

Algoritmalar

1. Hafta 25.09.2012

Algoritma: Bir problemin çözümü için ortaya koymuş şiralı, adım adım ve olus伦li çözüm yollarıdır.

Operasyonlar:

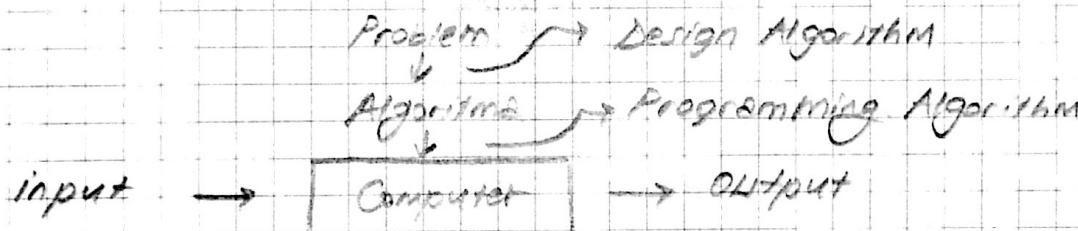
- Sequential → Sıralı islemler
- Conditional → Teslim " "
- Iterative → Döngülü "

Algoritma:

- Belirsizlik içermemeli (Definiteness)
- Verimli çalışmalıdır (Effectiveness)
- Kesinlikle doğru sonuc vermelidir. Bir inputta cevap yoksa bonus error mesajı okunur verilmelidir. (Correctness)
- Sonlu bir zamanda bitmelidir. (Holling) (Finiteness)

Algoritmaların ifade Lütfenisi:

1. Natural Languages
2. Pseudocode
3. Flowcharts
4. Programming languages
 - ↳ insan tarafından okunaklı 30%
 - ↳ Baska bir dile çevrilmesi 30%



Algoritma Design Etme:

1- Problem anla

- 2- Problemin nasıl çözüleceğine karar verme
- Yaklaşık bulma
 - Hangi veri yapısı kullanılacak
 - Hangi algoritma təcigi kullanılacak.

3- Algoritma dizayn etme

4- Doğruluğunu kontrol etme

Matematiksel ispatla

5- Algoritma Analizi yap

6- Algoritmayı Kodla

Algoritma Analizi:

Tarafsızlığı: Algoritma ne kadar zaman alıyor?
Algoritma ne kadar yer kaplıyor?

Yer tarafsızlığı:

Program geliştirildiğinde input için, output için ve programın içeriğine ayrı ayrı yerler gereklidir.

Zaman tarafsızlığı:

Saat cinsinden bir şenin upletir.

Adım sayısına göre ölçüt.

Primitive operasyonlar cinsinden bulunur.

Input'un size'sı zaman tarafsızlığında etkilidir.

Running \rightarrow input size

$$\text{Time } T(n) = \underbrace{\text{Cop.}}_{\text{Machineye bağlıdır.}} \cdot \underbrace{C(n)}_{\substack{\text{Operasyon} \\ \text{Sayısı}}} \Rightarrow \boxed{\Theta(C(n)) = T(n)}$$

\hookrightarrow Cop algoritmaya bağlı değil.

Machineye bağlıdır.
Operasyonun yapılma
sayısı

O'ri $\frac{1}{2}$

sum=0
for i=1 to n
 sum = sum + a[i]
report sum

total

1

atama

n+1

karsileştirme

n

atama

1

print

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

Temel operasyon toplamıdır.

* Recursive olmayan algoritmalarde zaman tarafsızlığı için:

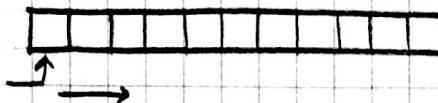
- input size'sı karar ver.
- Temel operasyony bul.
- Worst, average, best case durumlarını bul. (n için)

Worst Case: input size'n ıst sınırları olan running time ($T(n)$)

Best Case: input size'n alt sınırları ıst.

Average Case: Ortalama sonucu.

ör: Linear Search: her elemente sırayla bakarak, bulma.



Best Case: ilk baktığım, aradığım ıstorsa : $T(1) = 1$

Worst Case: Aradığım listede yoksa bütün elementlere bakımlı.

Average Case:

$$(\text{Orasılık}) P(I) = \frac{1}{n}$$

i: elementin arıyorumuz

$$T(I) = c$$

$$T(n) = \sum_{i=1}^n P(I) \cdot T(I)$$

$$\Rightarrow \frac{1 \cdot n(n+1)}{2} \sim$$

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

Approach: Temsilcimiz: algoritma. Ayrınlıkları yottur.

ör: *Insertion Sort: sıralanmış listeye element ekleyerek sıralama.

```
{  
    for j ← 2 to length[A]  
        do key ← A[j]  
        i ← j - 1  
        while i ≥ 0 and A[i] > key  
            do A[i + 1] ← A[i]  
            i ← i - 1  
        A[i + 1] ← key  
}
```

Temel Operasyon: tərkisishmə.

times

n
n-1
 $\sum_{i=1}^{n-1} i$
 $\sum_{i=1}^{n-1} i$
n-1

• Best Case: sıralıdm
 $T(n) = \Theta(n)$

• Worst Case: ters-sıralıdm
 $T(n) = \Theta(n^2)$

• Average Case: $j/2$
 $T(n) = \Theta(n^2)$

Polinom Hesaplama:

$$\hookrightarrow f(x) = a_0 \cdot x^0 + a_1 \cdot x^1 + a_2 \cdot x^2 + \dots + a_n \cdot x^n$$

$a_i \cdot x^i \rightarrow$ hesaplantırın (temel operasyon çarpma)

- $\underbrace{a_1 \cdot x \cdot x \cdot x \dots}_{i \text{ tane çarpma}}$

i tane çarpma

- $x \cdot x = x^2 \rightarrow 1 \text{ çarpma}$

$$a_1 \cdot x^2 \cdot x^3 \dots x \rightarrow \frac{1}{2} + 1 \text{ çarpma}$$

$$\underline{\underline{\underline{1/2 + 2}}}$$

Büyük sayılar da 2.
hesaplama deðer 1/2 li
çarplama.

$$\hookrightarrow P(n) = x \cdot P(n-1) + a_0 \quad \left. \begin{array}{l} n \text{ çarpma} \\ n \text{ toplama} \end{array} \right\}$$

$$P(n-1) = x \cdot P(n-2) + a_1 \quad \left. \begin{array}{l} \\ \\ \hline 2n = T(n) \end{array} \right\}$$

...

★ Her problem algoritmik olarak çözülebilir mi?

Halting Problem: Bir problemin her durumda çözülebilir ve çözülemez olamayız. (Unsolvable)

Hamiltonian Cycle Problem: Girdi içinde Hamilton Cycle's var mı sorusuna polinomik surekle certip cevap verilebilir. Ama çözülebilir. (NP-Complete Problem)

★ Problemin Tersanalılığı

Bir problemin çözümü için yapsılımız algoritmanın worse case denilen türküpü problemin tersanalığıdır.

O'n: Sort problemi için biriken en iyi algoritma Merge Sort
 $\min(\text{sorting}) = n \log n$

• Optimal algoritma: Best Case = Worst Case

O'n: Merge Sort $\rightarrow n \log n$

dr.*

Algorithm Enigma ($A[0, 1, \dots, n-1, 0, 1, 2, \dots, n-1]$)

```
if (n is odd)
    for i=0 to n-2 do
        if (A[i,i]=0)
            return false
    else
        for i=0 to n-2 do
            for j=i+1 to n-1 do
                if A[i,j] != A[j,i]
                    return false
    return true
```

if (n odd)

$$\left. \begin{array}{l} T_b(n) = \Theta(1) \\ T_w(n) = \Theta(n) \end{array} \right\} T(n) = O(n) = \Omega(1)$$

else (n is even)

$$\left. \begin{array}{l} T_b(n) = \Theta(1) \\ T_w(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 = \Theta(n^2) \end{array} \right\} T(n) = O(n^2) = \Omega(1)$$

→ Toplamda $T_b(n) = \Theta(1)$ $\left. \begin{array}{l} T(n) = O(n^2) = \Omega(1) \\ T_w(n) = \Theta(n^2) \end{array} \right\}$

Lecture 2 - Recurrence Relations

2. Hafta
02.10.2012

- Recursive formulation can be made once iterative version is easier to think

$$\left. \begin{array}{l} \text{foo}(n) \\ \equiv \\ \text{foo}(n/2) \\ \equiv \end{array} \right\} T(n) = f(n) + T(n/2)$$

Iterative
version

$$T(n) = c$$

dri: • Largest ($a[1, 2, \dots, n]$)

```

if n==1 return a[1]
else
    max = Largest (a[1, ..., n-1])
    if (max < a[n])
        return a[n]
    else
        return max
  
```

- input size: n

- terminal operation: \leq comparisons ($<$)

- iterative state + tree (K) approach

$$\boxed{\begin{aligned} T(n) &= 1 + T(n-1) \\ T(1) &= 0 \end{aligned}}$$

→ tree branching required.

$$\left. \begin{array}{l} T(1)=0 \\ T(2)=1 \\ T(3)=2 \\ \vdots \end{array} \right\} T(n)=n-1 \text{ diyelim. Bu } n \text{ ispatlayalım.}$$

Largest

≡

Largest

≡

$$\boxed{f(n)=1}$$

$$k=1 \text{ için } T(1)=0 \quad \checkmark$$

$$k=k-1 \text{ için } T(k-1)=k-2 \text{ olsun}$$

$$k=k \text{ için } T(k)=k-1 = 1+k-2 = \boxed{1+T(k-1)} \text{ doğrudur.}$$

$$\boxed{T(n)=n-1} \quad T(1)=0$$

2.C öðm

```

largest a[1, ..., n]
if n == 1 return a[1]
else
    max1 = largest (a[1, ..., ⌊n/2⌋])
    max2 = largest (a[⌈n/2⌉, ..., n])
    if max1 > max2
        return max1
    else
        return max2
  
```

$$\left. \begin{array}{l}
 \text{- Input size: } n \\
 \text{- terminal operation: } (>) \\
 \text{- } T(1) = 0 \\
 \text{- } f(n) = 1
 \end{array} \right\} \boxed{T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1} \quad \boxed{T(1) = 0}$$

$$\left. \begin{array}{l}
 T(1) = 0 \\
 T(2) = 1 \\
 T(3) = 2 \\
 T(4) = 3
 \end{array} \right\} \quad T(n) = n - 1 \quad \text{is partially true.} \quad \xrightarrow{\text{Forward Substitution}}$$

$$k=1 \text{ when } T(1)=0 \quad \checkmark$$

$$k=k/2 \quad T(k/2) = \lfloor k/2 \rfloor - 1$$

$$k=k/2 \quad T(\lceil k/2 \rceil) = \lceil k/2 \rceil - 1$$

$$\begin{aligned}
 k=k & \quad T(k) = \lfloor k/2 \rfloor + \lceil k/2 \rceil - 2 + 1 \\
 & = * -
 \end{aligned}$$

• Recursive Relations in computer sci.

Tan çözüm : - Forward substitution
- Backward " "
- Diferansiyel.

Asymptotic çözüm : - Guess and Prove
- Recursion Tree
- Master Theorem

Backward Substitution

ör:

$$\begin{aligned} T(n) &= T(n-1) + 1 \\ T(1) &= 0 \end{aligned} \quad \left. \right\}$$

$$\begin{aligned} T(n) &= T(n-1) + 1 \\ T(n) &= T(n-2) + 2 \\ T(n) &= T(n-3) + 3 \\ \dots \end{aligned}$$

$$\boxed{T(n) = T(n-k) + k} \rightarrow \text{yeni pattern elde ettil. Ispatlayalım.}$$

$$k=1 \text{ iken } T(n) = T(n-1) + 1 \quad \checkmark$$

$$k=i \text{ iken } T(n) = T(n-i) + i \text{ olsun.}$$

$$k=i+1 \text{ iken } T(n) = \underbrace{T(n-(i-1)}_{T(n-i)+i} + i+1$$

$$T(n) = T(n-i) + i \quad \text{degindir.}$$

• $i=n-1$ olsun. ($n-i=1$ olsun diye)

$$\boxed{T(n) = n-1}$$

• Recursive Relationlerin çözümüne rast.

Tan. çözim: - Forward substitution
- Backward "
- Diferansiyel.

Asymptotic çözim: - Guess and Prove
- Recursion Tree
- Master Theorem

Backward Substitution

Dr:

$$\begin{aligned} T(n) &= T(n-1) + 1 \\ T(1) &= 0 \end{aligned} \quad \left. \right\}$$

$$\begin{aligned} T(n) &= T(n-1) + 1 \\ T(n) &= T(n-2) + 2 \\ T(n) &= T(n-3) + 3 \\ \dots \end{aligned}$$

$$\boxed{T(n) = T(n-k) + k} \rightarrow \text{yeni şartın elide etmek. İşlem kaydırma.}$$

$$k=1 \text{ için } T(n) = T(n-1) + 1 \quad \checkmark$$

$$k=i \text{ için } T(n) = T(n-i) + i \text{ olsun.}$$

$$k=i+1 \text{ için } T(n) = \underbrace{T(n-i-1)}_{T(n-i)+i} + i+1$$

$$T(n) = T(n-i) + i \quad \text{olğundur.}$$

• $i = n-1$ olsun. ($n-i = 1$ olun diye)

$$\boxed{T(n) = n-1}$$

dr: Towers of Hanoi

$$\begin{aligned} T(n) &= 2T(n-1) + 1 \\ T(1) &= 1 \end{aligned} \quad \left. \begin{array}{l} \text{backward substitution yapalim.} \\ \text{---} \end{array} \right.$$

$$T(n) = 2 \cdot T(n-1) + 1$$

$$T(n) = 2^2 \cdot T(n-2) + 2 + 1$$

$$T(n) = 2^3 \cdot T(n-3) + 2^2 + 2 + 1$$

$$T(n) = 2^i \cdot T(n-i) + (2^i - 1)$$

ispatla.

- $n-i = 1$ için
 $i = n-1$ olur.

$$T(n) = 2^{n-1} \cdot 1 + 2^{n-1} - 1 = \underline{\underline{2^n - 1}} = \Theta(2^n)$$

Not: Her sefer forward ve backward substitution'ı kullanırız.
o're: fibonacci numbers.

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

$$T(0) = T(1) = 1$$

03.10.2012

Linear Recurrences with Constant Coefficients

→ Toplam halinde ve sadece tətbiqli ifadeler kullanılır.

$$\text{o're: } f(n) = -f(n-1) + 6 \cdot f(n-2) + 2^n - 1 \quad f(0) = \frac{1}{4}, \quad f(1) = \frac{1}{4}$$

$$r^2 + r - 6 = 0$$

$$(r-2)(r+3) = 0 \quad \text{homogen çözümü:}$$

$$f(n) = \underbrace{A \cdot 2^n + B \cdot (-3)^n}_{\text{Homogen çözüm}}, \quad \underbrace{Cn \cdot 2^n + D}_{\text{Partial çözüm}}$$

Homogen çözüm, Partial çözüm.

