

A MST can be grown from a forest of spanning trees by adding the smallest edge Subject: connecting two spanning trees.
Date:

(1) Kruskal Algorithm

Kruskal algorithm is a Greedy algorithm used to find the MST of a connected, undirected, weighted graph.

work:

→ sorting edge by weight

→ picking the smallest possible edge that does not form a cycle.

→ repeating until $n-1$ edges are chosen.

working:

i) sort all the edge of the graph in non-decreasing order of their weight.

ii) pick the smallest edge,

→ if it does not form a cycle, ~~include~~ add the edge in the MST

→ if ~~it not~~ it form a cycle, discard the edge.

iii) repeat step-2 until the spanning tree containing $(n-1)$ edges.

Pseudocode:

~~KRUSKAL(G, w):~~

~~1. $A = \text{empty set}$~~

~~2. For each vertex $v \in G.V$~~

~~3. MAKE-SET(v)~~

~~4. sort all edge of $G.E$ into nondecreasing order by weight(w)~~

~~5.~~

KRUSKAL(G, w):

1. $A = \text{empty set}$

2. sort all edge E of G in non-decreasing

3. order of weight

4. for each edge (u, v) in E (in sorted order):

5. if adding (u, v) does NOT form a cycle in mst:

6. add (u, v) to mst

7. else :

8. discard (u, v)

9. if $\text{size}(A) = v - 1$

10. break

11. return A

1. $A = \text{empty set}$

2. for each vertex $v \in G, v$

3. make-set (v)

4. sort the edges of G, E into non-decreasing order

5. for each edge $(u, v) \in G, E$

6. if find-set $(u) \neq \text{find-set}(v)$

7. $A = A \cup \{(u, v)\}$

8. UNION (u, v)

//DSU function.

9. make-set (x) :

parent $[u] = u$

rank $[u] = 0$

Subject:.....

Date:.....

FIND(u):

if $\text{parent}[u] = u$

$\text{parent}[u] = \text{FIND}(\text{parent}[u])$

return $\text{parent}[u]$

UNION(u, v):

$r_1 = \text{FIND}(u)$

$r_2 = \text{FIND}(v)$

if $r_1 = r_2$

return

if $\text{rank}[r_1] < \text{rank}[r_2]$:

$\text{parent}[r_1] = r_2$

else if $\text{rank}[r_1] > \text{rank}[r_2]$:

$\text{parent}[r_2] = r_1$

else

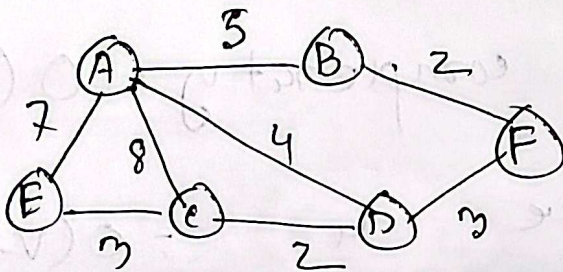
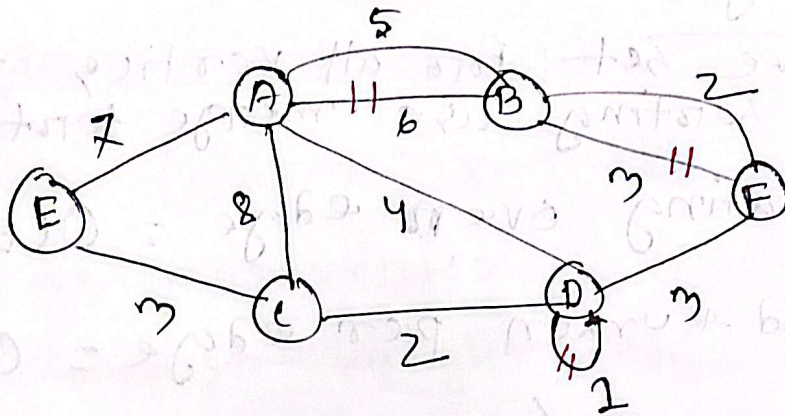
$\text{parent}[r_2] = r_1$

$\text{rank}[r_2] = \text{rank}[r_1] + 1$

return r_1

Subject:

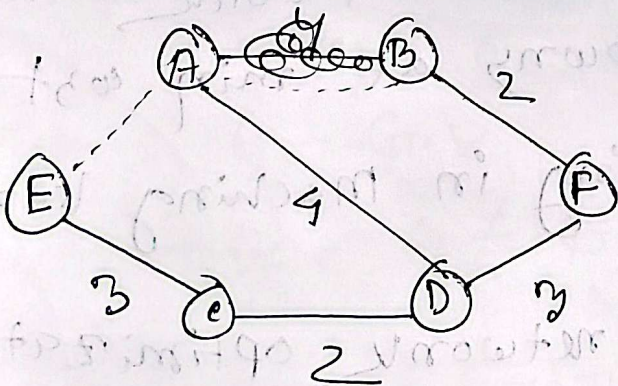
Date:



→ remove cycle and loop

→ remove max value edge in cycle.

→ sorting all edges into nondecreasing order.



$$\checkmark BF = 2$$

$$\checkmark CD = 2$$

$$\checkmark DF = 3$$

$$\checkmark CE = 3$$

$$\checkmark AD = 4$$

$$\times AB = 5$$

$$\times AC = 7$$

$$\checkmark AC = 8$$

remove for cycle

this is MST according to Kruskal Algorithm.

