AKTU

B.Tech 5th Sem

GW GATEWAY CLASSES

CS IT & CS Allied

DAA : Design & Analysis Of Algorithm

Unit-3 : Lecture-1

Today's Target

- Greedy Method
- Fractional knapsack problem
- AKTU PYQs

By Dr. Nidhi Parashar Ma'am
- M.Tech Gold Medalist
- Net Qualified

Design and Analysis of Algorithm

UNIT-III : Divide and Conquer, Greedy Methods

AKTU : Syllabus

Divide and Conquer with Examples Such as Sorting, Matrix Multiplication, Convex Hull and Searching.

Greedy Methods with Examples Such as Optimal Reliability Allocation, Knapsack, Minimum Spanning Trees – Prim's and Kruskal's Algorithms, Single Source Shortest Paths - Dijkstra's and Bellman Ford Algorithms.

Greedy Algorithm

- Used for solving optimization problems
- Selects the best option available at the moment without worrying that it will bring the overall optimal result.
- Never reverses the earlier decision even if the choice is wrong. It works in a top-down approach.
- May not produce the best result for all the problems.

Example:

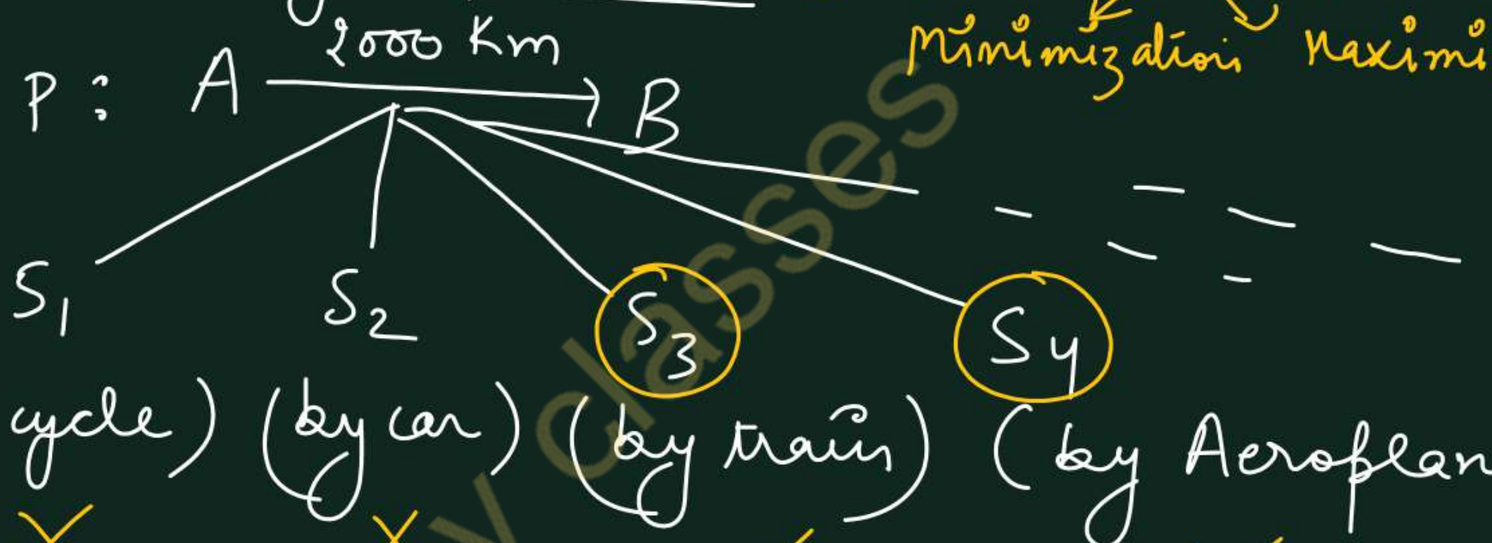
- Fractional Knapsack problem
- Prim's and Kruskal's algos to find minimum spanning tree

Greedy Approach (optimization problem)

optimization problem → Minimization, Maximization

Constraint –

Time : 12 hr.



S_1 ✗ (by cycle)

S_2 ✗ (by car)

S_3 ✓ (by train)

S_4 ✓ (by Aeroplane)

S_3, S_4 → Feasible solutions

S_4 ⇒ optimal solution

for minimum time we choose S_4

Pseudo-Code for Greedy Algorithm

```
Algorithm Greedy (a, n) //a[1:n] contains the n inputs.
{
        solution:=0; //initialize the solution.
        for i= 1 to n do
        {
                X = Select (a);
                if Feasible ( solution, X)  then
                        solution: = Union(solution, X);
        }
        return solution;
}
```

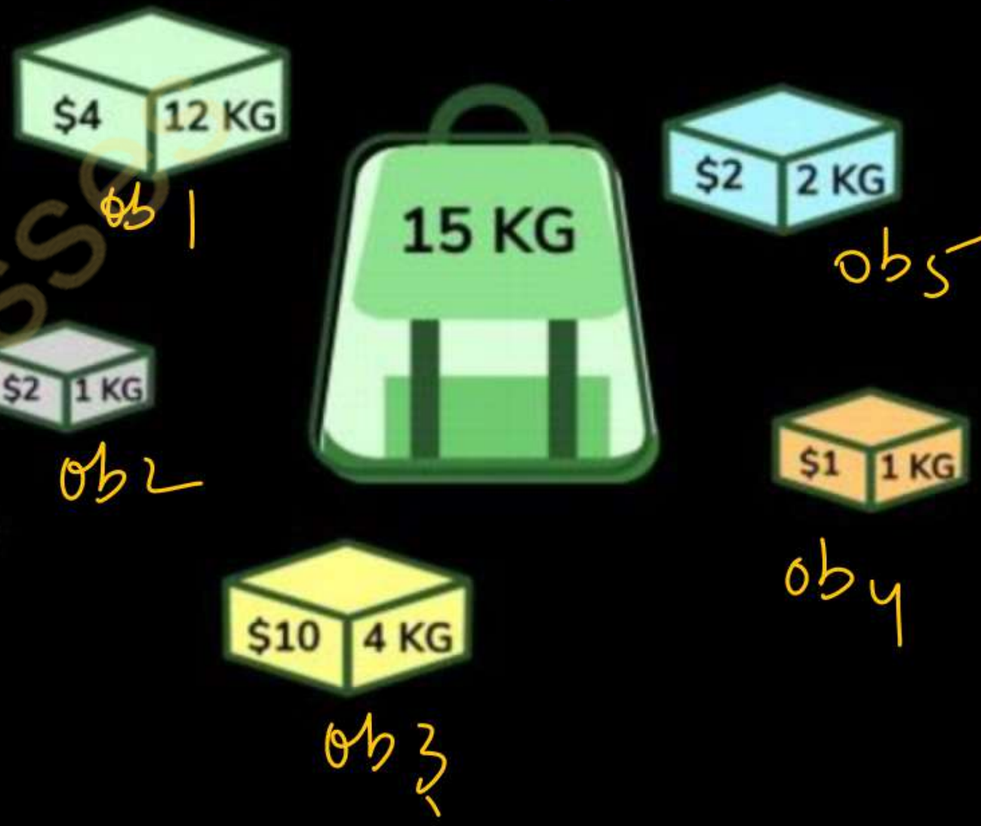
$a = \{ a_1, a_2, a_3 - - - - a_n \}$

- The function select selects an input from 'a', removes it and assigns its value to 'x'.
- Feasible is a Boolean valued function, which determines if 'x' can be included into the solution vector.

Fractional Knapsack Problem

divisible objects

- There are given some objects and a knapsack/bag.
- Object i has a weight w_i and knapsack has a capacity m.
- **If a fraction x_i $0<=x_i<=1$ of object i is placed into the knapsack, then a profit of $p_i x_i$ is earned.**
- **The objective is to fill the knapsack in a way that maximises the total profit earned.**
- Since the knapsack capacity is m, we require the total weight of all chosen objects to be at most m.



Formally, the problem can be stated as follows:

Problem objective

$$\text{maximize} \sum_{1 \le i \le n} p_i x_i$$

p_i → Profit earned by including that object

x_i → fraction of an object

n → no. of objects

Constraint

$$\text{subjected to} \sum_{1 \le i \le n} w_i x_i \le m$$

$$\text{and } 0 \le x_i \le 1, \quad 1 \le i \le n$$

x_i

0 [NOT INCLUDED]

1 [COMPLETELY INCLUDED]

Example:

Objects:	1	2	3	4	5	6	7
Profit (P):	5	10	15	7	8	9	4
Weight(w):	1	3	5	4	1	3	2

W (Weight of the knapsack): 15

n (no of items): 7

$x = \{0, 1, 1, 3/4, 1, 1, 0\}$

$x_1, x_2, x_3, x_4, x_5, x_6, x_7$

First approach: Select the items based on the maximum profit.

Object	Fraction of Object	Profit ($P_i * x_i$)	Weight	Remaining weight (15)
3	$x_3 = 1$	$15 \times 1 = 15$	5	$15 - 5 = 10$
2	$x_2 = 1$	$10 \times 1 = 10$	3	$10 - 3 = 7$
6	$x_6 = 1$	$9 \times 1 = 9$	3	$7 - 3 = 4$
5	$x_5 = 1$	$8 \times 1 = 8$	1	$4 - 1 = 3$
4	$x_4 = 3/4$	$7 \times 3/4 = 5.25$	3	$3 - 3 = 0$
$\sum P_i x_i = 15 + 10 + 9 + 8 + 5.25 = 47.25$				

$\frac{21}{4}$

Example:

Objects:	1	2	3	4	5	6	7
Profit (P):	5	10	15	7	8	9	4
Weight(w):	1	3	5	4	1	3	2

W : 15

Second approach: Select the items based on the minimum weight.

$x = \{ x_1, x_2, x_3, x_4, x_5, x_6, x_7 \} = \{ 1, 1, 1/5, 1, 1, 1, 1 \}$

Object	Fraction of Object	Profit $(P_i * x_i)$	Weight	Remaining weight (15)
1	$x_1 = 1$	$5 \times 1 = 5$	1	$15 - 1 = 14$
5	$x_5 = 1$	$8 \times 1 = 8$	1	$14 - 1 = 13$
7	$x_7 = 1$	$4 \times 1 = 4$	2	$13 - 2 = 11$
2	$x_2 = 1$	$10 \times 1 = 10$	3	$11 - 3 = 8$
6	$x_6 = 1$	$9 \times 1 = 9$	3	$8 - 3 = 5$
4	$x_4 = 1$	$7 \times 1 = 7$	4	$5 - 4 = 1$
3	$x_3 = 1/5$	$15 \times 1/5 = 3$	1	$1 - 1 = 0$
	$\sum P_i x_i = 46$			

Fraction Knapsack problem's optimal solution

Example:

Objects:	1	2	3	4	5	6	7
Profit (P):	5	10	15	7	8	9	4
Weight(w):	1	3	5	4	1	3	2
P/w:	5	3.3	3	1.7	8	3	2

W : 15

$x = \{ \underset{x_1}{1}, \underset{x_2}{1}, \underset{x_3}{1}, \underset{x_4}{0}, \underset{x_5}{1}, \underset{x_6}{1}, \underset{x_7}{1} \}$

Third ~~Second~~ approach: Select the items based on the profit/weight ratio.

Object	Fraction of Object	Profit ($P_i * x_i$)	Weight	Remaining weight (15)
5	$x_5 = 1$	$8 \times 1 = 8$	1	$15 - 1 = 14$
1	$x_1 = 1$	$5 \times 1 = 5$	1	$14 - 1 = 13$
2	$x_2 = 1$	$10 \times 1 = 10$	3	$13 - 3 = 10$
3	$x_3 = 1$	$15 \times 1 = 15$	5	$10 - 5 = 5$
6	$x_6 = 1$	$9 \times 1 = 9$	3	$5 - 3 = 2$
7	$x_7 = 1$	$4 \times 1 = 4$	2	$2 - 2 = 0$
	$\sum P_i x_i = 51$			

Fractional Knapsack - Greedy Solution

- Greedy-fractional-knapsack (w, v, W)

```
1  for i=1 to n          (for all objects)
2    do x[i]=0           // solution vector (initially empty)
3    weight=0            // initially empty Knapsack
4    while weight < W    → capacity of Knapsack
5    do i = best remaining item
6    if weight + w[i] ≤ W
7      then x[i]=1       // Include whole object
8      weight = weight + w[i]
9    else
10   x[i] = (W - weight)/w[i]   // Include fraction of the object
11   weight = W
12   return x
```

Ques (AKTU)

Items	: 1	2	3	4
Profits	: 45	30	45	10
w	: 3	5	9	5
P/w	: 15	6	5	2

$m = 16$

$x = \{ \underset{x_1}{1}, \underset{x_2}{1}, \underset{x_3}{8/9}, \underset{x_4}{0} \}$

object	Fraction	$(P_i * x_i)$	weight	Remaining weight (16)
1	$x_1 = 1$	$45 \times 1 = 45$	3	$16 - 3 = 13$
2	$x_2 = 1$	$30 \times 1 = 30$	5	$13 - 5 = 8$
3	$x_3 = 8/9$	$\cancel{45}^{5} \times \frac{8}{\cancel{9}_{1}} = 40$	8	$8 - 8 = 0$

$$\sum p_i x_i = 45 + 30 + 40 = 115$$

AKTU PYQs

1. Explain “greedy algorithm” Write its pseudo code to prove that fractional Knapsack problem has a greedy-choice property. (AKTU 2022-23)
2. Find optimal solution to the fractional knapsack instances n=7.M=15, (P1,p2…..p7)=(10,5,15,7,6,18,3) and (w1,w2…..w7)=(2,3,5,7,1,4,1). (AKTU 2015-16)
3. What is Knapsack problem? Solve Fractional knapsack problem using greedy programming for the following four items with their weights w (3, 5, 9, 5) and values P (45, 30, 45, 10) with knapsack capacity is 16. (AKTU 2021-22)
4. Consider following instance for simple knapsack problem. Find the solution using greedy method.

N:8
P : {11,21, 31, 33, 43, 53,55, 65}
W : {1, 11, 21,23,33, 43, 45, 55}
M: 110

(AKTU 2019-20)

AKTU

B.Tech 5th Sem

GW GATEWAY CLASSES

CS IT & CS Allied

DAA : Design & Analysis Of Algorithm

Unit-3 : Lecture-2

Today's Target

- Minimum Spanning Tree
- Prim's Algorithm
- AKTU PYQs

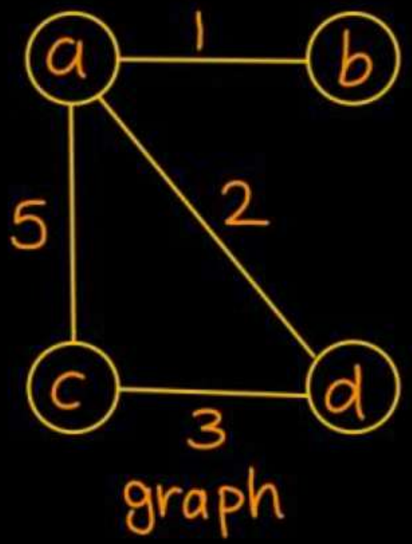
By Dr. Nidhi Parashar Ma'am
- M.Tech Gold Medalist
- Net Qualified

Minimum Cost Spanning Trees

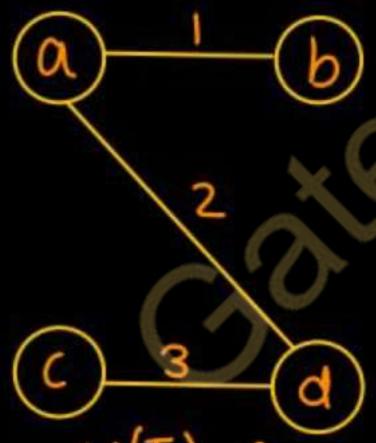
A **spanning tree** of a connected graph is its connected acyclic subgraph (i.e., a tree) that contains all the vertices of the graph.

A **minimum spanning tree** of a weighted connected graph is its spanning tree of the smallest weight, where the weight of a tree is defined as the sum of the **weights** on all its edges.

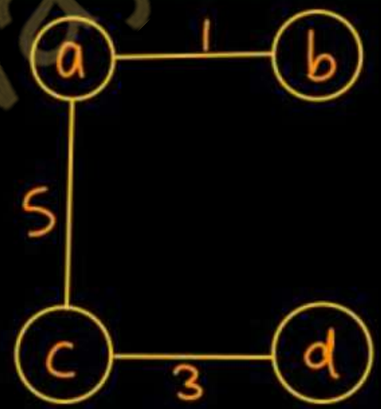
The **minimum spanning tree problem** is the problem of finding a minimum spanning tree for a given weighted connected graph.



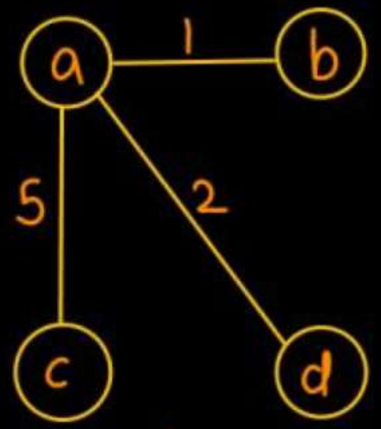
graph



$w(T_1)=6$



$w(T_2)=9$

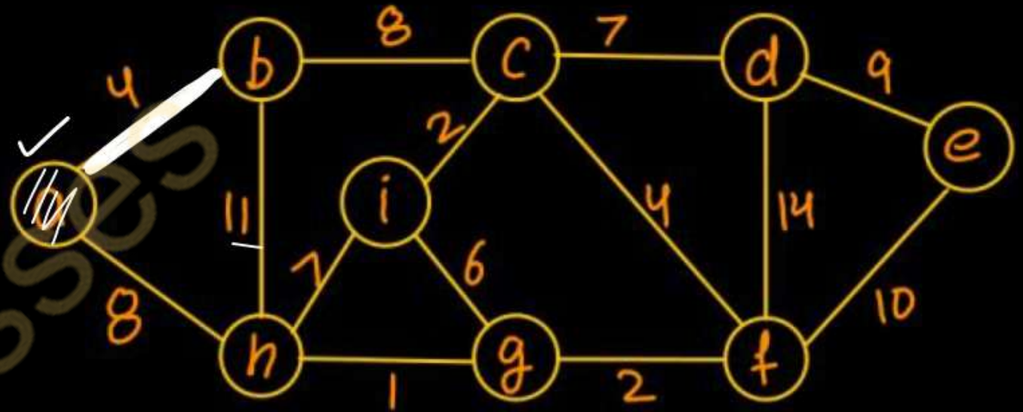


$w(T_3)=8$

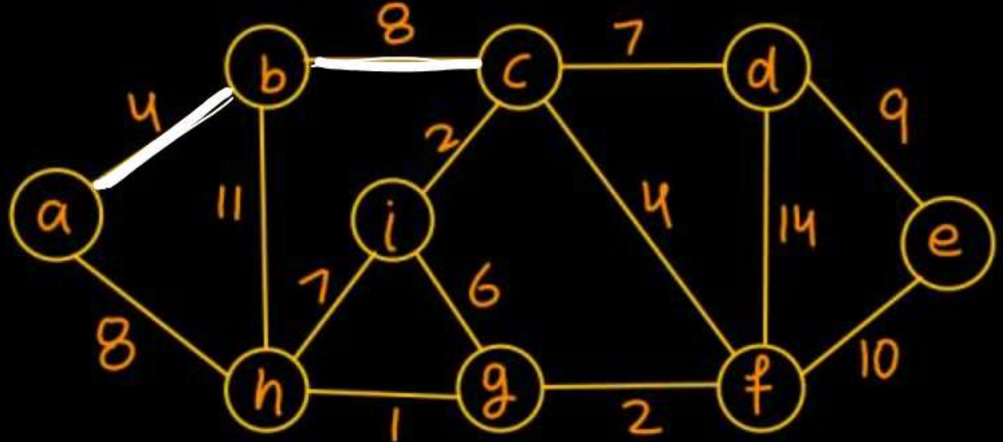
Prim's Algorithm

- Prim's algorithm constructs a minimum spanning tree through a sequence of expanding sub-trees.
- The initial subtree consists of a single vertex selected arbitrarily from the set V of the graph's vertices.
- On each iteration it expands the current tree in the greedy manner by simply attaching to it the nearest vertex not in that tree.
- The algorithm stops after all the graph's vertices have been included in the tree being constructed.
- Since the algorithm expands a tree by exactly one vertex on each of its iterations, the total number of such iterations is n - 1, where n is the number of vertices in the graph.

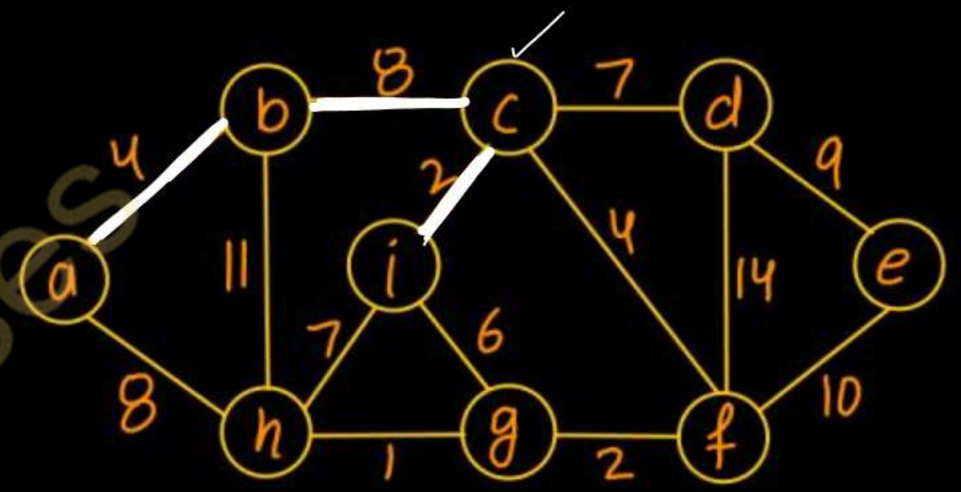
Tree Vertices	Remaining Vertices		Illustration
a(–,0)	b(a,4)	f(–,∞)	
	h(a,8)	g(–,∞)	
	c(–,∞)	i(–,∞)	
	d(–,∞)		
	e(–,∞)		

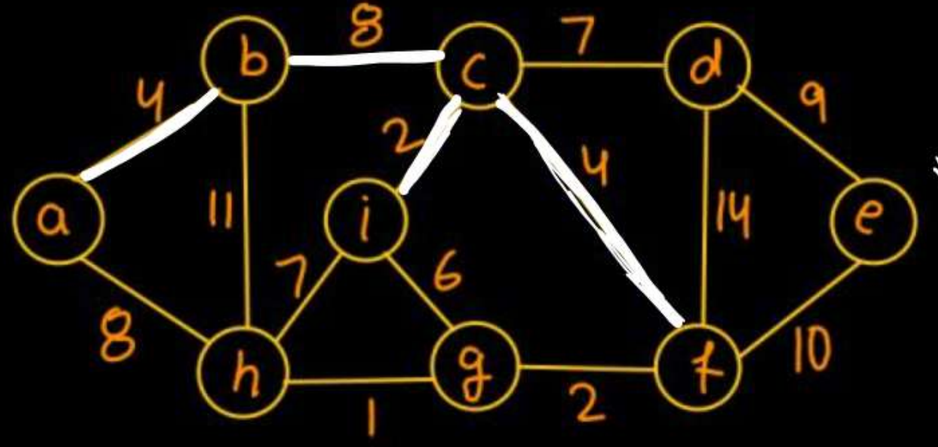


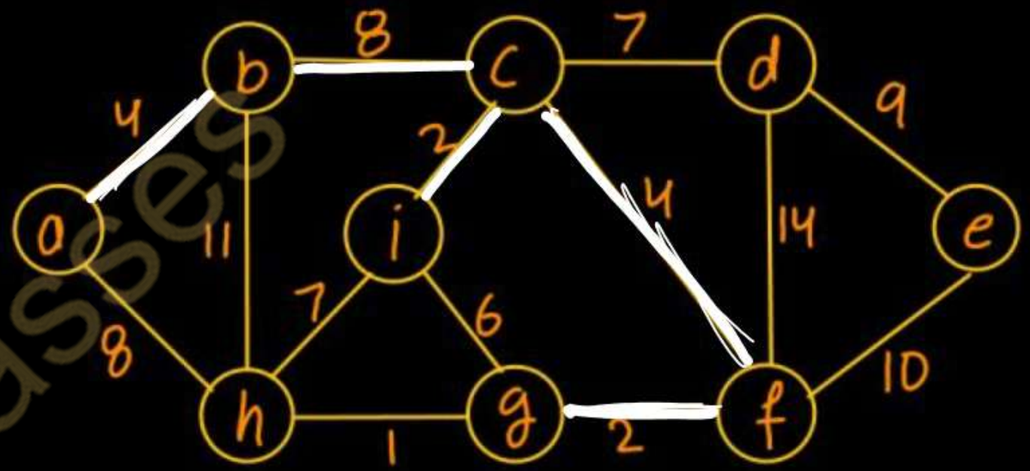
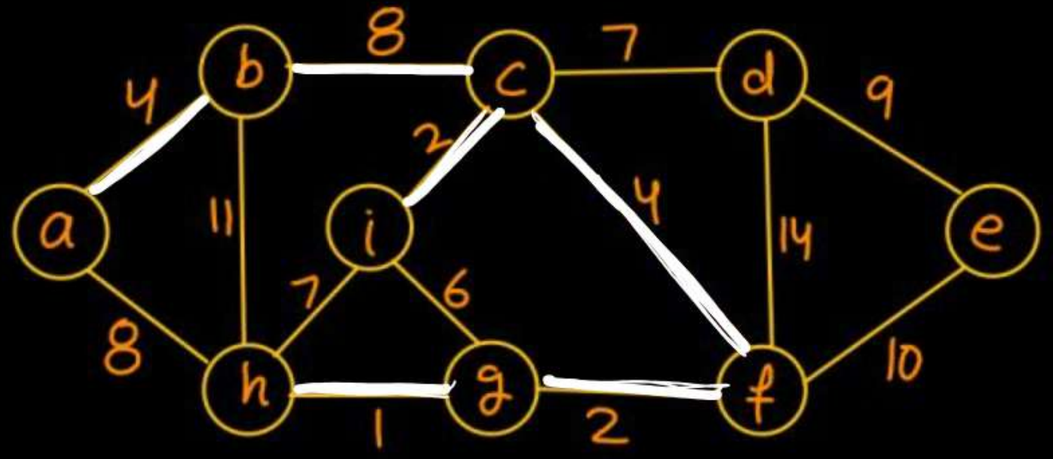
b(a,4)	h(a,8)	f(–,∞)	
	c(b,8)	g(–,∞)	
	d(–,∞)	i(–,∞)	
	e(–,∞)		

