\varphi \approx 73^{\circ}$$

dır.

Altın dikdörtgenler estetik açıdan göze en hoş görünen dikdörtgenlerdir. Kendilerine çeşitli dikdörtgen örnekleri gösterilip, en güzelini ve en çirkinini

seçmeleri istenilerek yapılan araştırmalarda, örneğin Fechner (1876) ve Lalo (108) 'nun anket sonuçları aşağıdaki gibi olmuştur. (Bilim ve Teknik, Cilt 25, Sayı 297,Sayfa 7)

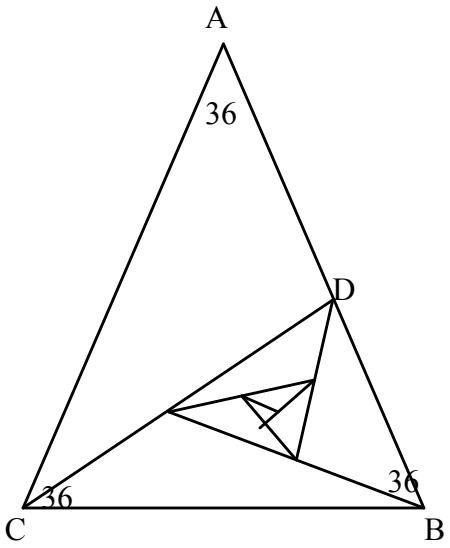
ORAN Genişlik/Uzunluk	EN GÜZEL DİKDÖRTGEN		EN ÇİRKİN DİKDÖRTGEN	
	Fechner (%)	Lalo (%)	Fechner (%)	Lalo (%)
1.00	3.0	11.7	27.8	22.5
0.83	0.2	1.0	1.7	16.6
0.80	2.0	1.3	9.4	9.1
0.75	2.5	9.5	2.5	9.1
0.69	7.7	5.6	1.2	2.5
0.67	20.6	11.0	0.4	0.6
0.62	35.0	30.3	0.0	0.0
0.57	20.0	6.3	0.8	0.6
0.50	7.5	8.0	2.5	12.5
0.40	1.5	15.3	35.7	26.6

Kenar uzunlukları oranı Altın Orana yakın olan diktörtgenlerin beğenilme yüzdesi büyük olarak gözlenmiştir. Karenin de beğenilmiş olmasına dikkat ediniz. Altın Dikdörtgene çirkin diyen bir tek kişi bile çıkmamıştır.

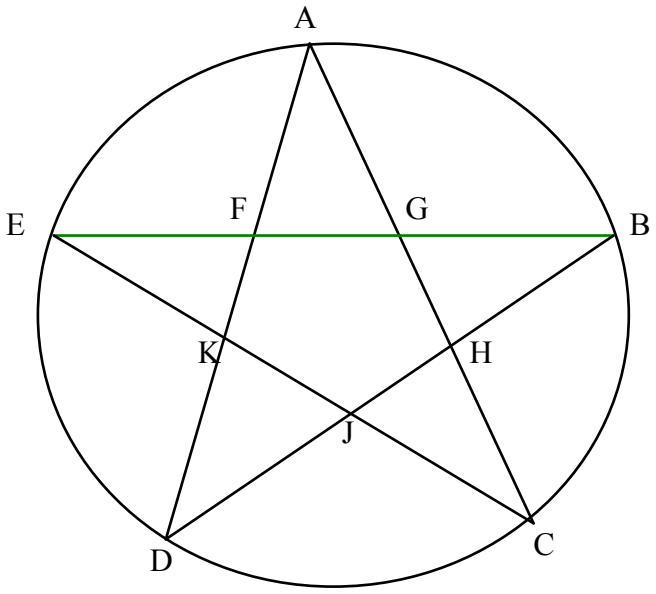
Mimaride, inşa edilecek yapının cephe görünüşünün daima bir Altın Dikdörtgen içine yerleştirilebilmesi dikkat edilecek ilk husus olmaktadır. Mimaride olduğu gibi, resimde de temel ögelerden biri Altın Oran'dır.

Altın Üçgen: Tepe açısı 36° olan ikizkenar üçgenlere Altın Üçgenler denir.

Altın Üçgen içine çizilen gitgide küçülen Altın Üçgenlerin köşelerinden de eşit açılı logaritmik spiral geçmektedir.