Here, $v = v'$ and

Complexity :

- make set for all vertices $= O(V)$
- For sorting use merge sort $= O(E \log E)$
- iterating over edge $= O(E)$
- Find + union per edge $= O(E \alpha(V))$
- MST add/break $= O(E)$

$\therefore$ total time complexity $= O(E \log E)$

$\therefore$ space $= O(V + E)$

Application:

- ① Telecommunication network — connecting cities or towns at min cost
- ② Data clustering in machine learning
- ③ Routine & network optimization
- ④ power grid design.

A mst can ~~form the~~ be grown from the current spanning tree by adding the edge connecting the nearest ~~vertex~~ ^{nearest vertex of the} vertex to the mst

Prim's Algorithm

~~same definition (Kruskal)~~

→ starts from any vertex

→ Grows mst vertex by vertex, always adding the smallest weight edge connecting mst to a new vertex.

Pseudocode:

PRIMS (G, w)

$\emptyset = \text{set of } G.V$ ~~for each~~ $v \in \emptyset$
for each $u \in G.V$

$u.key = \infty$

$u.\pi = \text{NIL}$

$\pi.key = 0$

$\emptyset = G.V$

while $\emptyset \neq \emptyset$

$u = \text{Extract_min}(\emptyset)$

for each $v \in G.V \text{ Adj } u$

if $v \in \emptyset$ and $w(u, v) < v.key$

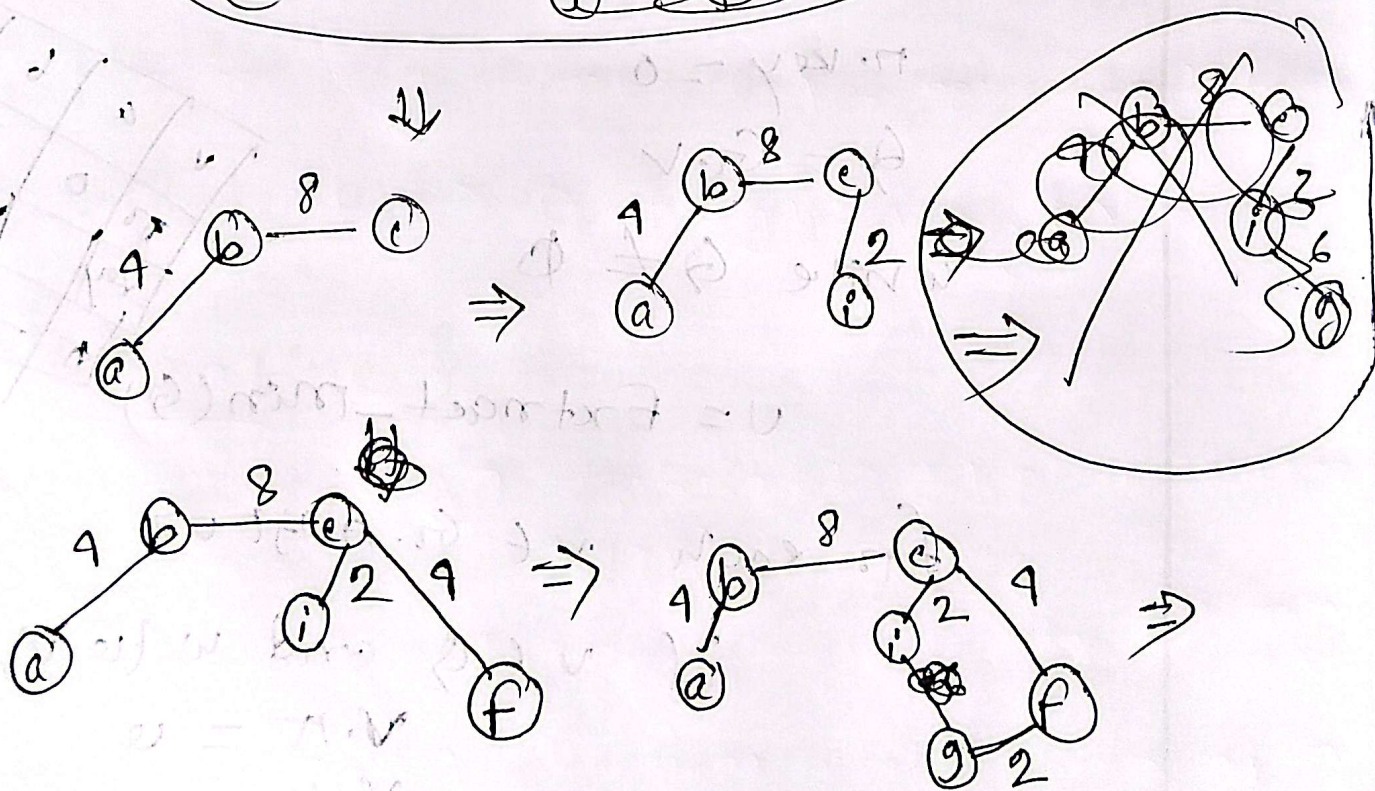
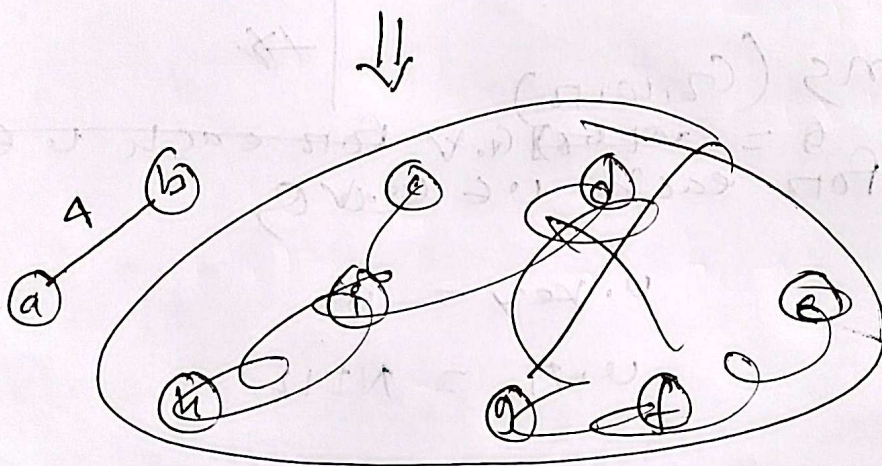
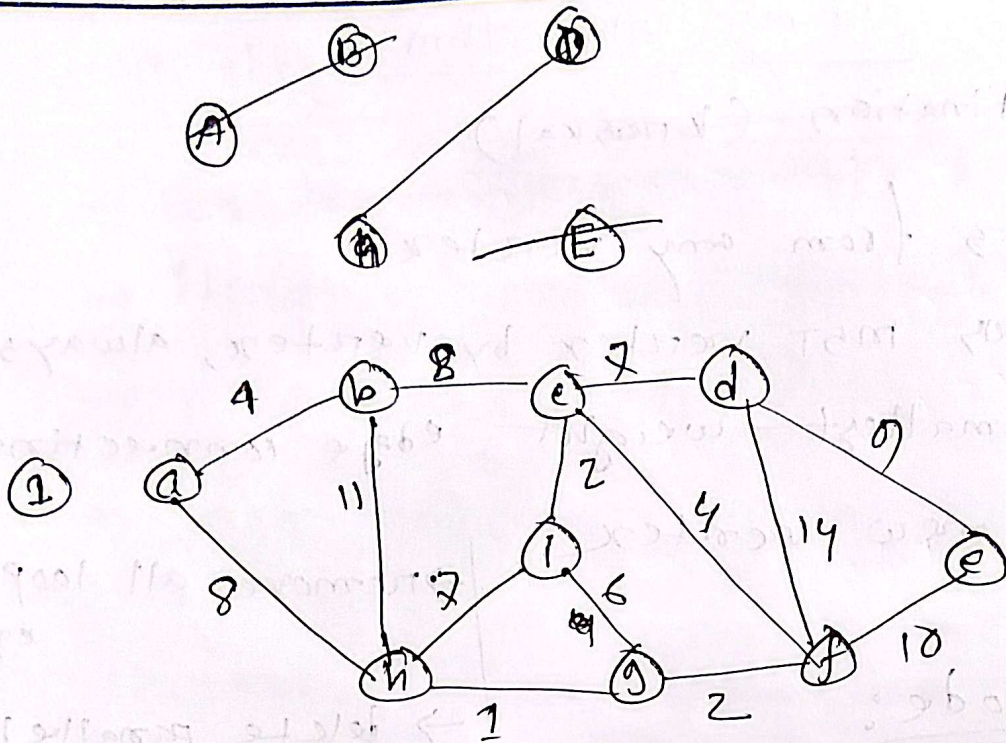
$v.\pi = u$

