ALGORİTMA ANALİZİ

..... / ..... / .....

* Bir algoritmanın karmaşıklığını belirleyen faktörler:
→ Eleman sayısı
→ Elemanların ne olduğu (tipi ve sırası)

* Bir algoritmayı nasıl ifade ederiz?
→ flowchart
→ Sözlü ifade (oral presentation)
→ Pseudo Kod ★
→ Programming language

* Basic issues related to algorithm
→ design algorithm
→ express algorithm
→ proving correctness
→ efficiency
→ optimality

* Algoritma tasarlama stratejileri:
① Brute force: olası seçenekleri tek tek denemek
② Divide and conquer: alt parçalara bölerek çözmek
③ Greedy approach: her aşamada en uygun seçeneği seçme
④ Dynamic programming:
⑤ Decrease and conquer:
⑥ Transform and conquer:
⑦ Backtracking and branch and bound:
⑧ Space and time tradeoffs:

* Bir algoritmanın iyi olması?
① correctness
② time efficiency
③ space efficiency

..... / ..... / .....

Lyrebird Studio

* Daha iyi algoritma var mı?
→ Sınırlara bak.
→ Optimal olup olmadığına bak.

ALGORİTMA

- Sonlu, tanımlı adımlar, input, output, efektif.

understand the problem → exact vs approximate solving algorithm design technique → design an algorithm → prove correctness → analyze the algorithm → code the algorithm

Algoritma analizinde dikkat edilmesi gereken noktalar

1. Simplicity
2. Generality
3. Time efficiency
4. Space efficiency

Algoritma karmaşıklığı

1. Best case
2. Average case
3. Worst case

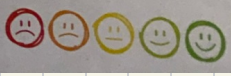
Linear search

Big-Oh notation

1 5 3 2 4 8 unsorted — O(n) (worst case) / O(1) (best case)
1 2 3 4 5 8 sorted — O(n) (worst case) / O(1) (best case)
number of elements : n

finding max within an array (n)

O(n) (worst and best case) unsorted
O(1) (worst and best case) sorted



..... / ..... / .....

Pseudocode Details

→ Control Flow
- if ... then ... [else]
- while ... do ...
- repeat ... until ...
- for ... do ...
- indentation replaces braces

→ Method declaration
- Algorithm method (arg, arg, ...)
- input ...
- output ...

```
Algorithm arrayMax(A, n)
Input: array A of n integers
Output: maximum element of A

currentMax ← A[0]
for i ← 1 to n-1 do
  if A[i] > currentMax then
    currentMax ← A[i]
return currentMax
```

! Büyüme hızı sabitlerden ve düşük dereceli terimlerden etkilenmez.
$10n^2 + 2n + 5$

Big-Oh Örnekleri:

→ $7n-2$

$7n-2$ is $O(n)$
need $c>0$ and $n_0 \geq 1$ such that $7n-2 \leq c \cdot n$ for $n \geq n_0$
this is true for $c=7$ and $n_0 = 1$

→ $3n^3 + 20n^2 + 5$

$3n^3 + 20n^2 + 5$ is $O(n^3)$
need $c>0$ and $n_0 \geq 1$ such that $3n^3 + 20n^2 + 5 \leq c \cdot n^3$ for $n \geq n_0$
this is true for $c=4$ and $n_0 = 21$

→ $3 \log n + 5$

$3 \log n + 5$ is $O(\log n)$
need $c>0$ and $n_0 \geq 1$ such that $3 \log n + 5 \geq c \cdot \log n$ for $n \geq n_0$
this is true for $c=8$ and $n_0 = 2$

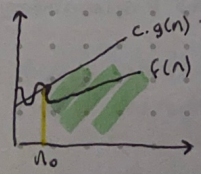
constant ≈ 1 sıralı bir dizide maksimum elemanı bulmak
logarithmic ≈ log n binary search
linear ≈ n sırasız bir dizide maksimum elemanı bulmak
NlogN ≈ nlogn heap sort
quadratic ≈ n^2 selection sort
cubic ≈ n^3 matris çarpımı
exponential ≈ 2^n alt küme sayısı
factorial ≈ n! permütasyon bulma

big-oh ⇒ upper bound
big-omega ⇒ lower bound Ω
big-theta ⇒ tight bound Θ
→ lower bound ile upper bound aynıyken tight bound kullanırız.

örnek
$f(n) = 4n^2 + 2n$ $g(n) = n$ $f(n) \in O(g(n))$ midir?
f(n) is O(g(n))
$f(n) \leq c \cdot g(n)$ $n \geq n_0$, $c > 0$
$4n^2 + 2n \leq c \cdot n$
$4n + 2 \leq c$
$n \leq \frac{c-2}{4}$ ⇒ yanlıştır.

örnek
$f(n) = 2n + 10$ $g(n) = n$ $f(n) \in O(g(n))$?
$2n + 10 \leq c \cdot n$ $n \geq n_0$, $c > 0$
$10 \leq (c-2)n$ $c = 3$, $n = 10$ ⇒ sağlar.

$g(n) \in \Omega(f(n))$ sağlar mı?
$n \leq c(2n + 10)$
$(1 - 2c)n \leq 10c$ $c = 1$ $n = 0$ ⇒ sağlar.

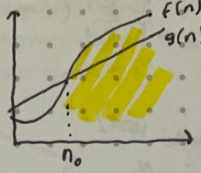


Big-Omega

$f(n) \in \Omega(g(n))$ if $c > 0$, $n_0 > 0$, $n \geq n_0$

↓ lowerbound

$f(n) \geq c \cdot g(n)$



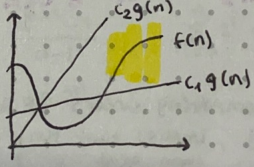
* sequential search için best case $\Omega(1)$ doğru mu? Evet.
* insertion sort için best case $\Omega(1)$ doğru mu? Evet ama en yakın lowerbound seçmek daha mantıklı olduğundan $\Omega(n)$ daha doğru olur.

Big-theta

$0 \leq c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$ $c_1 \leq c_2$, $n_0 > 0$, $n \geq n_0$

$f(n) \in \Theta(g(n))$

tightbound



örnek

$f(n) = 2n+5 \in \Theta(n)$?

$2n+5 \in O(n)$

$2n+5 \in \Omega(n)$

olmalı

$2n+5 \geq c_1 n$

$5 \geq (c_1-2)n$ $n_0 = 1$ $c_1 = 3$

$2n+5 \leq c_2 n$

$5 \leq (c_2-2)n$ $n_0 = 1$ $c_2 = 7$

} sağlar

Insertion sort

5 7 3 1 2 4 6

5 7 3 1 2 4 6

3 5 7 1 2 4 6

1 3 5 7 2 4 6

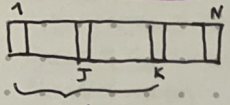
1 2 3 4 5 6 7

worst case $O(n^2)$, $\Omega(n^2)$, $\Theta(n^2)$

average case arka sayfada

best case $O(n)$, $\Omega(n)$, $\Theta(n)$

average case:



dış döngüde for, iç döngüde while kullanırız.

K-J+1 adım olacak. (içte)

$$\frac{1}{K}\sum_{J=1}^{K}(K-J+1) = \frac{1}{K}\left(K^2 - \frac{K(K+1)}{2} + K\right)$$

$$= \frac{2K^2 - K^2 - K + 2K}{2K} = \frac{K+1}{2}$$

(inner loop işlem sayısı)

(outer loop)

$$\sum_{K=2}^{N}\frac{K+1}{2} = \frac{1}{2}\cdot\frac{(N+1)(N+2)}{2} = \underbrace{\frac{1}{4}\left[N^2+3N+2\right]}_{\text{average case } O(N^2)}$$

örnek

$n \in O(n^2)$

$100n+5 \in O(n^2)$? ✓ $n \in O(n)$

$\frac{1}{2}n(n-1) \in O(n^2)$? ✓ $n^2 \in O(n^2)$

$n^3 \in O(n^2)$? ✗ $n^3 \in \Omega(n^2)$, $n^3 \notin O(n^2)$

Using limits for comparing orders of growth

$$\lim_{n\to\infty}\frac{t(n)}{g(n)} = \begin{cases} 0, & \text{implies that } t(n) \text{ has a smaller order of growth than } g(n) \\ c, & \text{" the same order "} \\ \infty, & \text{" a larger order "} \end{cases}$$

örnek

$t(n) = \frac{1}{2}n(n-1)$ $\quad g(n) = n^2$

$$\lim_{n\to\infty}\frac{\frac{1}{2}n(n-1)}{n^2} = \frac{1}{2}\lim_{n\to\infty}\frac{n^2-n}{n^2} = \frac{1}{2}\lim_{n\to\infty}\left(1-\frac{1}{n}\right) = \underbrace{\frac{1}{2}}_{\text{sonuç sabit}}$$

sonuç sabit olduğundan aynı büyüme hızındalar.

