RBT:

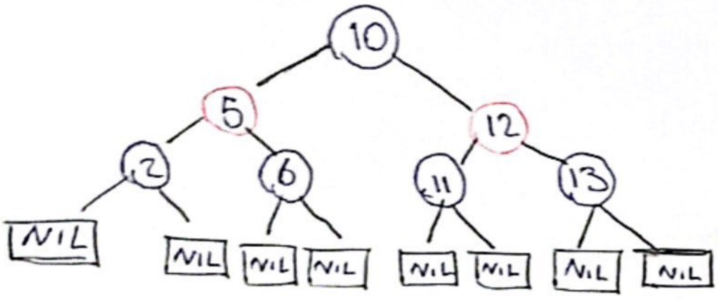
① each node is either Red - Black

② tree-root is always Black

③ leaves(NIL ~~are~~ nodes) are Black

④ if a node is Red then its children are Black

⑤ for each node, all possible path from node → to leaves will have the same amount of Black nodes.

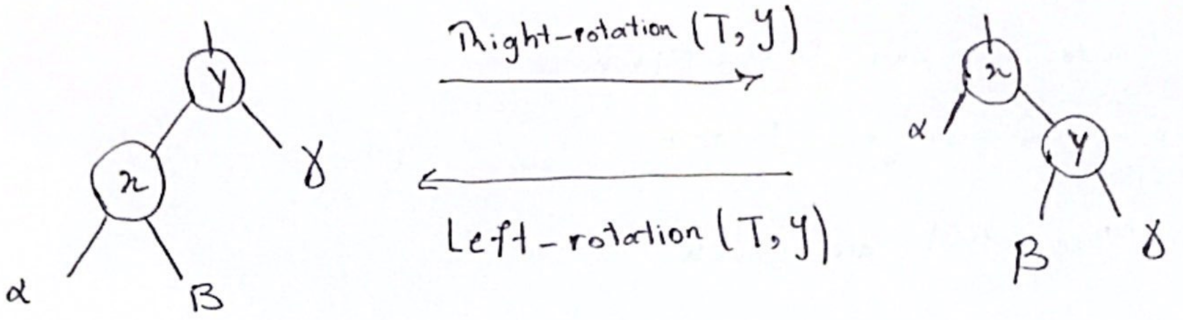


Black height: number of black nodes frome node x → to leaves exept x is called $(bh(x))$ Black height of x

- each ~~root~~ subtree with root x, it at least have $2^{bh(x)} - 1$ internal nodes

- A RBT with n internal nodes has height at most $2\log(n+1)$

rotations:



- NIL (null) node: we have a node which we call Null node, and the pointer to it is called T.NIL (T.Null)

RB-Insert-Fixup(T, Z)

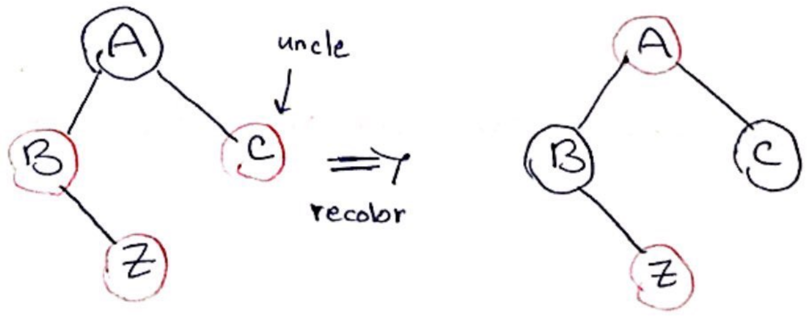
cases:

~~case 1: y is uncle of Z~~

Case 0: Z = T.root

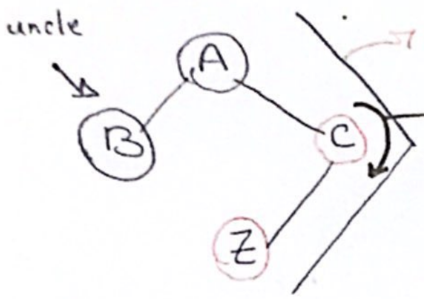
color Z to Black

case 1: Z.uncle = Red

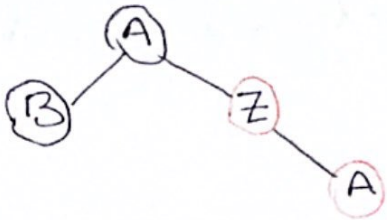


Z.uncle = Black
Z.p = Black
Z.p.P = Red

case 2: Z. uncle = Black and Triangle is formed

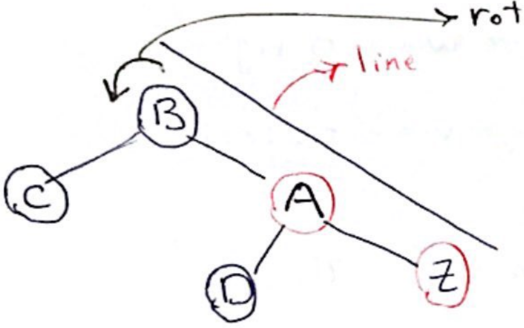


→ rotate in the opposite dir of Z

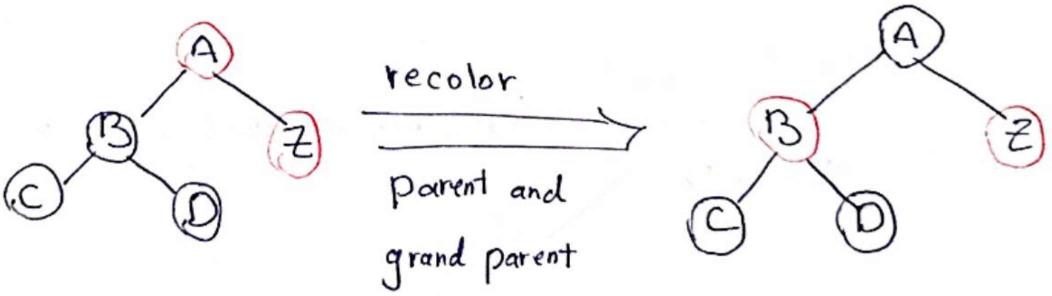
=> 

case 3: Z. uncle = Black and Line is formed

→ rotate in the opposite dir of Z



Z and Z.p are both left(right) child

=> 

RB_ Deletion cases

Case 0: node x is Red

1. color x Black

Case 1: x is Black and its sibling w is red

1. w → Black
2. x.p → Red
3. rotate x.p
   - x = x.p.left → left rotation
   - x = x.p.right → right rotation
4. change w
   - x = x.p.left → w = x.p.right
   - x = x.p.right → w = x.p.left
5. with x and new w decide on cas 2, 3, 4

Case 2: x = Black and w = Black and both w childrens are Black

1. w ← Red
2. x = x.p
   - if new x = Red → x = Black
   - if new x = Black → decide on case 1,2,3,4 + we have a new w

Case 3: x and w are both Black
- x = x.p.left, w.left = Red and w.right = Black
- x = x.p.right, w.left = Red and w.right = Black

1. color w's child Black
   - x = x.p.left → w.left = Black
   - x = x.p.right → w.right = Black
2. color w Red

3. rotate w
   - x = x.p.left → right rotation
   - x = x.p.right → left rotation

4. change w
   - x = x.p.left → w = x.p.right
   - x = x.p.right → w = x.p.left

5. go to case 4

case 4, x and w are Black and
- x is left child, w.right = Red
- x is right child, w.left = Red

1. w.color = x.p.color

2. x.p = Black

3. color w's child Black
   - x = x.p.left → w.right = Black
   - x = x.p.right → w.left = Black

4. rotate x.p
   - x = x.p.left → left rotation
   - x = x.p.right → right rotation