Düzgün bir ongen esasında 10 tane Altın Üçgen diliminden oluşmaktadır. Altın Üçgen düzgün beşgenlerde de karşımıza çıkmaktadır.



FGHJK düzgün beşgeninin kenar uzunluğu 1 birim olursa aşağıdaki özellikler yazılabilir.

$$AF = AG = BG = \ldots = \Phi$$
$$GJ = HK = JK = KG = FH = \Phi$$
$$KP / HP = GP / JP = \ldots = \Phi$$
$$OB / ON = OC / OS = \ldots = 2\Phi$$
$$ON / OK = \Phi / 2$$
$$OC / OF = \Phi^2$$
$$AB = BC = \ldots = \Phi^2$$
$$BD / OB = \sqrt{1 + \Phi^2}$$

Yıldızın kolları, çekirdeğini oluşturan FGHJK beşgeninin kenarlarından yukarıya doğru kıvrılıp birleştirilirse oluşan piramidin yüksekliğinin, tabanı çevreleyen çemberin yarıçapına oranı Φ dir.

1.20.3.İndirgeme Bağıntılarının Karmaşıklığına Ait Çeşitli Örnekler

Örnek.1.21 : Aşağıdaki fonksiyonların karmaşıklığını tahmin ediniz?

a) ($n\log n + n^2$) (n^3+2) b) ($n! + 2^n$) ($n^3 + \log (n^2+1)$)

Çözüm.1.21:

a) $\log n < n \Rightarrow n \log n < n^2$ ($n^2 + n^2$)

$n^3 + 2 \leq 2n^3$ ise ($2n^2$) ($2n^3$) olduğundan $O(n^2)$ **.O(** n^3) $= 0 (n^5)$

b) $n! < n^n$ $(n > 2)$ olduğundan

$\log (n^2 + 1) < \log 2n^2$ ve $\log (n^2 + 1) < \log 2 + 2.\log n$

$\log (n^2 + 1) < 3.\log n = O(\log n)$

$(n^2+2^n)(n^3 +\log(n^2 + 1))$

$n^n . n^3 = n^{3+n} = O(n^{3+n})$

Örnek.1.22 : Genel terimi $a_n = a_{n-1} + n$ ve başlangıç koşulu $a_0 = 1$ olan indirgeme bağıntısını ve karmaşıklığı bulunuz.

Çözüm.1.22:

$a_1 = a_0 + 1 \Rightarrow 1 + 1 = 2 = 1 + 1$

$a_2 = a_1 + 2 \Rightarrow 2 + 2 = 4 = 1 + 3$

$a_3 = a_2 + 3 \Rightarrow 4 + 3 = 7 = 1 + 6$

$a_4 = a_3 + 4 \Rightarrow 7 + 4 = 11 = 1 + 10$ ve buradan

$a_n = (1 + n . (n + 1)) / 2$ ise karmaşıklığı $n>2$ için $O(n^2)$ olur.

Örnek.1.23 : Genel terimi $a_n = 2n . a_{n-1}$ ve başlangıç koşulu $a_0 = 1$ olan indirgeme bağıntısı ve karmaşıklığı nedir.

Çözüm.1.23:

$a_1 = 2 . 1 = 2 . 1 . 1 = 2^1 . 1$

$a_2 = 4 . 2 = 2 . 2 . 2 = 2^2 . 2 . 1_3$

$a_3 = 6 . 8 = 2 . 3 . 2 . 2 . 2 = 2^3 . 3 . 2 .1$

$a_4 = 2 . 4 .a_3 = 2 . 4 . 2 . 3 . 2 . 2 . 2 = 2^4 . 4 . 3 . 2 .1$

.

$a_n = 2^n . n!$ ve karmaşıklığı $n>2$ için $O(n^{n+2})$ olur.

1.8. Optimizasyon

Optimizasyon, en basit anlamı ile eldeki kısıtlı kaynakların en uygun şekilde kullanılması olarak tanımlanır.

Matematiksel bir terim olarak ifade etmek gerekirse optimizasyon, kısaca bir fonksiyonun maksimize (en büyükleme) veya minimize (en küçükleme) edilmesi olarak tanımlanır.Başka bir ifade ile, optimizasyon, "en iyi amaç kriterinin en iyi değerini veren kısıtlardaki değişkenlerin değerini bulmak" şeklinde tanımlanır.

