UNIT I

* ADT

- Abstraction of a data structure (D + O)
- → Data Stored
  → Operations on the Data
  → Error Conditions

1) LISTS

- Finite Sequence of data items
- Create, Insert / Remove, Find, Find Next / Prev

2) STACK

- LIFO
- Push Peek (Top)
  Pop ↓F (or) ↓E

3) QUEUE

- FIFO
- Enqueue ↓F
  Dequeue ↓E
  Get Front

* Stable Sorting

- Two elements with same value appear in same order in output as in input

(10) (20) [20] (15)

(10) (15) (20) [20]

* RADIX Sort

{ 455, 61, 63, 15, 5, 27 }

→ Find maximum element (455)
→ Number of digits in it = 3 = Passes required

→ Sort from LSD to MSD
→ unit, ten, hundredth place

{ at, it, mike, can }
{ atAA, itAA, mike, canA }

* String : Same length = LSD to MSD

* MSD to LSD when not same length and use BUCKETS

* radix == base

* BUCKET SORT

- For uniform input distribution
- Separates elements into buckets / bins
- Then that bin is sorted

UNIT 2

* Topological Sort

- DAG : Directed Acyclic Graph

- Used to indicate precedences among events

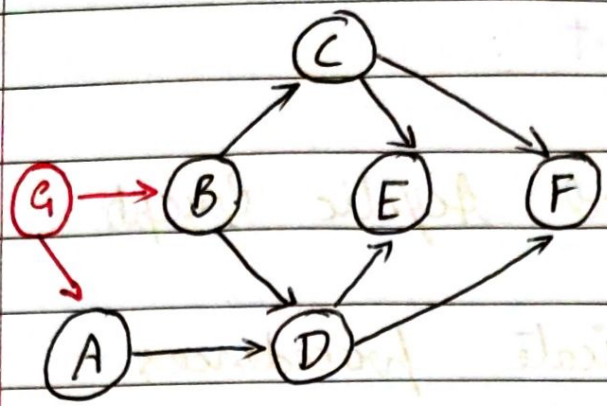
- An edge from u to v
  $\Rightarrow$ u must be done before v

- TS of a DAG is the linear ordering of all its vertices such that for any edge (u, v) u appears before v

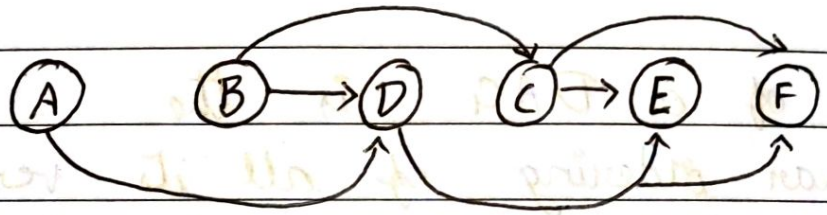
Idea

→ Create a "ghost node" that connects to all the vertices that don't have an incoming edge

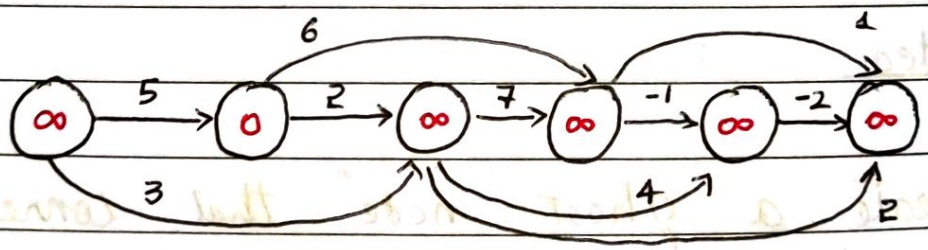
→ Perform BFS

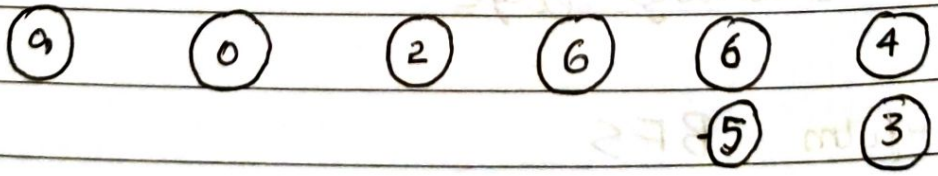


→ [G, A, B, D, C, E, F,]



