BLM3021 - Algorithm Analysis

"The Algorithm Design Manual"

"An Introduction to the Analysis of Algorithms"

[Sylabbus]

Vize	~~%40~~	* 2 Vize	%36
Lab	%10	* 3 Lab (2+1)	3 Uygulama
Ödev	%10	* 2 Ödev	%4 (2+1)
Final	%40		

Midterm I (17.11.2025)
Midterm II (15.12.2025)

FO
→ 40 ve üzeri alamazsan ders başarısız sayılacak

1. Ders

- Algoritma nedir? vs. vs...

How good is the algorithm?
- Correctness
- Efficiency (Time, Space)
- Simplicity
- Generality

Does there exist a better algorithm?
- Lower bounds
- Optimality

Reason to Analyze an Algorithm → Predict Performance
→ Compare Algorithms
→ Avoid Performance Bugs

System Independent

Algorithms

System Dependent

Hardware

...

OS

Emprical Analysis T(N)

Mathematical Models for Running Time

⇒ sum of (cost * frequency for all operations)

Asymptotic Notations

Big-O, Big-Teta, Big Omega $(O(N), \Omega(N^2), \Theta(N))$

Upper Bound — Lower Bound — Tight Bound

Amortized Analysis

A Large cost of operation is spread out over many operations (amortized)

3 Common Methods

- Aggregate Method
- Accounting Method
- Potential Method

Usage Examples

- Dynamic Arrays
- Binary Counter
- Stack Operations (Multipop)
- Trees (Red-Black Tree)

....

Amortized Analysis

Find Max

Best Case	n	$\Theta(n)$ tight bound
Average Case	n	
Worst Case	n	

```
int sequentialSearch (int n, int a[], int x)
{
    int i=0;
    while ((i<n) && (a[i]!=x))
    {
        i++;
    }
    if (i<n)
        return i;
    else
        return -1;
}
```

Best Case: $O(1)$, Worst Case: $O(n)$

Average Case: → probability

$0 \le p \le 1$ ith position, $\frac{p}{n}$

$$C_{avg} = \underbrace{\left[1*\frac{p}{n} + 2*\frac{p}{n} + \dots + n*\frac{p}{n}\right]}_{\text{successful search}} + \underbrace{n*(1-p)}_{\text{not in the array!}}$$

successful search → $p=1$ $\frac{1}{n}(1+2+\dots+n) = \frac{n+1}{2}$ → $O(n)$

$$= \frac{p}{n}\left[1+2+\dots+n\right] + n*(1-p)$$

unsuccessful search $p=0$

$C_{avg} = n*(1-0)$ → $O(n)$

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

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

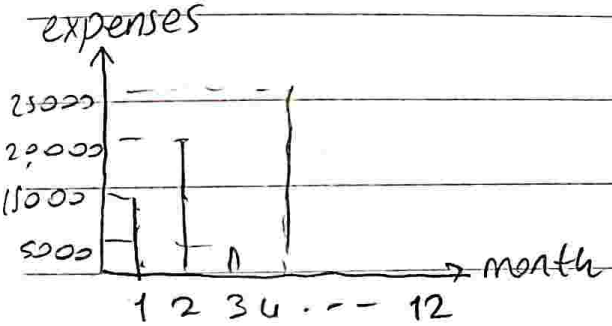
$$\sum_{i=1}^{n} i^2 = 1^2+\dots+n^2 = \frac{n(n+1)(2n+1)}{6} = \frac{n^3}{3}$$

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

Amortized Analysis ≠ Average Case Analysis

↳ worst case analysis

(1) The Aggregate Method, Brute Force



$$\frac{1}{12} * \sum_{i=1}^{12} \text{expense}[i]$$

k-bit binary counter:

8bit

	7	6	5	4	3	2	1	0	cost	total cost
0	0	0	0	0	0	0	0	0	0	0
1					0	0	0	1	1	1
2					0	0	1	0	2	3
3					0	0	1	1	1	4
4					0	1	0	0	3	7
5					0	1	0	1	1	8
6										
7										
⋮										
16										

for n increments:

bit 0 flips n times

bit 1 flips n/2 times

" 2 " n/4 "

" J " $n/2^J$ "

total flipped bits

$$\sum_{j=0}^{k-1} \left\lceil \frac{n}{2^j} \right\rceil \rightarrow n\left(1+\frac{1}{2}+\dots\frac{1}{2^{\log n}}\right)$$

$\underbrace{n + \frac{n}{2} + \cdots + \frac{n}{2^J}}_{F(n)} \leq c * (g(n))$

$F(n) \leq 2 * n \Rightarrow O(n)$

Amortized Analysis ile k-bit counter karmaşıklığı $O(n)$ bulundu.

(per increment: $\frac{O(n)}{n} = O(1)$)

WORST CASE hesabı ile:

0111
1000

k: bit sayısı ⇒ worst case = $O(k)$ ↷ per Increment

$k = \log_2 n$

↳ 1 increment

n defa değişim: $O(n * \log_2 n)$

Böyle değişimli algoritmalarda amortize analiz daha gerçekçi bir sonuç vermektedir.

Kullanım Alanları

Augment Stack

push (S, x) $O(1)$

pop (S) $O(1)$

multiPop (S, k): pops the min (k, |S|)

↳ if $k \leq S \rightarrow k$ $O(n)$ (worst case)

otherwise |S|

max num. of pops: n (stack size)

↳ $O(n)$

Amortized cost per operation:

# of pops OR # of multiple pops: ≤ # of pushes

≤ $O(n)$

Amortized Analysis $\frac{O(n)}{n} = O(1)$

Aggregate Method

1) Identify all operations (total of n)
2) Determine the actual cost of each operation
3) Sum all

Accounting Method

Assign amortized cost to each operation

Each increment: 1 coin

$0 \to 1$: 1 coin

$1 \to 0$: pay 1 coin

1) At most one $0 \to 1$ per increment

n increment: $\#(0 \to 1) : n$

2) Every $(1 \to 0)$ is paired with previous $(0 \to 1)$

$\#(1 \to 0) \leq \#(0 \to 1)$

3) # flips: $\#(0 \to 1) + \#(1 \to 0) \leq \underbrace{\#(0 \to 1)}_{n} + \underbrace{\#(0 \to 1)}_{n} \leq 2n$

Total Cost: $2n$ Amortized Cost: $\frac{n}{n} = O(1)$

Binary	Bits Flipped	Cost
000	0	0
001	1	1 gain
010	2	1 gain
~~011~~	1	2 gain
100	3 $0 \to 1 : 1$ $1 \to 0 : 2$	1 gain

+liç negatife düşmez!

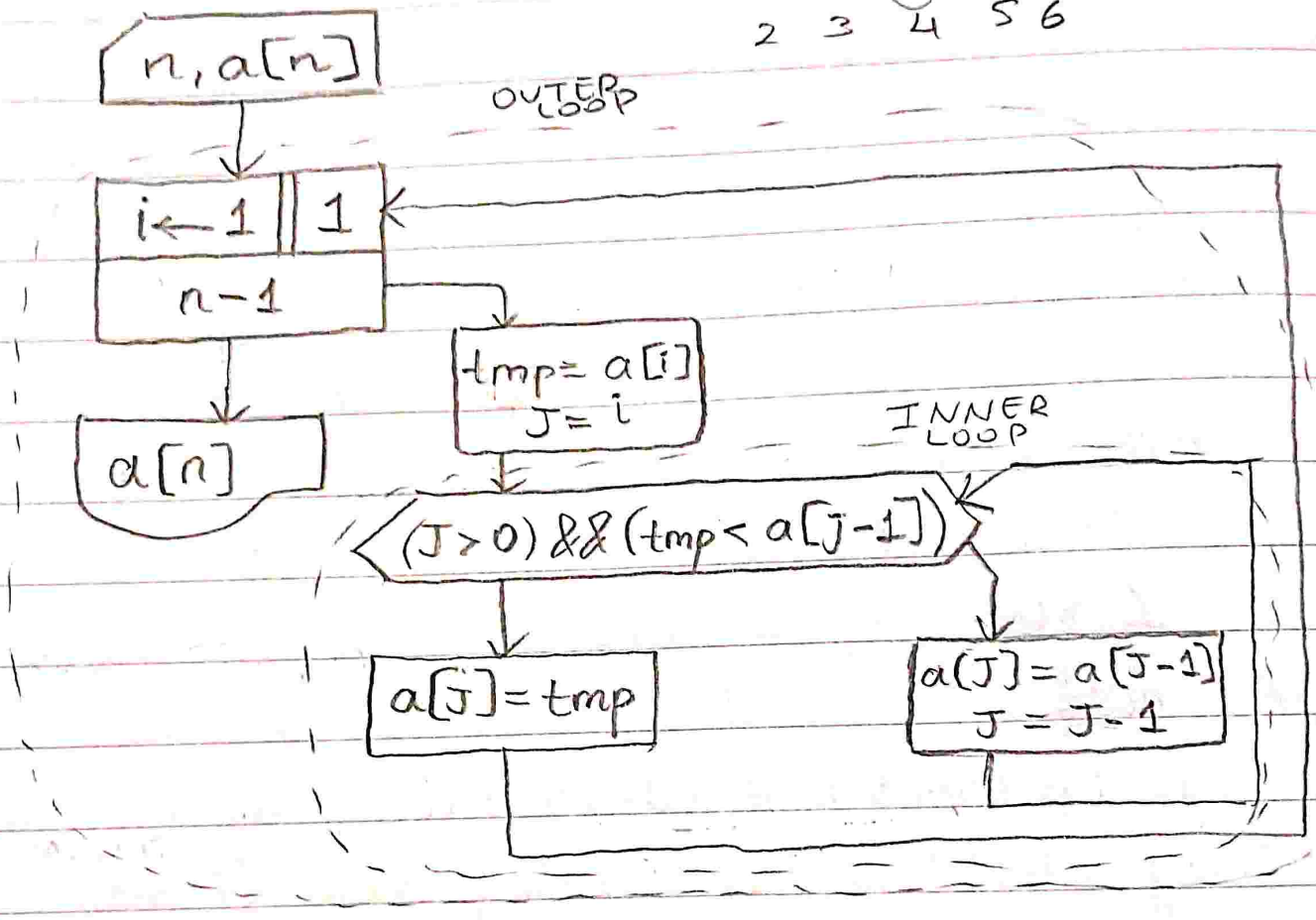
after n increment actual cost ≤ 2

Analysis of Recursive and Non-Recursive Algorithms

(Analysis of)
non-recursive algorithms

① Insertion Sort

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



⇒ Worst Case Büyük ⟹ Küçük
9 7 5 4 2

$$\sum_{i=1}^{n-1} i = \frac{n(n+1)}{2} = \frac{n^2}{2} \in O(n^2), \quad C_{worst} \in \Theta(n^2)$$ ✓✓

↳ tight bound

⇒ Best Case Küçük ⟹ Büyük
1 2 3 4 5

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

$$C_{best} \in O(n), \quad C_{best} \in \Theta(n)$$

⇒ Average Case

Expected cost of <u>random</u> input

Outer loop is fine, hard part is inner loop because WHILE

inner loop

3 5 9 11 7

k

⇒ element is at the k^{th} position
it winds up at position J

J

Final position: J

(K − J + 1 comparison)

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

(K − J + 1 ← # of comparison)

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

$= \frac{K+1}{2}$ → inner loop

outer loop

$\sum_{K=1}^{N-1}\left(\frac{K+2}{2}\right) = \frac{1}{2}\left(\frac{(N+1)(N+2)}{2}\right) = \frac{N^2+3N+2}{4}$

Coverage $\in O(N^2)$

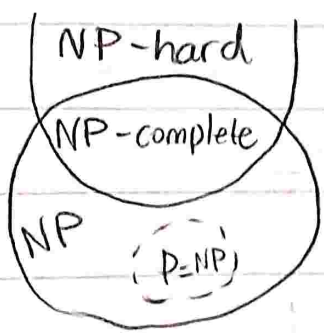
$\in \Theta(N^2)$

$C_{\text{insertion sort}} \in O(n^2)$

~~$\in \Theta(n^2)$~~

$\in \Omega(n)$ (Best Case)

(MEK BURAYI İŞLEMEDİ!)



(son hafta işledi)

Problem Sınıfı

P (polynomial time)

NP (non-det. poly. time)

NP-hard (atleast as hard as the hardest NP problem)

NP-complete (both NP and NP-hard, polynomially reducable to D

Bu sınıflandırmalar problemin çözülebilirliği ve çözme süresi ile ilişkilidir.

(Analysis of)
recursive functions

Recurrence Relation:

$$f(n) = \begin{cases} 1 & n=0 \\ n*f(n-1) & n>0 \end{cases}$$ (faktöriyel algoritması)

$M(n) = M(n-1) + 1$ ← 1. adımda 1 çarpma var!!!

$M(0) = 0 \quad n=0$

ÇÖZÜM YOLLARI

① Backward Substitution

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

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

$= \dots$

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

$n-i=0 \rightarrow n=i \rightarrow M(n) = \underbrace{M(0)}_{0} + n$

$M(n) = n$ ⟵ $M(n) \in O(n)$

$\in \Theta(n)$

$\in \Omega(n)$

② Forward Substitution

n	$m(n)$
0	0
1	1
2	2
⋮	
n	$m(n-1)+1 = n$

③ Master Theorem

Running Time (General Divide and Conquer Recurrence)

$T(n): a * T(n/b) + f(n)$

a: # of subproblems

b: size of each subproblem

A[p.....r]

```
void mergeSort (int a[], int p, int r)
{
    if (p<r)
    {
        q=(p+r)/2;
        mergeSort (A, p, q);
        mergeSort (A, q+1, r);
        merge (A, p, q, r);
    }
}
```

$$T(n) = \begin{cases} 1 & , \text{if } n=1 \\ 2*T(n/2)+n & , \text{if } n>1 \end{cases}$$

Merge sort → $2*T(n/2)$; merge → n

MASTER THEOREM $\quad T(n) = a * T(n/b) + f(n)$

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

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

(O veya Ω de olur)

Can't be used: $T(n)=\sin n$, $F(n)=2^n$, $a=2*n$ ...

$$T(n)_{\text{merge sort}} = \begin{cases} 1 & , n=1 \\ 2T(n/2)+n & , n>1 \end{cases}$$

Backwards Substitution

$T(n) = 2 * T(n/2) + n$

$T(n) = 2^* [2^* + (n/4) + n/2] + n \quad = 4T(n/4) + 2n$

$T(n) = 2^* [2^* [2T(n/8) + n/4] + n/2] + n \quad = 8T(n/8) + 3n$

$8T(n/8) + 3n \quad n = 2^i \quad i = \log_2 n$

$2^i, 2^i, 2^i, \ldots, 2^i$

$\hookrightarrow n * \underbrace{T(1)}_{1} + \underbrace{\log_2 n * n}$

$2^i * T(2^i/2^i) + i * 2^i$ $\qquad T(n) \in O(n \log n)$

Master Theorem

$a=2, b=2, d=1 \qquad T(n) \in \Theta(n^1 * \log n)$

python
dynamic array
amortized analysis

list.append
list.pop
list.insert

→ haftaya araştır.