- C ve D ığın formüle yerine boyalım.

$$Cn \cdot 2^n + D + C(n-1) \cdot 2^{n-1} + D = 6C(n-2) \cdot 2^{n-2} + 2^n - 1 + 6D$$

$$-9D - 2C \cdot 2^{n-2} = -12C \cdot 2^{n-2} + 2^n - 1$$

$$D = \frac{1}{4}$$

$$C = \frac{2}{5}$$

- A ve B ığın $f(1) = f(0) = \frac{1}{4}$ və kullanalım.

$$A = -\frac{4}{5}$$

$$B = \frac{4}{5}$$

Not:

1) Çarpım durumunda ifadeler iken:

$$f(n) = f(n-1) * f(n-2) \quad \text{her taraflı logunu al.}$$

$$\rightarrow \log f(n) = \log f(n-1) + \log f(n-2)$$

$$g(n) = g(n-1) + g(n-2), \rightarrow \text{cümlebilir formda.}$$

2) Sabit olmayan kat sayılar iken:

$$(n-1).f(n) = n.f(n-1) + (n-1) \quad n.(n-1) \text{'e böl.}$$

$$\rightarrow \frac{f(n)}{n} = \frac{f(n-1)}{n-1} + \frac{1}{n}$$

$$g(n) = g(n-1) + \frac{1}{n} \rightarrow \text{cümlebilir formda.}$$

3) $T(n) = T(n/2) + T(n/4) + n$ gibi ifadelerde,
 n yerine 2^m yaz.

$$T(2^m) = T(2^{m-1}) + T(2^{m-2}) + 2^m$$

$$g(m) = g(m-1) + g(m-2) + 2^m \rightarrow \text{cümlebilir.}$$

Asymptotic Solutions

- Guess and Prove

Dr. $T(n) = 2 \cdot T(\lfloor n/2 \rfloor) + n$ $\quad (\forall i \geq 1)$
 floor ifadesinin uygunluğunu.
 $T(n) = 2 \cdot T(n/2) + n \rightarrow$ Merge sort algoritmasına benzeyen.
 $T(1) = O(n \log n)$ olursa düşünelim, işaret yapalım.

$$T(n) \leq c \cdot n \cdot \log n \quad n \geq n_0$$

$$n=1 \text{ iken } T(1) \leq 0 \text{ olamaz.}$$

$$n=2 \text{ iken } T(2) = 2 \cdot T(\lfloor 1 \rfloor) + 2 = 4 \leq c \cdot 2$$

$$c \geq 2 \quad \text{için uygun.}$$

$$T(n) \leq cn \log n \quad c \geq 2 \text{ için sağlı. ispatlayalım.} \rightarrow$$

$n = k$ için

$$T(k) = 2 \cdot T(\lfloor \frac{k}{2} \rfloor) + k$$

$$n=2 \text{ için } T(2) = 4 \leq 2c \quad \checkmark$$

$$n = \lfloor \frac{k}{2} \rfloor \text{ için } T(\lfloor \frac{k}{2} \rfloor) \leq c \lfloor \frac{k}{2} \rfloor \cdot \log \lfloor \frac{k}{2} \rfloor \text{ olsun.}$$

$$n = k \text{ için } T(k) \leq 2 \cdot c \lfloor \frac{k}{2} \rfloor \cdot \log \lfloor \frac{k}{2} \rfloor + k$$

$$\leq c \cdot k \cdot (\log k - 1) + k = ck \cdot \log k + k(1-c)$$

$$\boxed{T(k) \leq ck \cdot \log k}$$

dogrular.

Dr.

$$T(n) = T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + 1 \text{ için}$$

$$T(n) = 2 \cdot T(\lfloor \frac{n}{2} \rfloor) + 1 \quad \underbrace{\text{olduğunu düşünelim}}_{1} \quad \log \text{deki büyütür.}$$

- $T(n) = n$ için düşünelim. $T(n) \leq cn$

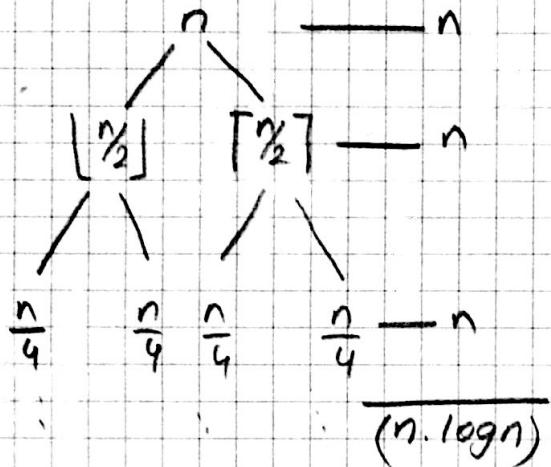
$$T(n) \leq c \cdot 2 \cdot \frac{n}{2} + 1 = cn + 1 \leq cn \quad \underline{\text{yanlış!}}$$

- $T(n) = cn - b$ için düşünelim. $T(n) \leq cn - b$

$$T(n) \leq 2 \cdot c \cdot \frac{n}{2} - 2b = cn - 2b \leq cn - b \quad \text{değildir. } b \geq 1 \text{ için.}$$

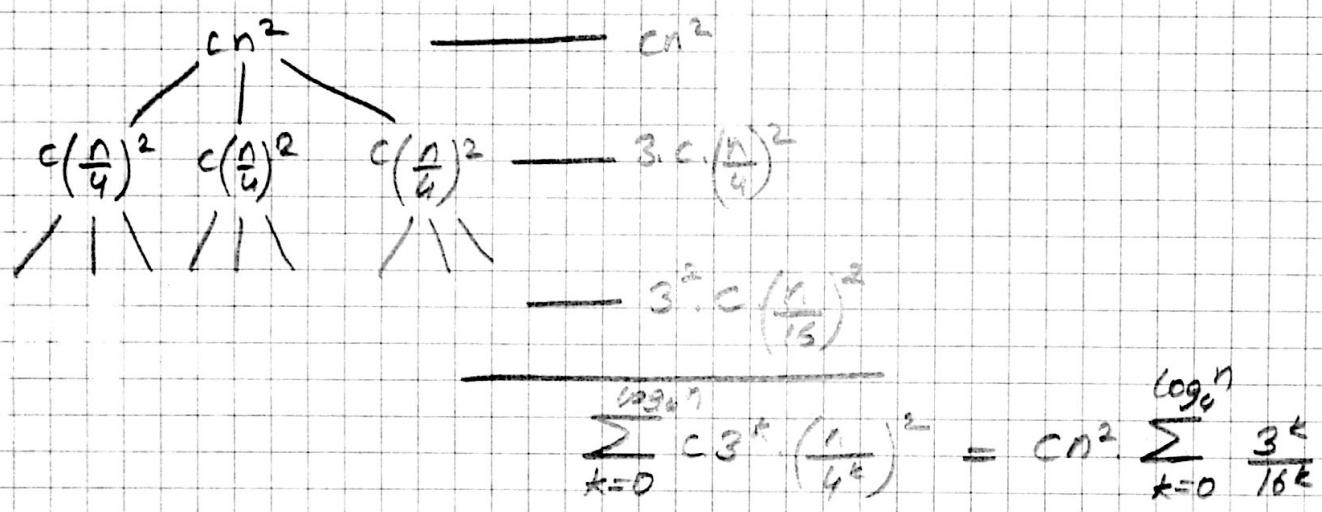
Recursion Tree

$$T(n) = T(\lceil \frac{n}{2} \rceil) + T(\lfloor \frac{n}{2} \rfloor) + n \quad \text{icin}$$



Orij:

$$T(n) = 3 \cdot T(\lfloor \frac{n}{4} \rfloor) + \Theta(n^2)$$



$$cn^2 \cdot \sum_{k=0}^{\log_4 n} \left(\frac{3}{16}\right)^k \leq cn^2 \sum_{k=0}^{\infty} \left(\frac{3}{16}\right)^k = cn^2 \cdot \frac{1}{1 - \frac{3}{16}} = \frac{16c \cdot n^2}{13}$$

$$T(n) = O(n^2) \text{ olker.}$$

Not:

$$\text{En alt seviyedeki ölenen sayısi } 3^{\log_4 n} = n^{\log_4 3} < n^2$$

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

Master Teorem:

$a \geq 1, b > 1$ iken $T(n) = a \cdot T(n/b) + f(n)$ tanımlanmıştır.

$T(n)$ için söylenebilecek 3 durum vardır:

1) $f(n) = O(n^{\log_b^a - \epsilon})$ $\epsilon > 0$ ise $T(n) = \Theta(n^{\log_b^a})$

2) $f(n) = \Theta(n^{\log_b^a})$ ise $T(n) = \Theta(n^{\log_b^a} \cdot \log n)$

3) $f(n) = \Omega(n^{\log_b^a + \epsilon})$ $\epsilon > 0$ $\left. \begin{array}{l} \text{ise } T(n) = \Theta(f(n)) \\ a \cdot f(n/b) \leq c \cdot f(n) \quad c > 1 \quad n > n_0 \end{array} \right\}$

Ör: $T(n) = 9 \cdot T(n/3) + n$ iken.

$$a=9, b=3, f(n)=n$$

$$n^{\log_3 9} = n^2 \text{ ve } f(n)=n \text{ i} \ddot{\text{e}} \text{kiyaslatalım.}$$

$n^2 > n$ oldugu için $T(n) = n^2$ olur. (1. durum)

Ör: $T(n) = T(2n/3) + 1$

$$a=1, b=3/2, f(n)=1$$

$$n^{\log_{3/2} 1} = 1 \text{ ve } 1 \text{ i kiyaslandik. } T(n) = \Theta(1 \cdot \log n) \text{ oldu.}$$

(2. durum)

Ör: $T(n) = 3 \cdot T(n/4) + n/\log n$

$$\textcircled{*} \quad a=3, b=4, f(n)=n/\log n$$

$$n^{\log_4 3} \text{ ve } n/\log n \text{ i kiyaslatalım}$$

$$n^{\log_4 3} < n/\log n$$

$$* \quad a \cdot f(n/b) \leq c \cdot f(n)$$

$$\frac{3}{4} \cdot n/\log n \leq c \cdot n/\log n$$

$$c \geq \frac{3}{4}$$

} $T(n) = n/\log n$ (3. durum)

Lecture 3 - Brute Force

3. Hafta

Brute force: problemin tüminden yola çıkarak çözmek.

$\Theta(n^n)$: brute force hesaplanması.

→ n tane sınıfın çarpımı: $\underbrace{n \times n \dots \times n}_n$

Sorting Problem:

→ n elemanlı listeyi sıralamak (küçükten büyüğe)

• Selection sort

→ Brute force bir yöntemdir.

→ En küçük element bul \Rightarrow base key, bir sonrakini bul.

$$\sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 = \Theta(n^2)$$

→ Extra yer sadece swap ile kullanılır ($\Theta(n)$)

→ En iyi algoritma olabilir. (Merge sort $\rightarrow \Theta(n \log n)$)

Selection Sort ($A[0 \dots n-1]$)

```
for i ← 0 to n-2 do
    min ← i
    for j = i+1 to n-1 do
        if A[j] < A[min]
            min ← j
    swap A[i] and A[min]
```

Search Problem:

→ n elemanlı bir listedeki bir item'i bulma

• Sequential Search

Sequential Search ($A[0, 1 \dots n], \text{key}$)

```
i ← 0
A[n] ← key
while A[i] ≠ key
    i ← i + 1
```

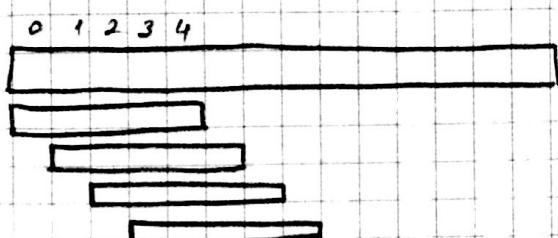
```
if i < n
    return i
return -1
```

$$T(n) = \Theta(n)$$

String Matching

Bir string text içinde, baska bir string'i aramak.

Brute force çözüm: Bütün kelimeyi text'te arat yoksa kaydirmak.



Algoritma:

' input1: $T[0, 1, \dots, n-1]$ \rightarrow n karakterli text
' input2: $P[0, 1, \dots, m-1]$ \rightarrow m karakterli key word.
' output: bulunduysa ilk harfin index i yoksa -1

```
for i  $\leftarrow$  0 to n-m do
    j  $\leftarrow$  0
    while j  $<$  m and  $P[j] = T[i+j]$  do
        j  $\leftarrow$  j+1
```

```
if j = m
    return i
return -1
```

$$\sum_{i=0}^{n-m} \sum_{j=0}^m = nm - m^2 \Rightarrow \Theta(nm)$$

+ Case: Aranın kelime yoksa ve aranılan kelimenin ilk elementi text'te yoksa. $\Theta(n)$

$$-(n) = \Omega(n) = O(nm)$$

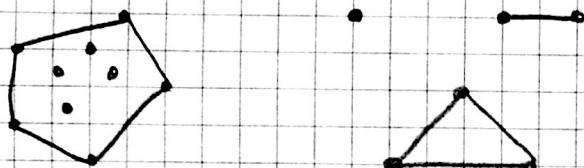
Closest-Pair Problem

→ 2. boyutlu bir problemden (x, y) koordinatları gerekli.

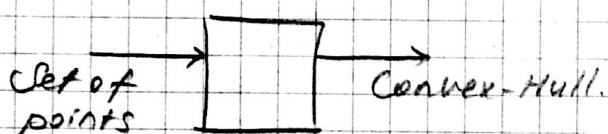
→ Brute force yöntemi bütün noktaların dercesmesi ve min'un bulılmasıdır.

Convex-Hull Problem

→ Verilen noktaların hepsini içine alıp en küçük convex yapmayı, bulmayı.



→ Convex hull'in köşeleri, verilen noktalardan oluşmalı.

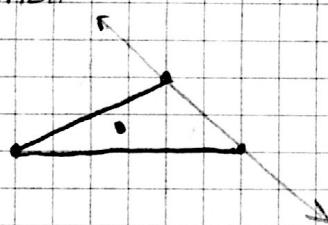


- $(x_i, y_i) \quad (1 \leq i \leq n)$
- input size : n
- output : convex hull'in kırılsız noktalardan oluşan array

Cüzdüm

→ Bütün grafikleri oluşturur ve convex-hull'yi seç.

→ Bir tane, ele alıfamızda, bütün noktalar aynı tarafa kalmalı.



- for $i = 1$ to n

 for $j = i+1$ to n
 calculate the line segment $(\overline{P_i P_j})$

 for $k = 1$ to n

 check whether P_k is on the same side

 if all on the same side

$(\overline{P_i P_j})$ is a side of convex-hull

$$T(n) = n^3$$

- En küçük sefer yarınla n^2 de yapılabilir.

• Travelling Salesman Problem

→ Büttün noktaları gezip siny noktaya ulaşabilen, en kısa turu bulmak.