$v.key = w(u, v)$

	a	b	c	d	e
v					
u	0	0	0	0	0
key					
π					

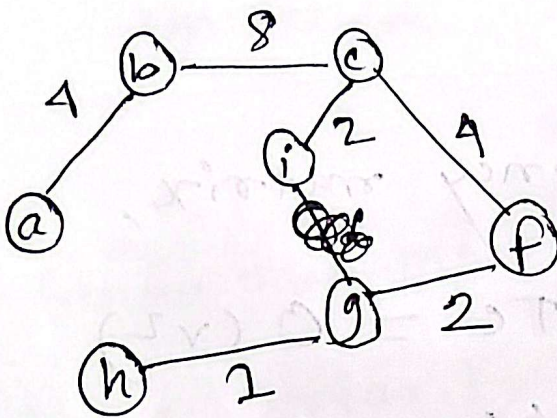
Subject:.....

Date:.....

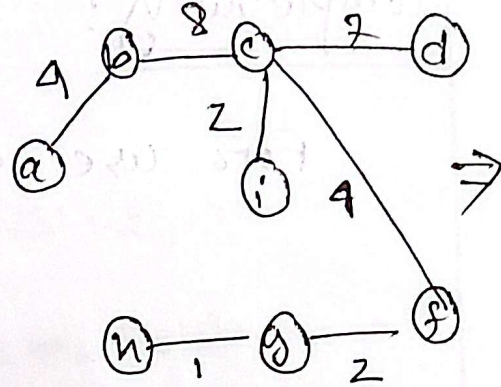


Subject:.....

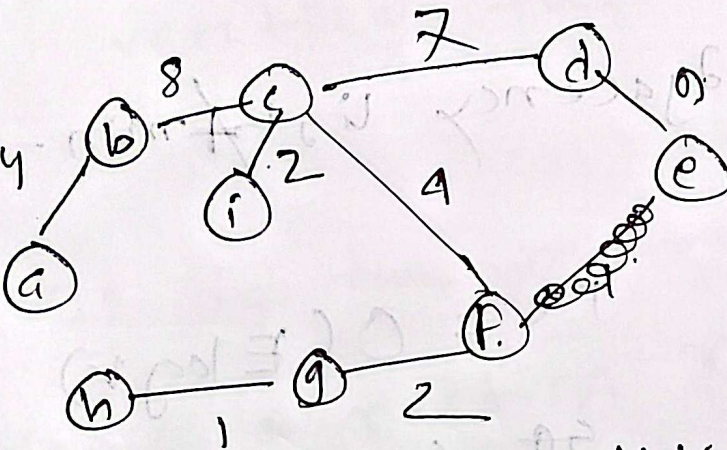
Date:.....