Divide and Conquer

Quick sort, Merge sort

$O(n^2) \rightarrow O(n \log n)$ sort algorithms

(1) Divide (2) Conquer (3) Combine

(3) 5 4 2 1 8

2 1 (3) 5 4 8

Decrease and Conquer

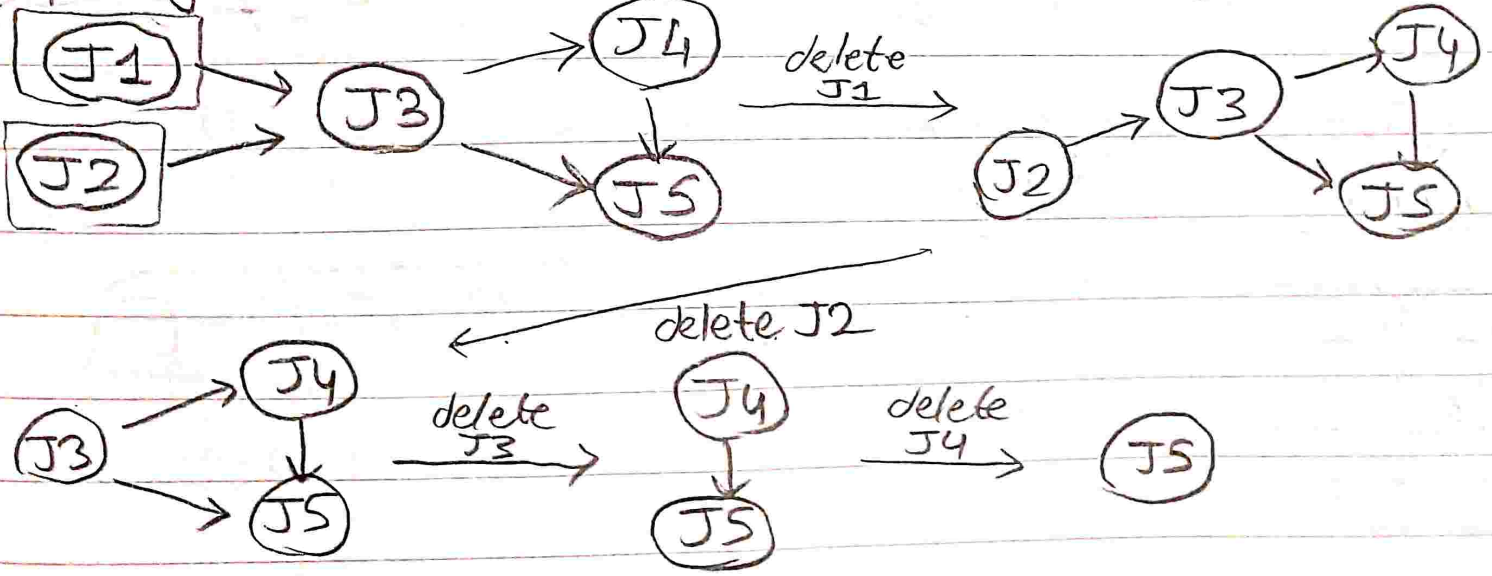
Binary Search, Insertion Sort, Topological Sort

3 yolu var

(1) Decrease by a constant (usually by one) Topological Sort, Insertion Sort

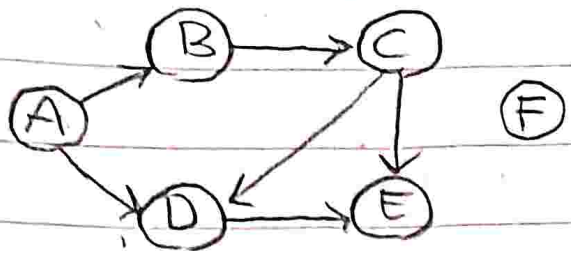
$f(n) = a^n = a^{n-1} * a$

Topological Sort (Graph must be Directed Acyclic Graph)



J1 - J2 - J3 - J4 - J5

J2 - J1 - J3 - J4 - J5



Indegree

0 A → [B] → [D] → NULL

1 0 B → [C] → NULL

1 0 C → [D] → [E] → NULL

2 1 0 D → [E] → NULL

2 1 0 E → NULL

0 F → NULL

(Bir listeden elemana ulaşılabiliyorsa Indegree'sine +1 eklenir)

Topological Sort

Her seferinde baştan 0 olan elemanı seç ve listeden sıradaki elemanların Indegree değerini 1 azalt

A → B → D ⇒ B-1 → 0 , D-1 → 1 Sırada B (0)

B → C ⇒ C-1 → 0 Sırada C

C → D → E ⇒ D-1 → 0 , E-1 → 1 Sırada D

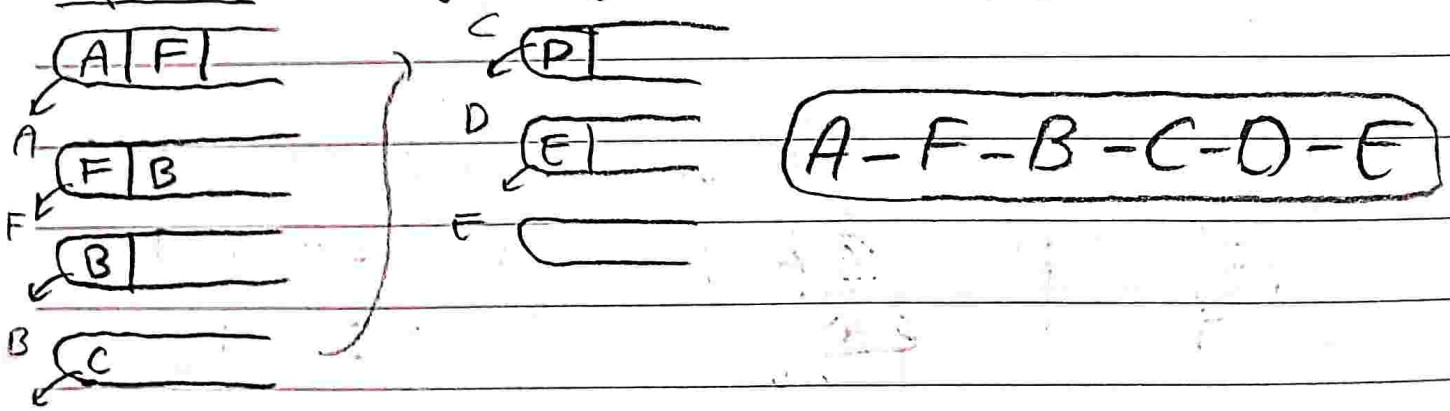
D → E ⇒ E-1 → 0 Sırada E

E → NULL Sırada F

F → NULL

Topological Sort ⇒ A-B-C-D-E-F

QUEUE → Indegree değeri 0 olanları kuyruğa ekle



Topological Sort

(1) Initialize $O(|E|)$ → store each vertex's indegree as an array

(2) Initialize queue with all indegree zero vertices $O(|V|)$

(1) indegree

A	B	C	D	E	F
0	1	1	2	2	0

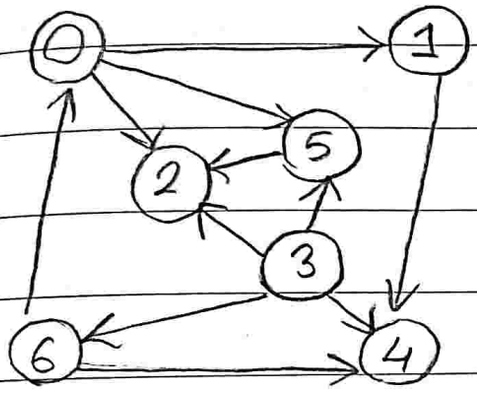
(2)

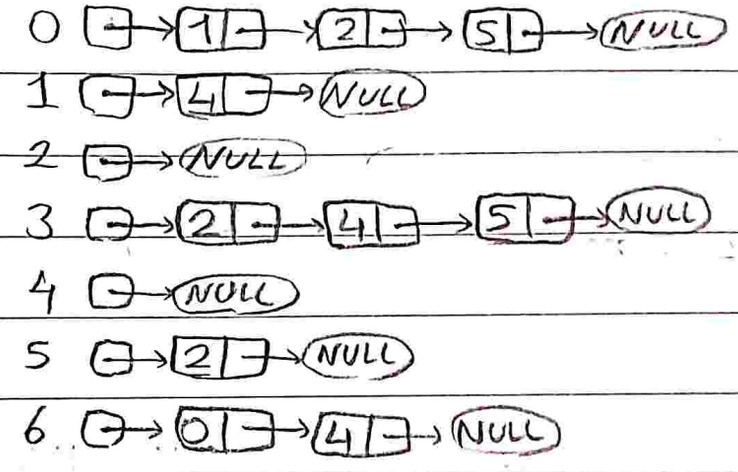
A	F				

queue

(3) While there are vertices remaining in the queue
- Dequeue and print $O(|V|)$
- Reduce indegree of all vertices adjacent to it by 1
- Enqueue adjacent vertices with indegree zero $O(|E|)$

→ $O(|V|+|E|)$ → Linear complexity





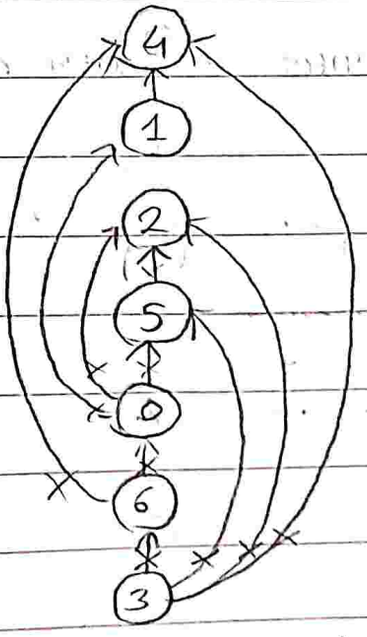
DFS

dfs(0)
- dfs(1)
  - dfs(4)
  - 4 done
- 1 done
- dfs(2)
- 2 done
- dfs(5)
  - check 2
- 5 done
- 0 done
- check 1
- check 2

dfs(3)
- check 2
- check 4
- check 5
- dfs(6)
  - check 0
  - check 4
- 6 done
- 3 done
- check 4
- check 5
- check 6

[DONE]

REVERSE ORDER



3 6 0 5 2 1 4
TOPOLOGICAL ORDER
3 6 0 1 4 5 2

Decrease and Conquer

② Decrease by a constant factor (Binary Search)

Comparison: $C_w(n) = \begin{cases} C_w(n/2) + 1 & \text{for } n > 1 \\ C_w(1) = 1 \end{cases}$ ← Backwards Substitution

↓

Backwards Substitution

$C_w(n) = C_w(n/2) + 1$

$= C_w(n/4) + 1 + 1$

$= C_w(n/8) + 1 + 1 + 1$

$C_w(n) = C_w(n/2^i) + i$

$n = 2^i$

for $n = 2^i$

$C_w(n) = C_w(2^i/2^i) + i$

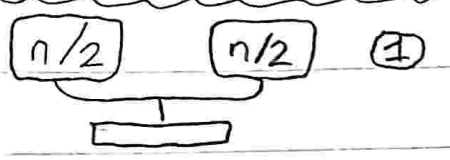
$= 1 + \log_2 n$ ↳ $n = 2^i$, $i = \log_2 n$

$C_{worst} \in O(\log_2 n)$

$C_{BS} \in O(\log_2 n)$

$C_{worst} \in \cancel{\Theta}(\log_2 n)$

Take Coin Problem

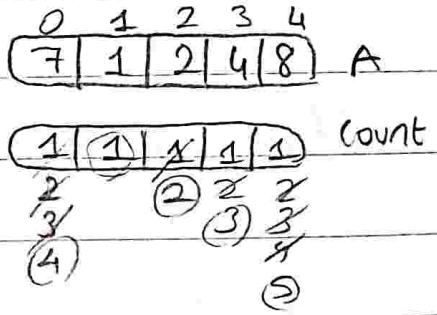


→ 7 paranın sahte olanını bulma (Decrease by a constant factor)

③ Variable Size Decrease Algorithms

finding the k^{th} smallest element!

linear sort with counting



Transform and Conquer

How do you find duplicates?

3 4 5 3 8 2 4

two stage process

① transformation stage for easier and faster solution

② conquering stage

Brute Force	Better
$O(n^2)$	sort $[O(n \log n)]$
Diziyi tarayıp yeni diziye aktarma	Compare $[O(n)]$

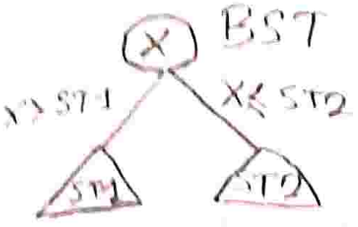
→ Presorting

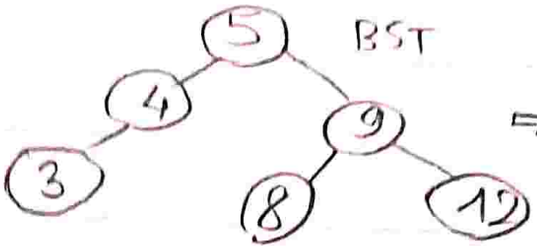
① unordered list → ordered list

② conquer: linear scan

Traversal and Complexity

Balanced Search Trees
(AVL-trees, Red-Black tree)





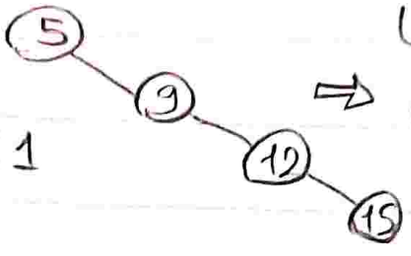
Search Complexity:

Balanced Tree Worst case: $O(\log_2 n)$

Balanced BST:

$|height(leftST) - height(rightST)| \leq 1$

(ST: Subtree)



Unbalanced BST Worst case: $O(n)$

We can transform it

AVL tree (Ağacın adı onu bulan insanların adından gelmektedir.)

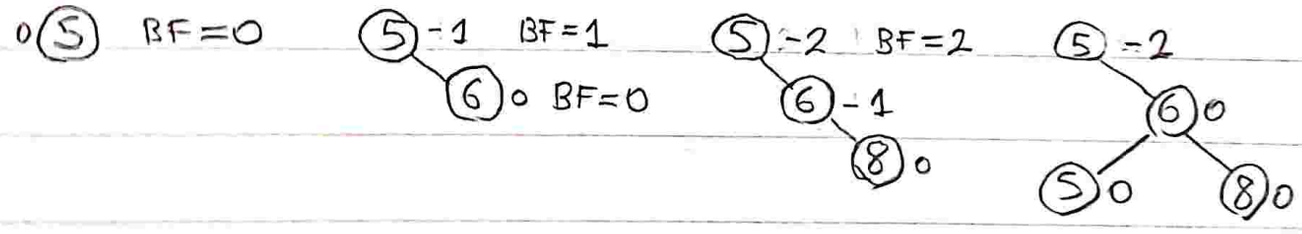
When searches are MORE FREQUENT than INSERTION and DELETION

Balance Factor:

Each node in an AVL tree:

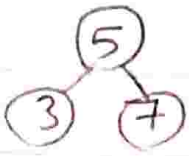
$BF = |height(leftST) - height(rightST)| \leq 1$ OLMALI

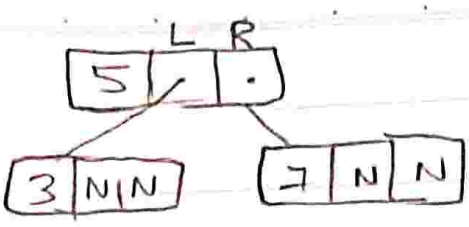
insert: 5, 6, 8

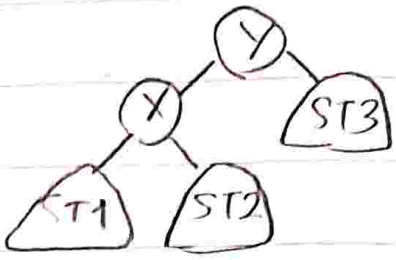


AVL BST Insertion

1. BST Insertion
2. Rebalancing
   1. Insert the key as in normal BST
   2. Update the height of every node
   3. Compute BF of each node
   4. If BF > +1 OR -1 perform ROTATIONS to restore Balance!