→ Brute force çözüm: büttün olasılıklar deneyip önce Hamilton mu diye bater, sonra en kısa turu seçer. (Exhaustive Search) (az meseleli)

$$T(n) = \underbrace{(n-1)!}_{\text{hamilton}} \cdot \underbrace{n}_{\text{en kısa cycle sayısı}} = n! \rightarrow n \text{ tüküt iken sona} \text{ dep.}$$

turnu bulma

→ Exhaustive Search : Brute force yöntemlerde biridir.

• Knapsack Problem

→ Ağırlıklı n tane elemenin, kapasitesi belli olan bir depoya en verimli şekilde yerlestirmek.

$$x_i = \begin{cases} 0, 1 \end{cases}$$

disinda \leftarrow iceride

$$\sum_{i=1}^n x_i w_i \leq w \rightarrow \text{depo eq.liği}$$

max olani aliyoruz.

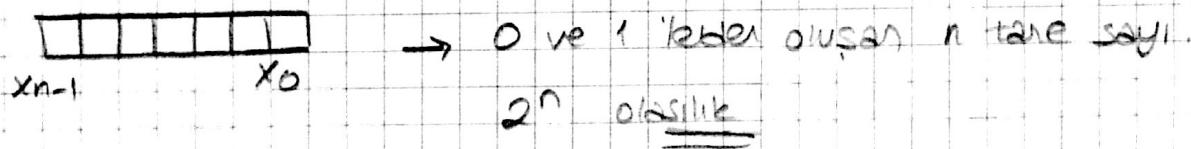
→ Exhaustive search kullanılabilir.

```

for each subset
    compute total weight
    if total weight  $\leq w$   $\rightarrow 2^n \cdot n$ 
        compute total value
        if total value  $>$  max  $\rightarrow 2^n$ 
            update max
    
```

$$O(2^n \cdot n)$$

→ Subset'leri bulmak için



→ Salesman ve Knapsak Problemleri NP hard problemdir. Polinomik çözümelli bulunamamıştır.

- Assignment Problem

→ n tane kişi için n tane işi en verimli şekilde dağıtmak.

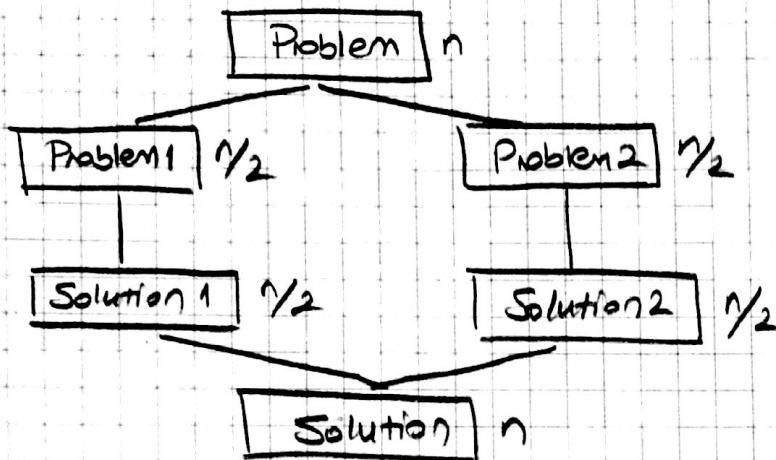
Orij:

| | Job1 | Job2 | Job3 | |
|----------|------|------|------|--------------------------|
| person 1 | 2 | 4 | 3 | → En iyi seçenek 2, 1, 3 |
| person 2 | 1 | 5 | 3 | |
| person 3 | 4 | 6 | 2 | |

→ Exhaustive Search ile bulabiliriz.

Lecture 4 - Divide and Conquer

- Problem, ana probleme aynı özellikteki parçalarla ayrılır.
küçük parçalar çözülür, birleştirilir.



Genel Algoritma:

DivideConquer (P)

if (P) is small then return non DivideConquer (P)

else

divide P into $P_1, P_2, P_3 \dots P_k$ $k \geq 1$

apply DivideConquer(P_i)

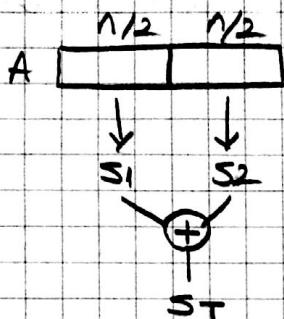
} return combine (DivideConquer(P_1) ... DivideConquer(P_k))

Genel Analiz:

$$\left\{ \begin{array}{l} T(n) = \sum_{i=1}^k T(n_i) + f(n) \\ T(n) = T'(n) \quad (\text{base case}) \end{array} \right. \xrightarrow{\text{divide and combine kisiminden gelir.}}$$

$$\Rightarrow T(n) = k \cdot T(n/b) + f(n)$$

Ör: n tane sayınin toplamı



Brute force: $\Theta(n)$

$\text{Sum}(A[1 \dots n])$

if($n == 1$) return $A[0]$ → Base case

$$m = n/2$$

$$S_1 = \text{Sum}(A[1 \dots m])$$

$$S_2 = \text{Sum}(A[m+1 \dots n])$$

→ Divide

} Apply

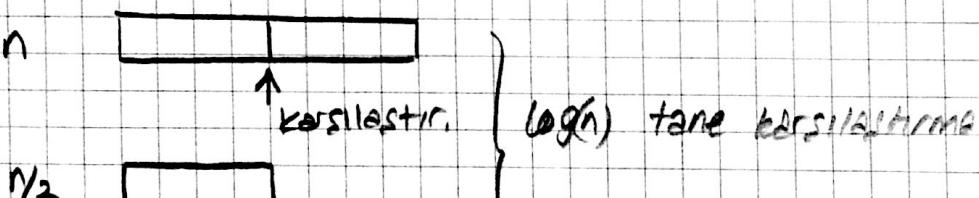
return $S_1 + S_2$ → Combine

$$\boxed{T(n) = 2 \cdot T(n/2) + 1}$$

$$\boxed{T(1) = 0}$$
 → $\Theta(n)$

Ör: Binary Search

→ Biranın key'i bulmak için sıralı array'in ortasındaki element key ile karşılaştırılır. Duruma göre array'in yarısının elemanı olur.



$\log(n)$ tane karşılaştırma

$$T_b(n) = \Theta(1)$$

$$T_w(n) = \log(n) + 1$$

$$T_{av}(n) = \Theta(\log n)$$

Maximum Subsequence Sum Problem:

→ Bir array içinde en büyük toplama sırası sıralı hâlde aranır.

Ör.

| | | | | |
|----|----|---|----|----|
| -4 | 10 | 3 | -1 | 12 |
|----|----|---|----|----|

Toplamı: 24 (max)



$$\rightarrow \max\left(\sum_{k=i}^j a_k\right) \quad 1 \leq i \leq j \leq n$$

- n elementli arrayde $\frac{n(n+1)}{2}$ subsequence olur.

Brute force çözüm:

```

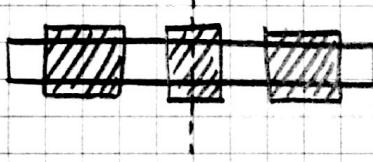
max = a[1]
for i=1 to n
    for j=i to n
        sum=0
        for k=i to j
            sum = sum + a[k]
        if sum > max
            max = sum
    
```

$$\left[\sum_{i=1}^n \sum_{j=i}^n \sum_{k=i}^j 1 \right] = \Theta(n^3)$$

toplama
islemi sayisi

Divide and Conquer ile

Problemi ikisiye bölelim, orta kesim için bir kez daha baktırın.



→ Üç olumsuz için bir kez daha vanagalm
en büyükini seçelim.

$MSS(a[1 \dots n])$

if $n==1$ return $a[1]$
else

$m=n/2$

$S_1 = MSS(a[1 \dots m])$

$S_2 = MSS(a[m+1 \dots n])$

$max = a[1]$

for $i=1$ to $n/2$

 for $j=\lceil \frac{n}{2} \rceil + 1$ to n

 sum=0

 for $k=i$ to j

 sum = sum + a[k]

 if sum > max

 max = sum

return max(S_1, S_2, max)

} Apply two part

Orta kesim için baktır.

} Combine

$$T(n) = 2 \cdot T(n/2) + \Theta(n^3)$$

$$T(1) = 0$$

} Master Teoreme gone
 $T(n) = \Theta(n^3)$

$\hookrightarrow n^3$ $f(n)$ eisiminden geldi. Bu kismi igne etmeliyim.

Düzelme 1:

... for $i=1$ to $n/2$

$$\text{sum} = 0$$

for $k=1$ to $n/2$

$$\text{sum} = \text{sum} + a[k]$$

for $j=n/2+1$ to n

$$\text{sum} = \text{sum} + a[j]$$

if $\text{sum} > \text{max}$
 $\text{max} = \text{sum}$

...

}

$$f(n) = \Theta(n^2)$$

$$T(n) = 2 \cdot T(n/2) + \Theta(n^2)$$

$$\boxed{T(n) = \Theta(n^2)}$$

Düzelme 2:

$$S_1 = a[n/2]$$

$$m_1 = a[n/2]$$

for $i=n/2-1$ to 1

$$S_1 = S_1 + a[i]$$

if $S_1 > m_1$

$$m_1 = S_1$$

$$S_2 = a[n/2+1]$$

$$m_2 = a[n/2+1]$$

for $j=n/2+2$ to n

$$S_2 = S_2 + a[j]$$

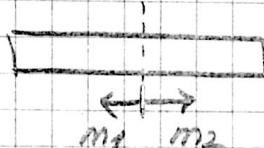
if $S_2 > m_2$

$$m_2 = S_2$$

$$\text{max} = m_1 + m_2$$

...

}



$$f(n) = \Theta(n)$$

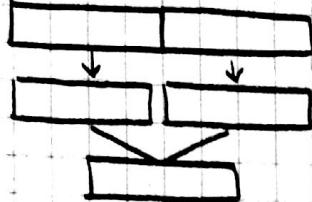
$$T(n) = 2 \cdot T(n/2) + \Theta(n)$$

$$\boxed{T(n) = \Theta(n \log n)}$$

Sorting

Sorting problemini Divide & Conquer ile çözen iki algoritma bulunmaktadır. Merge Sort, Quick Sort

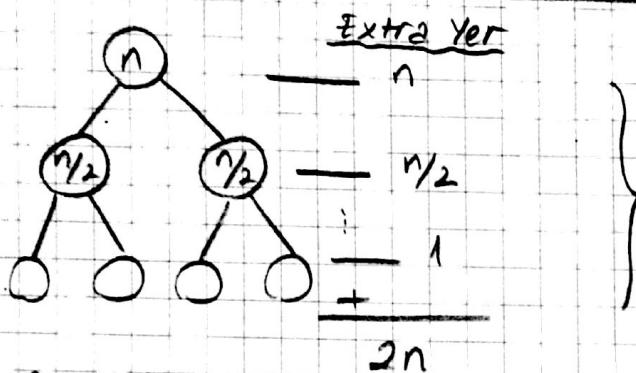
Merge Sort:



Sırasız liste ornek iceriye bölünür, sıralanır ve merge (birleştirilme) yapılır.

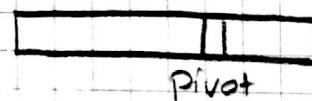
$$T(n) = 2 \cdot T(n/2) + \Theta(n)$$

$$\boxed{T(n) = \Theta(n \log n)}$$



Aynı anda $2n$ büyüklüğünde extra yere ihtiyaçları var.

Quick Sort:



Pivot seg, pivotin sağ büyüklerini bir tarafa kuantileri diğer tarafta topla, 2 parçayı sırala

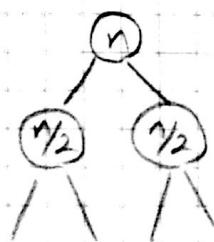
$$T_b(n) = 2 \cdot T(n/2) + \Theta(n)$$

$$T_b(n) = \Theta(n \log n)$$

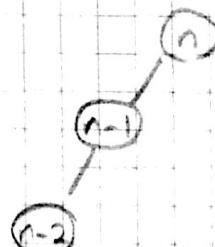
$$T_w(n) = T_w(n-1) + \Theta(n)$$

? Sıralı ve pivot ilk elemanı seçerse

$$T_w(n) = \Theta(n^2)$$



Best Case de extra $\log n$ yer kullanır.



Worst Case de extra n yer kullanır.

Static yer tamlanırsa 1 tanrı extra yer yeterli..

Quick sort için average case de zaman karmaşıklığı

$$T_{av}(n) = \sum_{|I|=n} P(I) \times T(I)$$

olasılık zaman

Pivotun durumuna göre

pivot en küçük

$$T(0) + T(n-1) + n+1$$

:

$$T(1) + T(n-2) + n+1$$

pivot en büyük

$$T(n-1) + T(0) + n+1$$

+

$$\sum_{k=0}^{n-1} 2 \cdot T(k) + (n+1)$$

$$P = 1/n$$

$$T_{av}(n) = \frac{1}{n} \cdot \left(\sum_{k=0}^{n-1} 2 \cdot T(k) + n(n+1) \right)$$

$$n \cdot T_{av}(n) = \sum_{k=0}^{n-1} 2 \cdot T(k) + n(n+1)$$

$$(n-1) \cdot T(n-1) = \sum_{k=0}^{n-2} 2 \cdot T(k) + n(n-1)$$

$$n \cdot T(n) - (n-1) \cdot T(n-1) = 2 \cdot T(n-1) + 2n$$

$$n \cdot T(n) = (n+1) T(n-1) + 2n$$

$$\frac{T(n)}{n+1} = \frac{T(n-1)}{n} + \frac{2}{n+1}$$

$$S(n) = S(n-1) + \frac{2}{n+1} \quad \rightarrow \quad S(n) = \Theta(\log(n))$$

$$\boxed{T_{av}(n) = \Theta(n \log n)}$$

Not: MergeSort optimal olmasının rağmen, extra yer kullanırı
fazla oldyuandan Quick Sort daha iyi

Multiplication of Large Integers

30.10.2012

$$\begin{array}{r} 7212 \\ \times 3518 \\ \hline \end{array} \quad \begin{array}{c} \rightarrow a \\ \rightarrow b \end{array} \quad \begin{array}{r} 72 \quad 12 \\ \hline a_1 \quad a_0 \end{array} \times \begin{array}{r} 35 \quad 18 \\ \hline b_1 \quad b_0 \end{array}$$

$$a \times b = \underbrace{(72 \times 35) \cdot 10^4}_{\text{Yeniden böl, çarp}} + (72 \cdot 18 + 35 \cdot 12) \cdot 10^2 + 12 \cdot 18$$

Yeniden böl, çarp

- input size: n basamaklı sayı
- temel operasyon: çarpma

$$\begin{array}{l} T(n) = 4 \cdot T(n/2) \\ T(1) = 1 \end{array} \quad \left. \begin{array}{l} \text{Master Teorem'e göre} \\ T(n) = \Theta(n^2) \end{array} \right.$$

• n^2 lik zaman kaybı 4'e bölmekten kaynaklıyor. Düşürmeyi deneyelim.