..... / ..... / .....

Useful Formulas for the Analysis of Algorithms

$\log x^y = y \cdot \log x$

$\log xy = \log x + \log y$

$\log x/y = \log x - \log y$

$\log a^x = \log_a b \log_b x$

$a^{\log_b x} = x^{\log_b a}$

Permütasyon → $P(n) = n!$

Kombinasyon → $C(n,k) = \frac{n!}{k!(n-k)!}$

Alt küme → 2^n

Toplam Formülleri

① $\sum_{i=\ell}^{u} 1 = 1+1+\dots+1 = u-\ell+1$ $\qquad \sum_{i=1}^{n} 1 = n$

② $\sum_{i=1}^{n} i = 1+2+3+\dots+n = \frac{n(n+1)}{2} \approx \frac{1}{2}n^2$

③ $\sum_{i=1}^{n} i^2 = 1^2+2^2+\dots+n^2 = \frac{n(n+1)(2n+1)}{6} \approx \frac{1}{3}n^3$

④ $\sum_{i=1}^{n} i^k = 1^k+2^k+\dots+n^k \approx \frac{1}{k+1}n^{k+1}$

⑤ $\sum_{i=0}^{n} a^i = 1+a+\dots+a^n = \frac{a^{n+1}-1}{a-1}$ $\qquad \sum_{i=0}^{n} 2^i = 2^{n+1}-1$

⑥ $\sum_{i=1}^{n} i\,2^i = 1\cdot 2+2\cdot 2^2+\dots+n\cdot 2^n = (n-1)2^{n+1}+2$

⑦ $\sum_{i=1}^{n} \frac{1}{i} = 1+\frac{1}{2}+\dots+\frac{1}{n} \approx \ln n + \gamma \quad (\gamma \approx 0{,}5772)$

⑧ $\sum_{i=1}^{n} \log i \approx n \log n$

Sum Manipulation Rules

① $\sum_{i=\ell}^{u} c \cdot a_i = c \cdot \sum_{i=\ell}^{u} a_i$

② $\sum_{i=\ell}^{u} (a_i \pm b_i) = \sum_{i=\ell}^{u} a_i \pm \sum_{i=\ell}^{u} b_i$

③ $\sum_{i=\ell}^{u} a_i = \sum_{i=\ell}^{m} a_i + \sum_{i=m+1}^{u} a_i$

Yuvarlama işlemler:

$\lfloor 3.8 \rfloor = 3 \quad \lceil 2,7 \rceil = 3$

$\lfloor -3,8 \rfloor = -4 \quad \lceil -2,7 \rceil = -2$

① $x-1 < \lfloor x \rfloor \leq x \leq \lceil x \rceil < x+1$

② $\lfloor x+n \rfloor = \lfloor x \rfloor + n$ and $\lceil x+n \rceil = \lceil x \rceil + n$ for real x and int n

③ $\lfloor n/2 \rfloor + \lceil n/2 \rceil = n$

④ $\lceil \log(n+1) \rceil = \lfloor \log n \rfloor + 1$

Rekürsif Olmayan Algoritmaların Analizi

★ Maksimum elemanı bulma:

```
MaxElement(A[0.... n-1])
    max ← A[0]
    for i ← 1 to n-1 do
        if A[i] > max
            max ← A[i]
    return max
```

ANALİZ İÇİN

n: number of elements

Basic operations: atama, aritmetik işlemler, karşılaştırma
→ en içteki döngüde basic op. bak.

★ basic op: karşılaştırma → A[i] > max
atama → max ← A[i]

} hangisi seçilmeli? 1. çünkü hep karşılaştırma yapılıyor.

$$C(n) = \sum_{i=1}^{n-1} 1 = n-1 \in \Theta(n)$$

$C_{worst}(n)$ or $C_{average}(n)$ or $C_{best}(n)$ yazmadık çünkü üçü için de geçerli analiz yapıyoruz

..... / ..... / .....

★ Bir dizideki elemanlar unique mi?

```
UniqueElements(A[0,...n-1])
  for i←0 to n-2 do
    for j←i+1 to n-1 do
      if A[i] = A[j] return false
  return true
```

basic op: comparison
if A[i] = A[j]

$$C_{worst}(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-2} [(n-1) - (i+1) + 1] = \sum_{i=0}^{n-2} (n-1-i)$$

$$= \sum_{i=0}^{n-2} (n-1) - \sum_{i=0}^{n-2} i = (n-1) \sum_{i=0}^{n-2} 1 - \frac{(n-2)(n-1)}{2}$$

$$= (n-1)\cdot(n-1) - \frac{(n-2)(n-1)}{2} = (n-1)^2 - \frac{(n-2)(n-1)}{2}$$

$$= \frac{(n-1)n}{2} \approx \frac{1}{2}n^2 \in \Theta(n^2)$$

$T(n) \approx c_c C(n) = c \cdot n^2$

running time

c_c → cost of basic op. (c)

c_c is the time of one comparison on the machine in question.

Rekürsif Algoritmaların Analizi

$n! = 1 \cdot 2 \cdot \ldots (n-1) \cdot n = (n-1)! \cdot n$ for $n \geq 1$

★ Factorial(n)

```
if n = 0 return 1
else return Factorial(n-1) * n
```

Basic op → Çarpma

genel ifade → $F(n) = F(n-1) \cdot n$ for $n > 0$

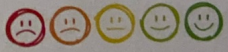
matematiksel ifade → $M(n) = \underbrace{M(n-1)}_{\text{to compute } F(n-1)} + \underbrace{1}_{\text{to multiply } F(n-1) \text{ by } n}$

⇒ Rekürans Bağıntısı

initial condition (durma noktası): if $n = 0$ return 1

$M(0) = 0$ ↳ no multiplication

the calls stop when $n = 0$



$M(n) = M(n-1) + 1$

$M(0) = 0$

Method of backward substitution

$M(n) = M(n-1) + 1$ substitute $M(n-1) = M(n-2) + 1$

$= [M(n-2) + 1] + 1 = M(n-2) + 2$ substitute $M(n-2) = M(n-3) + 1$

$= [M(n-3) + 1] + 2 = M(n-3) + 3$

$\vdots$

$M(n) = M(n-i) + i \quad i \leftarrow n$

$= M(n-n) + n$

$= M(0) + n$

$= 0 + n = n \rightarrow$ Algoritma karmasikligi

* Binary Digit sayisi hesaplama

Binary Digit(n)

if n=1 return 1

else return Binary Digit($\lfloor n/2 \rfloor$) + 1

basic op : toplama islemi

$A(n) = A(\lfloor n/2 \rfloor) + 1$ for $n > 1$

initial condition : if n=1 return 1

$A(1) = 0$

Smoothness Rule : $n = 2^k$

$A(2^k) = A(2^{k-1}) + 1$ for $k > 0$

$A(2^0) = 0$

$A(2^k) = A(2^{k-1}) + 1$ substitute $A(2^{k-1}) = A(2^{k-2}) + 1$

$= [A(2^{k-2}) + 1] + 1 = A(2^{k-2}) + 2$ substitute $A(2^{k-2}) = A(2^{k-3}) + 1$

$= [A(2^{k-3}) + 1] + 2 = A(2^{k-3}) + 3$

$\vdots$

$= A(2^{k-i}) + i \quad i \leftarrow k$

$= A(2^{k-k}) + k$

$= A(2^0) + k$

$= 0 + k$

$\log_2 n \in \Theta(\log n)$

..... / ..... / .....