Pratikte bir optimizasyon problemi üç evrede çözülür. Bunlar;

i. Gerçek hayat problemleri için iyi bir model veya formülasyonun bulunması. Örneğin;
   a) model için matematik çözümler, uygulamaya uygun olmalıdır ve
   b) matematik çözümler elde edilebilir olmalıdır.
   c) İleri sürülen model için bir çözüm üretecek etkili metodların bulunması,

ii. Çözüm metodlarının uygulanması,

iii. Sonuçların yorumlanmasıdır.

Gerçek hayat problemlerinin modellenmesi bir sanat olup, çok kompleks problemlerin bunlarla ilgili temel modellerin anlaşılmasına bağlıdır.

1.8.1. Birleşik Optimizasyon Problemleri - BOP

Birleşik Optimizasyon (Combinatorial Optimization), kesikli nesnelerin seçimi, ya da bir optimal dizinin, grubun veya sıralmanın bulunmasındaki matematik çalışmadır.

Birleşik optimizasyon, uygun çözümlerin sonlu bir sayısı ile karakterize edilen problemlerle ilgilenir. Birleşik optimizasyonda karşılaşılan tipik problemler şunlardır: Verilen konumlarda imkanların en uygun kullanımı, müşterilerin en iyi gruplaması, makineler de işlerin optimal (en uygun) sıralaması, işlerin (müşterilerin) acentelere(araç rotalarına) optimal dağıtımı, değişik yatırım ihtimalleri arasında optimal seçim v.b.

BOP'lar, eğer uygun çözümlerin kümesi boş değilse, bir optimal çözümün olması anlamında ekseriye iyi tanımlıdır. Bu problemlerin çoğunda optimal çözümü hesap yoluyla bulmak zordur. Bu nedenle uygun bir hesaplama süresinde büyük ölçekli problemler için optimal çözümleri sağlayabilecek yaklaşık algoritmalara (*heuristik*) sahip olmak önemlidir.

Birleşik optimizasyon, bazen *kesikli programlama (discrete programming)* olarak da ifade edilir. Böylece bir *BOP,* uygun çözümlerin kesikli bir kümesi üzerinde optimize (maksimize veya minimize) amaç fonksiyonunun bir çözümünü araştırır.

Genel bir BOP ;

BOP:min f(x) **(2)**

Kısıtlar $x \in X$

şeklinde tanımlanır.

Burada, $X \subseteq \Omega$, Ω uzayında bir uygun bölgeyi belirtir. Bir başka deyişle, X, zorunlu kısıtları sağlayacak uygun çözümlerin bir kümesidir. $f: X \rightarrow R$ fonksiyonu amaç fonksiyonu olarak bilinir ve R gerçel değerlerin kümesidir. $x \in X$ uygun çözümü, eğer diğer bir y uygun çözümü $f(y)<f(x)$ eşitsizliğini sağlamazsa, optimal çözümdür. X ve Ω kümeleri, özel uygulamalara göre değişiklik gösterirler. Eğer, X ve Ω kümeleri birleşik veya kesikli ise, bir P problemi, *BOP* olarak adlandırılır. Örneğin, n elemanlı bir sonlu kümenin ailesi, 2^n sonlu elemanı içerir.

1.8.2. BOP'ların Karmaşıklığı ve Problem Sınıfları

BOP problemleri genellikle tanımlanması çok kolay; fakat çözmesi çok zor olan problemlerdir. Hesaplama karmaşıklığı teorisi Cook tarafından ortaya konulmuştur. Buna göre, algoritmaların hesaplama gereksinimleri ve pratikte karşılaşılan sınıflandırılmış problemler *zor* veya *kolay* olarak tasnif edilmiştir.

Hesap karmaşıklığı teorisi, pratikte karşılaşılan problemlerin algoritmaları ve önemli örneklerinin her ikisinin de hesaplamalarının sınıflandırılmasını konu alır. Bu teorinin en önemli konusu, *NP Complete (Non Deterministic Polynomial Time Complete) sınıfı* problemlerdir .