Düzelme:

$$c = a * b = c_2 \cdot 10^2 + c_1 \cdot 10 + c_0$$

$$c_2 = a_1 \cdot b_1$$

$$c_0 = a_0 \cdot b_0$$

$$c_1 = (a_1 + a_0) * (b_1 + b_0) - (c_2 + c_0)$$

$$T(n) = \Theta(n \log_2^3)$$

→ Önceden hesaplandığı için çarpma sayısını 3'e düşür.

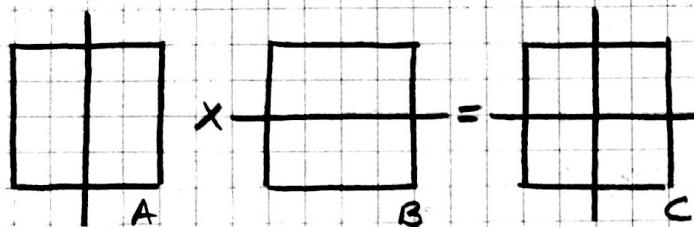
Brute Force : $\Theta(n^2)$

Divide & Conquer : $\Theta(n \log^3 n)$

$n > 600$ ise divide & conquer verimli olabilir.

Matrix Multiplication

Brute force ile $\Theta(n^3)$ de yapılır.



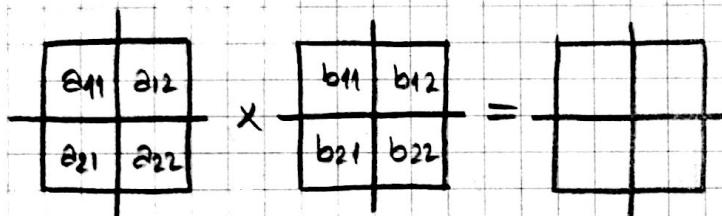
- input size : n boyutundan element sayısı
- temel operasyon : çarpma

$$M(n) = 4 \cdot M(n/2) \quad \left. \begin{array}{l} \\ \end{array} \right\} \text{Master Teoremi uygun değil}$$

$$M(1) = 1$$

$$\text{Gözünce } M(n) = \Theta(n^3)$$

Degisik bir algoritma :



- input size : n matrisin boyutu (n²)

$$M(n) = 8 \cdot M(n/2) \quad \left. \begin{array}{l} \\ \end{array} \right\} M(n) = \Theta(n^3)$$

$$M(1) = 1$$

Strassen'in Algoritması

- temel operasyon : çarpma

$$M(n) = 7 \cdot M(n/2) \quad \left. \begin{array}{l} \\ \end{array} \right\} \Theta(n^{\log_2 7})$$

$$M(1) = 1$$

- temel operasyon : toplama

$$A(n) = 7 \cdot A(n/2) + 18 n^2 / 4 \quad \left. \begin{array}{l} \\ \end{array} \right\} \Theta(n^{\log_2 7})$$

$$A(1) = 0$$

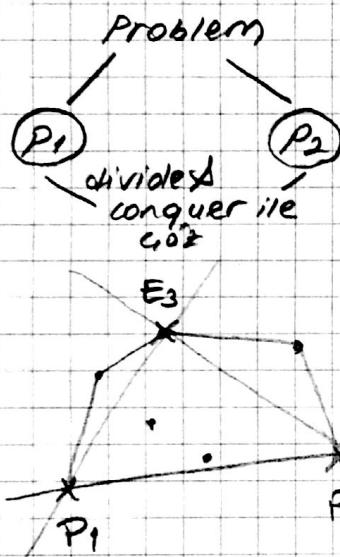
31.10.2012

Convex-Hull Problem

- Brute force'da n^3 de yapmistro.
- Divide & conquer ile Quick-Hull algoritması kullanılır.

Quick-Hull Algoritması

- Extrem noktalar bulunur. Bu noktaların teknikle convex-hull'da olduğu belidir. Ör: x' en büyük, en euklîdik vs.
- Extrem noktalar birleştirilir. Üstinde kalan noktalar için ve altinda kalan noktalar için divide & conquer yap.
- Problemin ilk bölünmesi divide & conquer değildir.



Yeni extreme noktasını ilk line'a ızaklarından buluruz. En uzak olan extreme noktasıdır. Bu noktasıyla diğer extremleri bulalım. $T(k) = \Theta(k)$

Daha sonra kalan noktalar冉ın yanlarında divide & conquer yap.

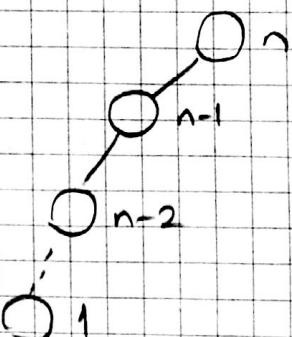
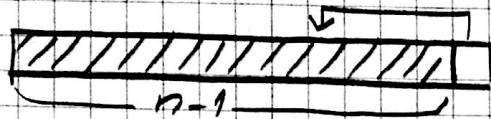
- Divide $\rightarrow \Theta(n)$
- Conquer
- Combine $\rightarrow \Theta(1)$

$$\left. \begin{array}{l} T_w = \Theta(n^2) \\ T_b = \Theta(n \log n) \\ T_{av} = \Theta(n \log n) \end{array} \right\} \begin{array}{l} \text{Noktalar Gember} \\ \text{Üzerinde ise} \end{array}$$

Lecture 6 : Decrease and Conquer

06.11.2012

- Problemi küçült, aynı problemin küçüğünü elde et, çöz, birleştir.



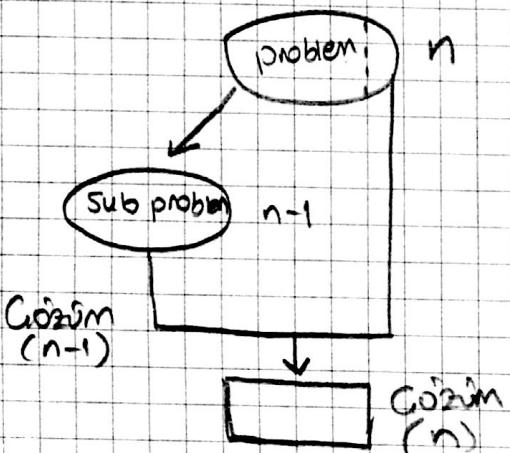
- iterative
- recursive } yararlı olabilir.

recursive : top down
iterative : bottom up

3 depository ways are available:

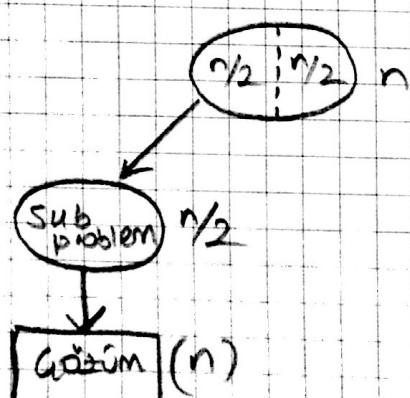
- 1 - Sabit bir depreme göre azaltır.
- 2 - Sabit bir çarpma veya bölme azaltır.
- 3 - Değerler bir depreme göre azaltır.

1 - Sabit bir depreme göre azaltır



```
int factorial(int a)
{
    if (a == 1) return 1
    return a * factorial(a-1);
}
```

2 -



- Genelde sayıları %6 tür kismı 02, içinde genetik almaz.
- Or: Binary Search.

```
int pow (a, n)
{
    if (n == 0) return 1
    x = pow(a, n/2);
    if n is even
        return x*x
    return a*x*x
}
```

Tarifet bulma algoritması, 16/17:

$$\begin{aligned} M(n) &= M(n/2) + 2 \\ M(0) &= 0 \end{aligned} \quad \left\{ \begin{array}{l} M(n)_w = 2 \lceil \log n \rceil \\ M(n)_b = \lceil \log n \rceil + 1 \end{array} \right\} \Theta(\log n)$$

Worst Case durumu: n in $2^k - 1$ olması.
Best Case durumu: n in 2^k olması.

3- Dairelerdeki en büyük ortak bölen bulma:

GCD: Greatest Common Divisor

$$\begin{aligned} \gcd(m, n) &= \gcd(n, m \% n) \\ \gcd(m, 0) &= m \end{aligned} \quad \left\{ \begin{array}{l} \text{logaritmik zorluk} \\ \text{çaplı formül?} \end{array} \right.$$

Insertion Sort

$$\begin{aligned} T_w(n) &= \Theta(n^2) \\ T_B(n) &= \Theta(n) \end{aligned} \quad \left\{ \begin{array}{l} \text{Sıralanmış listeeye eleman ekleme} \\ \text{bir bir} \end{array} \right.$$

Depth-First Search (DFS)

- Grafların
 - Başlangıç kontrol etmek için
 - Cycle içeriği içermemesi beklemek için
- Bir noktadan başlayıp gidilebilecek noktaya kadar git, bir tane de geri dönüp diğer noktaları done.
- Bunun için stack kullanılır.
- Decrease by one teknikinden
- Adjacency matrix için $\Theta(V^2)$ de
- Adjacency list için $\Theta(|V|+|E|)$ de çalışır.

Breadth First Search (BFS)

• Bir noktasın başlayıp bir adımda gidilebillererek nökteleri bulur, daha sonra 2 adımda gidilebilecek olanları bulur...

• Queue kullanır.

• Bir noktasın diğer noktaya en kısa sürede giden yolu verir.

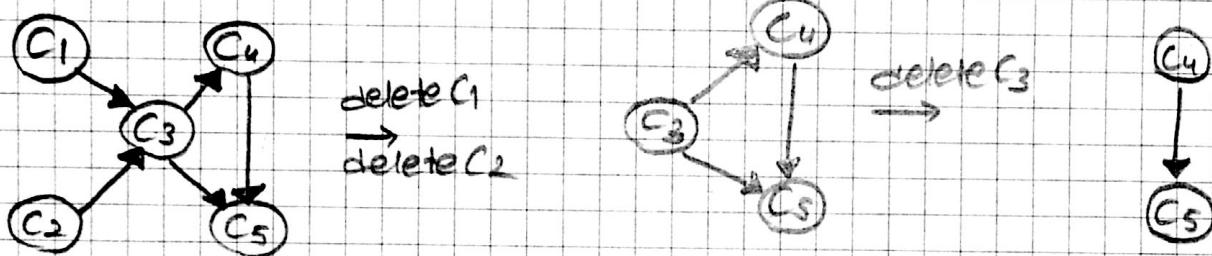
• Karmasılığında DFS gibi dir.

Topological Sorting

• Bağı ve cycle içermeyen graflerde uygulanır.

• Ör: Dörtlü derslerin ön şartları çizelgesi

• Bir node'un öncesi geçmesi bir node'nin yerine listede ilk olur.



del C4 → del C5 Silinme sırası:

C1, C2, C3, C4, C5

→ eğer bunlar ders sırası ise C1'yi almadan C3'ü alamazsan denemek oluyor.

Karmaşıklığı bili?

Decrease by a Constant-Factor Algorithms

Fake-Coin Problem

- Elimizde bozuk paralar olsun. Bu bozuk paralar aynı eşitliklere sahip, fakat bir taneyi alsa hafif - Hafif bozuk parayı bulmalıyz.

→ Bozuk paraları 2'ye ayır. Aradığımız, hafif olmada olmalı.

- Çift sayıda bozuk para varsa ikiye bölg.
- Tek sayıda bozuk para varsa



→ 2 büyük parça eşitse aradığımız tek olmalıdır.

$$T_w(n) = \begin{cases} 1 + T_w(n/2) & \text{3 her seferinde 4'ye bölük.} \\ 0 & \Theta(\log n) \end{cases}$$

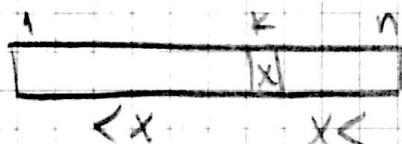
→ 3'e böleydim $T_w = \Theta(\log_3 n)$ olur.

(dikkat yeri)
Asimptotik olarak $\log_2 n = \log_3 n$ oluyamus

Variable Size Decrease Algorithms

Computing Median

Sırasız bir arrayde, arrayin orta elemanını bulma.



$k = n/2$ ise X mediandır.

$k > n/2$ ise $A[1 \dots k]$ arasında

$k < n/2$ ise $A[k+1 \dots n]$ arasında

Selection Problem

n elementli listede $k.$ en küçük element bulma.

- $k = n/2$ ise aralık median oluyor.

ör: $\underline{4}, 1, \underline{10}, 9, 7, 12, 8, 2, 15 \quad k=5$ olsun,
pivot

$\underline{4}, 1, \underline{10}, 9, 7, 12, 8, \underline{2}, 15$

$\underline{4}, 1, 2, \underline{9}, 7, 12, 8, 10, 15$

$1, 2, \underline{4}, \underline{9}, 7, 12, 8, 10, 15$

3. eleman \leftarrow burada bu arrayin 2. eleman olmalı.

pivot $\underline{9}, 7, \underline{12}, \underline{8}, 10, 15$

$\underline{9}, 7, \underline{8}, \underline{12}, 10, 15$

$7, 8, \underline{9}, 12, 10, 15$

\underline{k} burada 3. eleman
2. eleman.

$\underline{k=8}$ dir.

Worst Case

$$T_w(n) = T_w(n-1) + n+1 \quad \text{if } T_w(1) = 0 \quad T_w(n) = \Theta(n^2)$$

Best Case

$$T_b(n) = \Theta(n)$$

Çoğu durumda bilindiyse

$$T(n) = T(n/2) + \Theta(n)$$

$$T(n) = \Theta(n)$$

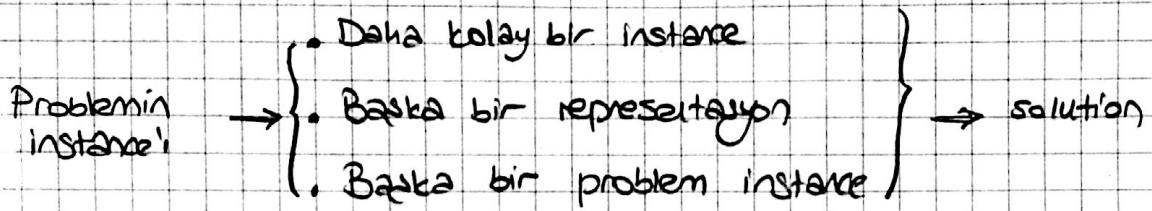
Average Case $T_a(n) = \Theta(n)$

17-11-2012 Algoritma Vizesi

- 1-
 - a- Algoritma nedir? Karakteristiği nedir? (10)
 - b- Bir file'in elementlerini sıralarken hangi sort alg. kullanılmalı? (5)
 - c- Binary Search tree'ye eklenebilen etelerken decrease-by-variable size kullanılabılır mı? Aşikle- (5)
 - d- $f(n) = O(n)$ ve $f(n) = O(n^2)$ olduğunu ispatlayınız. (5)
- 2- Convex-hull problemini divide and conquer ile çaren algoritmasını açıklayınız 12 elementli bir graf rəm göstəriniz. (15)
- 3- Prefix sum probleminin karakteristiği söyledir:
 $A[1 \dots n] \rightarrow$ input array'i , $S[1 \dots n] \rightarrow$ return array'i.
 $S[1] = A[1]$
 $S[2] = A[1] + A[2]$
 $S[n] = A[1] + A[2] \dots A[n]$
- 4-
 - a- Brute force algoritmasını yazınız, analizi ediniz.
 - b- Decrease and conquer "
 - c- Divide and conquer "
 - d- B4 algoritmaların performanslarını karşılaştırınız. (60)

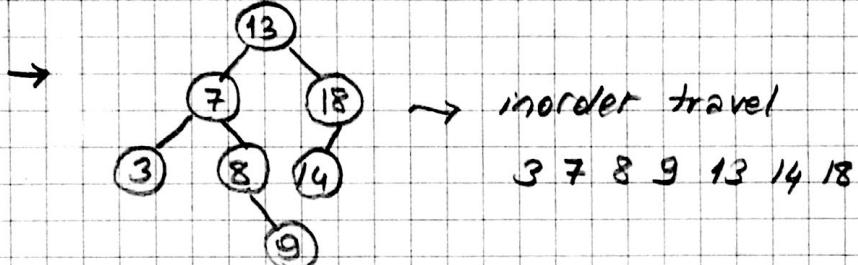
Lecture 7 - Transform and Conquer

- Problemi dönüştür ve çöz, sonra eski haline getir.