Exercise: $x(n) = x(n-1) + n$ for $n>0$, $x(0) = 0$

$n \leftarrow n-1$

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

$x(n) = [x(n-2) + n-1] + n = x(n-2) + (n-1) + n$

$= [x(n-3) + (n-2)] + (n-1) + n = x(n-3) + (n-2) + (n-1) + n$

$\vdots$

$= x(n-i) + (n-i+1) + (n-i+2) + \dots + n$

$i=n$ $\vdots$

$= x(0) + 1 + 2 + \dots + n = 0 + \frac{n \cdot (n+1)}{2} = \frac{n(n+1)}{2}$

Exercise: $x(n) = x(n/3) + 1$ for $n>1$, $x(1)=1$, $n = 3^k$

$= [x(n/9) + 1] + 1$

$= [x(n/27) + 1] + 2$

$= [x(n/3^3) + 1] + 3$

$\vdots$

$= [x(n/3^i) + 1] + i - 1$ $i=k$

$= [x(3^k/3^k) + 1] + k - 1$

$= [x(1) + 1] + k - 1$

$= k + 1$ //

Exercise: $\sum_{i=2}^{n-1} \log i^2$ Büyüme derecesi nedir?

$= 2 \sum_{i=2}^{n-1} \log_2 i = \underbrace{2 \sum_{i=1}^{n} \log_2 i}_{\in 2\Theta(n \log n)} - 2 \log_2 n$

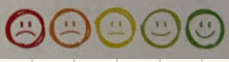
$\Theta(\log n) = \Theta(n \log n)$

Brute Force and Exhaustive Search

Brute force is a straightforward approach to solving a problem, usually directly based on the problem statement and definitions of the concepts involved.

1. $a^n = a \cdot a \cdot a \dots a$
2. $\gcd(m,n)$ → teker teker bölme işlemi
3. selection sort ve bubble sort → tüm elemanlara bakılır.
4. Linear (sequential) Search
5. Brute Force String Matching

6 closest pair
→ convex hull



..... / ..... / .....

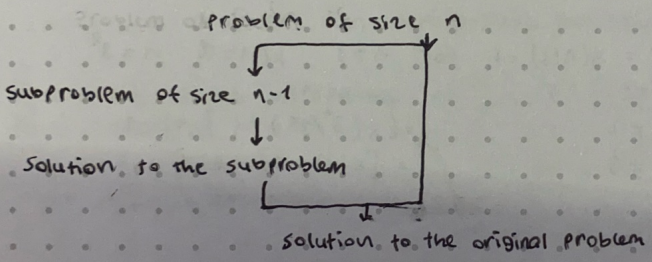
Lyrebird Studio

Exhaustive Search is simply a brute force approach to combinational problem.

1. Traveling salesman problem → Tüm şehirleri gezmek için kısa yol.
2. Knapsack Problem → Hırsız Guvalina neleri sığdırabilir?

Decrease and Conquer

→ Sabit bir değer ile azaltarak çözmek
→ Sabit bir değişken ile azaltarak çözmek
→ Değişen bir değişken ile azaltarak çözmek



* $a^n = a^{n-1} \cdot a$

$$f(n) = \begin{cases} f(n-1) \cdot a, & \text{if } n > 0 \\ 1, & \text{if } n = 0 \end{cases}$$ > decrease by one and conquer

* $a^n = (a^{n/2})^2$

$$a^n = \begin{cases} (a^{n/2})^2, & \text{if } n \text{ is even} \\ (a^{(n-1)/2})^2 \cdot a, & \text{if } n \text{ is odd} \\ 1, & \text{if } n = 0 \end{cases}$$ > decrease by a constant factor

* $gcd(m,n) = gcd(n, m \bmod n)$ > variable size decrease

Binary Search Algorithm

- decrease by a constant factor

input: an array A[0,... n-1] sorted in an ascending order and a search key K

output: index or -1 if there is no such element

pseudocode:

```
l ← 0 ; r ← n-1                          ← iterative çözüm
while ( l ≤ r ) do {
    m ← ⌊(l+r)/2⌋
    if K = A[m] return m
    else if K < A[m]   r ← m-1
    else   l ← m+1
}
return -1
```

$C_w(n) = C_w(\lfloor n/2 \rfloor) + 1$ for $n > 1$, $C_w(1) = 1$

$n = 2^k$

$C_w(2^k) = k+1 = \log_2 n + 1$

$C_w(n) = \lfloor \log_2 n \rfloor + 1 = \lceil \log_2(n+1) \rceil$

Divide and Conquer Algoritmaları

1. Problemi eşit sayıda daha küçük instance'lara ayırırız.
2. Küçük instance'lar çözülür. (genel olarak recursive)
3. İhtiyaç varsa, çözümler birleştirilir.

Master Theorem

If $f(n) \in \Theta(n^d)$ where $d \geq 0$ then

$$T(n) \in \begin{cases} \Theta(n^d) & \text{if } a < b^d \\ \Theta(n^d \log n) & \text{if } a = b^d \\ \Theta(n^{\log_b a}) & \text{if } a > b^d \end{cases}$$

$T(n) = a\,T(n/b) + f(n)$ → problem: bölme ve birleştirme süresi

- a → böldüğümüz parçalardan kaç tane çözmeyi çalıştığımız.
- n → instance sayısı
- b → bölüm sayısı

$A(n) = 2A(n/2) + 1 \rightarrow n^d$

$a = 2$

$b = 2$

$f(n) \in \Theta(n^d) \Rightarrow d = 0$

$A(n) \in \Theta(n^{\log_b a}) = \Theta(n^{\log_2 2^2}) = \Theta(n)$

Merge Sort vs QuickSort

merge → $O(n \log n)$ → best case
average case
worst case

Quick → $O(n \log n)$ → best
$O(n^2)$ → worst
$\Theta(n \log n)$ → average

```
mergesort(A[0...n-1])
  if n > 1
    copy A[0... ⌊n/2⌋-1] to B[0... ⌊n/2⌋-1]
    copy A[⌊n/2⌋ ..n-1] to C[0... ⌈n/2⌉-1]
    Mergesort(B[0... ⌊n/2⌋-1)
    Mergesort(C[0... ⌈n/2⌉-1)
    merge(B,C,A)

merge(B[0... p-1], C[0... q-1], A[0.. p+q-1])
  i←0; j←0; k←0
  while (i<p) and (j<q) do
    if B[i] ≤ C[j]
      A[k] ← B[i]; i←i+1
    else
      A[k] ← C[j]; j ← j+1
    k ← k+1
    if i=p
      copy C[j... q-1] to A[k.. p+q-1]
    else
      copy B[i... p-1] to A[k...p+q-1]
```

basic operation → $n>1$ compare

$C(n) = 2C(n/2) + C_{merge}(n)$ for $n>1$, $\underbrace{C(1)=0}_{\text{initial}}$

$C_{worst}(n) = 2C_{worst}(n/2) + \underbrace{n-1}_{f(n)}$

$a=2$, $b=2$, $d=1$ } $\in \Theta(n^d \log n)$ → mergesort $\in \Theta(n \log n)$

Quick sort

$A[0] \dots A[s-1]\ A[s]\ A[s+1] \dots A[n-1]$

```
Quicksort(A[l...r])
if l<r
    s ← Partition(A[l..r])
    Quicksort(A[l...s-1])
    Quicksort(A[s+1...r])
```

```
Hoare Partition(A[l...r])
p ← A[l]
i ← l ; j ← r+1
repeat
    repeat i ← i+1 until A[i] ≥ p
    repeat j ← j-1 until A[j] ≤ p
    swap(A[i], A[j])
until i ≥ j
swap(A[i], A[j])
swap(A[l], A[j])
return j
```

$C_{best}(n) = 2C_{best}(n/2) + n$ for $n>1$, $C_{best}(1)=0$

$a=2$ $b=2$ $d=1$ $C_{best} = n \log n$

$C_{worst}(n) = \underbrace{(n+1) + n + \dots + 3}_{f(n)} = \frac{(n-1)(n+2)}{2} - 3 \in \Theta(n^2)$

Randomly ordered array of size n.