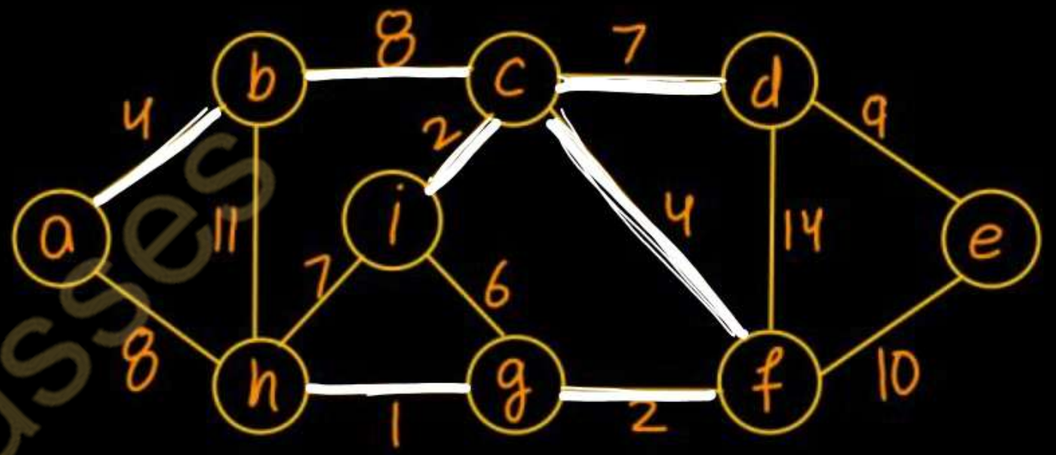
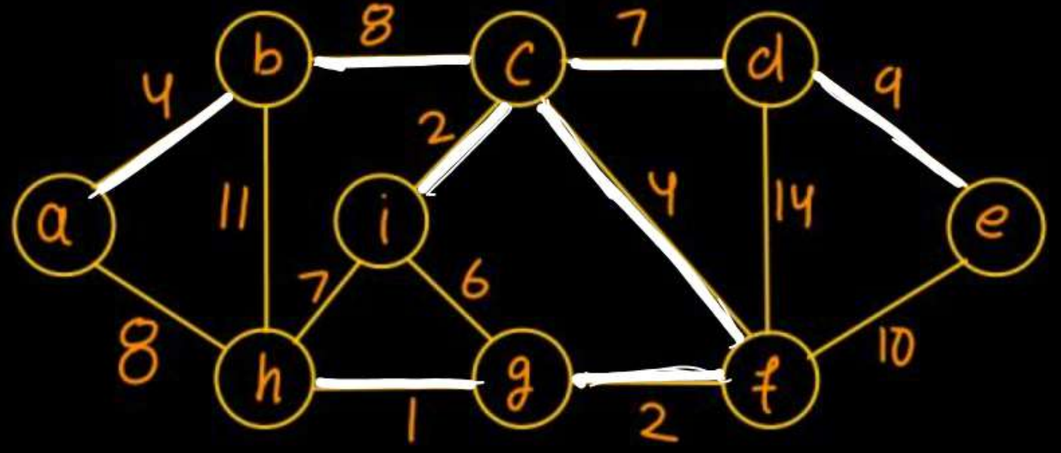


Tree Vertices	Remaining Vertices	Illustration
c(b,8)	i(c,2) d(c,7) f(c,4)	e(–,∞) g(–,∞) h(a,8)
i(c,2)	d(c,7) f(c,4) h(i,7)	e(–,∞) g(i,6)





Tree Vertices	Remaining Vertices	Illustration
f(c, 4)	g(f, 2) d(c, 7) e(f, 10) h(i, 7)	
g(f, 2)	h(g, 1) d(c, 7) e(f, 10)	

Tree Vertices	Remaining Vertices	Illustration
h(g,1)	d(c,7) e(f,10)	
d(c,7)	e(d,9)	
e(d,9)		

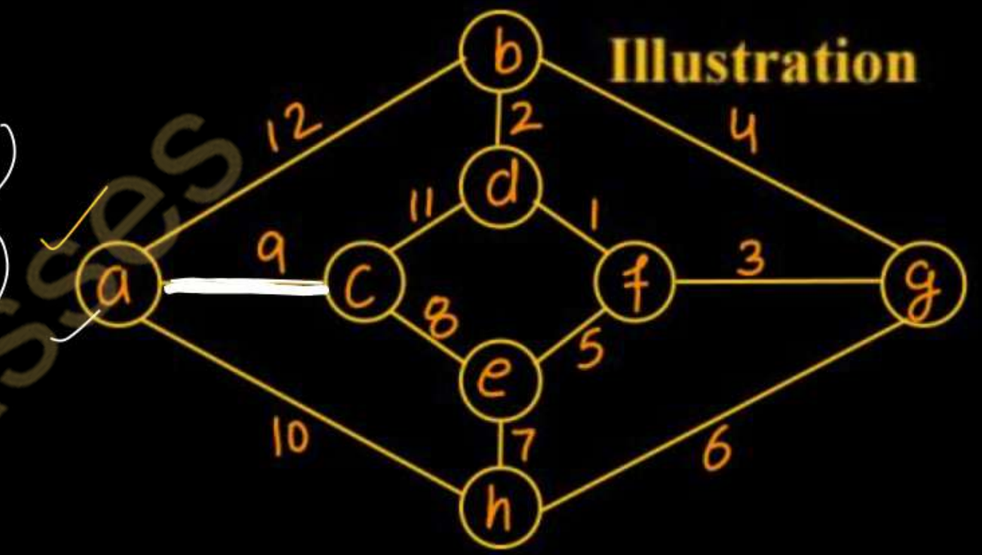
```
MST-PRIM(G,w,r)
1  for each u ∈ V[G]
2      do key[u] ← ∞
3         π[u] ← NIL          // π(u) : parent of u.
4  key[r] ← 0                 // Key of starting index is set to 0.
5  Q ← V[G]                   // All vertices are stored in priority queue (Q)
6  While Q ≠ 0                // Until all vertices are extracted
7      do u ← EXTRACT-MIN(Q)
8         for each v ∈ Adj[u]
9             do if v ∈ Q and w(u,v) < key[v]
10                then π[v] ← u
11                     key[v] ← w(u,v)
```

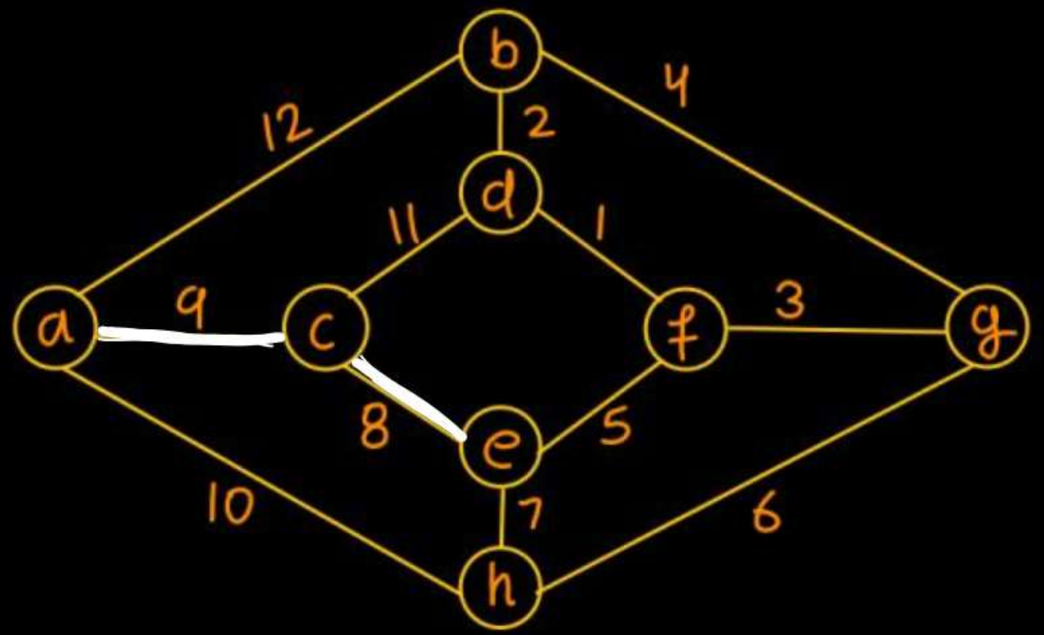
V : set of vertices
r : starting index

Correctness: Prim's algorithm always yields a minimum spanning tree.
Time Complexity: O(|E| log |V|)

Q.1 Find MST using Prim's algorithm with 'a' as starting vertex? (AKTU)

Tree Vertices	Remaining Vertices	Illustration
a(−,0)	b(a,12) | c(a,9) | d(−,∞) | e(−,∞) | f(−,∞) | g(−,∞) | h(a,10)	
c(a,9)	d(c,11) | e(c,8) | b(a,12) | f(−,∞) | g(−,∞) | h(a,10)	

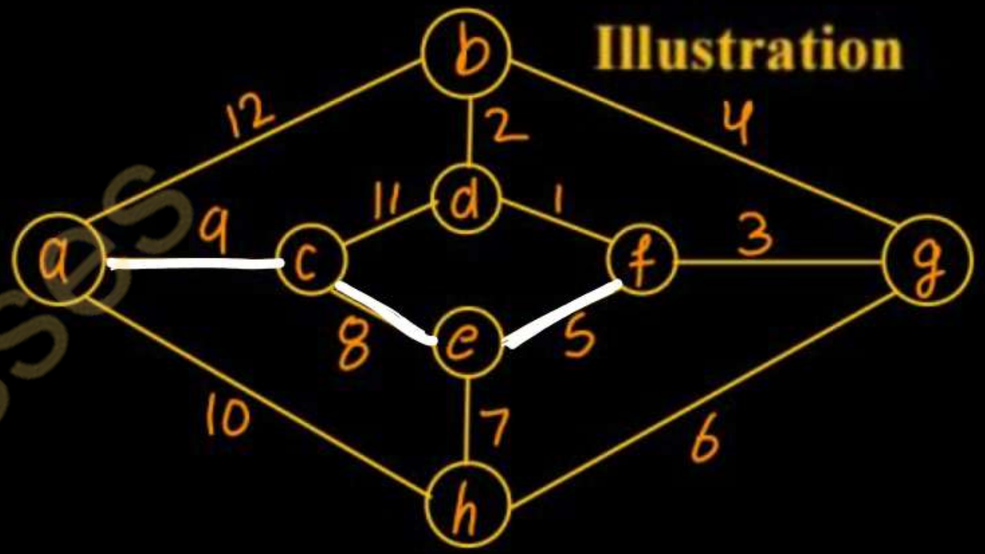


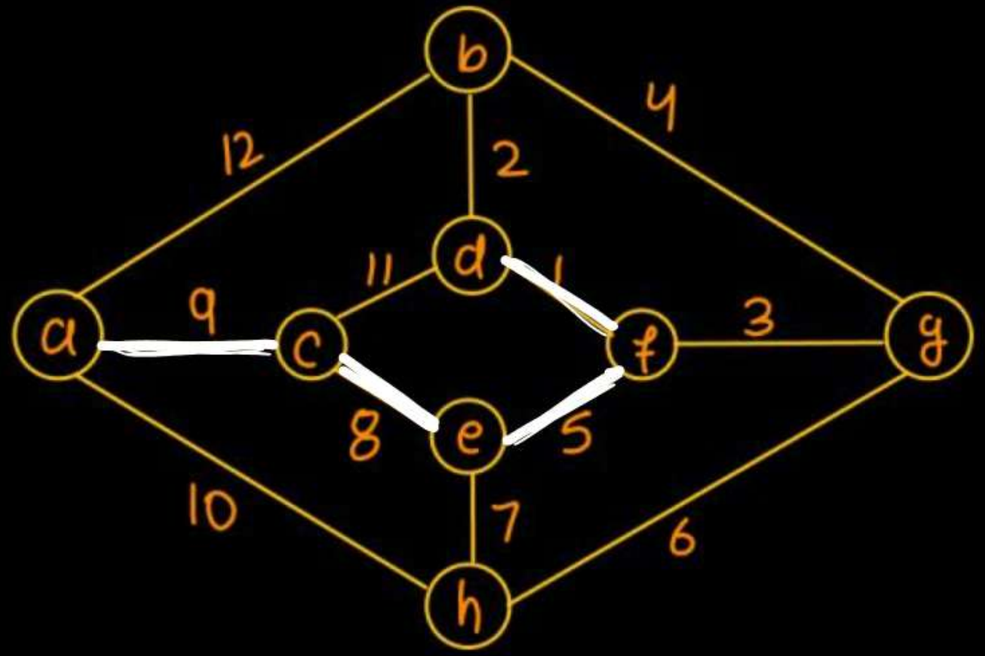


Q.2 Find MST using Prim's algorithm with 'a' as starting vertex? (AKTU)

Tree Vertices	Remaining Vertices	
e(c, 8)	f(e, 5)	b(a, 12)
	h(e, 7)	g(−, ∞)
	d(c, 11)	
f(e, 5)	d(f, 1)	
	g(f, 3)	
	b(a, 12)	
	h(e, 7)	

Illustration

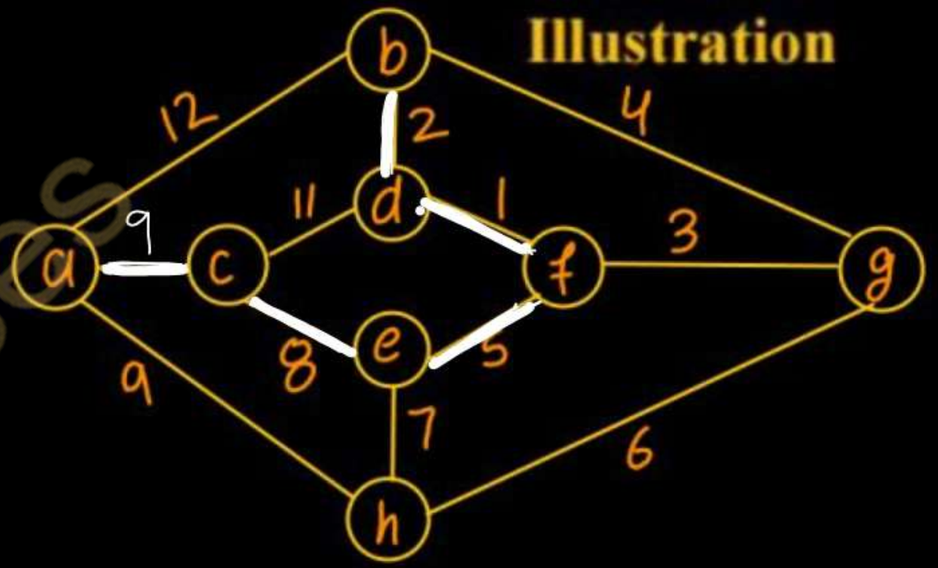


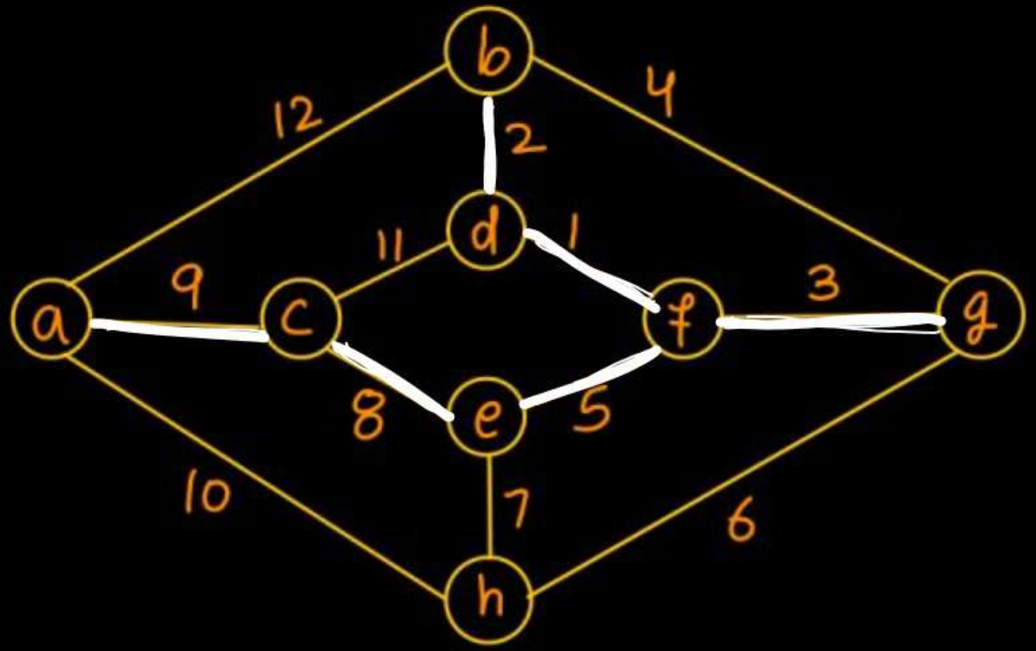


Q.3 Find MST using Prim's algorithm with 'a' as starting vertex? (AKTU)

Tree Vertices	Remaining Vertices
d(f,1)	g(f,3) b(d,2) h(e,7)
b(d,2)	g(f,3) h(e,7)

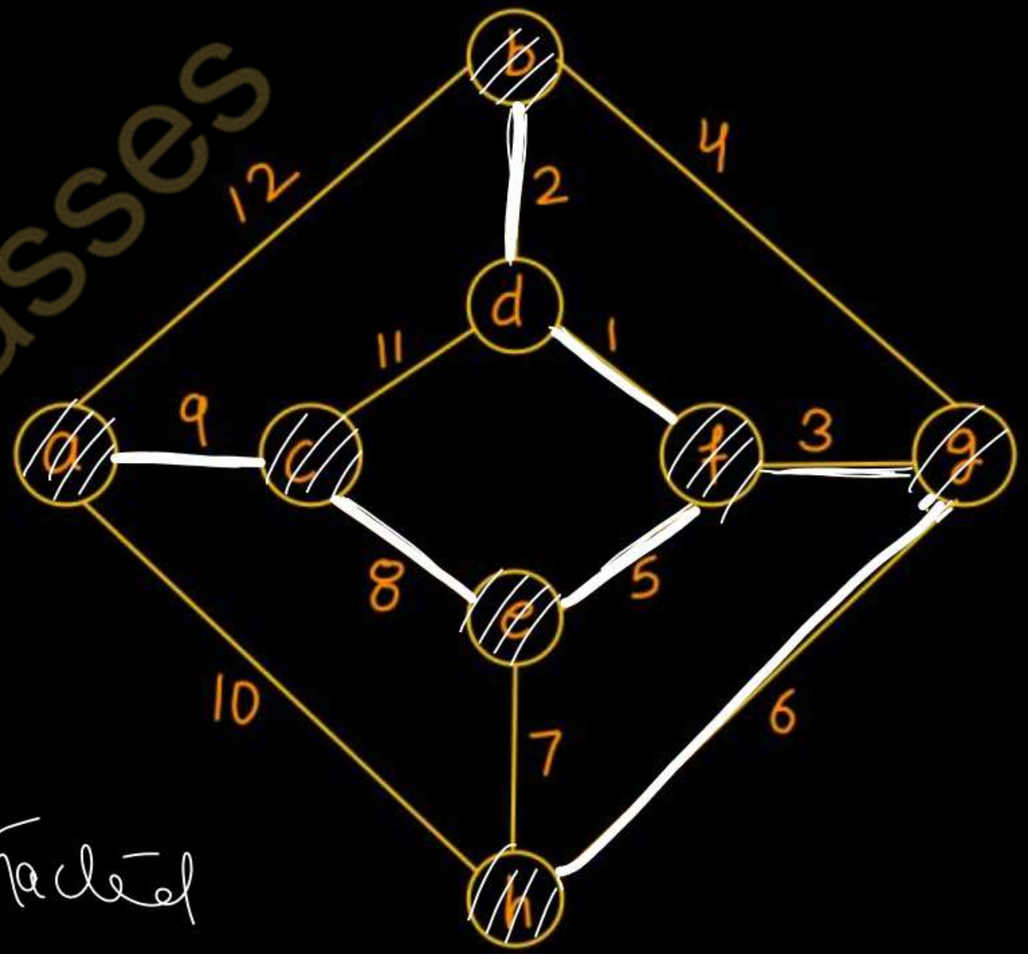
Illustration





Q.4 Find MST using Prim's algorithm with 'a' as starting vertex? (AKTU) (2018-19)

Tree Vertices	Remaining Vertices	Illustration
g(f,3)	h(g,6)	
h(g,6)	'Q' is empty All vertices are extracted	



AKTU

B.Tech 5th Sem

GW GATEWAY CLASSES

CS IT & CS Allied

DAA : Design & Analysis Of Algorithm

Unit-3 : Lecture-3

Today's Target

- Kruskal's Algorithm
- AKTU PYQs

By Dr. Nidhi Parashar Ma'am
- M.Tech Gold Medalist
- Net Qualified

Kruskal's Algorithm

- It is a greedy algorithm for the MST problem that always yields an optimal solution.
- To apply Kruskal's algorithm, the given graph must be weighted, connected and undirected.
- It begins by **sorting** the graph's edges in **non-decreasing** order of their **weights**.
- Starting with the empty subgraph, it adds the next edge to the current sub graph if it **does not create a cycle** or **skip the edge otherwise.**
- On each iteration, the algorithm takes the next edge (u, v) from the sorted list of the graph's edges, finds the trees containing the vertices u and v, and, if these trees are not the same, unites them in a larger tree by adding the edge (u, v).

Pseudocode

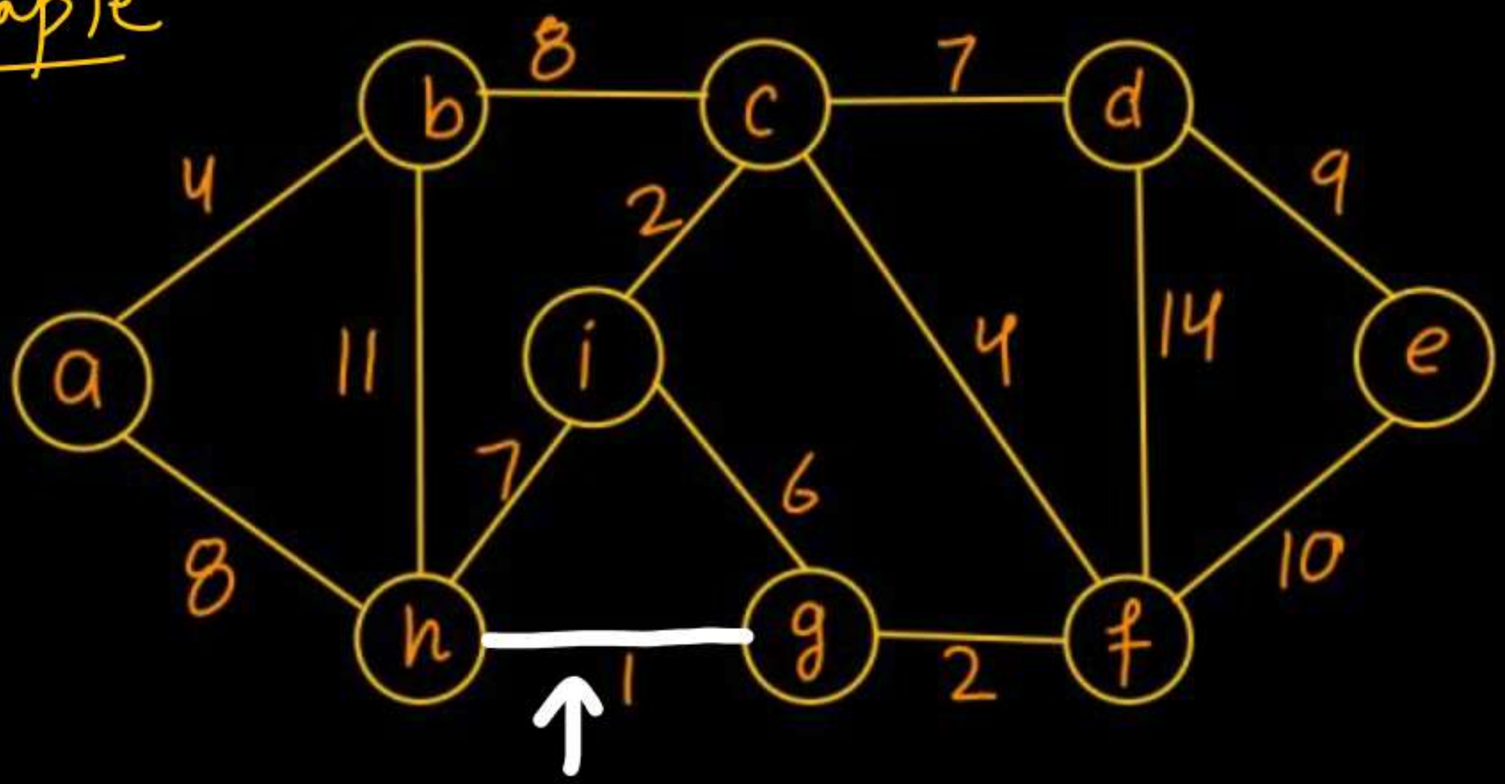
```
MST-KRUSHAL(G,w)
1  A = ∅      // A will carray final set of edges in MST.
2  for each vertex v∈G.V
3       MAKE-SET(v)
4  sort the edges of G.E into nondecreasing order by weight w
5  for each edge(u,v) ∈ G.E, taken in nondecreasing order by weight
6       if FIND-SET(u) ≠ FIND-SET(v)   // if u and v belongs to disjoint sets
7            A ≡ A∪{(u,v)}                and connecting them will not
8            UNION(u,v)                   create a cycle.
9  return A
```