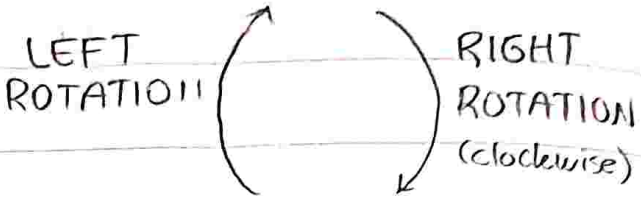


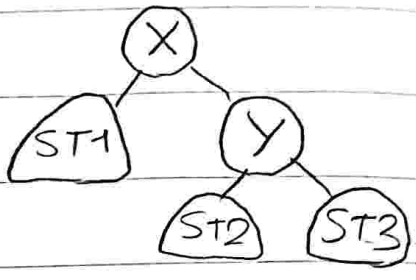





```
struct Node
{
    int key;
    struct Node *left;
    struct Node *right;
    int height;
}

int getHeight (struct Node *N)
{
    if (N == NULL)
        return 0;
    return N -> height;
}
```





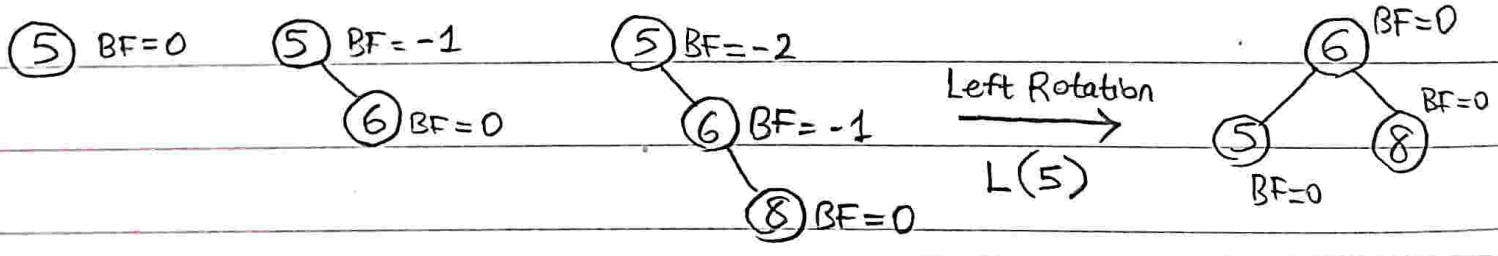
Rotations in AVL Tree

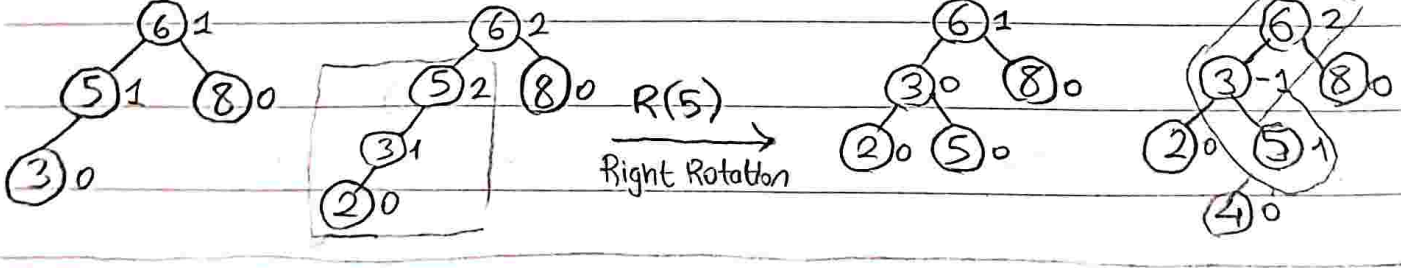
1. Left Rotation
2. Right Rotation
3. LR Rotation
   1. Left R.
   2. Right R.
4. RL Rotation
   1. Right R.
   2. Left R.

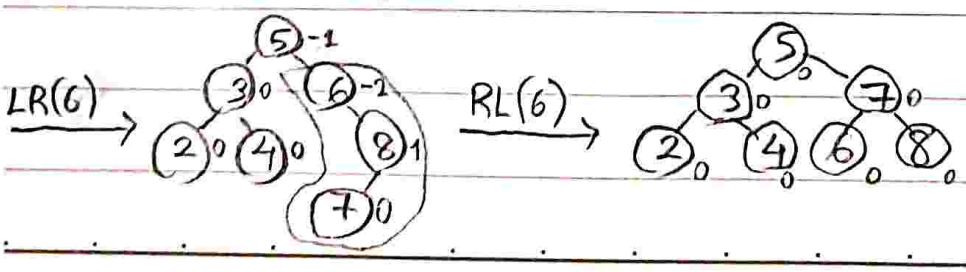
Example

Construct AVL Tree for:

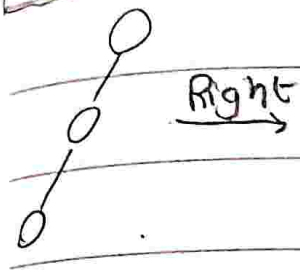
5, 6, 8, 3, 2, 4, 7

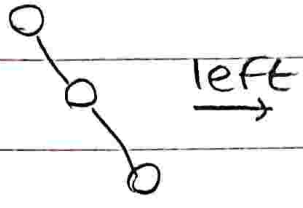




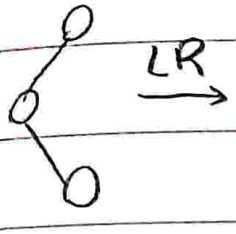


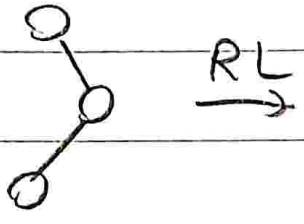
```
struct Node *rightRotate(struct Node *y)
{
    struct Node *x = y->left;
    struct Node *t2 = x->right;
    x->right = y;
    y->left = t2;
    return x;
}
```

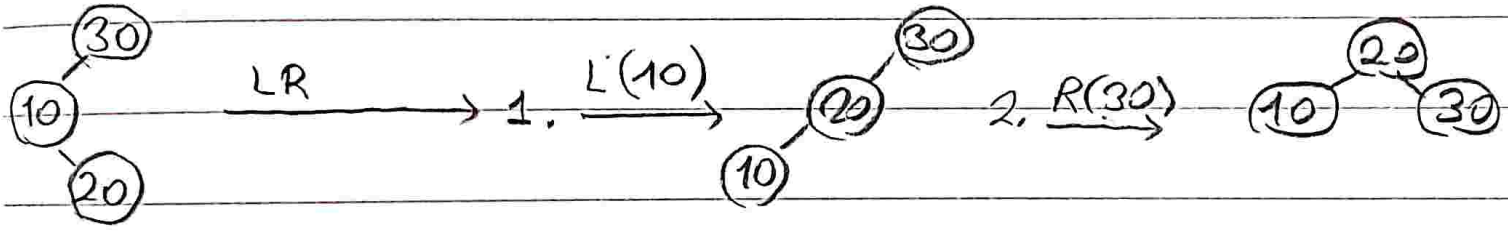


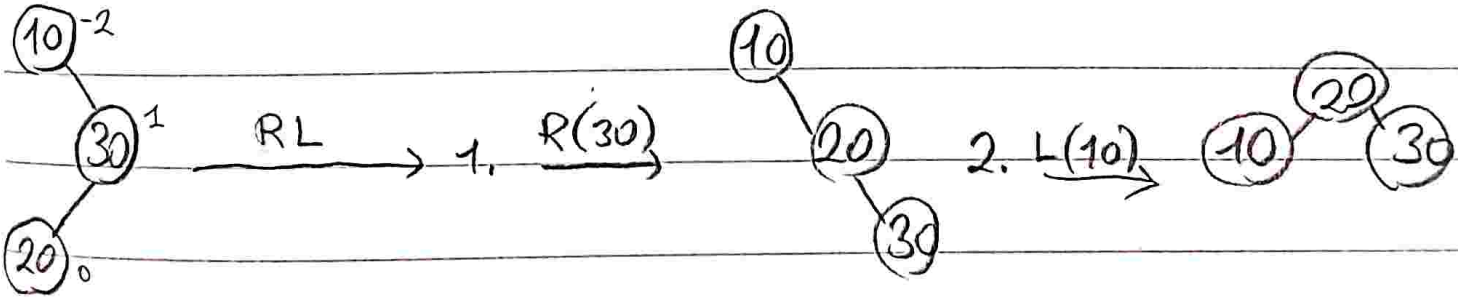


Genel Kural

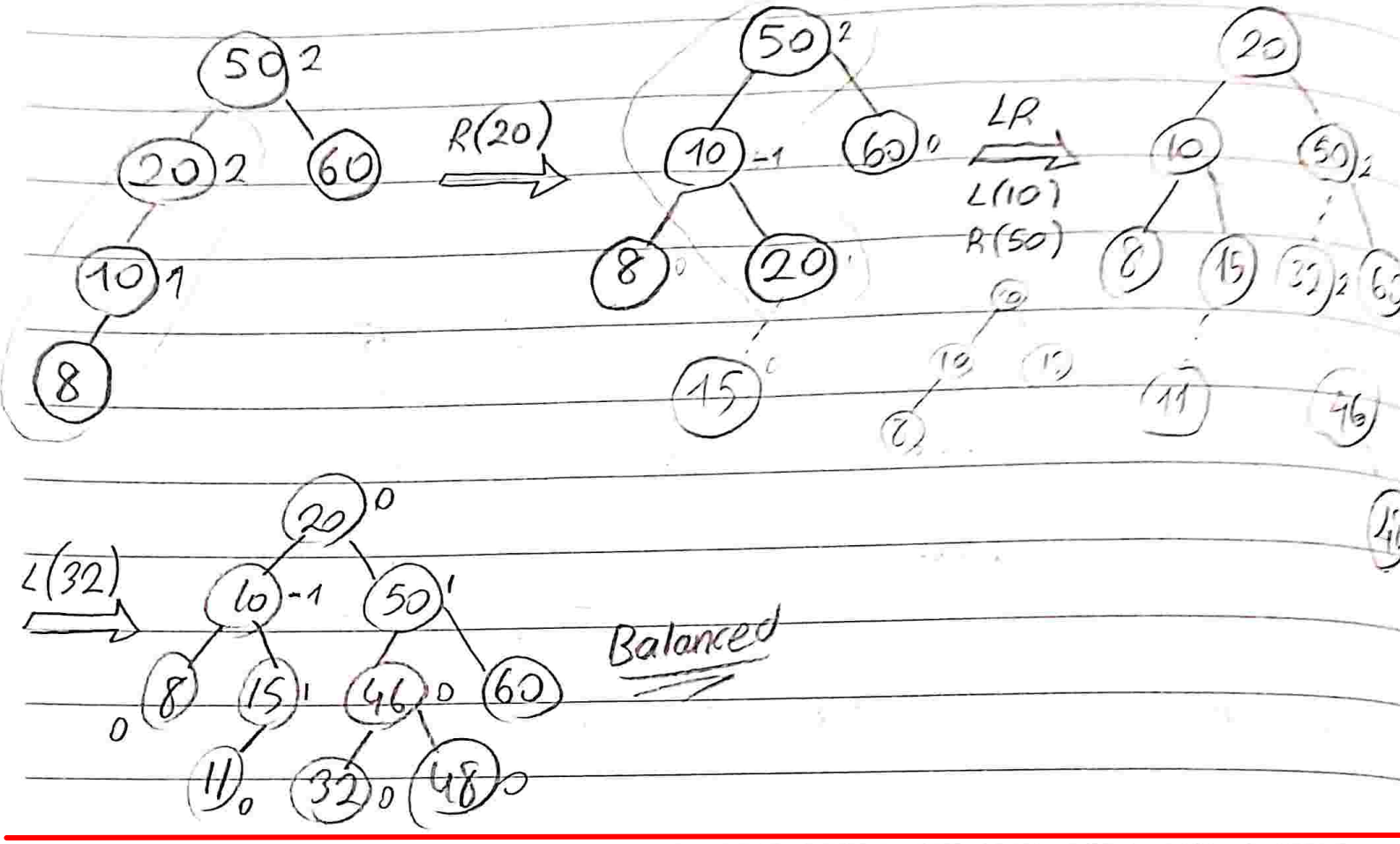








50, 20, 60, 10, 8, 15, 32, 46, 11, 48



UYGULAMA

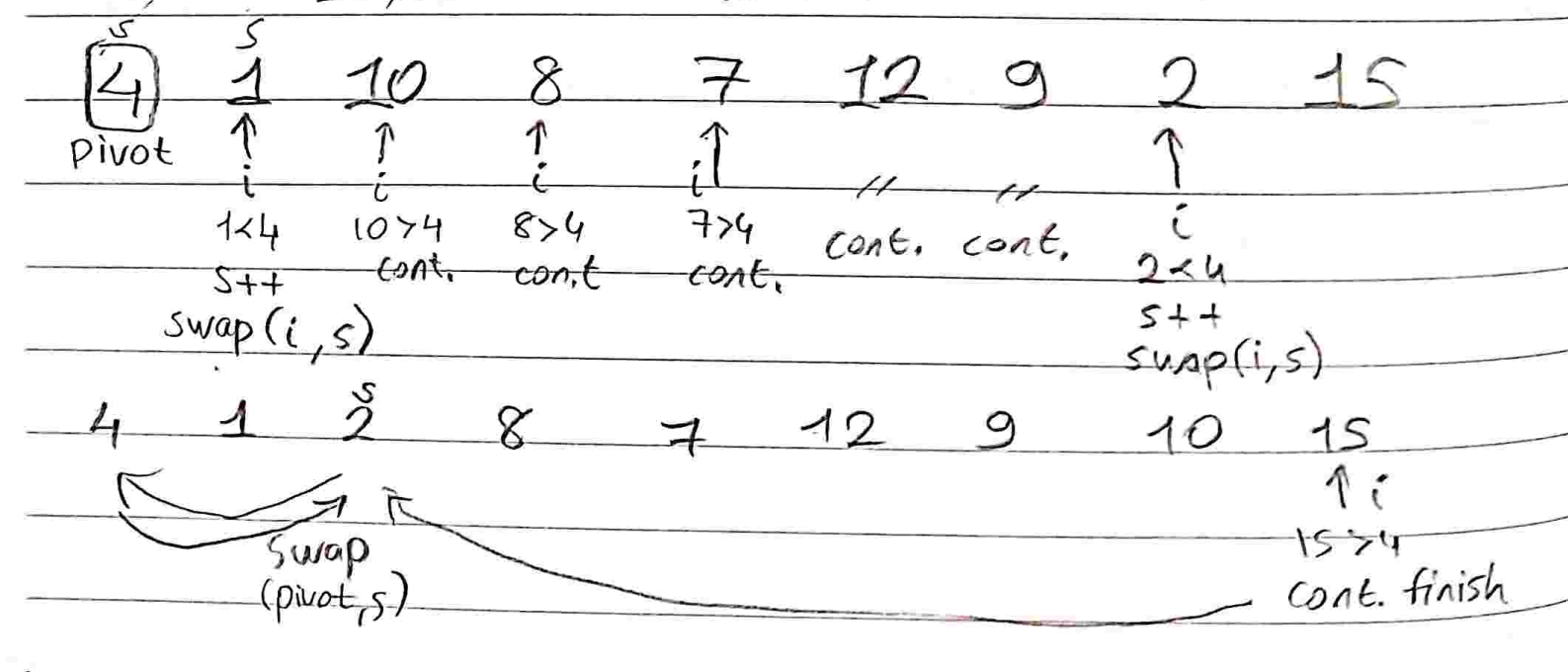
[K^th smallest number - pivot approach] UYGULAMA

QuickSelect

≤P (P) ≥P

2 k=3

k=5?