Eğer, belirli bir problemin her örneğini çözecek bir polinom zaman algoritması geliştirebilirsek, BOP'ların bu sınıfı *"kolay"* olarak adlandırılır ve *"P"* ile gösterilir. Polinom zaman algoritması, etkili bir algoritma olarak bilinir. Bir algoritmanın zaman karmaşıklığı, verilen bir problem örneğini çözmek için algoritmanın gereksinimi olan aritmetik işlem adımlarının sayısıyla ölçülür .

Bu durum, genellikle en kötü durum kriteri olarak bilinir. k, bir sabit ve n problem uzayının boyutunu göstermek üzere, eğer bir problem $O(n^k)$ hesap karmaşıklığında bir algoritmaya sahipse, bu problem kolaydır. Burada $O(n^k)$;

$a_k n^k + a_{k-1} n^{k-1} + \ldots\ldots\ldots\ldots\ldots + a_1 n + a_0$ (a_i'ler sabit) **(3)**

şeklinde bir fonksiyonu ifade eder. Bu şekilde bir karmaşıklık, polinom mertebeden olarak ifade edilirken; algoritma k. mertebeden polinom zaman algoritması olarak adlandırılır. Burada, k=3 olması tercih edilir. Eğer bir problem için o problemi çözecek etkili algoritmalar bulunamazsa, bu problem *"zor"* veya *"çözülemez"* olarak adlandırılır. Polinom zaman algoritması ile çözülemeyen *"zor"* problemler veya üstel işlem zamanı gerektiren, bilinen bütün problemler, üstel zaman algoritması ister.Üstel zaman algoritması isteyen problemlerden sadece küçük ölçekli olanlar çözülebilir. Dünyada bu konuda çalışma yapan araştırmacıların ortak görüşü şudur:

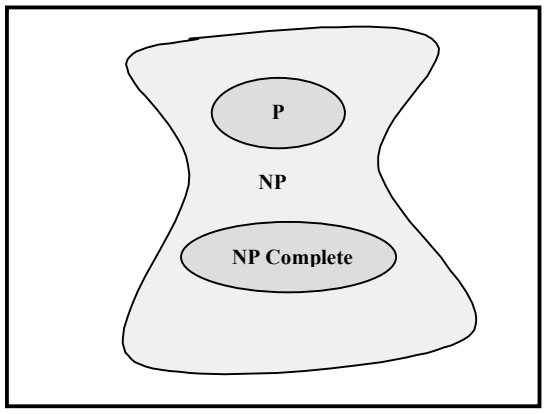
Zor olarak adlandırılan problemler için bu problemlerin herhangi birini çözebilecek etkili bir algoritma geliştirilmemiştir. Bu nedenle, bu zor BOP'ların çözümü için heuristikler ele alınmış ve algoritmalar sistemi içinde heuristik algoritmalar önemli bir yer bulmuştur.
Deterministik olmayan algoritmalar yardımıyla polinom zamanda çözülebilen karar problemlerinin sınıfı *NP(Non Deterministic Polynomial Time)* sınıfı olarak adlandırılır.

NP, *NP Complete (Non Deterministic Polynomial Time Complete)* olarak isimlendirilen problemlerin bir alt kümesini içerir. Bu alt kümedeki her bir problem, NP sınıfına aittir. Eğer, bu problem için etkili bir algoritma mevcutsa, NP sınıfındaki herbir problem için etkili bir algoritma mevcuttur. (Örneğin, P=NP) Bunun anlamı, NP Complete sınıfı problemlerin NP sınıfındaki en zor problem sınıfı olmasıdır.

Karşılaşılan yöneylem araştırması problemlerinin çoğu NP Complete sınıfındadır. Bir probleme uygulanan heuristikler, bazı sebeplerden ötürü NP complete olarak ele alınabilir

Eğer NP sınıfındaki bütün problemler polinom olarak bir probleme indirgenebilirse, bu problem *NP Hard* sınıfına aittir denir. Başka bir deyişle, eğer bir problem NP Hard sınıfına aitse, sadece *P=NP* olan bir polinom zaman algoritması ile çözülebilir .

Problemlerin bu sınıfları arasındaki ilişkiyi basit bir şema ile aşağıdaki gibi ifade edebiliriz.



