Algorithm Analysis And Design

With Infinity Teams

Eng: Abdulrahman Hamdi

INFINITY TEAM
Since 2020

Table of contents

Eng: Abdulrahman Hamdi

Table of contents

Eng: Abdulrahman Hamdi

03

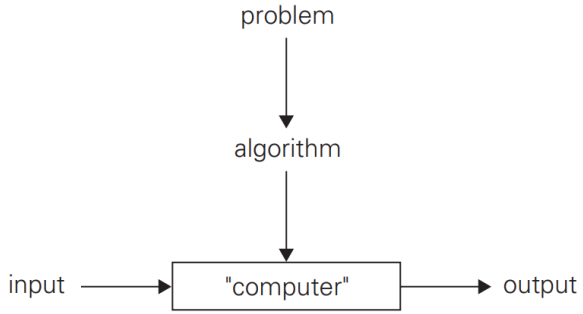
Introduction to Algorithm

Eng: Abdulrahman Hamdi

Algoritma Nedir?

Algoritma, Bir problemin çözümü için geliştirilmiş özel metot
– Girdileri çıktılara dönüştüren sıralı hesaplama adımları
– Tanımlanmış bir problemin çözümü için kullanılan Araç
– «Bir problemin çözümü için izlenilen sıralı ve anlaşılır buyruklar»

•Algoritma Hedefi:
– Sonlu bir zaman içinde
– Belirli girdiler ile
– İstenilen çıktıyı elde etmek



Eng: Abdulrahman Hamdi

Nasıl bir Algoritma ve Tarih?

Bir algoritma, Birden fazla biçimde sunulabilmeli
– Net ve anlaşılır olmalıdır
– Etkin ve faydalı olmalıdır
– Sonlu veya sonlandırılabilir olmalıdır
– Geliştirildiği problem için doğru olmalıdır

•Algoritmanın Tarihçesi:
• El Harezmi
– 9. yada yaşamış bir matematikçi
– Algoritma ve cebir kavramlarının «babası» olarak bilinir
– 0 sayısını ve x bilinmeyenini ilk kullanan kişi
• Euclid
– En büyük ortak böleni bulma problemi için geliştirdiği çözüm ilk algoritmalardan biri olarak kabul ediliyor

Eng: Abdulrahman Hamdi

EBOB Problemi

• **EBOB bulma problemi**
– İki negatif olmayan tam sayıyı kalansız bölen en büyük sayının bulunması
– gcd(m, n) = ?
• **Euclid Çözümü**
– (m mod n) işleminin sonucu 0 olana kadar
• gcd(m, n) = gcd(n, m mod n) işlemini tekrar et
– gcd(60, 24) = gcd(24, 12) = gcd(12, 0) = 12.

• **Euclid Çözümü**
– (m mod n) işleminin sonucu 0 olana kadar
• gcd(m, n) = gcd(n, m mod n) işlemini tekrar et
– gcd(60, 24) = gcd(24, 12) = gcd(12, 0) = 12.
– Adım 1 : Eğer n = 0 ise m değerini sonuç olarak döndür ve dur
• Değil ise 2. adıma git
– Adım2 : (m / n) işlemini yap, kalanı r'ye ata.
– Adım 3 : n değerini m'ye ata, r değerini n'ye ata (m = n, n = r).
Adım 1' dön.

Eng: Abdulrahman Hamdi

EBOB Problemi

• **Sözde Kod**

```
//Computes gcd(m, n) by Euclid's algorithm
//Input: Two nonnegative, not-both-zero integers m and n
//Output: Greatest common divisor of m and n
while n ≠ 0 do
    r ← m mod n
    m ← n
    n ← r
return m
```

• **Euclid'in algoritmasının sonu var mı?**

– N değeri her iterasyonda küçülüyor.

– 0'dan daha küçük, negatif, olamayacağı biliniyor.

• İki pozitif sayının bölme işleminden negatif sayı veya
kalan çıkamaz.

– Er yada geç sıfıra ulaşarak algoritma duracaktır.