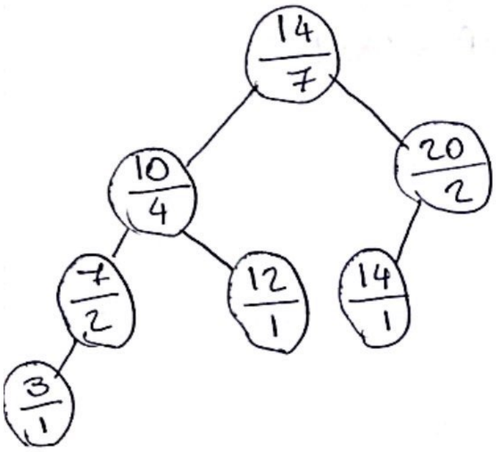
Augmented Data structure

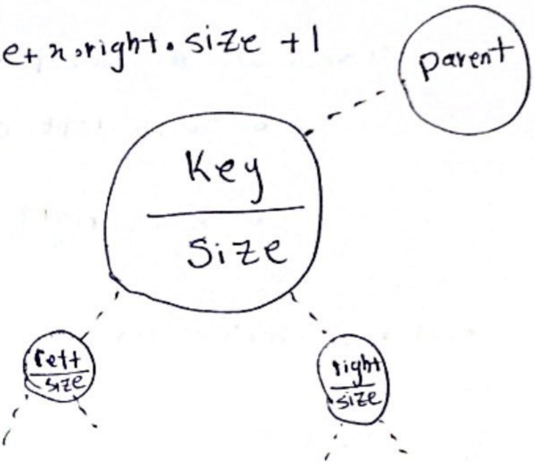
- the process of taking an Existing DS and castomizing it a little bit to fit your needs.

find K number from n number but n changes.

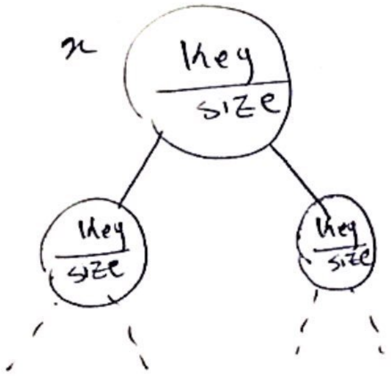
- order-static-tree
  - its an Augmented RBT
  - each Node has another data named "Size"

$x.size = x.left.size + x.right.size + 1$





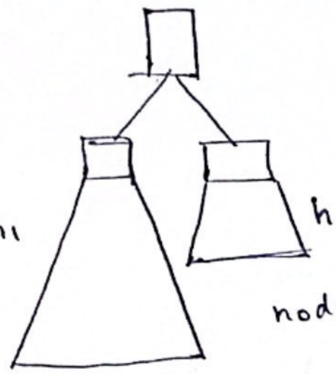
- find i'th smallest number in OST



$rank(x) = x.left.size + 1$

- if a subtree has height of h_1, the other subtree has a maximum height of $2h_1$

number of nodes in subtree 1 $= 2^{h_1}$ " " in subtree 2 $= 2^{2h_1}$



$h = 2h_1 + 1$

nodes $= 2^{h_1} = y$

$y + y^2 + 1 = n \rightarrow y = ?$

nodes $= 2^{2h_1} = y^2$

برای این که OS-select و OS-Rank کار کنند باید بعد insert و delete اندازه (size) را Fixup کنیم .

در insert به size گره ها از عنصر جدید تا T.root ، $\underline{1}$ واحد اضافه می کنیم

left_rotation:

```
//...
y.size = x.size
x.size = x.left.size + x.right.size + 1
```

in right_rotation switch $\underline{x}$ and $\underline{y}$

در delete از node پاک شده تا T.root باید size همه node ها را ۱ واحد کم کنی

Interval Trees:

an Augmented DS which use RBT and store some sort of Range

closed Interval $[t_1, t_2]$

open " (t_1, t_2)

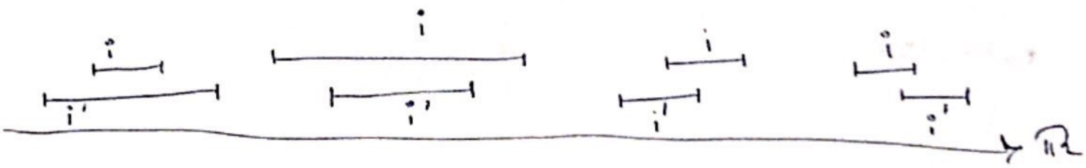
half open " $(t_1, t_2]$ || $[t_1, t_2)$

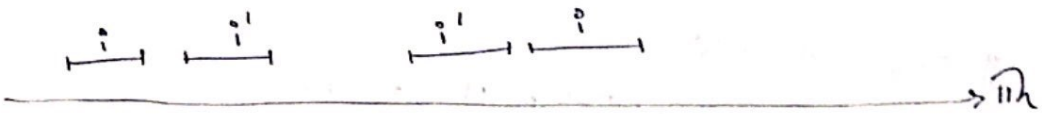
each Interval $\underset{[t_1, t_2]}{}$ is an object named i:

$i.low = t_1$ $\quad$ $i.high = t_2$

if $i \cap i' \neq \emptyset \Rightarrow$ overlap(i, i') == true

$\Rightarrow i.low \leq i'.high$ and $i'.low \leq i.high$





overlap has 3 states:

① They have overlap
② i is on left side of i'
③ i is on right side of i'

Augmented RBT

node x: $x.\overset{\text{interval}}{\text{int}}.\text{low}$, $x.\text{int}.\text{high}$] interval Range

$x.\text{max}$ → maximum number of high in x subtree

Interval-Insert (T, x)

یک گره x که ویژگی $x.int$ آن شامل یک بازه است در درخت درج می‌کند

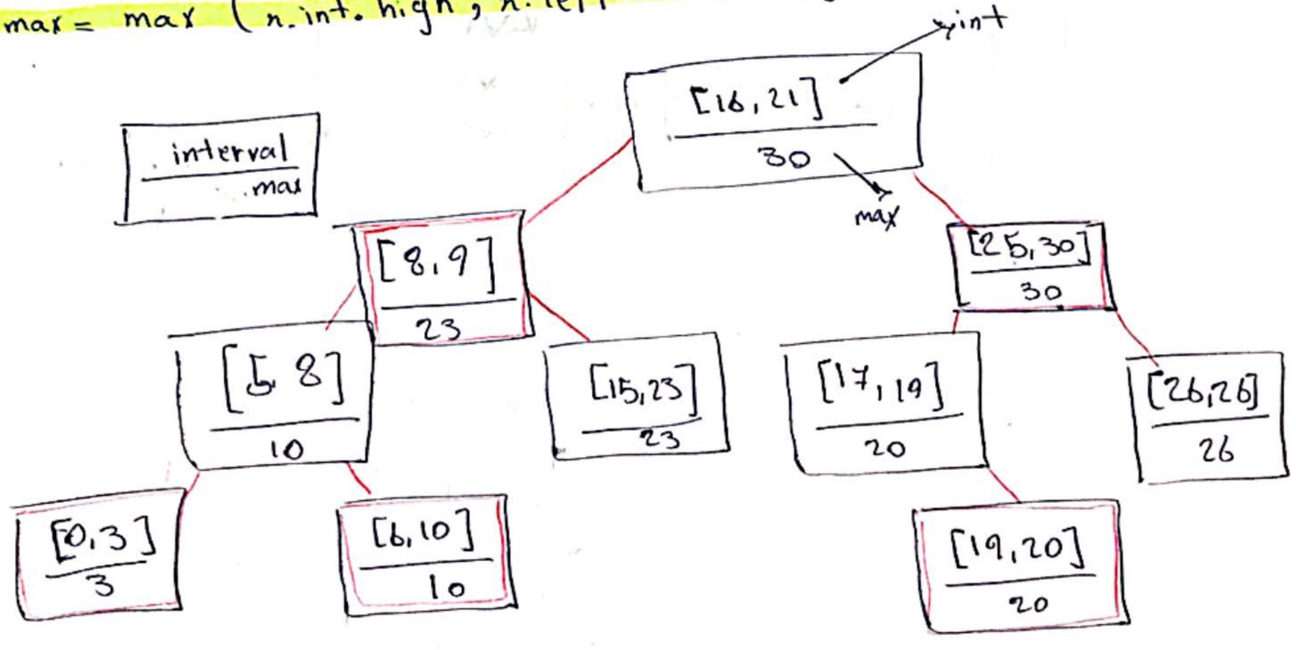
Interval-Delete (T, x)

یک گره x از درخت Interval حذف می‌کند

Interval-search (T, i)

اشاره‌گر به یک گره x در درخت T که $x.int$ با i همپوشانی (overlap) دارد

$x.\text{max} = \max(x.\text{int}.\text{high}, x.\text{left}.\text{max}, x.\text{right}.\text{max})$