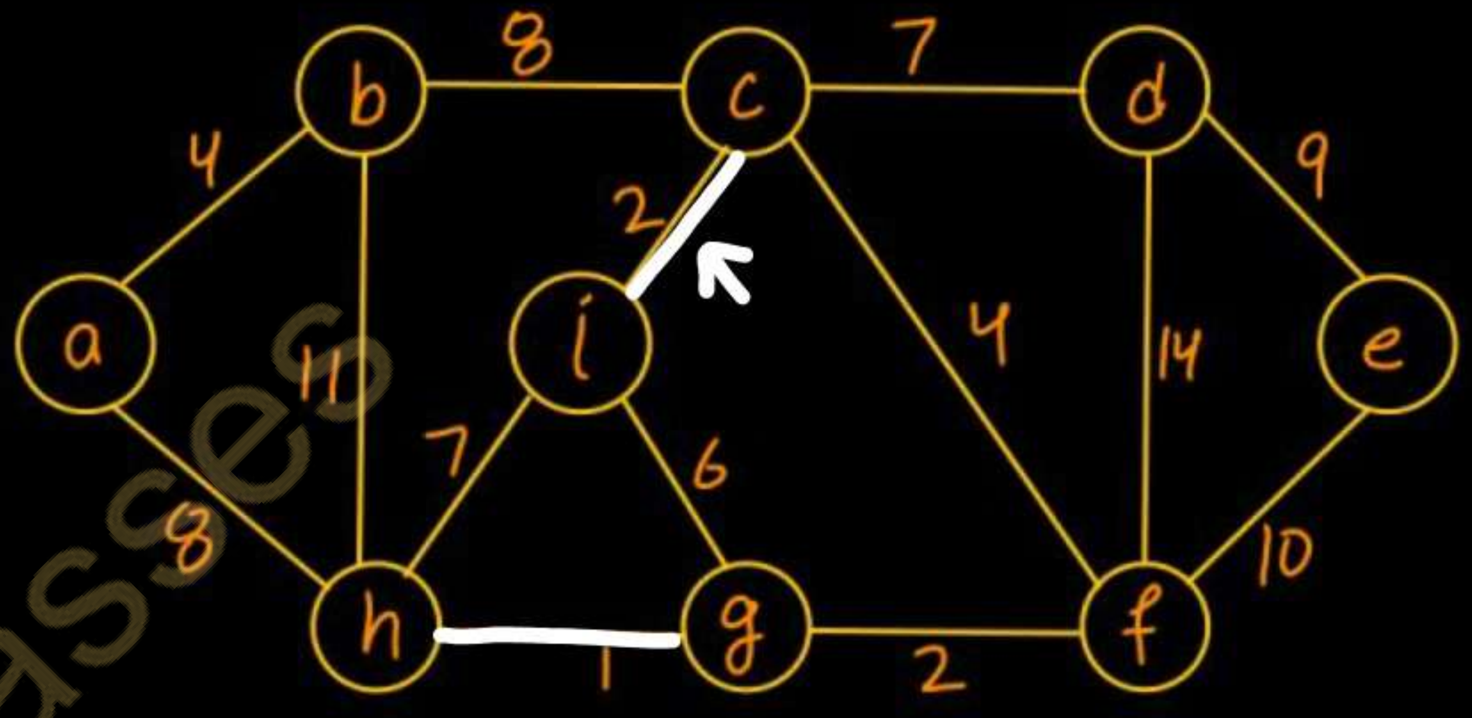
→ Include edge (u,v) into A

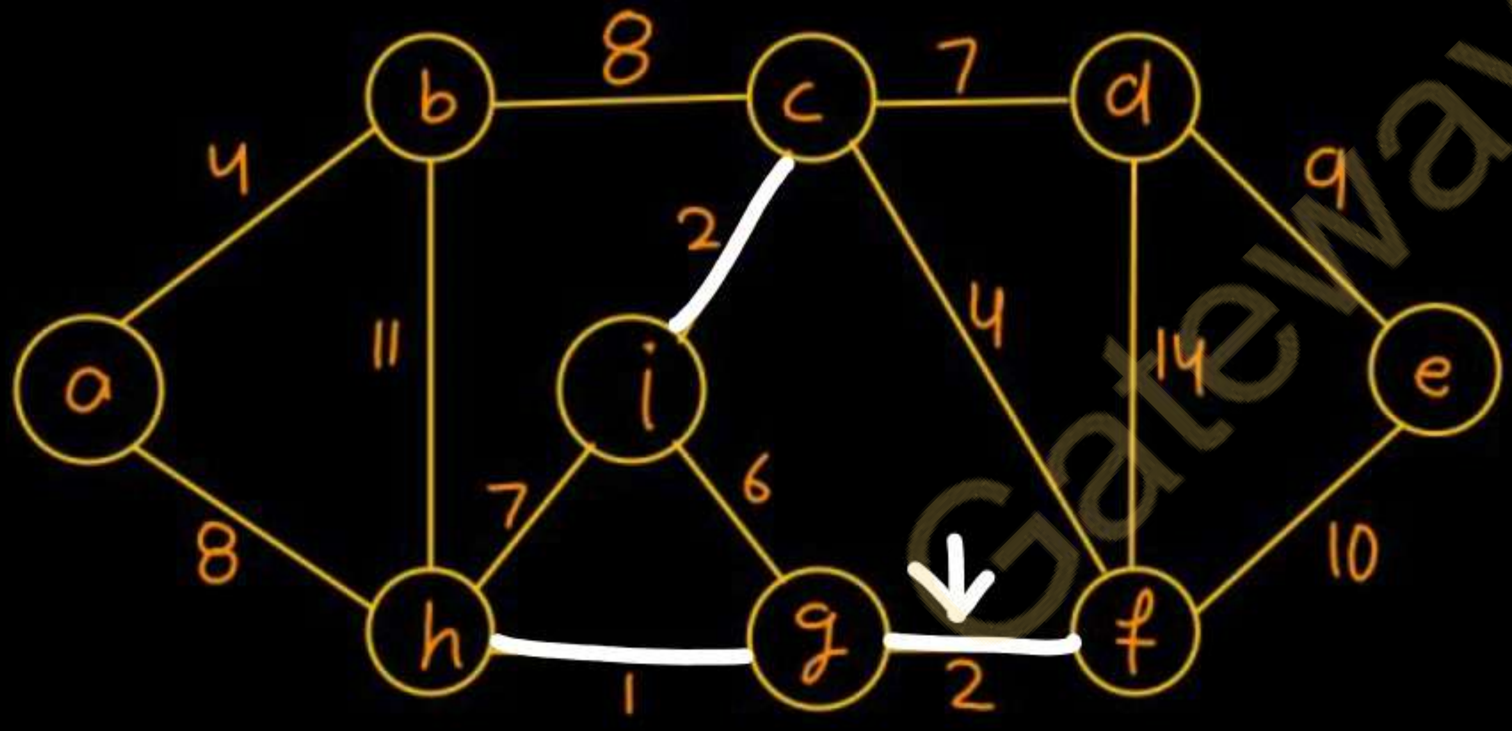
→ Connect vertex 'u' with vertex 'v'

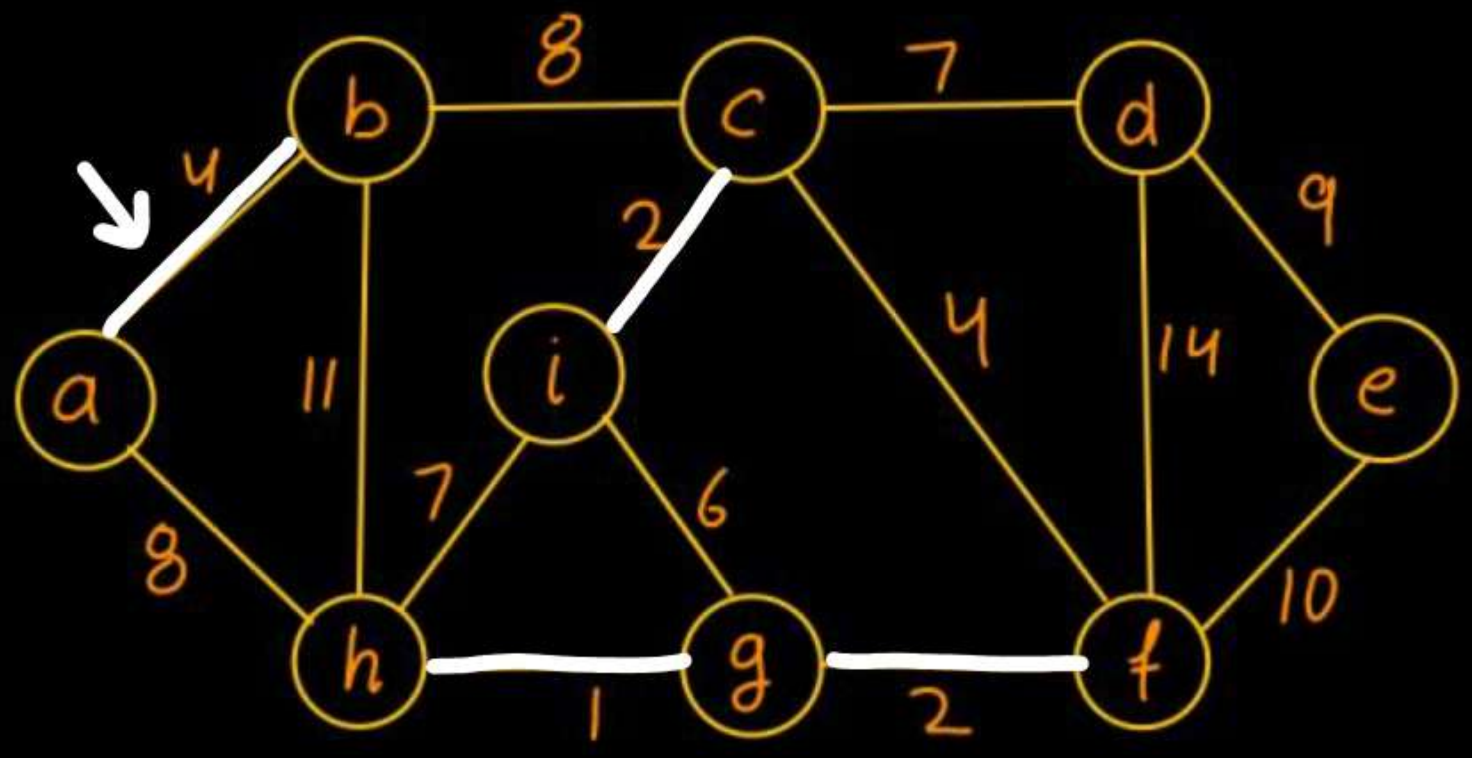
Time Complexity: O (|E| log |E|) or O (|E| log |V|)