EBOB Problemi

• **Ardışık Tamsayı Kontrol Algoritması ile Çözüm** (Consecutive integer checking algorithm)
– EBOB (m,n) ikilisinin küçük olanından daha büyük olamaz
• Min(m,n) değerinin EBOB olup olmadığı kontrol edilir
– Min(m,n) EBOB ise işlem sonlanır,
» Değil ise bir azaltılarak tekrar kontrol edilir (min(m,n)--)

• **Algoritma**
• Adım1: min{m, n} değerini t'ye ata.
• Adım 2: (m / t) işlemini yap, Eğer kalan = 0 ise Adım 3'e git;
– Değil ise adım 4'e git
• Adım 3 : (n/t) işlemini yap. Eğer kalan = 0 ise t değerini
sonuç olarak döndür
– Değil ise adım 4'e git
• Adım 4: t değerini 1 azalt, Adım 2'ye git

EBOB Problemi

• Euclid'in algoritmasından farklı olarak bu algoritma
– Değerlerden biri 0 ise doğru çalışmaz (Neden?)
– Algoritma girdilerinin kesin ve dikkatli bir şekilde belirlenmesi gerekliliği ve önemi
• **Asal Çarpanlara Ayırma**
– EBOB'u bulunacak sayılar asal çarpanlarına ayrılır
• Ortak asal çarpanların çarpımı EBOB'u verir
– Adım 1 : m'nin asal çarpanlarını bul
– Adım 2: n'nin asal çarpanlarını bul
– Adım 3: Adım1 ve Adım2'de hesaplanan asal çarpanlardan ortak olanları belirle
– Adım 4: Ortak asal çarpanları çarp

• Asal çarpanlara ayırma algoritmanın işlem karmaşıklığı Euclid'in algoritmasından daha fazla
• Asal çarpanlara ayırma işleminin algoritması açık değil

Asal Sayıları Bulma

• **Eratosthenes'in Eleme Algoritması?**

– 2 ile n arasındaki asal sayıları bulmak için kullanılır

– 1. iterasyonda 2'nin katları

– 2. iterasyonda 3'ün katları

– 3. iterasyonda 5'in katları elenir (?)

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
2	3		5		7		9		11		13		15		17		19		21		23		25
2	**3**		5		7				11		13				17		19				23		25
2	3		**5**		7				11		13				17		19				23		

• Sözde Kod:

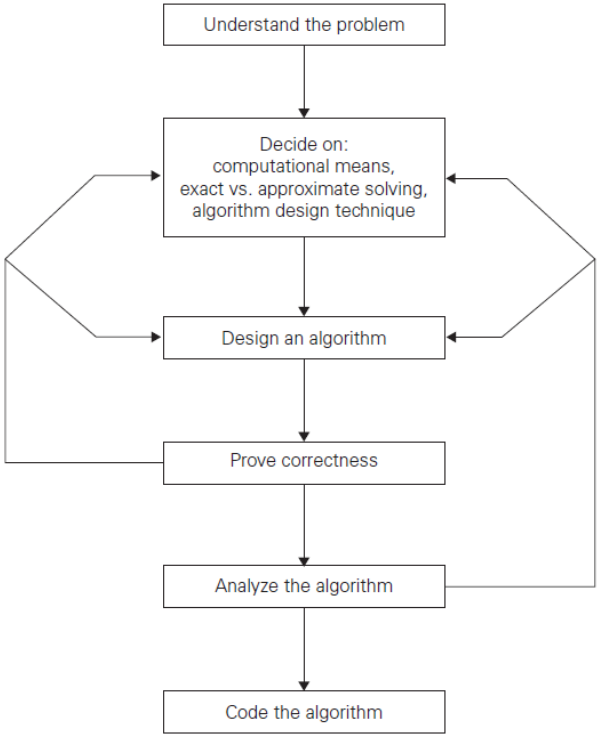
ALGORITHM *Sieve*(n)

//Implements the sieve of Eratosthenes
//Input: A positive integer $n > 1$
//Output: Array L of all prime numbers less than or equal to n

for $p \leftarrow 2$ **to** n **do** $A[p] \leftarrow p$
for $p \leftarrow 2$ **to** $\lfloor \sqrt{n} \rfloor$ **do** //see note before pseudocode
  if $A[p] \neq 0$ //p hasn't been eliminated on previous passes
    $j \leftarrow p * p$
    while $j \leq n$ **do**
      $A[j] \leftarrow 0$ //mark element as eliminated
      $j \leftarrow j + p$
//copy the remaining elements of A to array L of the primes
$i \leftarrow 0$
for $p \leftarrow 2$ **to** n **do**
  if $A[p] \neq 0$
    $L[i] \leftarrow A[p]$
    $i \leftarrow i + 1$
return L

Eng: Abdulrahman Hamdi

Algoritma Tasarım ve Analiz Süreci



Eng: Abdulrahman Hamdi

Algoritma Tasarım ve Analiz Süreci

1- Problemin Anlaşılması

– Problem net bir şekilde ifade edilmeli

• Anlaşılmayan hususlar için sorular

• Gerekliyse elle işletme

• İstisnai durumlar?