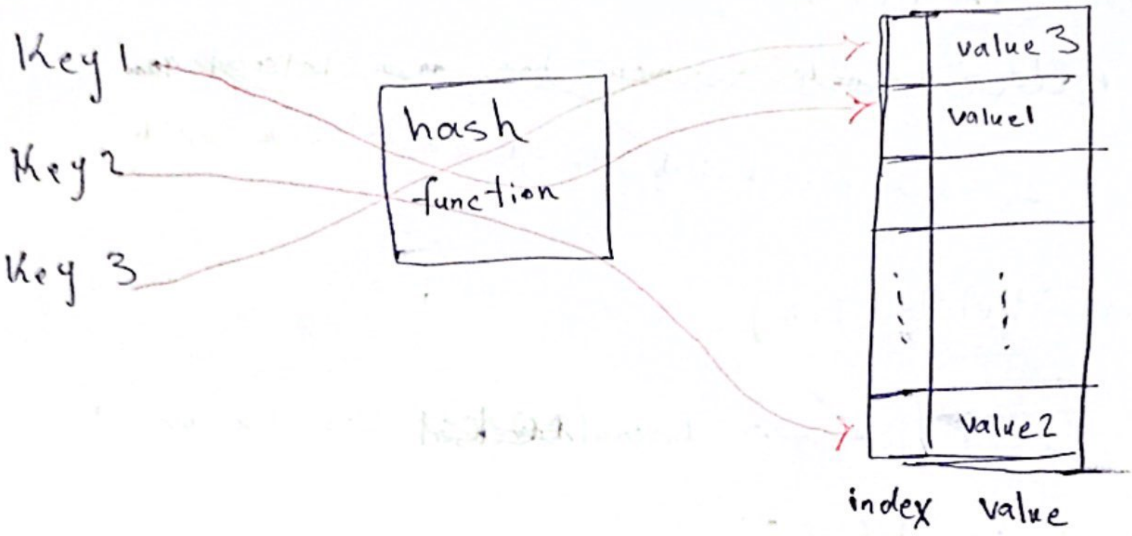


Serch [22, 25] ①

[11,14] ②

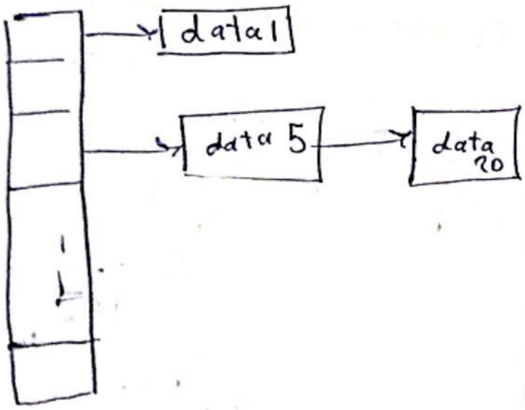
HashTable: is a DS which stores data in an associated manner.

in a hashTable, data is storred in array format, where each data value has its own unique index value



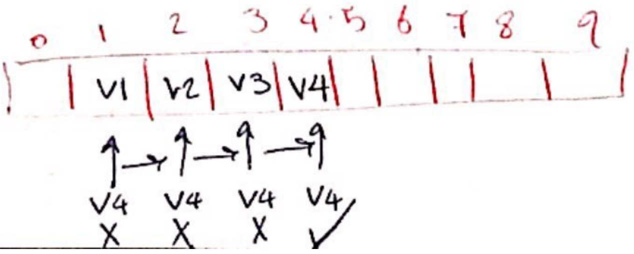
how to handle collisons?

chainig



open addressing

		hash(key)
K1	V1	1
K2	V2	2
K3	V3	3
K4	V4	1

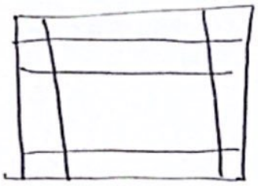


Graph :

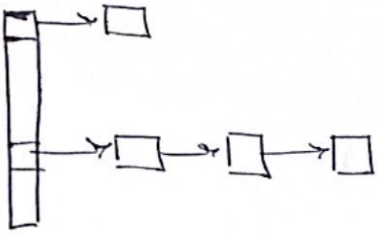
$G = (V, E)$

(رئوس) vertics

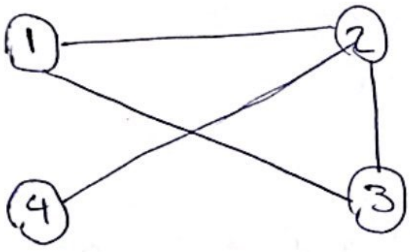
edges (یال)