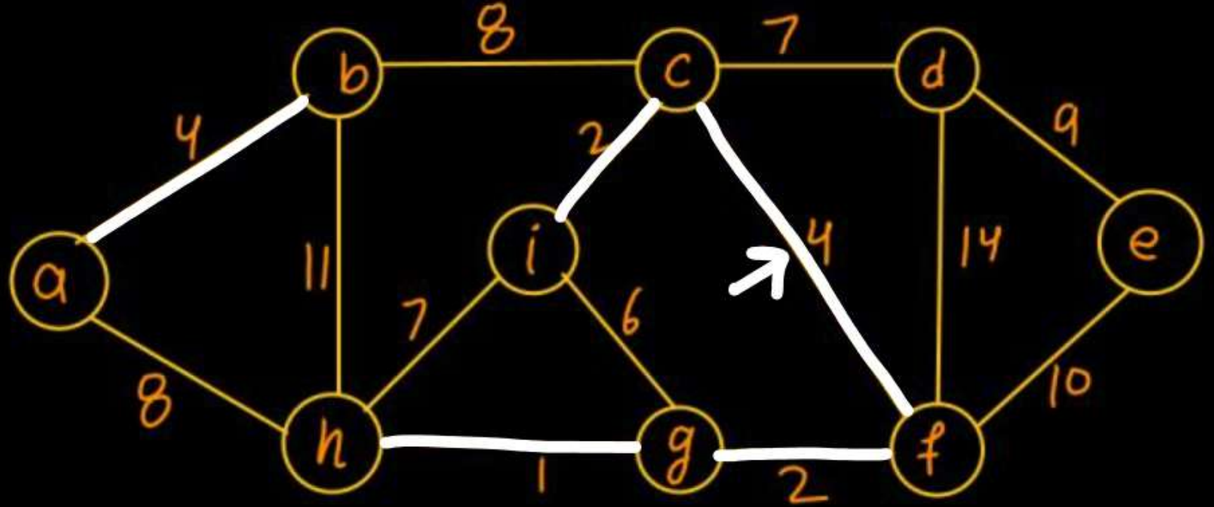
Example

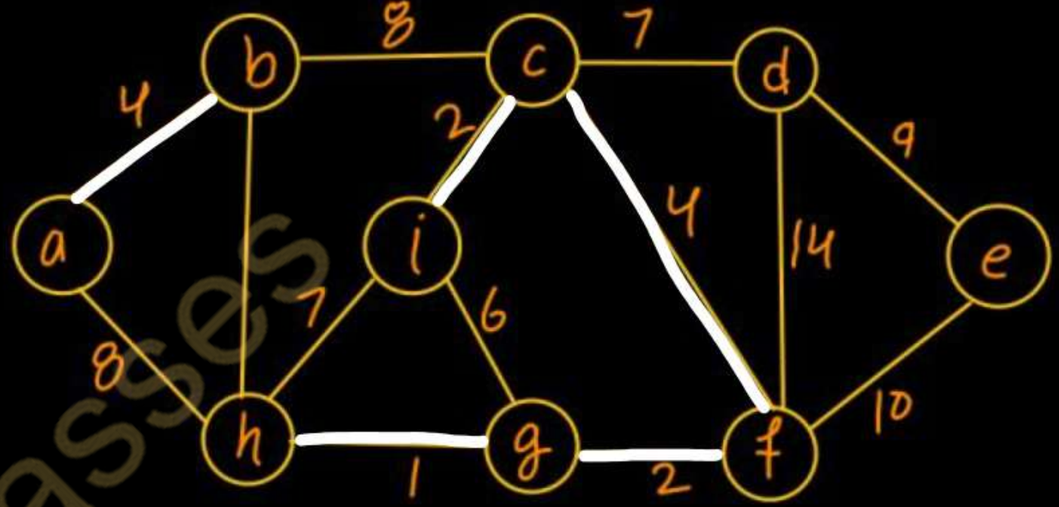


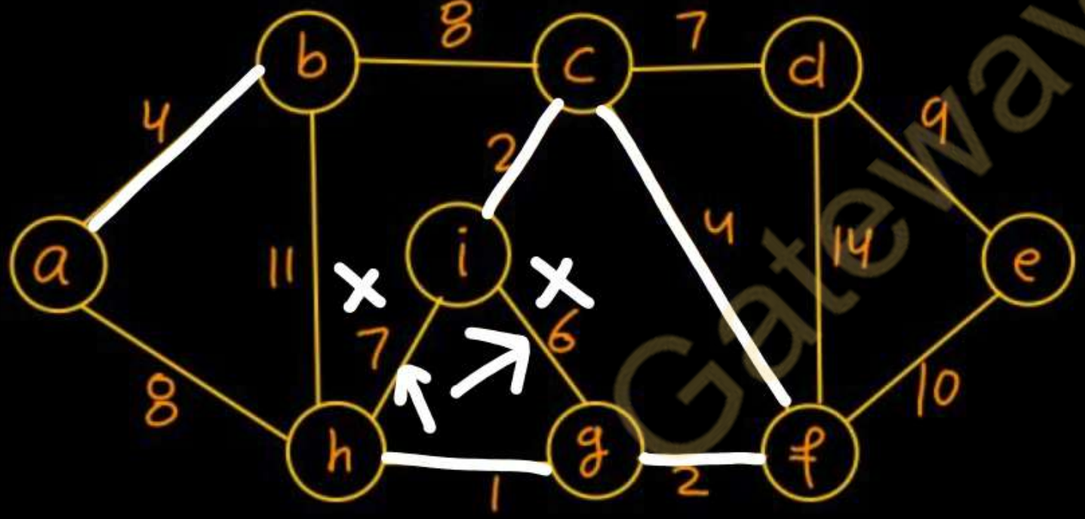


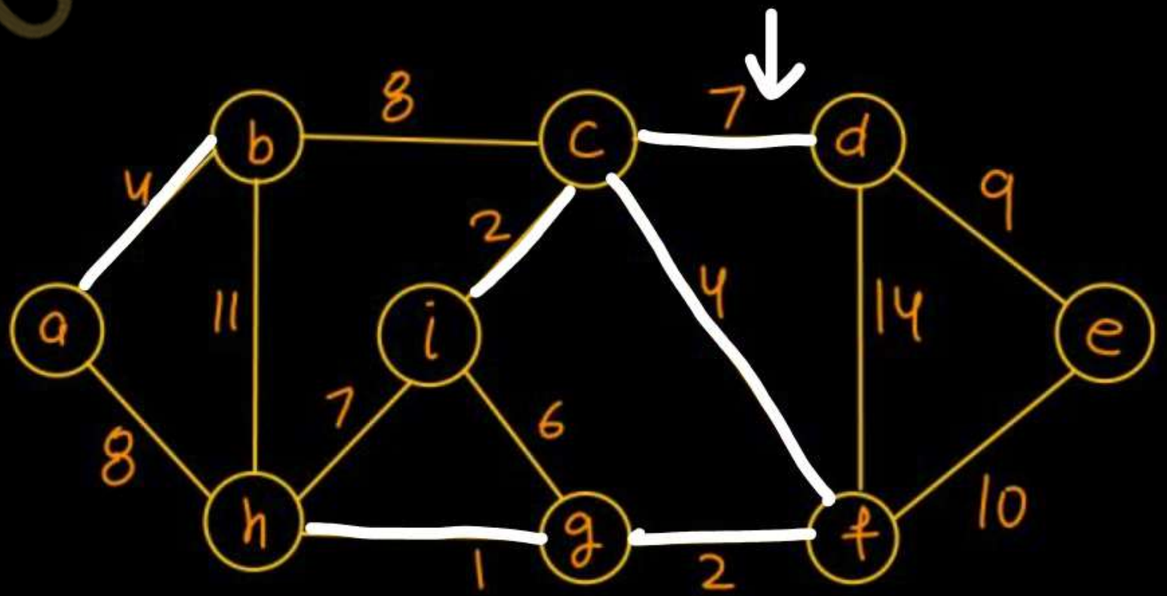


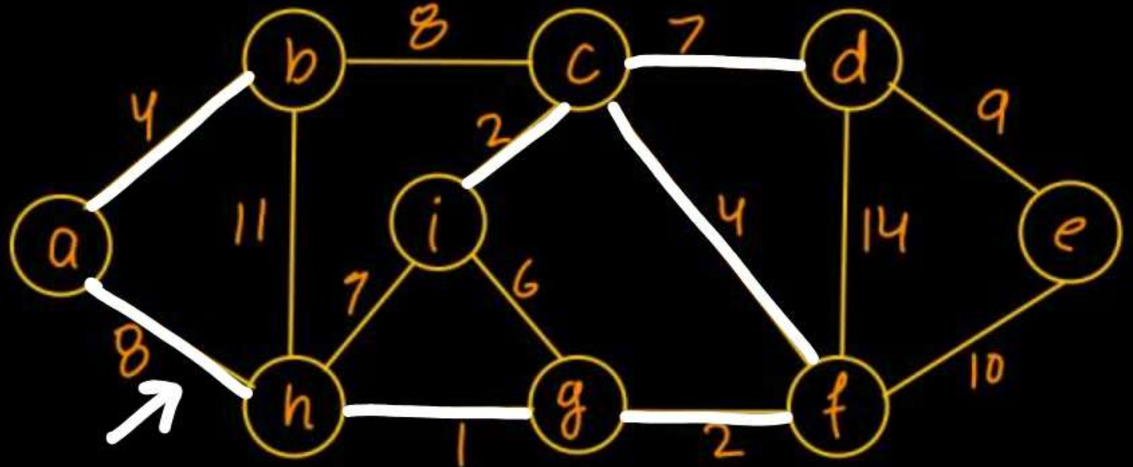


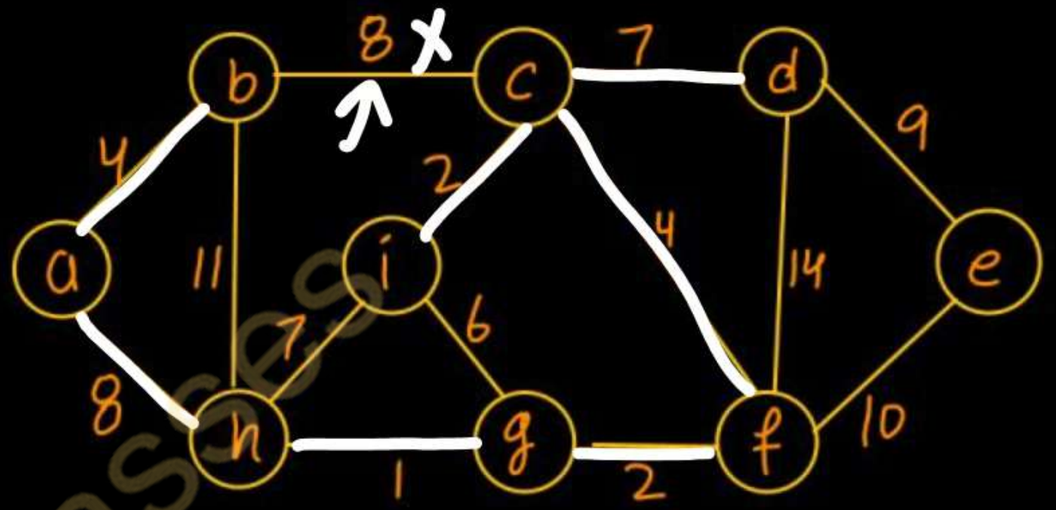


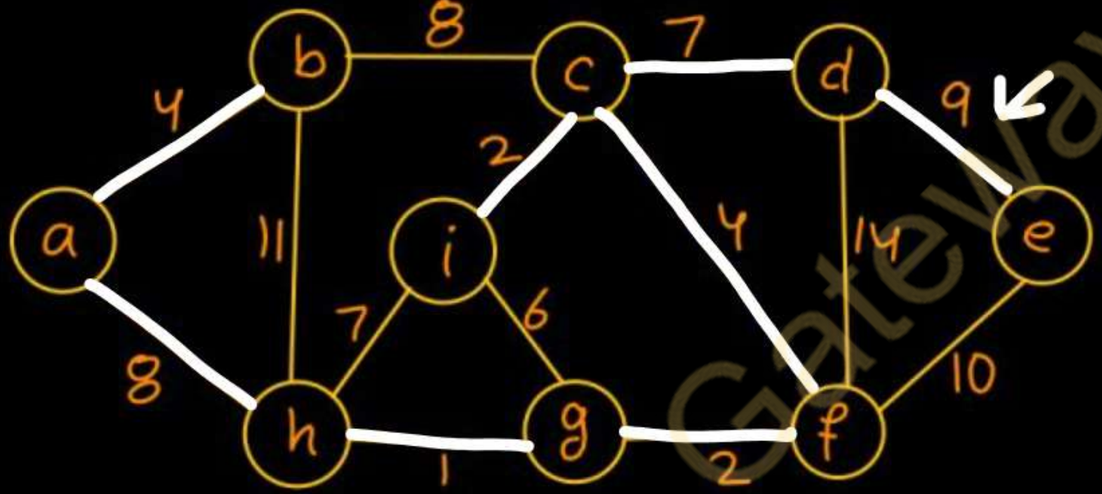


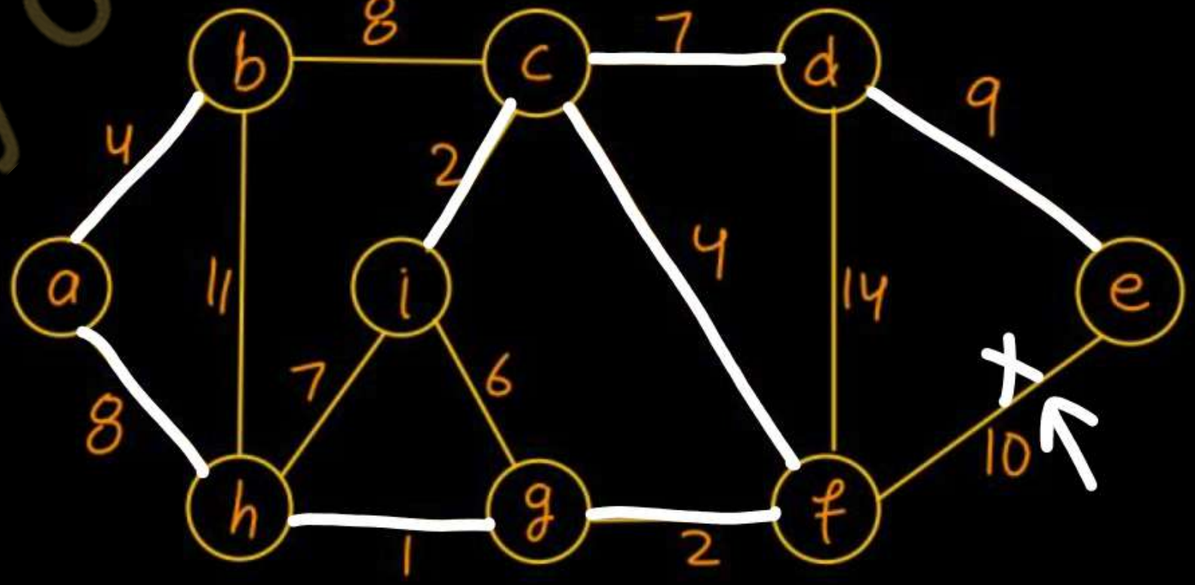


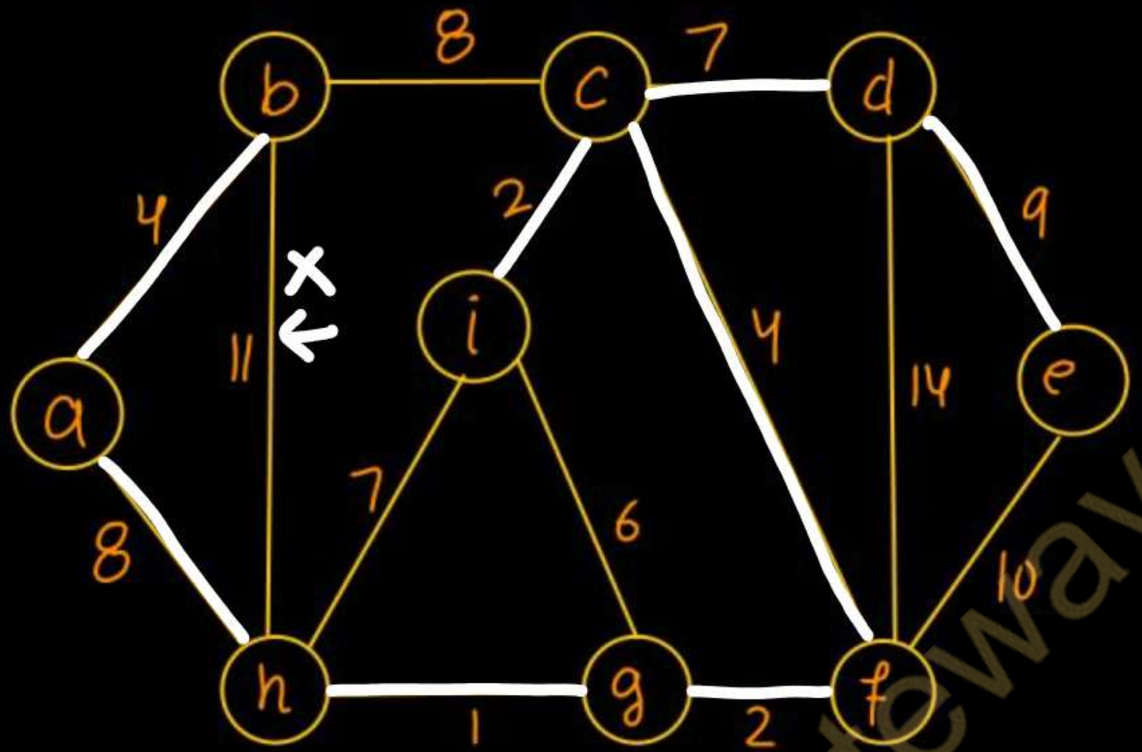


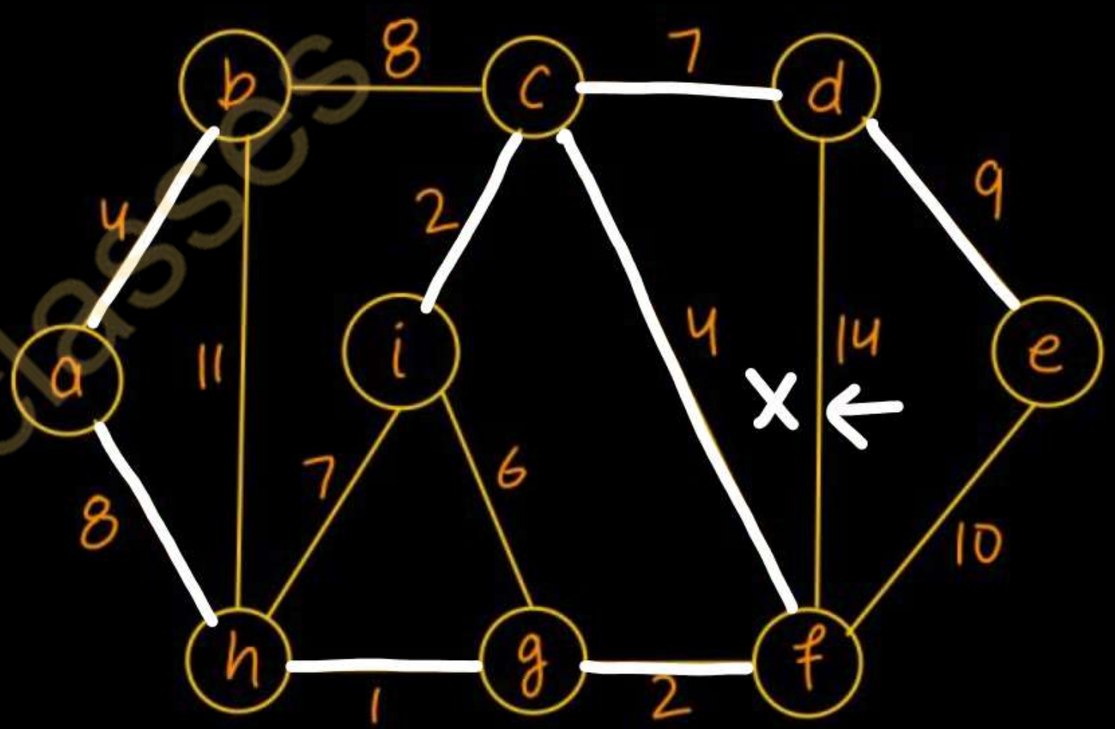






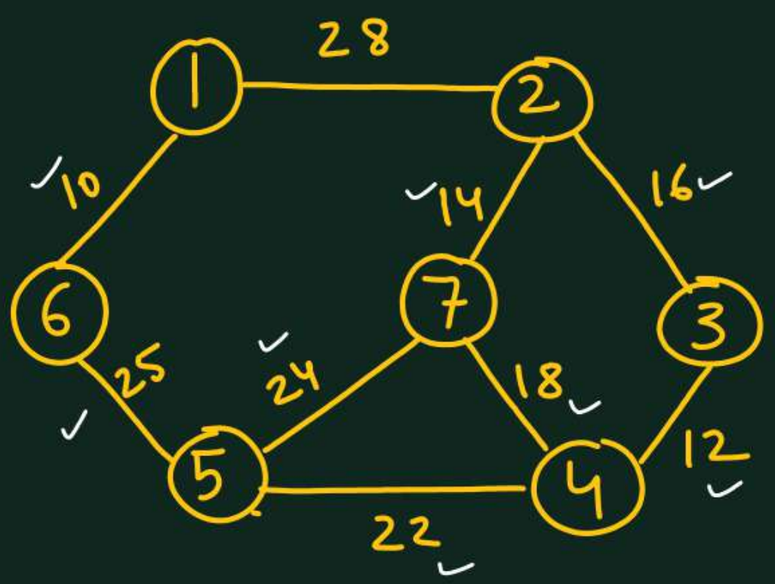


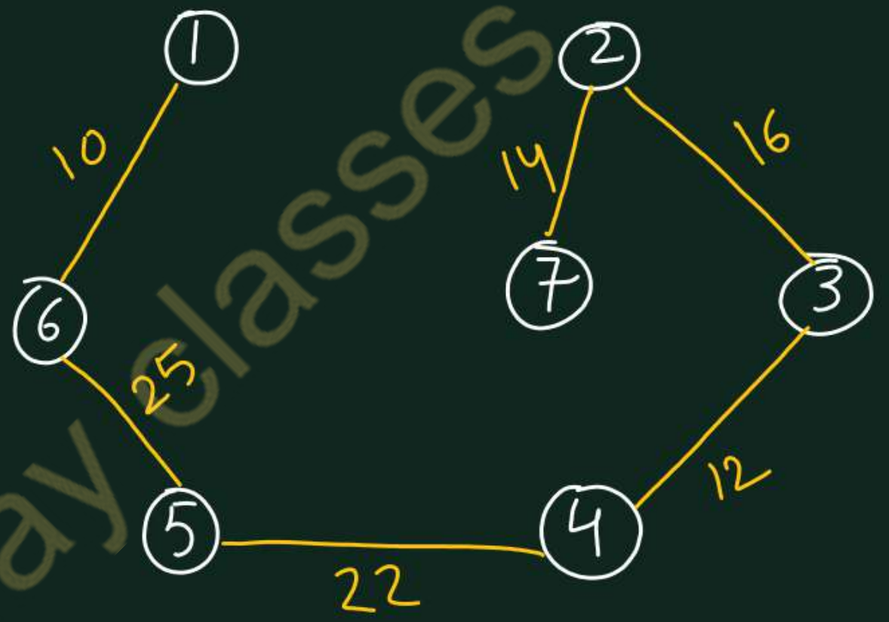




Final MST

Q- Find MST using Kruskals algorithm.





(1-6), (3-4), (2-7), (2-3), (4-7)
10 12 14 16 18

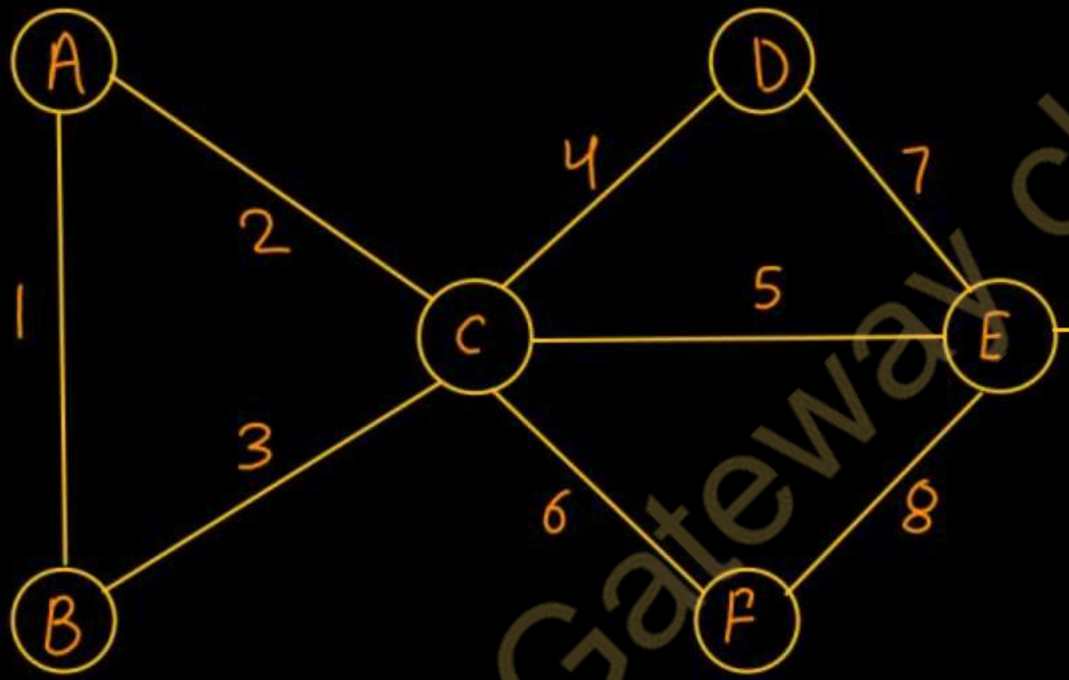
(4-5), (5-7), (5-6), (1-2)
22 24 25 28

Cost of MST = 10+12+14+16+22+25

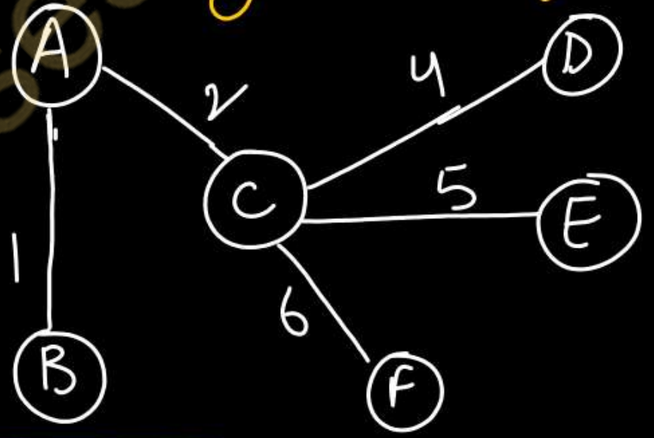
Important points about Prim's and Kruskal's Algorithm

1. If all the edge weights are distinct, then both the algorithms are guaranteed to find the same MST.

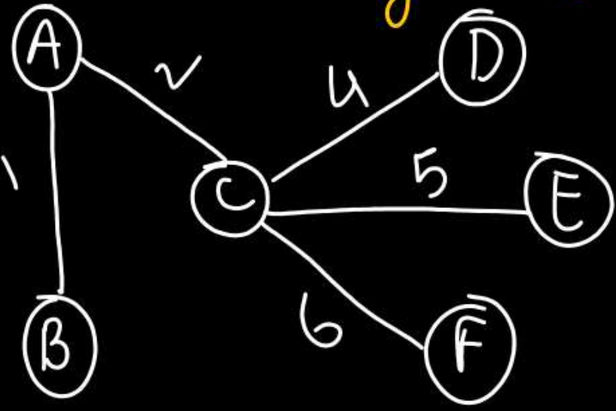
Example:



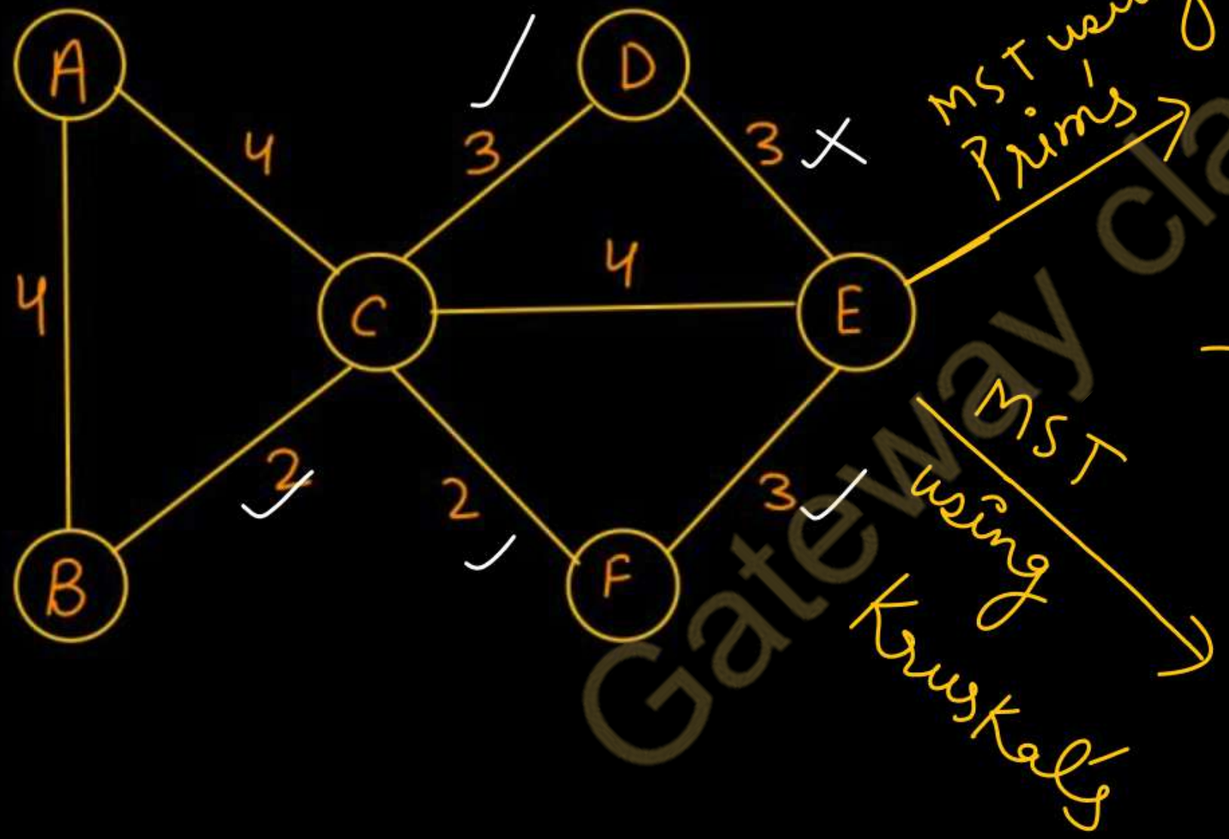
MST using Prim's Algo

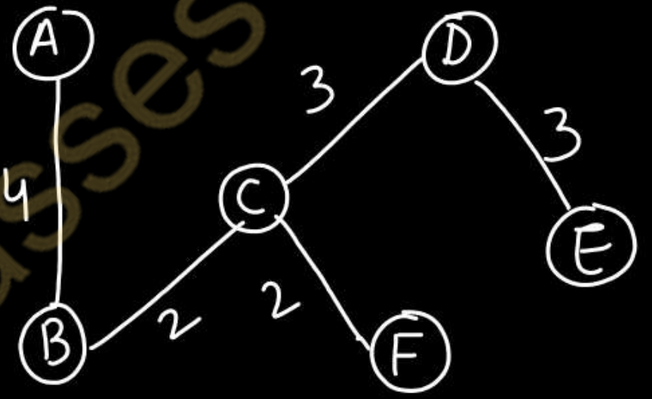


MST using Kruskal's algo

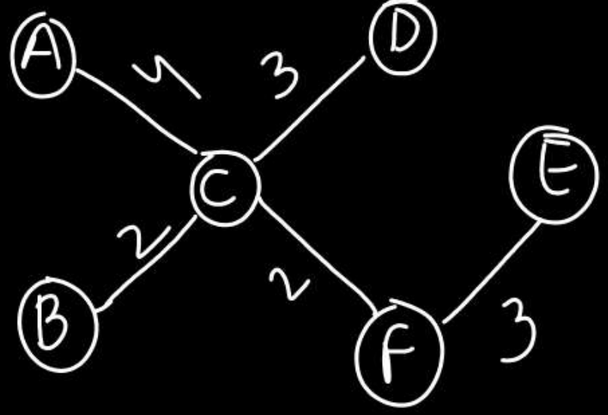


2. If all the edge weights are not distinct, then both the algorithms may not always produce the same MST. However, cost of both the MSTs would always be same in both the cases.





2+2+3+3+4



2+2+3+3+4

Kruskal's Algorithm is preferred when-

- The graph is sparse.
- There are less number of edges in the graph like $E = O(V)$
- The edges are already sorted or can be sorted in linear time.

Prim's Algorithm is preferred when-

- The graph is dense.
- There are large number of edges in the graph like $E = O(V^2)$.

Difference Between Prim's and Kruskal's Algorithm

Feature	Prim's Algorithm	Kruskal's Algorithm
Approach	Vertex-based, grows the MST one vertex at a time	Edge-based, adds edges in increasing order of weight
Graph Representation	Adjacency matrix	Edge list
Initialization	Starts from an arbitrary vertex	Starts with all vertices as separate trees (forest)
Edge Selection	Chooses the minimum weight edge from the connected vertices	Chooses the minimum weight edge from all edges
Suitable for	Dense graphs	Sparse graphs
Starting Point	Requires a starting vertex	No specific starting point, operates on global edges

AKTU PYQs

1. Define spanning tree. Write Kruskal's algorithm for finding minimum cost spanning tree. Describe how Kruskal's algorithm is different from Prim's algorithm for finding minimum cost spanning tree.

2. Prove that if the weights on the edge of the connected undirected graph are distinct then there is a unique Minimum Spanning Tree. Give an example in this regard. Also discuss Kruskal's Minimum Spanning Tree in detail.

AKTU

B.Tech 5th Sem

GW GATEWAY CLASSES

CS IT & CS Allied

DAA : Design & Analysis Of Algorithm

Unit-3 : Lecture-4

Today's Target

- Single Source Shortest path problem
- Dijkstra's Algorithm
- AKTU PYQs

By Dr. Nidhi Parashar Ma'am

- M.Tech Gold Medalist
- Net Qualified

Dijkstra's Algorithm

- Solves single source shortest path problem on a weighted and directed/undirected graph G=(V,E) where all edge weights are non-negative.
- Ensures optimal solution.

Relax(u, v, w)

$d(v) > d(u) + w(u,v)$

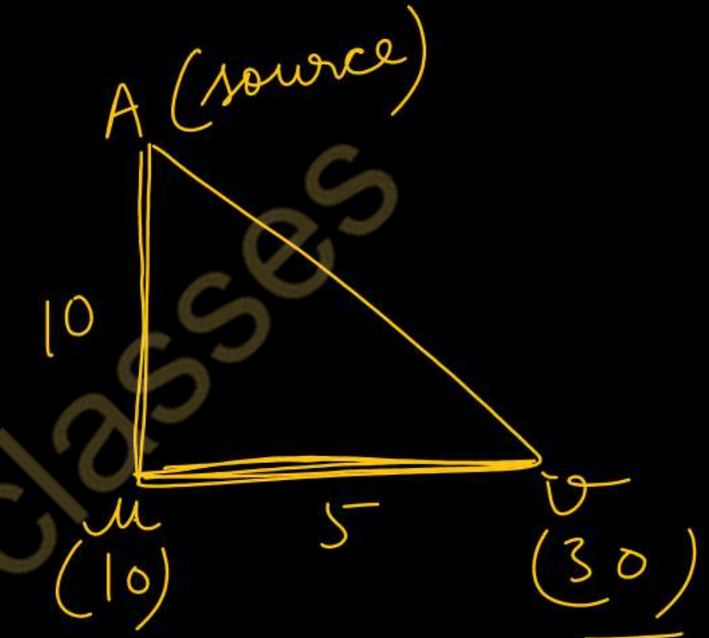
$30 > 10 + 5$

15

$d(v) = d(u) + w(u,v)$

Dijkstra's Algorithm Pseudocode

```
DIJKSTRA(G,w,s)
1  INITIALIZE-SINGLE-Source(G,s)
2  S=∅      // Empty Solution Set
3  Q=G.V
4  while Q≠∅
5      u=EXTRACT-MIN(Q)
6      S=S∪{u}
7      for each vertex v∈G.Adj[u]
8          RELAX(u,v,w)
```




```
RELAX(u,v,w)
1  if v.d > u.d + w(u,v)
2      v.d = u.d + w(u,v)
3      v.π = u
```

```
INITIALIZE-SINGLE-SOURCE(G,s)
1  for each vertex v∈G.V
2      v.d = ∞
3      v.π = NIL
4  s.d = 0
```

Dijkstra's Algorithm Pseudocode

[Time Complexity Analysis]

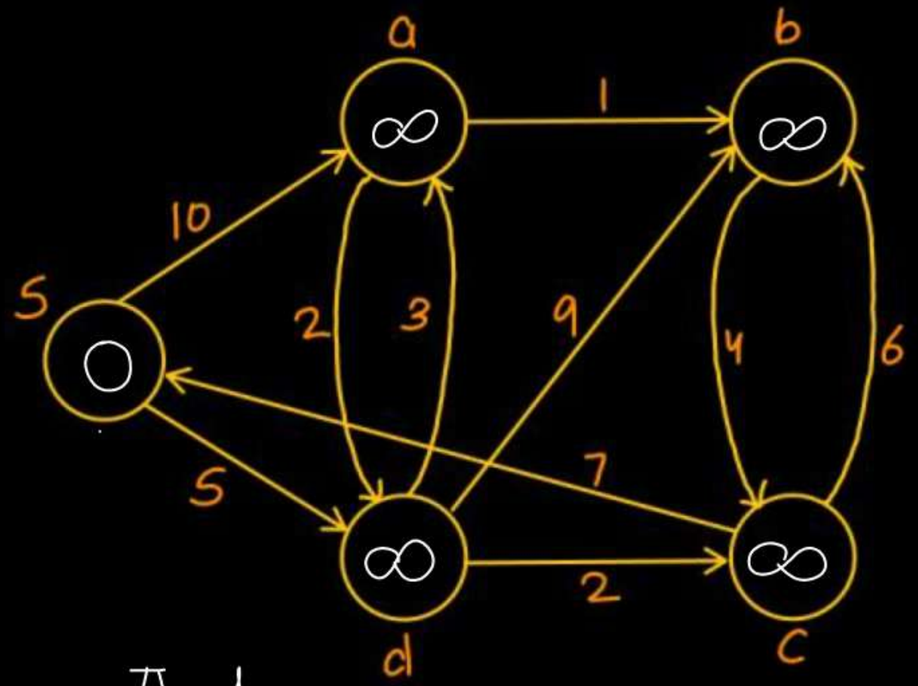
```
DIJKSTRA(G,w,s)
1 INITIALIZE-SINGLE-Source(G,s)        O(V)
2 S=∅
3 Q=G.V // Build heap ⇒               O(V)
4 While Q≠∅
5     u=EXTRACT-MIN(Q) // To extract one minimum vertex from heap = O(log V)
6     S=SU{u}                          for extracting V vertices = V log V
7     for each vertex v∈G.Adj[u]
8         RELAX(u,v,w) // Decrease Key operation ( log (V))
```

```
RELAX(u,v,w)
1 if v.d > u.d + w(u,v)
2     v.d = u.d + w(u,v)
3     v.π = u
```

```
INITIALIZE-SINGLE-SOURCE(G,s)
1 for each vertex v∈G.V
2     v.d = ∞                          O(V)
3     v.π = NIL
4 s.d = 0
```

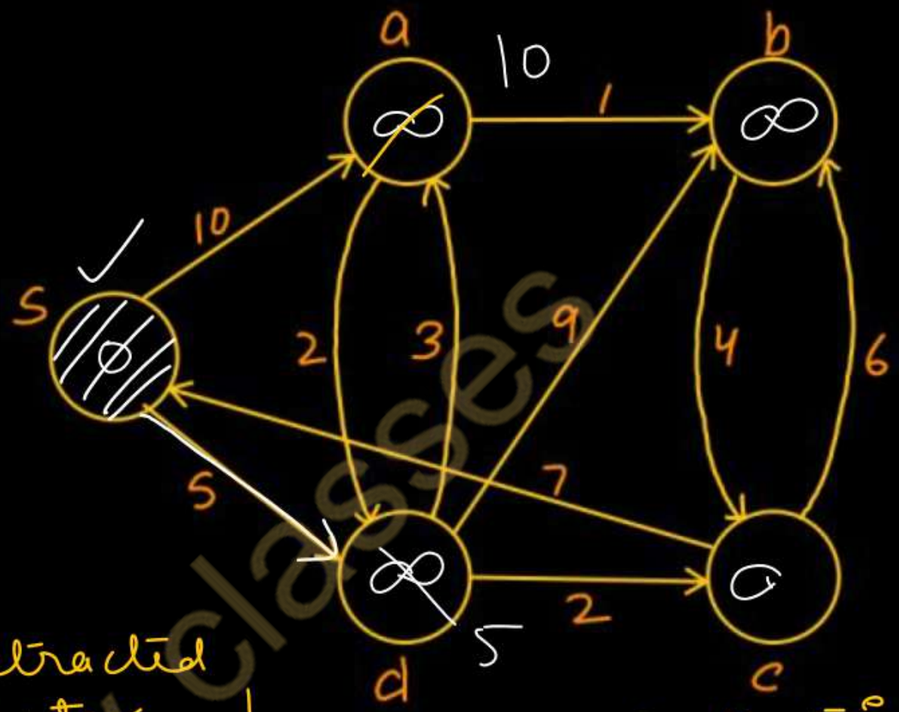
For all edges = E log V

Time Complexity = O(V) + O(V) + (V log V) + (E log V) ⇒ E log V



π, d
S(–, 0)
a(–, ∞)
b(–, ∞)
c(–, ∞)
d(–, ∞)