$\Rightarrow$



$\Rightarrow$



this is mst according to Prim's algorithm.

total vertices $V' = V = 9$
 edge $E' = |V| - 1$
 $= 9 - 1 = 8$

$\therefore$ primy done.

complexity:

For use adjacency matrix,

$$TC = O(V^2)$$

$$SC = O(V^2)$$

For use adjacency list + min-heap/priority queue.

$$TC = O(E \log V)$$

$$SC = O(V + E)$$

Each extract-min operation: $O(\log V)$

decrease-key: $O(\log V)$

loop run V times, edges processed E

time $\rightarrow O(E \log V + V \log V) \approx O(E \log V)$

Application:

① Network design - Internet, telephone, power grids

② Railway network - connect city with min cost

③ clustering - Hierarchical clustering in machine learning

Subject:.....

Date:.....

Prims algorithm variable:

Q:

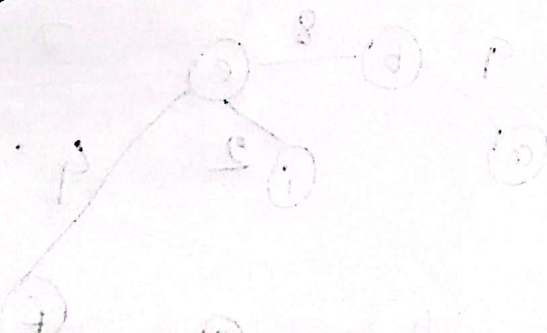
r: Grow the mst from the root vertex r

Q: is a priority queue, holding all vertices that are not in the tree now.

key[v]: is the minimum weight of any edge connecting v to a vertex in the tree

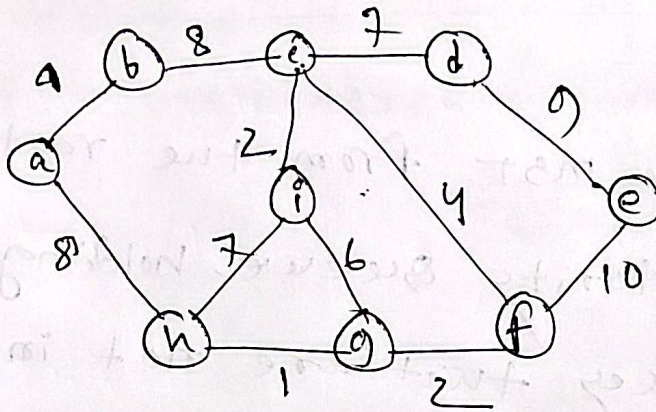
$\pi[v]$: name the parent in the tree.

$T[v]$: vertex v is already included in mst, if $T[v] = 1$, otherwise it is not included.



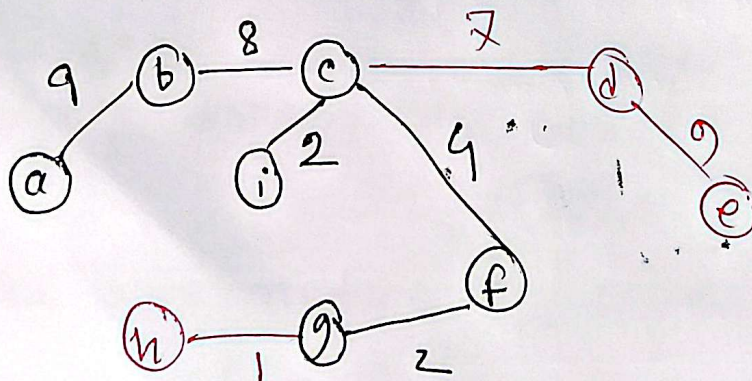
Subject:.....

Date:.....



Step:

v	a	b	c	d	e	f	g	h	i
T	1	0 ₁	0 ₁	0 ₁	0 ₁	0 ₁	0 ₁	0 ₁	0 ₁
Key	0	4	8	7	10 ₉	4	2	8 ₁	2
π	-1	a	b	c	f _d	c	f	a _g	c



Subject:.....

Date:.....

Kruskal algorithm

→ Edge based algorithm

→ add the edge one at a time, in increasing weight order

→ The algorithm maintains A - a forest of trees. An edge is accepted if it connects vertices of distinct trees.

→ we need a data structure that maintain a partition, i.e., a collection of disjoint sets.

— makeSet(s, x)

— union(s_i, s_j)

— FindSet(s, x)

Subject:.....

Date:.....

Disjoint set

→ keep a collection of sets $s_1, s_2, \dots, s_k$

— Each s_i is a set e.g. $s_1 = \{v_1, v_2, v_3\}$

→ Three option

→ $\text{make-set}(x)$ — creates a new set

whose only member is x

→ $\text{union}(x, y)$ — unites the sets

that contain x and y , say s_x

and s_y into a new set that is

the union of the two sets.

→ $\text{Find-set}(x)$ — returns a pointer
to the representative of the
set containing x .

Subject:

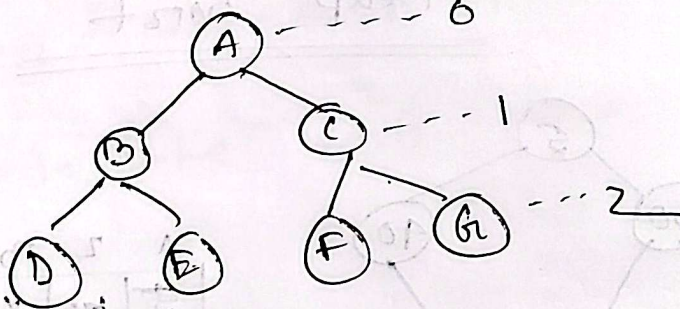
Date:

Heap Sort

Heap sort is a comparison-based sorting algorithm that uses a binary heap data structure to sort elements.