… Adjacency matrix



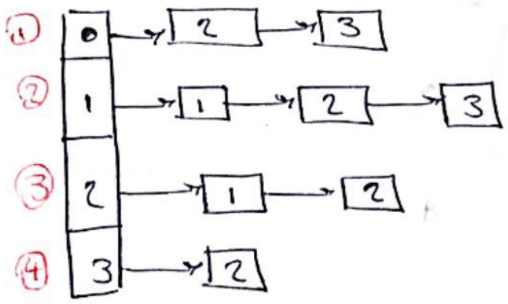
Adjancency List



Adj [4][4]

		①	②	③	④
		0	1	2	3
①	0	0	1	1	0
②	1	1	0	1	1
③	2	1	1	0	0
④	3	0	1	0	0

Adj List [4]

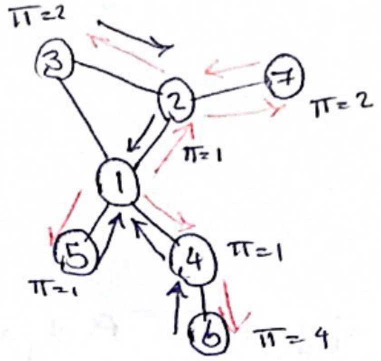


DFS: Depth first search

از راس شروع از طریق یال های که وجود دارد می‌رویم به یکی از همسایه های visited ا، و راس جاری ما آن می‌شود، سپس این کار را برای راس جاری تکرار می‌کنیم.

اگر همه همسایه ها visit شده باشند به parent آن node برمی‌گردیم (backtrack)

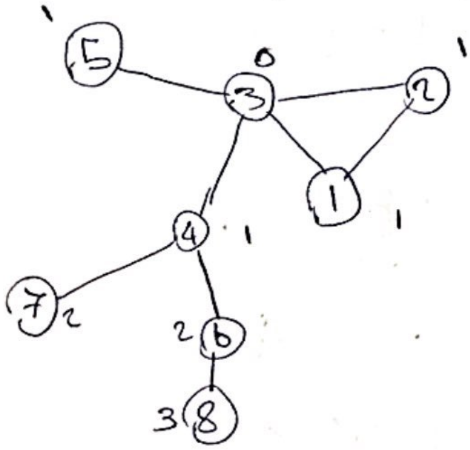
$G = (V, E)$



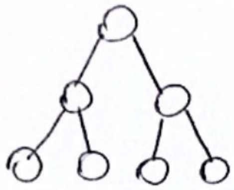
۴ فقط برای گراف همبند کار می‌کند

BFS: Breadth first search

از راس شروع همه همسایه های آن را visit می‌کنیم و به آن ها برچسب 1 می‌دهیم بعد به همسایه های آن‌ها 2
د...



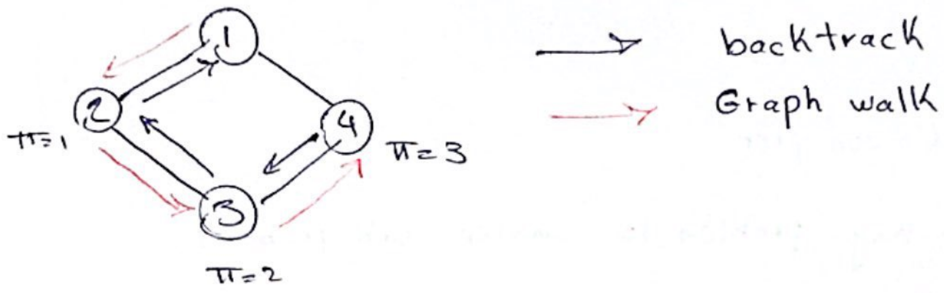
BFS shows shortest path from starting node usage:

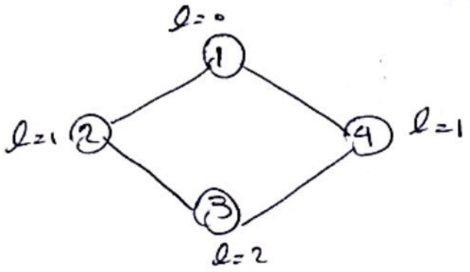


پیمایش درخت حالات
با DFS

BFS → shortest path

- we use DFS : Detect cycle in a Graph or directed Graph





DFS shows - either there is a cycle or not
it doesn't show how many cycles de we have

classic ways of designing Algorithms

① divide and conquer

② dynamic programing

③ Greedy Algorithm

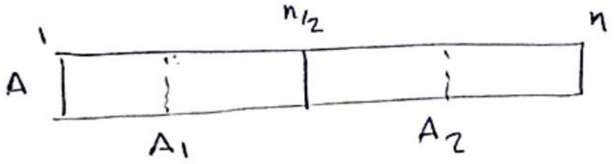
④ Branch and Bound Algorithm / back tracking

① divide and conquer

1) breaking problem to smaller sub problems

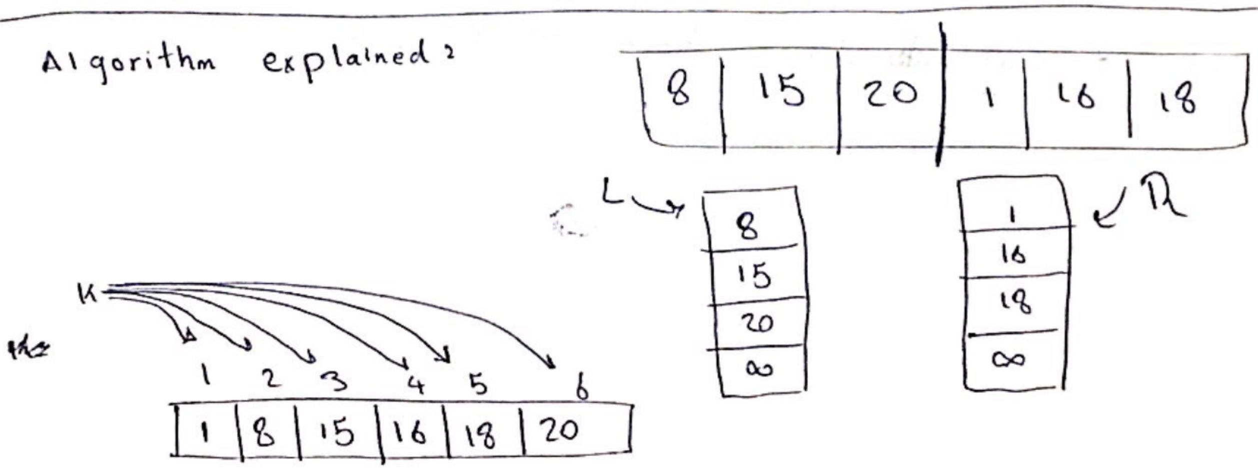
2) recursive : حل بازگشتی زیر مسئله

3) combine sub problems



merg sort

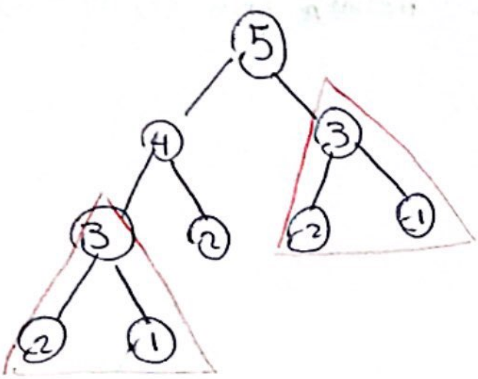
Algorithm explained :



Dynamic programing:

① یک حافظه در نظر می‌گیرد که مسائل تکراری را به خاطر می‌سپارد و بعد استفاده می‌کند

```
fibonacci (n)
    if  n=1 || n=2
        return 1
    return  fibonacci(n-1) + fibonacci(n-2)
```



Solution ?