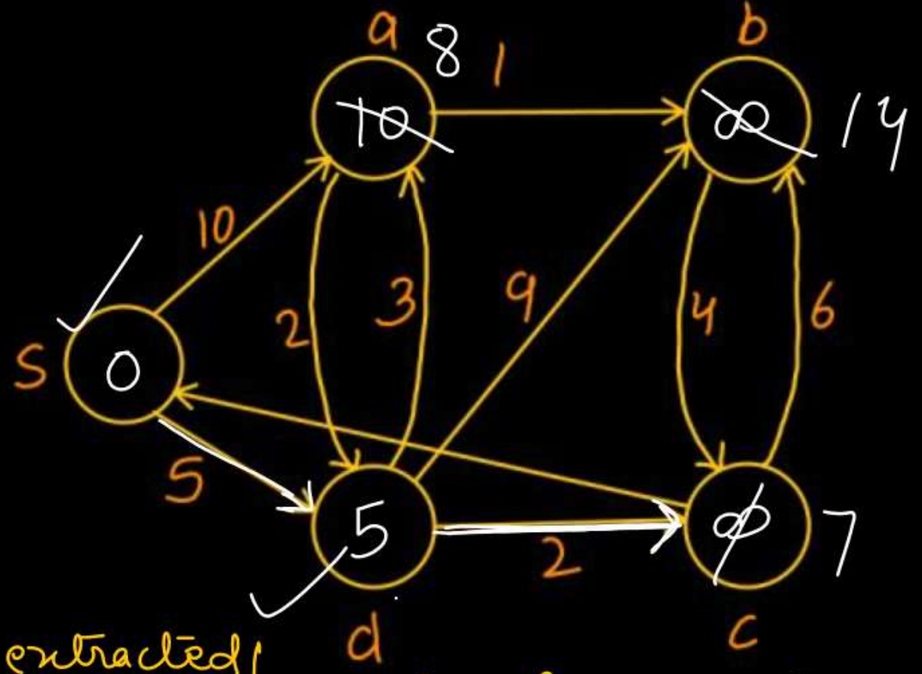
extracted Vertex
S(–, 0)



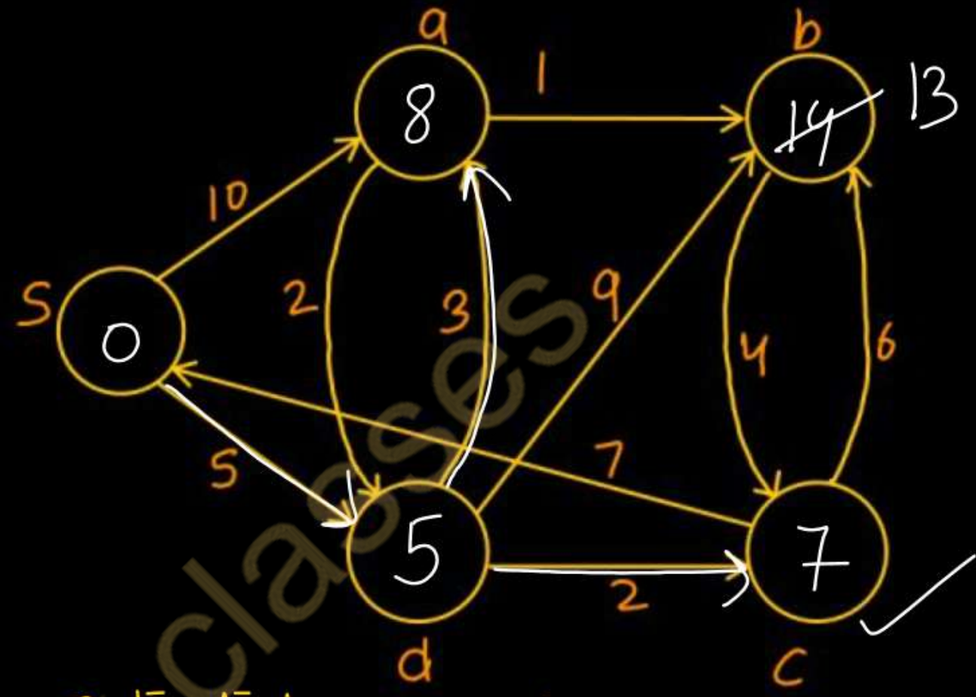
Remaining Vertices

a(S, 0+10) c(–, ∞)

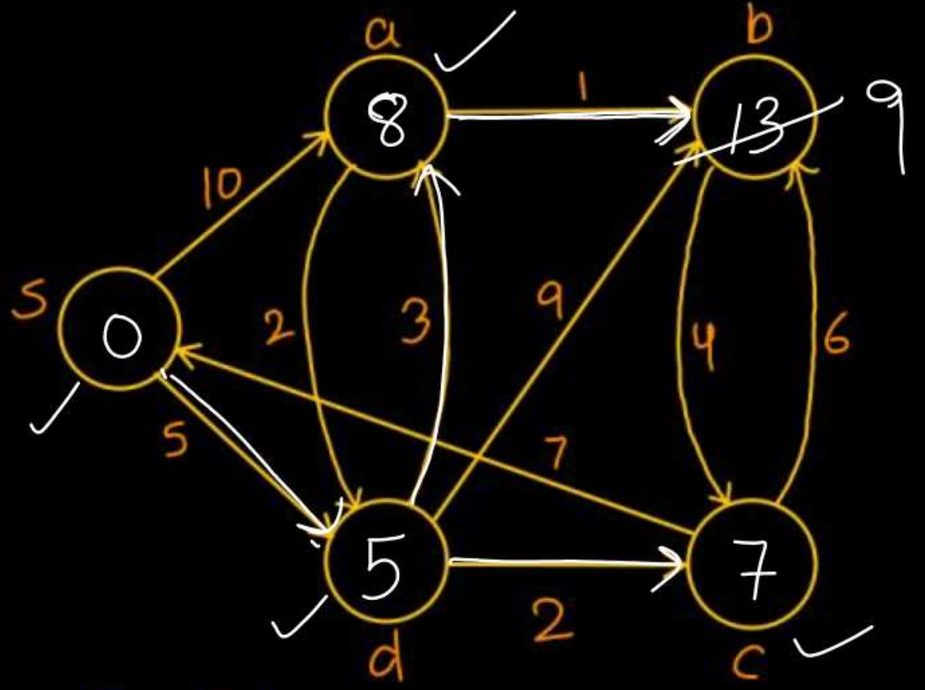
b(–, ∞) d(S, 5)

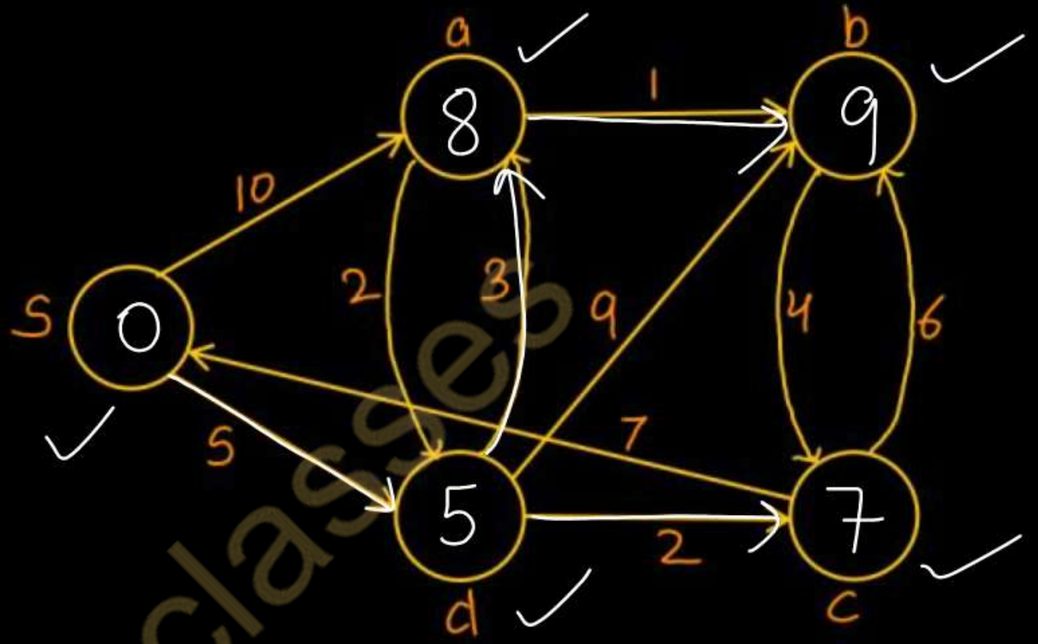


extracted Vertex	Remaining vertices	
d(S,5)	a(d,8)	c(d,7)
	b(d,14)	



extracted Vertex	Remaining Vertex
c(d,7)	a(d,8)
	b(c,13)

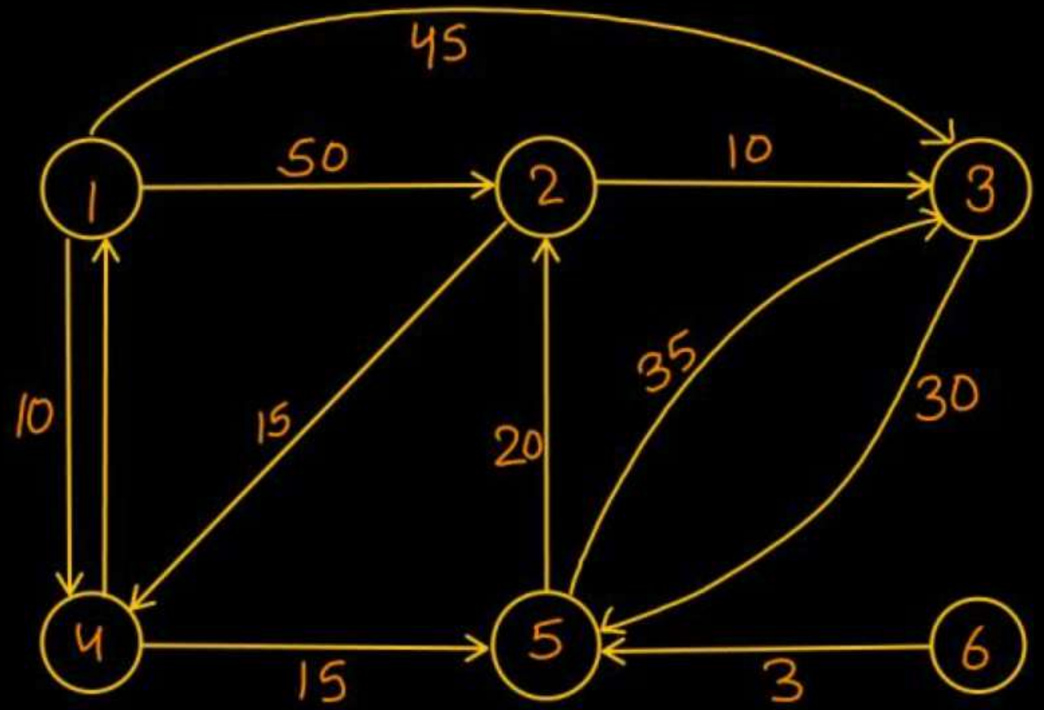




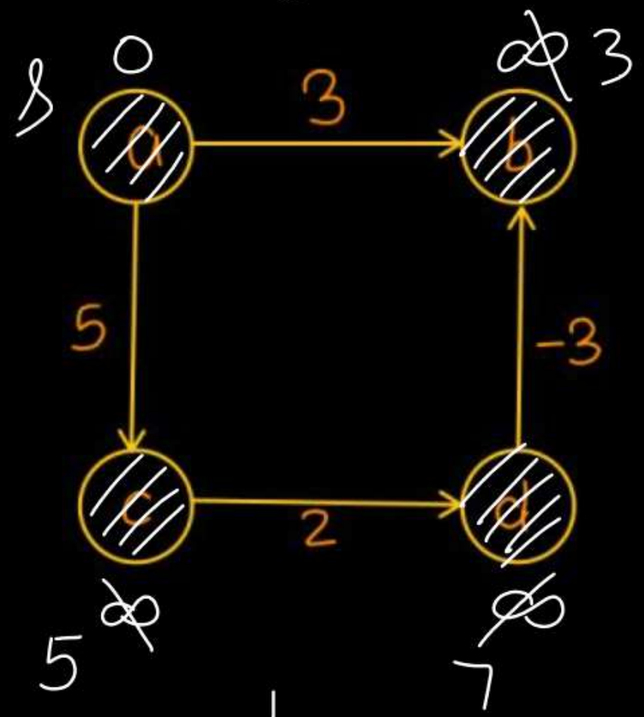
Extracted Vertex	Remaining Vertices
a(d, 8)	b(a, 9)

All vertices are Extracted.

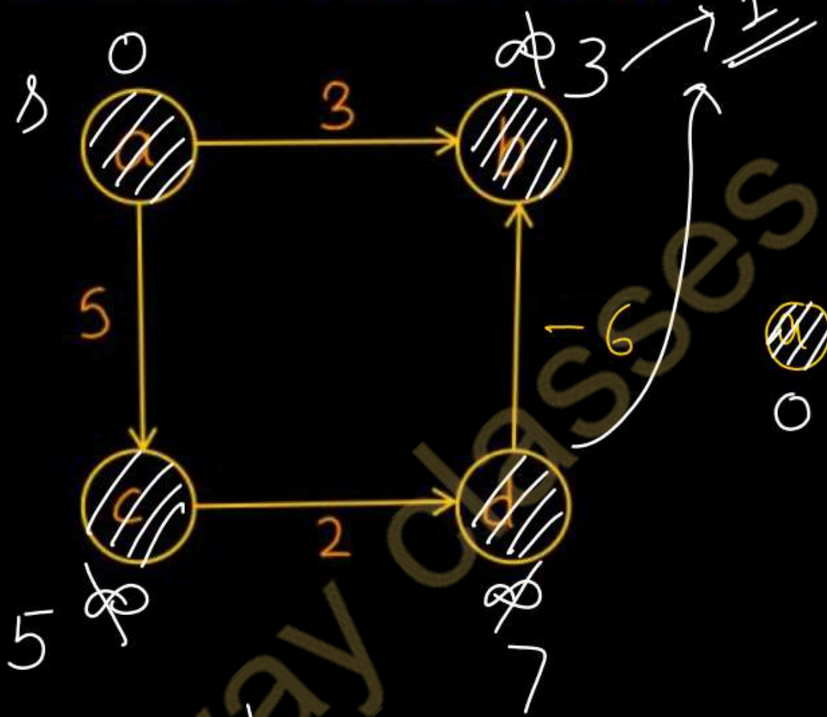
DPP



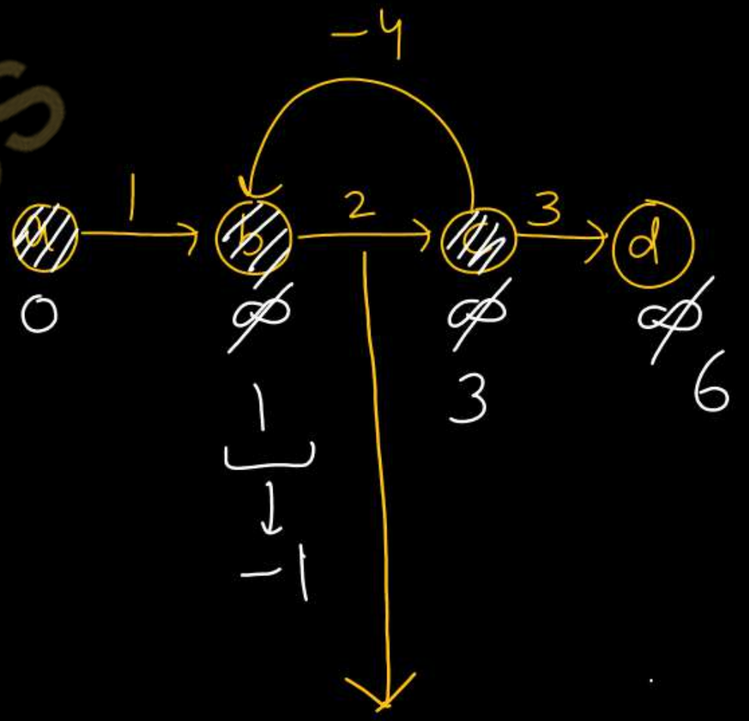
Working on Negative Edge



Dijkstra algo is working



Dijkstra algo is not working



negative weight cycle

2-4=-2

Disadvantage of Dijkstra's Algo — (Important)

(1) It doesn't ensure correct result if negtive weights are present in the graph edges.

(2) If -negative weight cycle is present in the graph, then it will surely fail

AKTU PYQs

1. What are single source shortest paths? Write down Dijkstra's algorithm for it. (AKTU 2022-23)
2. Write an algorithm of Dijkstra and implement it by taking an example. (AKTU 2023-24)

AKTU

B.Tech 5th Sem

GATEWAY CLASSES

CS IT & CS Allied

DAA : Design & Analysis Of Algorithm

Unit-3 : Lecture-5

Today's Target

- Single Source Shortest path problem
- Bellman Ford Algorithm
- AKTU PYQs

By Dr. Nidhi Parashar Ma'am
- M.Tech Gold Medalist
- Net Qualified

Bellman Ford Algorithm

→ Solves single source shortest path problem on weighted graph.

→ Can work on both directed or undirected (Same for Dijkstra)

→ follows dynamic programming approach.
(while Dijkstra algo follows greedy approach)

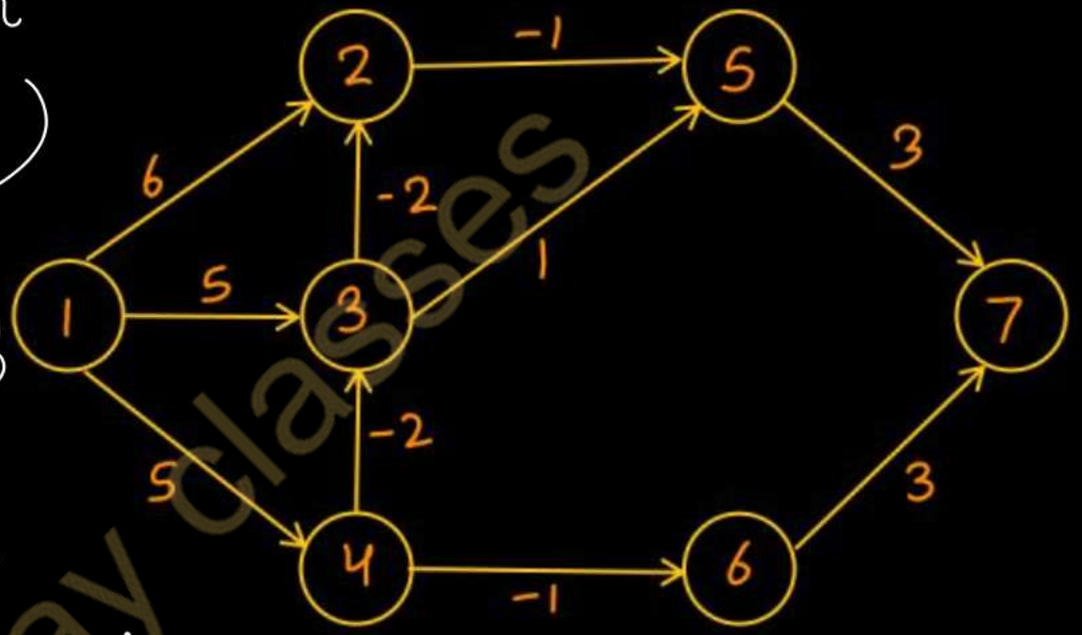
undirected graph

directed graph

⇒ Bellman-ford algo can detect the presence of negative weight cycle in a given graph.

Bellman Ford Algorithm

Idea → If there are V vertices in the graph and E edges then Relax all 'E' edges (|V|−1) times.



→ In the given graph, there are total 7 vertices so we will 'Relax' all the edges present in the graph [|V|−1] times means 6 times.

Total no. of iterations = |V|−1

⇒ (6)

Bellman Ford Algorithm

No of Iterations = 6 (Maximum)

Iteration 1 → Initialization

edgelist

① $(1 \to 2) \Rightarrow (0+6) < \infty \Rightarrow$ Relax ✓

② $(1 \to 3) \Rightarrow (0+5) < \infty \Rightarrow$ Relax ✓

③ $(1 \to 4) \Rightarrow (0+5) < \infty \Rightarrow$ Relax

④ $(2 \to 5) \Rightarrow (6+(-1)) < \infty \Rightarrow$ Relax ✓

⑤ $(3 \to 2) \Rightarrow (5+(-2)) < 6 \Rightarrow$ Relax

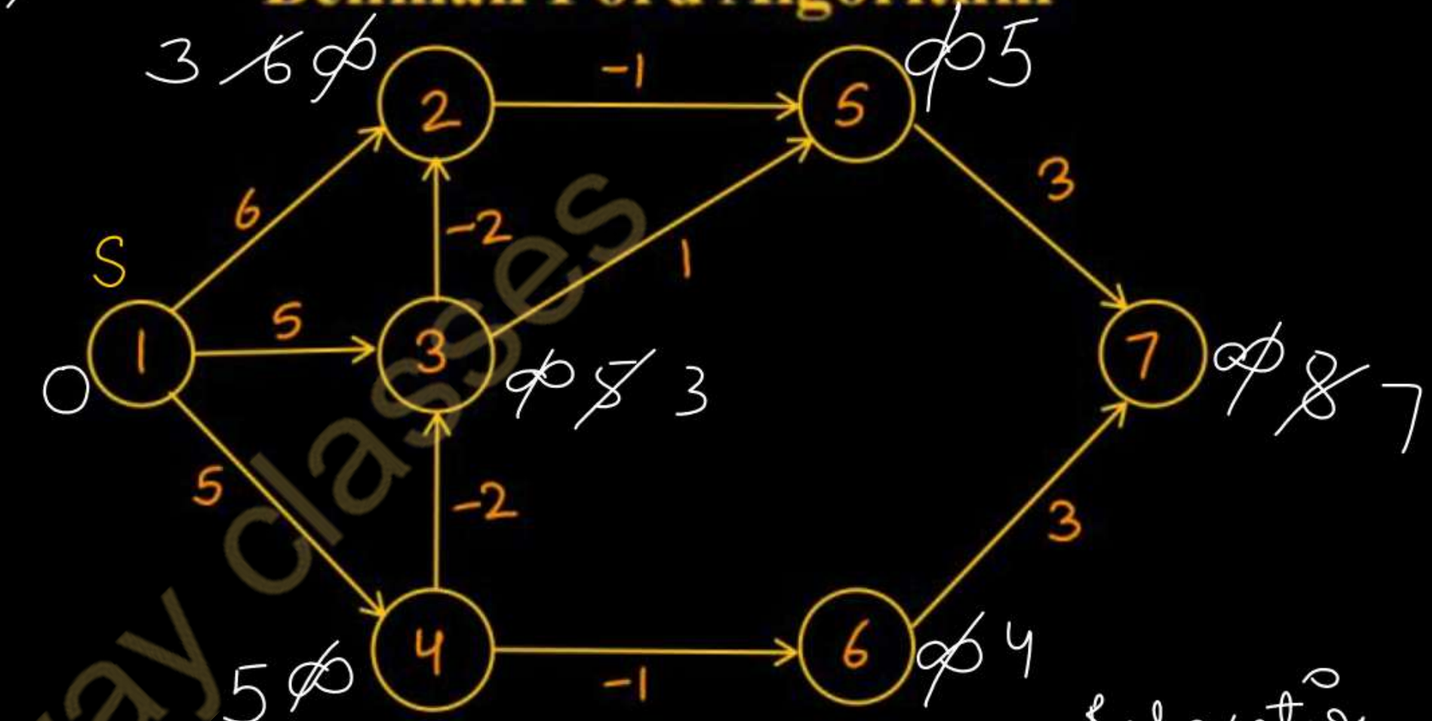
⑥ $(3 \to 5) \Rightarrow (5+1) > 5 \Rightarrow$ Key value not updated

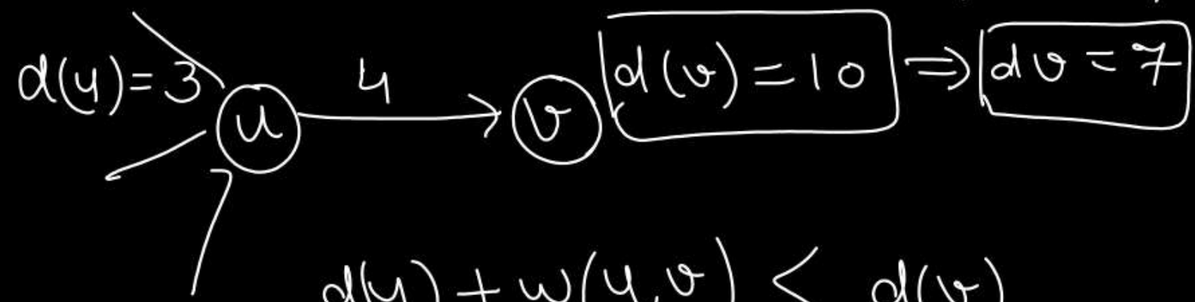
⑦ $(4 \to 3) \Rightarrow (5+(-2)) < 5 \Rightarrow$ Relax