→ complete binary tree.

Array representation of binary tree:



0	1	2	3	4	5	6
A	B	C	D	E	F	G

index $\rightarrow i$

left child $\rightarrow (2i) + 1$

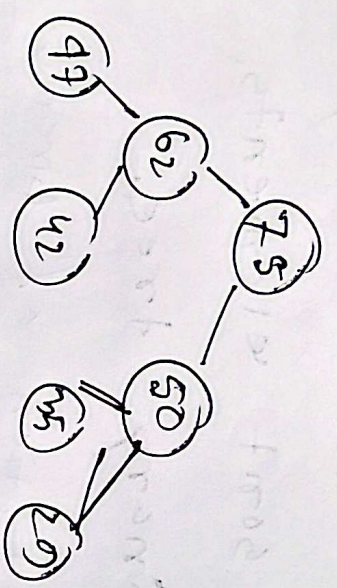
right child $\rightarrow (2i) + 2$

parent child $\rightarrow \lfloor (i-1)/2 \rfloor$

0	1	2	3	4	5	6
A	B	C	D	E	F	G

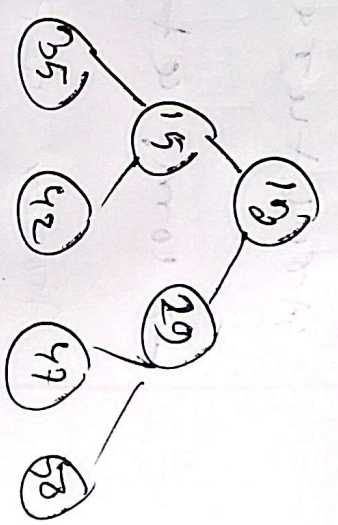
max heap

parent > child

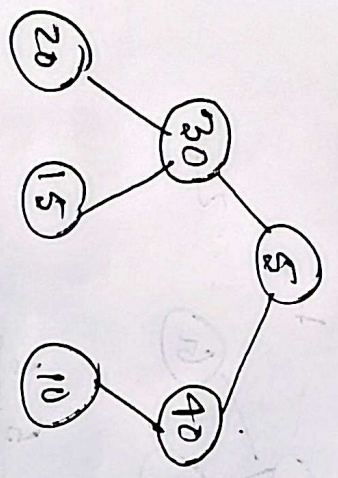


min heap

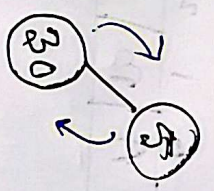
child > parent



Heap sort



1	2	3	4	5	6
5	30	40	20	15	10

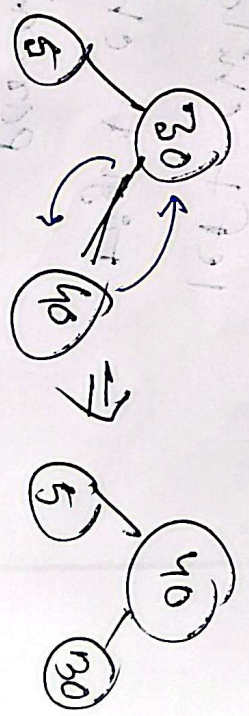


5 < 30

swap(30, 5)

1	2	3	4	5	6
30	5	40	20	15	10

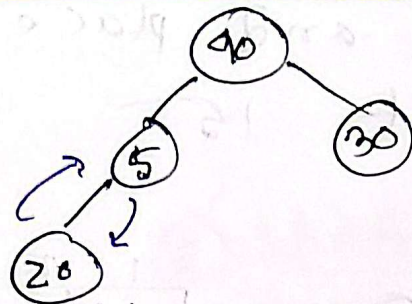
1st pass
 5 < (max) ← 51st
 5 < (max) ← 51st



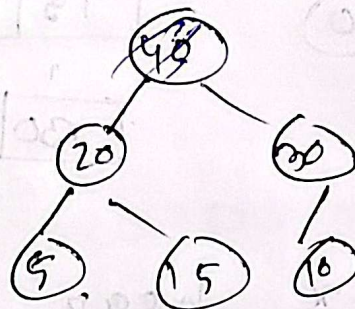
1	2	3	4	5	6
40	5	30	20	15	10

Subject:.....

Date:.....



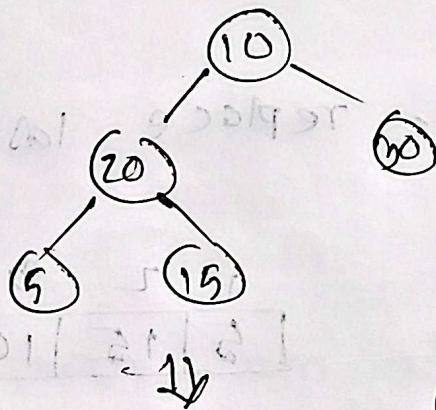
1	2	3	4	5	6
40	20	30	5	15	10



1	2	3	4	5	6
40	20	30	5	15	10

→ Final. Heap made.

Now apply deletion for sorting array.
40 → delete and place to 10 (last element)

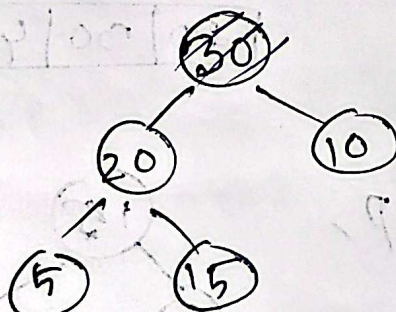


Now it is not max heap property.