"ör: n elemanlı arrayi sıralamak için binary search tree'ye çevirip çözüset daha kolay çözülür.

0 1 2 3 4 5 6
13 7 18 8 9 3 14



PRE SORTING

- Sort edildiğinde kolay解决 problemler için kullanılır.
dr: searching problem

Searching Problem

unsorted array key

- Searching Prob.
- Linear Search
- $O(n)$

Sort
MergeSort
 $\Theta(n \log n)$

sorted array key

- Searching Problem
- Binary Search
- $O(\log n)$

| | 1 element için | n element için |
|----------------|--------------------|---------------------|
| Linear Search | $O(n)$ | $O(n^2)$ |
| PreSort Search | $\Theta(n \log n)$ | $\Theta(n \log n)$ |

GAUSSIAN ELIMINATION

$$Ax = b$$

$$\begin{bmatrix} a_{11} & \dots & a_{1n} \\ \vdots & & \vdots \\ a_{n1} & \dots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix}$$

$$\begin{bmatrix} 0 & \dots & 0 \\ 0 & \dots & 0 \\ 0 & \dots & 0 \\ 0 & \dots & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix}$$

- Gauss Elimination yapılımına

$$\sum_{k=1}^n n-k = \Theta(n^2)$$
 de yapılır.

- Matris, Gauss Elimination'a $\Theta(n^2)$ de girir.

Representation Change

Balanced Search Tree

- dengeli ağaclar.

• AVL

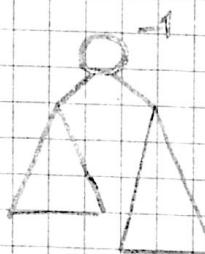
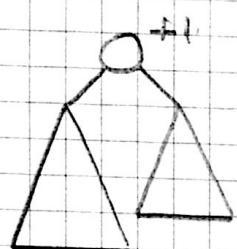
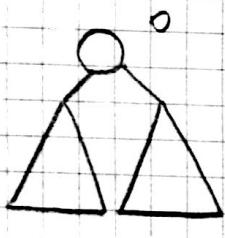
• 2-3-4 Tree

• Red-Black Tree

...

- Binary search tree'de her zaman $O(\log n)$ de işlem yapılımeye lütfen.
- Her zaman dengeli olmak gereklidir.

AVL Tree'ye dönüştürme

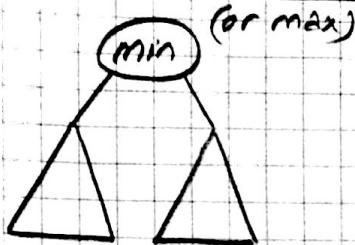


$$\begin{aligned} \text{Eleman eklemeye: } & \log 1 + \log 2 + \log 3 + \dots + \log n = \log n! = \log \frac{n^n}{e^n} \\ & = n \log n - c \cdot n = \Theta(n \log n) \end{aligned}$$

Bir diziyi AVL tree'ye dönüştürme costs.

Array $\xrightarrow{\Theta(n \log n)}$ AVL Tree $\xrightarrow{\Theta(n)}$ Sorted Sequence

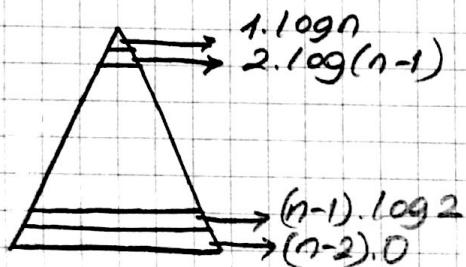
Heap'e dönüştürme :



- Eleman silmeyi en iyi ten yapar. (yani min elemanı siler.) Yerine kalanların en küçüğü gelir.
- Eleman eklemeyi en zayıf yapar, en alto gelen eleman yoksayıya yerler.

| | Unsorted Array | Heap |
|--------|----------------|------------------|
| find | $\Theta(n)$ | $\Theta(1)$ |
| delete | $\Theta(n)$ | $\Theta(\log n)$ |
| add | $\Theta(1)$ | $\Theta(\log n)$ |

- Array'in bütün elemanlarını heap'e eklemesi $\Theta(n \log n)$
- BuildHeap islemiyle $\Theta(n)$ de yapılın.



Yaklaşık çözüm:

$$A \leq 1 \cdot \log n + 2 \cdot \log n + 3 \cdot \log n + \dots + \frac{n}{2} \cdot \log n$$

$$A \leq \log n (1 + 2 + 3 + \dots + \frac{n}{2})$$

$$A \leq (\log n) \cdot \frac{n}{2} \Rightarrow A = \Theta(n \log n)$$

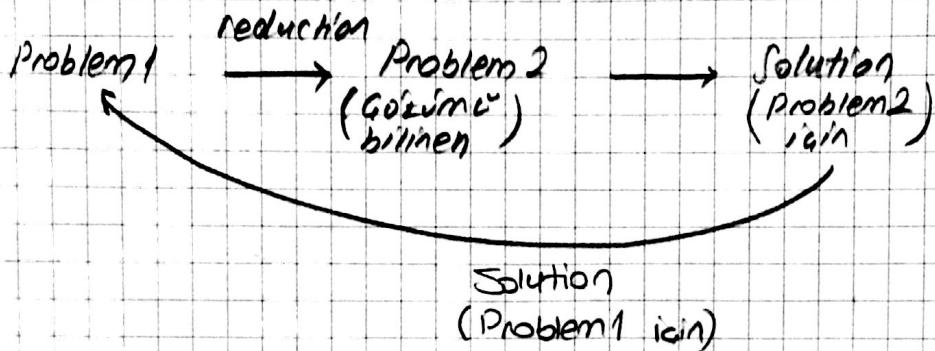
$$A = \sum_{i=2}^{\log n} \frac{n}{2^i} \cdot i = \Theta(n)$$

$$\text{Array} \xrightarrow{\Theta(n)} \text{Heap} \xrightarrow{\Theta(n \log n)} \text{Sorted Sequence}$$

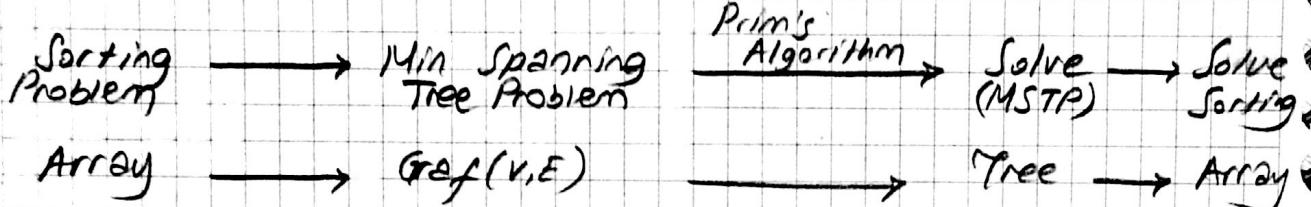
Heapsort optimal bir algoritmadır. $\Theta(n \log n)$

Problem Reduction

Bir problemi çözmenin bir başka bir problemi kullanmak

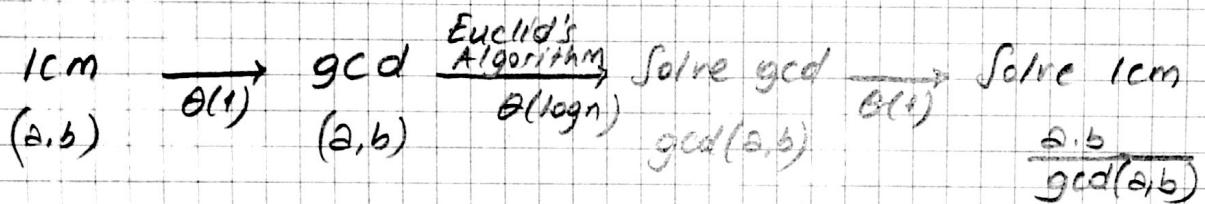


Or: Sorting Problem



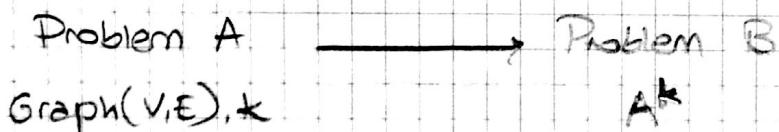
Least Common Multiple ekok

$$\text{lcm}(a,b) = \frac{a \cdot b}{\text{gcd}(a,b)}$$



$$T(n) = \theta(\log n)$$

Counting Paths in Graph



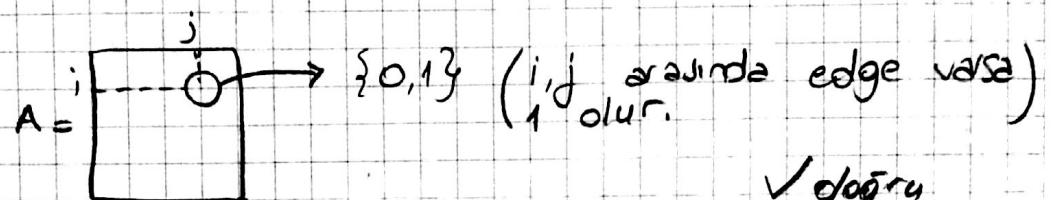
k: path uzanlığı
A: orjinal matris
formu

$U-V$ arasında k uzunluğunda kaç tane yol olduğunu bek.

$\underbrace{A^k}_{\substack{k \text{ tane} \\ A \cdot A \dots A}}[i,j] : U,V$ arasındaki path sayısı

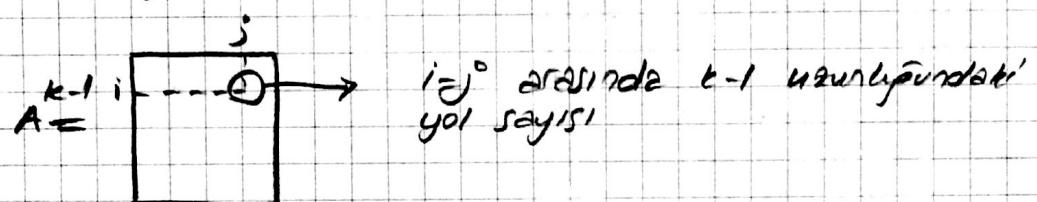
İşteleyelim:

$k=1$ için



✓ doğru

$k-1$ için doğru olduğunu düşünelim.



k için $A^{k-1} \cdot A = A^k$

$$\boxed{\begin{matrix} i \\ t \end{matrix}} \cdot \boxed{\begin{matrix} t \\ j \end{matrix}} = \boxed{\begin{matrix} i \\ j \end{matrix}}$$

$$\sum_{t=1}^n A^{k-1}[i,t] \cdot A[t,j] = A^k[i,j] \quad \checkmark \text{ doğru}$$

$$A^k = \underbrace{A \cdot A \dots A}_{k \text{ tane}}$$

divide and conquer ile

$$\Theta(\log k \cdot n^{\frac{3}{2}})$$

Reduction of Optimization Problems

- Bir fonksiyonun max (veya min) değerini arar.
- Gözüm: Fonksiyonun negatifinde min aranır ve bu arama gözümü buluruz.

Extrem point bulma: $f(x)$ için $f'(x)=0$

Linear Programming:

- Problemleri lineer hale getiremek gereklidir.

O'r: 100 milyonluk yatırım yapılacaktır.

| | Ker |
|-----------------------|-----|
| $x \rightarrow$ stock | % |
| $y \rightarrow$ bond | 10 |
| $z \rightarrow$ cash | 7 |
| | 3 |

hangisine yatırılsa en yüksek sonucu bulsun?

$$x + y + z = 100$$

$(0,1)x + (0,07)y + (0,03)z \Rightarrow$ max'una erdiğimizde olur.
 → Lineer hale getirildi.

★ Lineer problemlerin klasik çözümü simplex algoritması
 vs. Eksponentiyel zamanında gelir. (mazda casado)

Knapsak Problem

Knapsak problemini lineer programming problemine çevirelim.

$$P_1 \cdot x_1 + P_2 \cdot x_2 + \dots + P_n \cdot x_n \rightarrow \text{objective function}$$

$$0 \leq x_i \leq 1 \quad (\text{Obje boşlansırsa } x=0, 1)$$

$$W_1 \cdot x_1 + W_2 \cdot x_2 + \dots + W_n \cdot x_n \leq W \rightarrow \text{toplam ağırlık}$$

Lecture 8: Dynamic Programming

Problemleri recursive olarak paralel veya ve çözmeye çalışır.
 Bir kez bulunan sonuc bir daha hesaplanmasa. Once ettiksi
 gözüp sonra üstte girer. (Bottom-up)
 Tablolama kullanılır.