UYGULAMA

2	1	[4]	8	7	12	9	10	15
0	1	2	3	4	5	6	7	8

S = ? k-1 (k=5, 5. en küçük elemanı arıyoruz.)

2 < 4

2	1	4	[8]	7	12	9	10	15
			pivot	S, ↑i	↑i	↑	↑	↑i
				7<8	12>8	//	//	//
				S++	cont	cont	cont	cont
				swap				finish

swap(pivot, s)

2	1	4	7	[8]	12	9	10	15
0	1	2	3	4	5	6	7	8

S = ? k-1

4 = 4 ⟶ 8 = k^th smallest number (k=5)

Worst Case → Pivot her işlemde aynı kalırsa $O(n^2)$

Average Case → $O(n)$

UYGULAMA

Median of Two Sorted Arrays

$A = [a_1, a_2, \dots, a_n]$

$B = [b_1, b_2, \dots, b_m]$

Bu iki dizyi birleştirmeden birleşik dizinin ortanca değerini bulma

$A = 1, 4, 7$
$B = 2, 3, 5, 8$ → $1, 2, 3, (4), 5, 7, 8$?

	0	1	2	3	4	5
A =	1	12	15	26	38	

$n = 5$

	0	1	2	3	4	5
B =	2	13	17	30	45	60

$m = 6$

$L = \frac{m+n+1}{2} = 6$ → (0,5) low high

low high

$mid_1 = \frac{0+5}{2} = 2$, $mid_2 = L - mid_1 = 4$

	1	12	15	26	38

l_1 mid_1 r_1

	2	13	17	30	45	60

l_2 mid_2 r_2

$l_1 < r_2$ ✓

$l_2 < r_1$ X → mid_1 sağa kaymalı

low = mid + 1 = 3
high = 5

$mid_1 = \frac{3+5}{2} = 4$ $mid_2 = 6 - 4 = 2$

	1	12	15	26	38

l_1 r_1 mid_1

	2	13	17	30	45	60

l_2 r_2 mid_2

$l_1 < r_2$ X → low = 3
$l_2 < r_1$ ✓ high = $mid_1 - 1 = 3$
$mid_1 = 3$
$mid_2 = 3$

	1	12	15	26	38

l_1 r_1

	2	13	17	30	45	60

l_2 r_2

$l_1 < r_2$ ✓
$l_2 < r_1$ ✓ → MEDYAN:

$\max(A[mid_1], B[mid_2]) = 17/7$

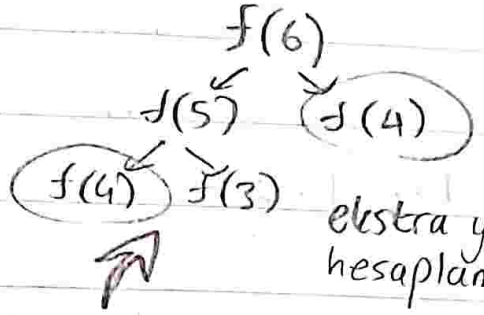
Dynamic Programming → planning

subproblems are DEPENDENT ⇒ recursion kullanılmamalı

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

1 1 2 3 5 8 ...

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



ekstra yeniden hesaplama var...

verimli değil

1. Define subproblems
2. Write the recurrence relation
3. Solve the BASE CASES
4. RECORD SOLUTIONS in a TABLE → array, matrix etc.

$$fibonacci[n] = \begin{cases} 0 & \text{if } n=0 \\ 1 & \text{if } n=1 \\ fibonacci[n-1] + fibonacci[n-2] & \text{if } n>1 \end{cases}$$

```
fib[0] = 0;
fib[1] = 1;
for(i=2; i<n; i++)
    fib[i] = fib[i-1] + fib[i-2];
```

minimum coin problem

you are given a set of coins: {1, 2, 5}

A target amount is: 11 TL

1+1+1... = (11)

2+2+...+2+1 = (6) (×5)

5+5+1 = (3)

⇒ A: target amount

dp[x]: minimum coins needed to make amount x — dp[11]=3

initial { dp[0] = 0 (zero coin): base case

dp[i] = ∞ i=1 to A

↳MAXINT ↳(örnek için 11TL)

x=1 → try coin 1: 1 + dp[1−1] = 1+0 = 1

(1 TL = c; x → 1−1 → c)

x=2 → try coin 1: 1 + dp[2−1] = 2 coin

try coin 2: 1 + dp[2−2] = 1 coin

} min(2,1) = 1 //

x=3 → try coin 1: 1 + dp[3−1] = 1+1 = 2 coin

try coin 2: 1 + dp[3−2] = 1+1 = 2 coin

} min(2,2) = 2 //

x=4 → try coin 1: 1 + dp[4−1] = 1+2 = 3 coin

try coin 2: 1 + dp[4−2] = 1+1 = 2 coin

} 2 //

Recurrence Relation:

$$dp[x] = \begin{cases} 0, & x=0 \\ \min(1+dp[x-c]) & x>0 \\ c \in coins,\ x-c \geq 0,\ \text{for all coins } c \end{cases}$$

initialize: $dp[x] = \begin{cases} 0 & x=0 \\ \infty & x>0 \end{cases}$

RUNNING TIME:

O(A * number of coins)

↳target amount

örnek: dp[6] = 1+dp[6−1] = 2

1+dp[6−2] = 3

1+dp[6−5] = 2

} 2 //

Brute Force:

2^n // 0-1 Knapsack Problem — W: maximum weight, # of items

↓ abandon ↘ select (can't repeat items)

value ↙ ↘ weight

↳ Gerçek zamanda çözülemez. (exhaustive search)

$1 \le i \le n$: weights : $w_1 \dots w_n$ maximize Σv_i subject to W

: value : $v_1 \dots v_n$ $\Sigma w_i \le W$

item ↓ weight →

P[i,j]: maximum profit at [i,j]

item weight

(örnek P[2,3] 2. item 3.kg için max kazanç)

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

case 1: thief takes item i

$P[i,w] = P[i-1, w-w_i] + v_i$

case 2: thief does NOT takes item i

$P[i,w] = P[i-1,w]$ ↓ BEFORE item i

W=5kg ↘

	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

1,2 ve 4 alındı

4. itemdayken 5kg ağırlıkla kazanç (P[4,5])

→ 3. item 3 kg durumu

$\max\{P[3,5-2]+15, P[3,5]\} = 37$ TL

(P[3,5-2]+15: taken, P[3,5]: not taken)

$$P[i,w] = \begin{cases} P[0,j]=0,\ P[i,0]=0 \text{ if } i=0, j=0 : \text{initialize!} \\ P[i-1,w] \quad \text{if } w_i > w \text{ (NOT TAKEN)} \\ \max\{(v_i + P[i-1, w-w_i]),\ P[i-1,w]\} \quad \text{otherwise} \end{cases}$$

($v_i + P[i-1, w-w_i]$: TAKEN! $P[i-1,w]$: NOT TAKEN!)

matrisle çözüldüğü için $2^n \longrightarrow O(n*w)$ ✓

hangi eşyaların alındığını bulmak için geri gitme:

START at P[n, w] (n: # of items)

when you go straight up: NOT TAKEN! ↑

(P[i, w] == P[i-1, w])

when you go LEFT up: item has TAKEN! ↖

RED BLACK TREE (RBT)

Another self-balancing BST (search, insert, delete → O(log n))

→ Every node is BLACK or RED

→ The root is BLACK

→ All null nodes are considered BLACK.

→ A RED node does not have a RED child

→ Every path from a given node to any of its descendant null leaves goes through the same number of BLACK nodes.

Edit Distance

spell checking

graffe → graph, graft, grail, giraffe

Edit Distance

each operation costs 1 points

→ Insert

→ Delete (remove)

→ Replace (modify)

kitten → sitting

maths → oats distance: 3

delete, insert

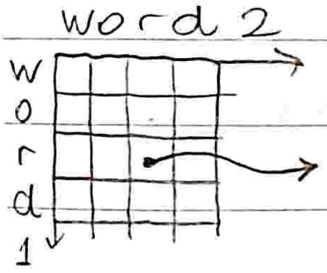
karga → (replace) kavga

Edit Distance / Levenshtein Distance:
minimum number of editing operations needed to transform a string into another string.

also used in computational biology

ACGGTA
TACGTA

$word1_N \longrightarrow word2_M$



ED[i][J]: word1[i] } word1 [0,.. i-1]
word2[j] } word2 [0,...J-1]

minimum number of edits for first i char of word 1 and first J char of word 2

① Initialization

ED[i][0] = i (no word2): i deletions

ED[0][J] = J (no word1); J insertions

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

word1 word2
"MA" " "

a) For Deletion

word1 → word2

KARA KAA

delete

word1 [0... i-1] } compare
word2 [0...J-1] }

If we delete LAST CHAR of word1, we need to match

X[0,.. i-1] → Y[0... J-1]

word1[i-1] = "R"

total cost:

ED[i][J] = ED[i-1][J] + 1

b.) For Insertion

KAS → KASA$_{insert}$

word1[0...i] → word2[0...J-1]

ED[i][J] = ED[i][J-1] + 1 → cost

$$ED[i][J] = \begin{cases} \text{Insert: } ED[i][J-1]+1 \leftarrow \text{(Extending word2 by one extra character} \\ \text{Delete: } ED[i-1][J]+1 \uparrow \\ \text{Replace: } ED[i-1][J-1]+1 \nwarrow \end{cases} \quad \text{minimum}$$

when word1[i][J] ≠ word2[i][J]

② Recurrence Relation:

$$ED[i,J] = \begin{cases} ED[i-1][J-1] & \text{if } word1[i] = word2[J] \\ \min \begin{cases} ED[i-1][J]+1 \uparrow D \\ ED[i][J-1]+1 \leftarrow I \\ ED[i-1][J-1]+1 \nwarrow R \end{cases} & \text{Otherwise} \end{cases}$$

		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

Time: O(N*M)

Space: O(N*M)

Backtracking: → Diagonal değerler eşitse harf aynı. Eşit değilse 3 operatör yönleri kıyaslanmalı

S
S
—
H S
— S
—
T H S
T — S
—
M A T H S
A R T — S

MATHS
-ARTS

MA_THS
-ART_S

```
for i = 1 to N
    for J-1 to M
        ED[i][J]....
```

Longest Common Subsequence

Subsequence:

yakamoz → yak, yaz, yaka, az, kaz

LCS is NOT Unique

word1	word2
yaka	yazmak

"yak" ↔ "yak"

common subsequence

$O(n * 2^m)$ ⚠

word2 →, word1 ↓

LCS[i][J]

case1: word1[i] = word2[J]: LCS[i][J] = LCS[i-1][J-1] + 1

LCS(word1[0...i], word2[0...J])

case2: word[i] ≠ word[J]

- word1[i] does not exist in subsequence: LCS[i][J] = LCS[i-1][J]
- word2[J] does not exist in subsequence: LCS[i][J] = LCS[i][J-1]

$$LCS[i][J] = \begin{cases} 0 & \text{if } i=0 \text{ or } J=0 \quad \text{(initialization)} \\ LCS[i-1][J-1] + 1 & \text{if } i>0, J>0, \text{word1}[i] = \text{word2}[J] \\ \max\{LCS[i-1][J], LCS[i][J-1]\} & \text{if } i>0, J>0, \text{word1}[i] \neq \text{word2}[J] \end{cases}$$

LCS[i-1][J]: word1[i] doesn't exist

LCS[i][J-1]: word2[J] doesn't exist

		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

"B" "BDCB" : LCS: 1

"BA" "BDCB" : LCS: 1

"BAC" "BDCB" : LCS: 2

"BACD" "BDCB" : LCS: 2 BD/BC

"BACDB" "BDCB" : LCS: 3 BCB BDB

String Search Algorithms

text

pattern : "NEEDLE"

'H'	'A'	'Y'	'S'	'T'	'A'	'C'	'K'	~~'#'~~	'W'	'I'	'T'	'H'	~~'#'~~	'A'	~~'#'~~	'N'	'E'	'E'	'D'	'L'	'E'

N E E D L E

X N E E D L E

X . . . . . .