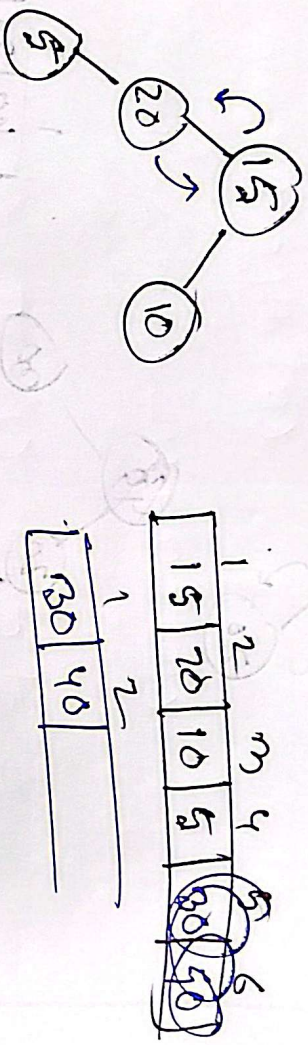
1	2	3	4	5	6
10	20	30	5	15	40

Queue [40]

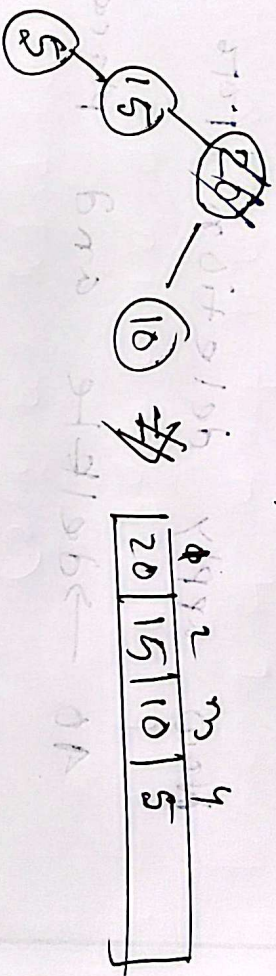
Now it is max heap.



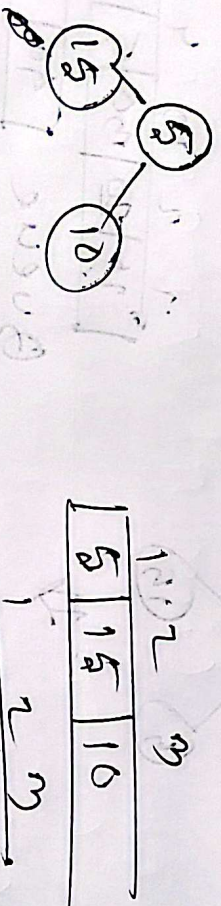
Now delete 20 and place as parent the
last element 15



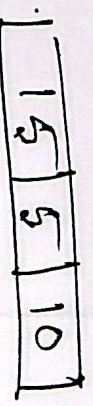
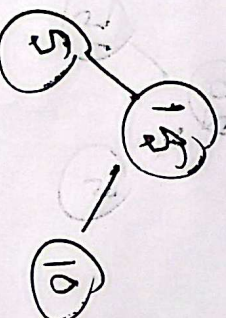
Now, make max heap



Now, delete 20, replace last element 5



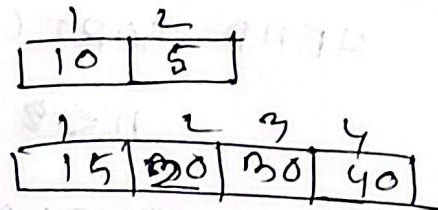
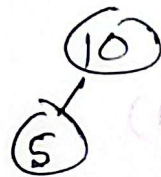
make max heap,



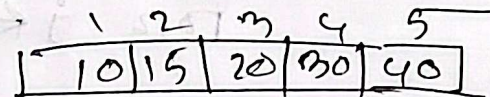
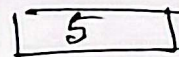
Subject:

Date:

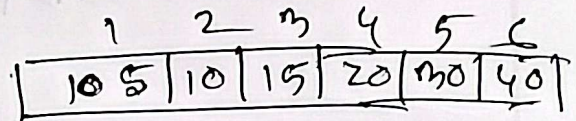
delete - 15



delete - 10



now delete -> 5



sorted array.

TC:

iteration heap total time $O(n \log n)$

deletion " " " $O(n \log n)$

$\therefore$ total TC = $O(n \log n)$

SK: do not need extra memory because it is in-place sorting algorithm

$O(1)$

all operation (swaps, heapify) are performed inside the same input array.

Subject:.....

Date:.....

Pseudocode:

HEAD-SORT(A):

$n \leftarrow \text{length}(A)$

BUILD-MAX-HEAP(A)

for $i \leftarrow (n/2 - 1)$ down to 0

MAX-HEAPIFY(A, i)

for $j \leftarrow n-1$ down to 1

swap $A[0]$ and $A[j]$

MAX-HEAPIFY(A, i, 0)

~~HEAD~~

MAX-HEAPIFY(A, n, i)

~~largest~~ $l \leftarrow \text{left}(i)$

$r \leftarrow \text{right}(i)$

if $l \leq n$ and $A[l] > A[i]$

largest = l

else

largest = i

Subject:
Date:

if $r \leq n$ and $A[r] > A[\text{largest}]$
 $\text{largest} = r$

if $\text{largest} \neq i$

~~exchange~~ swap $A[i]$ and $A[\text{largest}]$

max-heapify ($A, n, \text{largest}$)

Note:

heap creation need $O(n \log n)$ time

But when we use heapify method then
need only $O(n)$ which is faster.