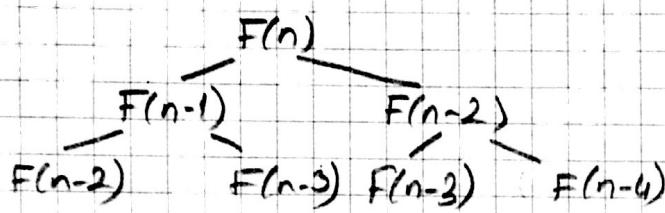
Dr. Fibonacci Numbers

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

$$\begin{aligned} F(0) &= 0 \\ F(1) &= 1 \end{aligned}$$

| | |
|----------|-------|
| $F(n)$ | 1 kez |
| $F(n-1)$ | 1 kez |
| $F(n-2)$ | 2 kez |
| $F(n-3)$ | 3 kez |

hesaplanır



- Normal hesaplama eksponansiyel zaman alır. $\Theta(2^n)$
- Dynamic programming'e göre

$$F(n), F(n-1), \dots, F(0)$$

} $n+1$ alt problem röldür.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 2 | 3 | | | |

$$T(n) = T(n-1) + T(n-2) + 1 \quad } \quad T(n) = \Theta(2^n) \quad \text{normalde D&C ile}$$

$$T(1) = T(0) = 0 \quad }$$

$$T(n) = \Theta(n) \quad } \quad \text{Dynamic Programming ile}$$

$$S(n) = \Theta(n) \quad }$$

$$\left\{ \begin{array}{l} A[0 \dots n] \\ \text{for } i=2 \text{ to } n \end{array} \right.$$

$$A[i] = A[i-1] + A[i-2]$$

$$\text{return } F(n)$$

}

Binomial Coefficients

$$\binom{n}{k} = \frac{n!}{(n-k)! \cdot k!}$$

(n, k integer values)

$$C(n, k) = C(n-1, k-1) + C(n-1, k)$$

$n > k > 0$

$$C(n, 0) = C(n, n) = 1$$

• Tabulation (Tabelle):

$$C(n, k) \quad , \quad C(i, j)$$

$0 \dots n \quad 0 \dots k$

| i\j | 0 | 1 | ... | k |
|-----|---|---|-----|---|
| 0 | 1 | | | |
| 1 | | 1 | 1 | |
| : | | | 1 | |
| n | | | | 1 |

Algorithm Binomial(n, k)

for $i=0$ to n

 for $j=0$ to $\min(i, k)$

 if $j=0$ or $j=i$
 $C[i, j] = 1$

 else
 $C[i, j] = C[i-1, j-1] + C[i-1, j]$

 return $C[n, k]$

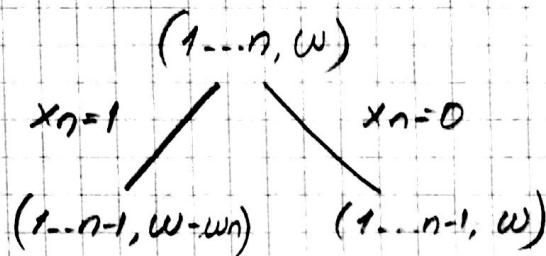
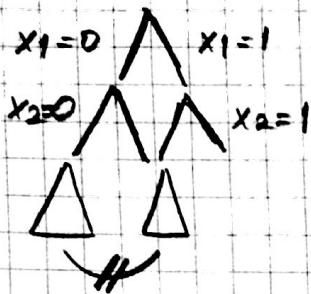
$$\sum_{i=0}^k \sum_{j=0}^{i-1} 1 + \sum_{i=k+1}^n \sum_{j=0}^k 1 = \sum_{i=0}^k (i-1) + \sum_{i=k+1}^n k$$

$$= \frac{k(k-1)}{2} + k(n-k) \in \Theta(nk)$$

Knapsack Problem

int weights : $w_1 w_2 w_3 w_4 \dots w_n$
 values : $v_1 v_2 v_3 v_4 \dots v_n$

$W \rightarrow \text{max weight}$
 $V = \{0, 1\}$



$$Knap(n, w) \xrightarrow{x_n} Knap(n-1, w-w_n, v_n) \quad (1)$$

$$Knap(n-1, w) \quad (2)$$

1'in optimal olması 2'nin optimal çözüm olmasına bağlıdır.

$$V(n, w) = \max \{ V(n-1, w-w_n) + v_n, V(n-1, w) \}$$

$$\frac{V(i, j)}{0 \dots n \quad 0 \dots w}$$

| i | 0 | 1 | 2 | j | w |
|---|---|----------------|----------------|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | | | | |
| 2 | 0 | ■ ¹ | ■ ² | | |
| 3 | 0 | | | | |
| n | 0 | | | | |

seçilen karelerden 1'inciinin degeri ile herdi
 degerini toplayın, 2 ile 2'si karşılaştırın
 büyük olan seçelim

Or: $w=5$ 4 tane item var

| item | weight | value |
|------|--------|-------|
| 1 | 2 | \$12 |
| 2 | 1 | \$10 |
| 3 | 3 | \$20 |
| 4 | 2 | \$15 |

| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|----|----|----|----|----|
| i | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 12 | 12 | 12 | 12 |
| 2 | 0 | 10 | 12 | 22 | 22 | 22 |
| 3 | 0 | 10 | 12 | 22 | 30 | 32 |
| 4 | 0 | 10 | 15 | 25 | 30 | 35 |

for $i=0$ to n) $\Theta(n)$
 $v(i, 0) = 0$

for $j=0$ to w) $\Theta(w)$
 $v(j, 0) = 0$

for $i=1$ to n
for $j=1$ to w

if $j - w_i < 0$

$v(i, j) = v(i-1, j)$

else

$v(i, j) = \max(v(i-1, j), v(i-1, j - w_i) + v_i)$

} $\Theta(n, w)$

$j=w$

for $i=n$ to 1

if $v[i, j] = v[i-1, j]$

$x_i = 0$

else

$x_i = 1$

$j = j - w_i$

} Optimal sequence 'i'
buildup

$\Theta(n)$

$T(n) = \Theta(nw)$

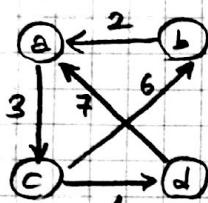
All-Pair Shortest Path Problem

- Bağlı ve ağırlıklı graf verilmiştir. Negatif 4zuntuk yoktur.
- Input $G(V, E)$

$$W = \begin{matrix} & & \\ & & \\ \begin{matrix} i \\ \downarrow \\ 1 \end{matrix} & & \end{matrix}$$

i ile j arasında edge varsa ∞ yazılırak.

- Output bir distance matrix (D)



graf

| | a | b | c | d |
|---|----------|----------|----------|---|
| a | 0 | 3 | ∞ | |
| b | 2 | 0 | ∞ | |
| c | ∞ | 7 | 0 | 1 |
| d | 6 | ∞ | 0 | 0 |

weigh matrix

Sequence of decisions

$x_1, x_2, x_3, \dots, x_n$: x_i : v_i vertexini shortest pathde kullanacak mıymış?

$\underbrace{i \dots k}_{\text{den}} \underbrace{j}_{\text{ye}}$

: eğer (i, j) arasındaki yol üzerinde k varsa (i, k) ve (k, j) yolları da optimaldır.

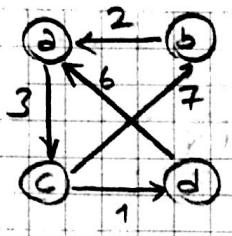
d_{ij} : i den j ye n tane vertexi kullanarak bulunan optimal yol.

$$d_{ij}^k = \min \left\{ (d_{ij}^{k-1}), (d_{ik}^{k-1} + d_{kj}^{k-1}) \right\}$$

$$W = D^0 \xrightarrow{\text{---}} D^1 \xrightarrow{\text{---}} D^2 \xrightarrow{\text{---}} \dots \xrightarrow{\text{---}} D^{n-1} \xrightarrow{\text{---}} D^n$$

i den j ye
giderken
başka vertex
yok

i den j ye
gitmek için
 v_i vertexini kullanımdan
min' u seçtim.



$$D^0 = \begin{bmatrix} a & b & c & d \\ a & 0 & 0 & 3 & \infty \\ b & 2 & 0 & \infty & \infty \\ c & \infty & 7 & 0 & 1 \\ d & 6 & \infty & \infty & 0 \end{bmatrix}$$

$$D^1 = \begin{bmatrix} a & b & c & d \\ a & 0 & \infty & 3 & \infty \\ b & 2 & 0 & 5 & \infty \\ c & \infty & 7 & 0 & 1 \\ d & 6 & \infty & 9 & 0 \end{bmatrix}$$

$$D^2 = \begin{bmatrix} a & b & c & d \\ a & 0 & 0 & 3 & \infty \\ b & 2 & 0 & 5 & \infty \\ c & 9 & 7 & 0 & 1 \\ d & 6 & \infty & 9 & 0 \end{bmatrix}$$

$b \rightarrow a \rightarrow c$

$c \rightarrow b \rightarrow a$

$$D^3 = \begin{bmatrix} a & b & c & d \\ a & 0 & 10 & 3 & 4 \\ b & 2 & 0 & 5 & 6 \\ c & 9 & 7 & 0 & 1 \\ d & 6 & 16 & 9 & 0 \end{bmatrix}$$

$$D^4 = \begin{bmatrix} 0 & 10 & 3 & 4 \\ 2 & 0 & 5 & 6 \\ 7 & 7 & 0 & 1 \\ 6 & 16 & 9 & 0 \end{bmatrix}$$

$a \rightarrow c \rightarrow b$
 $b \rightarrow a \rightarrow c \rightarrow d$

$c \rightarrow d \rightarrow a$

$D^0 \rightarrow$ vertex kullanmadan gitme

$D^1 \rightarrow a'yi kullanarak git$

$D^2 \rightarrow D^1 tablosunun üzerinde
b'yi kullanarak git
(a ve b'yi kullanarak ulas)$

for $k=1$ to n
for $i=1$ to n
for $j=1$ to n

$$D[i,j] = \min \{ D[i,j], D[i,k] + D[k,j] \}$$

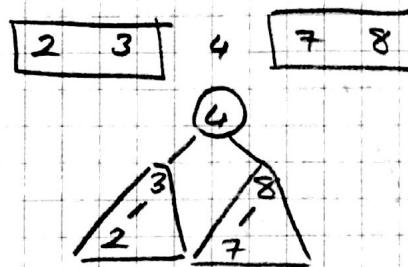
$$T(n) = \Theta(n^3)$$

$$S(n) = \Theta(n^2)$$

Optimal Binary Search Tree

Binary Search Tree'de optimallik:

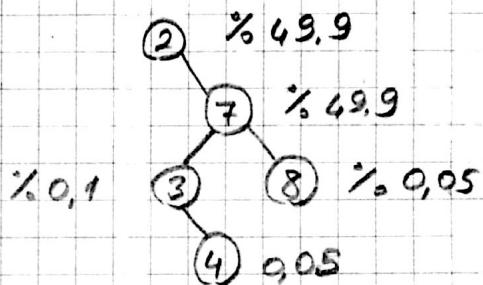
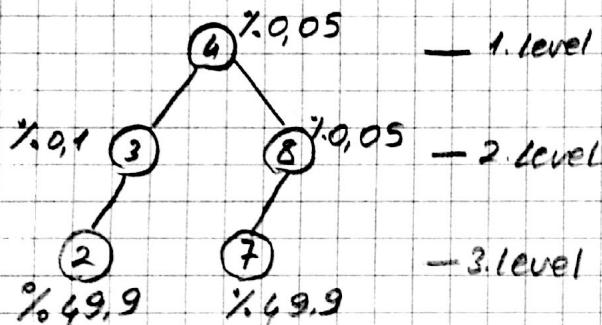
- min yükseklilik ise
 - array'ı sırala $\Theta(n \log n)$
 - tree oluştur.



- Elemanların bulunma ihtimallerinin min olması.

$$\min \sum_{i=1}^n p_i \cdot l_i \rightarrow \text{level}$$

olasılık ↴

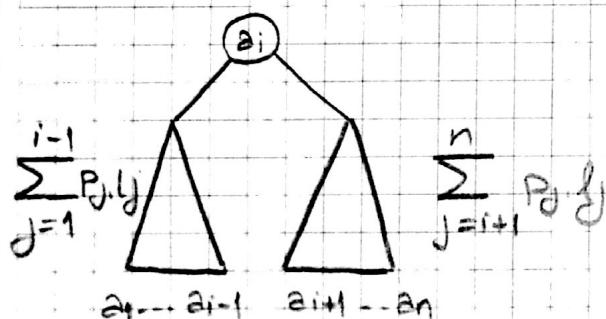


$$A = \frac{0,05 + 0,2 + 0,1 + 29,94}{100} \approx 3$$

$$A \approx 1,5$$

* dynamic programming'e göre :

- sequence of decisions



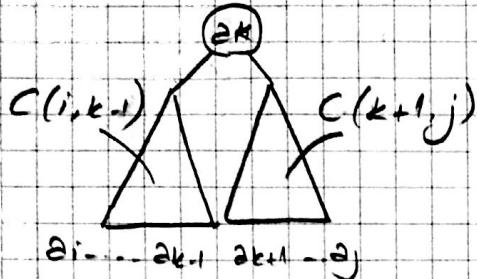
$a_1 \dots a_{i-1} \dots a_n \rightarrow$ sıralı elemanlar

- 1. kezde adıx : root'a ne seçilecektir
↳ 7 tane ihtimal

$$\underbrace{\sum_{j=1}^{i-1} P_j(l_j+1)}_{\text{sol tarafl}} + \underbrace{\sum_{j=i+1}^n P_j(l_j+1)}_{\text{sağ tarafl}} + P_i = \underbrace{\sum_{j=1}^{i-1} P_j \cdot l_j}_{\text{root}} + \underbrace{\sum_{j=i+1}^n P_j \cdot l_j}_{+ \sum_{j=i+1}^n P_j(l_j+1)}$$

$$\sum_{j=1}^{i-1} P_j \cdot l_j + \sum_{j=i+1}^n P_j \cdot l_j + \sum_{j=i+1}^n P_j(l_j+1)$$

- Eğer sonuc optimalse olsaydı haller optimaldır.



$C_{ij} = a_i \text{ olun } a_j \text{ ye kadar sayılmam optimal bst'si.}$

$$C(i,j) = \min \left(C(i, k-1) + C(k+1, j) + \underbrace{\sum_{m=i}^j p_m}_{w(i,j)} \right)$$

$$C(i,j) = 0 \quad j < i$$

$$\begin{matrix} C(i,j) \\ \swarrow \quad \searrow \\ 1 \dots n \quad 0 \dots n \end{matrix}$$

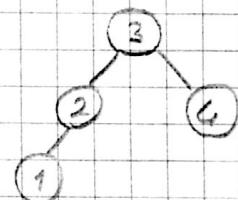
| | 0 | 1 | ... | n |
|-----|---|-------|-----|-------|
| 1 | 0 | p_1 | | |
| 2 | 0 | p_2 | | |
| 3 | 0 | p_3 | | |
| ⋮ | 0 | p_4 | | p_5 |
| n+1 | 0 | | | 0 |

dr:

| | |
|---|-------------------|
| A | $\rightarrow 0.1$ |
| B | $\rightarrow 0.2$ |
| C | $\rightarrow 0.4$ |
| D | $\rightarrow 0.3$ |

| | 0 | 1 | 2 | 3 | 4 |
|---|------|---------|---|---|---|
| 1 | 0.01 | 0.40197 | | | |
| 2 | 0 | 0.20834 | | | |
| 3 | 0 | 0.891 | | | |
| 4 | 0 | 0.03 | | | |
| 5 | 0 | | | | |

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 1 | 0 | 2 | 3 | 3 | 3 |
| 2 | | 2 | 3 | 3 | 3 |
| 3 | | | 2 | 4 | 4 |
| 4 | | | | 4 | 4 |
| 5 | | | | | |



main table

root table

$$C[1,2] = \min \left\{ \begin{array}{ll} C[1,0] + C[2,2] + p_1 + p_2 & k=1 \\ C[1,1] + C[3,2] + p_1 + p_2 & k=2 \end{array} \right\}$$

```

for i=n to 1
  for j=i to n
    w = w(i,j)
    C(i,j) = ∞
    for k=i to j
      x = C(i,k-1) + C(k+1,j)
      if x < C(i,j)
        C(i,j) = x
        w += p_k
    C(i,j) += w
  
```

$$T(n) = \sum_{i=1}^n \sum_{j=1}^n \sum_{k=i}^j 1 = \Theta(n^3)$$

↳ tabloyu dolurma costu

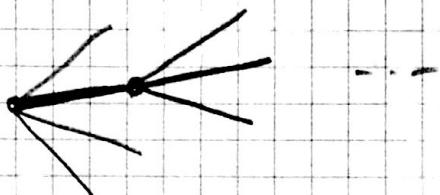
- Root table etkileşimi.
- Root table dan okuma $\Theta(n)$ (recursive)

Greedy Technique

05.12.2012

- Optimizasyon problemlerde kullanılır.
- Karar adımlarından birini seçip从此:

 - feasible olsalı, problemin kısıtlamalarını sağlamalı.
 - locally optimal olsalı, local en iyi olsalı.
 - irreversible olsalı, geri çeviremez olmalı, sonra geri dönmeyebilir.