X

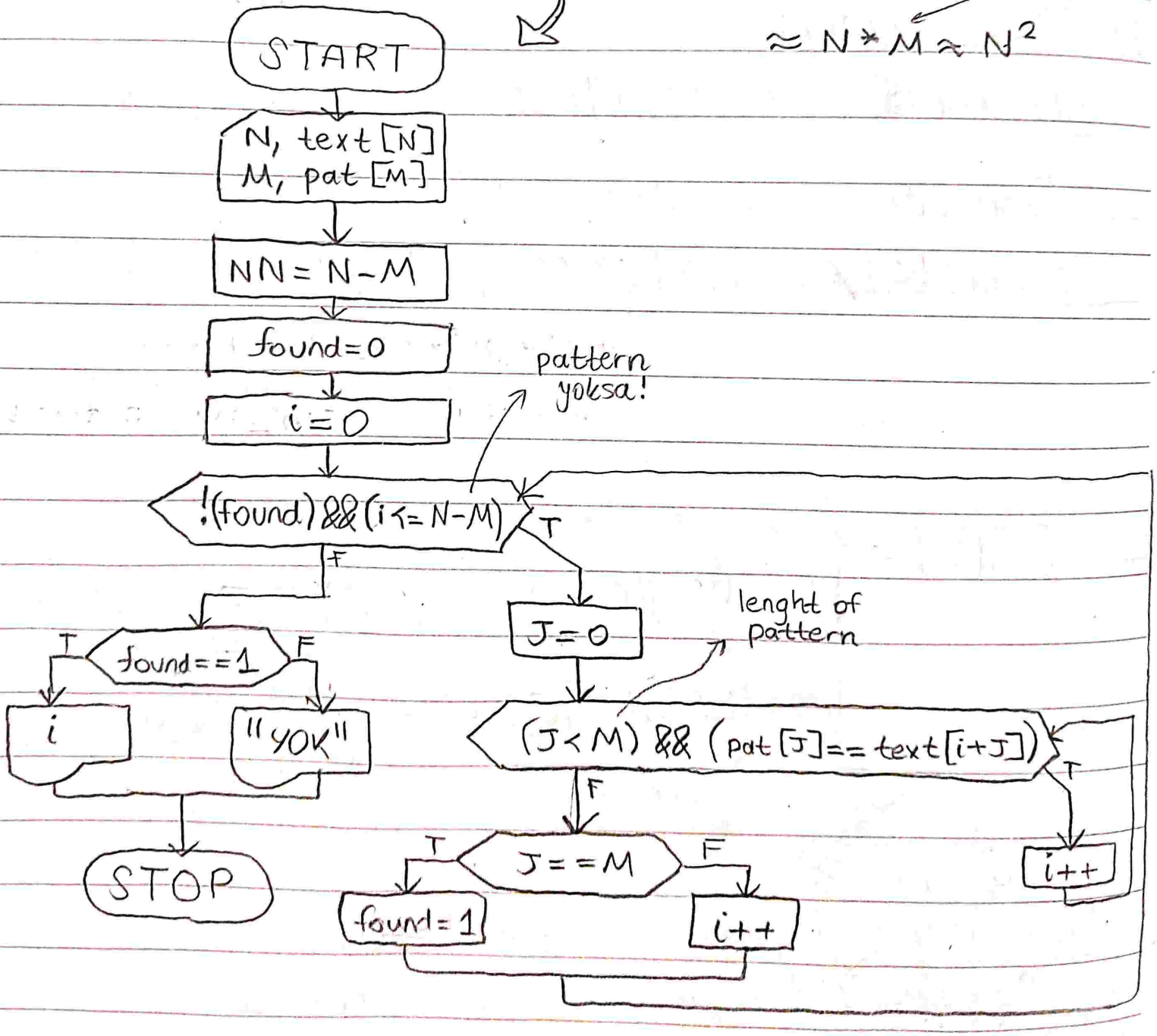
Brute Force

substring search →

→ Best Case: M ← string lenght

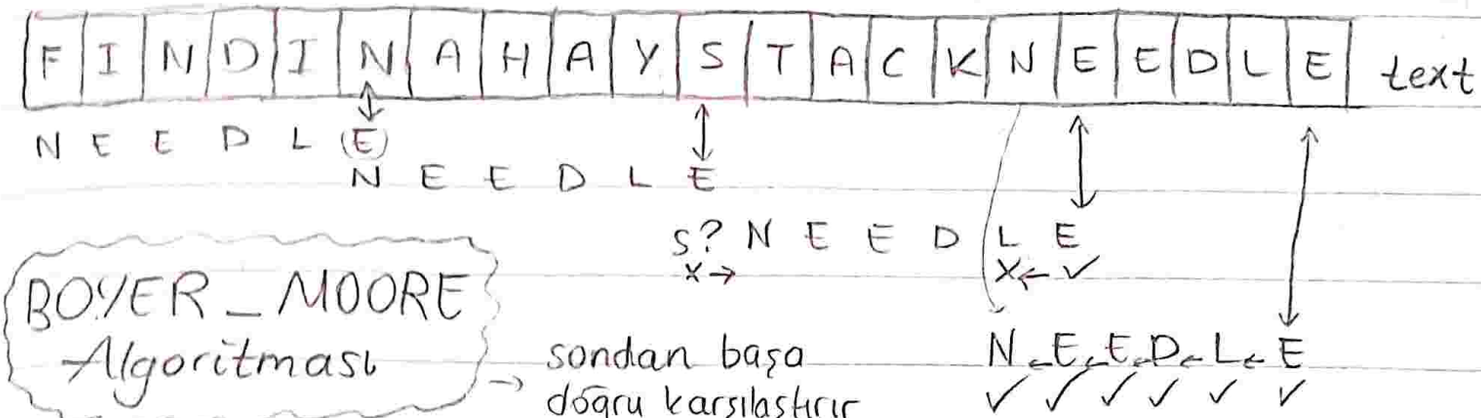
→ Worst Case: $(N-M+1) * M$

$\approx N * M \approx N^2$



String Search Algorithms

pattern : "NEEDLE"



BOYER_MOORE Algoritmasi → sondan başa doğru karşılaştırır

↳ How much to skip?

Case 1: mismatch character is NOT in the pattern

NAHAYS
NEEDLE
× × ××× → pattern[J]

skip = J+1 //

Case 2: mismatch character is in the pattern

FINDIN
NEEDLE
××× × ←

skip = J - rightMostOccurenceInPattern

```
int bayerMoore ( char pat[], char text[])
{
    int i, J, skip=0, N= strlen(text), M= strlen(pat);

    for (i=0; i<= N-M; i+=skip)
    {
        J= M-1;
        while ((J>=0)&&(pat[J]==text[i+J]))
        {
            J--;
        }
        if (J>=0)      // pattern bulunamadi!
        {
            skip= J - table[text[i+J]];
            if (skip<0)
                skip= 1;
        }
        else return i;    // index in text
    }
    return -1;            // not found
}
```

Sınavlarda çok severiz bu konuyu - hoca

String Search Algorithms

NEEDLE
0 1 2 3 4 5

'A'	'B'	'C'	'D'	'E'	....	'L'	..	'N'	....	'Z'
-1	-1	-1	3	5	....	4	..		---	-1

↑ pattern'in kaçıncı karakteri

```
void shiftTABLE(char pattern[], int table, int size)
                                        → // tablo uzunluğu
{
    int i, M = strlen(pattern);

    for(i=0; i<size; i++)
        table[i] = -1;
    for(i=0; i<M; i++)
    {
        table[pattern[i] - 'A'] = i;
    }
}
```

BAYER_MOORE → Average: N/M

→ Worst: N*M

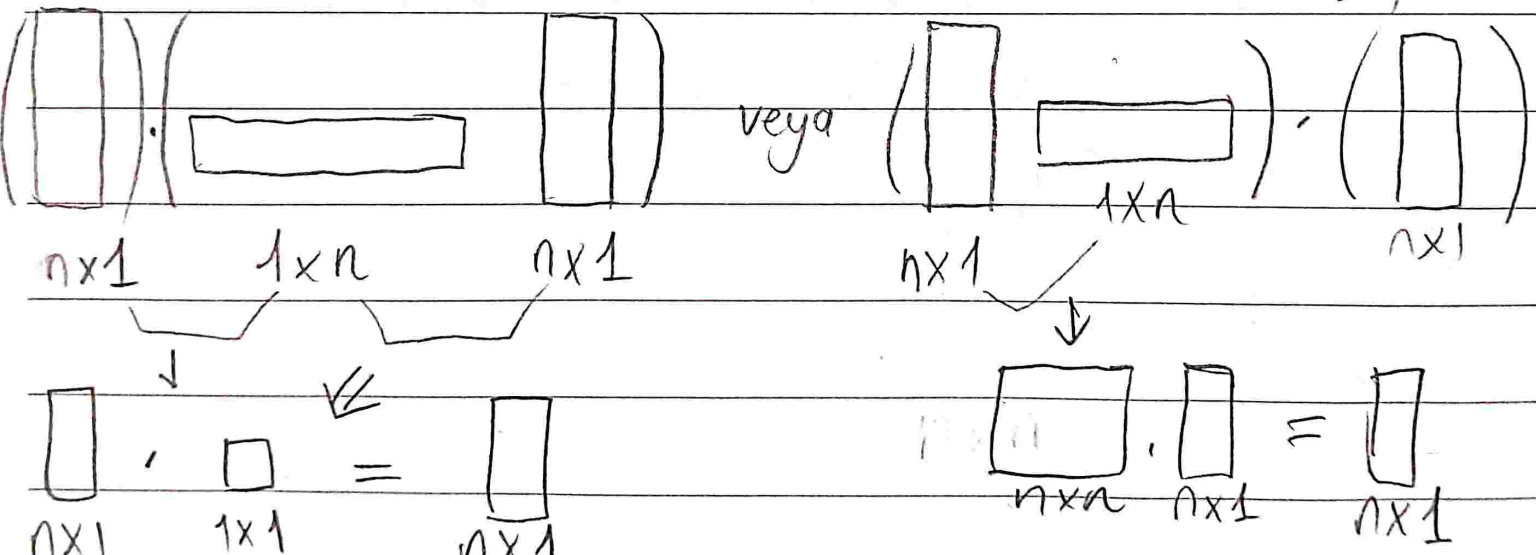
UYGULAMA

UYGULAMA 2

Dinamik Programlama

Matrix Chain Multiplication

$A_1 A_2 A_3 A_4 \ldots\ldots A_n$ en az maliyetle tüm matrisleri nasıl çarparız?



UYGULAMA

A_1 A_2 A_3

10x30 30x5 5x60

$(A_1.A_2)A_3$ veya $(A_1)(A_2.A_3)$ 30x60

10.30.5 = 1500 çarpma işlemi

10x5 5x60

10.5.60 = 3000 çarpma işlemi

4500 çarpma maliyeti

30.5.60 = 9000

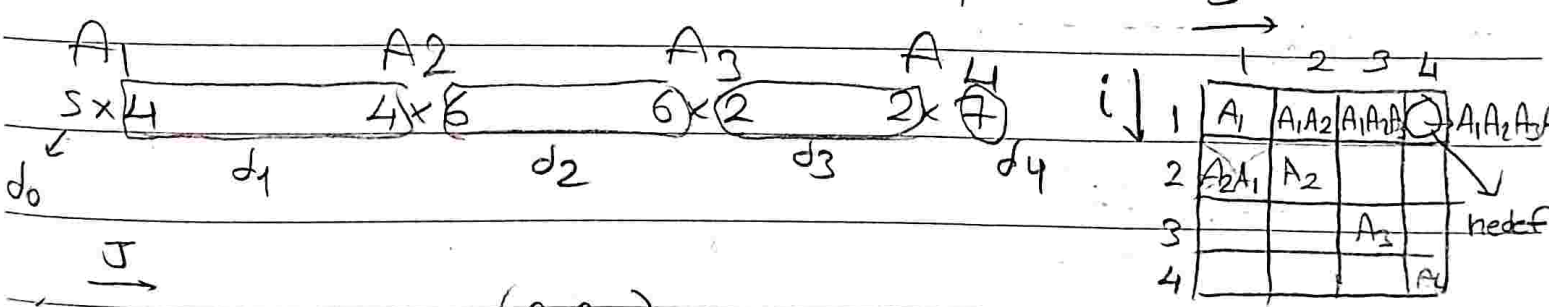
10.30.60 = 18000

27000 çarpma maliyeti

örneğin n=5 için $\frac{1}{n}\left(\frac{2(n-1)}{n-1}\right) = 14$ farklı çarpım şeklimiz olur.

n=10 için 16796 farklı şekil var, dinamik programlama önemli.

Matrix Chain Multiplication



(sadece diagonal ve üstü önemli)

Maliyet → M

J

i

M

0	120	88	
x	0	48	
x	x	0	84
x	x	x	0

(A_1A_2)

M[1][2] = 5x4x6 = 120

M[2][3] = 4x6x2 = 48

M[3][4] = 6x2x7 = 84

$(A_1)(A_2A_3)$ $(A_1A_2)(A_3)$ → M[1][2] + M[3][3] + 5.6.2 = 180

M[1][1] + M[2][3] + 5x4x2 = 88 ($A_1.A_{23}$)

MIN = 88 → M[1][3] = 88

k

S

0	1	1	
	0	2	
		0	3
			0

paranlez konumu nerede?

UYGULAMA

A_1 A_2 A_3 A_4
5×4 4×6 6×2 2×7

$A_2 . A_3 . A_4$

$(A_2)(A_3 . A_4)$

$M[2][2] + M[3][4] + 4 \times 6 \times 7 = 252$

$(A_2 A_3)(A_4)$

$M[2][3] + M[4][4] + 4 \times 2 \times 7 = 104$

MIN 104 → $k = 3$

M

0	120	88	158
X	0	48	104
X	X	0	84
X	X	X	0

S (k)

0	1	1	3
X	0	2	3
X	X	0	3
X	X	X	0

a) $(A_1)(A_2 A_3 A_4)$

b) $(A_1 A_2)(A_3 A_4)$

c) $(A_1 A_2 A_3)(A_4)$

$M[1][4] = \min$
- a) $M[1][1] + M[2][4] + 5 \times 4 \times 7 = 244$
- b) $M[1][2] + M[3][4] + 5 \times 6 \times 7 = 414$
- c) $M[1][3] + M[4][4] + 5 \times 2 \times 7 = 158$

$= 158$

$k \rightarrow \min \Rightarrow 3$

Matrix Multiplication

$M[i][j] = M[i][k] + M[k+1][j] + (d_{i-1} \cdot d_k \cdot d_j)$

$S \rightarrow ((A_1) A_2 \ A_3) A_4$

$k=3$

$k=1$

// cost

Let M nxn matrix, d array dimensions

Let S nxn matrix // index

```
for i=1 to n do
    M[i][i]=0
for L=1 to n-1 do      // L = # of matrix -1 (n-1)
    for i=1 to n-L do
        J=i+L
        for k=i to J-1
            cost=M[i][k]+M[k+1][J]
                 +(d[i-1]*d[k]*d[J])
            if(cost<M[i][J])
                M[i][J]=cost
                S[i][J]=k
```

3×5×3 4C

(3×2 2×5 5×3) 3×2×3 18

30 15+ 60, 2 6

O n^2

24+

UYGULAMA

ACTIVITY SELECTION (/WEIGHTED INTERVAL SCHEDULING)

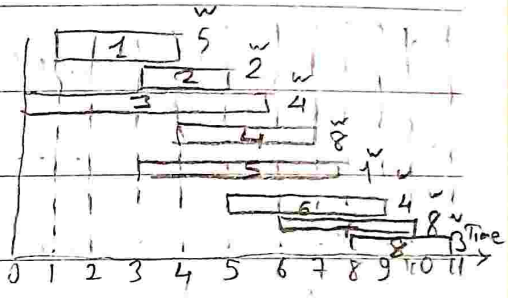
N jobs → weight
→ start time - finish time } max profit

Örnek → 6 jobs

	1	2	3	4	5	6
start:	1	3	0	5	8	5
finish:	2	4	6	7	9	9
	✓	✓	X	✓	✓	X

1, 2, 4, 5

(Çakışmayan en fazla sayıda işi seçme problemi)



→ better to sort jobs by finished time

J	w(J)	q(J)	MP → max profit
0	-	-	0
1	5	0	5 → max(MP[0], w(1)+MP[q1]) = 5
2	2	0	5 → max(MP[1], w(2)+MP[q2]) = 5
3	4	0	5 → max(MP[2], w(3)+MP[q3]) = 5
4	8	1	13 → max(MP[3], w(4)+MP[q4]) = 13
5	1	0	13
6	4	2	13
7	8	3	13
8	3	5	16 → max(MP[7], w(8)+mp(q8)) = 16

$$Max\_Profit = \begin{cases} 0 & , J=0 \\ \max(MP(J-1),\ w(J)+MP(q_J)) & , \text{otherwise} \end{cases}$$

```
find_solution(J):
    if(J==0)
        return 0
    else if w(J)+MP(qJ) > MP(J-1)
        return findSolution(qJ)
    else
        return findSolution(J-1)
```

```
MP[0]=0
for J=1 to N do
MP[J]=max...
```

Hashing

Worst Case →	Linear Search	Binary Search	BST	Balanced BST
Searching	n	$\log_2 n$	n	$\log_2 n$
Sorting				

1-100 arasında öğrenci numarası aransın:

search, insert, delete → O(1)

1- 1.000.000 arası numara

↓

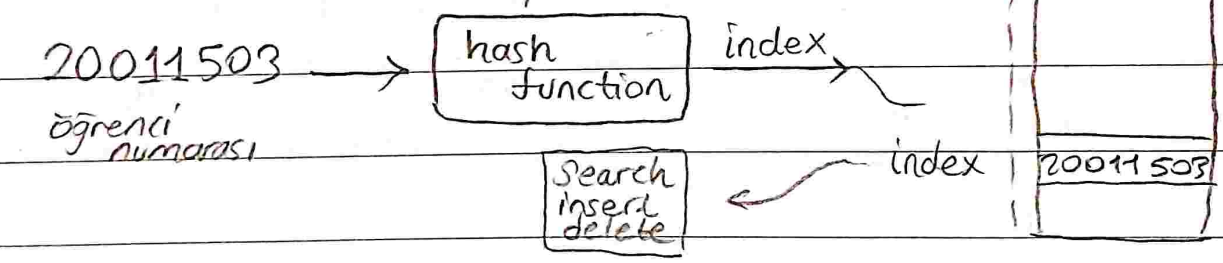
100 ← number of objects

1.000.000 array size??