⑧ $(4 \to 6) \Rightarrow (5+(-1)) < \infty \Rightarrow$ Relax

⑨ $(5 \to 7) \Rightarrow (5+3) < \infty \Rightarrow$ Relax

⑩ $(6 \to 7) \Rightarrow (4+3) < \infty \Rightarrow$ Relax ✓





$d(u) + w(u,v) < d(v)$

$3 + 4 < 10$

Bellman Ford Algorithm

Iteration-2

Edge-List

$(1 \to 2) \Rightarrow 0+6 > 3 \Rightarrow$ Not Relaxed

$(1 \to 3) \Rightarrow 0+5 > 3 \Rightarrow$ Not Relaxed

$(1 \to 4) \Rightarrow 0+5 = 5 \Rightarrow$ Not Relaxed

$(2 \to 5) \Rightarrow 3+(-1) < 5 \Rightarrow$ Relax ✓ ✓

$(3 \to 2) \Rightarrow 3+(-2) < 3 \Rightarrow$ Relax ✓ ✓

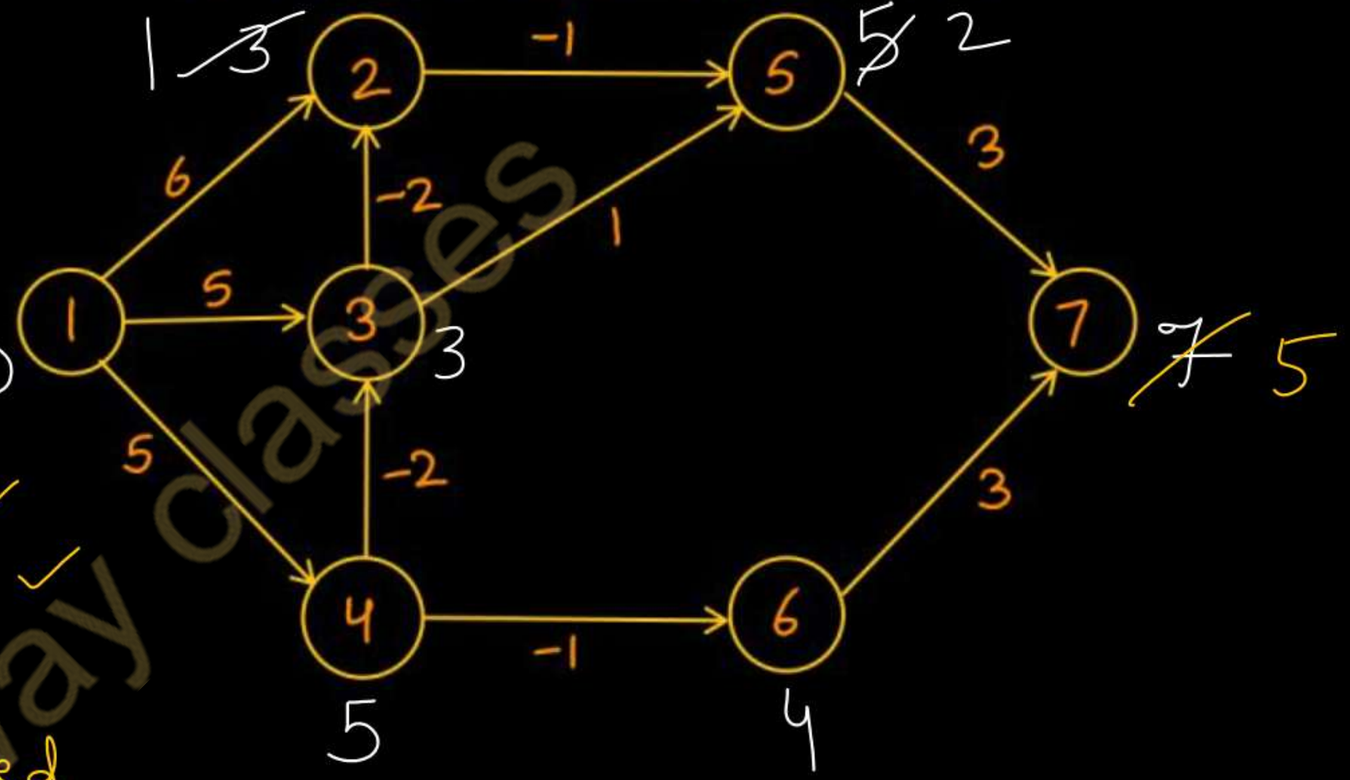
$(3 \to 5) \Rightarrow 3+1 > 2 \Rightarrow$ Not Relaxed

$(4 \to 3) \Rightarrow (5+(-2)) = 3 \Rightarrow$ Not Relaxed

$(4 \to 6) \Rightarrow 5+(-1) = 4 \Rightarrow$ Not Relaxed

$(5 \to 7) \Rightarrow (2+3) < 7 \Rightarrow$ Relax ✓ ✓

$(6 \to 7) \Rightarrow (4+7) > 5 \Rightarrow$ Not Relaxed



Bellman Ford Algorithm

Iteration-3

Edge list.

$(1 \to 2) \Rightarrow$ Not Relaxed

$(1 \to 3) \Rightarrow$ Not Relaxed

$(1 \to 4) \Rightarrow$ Not Relaxed

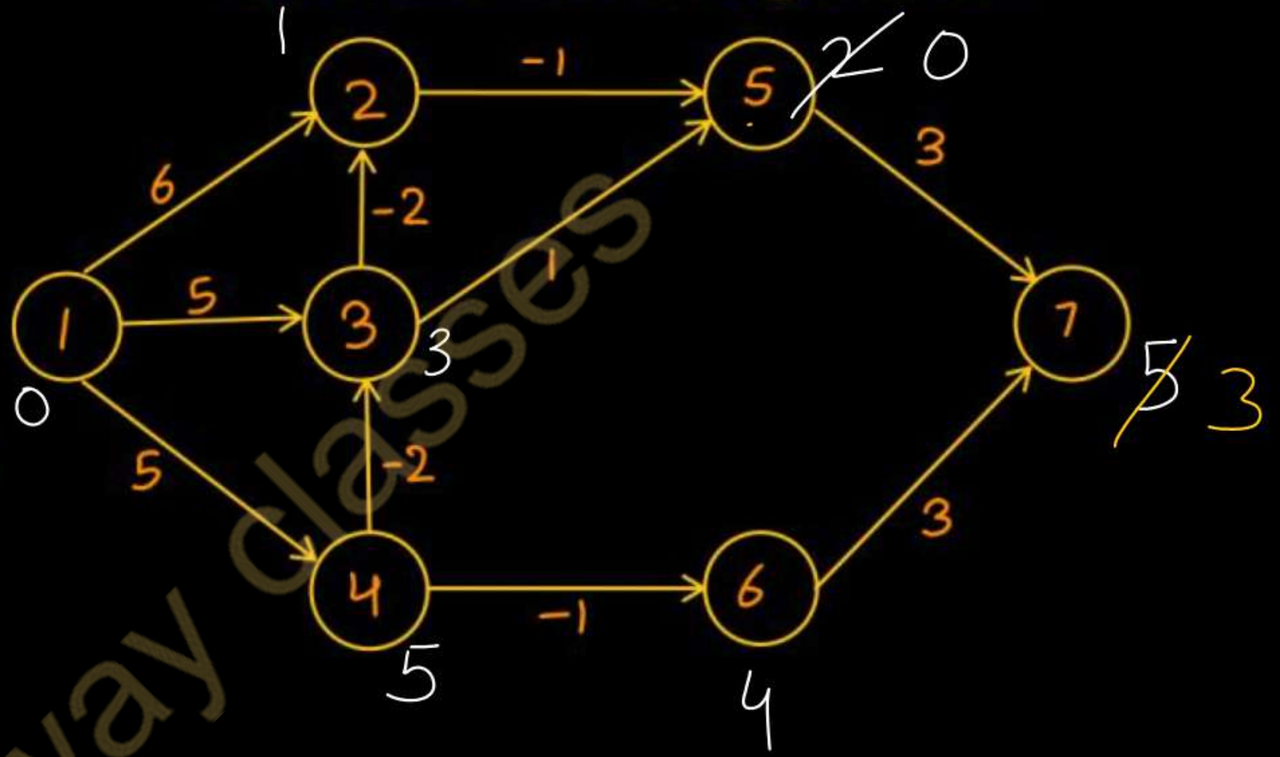
$(2 \to 5) \Rightarrow 1 + (-1) < 2 \Rightarrow$ Relax

$(3 \to 2), (3 \to 5) \Rightarrow$ Not Relaxed

$(4 \to 3), (4 \to 6) \Rightarrow$ Not Relaxed

$(5 \to 7) \Rightarrow 0 + 3 < 5 \Rightarrow$ Relax

$(6 \to 7) \Rightarrow$ Not Relaxed



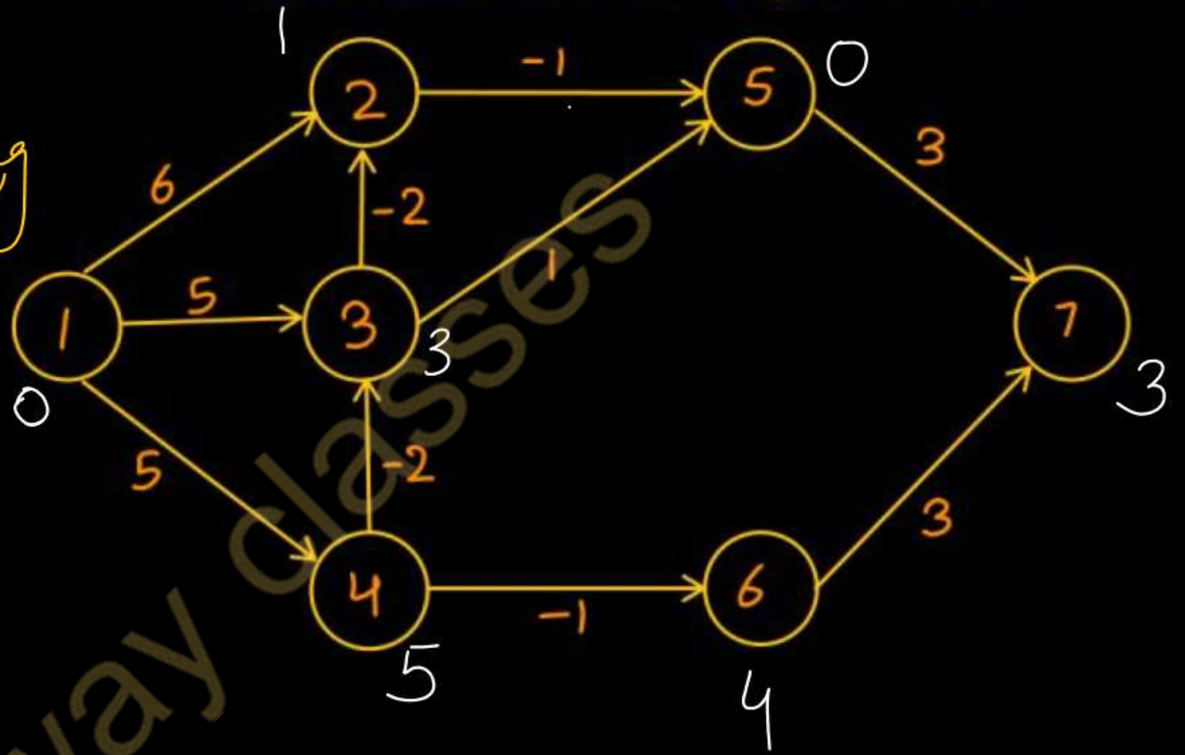
Bellman Ford Algorithm

Iteration - 4

Edge list

(1→2)
(1→3)
(1→4)
(2→5)
(3→2)
(3→5)
(4→3)
(4→6)
(5→7)
(6→7)

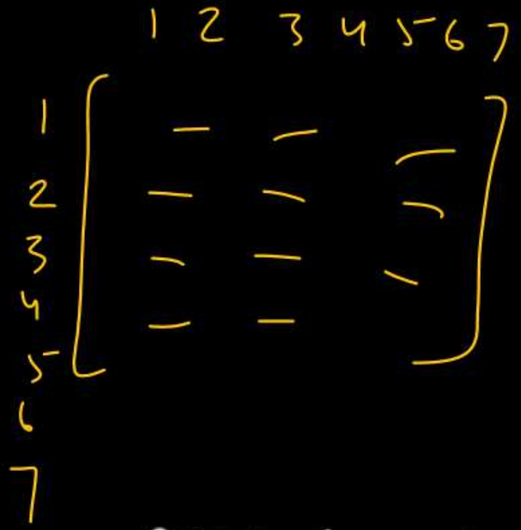
No edges are relaxing in this iteration so we have reached to the solution.



Solution

1→2	1
1→3	3
1→4	5
1→5	0
1→6	4
1→7	3

Bellman Ford Algorithm



```
BELLMAN-FORD(G,w,s)
1 INITIALIZE-SINGLE-SOURCE(G,s)
2 for i=1 to |G.V|-1
3     foreach edge(u,v)∈G.E
4         RELAX(u,v,w)
5 for each edge(u,v)∈G.E
6     if v.d > u.d+w(u,v)
7         return FALSE
8 return TRUE
```

This code detects the presence of negative weight cycle.

```
RELAX(u,v,w)
1 if v.d > u.d+w(u,v)
2     v.d = u.d+w(u,v)
3     v.π = u
```

```
INITIALIZE-SINGLE-SOURCE(G,s)
1 for each vertex v∈G.V
2     v.d = ∞
3     v.π = NIL
4 s.d = 0
```

(Time Complexity)

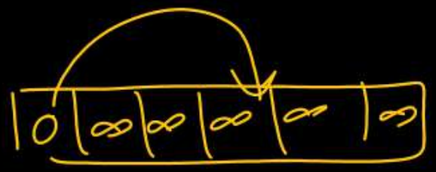
Bellman Ford Algorithm

```
BELLMAN-FORD(G,w,s)
1 INITIALIZE-SINGLE-SOURCE(G,s)        → O(V)
2 for i=1 to |G.V|-1                   O(|V|-1)
3     foreach edge(u,v)∈G.E            O(E)
4         RELAX(u,v,w)                 O(1)
5 for each edge(u,v)∈G.E                O(E)
6     if v.d > u.d+w(u,v)
7         return FALSE
8 return TRUE
```

$O(V) + O((|V|-1) \times O(E)) + O(E)$

$\Rightarrow O(V \cdot E) + O(E)$

$= O(V \cdot E)$



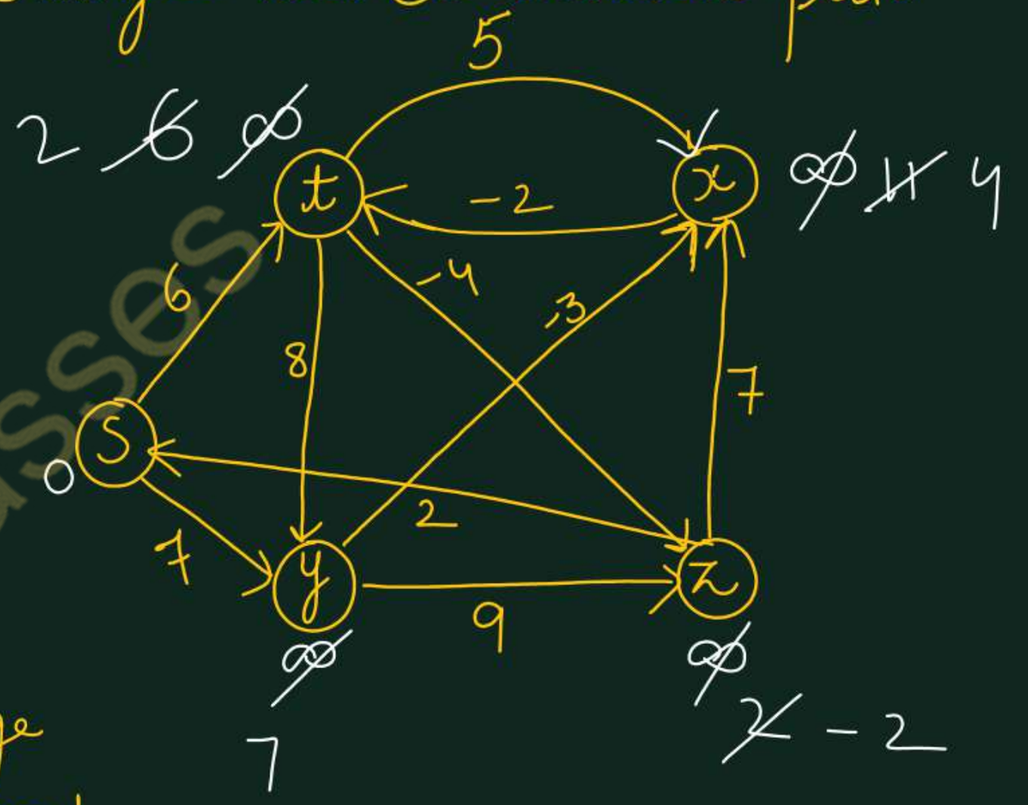

```
RELAX(u,v,w)
1 if v.d > u.d+w(u,v)
2     v.d = u.d+w(u,v)
3     v.π = u
```

$O(1)$

```
INITIALIZE-SINGLE-SOURCE(G,s)
1 for each vertex v∈G.V
2     v.d = ∞
3     v.π = NIL
4 s.d = 0
```

$O(V)$

Ques - Apply Bellman ford algorithm to solve single source shortest path problem on the following graph.



edge list	Iteration-1	Iteration-2	Iteration-3
$(s \to t)$	✓	×	×
$(s \to y)$	✓	×	×
$(t \to x)$	✓	×	×
$(t \to y)$	×	×	×
$(t \to z)$	✓	✓	×
$(y \to x)$	✓	×	×
$(y \to z)$	×	×	×
$(x \to t)$	✓	×	×
$(z \to s)$	×	×	×
$(z \to x)$	×	×	×

no edge is relaxed even in 4th iteration

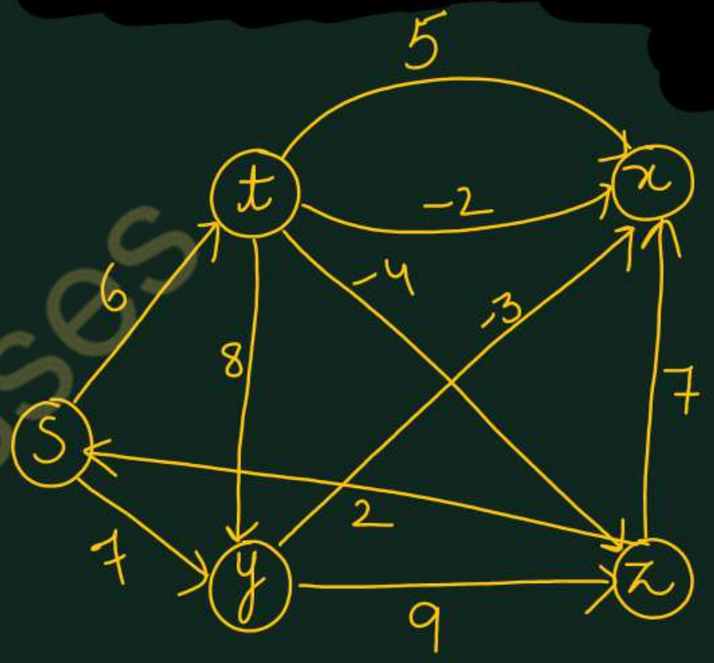
$|V| = 5$

Iterations $= 5 - 1 = 4$

To detect -ve weight cycle —

→ first show the working of 4 iterations.

→ After that perform one additional iteration to detect the presence of negative weight cycle.

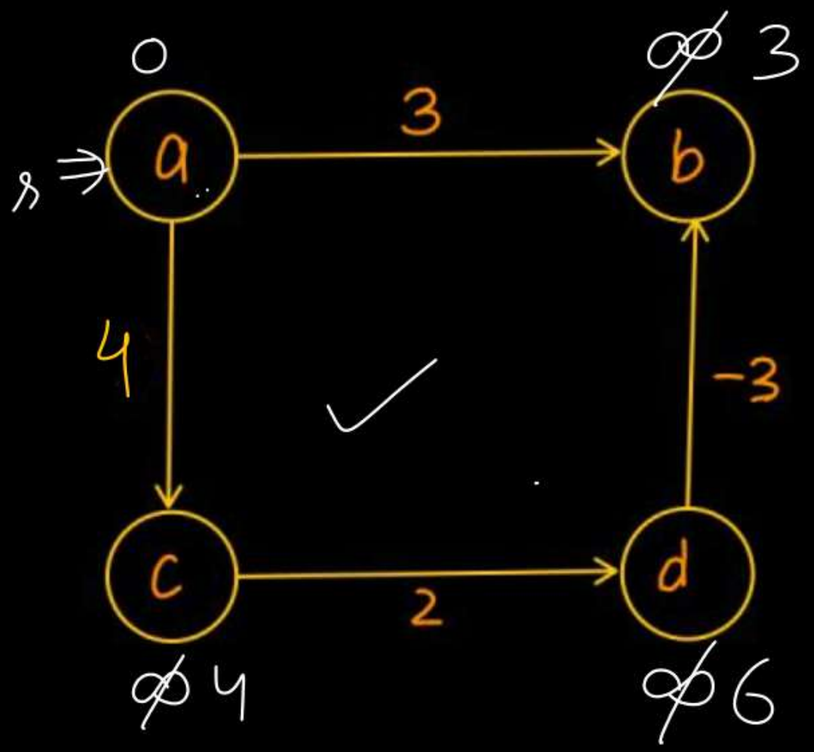


Note

1) Bellman-ford algorithm can detect presence of negative weight cycle by can not locate it directly.

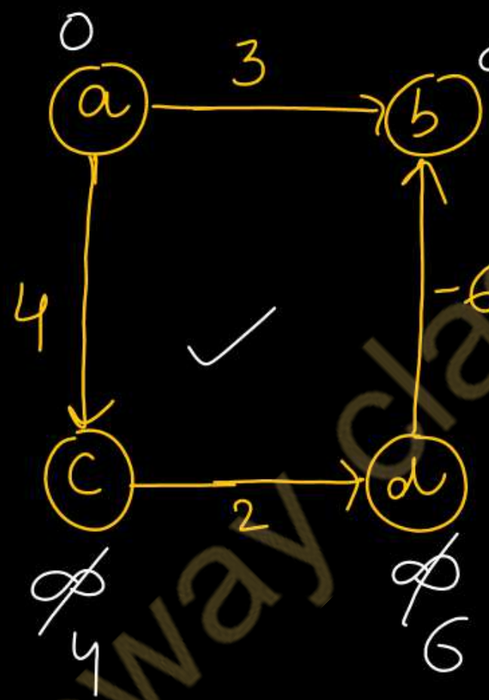
Working on Negative Edge

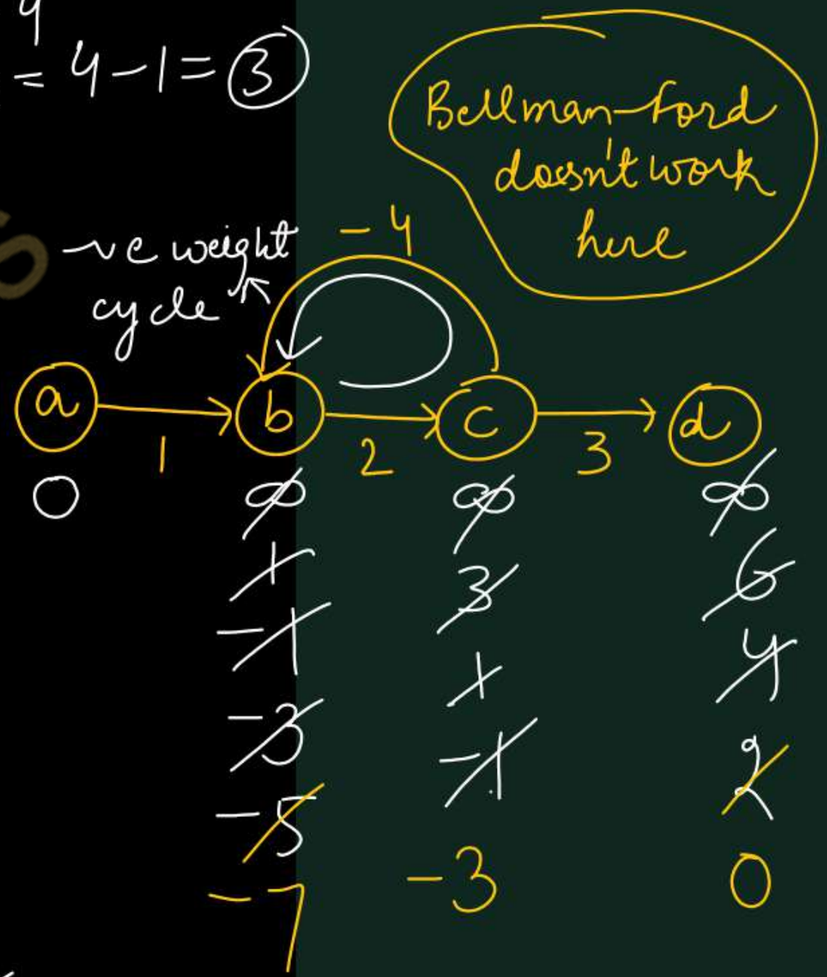
no. of vertices = 4
iterations = 4 − 1 = (3)



Iteration

a → b
a → c
c → d
d → b





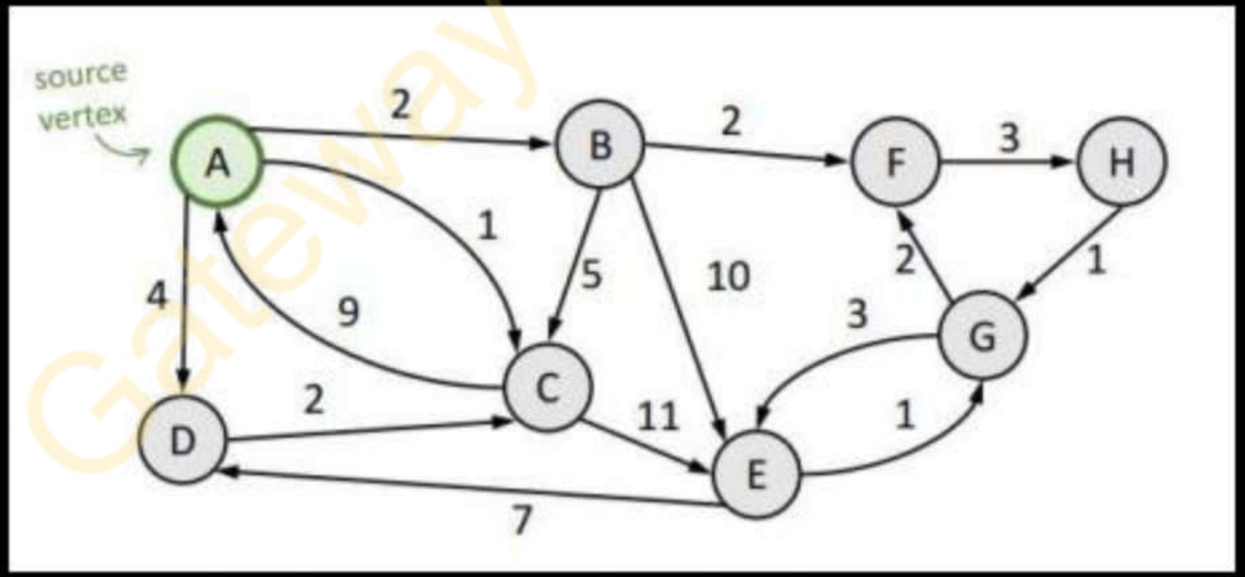
I-1 ✓
I-2 ✓
I-3 ✓
I-4 ⇒ edges are relaxing.