* Shortest Path in DAG



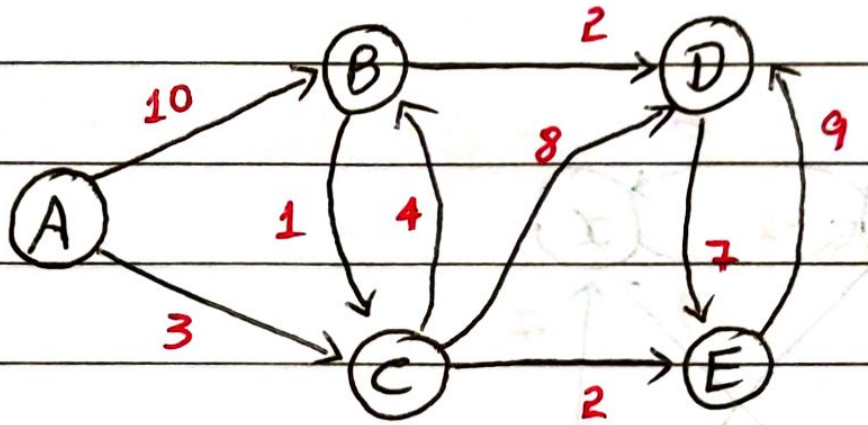


- For each node
- If it directed to another node
- Change that node's value if its min

O(V+E)

* DJIKSTRA Algorithm

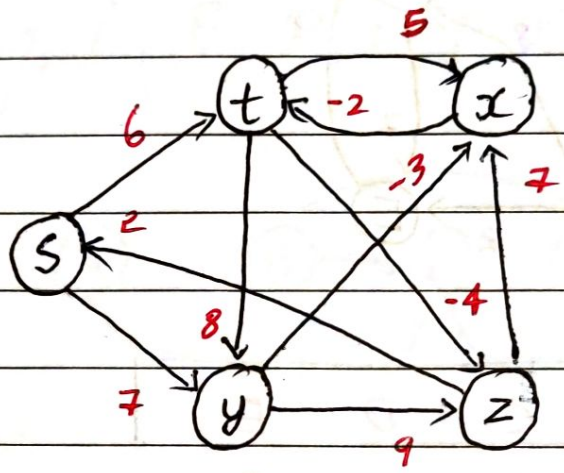
- Greedy
- No negative cycle
- $O(V^2 + E)$



Q	A	B	C	D	E
	(0)	∞	∞	∞	∞
		10	(3)		
		7		11	(5)
		(7)		11	
				(9)	
	0	7	3	9	5

* BELLMAN-FORD Algorithm

- Dynamic Programming
- Works with negative cycles
- $|V| - 1$ iterations
- $O(|V| \cdot |E|)$



I

s	t	x	y	z
0	∞	∞	∞	∞
0	6	∞	7	∞
0	6	11	7	2
0	6	11	7	2
0	6	4	7	2
0	6	4	7	2

II

⋮

III

⋮

IV

* Minimum Spanning Tree

For a graph G (V, E)
acyclic subset of edges that connects
all vertices V, whose total weight is min

* Prim's Algorithm $O(|V|^2)$

* Kruskal's Algorithm $O(|V||E|)$

* Ford - Fulkerson Algorithm

- Maximum flow from source node (s) to sink node (t)