Hashing: Searching O(1)

~Hashing~

(for IBM-701 computers → 1953)



→ farklı numaralar için aynı indis verilirse? → collusion problem

Hash Function

must be → easy to compute
→ consistent

→ Division Method
→ Multiplication Method
(en çok kullanılan yöntemler)

(1) Division Method

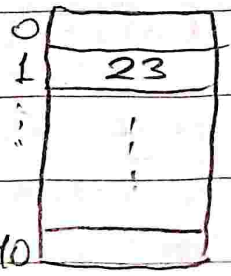
hashfunction(key) = key % m

↓ example: 23

↓ hash table size

key: 23, table size: 11

→ index = 23 mod 11 = 1



Hashing

(2) Multiplication Method

Knuth'ın

$A = \frac{\sqrt{5}-1}{2} = 0.618033$

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

m: table size, A: constant

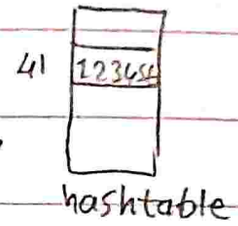
$m = 10.000 \quad key = 123456$

$hashFunction(123456) = \lfloor 10000 * (123456 * 0.618033) \bmod 1 \rfloor$

76300.004151

$10000 * 0.004151$

$\lfloor 41.151 \rfloor = 41$

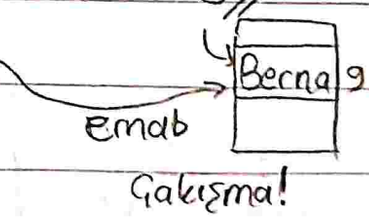


→ Key must be unique ⇨ TCKN, student Id, phone Nmbr...

If not ↘

(1) String "berna" hashF("berna") → integer Index

"ernab" = → 42 → 'b'-'a' + 'e'-'b' + ... = 42 % m (11) = 9



Horner's Method

R=31 L = strlen("berna")

$31^{L-1} * str[0] + 31^{L-2} * str[1] + \dots + 31^{0} * str[L-1]$

(Bir asal sayı kullanılarak her harfe farklı ağırlık verilir.)

```
≈
    lenght = strlen(str);
    key = 0;
    for (i=0; i<lenght; i++)
    {
        tmp = str[i] - 'A' + 1;
        key = 31 * key + tmp;
    }
≈
```

prime number

JAVA: Binary representation of a string

Hashing

② Compound Key

day-month-year ⇝ key = $[(day * R + month) \% m + R \% year] \% m$ (R: prime integer)

③ Floating Point

key = 35.780 ⟶ round [key * 10000] % m (10000: constant)

Çakışma problemi nasıl çözülür

a) Open Adressing

a.1) Linear Probing

each check is a PROBE

Use an array of size M >> N (M: table size, N: number of elements)

Yer doluysa (lineer şekilde) boş yer aranır

prime number

hashFunction(key, i) = [hashFunction(key) + i] mod m

i: step size (check), hashFunction(key): key % m

m = 7

0	NULL	19
1	NULL	
2	NULL	
3	NULL	
4	NULL	
5	NULL	5
6	NULL	12

key = 5 → adr: 5 % 7 = 5

key = 12 → adr: 12 % 7 = 5 → DOLU

→ key = (12 % 7 + i) % 7 = 6 (i = 1)

key = 19 → 19 % 7 = 5 → DOLU i = 1 → DOLU

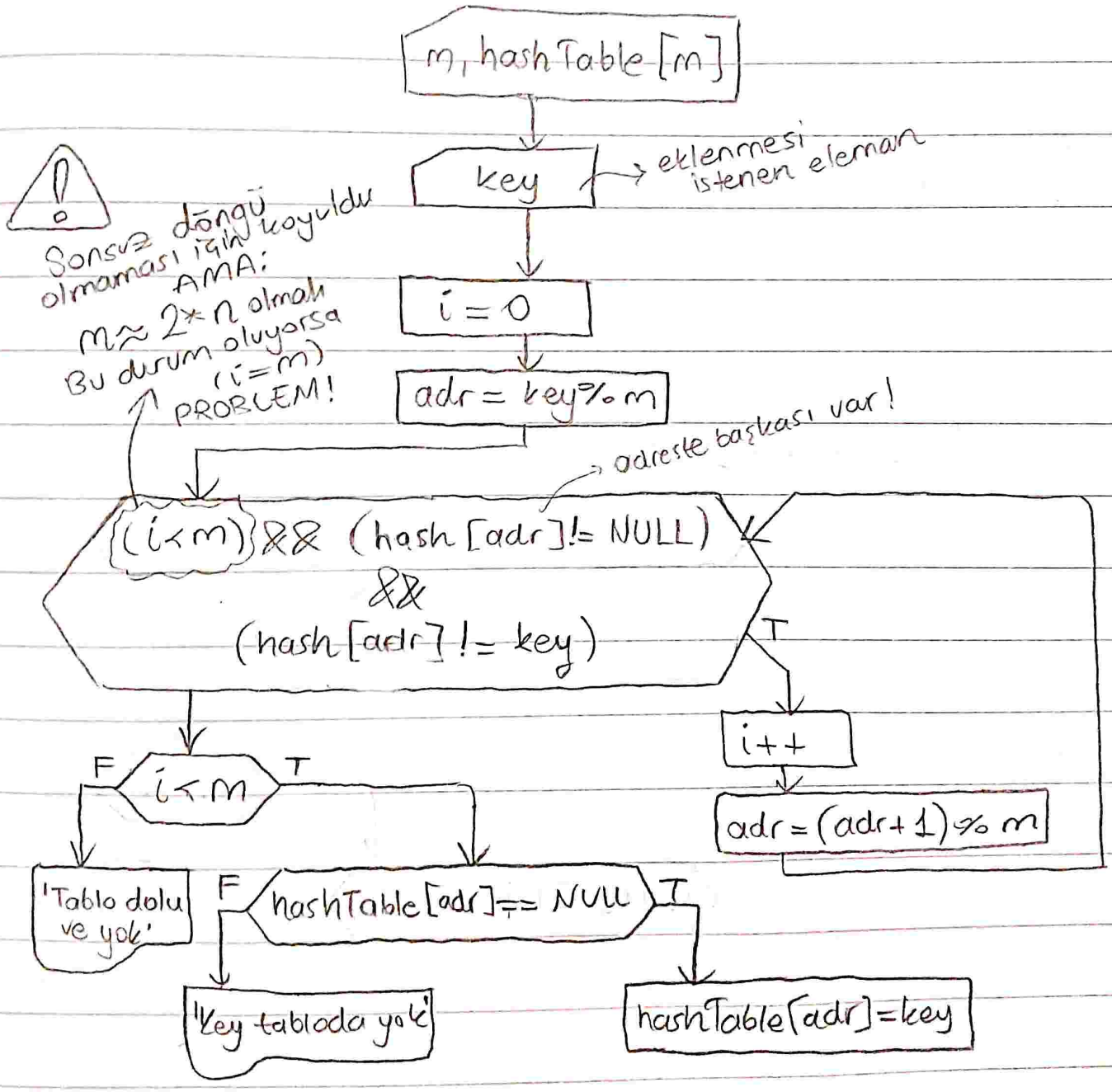
→ key = (19 % 7 + 2) % 7 = 0 (i = 2)

→ M genelde 2 kat seçilir.

→ key = 0 gelse? → CLUSTERING PROBLEM

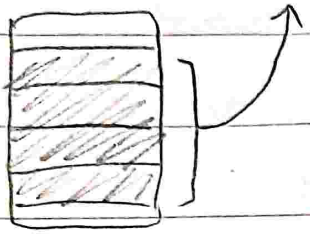
(Linear Probing)

Bir hash tablosuna Linear Probing ile yeni eleman eklerecek. Eleman daha önce eklliyse kullanıcıya uyarı mesajı veren, yoksa tabloya yerleştiren algoritmayı tasarlayınız.



Hashing

Clustering Problem



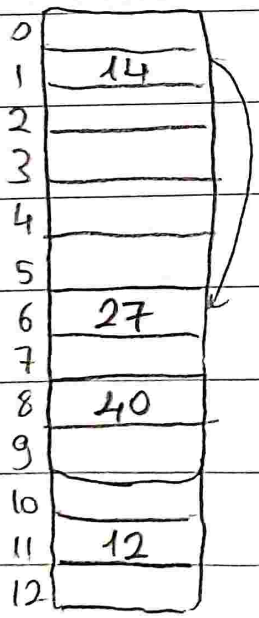
Çözüm
~~Linear Probing~~ → Double Hashing

a.2) Double Hashing

(Table size = m = 13)

$$hashFunction(key, i) = [hash1(key) + i * hash2(key)] \% m$$

- i → i^{th} step (check)
- $hash1(key)$ → key % 13 (↑ m)
- $hash2(key)$ → key % 11 (genellikle m'den küçük, kaç adım ileri gidilecek)



key = 27

hash1(key) = 27 % 13 = 1 //

i = 1

1 + 1 * hash2(key) = 6

↑ hash1(key)

27 % 11 = 5

6 % 13 = 6 //

key = 40

40 % 13 = 1 → dolu

(1 + 1 * (40 % 11)) % 13 => 8 //

(40 % 11 = 7)

Double Hashing ile peş peşe birikmenin önüne geçilmiş olunur.

Hashing

Peki nasıl O(1)?

Load Factor $\alpha = \frac{N}{M}$ → number of elements; → table size

$\alpha \approx 1/2$

Linear Probing

	%50	%66	%75	%90 ⇨ (doluluk oranı)
search →	1.5	2.0	3.0	5.5
insert →	2.5	5.0	8.5	55.5

Double Hashing

search →	1.4	1.6	1.8	2.6
insert →	1.5	2.0	3.0	5.5

Birthday Paradox

Birthday Paradox

Uniform Hashing Assumptions

Table size: m key = m, 2m, 3m... → Each key is equally likely to hash to any slot in the table

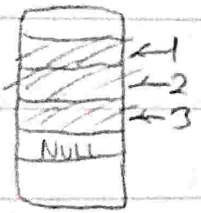
$Pr[\text{hashfunction}(key) = i] = \frac{1}{m}$

Expected value: $E[X] = \sum_{i=1}^{n} i * \text{Probability}(x = i)$ → uzun vadede ağırlıklı ortalama

Zar problemi: $E[X] = \sum_{i=1}^{6} i * (\underbrace{Pr[i]}_{1/6}) = 3.5$

Linear Probe

clustering:
$i \to i+1 \to i+2 \ldots$



probability of a slot → full α / empty $1-\alpha$

probability of a second slot is empty: $(\alpha \cdot (1-\alpha))$

k^{th} slot is empty: $(\alpha^{k-1}(1-\alpha))$

ne yapmalıyız ki karmaşıklık O(1) olsun?

Load Factor → $\alpha = \frac{n}{m}$ → # of elements / table size

Hashing

$P(k \text{ full cells before empty}) = \alpha^k \times (1-\alpha)$ → $(k+1)^{th}$ empty

Geometrik dağılım

Expected value: $E[k \text{ full cells}] = \frac{1-p}{p} = \frac{1-(1-\alpha)}{1-\alpha} = \frac{\alpha}{1-\alpha}$

$E[\text{full}] = \frac{1}{p} = \frac{1}{1-\alpha}$

Insert için toplam bakılarak probe sayısı
↳ unsuccessfull search

Search: $E[\text{succesfull search}] = \frac{1}{2}\left[1 + \frac{1}{1-\alpha}\right] = \frac{1}{2}\left[\frac{2-\alpha}{1-\alpha}\right]$

yaklaşık olarak cluster'ın ortasında yaklaşılır

Örnek: Dolulu oranı 1/2 ise → $\alpha = \frac{n}{m} = \frac{5}{10} = 0.5$ $\frac{3}{2} = 1.5$

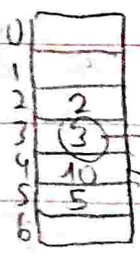
olasılıksal olarak 1.5 adım sonra aranan eleman bulunacak

olasılıksal olarak 2 adım 2 sonra insert edilecek

(burayı tam yazamadım, hoca işlemlerde bir hata var gibi bir şey demişti sanırım)

	worst case			average case			
	Search	Insert	Delete	Search	Insert	Delete	
Sorted Array	logN	N	N	logN	N/2	N/2	???
Unsorted Array	N	N	N	N/2	N/2	N/2	
Linear Probing	N	N	N	1	1	1	Universal Hashing Assumption

Deletion Problem



DELETE? → Silinen yerleri işaretlesek?

3 silindi → NULL

10'u ararsak → 10%7 = 3 → NULL → 10 yok!?

* table size should be usually prime

n→5 m=11

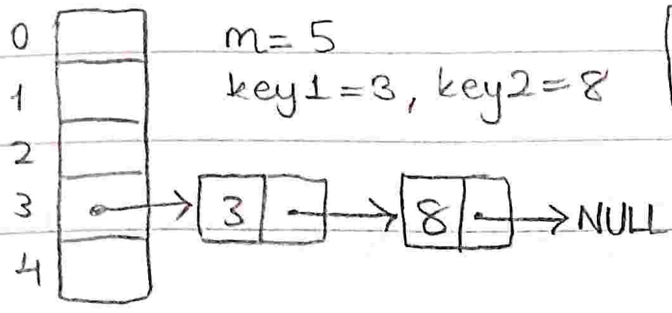
n→10 m=23

neden? ↴

m: 8 → keys 64, 400, 128...