Şekil 1.3 Farklı problem sınıfları arasındaki ilişki

NP Complete teorisi *"evet"* veya *"hayır"* cevabı gerektiren problemlerden oluşur. Bu evet/hayır cevabı isteyen problemlere, karar problemi denir. Herbir *BOP,* $\overline{BOP}$ ile gösterilen bir karar problemi ile birleştirilir. Verilen bir çözüm uzayı *X*, bir amaç fonksiyonu *f ve* α bir baslangıç değer ise, karar problemi, $f(x) \leq \alpha$ *? Gezgin Satıcı Problemi (Travelling Salesman Problem) (TSP)* şeklinde bir $x \in X$ uygun çözümü mevcut mu şeklinde bir soruyla tanımlanır.

Bunu TSP(Gezgin Satıcı Problemi) üzerinde tanımlayalım:Verilen bir grafik *G(N,A)* ve tamsayı yay uzunluğu C_{ij} olmak üzere herbir $(i,j) \in A$ yayı ile birleştirilsin.

Bu durumda TSP, networkde herbir düğümü dolaşarak;

$$f(w)= \sum_{(i,j) \in w} C_{ij} \qquad \textbf{(4)}$$

tur uzunluğunun mümkün en küçük değerinin tam olarak belirleneceği bir W turunu elde etmektir. TSP için karar problemi $\overline{\text{TSP}}$, W turunu içeren bir networkte f(W) toplam uzunluğunun α verilen bir tamsayı iken α'dan küçük olmasıdır. ($f(W)< \alpha$)

Eğer biz TSP'yi polinom zamanda çözebiliyorsak, TSP'yi optimal turun α'dan küçük olup olmadığını kontrol ederek TSP'nin karar problemine ($\overline{\text{TSP}}$'ye) bir cevap bulmak için kullanabiliriz.

Fakat bunun tersi doğru değildir. Yani, karar problemi $\overline{TSP}$),TSP'nin bir cevabını bulmada kullanılamaz. Böylece, $\overline{TSP}$ ve onun karar problemi olan $\overline{TSP}$, polinom zamanda çözümlerinin olup olmamasına göre eşittirler.Eğer BOP2 için bir algoritma kullanarak BOP1'i çözebilirsek, BOP1, BOP2'ye indirgendi denir. Buradan da TSP, TSP'ye indirgenebilir. Eğer BOP1'in herhangi bir örneği polinom zamanda BOP2'nin bir örneğine dönüştürülebilirse, BOP1 problemi, BOP2 problemine polinom olarak indirgenebilir. Bu, şunu gerektirir:

Eğer BOP1 polinom olarak BOP2'ye indirgenir ve bazı polinom zaman algoritması BOP2'yi çözerse, bu polinom-zaman algoritması BOP1'i de çözer.

Problemlerin dört sınıfı vardır: Bunlar sırasıyla P sınıfı, NP sınıfı, NP hard sınıfı ve NP Complete sınıfıdır .

P-Sınıfı: Eğer bazı polinom-zaman algoritması $\overline{BOP}$ karar problemini çözerse, BOP, P-sınıfına aittir denir. P-sınıfı problem örneklerinin bazıları; Minimum Maliyetle Şebeke (network) Akış Problemi, Atama Problemi, En Kısa Yol Problemi ve bunun gibi verilebilir.

NP Sınıfı: Bu sınıfa ait problemler iki şekilde karakterize edilirler;

- NP'deki her problem, sadece bütün sorulara birer birer cevap vererek çözülebilir.
- Karar problemlerinin ($\overline{BOP}$) herbiri, polinom zamanda çözülebilir.

NP Hard Sınıfı: Eğer NP sınıfındaki herhangi bir *BOP1* problemi *BOP*'a indirgenirse, $\overline{BOP}$ karar problemi NP hard sınıfına aittir denir.

NP Complete Sınıfı: Eğer bir $\overline{BOP}$ karar problemi NP ve NP hard sınıflarının her ikisine de aitse, BOP, NP Complete sınıfına aittir denir.

Bu problem sınıfları arasında $P \subseteq NP$ ve $NP\ Complete \subseteq NP$ şeklinde bir ilişki vardır. Bundan dolayı, *NP Complete sınıfı, NP* sınıfındaki en zor problem sınıfıdır.