→ Find augmenting path from s to t

  → path whose edges are either,
  → non-full & forward,
    non-empty & backward

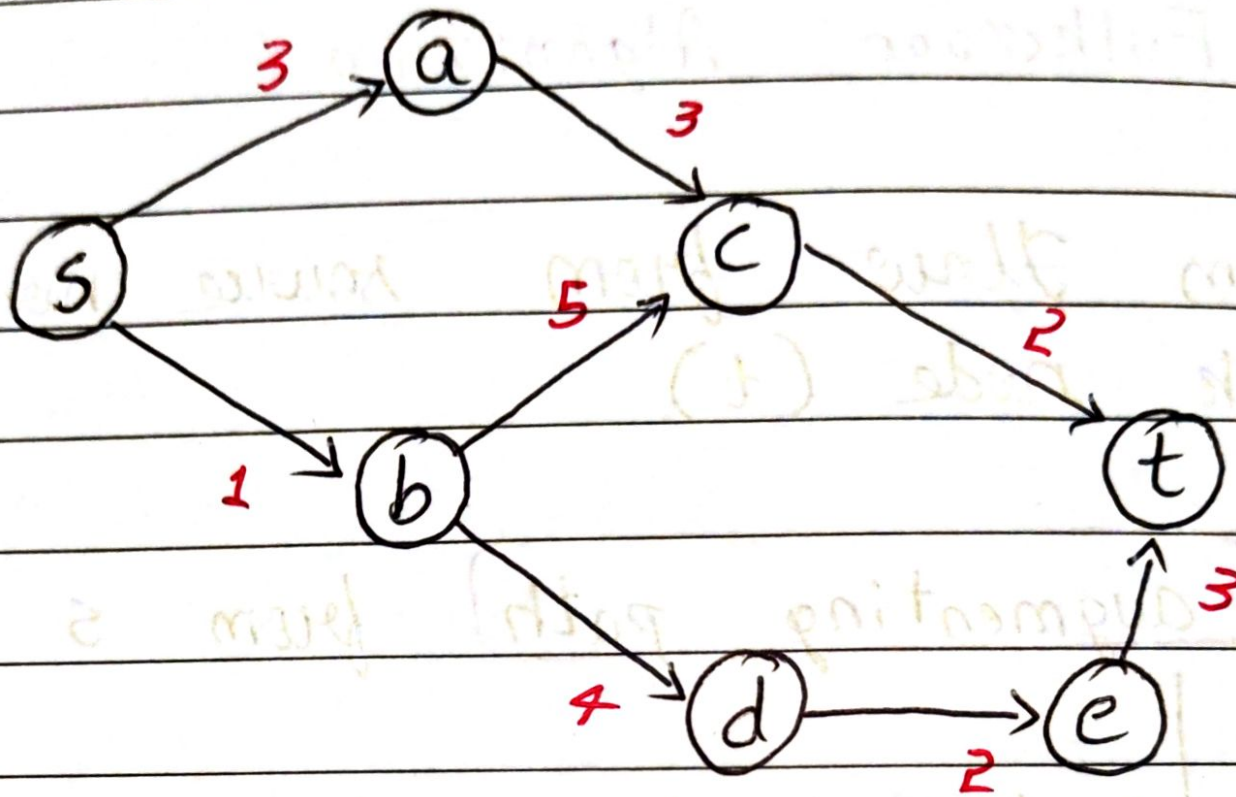
→ Compute bottleneck capacity

→ Augment each edge & total flow

* augmenting path is just a path in which the water can flow ($c - f > 0$)

* When there's no aug path, max flow is reached

* bottleneck capacity : Smallest Edge capacity


s
a
b
c
d
e
t
3
3
5
2
1
4
2
3

UNIT 3

* Miller Rabin Primality Test

- Probabilistic Primality Test
- Checks whether a specific property (which holds for prime numbers) is applicable for number under the test

– Algorithm

i) Find $n - 1 = 2^k \times m$

ii) Choose a, $1 < a < n - 1$

iii) Compute b_0, $b_0 = a^m \% n$

$b_1 = b_0^2 \% n$

$b_i = b_{i-1}^2 \% n$

Compute till $+1$ (Composite)