Genel olarak anlamadığımız bir şeyi bir başkasına öğretemezsiniz. Algoritma da çözümü bilgisayara öğretmektir. Problem yeterince anlaşılırsa çözümde detaylarıyla ortaya çıkar, ve bu detayların her biri bir adım olarak algoritmada yer alır. Ayrıca problemi yeterince anladığınızda ortaya çıkacak algoritmanın hangi girdilerle çalışacağına da karar vermiş olursunuz.

2- Sahip olunan / mevcut olan donanımı anlamak:

– RAM model

• Birçok temel algoritmanın denendiği sistem

Elimizdeki bilgisayarın neyi yapıp neyi yapamayacağından emin olmamız gerekir. Algoritmayı buna göre şekillendiririz. Örneğin elimizdeki bilgisayar bir GPU 'ya sahipse algoritmayı bu GPU üzerinde çalışacak şekilde tasarlayabiliriz.

Algoritma Tasarım ve Analiz Süreci

3- Probleme yaklaşık ya da tam çözüm geliştirmek:

– Tam sonuca ulaşılabilir mi? (Exact Algorithm)

– Yaklaşık sonuç mu bulunacak? (Approximation Algorithm)

• Lineer olmayan denklemler

• Karekök

– Bazı durumlarda kesin sonuca ulaşmak için

• Uzun işlem süresi

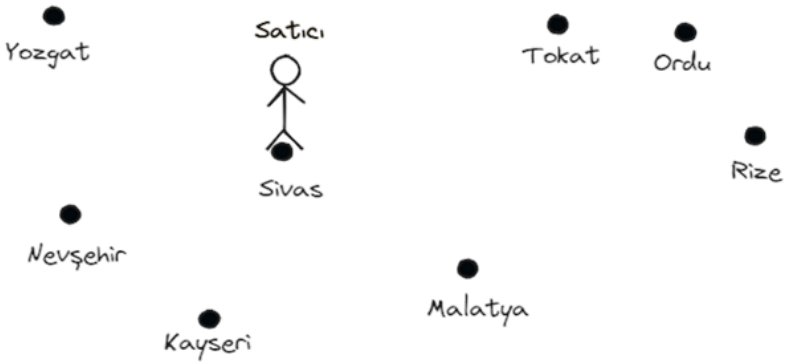
• Karmaşık işlemler gerekiyorsa yaklaşık sonuç kabul edilebilir

Elimizdeki probleme en iyi en kesin çözümü bulabiliyor muyuz , bunu anlamamız gerekir.
Ayrıca bu çözümün makul bir sürede çalışması ve hafıza da makul bir yer tutması gerekir.
Yoksa tam/kesin (exact) çözüm değil yaklaşık bir çözüm geliştirilir. Bu sorunun tam ,
kesin bir çözümü değildir, suboptimaldir , fakat belirli bir seviyeye kadar bizi tatmin eder
(hiç yoktan iyidir).
Yaklaşık çözüm bulmaya örnek olarak gezgin satıcı problemi (traveling salesman problem) verilebilir.
Bu problemde bir satıcının olabilecek en kısa yolu (shortest path)
izleyerek birtakım şehirleri ziyaret etmesi ve daha sonra başladığı yere dönmesi istenir.
Gezgin satıcı problemi örneği

Eng: Abdulrahman Hamdi

Algoritma Tasarım ve Analiz Süreci

3- Probleme yaklaşık ya da tam çözüm geliştirmek:



Satıcının olabilecek en kısa yolla tüm şehirleri ziyaret ederek tekrar Sivas 'a gelmesi gerekmektedir.
Bu tarz problemlerde şehir sayısı az iken olup olabilecek tüm yollar araştırılarak en kısa yola karar verilebilir. Bu kesin / tam çözüm olur. Fakat şehir sayısı arttıkça tam çözümü bulmak imkânsız olmaya başlar, bu durumda yaklaşık çözüm geliştirilir.