Assuming that partition split can happen in each position s $(0 \le s \le n-1)$ with the same probability $1/n$

$C_{avg}(n) = \frac{1}{n} \sum_{s=0}^{n-1} \left[(n+1) + C_{avg}(s) + C_{avg}(n-1-s) \right]$ for $n>1$, $C_{avg}(0)=0$, $C_{avg}(1)=0$

$C_{avg}(n) \to 2n \ln n \approx 1{,}38\, n \log n$ → 38% more comparisons

Search Operations (Algorithms)

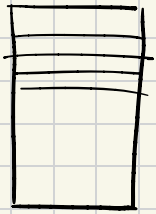
worst case

- Linear (Sequential) Search — O(n)
  - Unsorted dataset
  - Sorted dataset
- Binary search (Sorted dataset) — $O(\log_2 n)$
- Binary search tree — O(n)
- Hashing — O(1)

HASHING

Key
- Std Id
- tc id
- personal id
- unique id

→ [hash func] →



hash table

hash table operations: search, insert, delete

1) Easy to compute (minimal mathematical operations)
2) Consistency
3) uniform address generation

example | key mod m
↳ table length

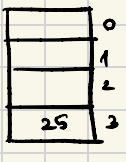
if m = 10 → key → 100, 40, 50 için aynı değerler

Most Common Hash Function

1) Division method

hash (key) = key mod m

m = 11
key = 25
} hash (25) = 3



Hashing disadvantage): Collision
↳ Birden fazla key'in aynı yeri işaret etmesi

2) Multiplication Method

$A = \frac{\sqrt{5}-1}{2} = 0,618033$

A: constant $\quad 0 < A < 1$

$hash(key) = \lfloor m * [(key * A) \bmod 1] \rfloor$

$= key * A \bmod 1 = key * A - \lfloor key * A \rfloor$

$m = 10000 \quad key = 123456$

$hash(key) = \lfloor 10000 * (\underbrace{123456 * 0,618033}_{76300,0041151}) \bmod 1 \rfloor$

$= 10000 * 0,0041151$

$= 41,1151$

$hash(123456) = 41 \Rightarrow 123456 \rightarrow$ [hf] $\rightarrow 41$

Birthday Paradoxi: Bir ortamdaki iki kişinin doğum günlerinin aynı olma olasılığı nedir?

$365 \rightarrow \frac{364}{365} \cdot \frac{363}{365} \cdots \frac{365-(k-1)}{365}$ → farklı olma olasılığı (k sayının çakışmama olasılığı)

Hash Table için

Table size = 365

$\frac{m-1}{m} \cdot \frac{m-2}{m} \cdots \frac{m-(k-1)}{m} \sim e^{\frac{-k(k-1)}{m}}$ } k sayının çakışmama olasılığı

Hash collusion probability: $1 - e^{\frac{-k(k-1)}{m}}$

$M = 2^{32}$ 32 bit hash values → after 250000 insert %100

→ after 77163 insert %50

Key

1. integer
2. floating number → convert to integer
3. string

'ELMA', 'MALE', 'ALME' should not generate the same address

a) $h \leftarrow 0$ (size of string)

for $i \leftarrow 0$ to $s-1$ do

$h \leftarrow (h * c + ord(c_i)) \bmod m$ (c: constant, m: table length)

b) Horner's method

$key = R^{L-1} * str[0] + \dots + R^0 * str[L-1]$

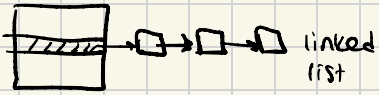
31 — length — $str[0] - 'A' + 1$

4. Compound Keys

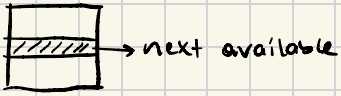
Birthdate: $31.08.1997 \rightarrow 31 \cdot R + 8 \cdot R^2 + 1997 \cdot R^3$

Solution for Collision Problem

1. Open Hashing (Seperate Chaining)



2. Closed Hashing (Open Addressing)



Open Addressing

a) Linear Probing

$hash(key, i) = [h'(key) + i] \bmod m$

$m = 7$

$key = 9$

$hash(9, 0) = 9 \% 7 = 2$

$key = 16$

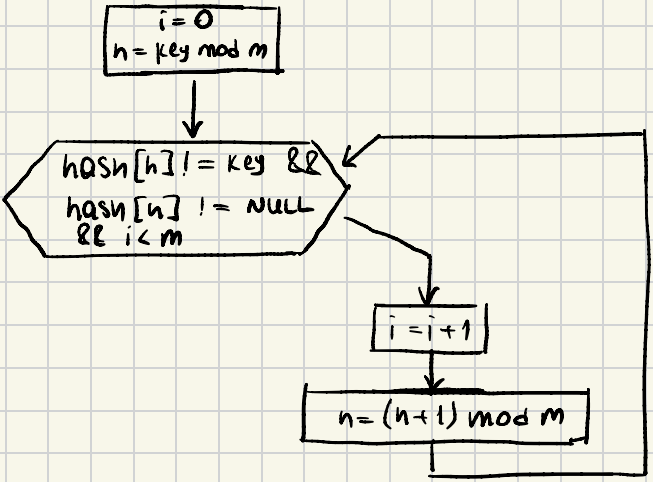
$hash(16, 0) = 2$

$hash(16, 1) = (16+1) \% 7 = 3$

0	23
1	8
2	9
3	16
4	11
5	12
6	13

If the table is almost full → REHASHING

Search Operation



Delete Operation

- -1 → delete flag
- Çok fazla silersek → redesign.

Problem : Clustering

each element positions itself after its expected place.

b) Quadratic Probing

$h(k,i) = (h'(k) + i^2) \bmod m$

$m = 7$

$hash(9,0) = 2$

$hash(9,1) = (2+1^2) \bmod 7 = 3$

$hash(9,2) = (2+2^2) \bmod 7 = 6$

0	
1	
2	2
3	3
4	
5	
6	9

ya da) $h(k,i) = (h'(k) + c_1 \cdot i + c_2 \cdot i)$

$i = 1 \quad c_1 = 2 \quad c_2 = 2$

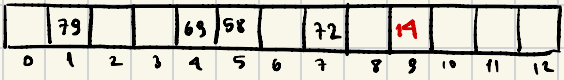
$hash(9,1) = (2 + 2 \cdot 1 + 1 \cdot 1^2) \bmod m = 5$

c) Double Hashing

$hash(key, i) = [h_1(key) + i \cdot h_2(key)] \bmod m$

$h_1(key) = key \bmod m$

$h_2(key) = 1 + (key \bmod K) \quad K < m$



$h_1(key) = key \bmod 13$

$h_2(key) = 1 + (key \bmod 11)$

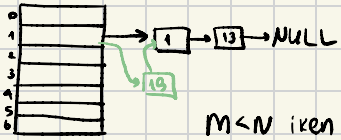
key = 14

$hash(14,0) = 14 \bmod 13 = 1$ dolu

$hash(14,1) = [(14 \bmod 13) + 1 \cdot (1 + (14 \bmod 11))] \bmod 13 = 5$ dolu

$hash(14,2) = [(14 \bmod 13) + 2 (1 + (14 \bmod 11))] \bmod 13 = 9$

Seperate Chaining



$m < N$ iken mantıklıdır.

key = 13 → 13 % 6 = 1

key = 1 → 1 % 6 = 1

key = 19 → 19 % 6 = 1

→ insert
→ delete
→ search

```
struct hash{
    struct node *head;
    int count;
}
```

```
struct node{
    int key;
    char name[40];
    struct node *next;
}
```

```
Struct nash * hashTable = NULL;
int main () {
   int key;
   char name [40];
   hashTable = (struct hash*) calloc (M, sizeof (struct hash));
   if (hashTable == NULL) {
      printf ("Yer acilamadi.");
      exit(0);
   }
   insertTo Hash (key, name);
   search In Hash (key);
   delete From Hash (key);
}

void insert To Hash (int key, char name [ ]) {
   int hashIndex = key % M;
   struct node * newNode = create Node (key, name);
   if (! hashTable [hashIndex]. head) {
      hash Table [hashIndex]. head = newNode;
      hash Table [hashIndex]. count = 1;
      return;
   }
   newNode → next = (hashTable [hashIndex]. head);
   hash Table [hashIndex]. head = newNode;
   hashTable [hashIndex]. count += 1;
   return;
}

Struct node * createNode (int key, char *name) {
   Struct node * newNode = (struct node*) malloc (size (Struct node));
   newNode → key = key;
   strcpy (newNode → name, name);
   newNode → next = Null;
   return newNode;
}
```

```
void searchInHash(int key){
  // linked listte aroma
}
```

search sorulabilir.