-1 (Prime)

$n = 561$

$2^4 \times 35$

i) $n - 1 = 560 \Rightarrow$

0	1	2	3	4
560 →	280 →	140 →	70 →	35 → $\frac{35}{2}$

ii) $1 < a < 560 \Rightarrow a = 2$

iii) $b_0 = a^m \% n = 2^{35} \% 561 = 263$

$b_3 = +1$

$2^{35} \div 561 = \text{ans}$

$2^{35} - \text{ans} \times 561$

$n = 341$

1) $n - 1 = 340$

$n - 1 = 2^k \times m$

$340 = 2^2 \times 85$

2) $a = 2 \quad 1 < a < n - 1$

3) $b_0 = a^m \% n$

$b_0 = 2^{85} \% 341$

$b_0 = 4$

$b_1 = (b_0)^2 \% 341$

$2^{85} \% 340$

$2^1 \% 340 = 2$

$2^2 \% 340 = 4$

$2^8 \% 340 = 2^2 \times 2^2 \times 2^2 = 4 \times 4 \times 4 = 64 \% 340$

$= 64$

$2^{85} \% 340 = 2^5 \times 2^2 = 2^2 2^2 2^2 2^2 2^2 = 1024 \% 340$

$= 4$

$2^{10} \% 340 = 2^5 \times 2^5 = 1024 \times 1024 = 1048576$

$2^{10} \% 340 = 16$

$2^{85} = 2^{10} \times 2^{10} \times 2^{10} \times 2^{10} \times 2^{10} \times 2^{10} \times 2^{10} \times 2^{10} \times 2^5$

$=$

$2^{20} \% 340 = 2^{10} \times 2^{10} = 16 \times 16 = 256 \% 340 = 256$

$2^{40} \% 340 = 256 \times 256 = 65536 \% 340 = 256$

$2^{85} = 2^{40} \times 2^{40} \times 2^5 = 262144 \% 340$

$= 4$

UNIT 4

* TRIE

- Derived from "retrieval"
- Similar to binary search tree
- BUT each node has only ONE outgo edge

COMPUTER

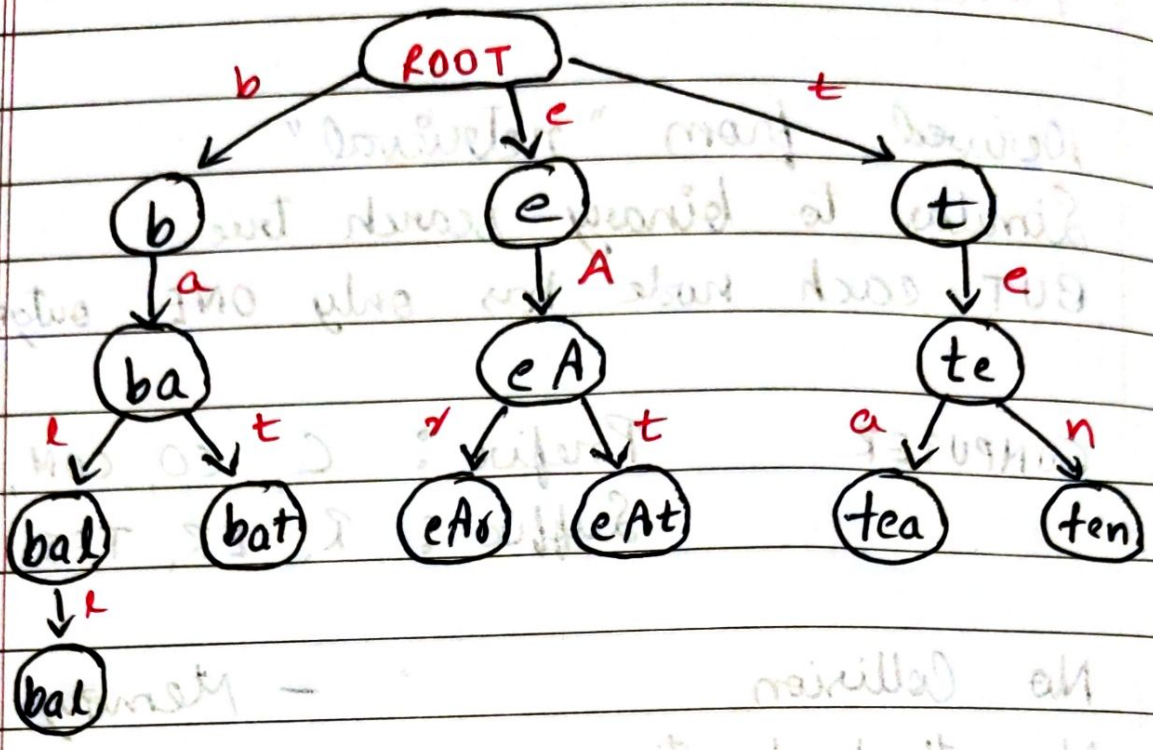
Prefix : C, CO, COM, COMP
Suffix : R, ER, TER, UTER