Hashing

Separate Chaining



$$m = \frac{N}{5}$$

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

struct Hash
{
    struct Node *head;
    int count;
}

struct hash *hashtable = NULL;

void insertToHashtable (int key, char name)
{
    int hashIndex;
    hashIndex = key % M;
    struct node *newNode = createNode (key, name);
    if(!hashTable[hashIndex].head)
    {
        hashtable[hashIndex].head = newNode;
        hashtable[hashIndex].count = 1;
    }
    else
    {
        newNode->next = hashtable[hashIndex].head;
        hashTable[hashIndex].head = newNode;
        hashTable[hashIndex].count++;
    }
}
```

Hashing

```
int main ( )
{
    int key;
    char name;
    ≈
    hashTable = (struct hash*) calloc (M, sizeof(struct Hash));
    ≈
    ≈
    insertto HashTable (key, name);
    ≈
    ≈
}
```

M >>> N → çok boş

↘ delete için iyi

	worst			average		
	Search	Insert	Delete	S	I	D
Separate Chaining	N	1	N	1	1	1

Universal Hashing

Bad keys → Bad hashing

key → m, 2m, 3m

h1( ), h2( )...

randomly choosen

Hashing Application

	key
spell checker	(word)
browser	(URL)
Spam filter	(IP)
credit cards	(card number)

p: large prime $a \in \{1 \dots p-1\}$, $b \in \{0 \dots p-1\}$

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

table size is not random!

Example: p:17, m=7, a=3, b=4

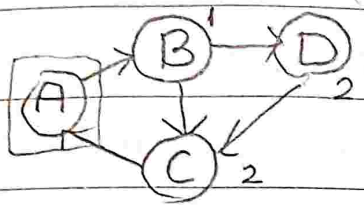
$h_{3,4}(8) = ((3*8+4) \% 17) \% 7 = 4$ //

(3 ↓ a, 4 ↓ b; 3*8+4 = 28)

Shortest Path

Shortest Path

BFS



A	B	C	D
0	∅	∅	∅
	1	2	2

Queue [A|B]

A

[B|C|D]

B

...

(unweighted digraph)

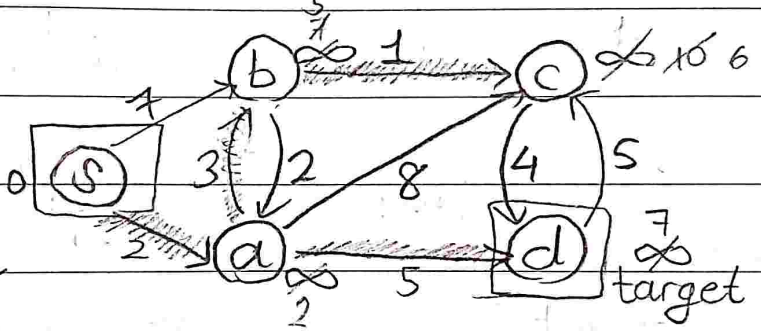
(weighted di-graph positive weights) ?

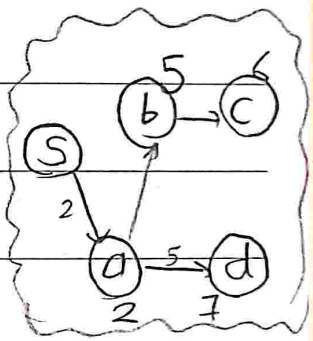
source → target

SHORTEST PATH?

Greedy Algorithm

Dijkstra Algorithm





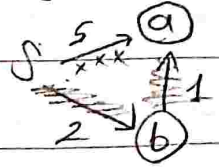
① Initialization

set distance of source = 0
set distance of all others = ∞
mark all nodes unvisited

② Select closest unvisited node

choose node u with minimum distance
mark u as visited

③ Edge Relation



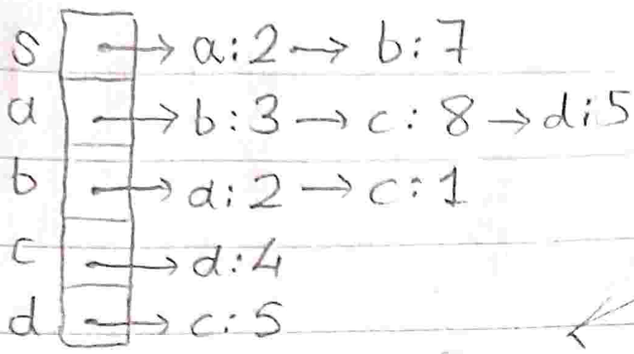
if dist[u] + w(u,v) < dist[v]

dist[v] = dist[u] + w(u,v)

④ REPEAT from 2

UNTIL TARGET NODE IS FINALIZED

Shortest Path: Dijkstra



Initialization

v	s	a	b	c	d
d[v]	0	∞	∞	∞	∞
pred[v]	NULL	NULL	NULL	NULL	NULL
color[v]	w	w	w	w	w

↳ w: unvisited, b: visited

priority queue (Heap tree)

v	s	a	b	c	d
d[v]	0	~~∞~~	~~∞~~	∞	∞

dequeue

(her adımda en küçük değer seçilir)

v	a	b	c	d
d[v]	2	7	∞	∞

dequeue

v	b	c	d
d[v]	5	10	7

dequeue

v	c	d
d[v]	6	7

dequeue

v	d
d[v]	7

dequeue

v	s	a	b	c	d
d[v]	0	2	7	∞	∞
pred[v]	N	S	S	N	N
color[v]	b	w	w	w	w

v	s	a	b	c	d
d[v]	0	2	5	10	7
pred[v]	N	S	a	a	a
color[v]	b	b	w	w	w

v	s	a	b	c	d
d[v]	0	2	5	6	7
pred[v]	N	s	a	b	a
color[v]	b	b	b	w	w

v	s	a	b	c	d
d[v]	0	2	5	6	7
pred[v]	N	S	a	b	a
color[v]	b	b	b	b	~~w~~ b

Shortest Path: $s \to a \to d : 7$ //

(d[v] => distance to source, pred[v] => v'ye en son nereden gelindiği, color[v] => düğüme uğrama durumu)

Shortest Path: Dijkstra

```
Dijkstra (G, s, w)
{
    for (each u ∈ V)
    {
        d[u] = ∞
        color[u] = white
    }
    d[s] = 0
    pred[s] = NULL
    PQ = (queue with all vertices)
    while (non-empty(PQ))
    {
        u = extract-min (PQ)
        for(each v ∈ Adj[u])
        {
            if(d[u] + w(u,v) < d[v])
            {   d[v] = d[u] + w[u,v]
                decrease-key(PQ, v, d[v])
                pred[v] = u
            }
            color[u] = black
        }
    }
```

extract-min ⇒ heap ile: $O(\log V)$

decrease-key ⇒ $O(\log V)$; en fazla E kez

Average case: $O((V+E)\log V)$

Worst case: $O(V^2 \log V)$ → en küçük mesafeyi bulma

Shortest Path: Floyd-Warshall

Floyd-Warshall — All Pair Shortest Path (APSP)

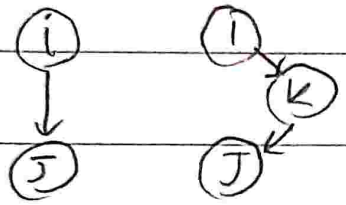
Finds shortest path between all pairs of vertices!

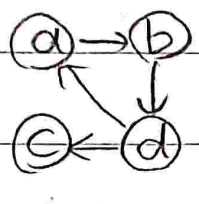
① Warshall Algorithm

Is there a path from i to j

USE TRANSITIVE CLOSURE{

$R[i][j] = R[i][j] \vee (R[i][k] \wedge R[k][j])$





$A = (R_0)$

	a	b	c	d
a	0	1	0	0
b	0	0	0	1
c	0	0	0	0
d	1	0	1	0

$T =$ (transitive closure)

	a	b	c	d
a	1	1	1	1
b	1	1	1	1
c	0	0	0	0
d	1	1	1	1

$R_k[i][j] = 1$ if

- $R_{k-1}[i][j] = 0$
- AND
- $R_{k-1}[i][k] = 1$
- AND
- $R_{k-1}[k][j] = 1$
- OR
- $R_{k-1}[i][j] = 1$

$R_0 = A$

örnek:

R_0 a

o [...]

R_1:

	a	b	c	d
a	0	1	0	0
b	0	0	0	1
c	0	0	0	0
d	1	1	1	0

$d \to b$: { $d \to d$: 1, $d \to b$: 1 } $\Rightarrow$ 1

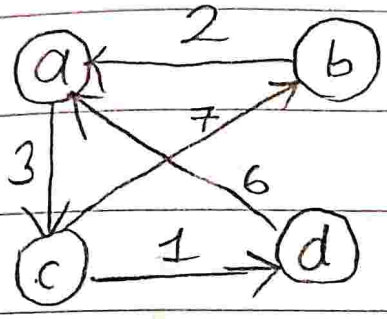
```
for k=1 to n do
    for i=1 to n do
        for j=1 to n do
            R_k[i][j] = R_{k-1}[i][j] OR
            [R_{k-1}[i][k] AND R_{k-1}[k][j]]
```

Shortest Path: Floyd-Warshall

Floyd-Warshall APSP

DP → (Dinamik Programlama)

distance[i][j] = min(distance[i][j], distance[i][k], distance[k][j])



$$W = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{pmatrix} 0 & \infty & 3 & \infty \\ 2 & 0 & \infty & \infty \\ \infty & 7 & 0 & 1 \\ 6 & \infty & \infty & 0 \end{pmatrix} \end{array}$$

D_0

$$\begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & & & \end{array} \\ \begin{pmatrix} 0 & \infty & 3 & \infty \\ 2 & 0 & \infty & \infty \\ \infty & 7 & 0 & 1 \\ 6 & \infty & \infty & 0 \end{pmatrix} \end{array} \Rightarrow$$

D_1

$$\begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{pmatrix} 0 & \infty & 3 & \infty \\ 2 & 0 & \mathbf{5} & \infty \\ \infty & 7 & 0 & 1 \\ 6 & \infty & \mathbf{9} & 0 \end{pmatrix} \end{array}$$

D_1

$$\begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} & b & & \end{array} \\ \begin{pmatrix} 0 & \infty & 3 & \infty \\ 2 & 0 & 5 & \infty \\ \infty & 7 & 0 & 1 \\ 6 & \infty & 9 & 0 \end{pmatrix} \end{array} \Rightarrow$$

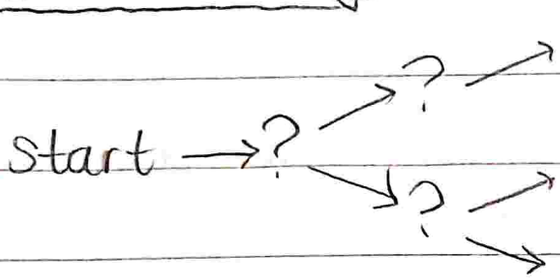
D_2

$$\begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{pmatrix} 0 & \infty & 3 & \infty \\ 2 & 0 & 5 & \infty \\ \mathbf{9} & 7 & 0 & 1 \\ 6 & \infty & 9 & 0 \end{pmatrix} \end{array}$$

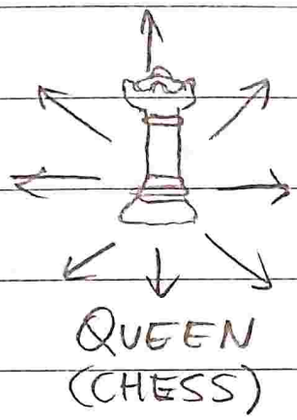
. . . .

Backtracking

Backtracking



1. TRY
2. CHECK
3. IF FAILS TRY ANOTHER OPTION!



N-QUEENS PROBLEM

Hiçbiri birbirine çakışmadan yerleştirme

1. BruteForce → Check ALL possible solutions

$\binom{64}{8} = 4,426,165,368$ positions to try...

2. One per row:

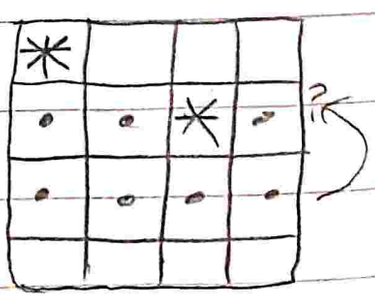
$8^8 = 16,777,216$ positions

3. One per row and per column

$8! = 40,320$ positions

4. BACKTRACKING

Only 2057 trials!

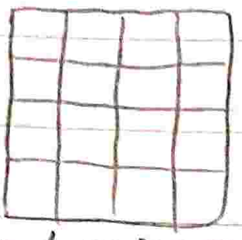


Brute Force	Backtracking
tries all possibilities	prunes invalid solutions early
no structure	recursive
always exponential (most of the time)	constraint driven
	still exponential but FASTER!

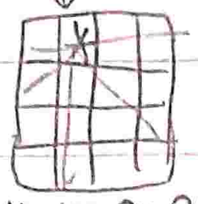
Backtracking

state space tree

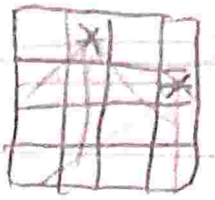
4 queen



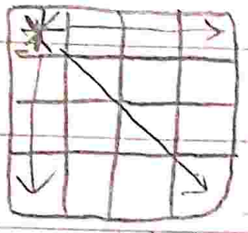
1st row → 1✓ 2*



2nd: 1✓ 2✓ 3✓ 4✓

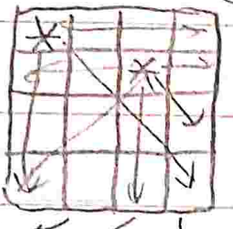


3rd: 1✓



2nd row: 1* 2* 3✓✗ 4✓✗

DEADEND! (BACKTRACK)

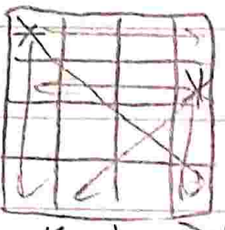


3rd row: 1* 2* 3* 4*

✗

DEAD END!

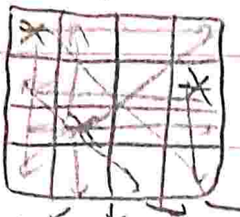
(BACKTRACK)



1* 2✓✗ 3* 4*

DEADEND!

(BACKTRACK)



4rd row: 1* 2* 3* 4*

✗

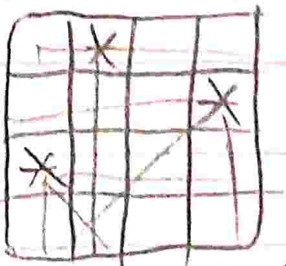
DEADEND!

(BACKTRACK)

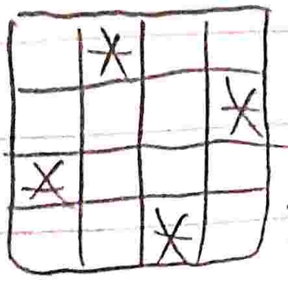
(Depth First Search)

Backtracking nasıl yapılabilir?

Stack, Recursion



4th: 1* 2* 3✓

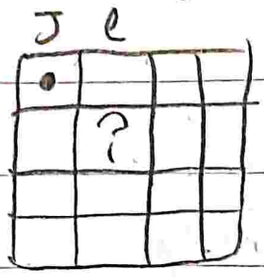


✓ SOLUTION

Backtracking

Diagonalleri basit kontrol etme yaklaşımı

M[i][j]
M[k][l]



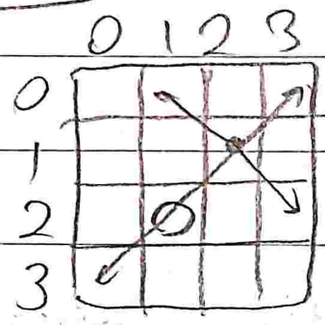
↓

they are on the same diagonal.

$$|j-l| = |i-k|$$

$i-j = k-l$
$i+j = k+l$
$j-l = i-k$
$j-l = k-i$

herhangi birini sağlar



$i=1\ j=2\ k=2\ l=1$

$|2-1| = |1-2|$

0..7
N-1

queen: 0 ..... N-1

| 1 | 3 | 0 | 2 |

(hangi sütunda!)

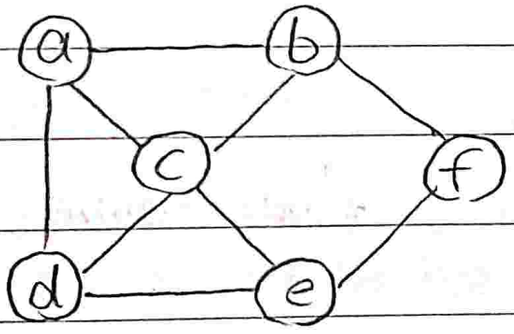
```
Backtrack (row)
{
   if(row == N)
        print queen
   end if
   for col=0 to N-1 do
        if IsSafe(row, col)      // boş mu?
             queen[row] = col
             Backtrack(row+1)
             queen[row] = -1     // (undo choice)
        end if
   end for
END Backtrack
```

Backtracking