Universal Hashing

$h_1()$
$h_2()$
$h_3()$
$\vdots$
$h_k()$

} randomly choose ni a hash func.

$h_{a,b}(key) = [(a * k + b) \bmod p] \bmod m$

a, b, p → chosen randomly

p = 17 m = 6 a = 3 b = 4
↳ not random

key = 8 ⇒ $h(8) = ((3 * 8 + 4) \bmod 17) \bmod 6 = 5$

open addressing : $M > N$
seperate chaining : $M < N$

$\alpha = \frac{N}{M}$ load factor

$\alpha = 0,5$

α	# of (average probe)
0,1	1,11
0,2	1,28
0,3	1,52
0,4	1,89
0,5	2,5
0,6	3,62
0,7	
0,8	13,00
0,9	50,5

open addressing için ort. adım sayısı

unsuccessful search

p(first location is occupied) $= \alpha$
p(empty cell) $= 1-\alpha$
p(first two is occupied) $= \alpha \cdot \alpha$
p(probe terminates 2 steps) $= \alpha \cdot (1-\alpha)$
p(probe terminates k steps) $= \alpha^{k-1} \cdot (1-\alpha)$

average number of steps

$$\sum_{k=1}^{m} k \cdot \alpha^{k-1} \cdot (1-\alpha) \Rightarrow \text{geometric series} = \frac{1}{1-\alpha}$$

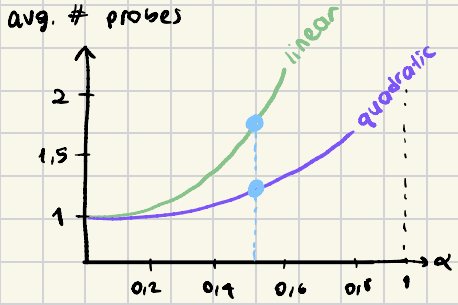
$S \approx \frac{1}{2}\left(1+\frac{1}{1-\alpha}\right)$ and $U = \frac{1}{2}\left(1+\frac{1}{(1-\alpha)^2}\right)$ } open addressing

α	S	U
50%	1,5	2,5
75%	2,5	8,5
90%	5,5	50,5

$S \approx 1 + \frac{\alpha}{2}$ $\quad U = \alpha$ } seperate chaining

$N = 2$, $m = 1$ $\Rightarrow \alpha = 2$

$N = 100$, $m = 25$ $\Rightarrow \alpha = 4$



α, 1'le sınırlı olduğundan open addressing yöntemlerinin grafiği.

Bir kesit alındığında successful search için linear probingde daha fazla deneme yapmak gerekir.

DYNAMIC PROGRAMMING

- Optimization Problems → graf en az renkle boyama problemi gibi
- Combinatoric Problems → n tane özellikten en iyi başarıyı sağlayan kombinasyon = $2^n - 1$

Dynamic programming is a technique for solving problems with overlapping subproblems. Typically, these subproblems arise from a reccurence relating a given problem's solution to solutions of its smaller subproblems.

Dynamic programming suggest solving each of the smaller subproblems only once and recording the results in a table from which a solution to the original problem can then be obtained.

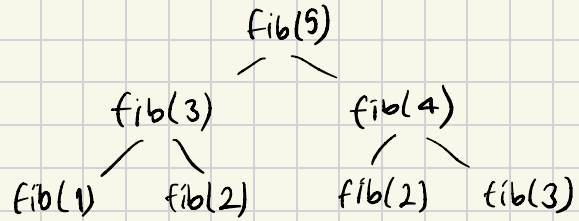
Principal of optimality → Bir problemin optimal sonucuna ulaşmak için küçük problemlerin optimal sonuçları bulunabilir.

Fibonacci Numbers

0, 1, 1, 2, 3, 5, 8, 13, 21, ...

$F(n) = F(n-1) + F(n-2)$ for $n > 1$ $\quad F(0) = 0 \quad F(1) = 1$

```
int fibonacci(int n) {
    if(n==0)
        return 0
    if(n==1)
        return 1
    return fibonacci(n-1) + fibonacci(n-2)
```



0	1	2	3	4	5	6	7	8
0	1	1	2	3	5	8	13	21

space complexity

Coin-row Problem

$c_1, c_2, c_3, \ldots, c_n$ — 5, 1, 2, 10, 6, 2

5, 2, 6

yan yana olan coinler alınmaz — 1, 10, 2

5, 10, 2

index	∅	1	2	3	4	5	6
coins ← C		5	1	2	10	6	2
F	∅	5	5	7	15	15	17

elde edilebilecek max cost

5 + 2 + 6 = 13 oldu, 15 daha büyük

$$F(n) = \max\{(c_n + c_{n-2}), c_{n-1}\}$$
$$= \max\{c_n + F(n-2), F(n-1)\} \quad \text{for } n > 1$$

$F(\emptyset) = 0 \quad F_1 = c_1$

$F(0) = 0, \quad F_1 = 5$

$F(2) = \max\{1+\emptyset, 5\} = 5$

$F(3) = \max\{2+5, 5\} = 7$

$F(4) = \max\{10+5, 7\} = 15$

$F(5) = \max\{6+7, 15\} = 15$

$F(6) = \max\{2+15, 15\} = 17$

hangi coinleri aldık?

index	∅	1	2	3	4	5	6
C		5	1	2	10	6	2
F	∅	5	5	7	15	15	17

değişmemiş 1'i aldık — değişmiş 4'ü aldık — değişmiş 6'yı aldık

time complexity $\Theta(n)$

space complexity $\Theta(n)$

Knapsack Problem

W = weight
n = number of items
} Brute force ile $2^n - 1$

W = 5

item	weight	value
1	2	10
2	1	20
3	3	15
4	4	50
5	5	90

1, 2 → 30
2, 3 → 35
1, 3 → 25
2, 4 → 70
5 → 90

W = 5

item	weight	value
1	2	12
2	1	10
3	3	20
4	2	15

weight →, item ↓

	0	1	2	3	4	5
0	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	37

max

Maximize Σv_i Subject to W

- case 1: take item i
  - $P(i, w) = P(i-1, w-w_i) + v_i$
- case 2: not take item i
  - $P(i, w) = P(i-1, w)$

$P(3,3) = P(2,1) + v_3$

$P(1,3) = \max\{P(0,1) + 12, P(0,3)\}$

$P(0,1) + 12 = 0 + 12$, $P(0,3) = 0$

$= 12$

$$P[i,w] = \begin{cases} P[i-1, w] & , \text{if } w_i > w \\ \max\begin{cases} P[i-1, w-w_i] + v_i \\ P[i-1, w] \end{cases} & , \text{other} \end{cases}$$

initialization
$P[0, w] = 0$
$P[i, 0] = 0$

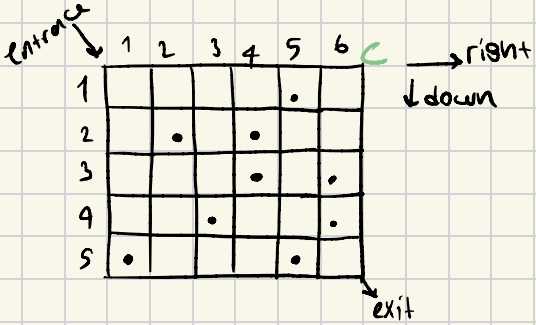
	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	12	12	12
2	0	10	12	22	22	22
3	0	10	12	22	30	32
4	0	10	15	25	30	37

↑ not taken: $P[i,j] = P[i-1, j]$

↖ taken: $P[i-1, w-w_i]$ $= P[3,3]$

$4 \to 3$ $5-2 \to 3$

Robot Coin Collection Problem



C

	1	2	3	4	5	6
1	0	0	0	0	1	1
2	0	1	1	2	2	2
3	0	1	1	3	3	4
4	0	1	2	3	3	5
5	1	1	2	3	4	5

F

Time complexity = $\Theta(nm)$

Space complexity = $\Theta(nm)$

Reconstruction of Optimal Solution

```
F[1,1] ← C[1,1]
for j ← 2 to m do
    F[1,j] ← F[1,j-1] + C[1,j]
for i ← 2 to n do
    F[i,1] ← F[i-1,1] + C[i,1]
    for j ← 2 to m do
        F[i,j] ← max(F[i-1,j], F[i,j-1]) + C[i,j]
return F[n,m]
```

EDIT DISTANCE (Levenshtein Algorithm)

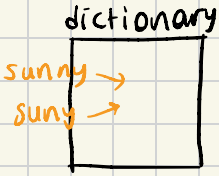
The Levenshtein distance is a string metric for measuring difference between two sequences.