+ No Collision
+ No Hash function
+ Prefix based Search

- Memory

* Structure of Tree
* Root Node is Empty String
* Node = String, Edge = Character

* ball, bat, eAt, eAt, tea, ten



* String termination '\0'

* Find Insert Delete

Time to F, I, D = O(k)

Time to build = O(Nx)

N → No of string

x → Average length of string

UNIT 5

* KMP

t = ababcabcababa
p = abab

i	1	2	3	4
p	a	b	a	b
π(i)	0	0	1	1

*
a b a b c a b c
a b a b

k = matches = 4
π(4) = 1

Shift by k - π(k) = 4 - 1 = 3

*
a b a b c a b c
a b a b

k = 1
Shift by 1

DSA

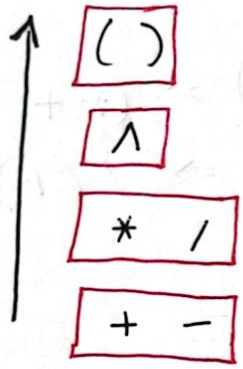
* Notations

→ Infix ⟶ a + b

→ Prefix (Polish) ⟶ + a b

→ Postfix (Reverse Polish) → a b +

* Preference



* Infix to Postfix

→ If its operand (a, b, ...) Print

→ If its operator, push to stack

→ Next operator = ~~low precendence~~ = ~~Print & PUT~~

= high precendence = Push to stack

= same precendence (or equal) = Pop and print

= Push input to stack

→ If operator is) = pop & print until (

* Infix to Prefix

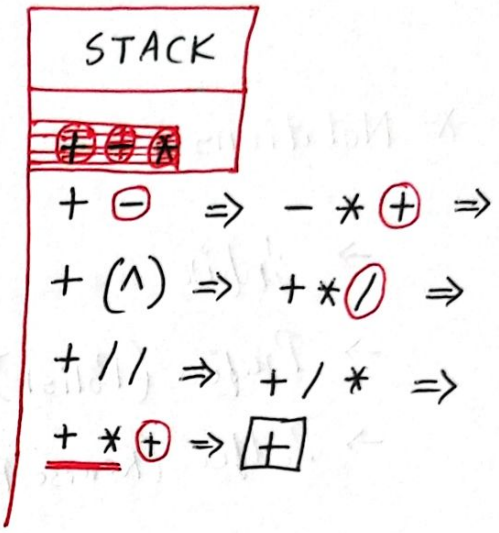
→ Reverse the expression i.e (a+b) ⇒ (b+a)