Drawback of Bellman ford Algorithm

1) Does not work if negative weight cycle is present.
2) Slower than Dijkstra's algorithm so it is less efficient for large graphs. (where no. of vertices are more).
3) It is not effective in case of topology change as updates are spread node by node ⇒ slow process

AKTU PYQs

1. Explain Bellman ford algorithm for single source shortest path in detail. (AKTU 2019-20)
2. What is the drawback of bellman ford algorithm. (AKTU 2015-16).
3. When do Dijkstra and the Bellman-Ford algorithm both fail to find a shortest path? Can Bellman ford detect all negative weight cycles in a graph? Apply Bellman Ford Algorithm on the following graph:



AKTU

B.Tech 5th Sem

GW GATEWAY CLASSES

CS IT & CS Allied

DAA : Design & Analysis Of Algorithm

Unit-3 : Lecture-6

Today's Target

Divide and Conquer

- Strassen's Matrix Multiplication
- Binary Search
- AKTU PYQs

By Dr. Nidhi Parashar Ma'am

- M.Tech Gold Medalist
- Net Qualified

Matrix Multiplication

$$A = \begin{array}{c|cc} & 1 & 2 \\ 1 & a_{11} & a_{12} \\ 2 & a_{21} & a_{22} \end{array} \qquad B = \begin{array}{c|cc} & 1 & 2 \\ 1 & b_{11} & b_{12} \\ 2 & b_{21} & b_{22} \end{array}$$

Resultant Matrix

$$C = A * B$$

$2 \times \boxed{2 \quad 2} \times 2$

Can be multiplied

$$C = \begin{array}{c|cc} & 1 & 2 \\ 1 & c_{11} & c_{12} \\ 2 & c_{21} & c_{22} \end{array}$$

$$c_{11} = a_{11} * b_{11} + a_{12} * b_{21}$$

$$c_{12} = a_{11} * b_{12} + a_{12} * b_{22}$$

$$c_{21} = a_{21} * b_{11} + a_{22} * b_{21}$$

$$c_{22} = a_{21} * b_{12} + a_{22} * b_{22}$$

...Matrix Multiplication

i → rows
j = columns

```
for( i=1; i<=2; i++)
  {
    for(j=1; j<=2; j++)
    {
        c[i][j]=0;
        for(K=1; K<=2; k++)
        {
          c[i][j]= c[i][j]+ a[i][k]*b[k][j];
        }
    }
  }
```

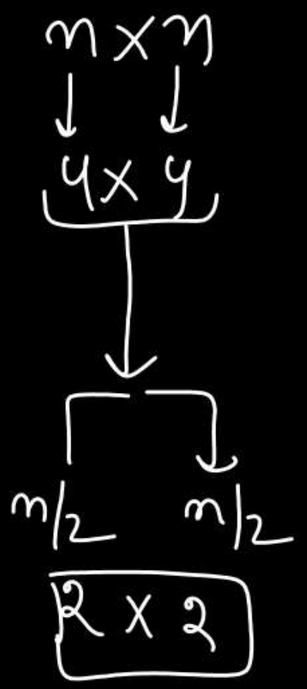
For n x n Matrix

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

Matrix Multiplication (Divide and Conquer)

Assume that order of Matrix is in the power of 2.

Suppose we have two matrices A and B, both (4 x 4)



$$A = \begin{array}{c|cc|cc} & 1 & 2 & 3 & 4 \\ \hline 1 & a_{11} & a_{12} & a_{13} & a_{14} \\ 2 & a_{21} & a_{22} & a_{23} & a_{24} \\ \hline 3 & a_{31} & a_{32} & a_{33} & a_{34} \\ 4 & a_{41} & a_{42} & a_{43} & a_{44} \end{array} \qquad A_{11} \quad A_{12} \quad A_{21} \quad A_{22}$$

$$B = \begin{array}{c|cc|cc} & 1 & 2 & 3 & 4 \\ \hline 1 & b_{11} & b_{12} & b_{13} & b_{14} \\ 2 & b_{21} & b_{22} & b_{23} & b_{24} \\ \hline 3 & b_{31} & b_{32} & b_{33} & b_{34} \\ 4 & b_{41} & b_{42} & b_{43} & b_{44} \end{array} \qquad B_{11} \quad B_{12} \quad B_{21} \quad B_{22}$$

Matrix Multiplication (Divide and Conquer)

Resultant Matrix $C = A * B$

$$C = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

total 4 additions and

8 multiplications → major operation

$$C_{11} = A_{11} * B_{11} + A_{12} * B_{21}$$
$$C_{12} = A_{11} * B_{12} + A_{12} * B_{22}$$
$$C_{21} = A_{21} * B_{11} + A_{22} * B_{21}$$
$$C_{22} = A_{21} * B_{12} + A_{22} * B_{22}$$

All these A_{11}... are 2x2 matrices.

Matrix Multiplication (Divide and Conquer)

(Recursive Solution)

```
MM(A, B, n)        ← n
{ if(n≤2) }  → base Condition
}
C11 = a11 * b11 + a12 * b21
C12 = a11 * b12 + a12 * b22
C21 = a21 * b11 + a22 * b21
C22 = a21 * b12 + a22 * b22
}
else
```

$$T(n) = \underbrace{8}_{\text{No. of subproblems}}\ \underbrace{T(n/2)}_{\text{size of subproblem}} + 4n^2$$

Divide A & B with order n x n into 4 Submat each with order n/2 * n/2

```
{ MM(A11, B11, n/2) + MM(A12, B21, n/2)
  MM(A11, B12, n/2) + MM(A12, B22, n/2)
  MM(A21, B11, n/2) + MM(A22, B21, n/2)
  MM(A21, B12, n/2) + MM(A22, B22, n/2)
}
```

For Divide and Conquer Method

$$T(n) = 8\,T(n/2) + 4n^2$$

$$T(n) = O(n^{\log_2 8})$$

$$\boxed{T(n) = O(n^3)}$$

⇓

same as the time taken by iterative method.

⇓

no additional advantage is achieved.

Space Complexity

→ More than Iterative method because it uses additional space in the form of stack to store function calls.

Strassen's Matrix Multiplication

$$\begin{vmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{vmatrix} = \underbrace{\begin{vmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{vmatrix}}_{A} \times \underbrace{\begin{vmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{vmatrix}}_{B}$$

$P_1 = (A_{11} + A_{22}) * (B_{11} + B_{22})$

$P_2 = (A_{21} + A_{22}) * B_{11}$

$P_3 = A_{11} * (B_{12} - B_{22})$

$P_4 = A_{22} * (B_{21} - B_{11})$

$P_5 = (A_{11} + A_{12}) * B_{22}$

$P_6 = (A_{21} - A_{11}) * (B_{11} + B_{12})$

$P_7 = (A_{12} - A_{22}) * (B_{21} + B_{22})$

$C_{11} = P_1 + P_4 - P_5 + P_7$

$C_{12} = P_3 + P_5$

$C_{21} = P_2 + P_4$

$C_{22} = P_1 + P_3 - P_2 + P_6$

No. of multiplications = (7)

No. of addition/subtractions = (18)

Idea behind Strassen's algo

↓

Reduce the total no. of multiplication operations without caring about the increase in no. of additions or subtractions.

Strassen's Matrix Multiplication (Time Complexity)

$$\begin{vmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{vmatrix} = \begin{vmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{vmatrix} \begin{vmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{vmatrix}$$

$P_1 = (A_{11}+A_{22})(B_{11}+B_{22})$

$P_2 = (A_{21}+A_{22}) * B_{11}$

$P_3 = A_{11} * (B_{12}-B_{22})$

$P_4 = A_{22} * (B_{21}-B_{11})$

$P_5 = (A_{11}+A_{12}) * B_{22}$

$P_6 = (A_{21}-A_{11}) * (B_{11}+B_{12})$

$P_7 = (A_{12}-A_{22}) * (B_{21}+B_{22})$

$C_{11} = P_1 + P_4 - P_5 + P_7$

$C_{12} = P_3 + P_5$

$C_{21} = P_2 + P_4$

$C_{22} = P_1 + P_3 - P_2 + P_6$

$$T(n) = 7\,T(n/2) + 18\,n^2$$

$$T(n) = O(n^{\log_2 7})$$

$$\boxed{T(n) \simeq O(n^{2.8})}$$

⇓

Reduced time complexity for Matrix multiplication.

Strassen's Matrix Multiplication

$$\begin{vmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{vmatrix} = \begin{vmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{vmatrix} \begin{vmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{vmatrix}$$

$P_1 = (A_{11}+A_{22})(B_{11}+B_{22})$

$P_2 = (A_{21}+A_{22}) * B_{11}$

$P_3 = A_{11} * (B_{12}-B_{22})$

$P_4 = A_{22} * (B_{21}-B_{11})$

$P_5 = (A_{11}+A_{12}) * B_{22}$

$P_6 = (A_{21}-A_{11}) * (B_{11}+B_{12})$

$P_7 = (A_{12}-A_{22}) * (B_{21}+B_{22})$

$C_{11} = P_1 + P_4 - P_5 + P_7$

$C_{12} = P_3 + P_5$

$C_{21} = P_2 + P_4$

$C_{22} = P_1 + P_3 - P_2 + P_6$

$$A = \begin{vmatrix} 1 & 2 \\ 3 & 4 \end{vmatrix} \quad B = \begin{vmatrix} 2 & 3 \\ 1 & 4 \end{vmatrix}$$

$C = A \times B$

$$C = \begin{vmatrix} 1\times2+2\times1 & 1\times3+2\times4 \\ 3\times2+4\times1 & 3\times3+4\times4 \end{vmatrix}$$

$$C = \begin{vmatrix} 4 & 11 \\ 10 & 25 \end{vmatrix}$$

Strassen's Matrix Multiplication

$$\begin{vmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{vmatrix} = \begin{vmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{vmatrix} \begin{vmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{vmatrix}$$

$P_1 = (A_{11}+A_{22})(B_{11}+B_{22})$

$P_2 = (A_{21}+A_{22}) * B_{11}$

$P_3 = A_{11} * (B_{12}-B_{22})$

$P_4 = A_{22} * (B_{21}-B_{11})$

$P_5 = (A_{11}+A_{12}) * B_{22}$

$P_6 = (A_{21}-A_{11}) * (B_{11}+B_{12})$

$P_7 = (A_{12}-A_{22}) * (B_{21}+B_{22})$

$C_{11} = P_1 + P_4 - P_5 + P_7$

$C_{12} = P_3 + P_5$

$C_{21} = P_2 + P_4$

$C_{22} = P_1 + P_3 - P_2 + P_6$

$A = \begin{vmatrix} 1 & 2 \\ 3 & 4 \end{vmatrix}$ $B = \begin{vmatrix} 2 & 3 \\ 1 & 4 \end{vmatrix}$

$A_{11}=1$, $A_{12}=2$, $A_{21}=3$, $A_{22}=4$

$B_{11}=2$, $B_{12}=3$, $B_{21}=1$, $B_{22}=4$

$P_1 = (1+4) * (2+4) = 30$

$P_2 = (3+4) * 2 = 14$

$P_3 = 1 * (3-4) = -1$

$P_4 = 4 * (1-2) = -4$

Strassen's Matrix Multiplication

$$\begin{vmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{vmatrix} = \begin{vmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{vmatrix} \begin{vmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{vmatrix}$$

$$A = \begin{vmatrix} 1 & 2 \\ 3 & 4 \end{vmatrix} \quad B = \begin{vmatrix} 2 & 3 \\ 1 & 4 \end{vmatrix}$$

$A_{11} = 1$, $A_{12} = 2$, $A_{21} = 3$, $A_{22} = 4$

$B_{11} = 2$, $B_{12} = 3$, $B_{21} = 1$, $B_{22} = 4$

$P_1 = (A_{11} + A_{22})(B_{11} + B_{22})$

$P_2 = (A_{21} + A_{22}) * B_{11}$

$P_3 = A_{11} * (B_{12} - B_{22})$

$P_4 = A_{22} * (B_{21} - B_{11})$

$P_5 = (A_{11} + A_{12}) * B_{22}$

$P_6 = (A_{21} - A_{11}) * (B_{11} + B_{12})$

$P_7 = (A_{12} - A_{22}) * (B_{21} + B_{22})$

$C_{11} = P_1 + P_4 - P_5 + P_7$

$C_{12} = P_3 + P_5$

$C_{21} = P_2 + P_4$

$C_{22} = P_1 + P_3 - P_2 + P_6$

$P_5 = (1+2) * 4 = 12$

$P_6 = (3-1) * (2+3) = 10$

$P_7 = (2-4) * (1+4) = -10$

Strassen's Matrix Multiplication

$$\begin{vmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{vmatrix} = \begin{vmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{vmatrix} \begin{vmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{vmatrix}$$

$P_1 = (A_{11}+A_{22})(B_{11}+B_{22})$

$P_2 = (A_{21}+A_{22}) * B_{11}$

$P_3 = A_{11} * (B_{12}-B_{22})$

$P_4 = A_{22} * (B_{21}-B_{11})$

$P_5 = (A_{11}+A_{12}) * B_{22}$

$P_6 = (A_{21}-A_{11}) * (B_{11}+B_{12})$

$P_7 = (A_{12}-A_{22}) * (B_{21}+B_{22})$

$C_{11} = P_1 + P_4 - P_5 + P_7$

$C_{12} = P_3 + P_5$

$C_{21} = P_2 + P_4$

$C_{22} = P_1 + P_3 - P_2 + P_6$

$A = \begin{vmatrix} 1 & 2 \\ 3 & 4 \end{vmatrix}$ $B = \begin{vmatrix} 2 & 3 \\ 1 & 4 \end{vmatrix}$ $C = \begin{vmatrix} 4 & 11 \\ 10 & 25 \end{vmatrix}$

$P_1 = 30$ $P_4 = -4$

$P_2 = 14$ $P_5 = 12$ $P_7 = -10$

$P_3 = -1$ $P_6 = 10$

$C_{11} = 30 - 4 - 12 - 10 \Rightarrow (4)$ ✓

$C_{12} = -1 + 12 = (11)$ ✓

$C_{21} = 14 - 4 = (10)$ ✓

$C_{22} = 30 - 1 - 14 + 10$

$= 40 - 15 = (25)$ ✓

Searching

- Searching is the process of finding some particular element in the list.
- If the element is present in the list, then the process is called successful, and the process returns the location of that element. Otherwise, the search is called unsuccessful.
- Two types:
  - Linear Search
  - Binary Search

⇒ for n elements

$$T(n) = O(n)$$

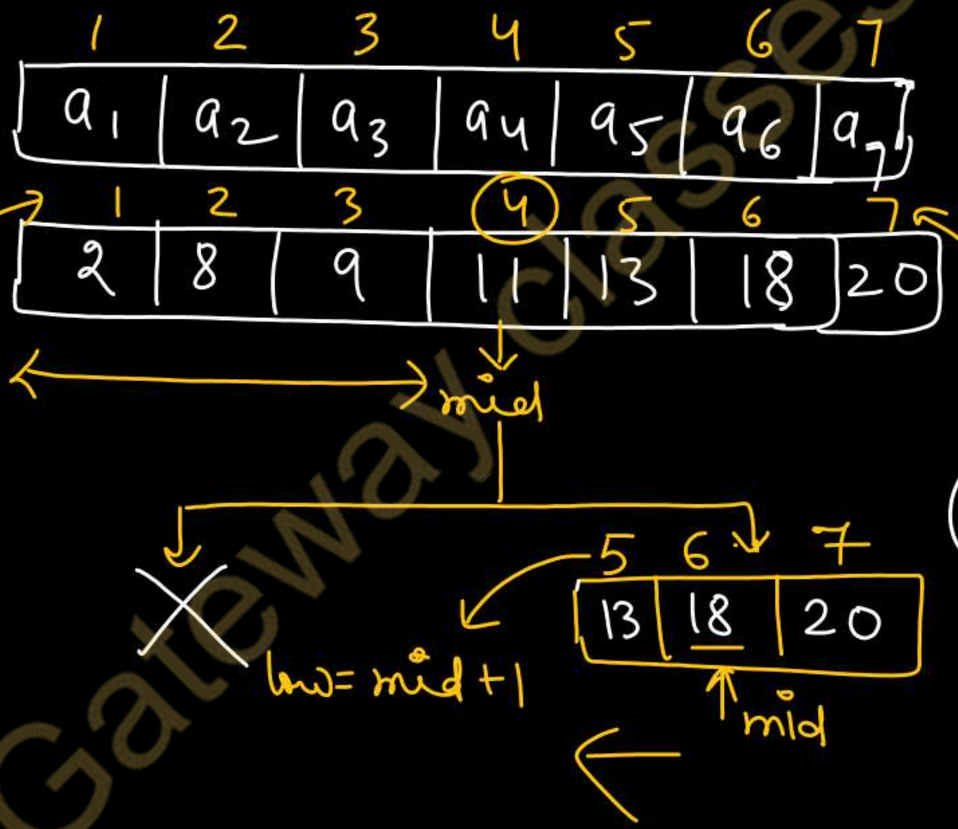
Binary Search

- Binary search can be implemented on sorted array elements. If the list elements are not arranged in a sorted manner, we have first to sort them.

given that

~~$a_1 < a_2 < a_3 < a_4 < a_5 < a_6 < a_7$~~

$x = 13$



$mid = \left\lfloor \frac{low + high}{2} \right\rfloor$

$mid = \left\lfloor \frac{1+7}{2} \right\rfloor = 4$

(only one subproblem)

$mid = \left\lfloor \frac{5+7}{2} \right\rfloor = 6$

Binary Search Example

Case-1 $x = 26$ ✓ ← To be searched

1	2	3	4	5	6	7	8	9	10	11	12
3	8	10	12	14	19	22	26	32	38	45	54

low, high, mid, mid

if $x == mid \Rightarrow$ found

if $x < mid \Rightarrow high = mid - 1$

if $x > mid \Rightarrow low = mid + 1$

low	high	mid =
1	12	6
$6+1 = 7$	12	9
7	$9-1 = 8$	7
8	8	8
		found

$\lfloor \frac{1+12}{2} \rfloor = 6$

$\lfloor \frac{7+12}{2} \rfloor = 9$

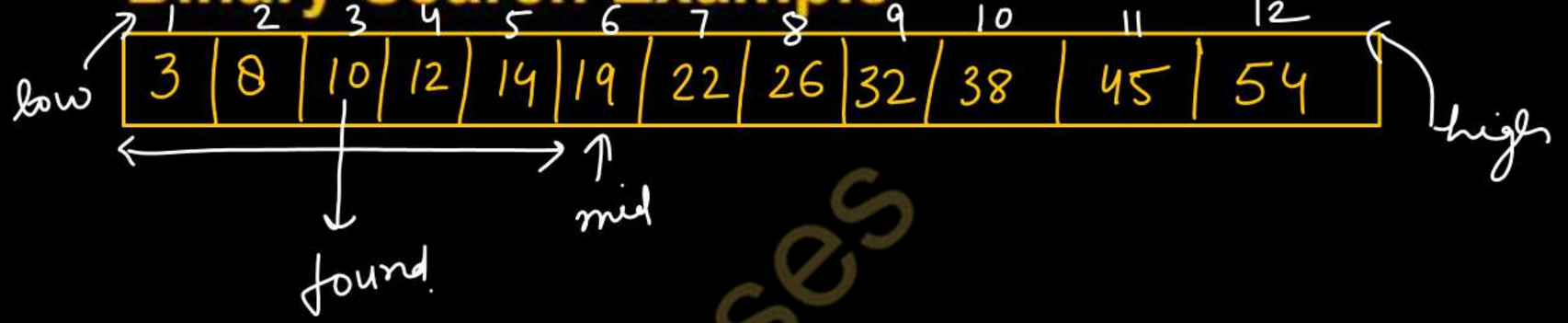
$\lfloor \frac{7+8}{2} \rfloor = 7$

$\lfloor \frac{8+8}{2} \rfloor = 8$

Binary Search Example

Case-2

$x = 10$



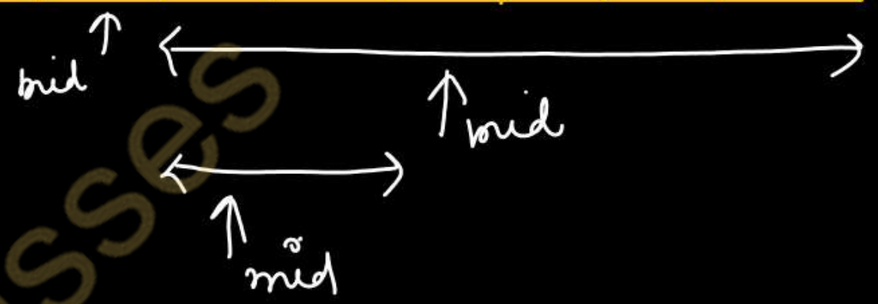
Low	High	Mid		
1	12	6	$\lfloor \frac{1+12}{2} \rfloor = 6$	$a[6] > x$
1	5	3	$\lfloor \frac{1+5}{2} \rfloor = 3$	$19 > 10$
		found		

Binary Search Example

Case-3

$x = 21$

1	2	3	4	5	6	7	8	9	10	11	12
3	8	10	12	14	19	22	26	32	38	45	54



low	high	mid
1	12	6
7	12	9
7	8	7
7	6	
low	high	

$\lfloor \frac{7+12}{2} \rfloor = 9$

$\lfloor \frac{7+8}{2} \rfloor = 7$

$\Longrightarrow$ Element (21) does not exist.

low > high

Recursive Binary Search

```
Algorithm binarySearchDC(A, x, low, high)
if (low > high)
        return -1
mid = (low + high) / 2
if A[mid] == x
        return mid
else if A[mid] > x
        return binarySearchDC(A, x, low, mid-1)
else if A[mid] < x
        return binarySearchDC(A, x, mid+1, high)
End
```

x : element to be searched

Recursive Binary Search Time Complexity

```
Algorithm binarySearchDC(A, x, low, high)
if (low > high)
        return -1
mid = (low + high) / 2
if A[mid] == x
        return mid
else if A[mid] > x
        return binarySearchDC(A, x, low, mid-1)
else if A[mid] < x
        return binarySearchDC(A, x, mid+1, high)
End
```

$T(n)$

$T(n/2)$

$T(n/2)$

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

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

$a = 1$, $b = 2$, $k = 0$, $p = 0$

$a = b^k$

$1 = 2^0$ case-2(ii)

$n^k \log^p n$

$$T(n) = O(n^{\log_b a} \log^{p+1} n)$$

$$= O(n^{\log_2 1} \log^{0+1} n)$$

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

Recursive Binary Search Illustration

x = 15

main method

	1	2	3	4	5	6	7	8
A	3	5	8	10	13	15	18	23

binarySearchDC(A, 15, 1, 8)

$mid = \lfloor \frac{1+8}{2} \rfloor = 4$

a[4] = 10

15 > 10 (go to right side of array)

low = mid + 1

6

binarySearchDC(A, 15, 5, 8)

$mid = \lfloor \frac{5+8}{2} \rfloor = 6$

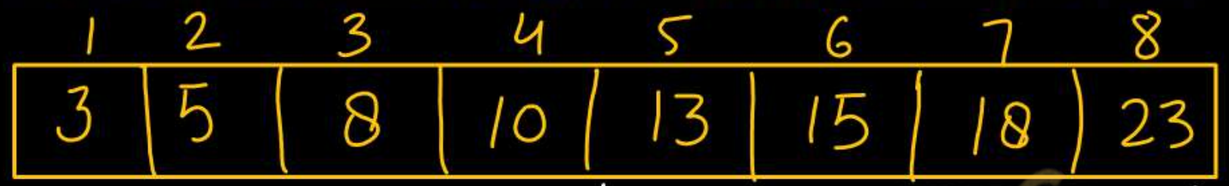
a[6] = 15

15 = 15 ✓ element found

6

Recursive Binary Search Illustration

$x = 17$



main calling function

-1

binarySearchDC(A, 17, 1, 8)

$mid = \lfloor \frac{1+8}{2} \rfloor = 4$

-1

binarySearchDC(A, 17, 5, 8)

$mid = 6$

-1

binarySearchDC(A, 17, 7, 8)

$mid = 7$

-1

binarySearchDC(A, 17, 7, 6)

17 doesn't exist in the array.