Algoritma Tasarım ve Analiz Süreci

4- Elimizdeki problem için genel olarak şu dizaynlardan / stratejilerden biri seçilir :

- Brute Force (Kaba kuvvet yaklaşımı)
- Böl ve yönet (Divide and conquer)
- Özyineleme (Recursion)
- Aç gözlü algoritma (Greedy approach).
- Dinamik programlama

5- Algoritma doğruluğunun test edilmesi:

– Denemeler
– İstisnalar
– İspatlar

Bir algoritmanın çalışmadığını / hatalı olduğunu göstermek kolaydır. Herhangi bir girdi için yanlış çıktı verdiği gösterilebilirse algoritmanın yanlışlığı gösterilmiş olur. Tıpkı bir teoremi çürütmek için tek bir örnek bulunması gibi, Fakat bir algoritmanın çalıştığını göstermek güçtür , kesin bir yöntemi yoktur. Yine de pozitif tam sayılarla çalışan bir algoritmanız varsa bunun doğruluğunu matematiksel tümevarım (induction) ile gösterebiliriz. Örneğin 1' den n 'e kadar olan sayıların toplamına dönen şu algoritmanın doğrulunu gösterelim. Fakat bundan önce tümevarımla ispatın ne olduğunu hatırlayalım.

Eng: Abdulrahman Hamdi

Algoritma Tasarım ve Analiz Süreci

1- Temel durum : Bir n için doğru olduğunu göster (n genelde 0 ya da 1 alınır)
2- Tümevarım durumu : Bir k için teoremin / algoritmanın doğru olduğunu kabul et .
k + 1 için de bu algoritmanın / teoremin doğru olduğunu ispat et.

topla(n)

// Girdi : n pozitif tam sayısı

// Çıktı : 1'den n'e kadar olan tamsayıların toplamı

Adım 1 : dön $\frac{n(n+1)}{2}$

Kanıt:

$n = 1$ için **topla(1)** = $\frac{1(2)}{2} = 1$ olup algoritma doğrudur.

$n = k$ için algoritma doğru olsun. Yani:

$$toplam(k) = 1 + 2 + ... + k$$

Bu durumda:

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

(Bunun doğru olduğunu kabul ediyoruz.)

$n = k + 1$ için:

$$toplam(k+1) = \frac{(k+1)(k+2)}{2} = 1 + 2 + ... + k + (k+1)$$

Bu kod her iki tarafa $(k+1)$ eklenirse:

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

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

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

$$\frac{(k+1)(k+2)}{2} = toplam(k+1)$$

olduğundan **algoritma doğrudur.**

Eng: Abdulrahman Hamdi

Algoritma Tasarım ve Analiz Süreci

6- Algoritma Analizi:
– Algoritmanın doğruluğu en önemli unsur
– Etkinlik
– Zaman Etkinliği
– Basitlik
5.aşamadan geçtikten sonra algoritmamızın doğru çalıştığına emin oluruz. En son olarak
bu algoritma
- Daha efektif yapılabilir mi ? (az zaman , az hafıza)
- Daha basitleştirilebilir mi ? (anlaşılabilirlik)
- Daha genelleştirilebilir mi ? (Daha başka soruların çözümünü kapsayacak şekilde
genelleştirilebilir miyiz ?)

Örneğin aldığı iki sayı aralarında asal mı değil mi diye karar veren bir algoritma , aldığı iki sayının önce ebob 'unu bulup sonra bu ebob 1 ise aralarında asaldır, değilse bu sayılar aralarında asal değildir şeklinde karar verirse bu algoritma aynı zamanda ebob bulma için de kullanılabilir.
Şimdi son olarak aldığı bir dizideki elemanlar arasındaki en küçük uzaklığa dönen bir brute force algoritmanın nasıl daha efektif hale getirilebileceğine bakalım.