→ Convert to Postfix

→ Reverse the expression

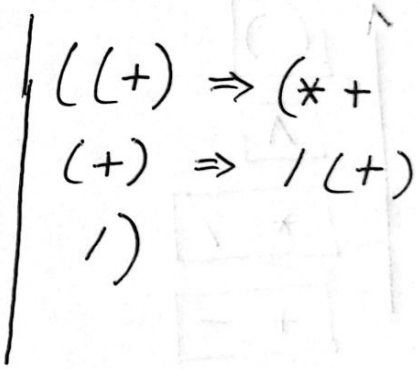
K + L - M * N + (O ^ P) * W / U / V * T + Q

KL+MN*-OP^W*U/V/T*+Q+



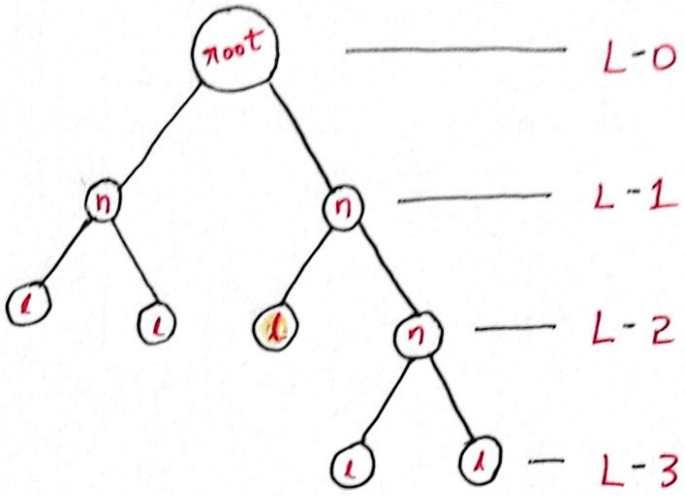
((A+B) * C+ D) / (E + F + G))

AB+C*D+EF+G+/



Binary Tree

- Hierarchial Structure
- Each node can have AT MOST two children



* Node (n) → root, internal node, leaves
* Height (h) → maximum depth → $\max l = 3$
* Level (l) → no of ancestors

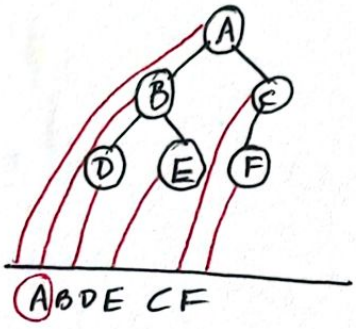
* Max number of nodes at level $l = 2^l$
* Nodes in tree of height $h = 2^{h+1} - 1$
* Min height/level for n nodes = $\log(n+1)$
* If there's X leaves, then levels = $\lfloor \log X \rfloor + 1$
* Edges $e = n - 1$

ORDER Traversals

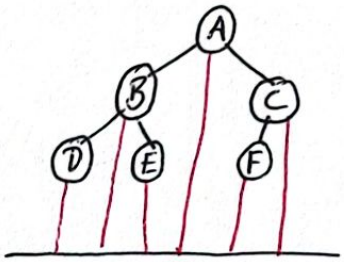
Preorder – Root, L, R – N, L (NLR), R (NLR)

Inorder – L, Root, R – L (LNR), N, R (LNR)

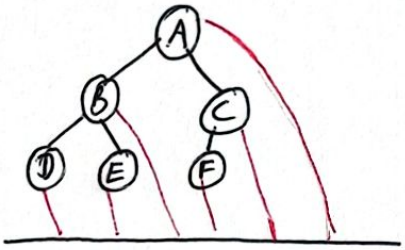
Postorder – L, R, Root – L (LNR), R (LNR), N