① sunny → sumny
② sunny → suny
③ sunny → sunnyy

- substition
- replacement

- deletion

- insertion



dist(MATHS, ARTS)

M	A	T	H	S
_	A	R	T	S
↑	x	↑	↑	x

insert, change

distance = 3

M	A	T	H	S
A	R	T	_	S
↑	↑	x	↑	x

change, insert

distance = 3

String 1: $X[1 \dots m]$
String 2: $Y[1 \dots n]$

① copy $X[i] = Y[j]$

② change $X[i] \neq Y[j]$

③ insert $X[i] \downarrow -$

MATHS
ART_S

④ delete $- \downarrow Y[j]$

MA_S
ARTS
↑ ↑ ↑ x

		A	R	T	S
	0	1	2	3	4
M	1	1	2	3	4
A	2	1	2	3	4
T	3	2	2	2	3
H	4	3	3	3	3
S	5	4	4	4	3

case 1: copy
$ED[i,j] = ED[i-1, j-1]$
case 2: change
$ED[i,j] = ED[i-1, j-1] + cost$
case 3: insert
$ED[i,j] = ED[i-1, j] + cost$
case 4: delete
$ED[i,j] = ED[i, j-1] + cost$

eşitlik olduğunca:
min {sağ, sol}
eşitlik yoksa:
çapraz + yeni'ye de bak

Recurrence Relation

$$ED[i,j] = \begin{cases} ED[i-1, j-1] + \emptyset & , \text{if } (x[i] = y[j]) \\ \min \begin{cases} ED[i-1, j-1] \\ ED[i-1, j] \\ ED[i, j-1] \end{cases} & , \text{otherwise} \end{cases}$$

initialize: $ED[i, \emptyset] = i$ insert
$ED[\emptyset, j] = j$ delete

	_	H	Y	U	N	D	A	I
_	0	1	2	3	4	5	6	7
H	1	0	1	2	3	4	5	6
O	2	1	1	2	3	4	5	6
N	3	2	2	2	2	3	4	5
D	4	3	3	3	3	2	3	4
A	5	4	4	4	4	3	2	3

dist(HONDA, HYUNDAI) = 3

H O _ N D A _
H Y U N D A I

LONGEST COMMON SUBSEQUENCE

X: ATCTGAT
Y: TGCATA

longest common subsequence of X and Y
LCS(X, Y) = TCTA
LCS(X, Y) = TGAT

KO2A1AK → KAI, KOI, KO1A, KAIK
(subsequences)

$X = \langle x_1, x_2 \dots, x_m \rangle$
$Y = \langle y_1, y_2 \dots, y_n \rangle$
} LCS

$S = \langle s_1, s_2, \dots, s_k \rangle$
$s_{i_1}, s_{i_2}, \dots s_{i_k}$
$i_1 < i_2 < \dots < i_k$

X: <BACDB>
Y: <BDCB>

Y →
X ↓
$X_i, Y_j \rightarrow C[i,j]$
• X_m, Y_m

		B	D	C	B
	0	0	0	0	0
B	0	1	1	1	1
A	0	1	1	1	1
C	0	1	1	2	2
D	0	1	2	2	2
B	0	1	2	2	3

* esteime vor, coprin

LCS(X, Y) = 3
(BCB), (BDB)

if $i = \emptyset \rightarrow C[\emptyset, j] = \emptyset$
if $j = \emptyset \rightarrow C[i, \emptyset] = \emptyset$

$$C[i,j] = \begin{cases} \emptyset & \text{, if } i=\emptyset \text{ or } j=\emptyset \\ C[i-1, j-1] + 1 & \text{, } i,j > 0 \text{, } x_i = y_j \\ \max\{C[i-1,j], C[i,j-1]\} & \text{, } x_i \neq y_j \end{cases}$$

- ADD
- SKIP X
- SKIP Y

$x_i = y_j \Rightarrow B[i,j] = \text{ADD} \rightarrow \emptyset$
$x_i \neq y_j \Rightarrow C[i-1, j] >= C[i, j-1]$ then
X[i] not in LCS
B[i,j] = SKIPX
else
B[i,j] = SKIPY
Y[j] not in LCS

```
while ( (i != 0) && (j != 0) ) {
  switch (B[i][j]) {
    case ADD: {
      add x[i] to front of LCS
      i--;
      j--;
      break;
    }
    case SKIPX: {
      i--;
      break;
    }
    case SKIPY: {
      j--;
      break;
    }
  }
}
```

GRAPHS

- Binary Search Tree
- Shortest Path algorithm

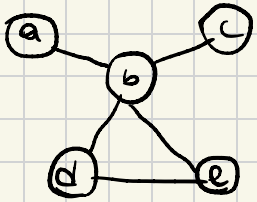
Graph $G = \langle V, E \rangle$

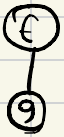
1. Directed (Digraph)
2. Undirected

Adjacency Linked List → kenar sayısı azsa kullanılır.
Adjacency matrix → düğüm sayısı kenar sayısına yakınsa.

Dense	Sparse
matrix	Adjacency linked list

Complete Graph: Tüm düğümlerin bir kenar bağlantısı vardır.
Weight Graph: Kenarlar değerlere sahip.

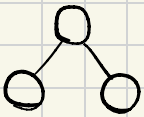


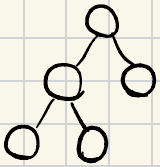


2 connected component.

a'ya giden gidilemez.

Connectivity: Her pair için bir pairden diğer paire yol olmalı.





} forest
(random forest → karar algoritması)

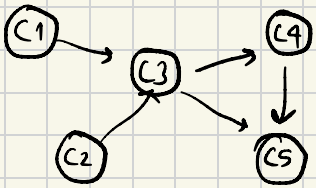
Depth-first-search → stack
Breath-first-search → queue

```
Algorithm DFS(G)
  mark each vertex in V with ø as a mark of being 'unvisited'
  count ← ø
  for each vertex v in V do
    if v is marked with ø
      dfs(v)

- - - - - - - -

  dfs(v)
    count ← count+1; mark v with count
    for each vertex w in V adjacent to v do
      if w is marked with ø
        dfs(w)
```

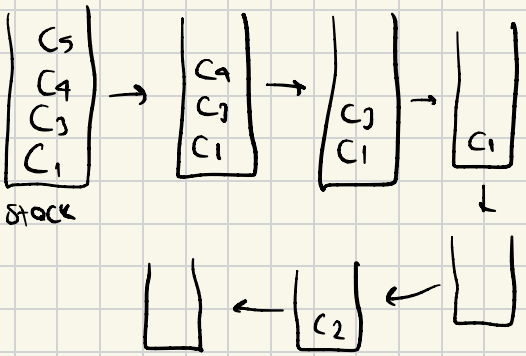
```
Algorithm BFS(G)
  mark each vertex in V with ∅ as a mark of being 'unvisited'
  count ← ∅
  for each vertex v in V do
    if v is marked with ∅
      bfs(v)


  bfs(v)
    count ← count + 1 ; mark v with count and initialize a queue with v
    while the queue is not empty do
      for each vertex w in V adjacent to the front vertex do
        if w is marked with ∅
          count ← count + 1 ; mark w with count
          add w to the queue
      remove the front vertex from the queue
```

	DFS	BFS
data structure	stack	queue
# of vertex ordering	two ordering	one ordering
edge types	tree and back edges	tree and cross edge
efficiency for adjacency matrix	$\Theta(\|V^2\|)$	$\Theta(\|V^2\|)$
efficiency for adjacency list	$\Theta(\|V\| + \|E\|)$	$\Theta(\|V\| + \|E\|)$