Eng: Abdulrahman Hamdi

Algoritma Tasarım ve Analiz Süreci

ALGORITHM *MinDistance*$(A[0..n-1])$

//Input: Array $A[0..n-1]$ of numbers

//Output: Minimum distance between two of its elements

$dmin \leftarrow \infty$

for $i \leftarrow 0$ **to** $n-1$ **do**

for $j \leftarrow 0$ **to** $n-1$ **do**

if $i \neq j$ **and** $|A[i]-A[j]| < dmin$

$dmin \leftarrow |A[i]-A[j]|$

return $dmin$

Bu algoritmanın sorunu hesapladığı uzaklıkları birden fazla kez hesaplayarak tekrara düşmesidir.
Örnek olarak A = [-2,4,9,3,6] gibi bir dizi alalım.

Explanation of Redundant Distance Calculations in the Algorithm

For an array **A = [-2, 4, 9, 3, 6]**, the algorithm iterates through all possible pairs (i, j) to compute the absolute differences:

When $i = 0$:

- **Comparisons made:**
  - $|-2-4|$
  - $|-2-9|$
  - $|-2-3|$
  - $|-2-6|$

→ **All distances from $A[0] = -2$ to the rest of the elements are calculated.**

When $i = 1$:

- **Comparisons made:**
  - $|4-9|$
  - $|4-3|$
  - $|4-6|$
  - $|4-(-2)|$ (This was already calculated when $i = 0$, leading to redundancy.)

→ **The distance between $A[1] = 4$ and $A[0] = -2$ is recalculated, which is unnecessary.**

Eng: Abdulrahman Hamdi

Algoritma Tasarım ve Analiz Süreci

Burada yapılması gereken tekrara düşmemek adına j ' nın i +1 'den başlaması geriye dönmemesidir. Bu durumda

Further Redundancy in Distance Calculation

For the array **A = [-2, 4, 9, 3, 6]**, the algorithm processes each pair of elements:

- **When $j = 1$:**
  - Comparisons are made with all subsequent elements:
    - $|4 - 9|$
    - $|4 - 3|$
    - $|4 - 6|$

This visualization reinforces the redundancy problem:

- The absolute difference between $A[0]$ **and** $A[1]$ (i.e., $|-2-4|$) was calculated earlier.
- Similarly, when $j = 1$, comparisons with elements at later indices (e.g., 9, 3, 6) are repeated.

O halde yeni efektif algoritmamız şöyle olur:

```
minMesafe(A[0, ..., n-1])
dmin ← ∞

for i = 0 to n-1:
    for j = i+1 to n-1:
        if |A[i] - A[j]| < dmin:
            dmin ← |A[i] - A[j]|

return dmin
```

Eng: Abdulrahman Hamdi

Algoritma Tasarım ve Analiz Süreci

7- Algoritma Açıklama Yöntemleri:
– Akış diyagramı
– Sözde kod
– Kelimeler ile anlatım

8- Algoritmanın kodlanması:
– Belirlenen özelliklerin, veri yapılarının uygulanabileceği bir dil seçilmesi

Eng: Abdulrahman Hamdi

Önemli Problem Türleri

- Sıralama
- Arama
- String İşlemleri
- Graph problemleri
- Kombinasyonel Problemler
- Geometrik problemler
- Nümerik problemler

Önemli Problem Türleri

• **Sıralama**
– Bir listedeki öğeleri artan sırada düzenlemek
• Girdi : n adet sayıdan oluşan dizi $< a_1, a_2, \dots, a_n >$
• Çıktı : Girdinin $a_1 \leq a_2 \leq \cdots \leq a_n$ Şeklinde yeniden düzenlenmesi
– Neden Sıralama?
• Arama işlemini kolaylaştırmak
• Birçok algoritmanın altyordamı
– Sıralama Anahtarı
• Veri bütününün sıralamayı yönlendirmek için seçilmiş özel parçası
– Bir başarı listesinin sıralanması (İsme, numaraya, nota göre)

Örnek sıralama algoritmaları
– Selection
– Bubble sort
– Insertion sort
– Merge sort
– Heap sort …

Önemli Problem Türleri

• **Sıralama**
• Sıralama algoritması karmaşıklığını değerlendirmek
– Yapılan Karşılaştırma sayısı
• Sıralama algoritmaları için iki önemli özellik
– Stabilite: İki eşit elemanın birbirlerine göre sıralanmadan önceki pozisyonlarında kalması
• Bir dizide birbirine eşit iki eleman var.
• Sıralamadan önceki konumları i ve j, i<j
• Sıralamadan sonraki konumları i' ve j' ve i'<j' ise algoritma stabil bir sıralama algoritmasıdır
– Yerinde (In place) :
• Fazladan bellek alanı gerektirmeyen sıralama algoritmaları
• İstisna olarak küçük bellek birimleri kullanabilir
– Dizinin tamamı kadar değil, bir değişken kadar