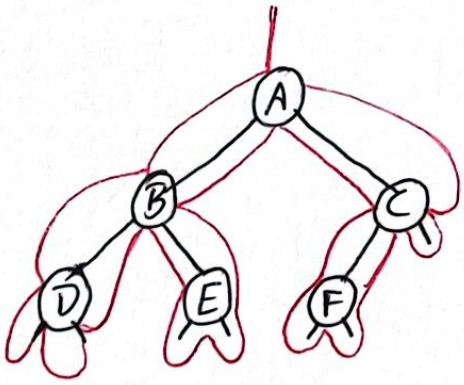
A B D E C F



D B E A F C



D E B F C A



PRE → Print when node is visited once → A B D E C F

IN → Print twice → D B E A F C

POST → Print thrice → D E B F C A

KNAPSACK Problem

Greedy method

Profits	P =	10	5	15	7	6	18	3
Weights	W =	2	3	5	7	1	4	1

$m = 15$

$\frac{P}{W}$	5	1.3	3	1	6	4.5	3
	2	6	4	7	1	3	5
x	1	$2/3$	1	0	1	1	1

(15) (14) (12) (8) (3) (2) [0]

1) Find P/W Profit per Weight

2) Fill the bag from max P/W

From m, subtract weights (W) corresponding to maximum (P/W)

3) Find (x) where $x = \frac{\text{weight added to bag}}{\text{weight}}$

4) PROFIT $= \sum x_i P_i$

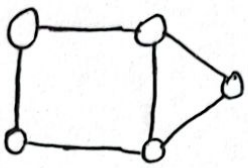
$= 10 + \frac{2}{3} \times 5 + 15 + 6 + 18 + 3$

$= 55.33$

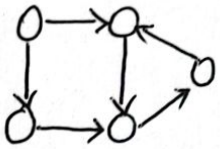
GRAPH

O = Nodes (or) Vertex

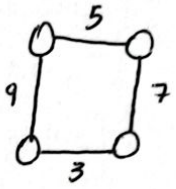
| = Edges



Undirected

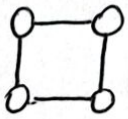


Directed



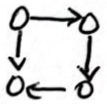
Weighted

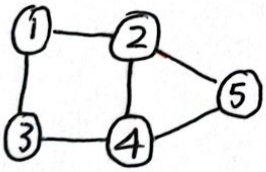
Cyclic



→ start at a node
→ end at that node

acyclic





Path = (1, 2, 5, 4), (1, 2, 4, 2, 5)

Degree = edges connected = D(2) = 3

Total Degree of Graph = 2 × Edges

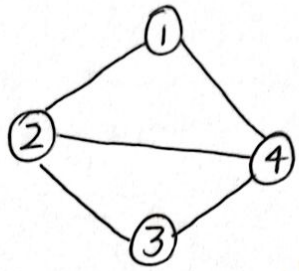
For directed graph, In-degree and out-degree

SPANNING TREE

Given a graph G_T with V vertex and E edge,

minimum spanning tree is the

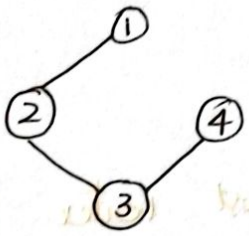
- subgraph of G_T
- has every vertex of G_T
- no cycle

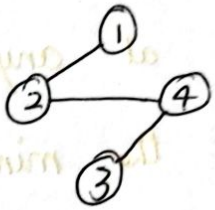


G_T

$V = 4$

$E = 5$





...

~~ST~~ ST

$\overline{V} = V = 4$

$\overline{E} = V - 1 = 3$

Spanning tree has V number of vertices just as it is in G, and $(V-1)$ number of edges

Number of spanning trees possible from graph G_T with (E, V) = ${}^{|E|}C_{|V|-1}$ - no of cycles

In the above graph, ${}^{5}C_{4-1} - 2$

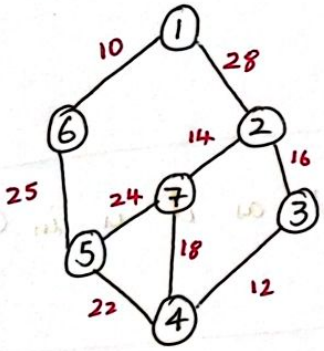
Minimum Cost Spanning Tree

For a weighted graph, ST with minimal cost

How to find? Prim's, Kruskal's

* Prim's [— greedy
[— grow vertex

~ start at any node
~ choose the minimum cost vertex to the node
~ add it to priority queue M
~ now choose the minimum cost vertex to any nodes in M