$17 > 10 \Rightarrow$ search into right side subarray

low = mid + 1

$15 < 17 \Rightarrow$ search in right side subarray

low = mid + 1

$18 > 17 \Rightarrow$ search in left subarray

high = mid - 1

AKTU

B.Tech 5th Sem

GW GATEWAY CLASSES

CS IT & CS Allied

DAA : Design & Analysis Of Algorithm

Unit-3 : Lecture-7

Today's Target

Divide and Conquer

- Convex Hull
- AKTU PYQs

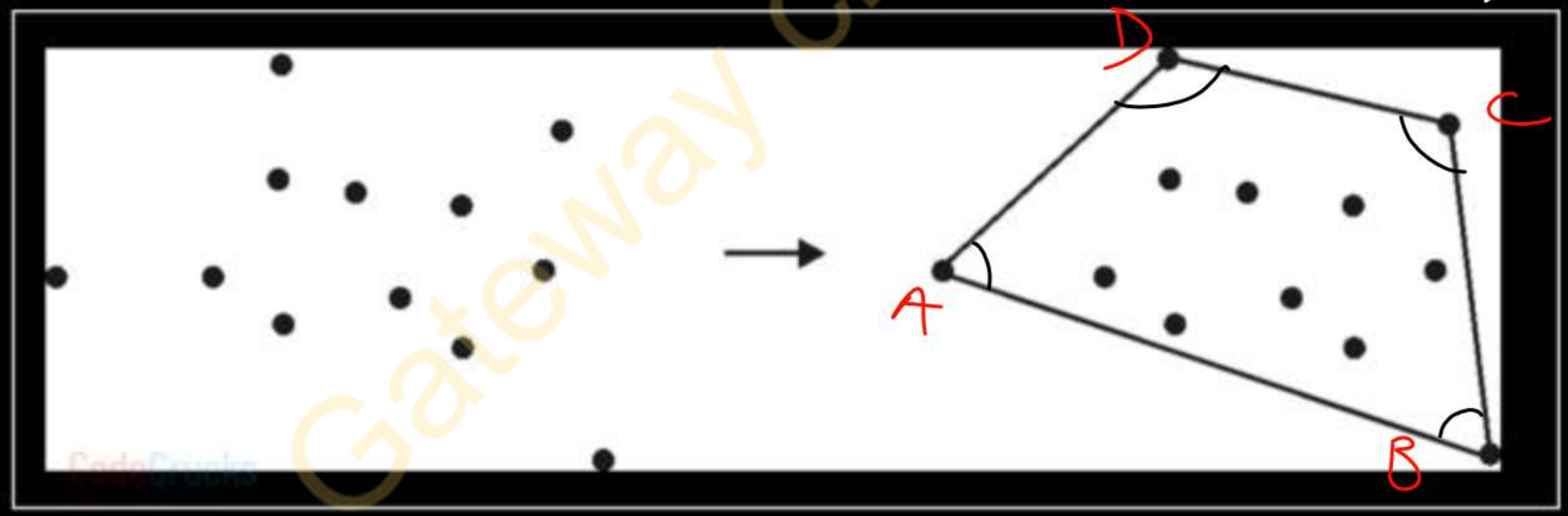
By Dr. Nidhi Parashar Ma'am

- M.Tech Gold Medalist
- Net Qualified

Convex Hull

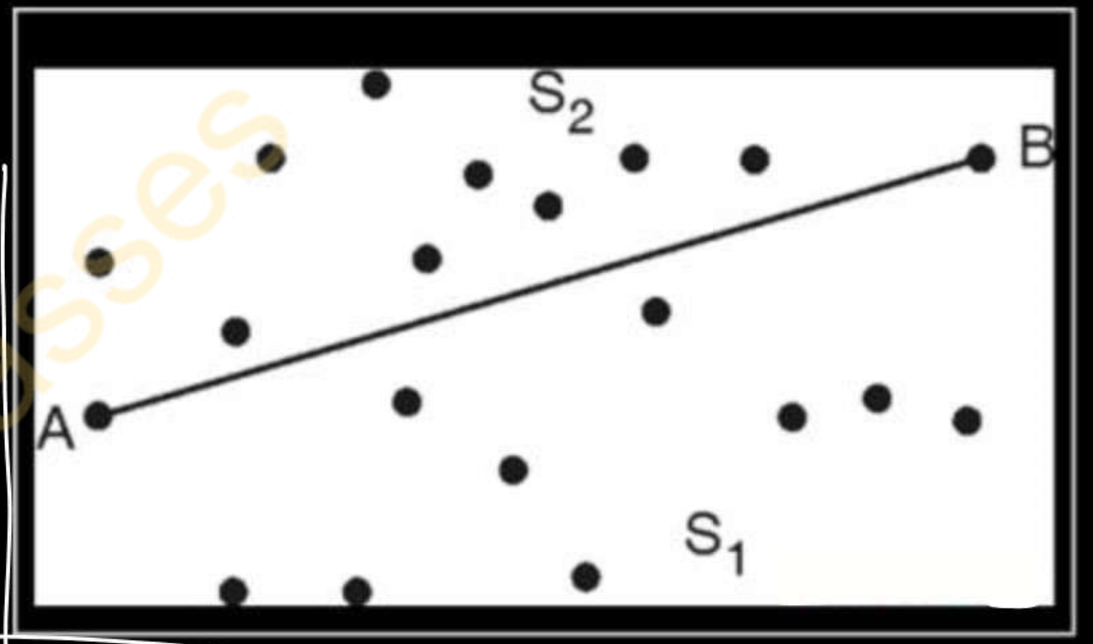
- The **convex hull** of the set of points Q is the convex polygon P that encompasses all of the points given.
- The problem of finding the smallest polygon P such that all the points of set Q are either on the boundary of P or inside P is known as the convex hull problem.

$S = \{AB, BC, CD, DA\}$



Divide and Conquer Approach to solve convex hull problem

Step 1: Sort all points by their X coordinates. Determine leftmost point A and rightmost point B. Add lines AB and BA to solution set.



Example: Label all points on the right of AB as S_1 and all the points on the right of BA as S_2.

Solution = {AB, BA}

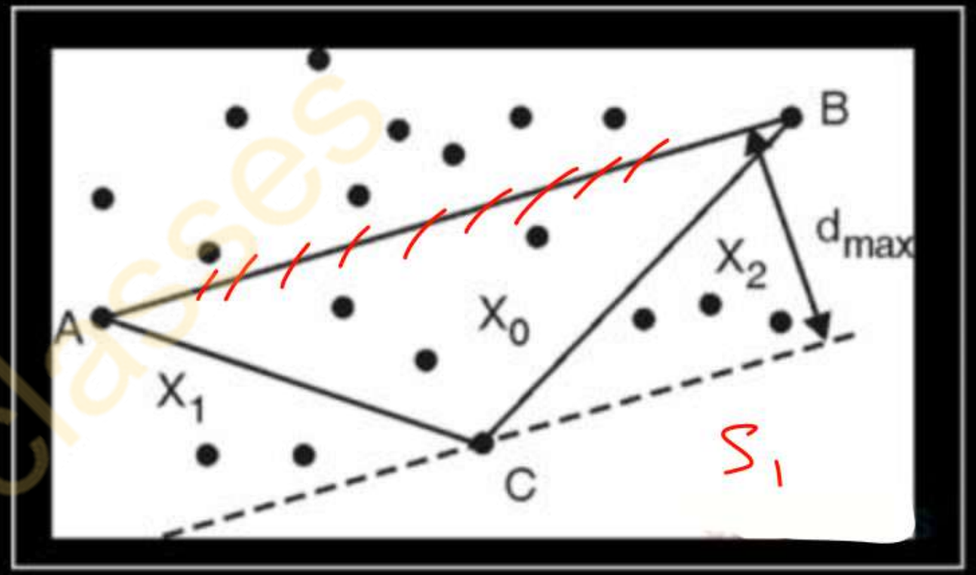
Make a recursive call to **FindHull (S_1, A, B) and FindHull(S_2 ,B, A)**

Divide and Conquer Approach to solve convex hull problem

Step 2:

FindHull(S_1 ,A, B)

Find point C orthogonally farthest from line AB



Solution = [Solution – { AB }] ∪ {AC, CB}

= {AC, CB, BA}

Label regions X_0, X_1 and X_2 as shown in above figure

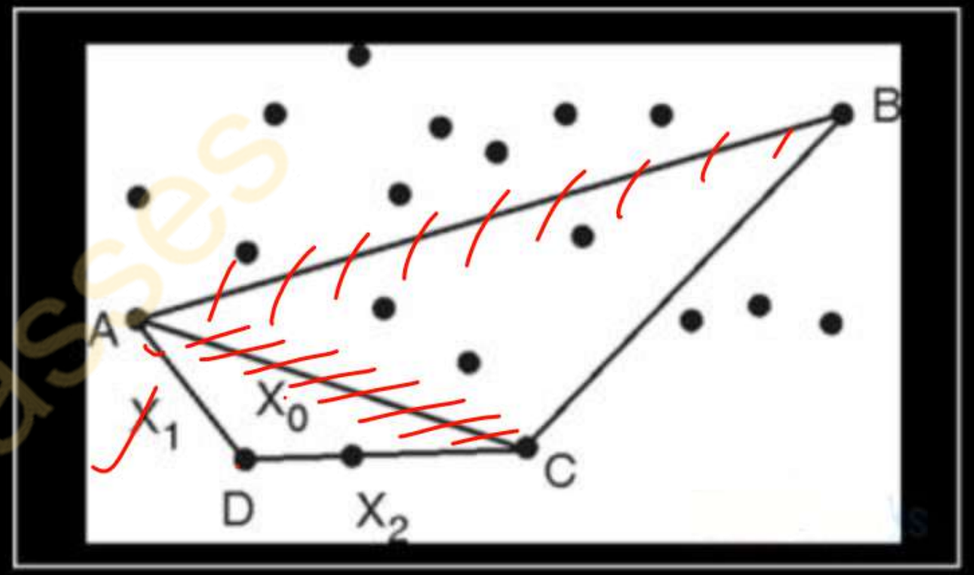
Make recursive calls: **FindHull (X_1, A, C) and FindHull (X_2, C, B)**

Divide and Conquer Approach to solve convex hull problem

Step 3:

FindHull(X_1, A, C)

Find point D orthogonally farthest from line AC



Solution = [Solution – {AC}] ∪ {AD, DC}

= {AD, DC, CB, BA}

Label regions X_0, X_1 and X_2 as shown in above figure

Make recursive calls: **FindHull (X_1, A, D) and FindHull (X_2, D, C)**

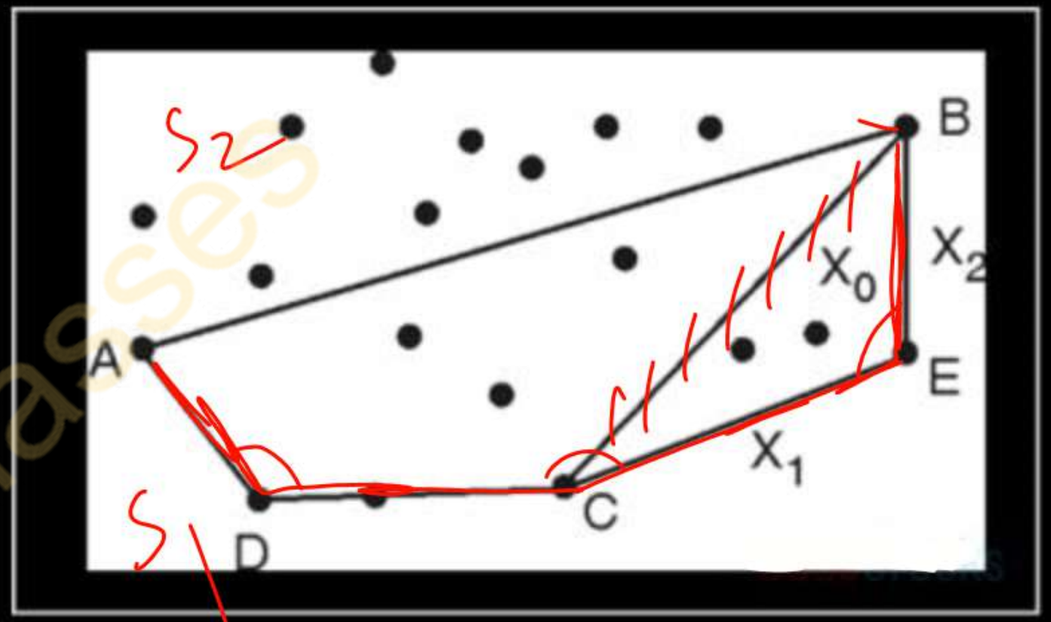
But X_1 and X_2 sets are empty, so algorithm returns

Divide and Conquer Approach to solve convex hull problem

Step 4:

FindHull(X_2, C, B)

Find point E orthogonally farthest from line CB



Solution = Solution – {CB} ∪ {CE, EB}

= {AD, DC, CE, EB, BA}

Label regions X_0, X_1 and X_2

Make recursive calls: FindHull (X_1, C, E) and FindHull (X_2, E, B).

But X_1 and X_2 sets are empty, so algorithm returns

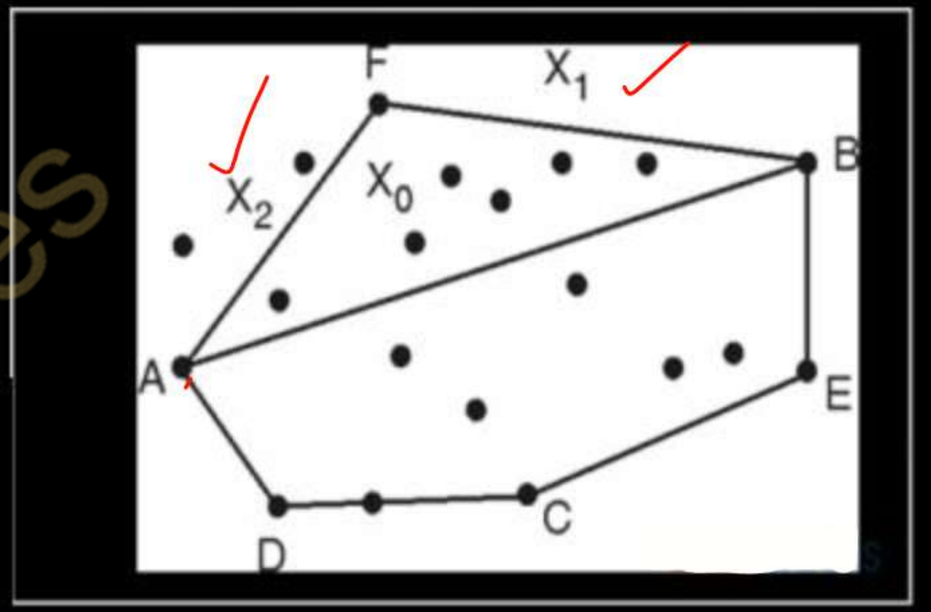
Now we will explore the points in S2, on the right-hand side of the line BA

Divide and Conquer Approach to solve convex hull problem

Step 5:

FindHull(S_2 ,B, A)

Find point F orthogonally farthest from line BA



Solution = Solution – {BA} ∪ {BF, FA}

= {AD, DC, CE, EB, BF, FA}

Label regions X_0, X_1 and X_2

Make recursive calls : FindHull (X_1, B, F) and FindHull (X_2, F, A)

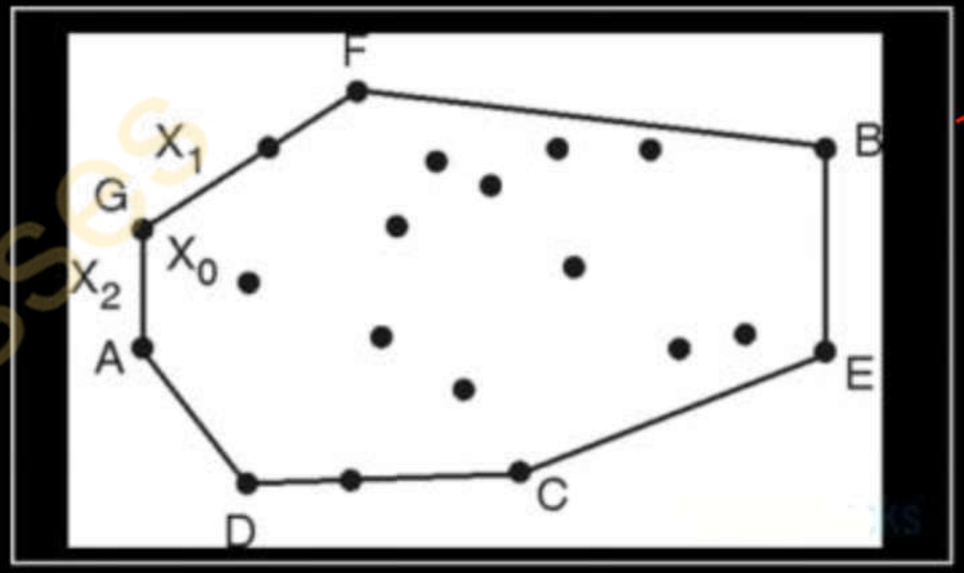
But X_1 set is empty, so call to FindHull (X_1, B, F) returns

Divide and Conquer Approach to solve convex hull problem

Step 6:

FindHull (X_2, F, A)

Find point G orthogonally farthest from line FA



Solution = Solution – {FA} ∪ {FG, GA}

= {AD, DC, CE, EB, BF, FG, GA}

Label regions X_0

Make recursive calls: **FindHull (X_1, F, G) and FindHull (X_2, G, A)**

X_1 and X_2 sets are empty, so algorithm returns.

No more recursive calls are left.

So polygon with edges (AD, DC, CE, EB, BF, FG, GA) is the convex hull of given points.

Time Complexity

T(n)

- Sort the points according to increasing order of their X coordinate: O(nlogn)
- Finding two farthest points from the sorted list: O(1)
- Dividing points into two halves S_1 and S_2 by joining A and B: O(1)
- Recursively computing the convex hull of A and B: T(n/2) each.
- Merging of two convex hulls is done in linear time O(n)

$$T(n) = 2T(n/2) + O(n) + O(1)$$

$$T(n) = O(n.Logn)$$

Grahm Scan Algorithm to solve convex hull problem

Phase I. First, select base point p_0 in set Q of n points in the plane, with min y-coordinate. In case of tie, select leftmost point (min x-coordinate).

Phase II. Next, sort the remaining points by polar angle in counterclockwise order around p_0.

Phase III. Push p_0 onto stack S, and scan through the points in counterclockwise order, maintaining at each step a stack S containing a convex chain surrounding the points scanned so far.

Each time we consider the new point p_i, test:

a. If p_i forms a left turn with last two points in S, or if S contains fewer than two points, then push p_i onto S.

b. Otherwise, pop the last point from the stack S and repeat the test for p_i.

We halt when we return to the anchor point p_0, at which point stack S stores the vertices of the convex hull of Q in counterclockwise order.

GRAHAM_SCAN(Q) (Important)

1. Find p_0 in Q
2. Sort Q − $\{p_0\}$) by polar angle in counterclockwise order with respect to p_0.
3. TOP [S] = 0 ▷ Lines 3-6 initialize the stack to contain first three points.
4. PUSH (p_0, S)
5. PUSH (p_1, S)
6. PUSH (p_2, S)
7. **for** i = 3 **to** n ▷ Perform test for each point p_3, …, p_n.
8. **do while** angle between NEXT_TO_TOP[S], TOP[S], and p_i makes a nonleft turn
9. **do** POP(S)
10. PUSH (S, p_i)
11. **return** S

```
GRAHAM_SCAN(Q)
1.  Find p_0 in Q
2.  Sort Q − {p_0}) by polar angle in counterclockwise order with respect to p_0.
3.  TOP [S] = 0          ▷ Lines 3-6 initialize the stack to contain first three points.
4.  PUSH (p_0, S)
5.  PUSH (p_1, S)
6.  PUSH (p_2, S)
7.  for i = 3 to n       ▷ Perform test for each point p_3, ..., p_n.
8.      do while angle between NEXT_TO_TOP[S], TOP[S], and p_i makes a nonleft turn
9.          do POP(S)
10.     PUSH (S, p_i)
11. return S
```

$O(1)$

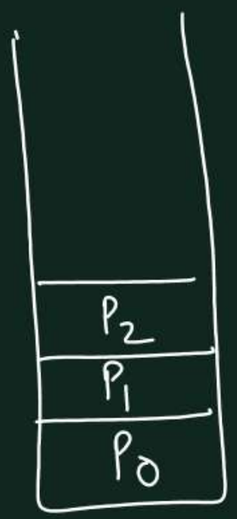
$O(n)$

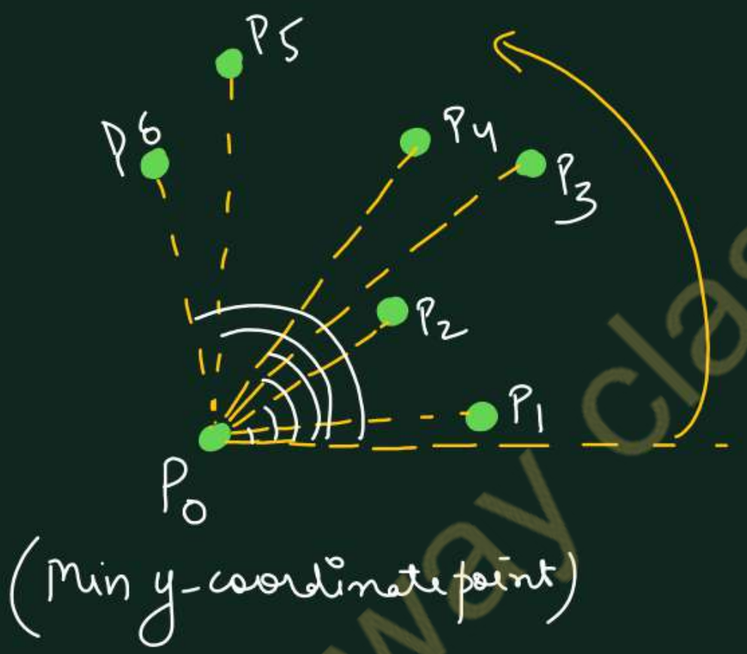
$O(n)$

$O(n \log n)$

$T(n) = O(n) + O(n \log n) + O(n) + O(1)$

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

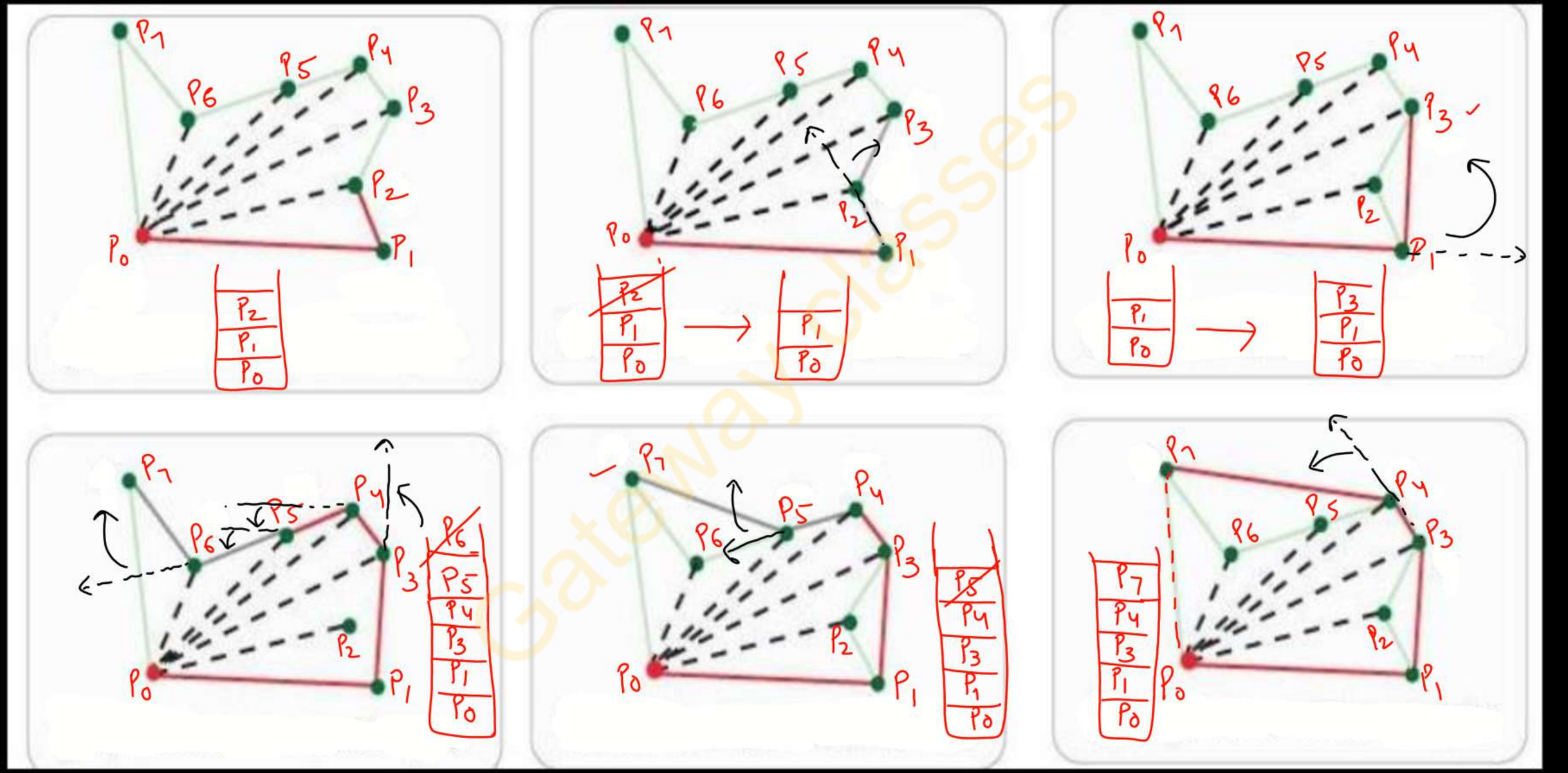




$S \Rightarrow$ set of points

$P_0 \Rightarrow$ Anchor point

NEXT_TO_TOP[S] referred as p, TOP[S] referred as c , p_i referred as



Applications of Convex Hull

1. Collision avoidance: Calculating collision-free routes is considerably easier with a convex hull of vehicle.

2. Smallest Box: Convex hull helps to determine the minimum size required to put object in in.

3. Shape Analysis: Convex hull of object is useful to analyze the shape of object.

AKTU PYQs

1. What do you mean by convex hull? Write the algorithm that solves convex hull problem. Find complexity of the algorithm. (AKTU 2019-20)