Application

- sorting large dataset
- priority queue implementation
- selection algorithm (Finding the largest & smallest k element)
- Graph algorithm (prim's, Dijkstra's algorithm)
- Online streaming data
- Real-time system - deterministic behavior.

Subject:.....

Date:.....

Bellman Ford - algorithm

Bellman Ford is a single source shortest path algorithm that works even if the graph has negative weight edges.

→ It can also detect negative weight cycle.

→ Dijkstra cannot handle negative edge, but Bellman Ford can it.

work:

- Find shortest distance from a source node to all other nodes.

- repeatedly relaxes all edges.

- If after $V-1$ relaxations any edge can still be relaxed → there is a negative cycle.

Subject:

Date:

Relaxation:

$$if (d(u) + e(u, v) < d(v))$$

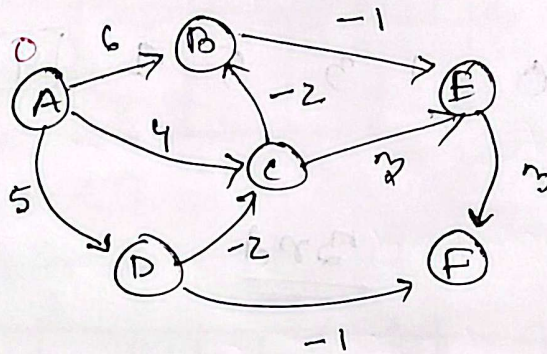
$$d(v) = d(u) + e(u, v)$$

iteration

$(n-1)$

↓
no. of vertices.

simulation:



total iteration need = $6 - 1 = 5$

selected
vertices
↓

1st iteration

	A	B	C	D	E	F
A	0	∞	∞	∞	∞	∞
B	0	6	4	5	∞	∞
C	0	6	4	5	5	∞
D	0	6	4	5	5	∞
E	0	2	3	5	5	4
F	6	2	3	5	5	4

Subject:.....

Date :.....

2nd

	A	B	C	D	E	F
A	0	2	3	5	5	4
B	0	2	3	5	5	4
C	0	2	3	5	1	4
D	0	1	3	5	1	4
E	0	1	3	5	1	4
F	0	1	3	5	1	4

3rd

	A	B	C	D	E	F
A	0	1	3	5	1	4
B	0	1	3	5	1	4
C	0	1	3	5	0	4
D	0	1	3	5	0	4
E	0	1	3	5	0	4
F	0	1	3	5	0	3

~~3rd~~

Subject:.....

Date:.....

4th

	A	B	C	D	E	F
A	0	1	3	5	0	3
B	0	1	3	5	0	3
C	0	1	3	5	0	3
D	0	1	3	5	0	3
E	0	1	3	5	0	3
F	0	1	3	5	0	3

~~No we can see that 3rd and 4th iteration~~

we can see that after 3rd iteration do not change (in the 4th iteration).

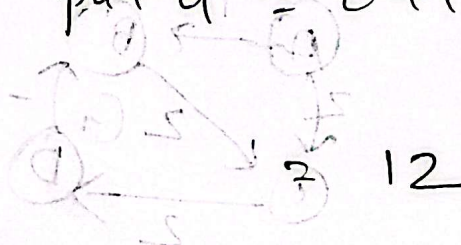
so do not need next iterations.

this is advantage of this algorithm. we are do not need 5th iteration.

Now shortest path = $0 + 1 + 3 + 5 + 0 + 3$

A=0
B=1
C=3

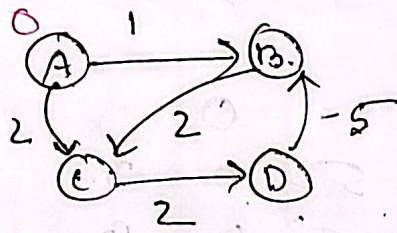
D=5
E=0
F=3



Subject:.....

Date:.....

Draw back



$$\begin{aligned} \text{iteration} &= |V| - 1 \\ &= 4 - 1 \\ &= 3 \end{aligned}$$

1st

	A	B	C	D
A	0	1	2	∞
B	0	1	2	∞
C	0	1	2	∞
D	0	1	2	4
	0	-1	2	4

2nd

	A	B	C	D
A	0	-1	2	4
B	0	-1	2	4
C	0	-1	2	4
	0	-1	2	3
	0	-2	1	3

3rd

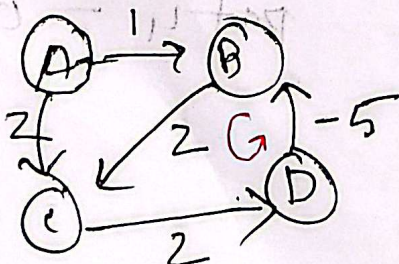
	A	B	C	D
A	0	-2	1	3
B	0	-2	1	3
C	0	-2	0	3
D	0	-2	0	2
	0	-3	0	2

4th

	A	B	C	D
A	0	-3	0	2
B	0	-3	0	2
C	0	-3	-1	2
D	0	-3	-1	1
	0	-4	-1	1

here ~~change~~ here need max
3rd iteration, but
there is no matching
result as last iteration

so, this a drawback of
Bellman-Ford algorithm



→ there is a cycle
where total = $2 + 2 - 5$
= -1

Subject:

Date:

if summation of cycle edge comes negative then which graph on do not work bellman Ford algorithm.

→ A negative weight cycle in a graph whose sum of edge weights is negative. If we traverse the cycle, the total weight accumulated would be less than zero.