Önemli Problem Türleri

- **Arama**
  - Bir değeri verilen veri seti içinde bulma
- Sıralama algoritması örnekleri
  - Sıralı arama
  - İkili arama…
- Her durum için ideal arama algoritması yok
  - Hızlı fakat fazla bellek alanı
  - Sadece sıralı verilerde arama
- Sıralı dizi içerisinde 19 değeri aranıyor
- 1,2,3,5,6,7,8,10,12,13,15,16,18,19,20,22
  - 1,2,3,5,6,7,8,10
  - 12,13,15,16,18,19,20,22
- 12,13,15,16
- 18,19,20,22
  - 18,19
  - 20,22

```
Procedure ikiliarama(x:integer, a1,a2,...,an: artan tamsayılar)
i:=1; {i, arama aralığının sol bitiş noktasını gösterir}
j:=n; {j, arama aralığının sağ bitiş noktasını gösterir}
while i<j do
begin
   m:=[(i+j)/2];
   if x>am then i:=m+1;
   else j := m;
end
if x=ai then konum:=i
else konum:=0;
{konum, x'e eşit olan terimin indisidir veya eğer x
   bulunamamış ise değeri sıfırdır}
```

Eng: Abdulrahman Hamdi

Önemli Problem Türleri

- **String İşleme**
- Bir alfabede yer alan karakterlerden oluşmuş dizi
  – Alfabetik, nümerik, özel karakterlerden oluşabilir
  – Örnek
- Bir metin içerisinde www ifadesini aramak
- @ karakterini aramak
- Derleyiciler

- **Graph**
- Birbirine sınır adı verilen hatlarla bağlı noktalar grubu
  – Gerçek hayat problemleri
- WWW modellemesi
- Haberleşme ağları
- Proje planlaması
  – Graph örnekleri
- En kısa yol
- Topolojik sıralama

Önemli Problem Türleri

- **Kombinasyonel problemler**

– Çeşitli kısıtlılıkları sağlayan kombinasyonel çözümlerin bulunması

• Maksimum değer, minimum maliyet

• Sorunlar

– Probleme göre kombinasyonel çözümler çok büyüyebilir

– Bu tip problemleri, kabul edilebilir bir sürede çözebilecek bilinen bir algoritma yok

- **Geometrik problemler**

– Nokta, doğru, çokgen gibi geometrik nesneler ile ilgilenen formülleri

• Uygulama Alanları

– Robotik

– Bilgisayar Grafikleri

– Tomografi

• En bilinen problemler

– En yakın çift (Closest Pair)

– Dışbükey gövde (Convex Hull)

Önemli Problem Türleri

- **Nümerik Problemler**

– Yaklaşık çözüme ulaşılmış problemlerin kesin sonuca ulaştırılmaya çalışılması

• Denklem, denklem sistemi çözümleri

• Kısıtlı integraller

- **Geometrik problemler**

– Nokta, doğru, çokgen gibi geometrik nesneler ile ilgilenen formülleri

• Uygulama Alanları

– Robotik

– Bilgisayar Grafikleri

– Tomografi

• En bilinen problemler

– En yakın çift (Closest Pair)

– Dışbükey gövde (Convex Hull)

Eng: Abdulrahman Hamdi

En Yakın Çift Problemi

- Verilen nokta kümesi içerisinde birbirine en yakın noktaları bulmak

NOKTA	X KOORDİNATI	Y KOORDİNATI
A	3	2
B	2	5
C	5	-3
D	-2	0
E	4	6
F	0	-4

- İki Nokta Arası Mesafe :

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Eng: Abdulrahman Hamdi

Metin Arama

- Bir metin içerisinde aranan ifadeyi bulmak
  – Büyük / Küçük Harf duyarlılığı ?
  – Tam uyum / Ek almış ?

Eng: Abdulrahman Hamdi

Algorithm Analysis And Design

Thanks for listening!

Eng: Abdulrahman Hamdi