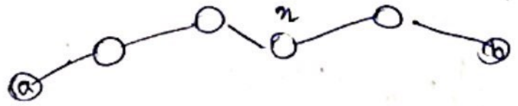
A [n] =

1	1	2	3	~	n

Fill the array each step a new n is calculated

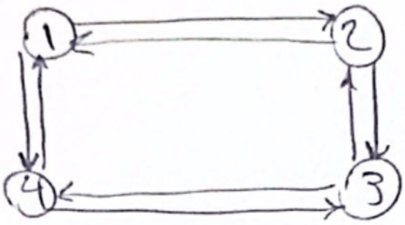
② optimal sub structure

consider shortest path between 2 nodes



each substructure of shortest path is still a shortest path

if x is the shortest path between a,b ~~the~~ part of path between x and a is still shortest path



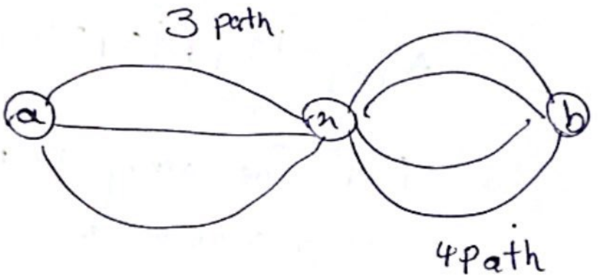
①→②→③→④

longest path from ① to ④

①→②

longest path from ① to ②

①→④→③→② longest path ~~between~~ from ① to ②



~~Back~~ Branch and bound:

پرانتز گذاری ضرب ماتریس‌ها:

① all possible moves.

count = عدد کاتالان

$(M_1 \times M_2 \times \_\_\_ \times M_k)(M_{k+1} \times M_{k+2} \times \dots \times M_n)$

$$P(n) = \sum_{k=1}^{n-1} P(k) \times P(n-k)$$

عدد کاتالان $= \frac{1}{n+1}\binom{2n}{n}$

راه بهتر → Dynamic programing

$$P_{i,j} = M_i \times M_{i+1} \times \text{—} \times M_j$$

$C_{i,j}$ = تعداد ضرب های زیرمسئله 1 + " " 2 + تعداد ضرب های ضرب آخر

تعداد ضرب ها

$$= C_{i,k} + C_{k+1,j} + d_{i-1} \times d_k \times d_j$$

$$C_{i,j} = \min \left\{ C_{i,k} + C_{k+1,j} + d_{i-1} \times d_k \times d_j \right\}$$

$$i \le k < j$$

M_1	M_2	M_3	M_4
5×100	100×10	10×50	50×25
d_0 d_1	d_1 d_2	d_2 d_3	d_3 d_4

i \ j	1	2	3	4
1	0	k=1 50000		
2		0	50000	
3			0	12,500
4				0

$$\dot{c}_{1,2} = \min \left\{ C_{1,k} + C_{k+1,2} + d_0 d_k d_2 \right\}$$

$$1 \le k < 2$$

$$k=1 \rightarrow C_{1,2} = C_{11} + C_{22} + d_0 d_1 d_2$$

$$= 50,000$$

$c_{3,4}$ = ~

$c_{2,3}$ = —

$$C_{1,3} = \min \left\{ c_{1,k} + C_{k+1,3} + d_0 d_k d_3 \right\}$$

$k=1 \longrightarrow c_{13} = 75000$ ⇒ $(m_1 \times m_2) \times m_3$

$k=2 \longrightarrow c_{13} = 7,500$

k = 2

7,500

$C_{2,4}$ —

$C_{1,4}$ =

$m_1 \times m_2 \times m_3 \times m_4$

$$C_{14} = \min \left\{ C_{1K} + C_{K+1,4} + d_0 d_K d_4 \right\}$$

$1 \le K < 4$

$K = 1 \longrightarrow C_{14} = 50{,}000$

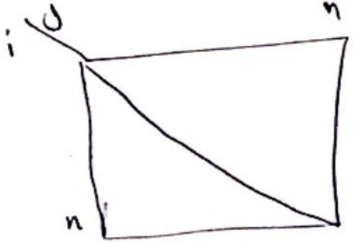
$K = 2 \longrightarrow C_{14} = 18{,}750$

$\boxed{K = 3} \longrightarrow C_{14} = 13{,}750$

$$\Rightarrow \underbrace{(m_1 \times m_2 \times m_3)}_{\text{از قبل داریم}} m_4$$

$(m_1 \times m_2) m_3$

$$\Rightarrow ((m_1 \times m_2) \times m_3) \times m_4 \Rightarrow \text{Best case}$$



نصف ماتریس باید پر شود یعنی $O(n^2)$

در هر مرحله از $n-1$ حالت باید min گرفته شود

$$O(n^2) \times O(n) = O(n^3)$$