Bütün durumlarda baktılmasız.

Greedy Algorithm ($a[1 \dots N]$)

```

solution = 0
for i=1 to N
    x = select (a)
    if feasible (solution, x)
        solution = solution U {x}
return solution
    }
```

- Optimal sonuc vermeyen greedy teknik algoritması olabilir, fakat yine de kullanışlı olurular ve çok etkisiz olabilecek sorunlarla (Travel Salesman vb.) kullanılır.

Knapsack Problem

1. segim : en hafifli ol.
2. segim : sonraki en hafifli ol.

| | 1 | 2 | 3 |
|---|---|---|---|
| P | 4 | 5 | 6 |
| w | 4 | 5 | 6 |

$$m=10 \text{ için } \frac{4+5}{4+5} = 1 = 0$$

$$\text{optimal cutting} = 4+6 = 10$$

optimal adetle vermedi

kadar adetin değiştirelim.

- 3) 1. segim : en değerliyi ol.

| | 1 | 2 | 3 |
|---|---|---|---|
| P | 6 | 5 | 6 |
| w | 4 | 5 | 6 |

$$m=9 \text{ için } \frac{6+5}{6+5} = 1 = 0$$

$$\text{algorithm} = 6$$

$$\text{optimal cutting} = 9$$

optimal vermedi.

kadar adetin değiştirelim.

3) hem ağırlığı, hem değeri batacak.

$$\max \frac{P_i}{w_i}$$

Karar adımı: $\max \frac{P_i}{w_i}$ 'yi seç

| | 1 | 2 | 3 |
|---|---|---|---|
| P | 4 | 5 | 6 |
| w | 3 | 4 | 6 |

$$m=9$$

$$\text{algoritma: } 3+4=7w-9p$$

$$\text{optimal: } 3+6=9w-10p$$

optimal çözümü vermedi.

Fractional Knapsack Problem:

Nesnelerin hepsi sızımlı almak zorunda değilse.

$$0 \leq x_i \leq 1$$

1) En düşük ağırlığı alalım.

| | 1 | 2 | 3 |
|---|---|---|---|
| P | 3 | 4 | 6 |
| w | 4 | 5 | 6 |

$$m=10$$

$$\text{algoritma: } 3+4+1=8p$$

$$\text{optimal: } 6+\frac{4}{5} \cdot 4 = 9,2p$$

optimal çözümü vermemis.

2) En yüksek değeri batacak.

| | 1 | 2 | 3 |
|---|---|---|---|
| P | 3 | 4 | 4 |
| w | 4 | 5 | 6 |

$$m=10$$

3) Hem ağırlığı, hem değeri batacak.

| | 1 | 2 | 3 |
|---|---|---|---|
| P | 3 | 4 | 4 |
| w | 4 | 5 | 6 |

$$m=10$$

$$\text{algoritma: } 4+3+4 \cdot \frac{1}{6}$$

$$= 7,67$$

→ optimal çözüm

- $\Theta(n^2)$ de yapılabilir.

- hep küləsət $\Theta(n \log n)$

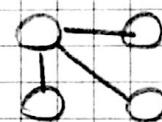
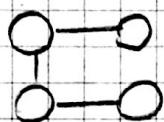
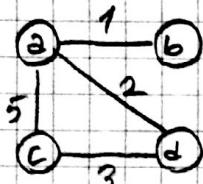
- presort ke $\Theta(n \log n)$ de yapılabilir.

Min spanning Tree (MST)

- Ağırlıklı, bağılı, yörük graf verilmiştir.
- Bir grafın bütün node'larına ulaşan tree, spanning tree.

$G(V, E)$ \rightarrow input graf

$G'(V, E')$ \rightarrow tree $E' \subseteq E$



bir spanning tree'dır.

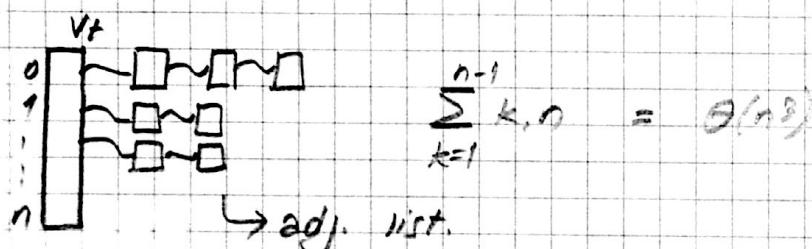
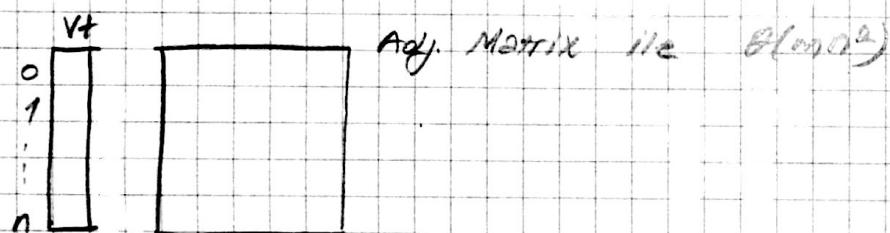
- Spanning tree'yein edge sayısı $(n-1)$

Greedy çözüm: (Prim's Algorithm)

Bir vertexden başlayıp, en küçük edge'e giderim. Bu iki vertexden gidişebilerek en küçük edge'e giderim ...

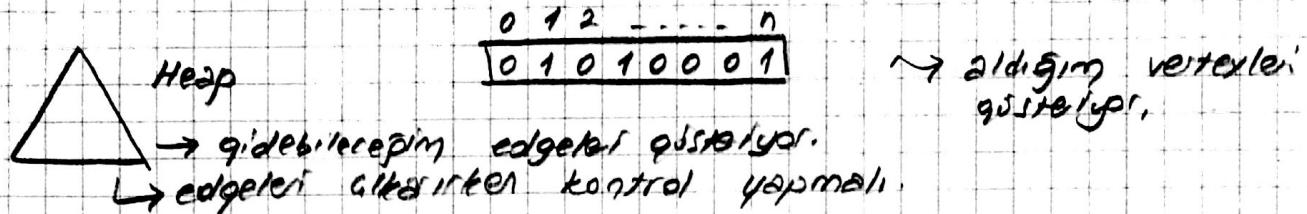
$$|V|=n \quad |E|=m$$

- tek bir edge etlenmek için $m.n$ işlem yapılıyorsa
- spanning tree için



En küçük edge'i bulma kısmını hazırladıralım.

- Edgeleri heap'e koymalı.



$$\text{heap'e ekleme : } \underbrace{2m \cdot \log(n^2)}_{\Theta(m \log n)} = \Theta(m \log n)$$

- Edge'ler yerine vertexleri seçelim. Vertex'ler içlerinden min edge'i tutsun.

| | | | | | |
|---|------------|---|---|---|---|
| a | b | c | d | e | f |
| ∞ | (2,5)(2,3) | ∞ | ∞ | ∞ | |

- for $i=1$ to $n-1$
[find $\rightarrow \Theta(n^2)$
add $\rightarrow O(n)$
update $\rightarrow \underbrace{O(m)}_{\text{Adj. List}} \sim \underbrace{O(n^2)}_{\text{Adj. Matrix}}$

} $\Theta(n^2)$ de yapılırs.

deney $\rightarrow m = \Theta(n^2)$ \rightarrow ilk heap'li algoritma daha iyi! ⚡
sparse $\rightarrow m = \Theta(n)$ \rightarrow $\Theta(n^2)$ 'li algoritma daha iyi! ⚡
(seyrekt)

- liste yerine heap'e geçsek

for $i=1$ to $n-1$

$$\begin{array}{l} \text{find } \rightarrow O(n \log n) \\ \text{add } \rightarrow O(n) \\ \text{update } \rightarrow O(m \log n) \\ \hline O(m \log n) \end{array}$$

$$T_0 = (\{v_0\}, \emptyset) \subseteq MST$$

$$T_1 = (\{v_0, v_1\}, \{(v_0, v_1)\}) \subseteq MST$$

$$T_2 = (\{v_0, v_1, v_2\}, \{(v_0, v_1), (v_0, v_2)\}) \subseteq MST$$

$T_{n-1} = (V, E') \leftarrow$ Bu tree MST midir?

$$T_{n-1} = (V, E') \subseteq MST = (V, E^*)$$

$$(n) \sqsubset (n-1) \quad (n) \sqsubset (n-1)$$

Bu durumda
 T_{n-1} = MST

proof by induction:

$$T_0 \rightarrow T_1 \rightarrow T_2 \dots \rightarrow T_{n-1} \quad 0 \leq i \leq n-1 \quad (T_i)$$

- $i=0$ için $T_0 = (\{v_0\}, \emptyset) \subseteq MST$ olmalıdır.

- assume $i=k$ için

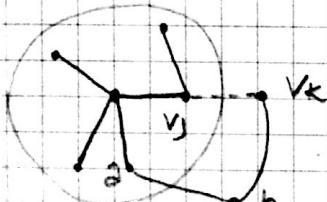
$$T_k = (\underbrace{\{v_0, \dots, v_k\}}_{V_k}, \underbrace{\{(\) (\) \dots\}}_{E_k}) \subseteq MST \text{ olmalıdır.}$$

- proof $i=k+1$ için

$$T_{k+1} = (V_{k+1}, E_{k+1}) \subseteq MST \quad \Rightarrow \text{Proof by contradiction.}$$

$$V_{k+1} = V_k \cup \{v_{k+1}\}$$

$$E_{k+1} = E_k \cup \{(v_j, v_{k+1})\} \quad v_j \in V_k$$



v_k etendeğinde $\{(v_j, v_k)\}$ in MST olusmadığını disimilem.

$$(v_j, v_k) \quad v_j \in V_k$$

$$v_k \notin V_k$$

$$c(v_j, v_k) \leq c(v_a, v_b)$$

$$(v_a, v_b) \quad v_a \in V_k$$

$$v_b \notin V_k$$

$$MST = (V, E^*) \quad T = (V, E^* - \{(a, b)\} \cup \{(v_j, v_k)\})$$

$$C(MST) \geq \underbrace{\sum_{e \in E^*} c(e)}_{C(T)} - c(a, b) + c(v_j, v_k) \leq 0$$

$C(MST) = C(T)$ Hicbir ağız MST'den 22 costlu olameyeceğini
için T MST olur.

Kruskal'ın Algoritması:

$$G_0 = (V, \emptyset) \quad n \text{ tane ağız}$$

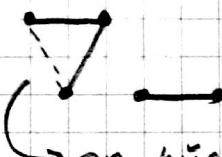
$$G_1 = (V, \{e\}) \quad n-1 \text{ tane ağız}$$

$$G_2 = (V, \{e_1, e_2\}) \quad n-2 \text{ tane ağız}$$

$$G_{n-1} = (V, \{e_1, e_2, \dots, e_{n-1}\}) \quad 1 \text{ tane ağız}$$

MST ıgin formu

- her adımı en küçük edge'si seç
- 2 ağızı birleştirir
- cycle oluşmasını engeller



en küçük edge olsa da eklemese. tek ağız oldugu için.

approach

$$\left[\begin{array}{l} \text{sort } w(e_1) \leq \dots \leq w(e_m) \\ \dots \end{array} \right] \quad \left. \begin{array}{l} \text{mloop} \\ \dots \end{array} \right]$$

while
acyclic

$\left. \begin{array}{l} m \\ n \end{array} \right]$

$$\frac{+}{m \log m + mn}$$

acyclic : depth first search bulanıca $\Theta(n)$ 'de yaparız.

$$K = G_{n-1} = (V, \{e_1, e_2, \dots, e_{n-1}\})$$

$c(e_1) \leq c(e_2) \leq c(e_3) \dots$

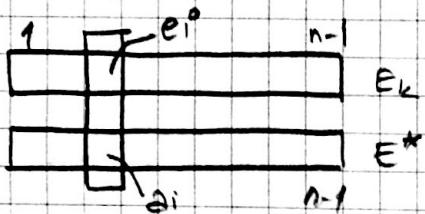
$$MST \Rightarrow G^* = (V, \{e_1, e_2, \dots, e_{n-1}\})$$

$c(e_1) \leq c(e_2) \dots$

$K = G^*$ olursa Kruskal optimal olur.
 $K \neq G^*$ ise farklılığı bulup düzelt.

$$G^* \rightarrow G_{(1)}^* \rightarrow G_{(2)}^* \dots G_{(k)}^* \rightarrow K$$

↑
- feasible'lik degismeyecek
- optimal olacak



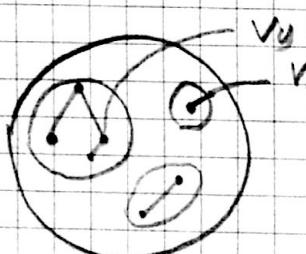
$e_i \neq a_i$ ise E^* 'a e_i yi eklem, cycle oluşturulan edge'lerden birini çıkarırırmı.

$$c(G^*) \geq c(G^*(1))$$

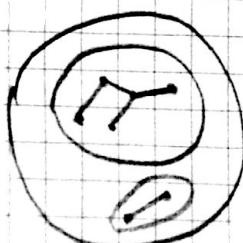
\leq
=

Disjoint Subsets Union-Find

Graf'in bağımlı alt kümeleri



$$(e_{12}) = (Vx, Vy)$$



find \rightarrow Vx, Vy inin subsetlerini bul.

$$G_0 = (V, \emptyset)$$

\hookrightarrow makeset \rightarrow n tane disjoint set oluştur.