TOPOLOGICAL SORTING

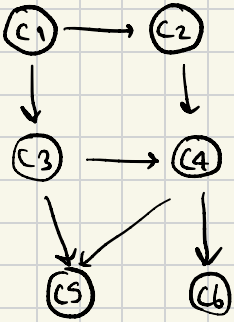


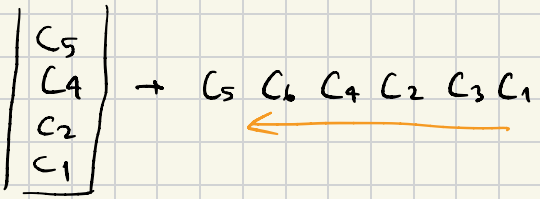
acyclic



popping off order

C5 C2
C4
C3
C1

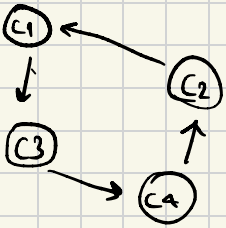




Decrease And Conquer

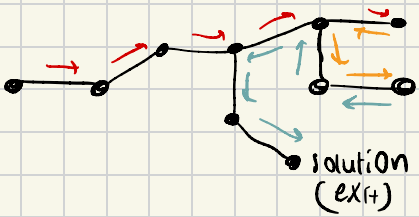
C_1 → başlangıç, kendisine giren ok yok.

C_1 alınıp silinir.
C_2 alınıp silinir.
C_3 alınıp silinir.
C_4 alınıp silinir.
C_5 alınıp silinir
C_6 alınıp silinir.



Başlangıç yok. Decrease and conquer nasıl uygulanır? —

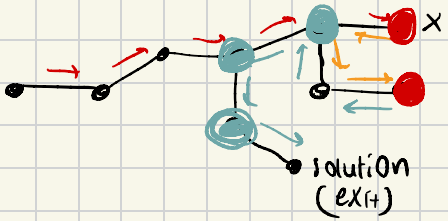
BACKTRACKING



★ maze

DFS with pruning → ağacın belirli opsiyonlarının kaldırılması
↳ exhaustive 'i iyileştirir

State space tree
→ promising ●
→ non promising ●



Subset Sum Problem

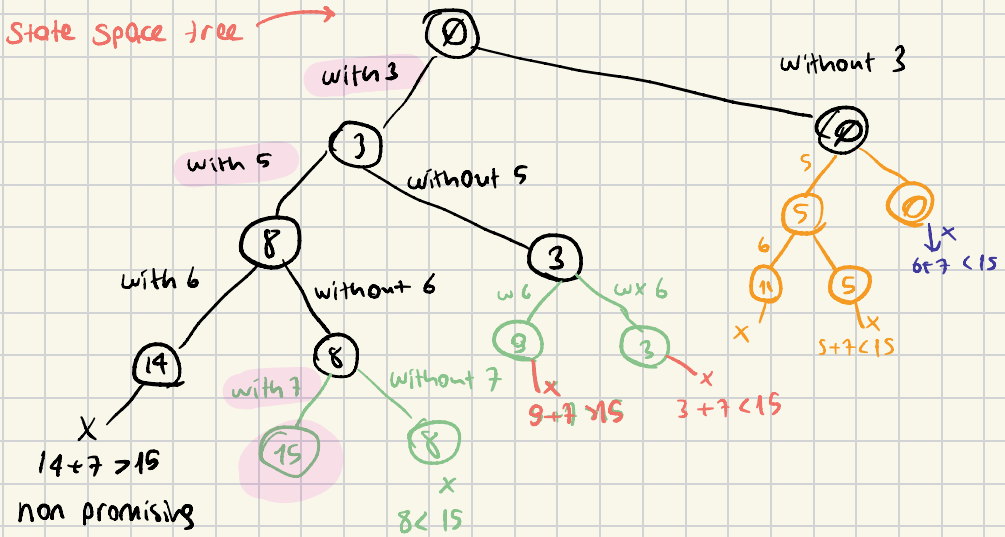
$A = \{a_1, \ldots, a_n\}$ n positive integer → toplamları d yapan sayılar

$A = \{1, 2, 5, 6, 8\}$ $d = 9$ $\Rightarrow$ $\{1, 2, 6\}$, $\{1, 8\}$

$S = \{s_1, s_2 \ldots s_n\}$

$S = \{3, 5, 6, 7\}$ $d = 15$

$s_1 \leq s_2 \leq s_3 \ldots$ olmalı (sıralı değilse sırala)



If s is not equal to d, we can terminate the node as non promising if either of the following two inequalities holds

$s + q_{i+1} > d$ (the sums is too large)

$s + \sum_{j=i+1}^{n} q_j < d$ (the sums is too small)

n - queen problem

8-queen problem

1) Brute force ile $\binom{64}{8} = \frac{64!}{8!\,56!} = 4,426,165,368$

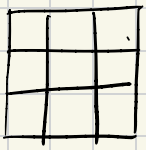
2) one per row $n^n = 8^8 = 16,777,216$

3) one per row, one per column $n! = 8! = 40,320$

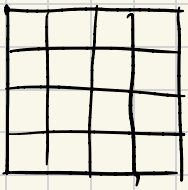
4) Backtracking 2057 steps

1x1 vezirler yerleştirilemez

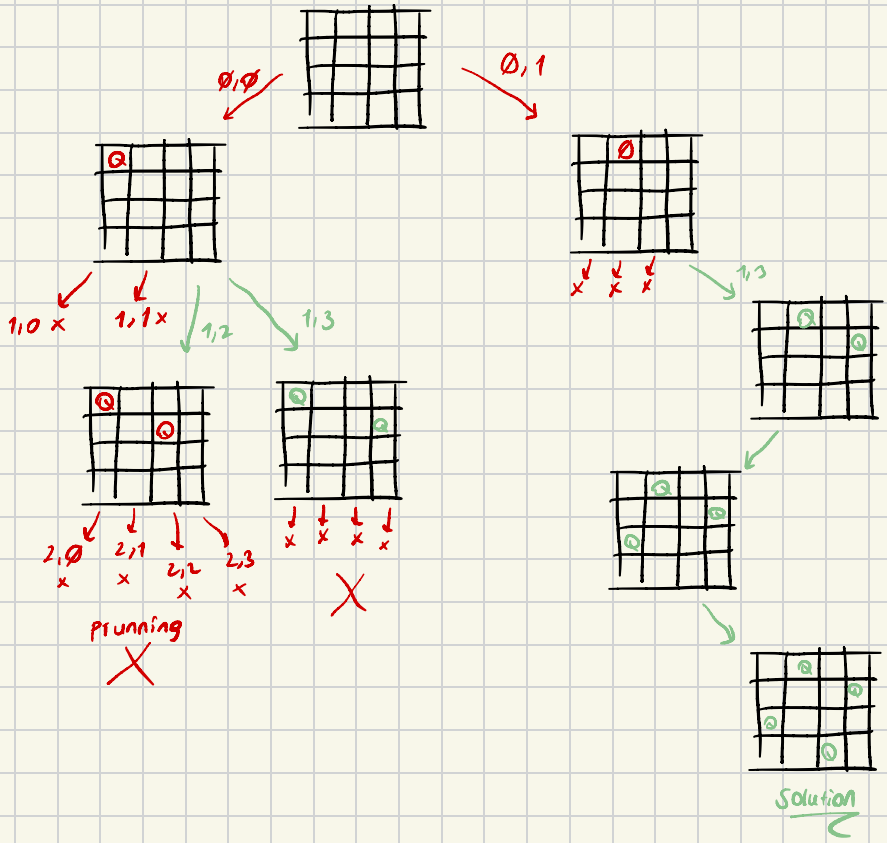
2x2 vezirler yerleştirilemez



3x3 vezirler yerleştirilemez



4x4 min bu olmalı



col Ø
row Ø
0,0

col 1
row 1
1,1

col 2
row 2
2,2

row Ø
col Ø
0,2

row 1
col 1
1,1

row 2
col 2
2,0

✗ $|col2 - col1| = row2 - row1$

✓ $|col_i - col_j| = |row_i - row_j|$

1	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

→ queenlerin yerini matris değil dizi ile saklarız.