```
Is Safe (row, col)
{
    for i=0 to row-1 do
        if queen[i] == col
            return 0;        // false!
    if abs(queen[i]-col) == abs(i-row)
        return 0;
    end for
    return 1;                // yerleşebilir!
}
```

Hamiltonian-Circuit Problem

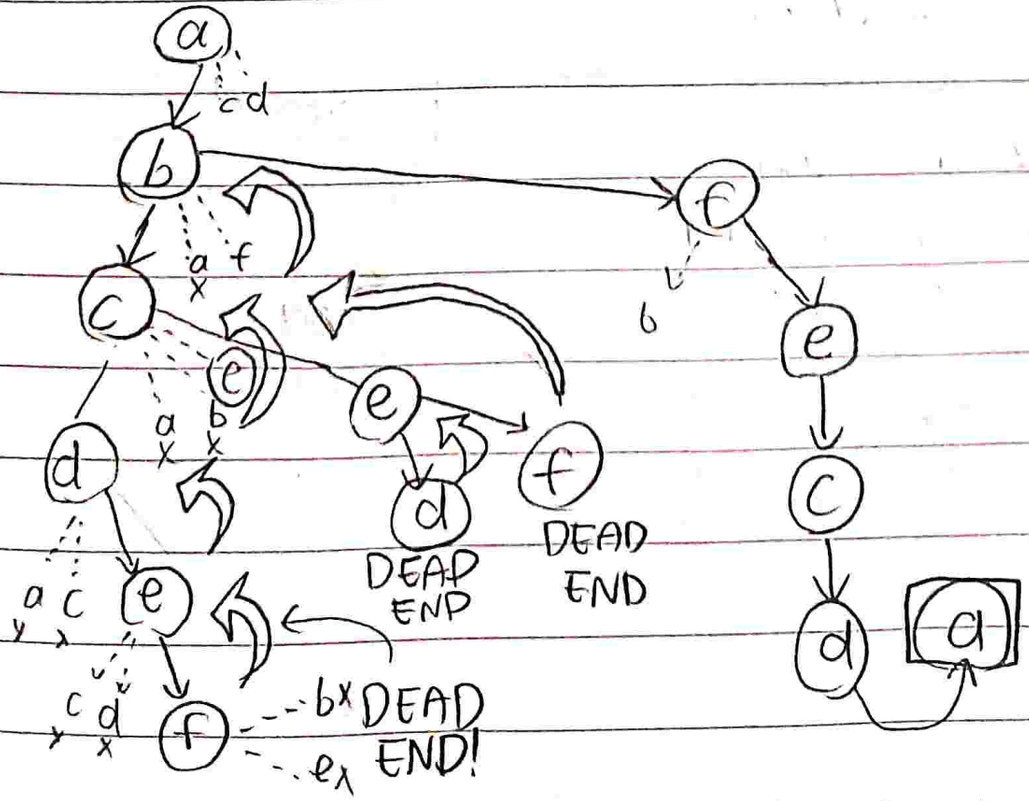
her düğüme sadece 1 defa uğrayıp gezilebilir mi?



Graph(V, E)
visit every node
exactly ONE TIME!

also check:
3 coloring problem

state space tree

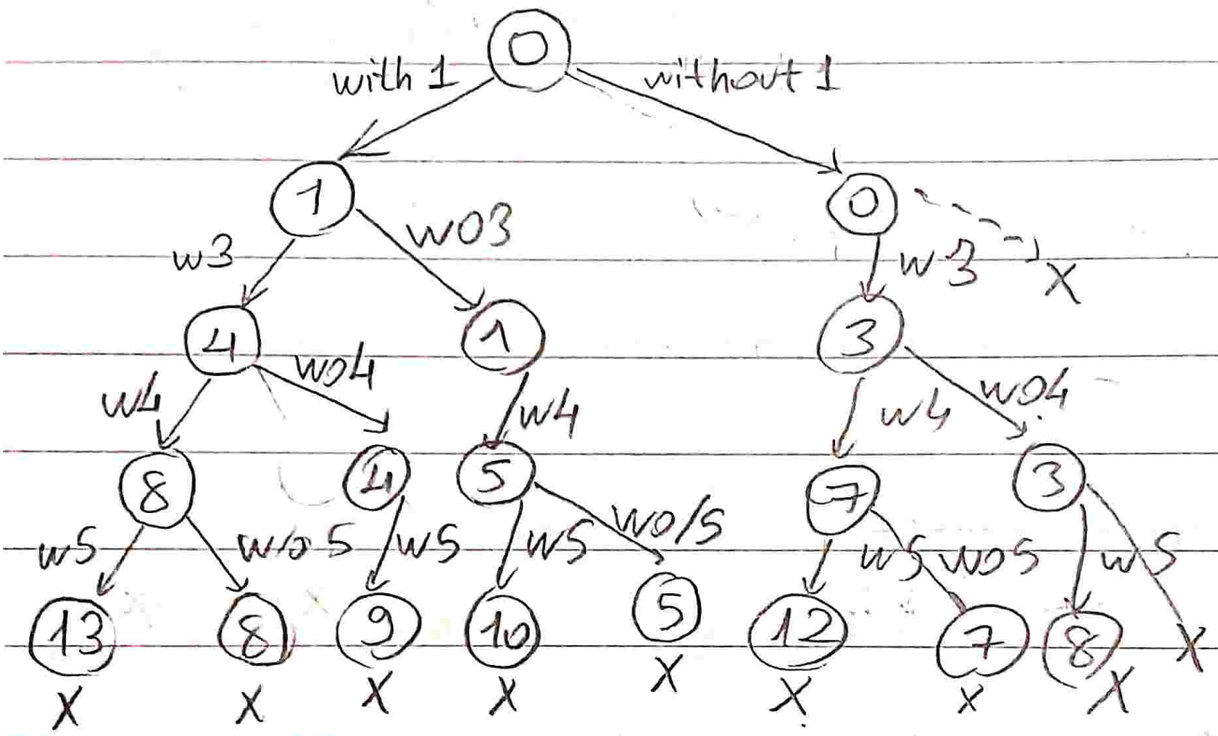


Backtracking

SUBSET SUM PROBLEM

$S = \{S1, \dots, Sn\}$ positive integer
Ascending ↑ (d) → can we get this?

$S = \{1, 3, 4, 5\}$ $d = 11$?



ASSIGNMENT PROBLEM

Bir işyerinde bir işe 4 kişi başvuruyor, ve her kişinin işler için istediği ücretler farklı. 4 kişi en az parayla nasıl yerleştirilir?

N → number of possible assignments : Brute Force : N!
worst case ∈ O(n!)

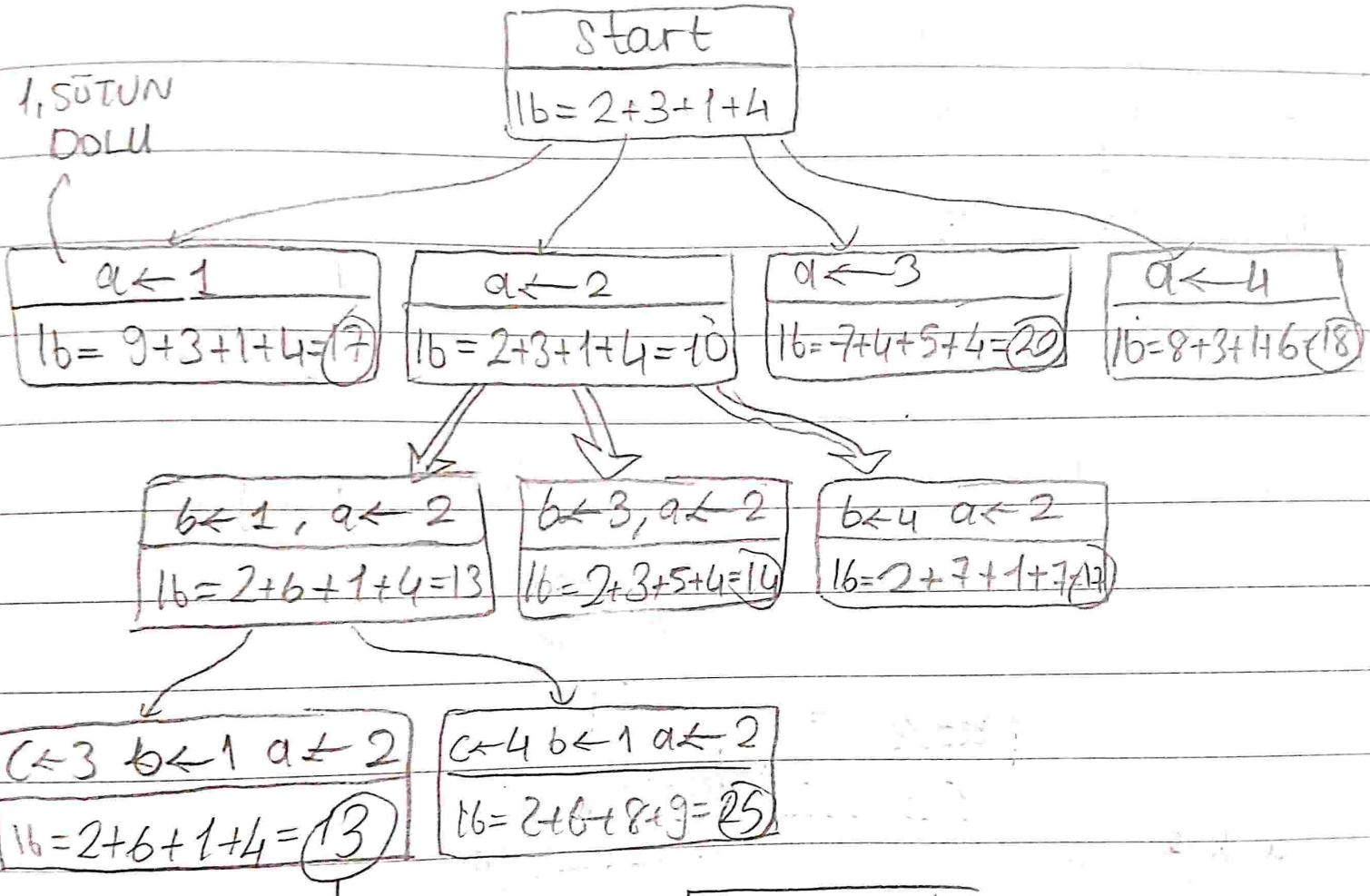
PRUNING may be much better

	J1	J2	J3	J4	
C =	9	*2	7	8	1
	6	4	*3	7	2
	5	8	*1	8	3
	7	6	9	*4	4

Start
lower bound: lb = 2+3+1+4 = 19
↓
sığmadı

Backtracking: Assignment Problem

state space tree



13 in altına inmek imkansız

SOLUTION

a←2, b←1, c←3, d←4

13

inebilseydik en düşük ikinci değere BACKTRACKING

Backtracking

Knapsack Problem

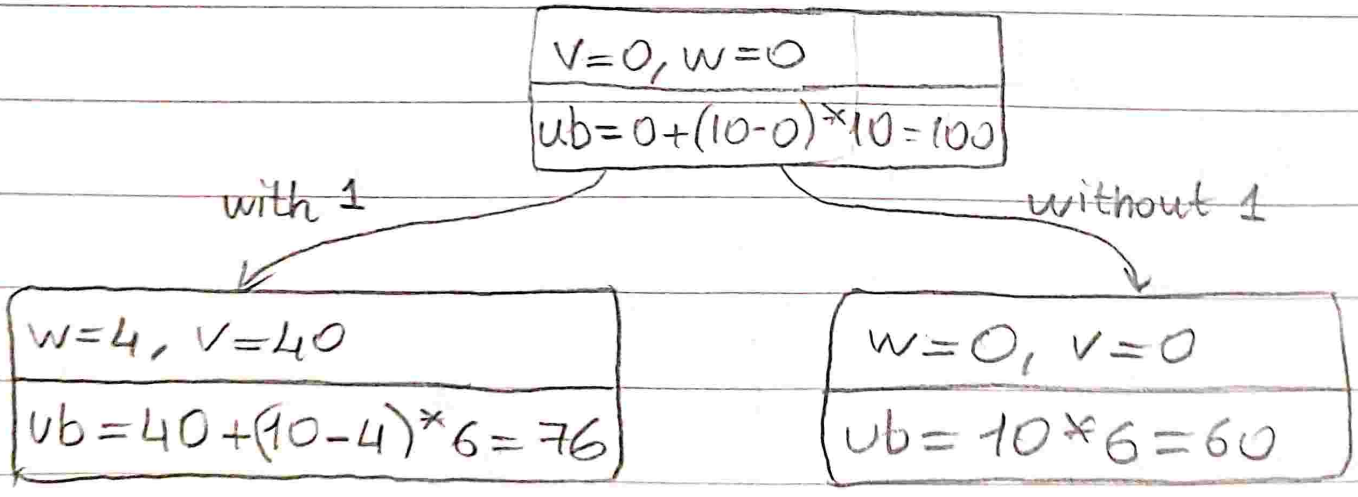
item	weight	value	v_i/w_i → birim kazanç
1 →	4	40	10
2	7	42	6
3	5	25	5
4	3	12	4

(sıralı)

$W = 10$ //

Hangi item'leri alacağını **Branch and Bound** ile çözün.

state space tree
of BRANCH AND BOUND



// kalanını kendiniz deneyin!

Back to Algorithm Analysis

polynomial n^2, n^5, n^{10000} → efficient

np: non deterministic polynomial

tractable problems can be solved in polynomial time.

↳ reverse: non-tractable/unsolvable

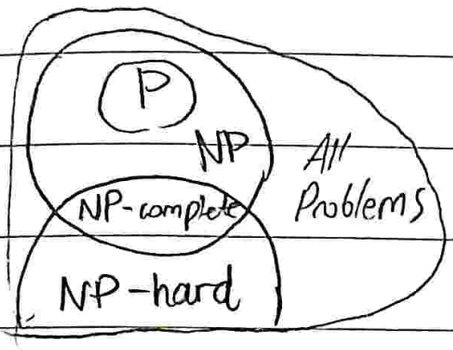
np np-hard np-complete

deterministic vs. non-deterministic algorithms

↳ guessing stage

generates random candidate solution

verification stage → polynomial part



(slaytta kalan her şey var)

Decision problemi mi?

↳ NP-complete olamaz!

↳ Evet

↳ Çözüm polinomsal zamanda doğrulanıyor mu?

↳ Hayır: NP değil

↳ Evet: NP

↳ NP-complete bir problem buna indirgenebiliyor mu?

↳ Hayır: NP

↳ Evet: Complete-NP

Problem:

Optimizasyon: NP-hard

Decision: NP-complete

Örn: Traveling Salesman → En kısa yol nedir? → Optimizasyon (NP-hard)

↳ toplam maliyeti <= K olan bir HC var mı? → Decision → NP