- find
 - makeset
 - union
- \rightarrow bul
 \rightarrow oluştur.
 \rightarrow birleştir.

$$S = \{1, 2, 3, \dots, 6\}$$

makeSet: $\{\{1\}, \{2\}, \{3\}, \{4\}, \{5\}, \{6\}\}$

union : $(1, 4)$ and union $(5, 2)$

$$\{\{1, 4\}, \{5, 2\}, \{3\}, \{6\}\}$$

union : $(4, 5) \rightarrow$ once find yap sonra union

$$\{\{1, 2, 4, 5\}, \{3\}, \{6\}\}$$

| index | 1 | 2 | 3 | 4 | 5 | 6 |
|-------|---|---|---|---|---|---|
| set | 1 | 2 | 3 | 4 | 5 | 6 |

← makeSet

makeSet $\Theta(1)$
find $\Theta(1)$
union $\Theta(n)$

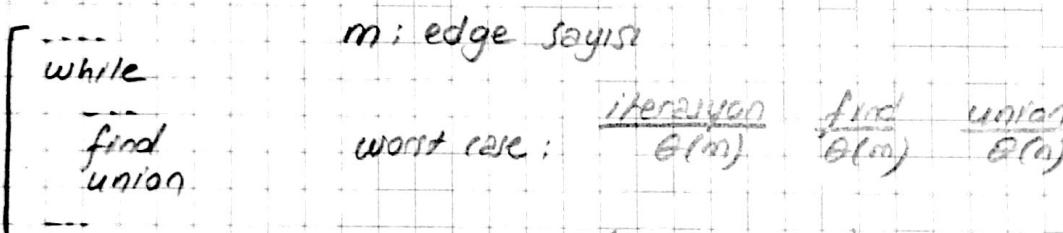
| index | 1 | 2 | 3 | 4 | 5 | 6 |
|-------|---|---|---|---|---|---|
| set | 1 | 5 | 3 | 1 | 5 | 6 |

union $(1, 4)$
union $(2, 5)$

| index | 1 | 2 | 3 | 4 | 5 | 6 |
|-------|---|---|---|---|---|---|
| set | 1 | 1 | 3 | 1 | 1 | 6 |

union $(4, 5)$ hepsi 1 yap, ya da
hepsi 5 yap.

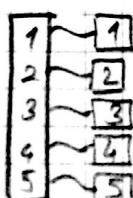
Kruskal ıgın düşünelim:



Kruskal: $O(n^2 + m \log m)$

• Algoritmayı hızlandırmak için nasıl geliştirilebilir?

array yerine list tutelim.



makeSet $\Theta(1)$
union $\Theta(1)$
find $\Theta(n)$

Kruskal: $\Theta(mn) \approx \Theta(n^2)$

array ile liste bir anda tutarsak

makeset $\Theta(1)$

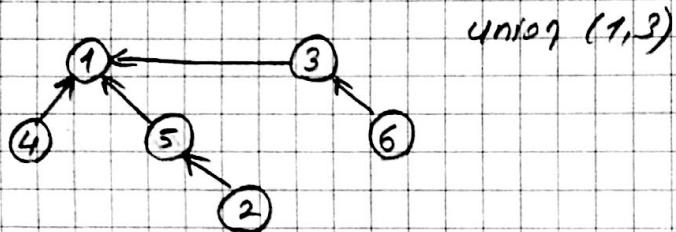
find $\Theta(1)$

union, $\Theta(n)$

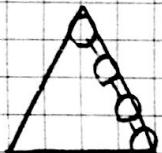
$$K = \Theta(n^2)$$

Büyüğe küçüğü ekleyince $\Theta(1)$ olur.

quick union



path compression:



yol üzerindeki bütün elementler root'a bağla.

makeset $\Theta(1)$

find $\Theta(1)$

union $\Theta(1)$

$$\} K = \Theta(m + 2n + m \log n)$$

$$= \Theta(m \log m)$$

Single Source Shortest Path

Tek bir belli source vertexinden bulmak istiyoruz.

- Dijkstra's Algorithm
dijitaline en kısa yoldan

- input: undirected, connected graph
- Problem instance: graph

$G = (V, E)$ adjacency matrix - usf. olabilir.

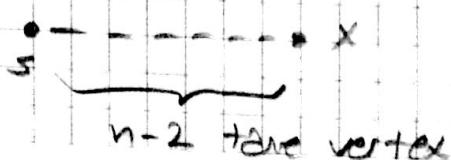
- Problem output:

1) $(n-1)$ tane list

- min $\Theta(n)$

- max $\Theta(n^2)$

yer
İhtiyaçlı



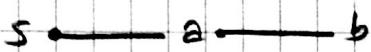
$$G = (V, E)$$

\downarrow
 $(n-1)$
shortest
paths

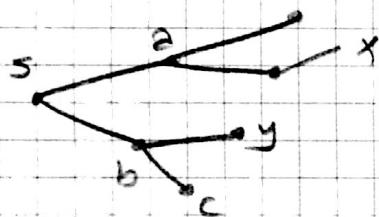
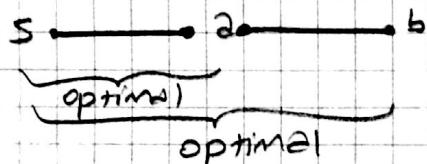
Yer intisici
list $\rightarrow \Theta(n+m)$
matrix $\rightarrow \Theta(n^2)$

min $\Theta(1)$
max $\Theta(n^2)$

- 2) Path'leri depisit bir representasyon olarak tutalim.
• shortest paths tree!



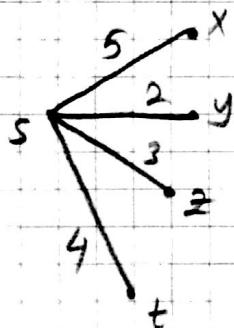
s'den b'ye giden en kisa yol s'den geciyorsa ikisi birlikte tutalim. (optimallik presribi)



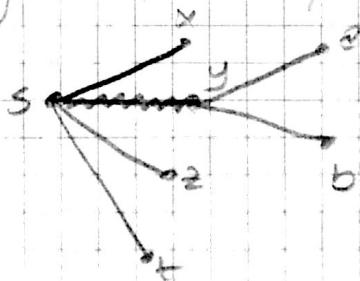
n tane vertex
n-1 tane edge var

$$\text{list} \rightarrow \Theta(n+n-1) = \Theta(n)$$

Dijkstra'nin Algoritması



- 1. kizit adimi: s'den kisa olan sec.
 $\rightarrow (s, y)$ yi secilek. s'den y'ye gidebilecek en kisa yel olusanday emrine. Guncel oliper hali yel onun hali kisa olmas.
- 2. kizit adimi: s'in ve y'nin gidebilecegi en kisa yolu sec.



$\{x, z, t, y\}$
kumelerde
en kucuk olan
sec.

$$T_0 = (s, \emptyset)$$

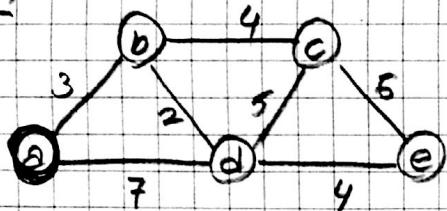
$$T_1 = (\{s, y\}, \{s, y\})$$

$$T_2 = (\{s, y, c\}, \{s, y, (y, c)\})$$

$$T_3 = (\{s, y, c, t\}, \{s, y, (y, c), (s, t)\})$$

$$T_{n-1} = (V, E')$$

dr:



a source olsun.

yol \leftarrow usuluk

$$1. \text{ addm}) \quad a(-, 0)$$

$$\underline{b(0, 3)}, \underline{c(-, \infty)}, \underline{d(2, 2)}, \underline{e(-, \infty)}$$

$$2. \text{ addm}) \quad b(2, 3)$$

$$\underline{c(b, 7)}, \underline{d(b, 5)}, \underline{e(-, \infty)}$$

$$3. \text{ addm}) \quad d(b, 5)$$

$$\underline{c(b, 7)}, \underline{e(d, 9)}$$

$$4. \text{ addm}) \quad c(b, 7)$$

$$\underline{e(d, 9)}$$

$$5. \text{ addm}) \quad e(d, 9)$$

| | a | b | c | d | e |
|---|---|---|---|---|---|
| s | a | b | b | d | |

enqueue sırası: $e \rightarrow d \rightarrow b \rightarrow a$

- for $i=0$ to $n-1$

$dr \rightarrow \infty$

$pv \rightarrow \text{null}$

Insert(Q, v, dr)

$ds \rightarrow 0$

Decrease(Q, s, ds)

$V_T \rightarrow 0$

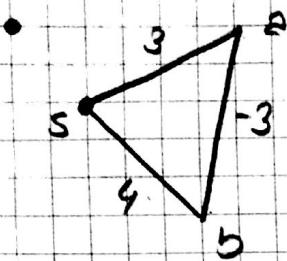
for $i=0$ to $n-1$

$min = \text{DeleteMin}(Q)$

$V_T = V_T + min$

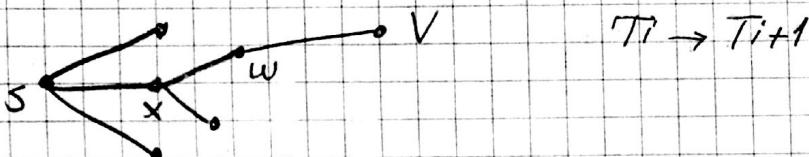
for (V_T de bulunmayan elementlerin min 'e göre update et)

Ispatlayalım:



- v 'in 3 sekilli, fakat v 'in degeri 1 olmasa iyi.
- Negatif edge olunca dijkstra uygun deger.

• Birin edge'lerin pozitif oldugu durumda optimal midir?



s'den v'ye giden yolu $s-x-w-v$ secdigimizi dusurelim.
Her adimda secdigim edge'e pos update yaptim. Burden
oldayi en kisa yolu secdim. Baska bir edge'i secme
ikhtimalim olamaz guncel en kisa yolları seceret ilerledim

- Onceki gittigim vertexlerden biri oldunde, gunku eger en kisa yol ordeki gelseydi onu secdim.
- Gitmedigim vertexlerden bir oldunde, gunku eger en kisa yolu verseydi, onceki gittigini oldurdum.

Complexity

Bit problem' iin diziyeinous algoritmların karmaşıklığı

Problem

Sıralama

Algoritma

Quicksort
Mergesort

Analiz

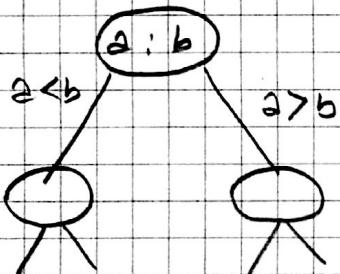
$O(n^2)$
 $\Theta(n \log n)$

Upper Bound Complexity: $\downarrow O(n^2)$
 $\Theta(n \log n)$

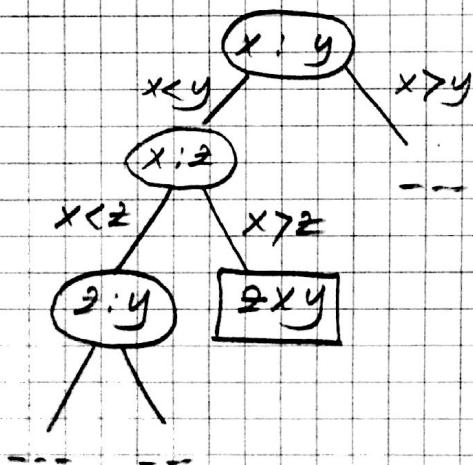
Lower Bound Complexity: $\uparrow \Omega(n)$

- Trivial lower bound
- Decision Tree
- Kısıtlı süre zarfı algoritmalar için.

Decision Tree



$d'_r \{x, y, z\}^r y_i$ sıralamak için



- n tane elemanı sıralayan bir n! tane farklı durum oluyur.
- Worst Case: her bir günükü



yükseklik $n!$



yükseklik
 $\log(n!) \approx \frac{1}{2} \log n!$

- Sıralama algoritması için lower bound $\Omega(n \log n)$ bulundu.
- Bu durumda sıralama algoritması $\Theta(n \log n)$ dir.

Tractable: Polynomially solvable problems

- Tractable olmayan problemlere çözüm:

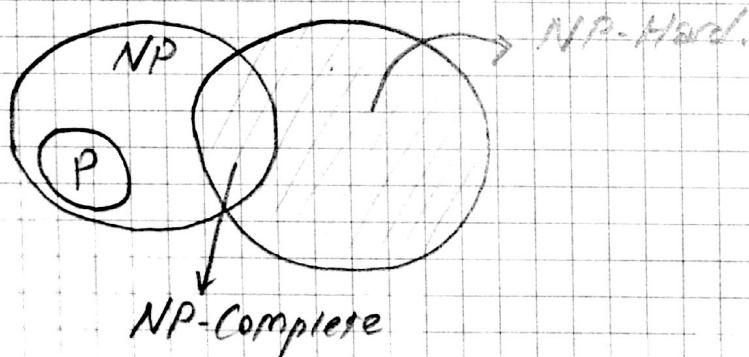
- optimallıklık veya game . (yaklaşık sonuc)
- özel bir durum , oynama -
- denarimsal (zamandan fedakarlık)
 - ↳ backtracking
 - ↳ branch & bound.

P: polynomial

NP: non-deterministic polynomial : polinomik zamanda çözünen algoritmları herkesin hedefi.

NP-Complete : 2 or NP problems

NP-Hard : genellikle optimizasyon problemleridir. Polinomik zamanda ispat yapmayı gereklidir.



Algoritma Final Soruları:

1-

```
for i=0 to n-1  
  for j=i+1 to n-1  
    if A[i]+A[j]==k  
      return true  
  return false.
```

Algoritmanın zaman
kompleksitesi bulunuz.

- a- $T(n)$
- b- $Tw(n)$

(5)

2- Bir listenin en büyük elemanını bulan D&C, Decrease & Conquer ve T&C algoritmalarını yazınız. Zaman orası karşılaştırın.
(25)

3- Coin Change problemini çözün

- a- Dynamic programming algoritması (30)
- b- Greedy algoritması (20)
- c- 2 algoritmayı karşılaştırın (15)

4- Verilen graf için B node'una ait single source shortest path'lerini bulunuz. (10)

5- BubbleSort algoritması verilmiştir. Decision Tree'sini çiziniz. Kası kesişmesine yapınız.
(15)

(110 puan
üçinden)

Algoritma Final soruları:

- 1- `for i=0 to n-1
 for j=i+1 to n-1
 if A[i][j] + A[j][i] == k
 return true
 return false.`
- Algoritminin zaman karmaşıklığını bulunuz.
a- $T(n)$
b- $T(n^2)$ (5)
- 2- Bir listenin en büyük elemenini bulan D&C, Decrease & Conquer algoritmalarını yazınız. Zaman olarak kıyaslayın. (25)
- 3- Coin Change problemini çözün
a- Dynamic programming algoritması (30)
b- Greedy algoritması (20)
c- 2 algoritmayı kıyaslayınız. (15)
- 4- Verilen graf için B node'una ait single source shortest path'lerini bulunuz. (10)
- 5- BubbleSort algoritması verilmiştir. Decision Tree'sini çiziniz. Kasıtlılaşmamış yapınız. (15)
- (110 puan üzerinden)

2012 yıl Algoritma Notları
Hoca: Erdal Sarıger

Mehmet ERDAL