0	1	2	3
2	3	Ø	Ø

Ø. satırda 2. sütunda bir queen var.
→ indisler ⇒ satır
→ dizi değeri ⇒ sütun

Algorithm Backtracking(X[1...i])
  if X[1...i] is a solution write X[1..i]
  else
    for each element $x \in S_{i+1}$ consistent with X[1..i] and the constraints do
      X[i+1] ← x
      Backtrack (X[1.. i+1])

$1 + c_1 + c_1c_2 + \dots + c_1c_2..c_n$

Branch and Bound

Optimization problem
- ↳ maximization
- ↳ minimization
- ↳ objective function

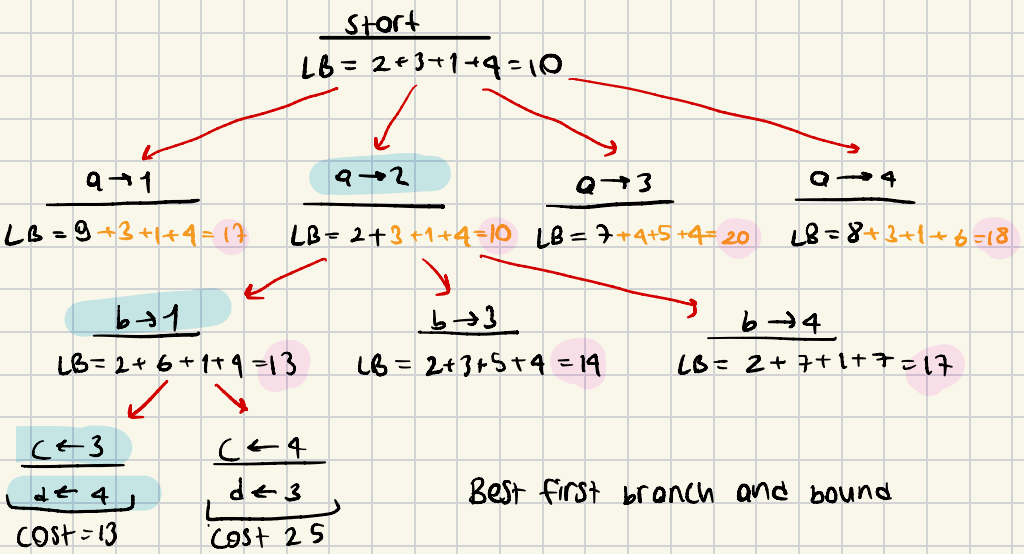
under certain constraints

→ Feasible solution (her bir constraint'i karşılar)
→ Optimal solution (feasible'lar arasında en iyisi)

→ Assignment Problem

	Job1	Job2	Job3	Job4	
C =	9	2	7	8	person a
	6	4	3	7	person b
	5	8	1	8	person c
	7	6	9	4	person d

En iyi kombinasyonda 2+3+1+4 = 10 birim maliyet ama Job1 boş



Best first branch and bound

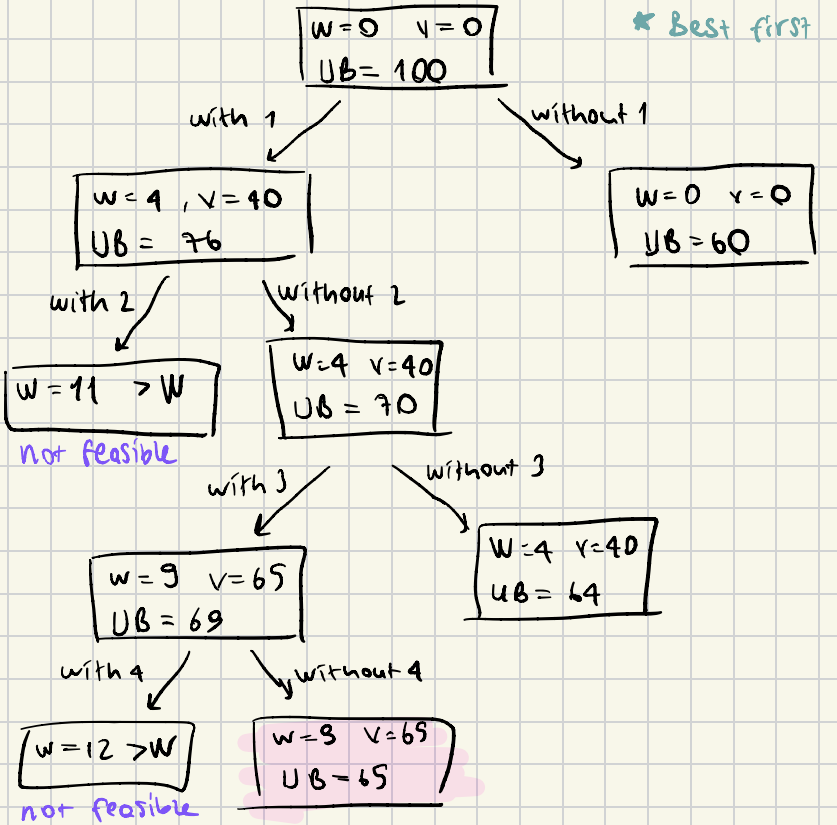
→ Knapsack Problem

W = 10

item	weight	value	value/weight
1	4	40	10
2	7	42	6
3	5	25	5
4	3	12	4

$\frac{v_1}{w_1} \geq \frac{v_2}{w_2} \geq \cdots \geq \frac{v_n}{w_n}$

* Best first branch and bound



→ Hungarian Method

P, NP, NP COMPLETE, NP HARD

Definition: We say that an algorithm solves a problem in polynomial time if its worst-case time efficiency belongs to $O(p(n))$ where $p(n)$ is a polynomial of the problem's input size n.

Polynomial → tracktable
not solved in polynomial time → intractable

Definition: Class P is a class of decision problems that can be solved in polynomial time by (deterministic) algorithm. This class of problems is called polynomial.

undecidable → Halting Problem (→ NP hard) decidable

A: algorithm P: program I: input

$$A(P,I) = \begin{cases} 1, & \text{if program } P \text{ halts on input } I \\ \emptyset, & \text{if program } P \text{ does not halt on input } I. \end{cases}$$

$$Q(P) = \begin{cases} \text{halts} & \text{, if } A(P,P) = \emptyset \text{ i.e if program } P \text{ does not halt on input } P \\ \text{does not halt,} & \text{if } A(P,P) = 1 \text{, i.e if program } P \text{ halts on input } P \end{cases}$$

$$Q(Q) = \begin{cases} \text{halts} & \text{, if } A(Q,Q) = \emptyset \text{ i.e if program } Q \text{ does not halt on input } Q \\ \text{does not halt,} & \text{if } A(Q,Q) = 1 \text{, i.e if program } Q \text{ halts on input } Q \end{cases}$$

Definition: A nondeterministic algorithm is a two stage procedure that takes as its input an instance I of a decision problem and does the following:

Non-deterministic ("guessing") stage: An arbitrary string S is generated that can be thought of as a candidate solution to the given instance I (but may be complete gibberish as well)

Deterministic ("verification") stage: A deterministic algorithm takes both I and S as its input and outputs "yes" if S represents a solution to instance I.

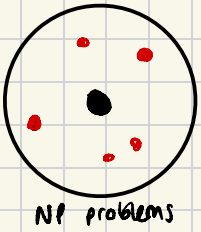
Definition: Class NP is the class of decision problems that can be solved by nondeterministic polynomial algorithms. This class of problems is called nondeterministic polynomial.

→ NP complete

Definition: A decision problem D_1 is said to be polynomially reducible to a decision problem D_2, if there exists a function t that transforms instances of D1 to instances of D_2 such that:

1. t maps all 'yes' instances of D_1 to 'yes' instances of D_2 and all 'no' instances D_1 to 'no' instances of D_2
2. t is computable by a polynomial time algorithm.

Definition: A decision problem D is said to be NP-Complete if
1. it belongs to class NP
2. every problem in NP is polynomially reducible to D.



Tüm kırmızı problemler bizim problemimize indirgenebiliyorsa → np complete
örn: Hamilton circuit ← traveling salesmen
(hamilton'a indirgenir)