[1, 6, 5, 4, 3, 2, 7]

1 $\overset{28}{\_\_}$ 2
① ~~1 $\overset{10}{\_\_}$ 6~~
② 6 $\overset{25}{\_\_}$ 5
~~5~~ 5 $\overset{24}{\_\_}$ 7
③ 5 $\overset{22}{\_\_}$ 4
4 $\overset{18}{\_\_}$ 7
④ 4 $\overset{12}{\_\_}$ 3
⑤ 3 $\overset{16}{\_\_}$ 2
⑥ 2 $\overset{14}{\_\_}$ 7

$O(V^2)$

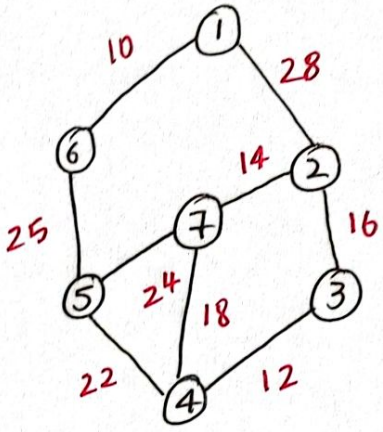
$O((E+V) \log V)$

* Kruskal's E- greedy
E- grow edge

~ start with edge having minimal weight

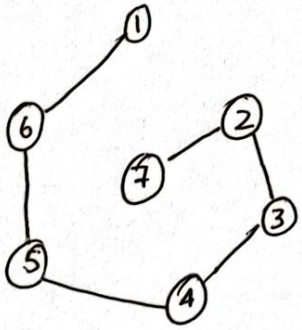
~ keep choosing next minimal edge

~ don't form cycle



10, 12, 14, 16, 18, 22, 24, 25, 28

→ forms cycle
→ b/w 7 & 4
→ skip it



Time → O(V)

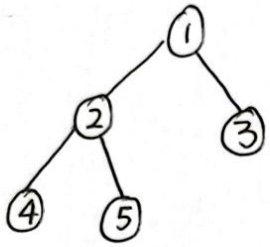
→ O(E log V)

HEAP

min heap

node x

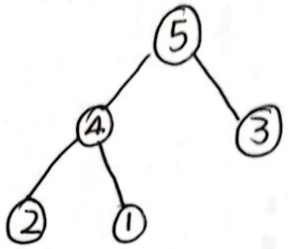
x's children are larger



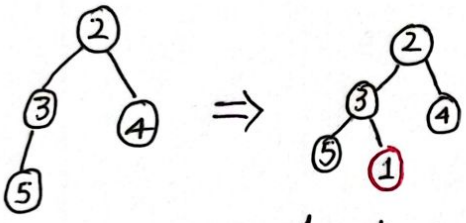
max heap

node x

x's children are smaller

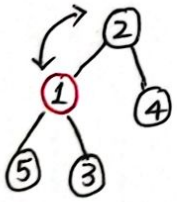


INSERT [1]

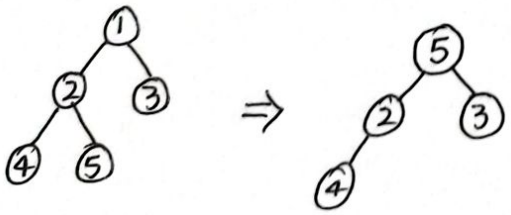


Insert 1 to first available space [LEFT to RIGHT]

Swap Until Heap Property

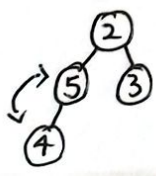


DELETE [1]



Delete [1]
Replace it with last element [5]

Swap Until Property (with minimal)



$\because 2 < 3$