→ In the presence of negative weight cycle in the graph, the shortest path doesn't exist. because with each traversal of the cycle shortest path keep decreasing.

complexity:

Bellman - Ford relaxation all E edges
exactly $(V-1)$ times,

$$\text{So, } T = (V-1) \times E$$

$$T = O(VE)$$

~~if the gr~~

$\therefore$ Best and average case $= O(VE)$

If the graph is dense then ~~the~~

$$E \sim V^2 \quad E \sim V$$

$$\text{total edge} = \frac{E(E-1)}{2} = \frac{V(V-1)}{2}$$

$$\text{time required} = (V-1)$$

$$\therefore \text{total time} = \frac{V(V-1)}{2} \times (V-1)$$

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

$$= \frac{1}{2} (V^3 - V^2 - V + 1)$$

$\therefore$ worst case

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

Subject:.....

Date:.....

Pseudocode:

Bellman-Ford (G, w, s)

Initialize - single-source (G, s)

for $i = 1$ to $|G.V| - 1$

do for each edge $(u, v) \in G.E = O(E)$

do Relax (u, v, w) $O(V)$

for each edge $(u, v) \in E = O(E)$

if $d[v] > d[u] + w(u, v)$ $O(1)$

return false

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

return true.

$T.C = O(V \cdot E)$

Subject:.....

Date:.....

Relaxation :

Initialize-single-source (G, s)

for each vertex $v \in G.V$

$$d[v] = \infty$$

$$\pi[v] = \text{Nil}$$

$$d[s] = 0$$

Relax(u, v, w)

if $d[v] > d[u] + w(u, v)$

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

$$\pi[v] = u$$

Subject:

Date:

Dijkstra's Algorithm

SSSP: The single source shortest path means finding the minimum cost (shortest) path from one source vertex to all other vertices in a graph.

→ Only one source, but many destinations.

Algorithm that solve SSSP

- ① Dijkstra's algorithm
 - all edge weights are non-negative
 - very fast
 - TC: $O(E \log V)$
- ② Bellman Ford
 - works even with negative weights
 - detect negative cycle
- ③ DAG shortest path algorithm

↓

Directed Acyclic Graph (DAG)

TC: $O(V + E)$

uses topological sorting.

Core Idea:

All ssdp algorithm use a common idea:
Update (relax) the distance to each vertex whenever we find a shortest path

Relaxation:

if $\text{dis}[u] + \text{weight}(u,v) < \text{dis}[v]$

$\text{dis}[v] = \text{dis}[u] + \text{weight}(u,v)$

General steps:

① Initialization

• $\text{dis}[\text{source}] = 0$

• $\text{dis}[\text{other}] = \infty$

② Relaxation

• For each edge (u,v) try to reduce.

$\text{dis}[v]$

③ Repeat or finalize:

- Dijkstra's finalize nodes one by one
- Bellman-Ford repeat relaxation $V-1$ times

① cat put: dis array = shortest distance from source to all vertices.

Application:

- Google maps / GPS navigation
- Internet routing algorithm
- Game AI path finding
- Transportation networks
- Robotics navigation.

Dijkstra's Algorithm

is a graph SSP algorithm used to find the shortest path from a source node to all other destination nodes in a weighted graph.

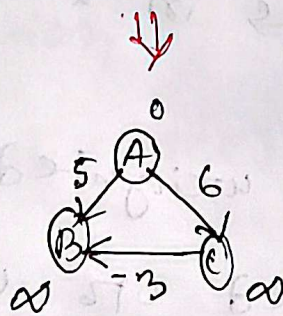
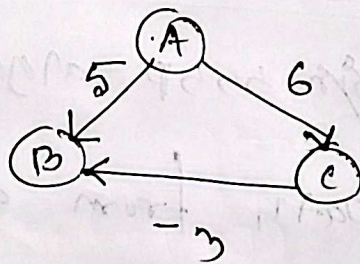
- Use:
- ~~weighted~~ weighted Graph
 - Directed or undirected graph
 - Non-negative weight only.

Why Dijkstra's doesn't work on negative weights?

Because it selects the minimum distance vertex and finalize it.

If negative weights existed, a later update could reduce that value, which break the logic.

Bx:



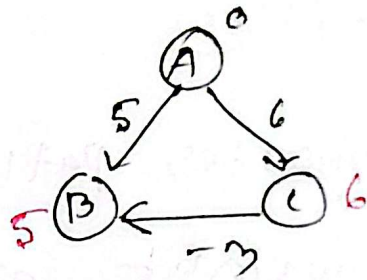
~~here, distance of A is~~

here, distance of A to B is 5 (because $5 < \infty$)

similarly A to C is 6

Subject: _____

Date: _____



$$\text{now, } A \rightarrow B = 5$$

$$A \rightarrow C = 6$$

$$C \rightarrow B = -3$$

Dijkstra mark B visited with distance 5. But the real shortest path is

$$A \rightarrow C \rightarrow B = 6 + (-3) = 3$$

As 3 is than 5, but dijkstra gives the incorrect ans as 5, which is not the shortest distance. Therefore dijkstra fails for negative graph.

Note: If the graph has a negative edge, after relaxation can produce a smaller distance, which dijkstra never check again because the node is already 'visited'.

Algorithm

① single source shortest path problem:

→ No negative weight edges $w(u,v) \geq 0 \forall (u,v) \in E$

② maintain two set of vertices:

S = vertices whose final shortest path weights have already been determined

Q = vertices in $V-S$: min priority queue

Keys in Q are estimates of shortest-path weights ($d[v]$)

Pseudocode:

DIJKSTRA (G, w, s)

Initialize-single-source (G, s) → $O(V)$

$S = \emptyset$

$Q = V[G]$

→ $O(V)$ build min heap

while $Q \neq \emptyset$

→ $O(V)$ times

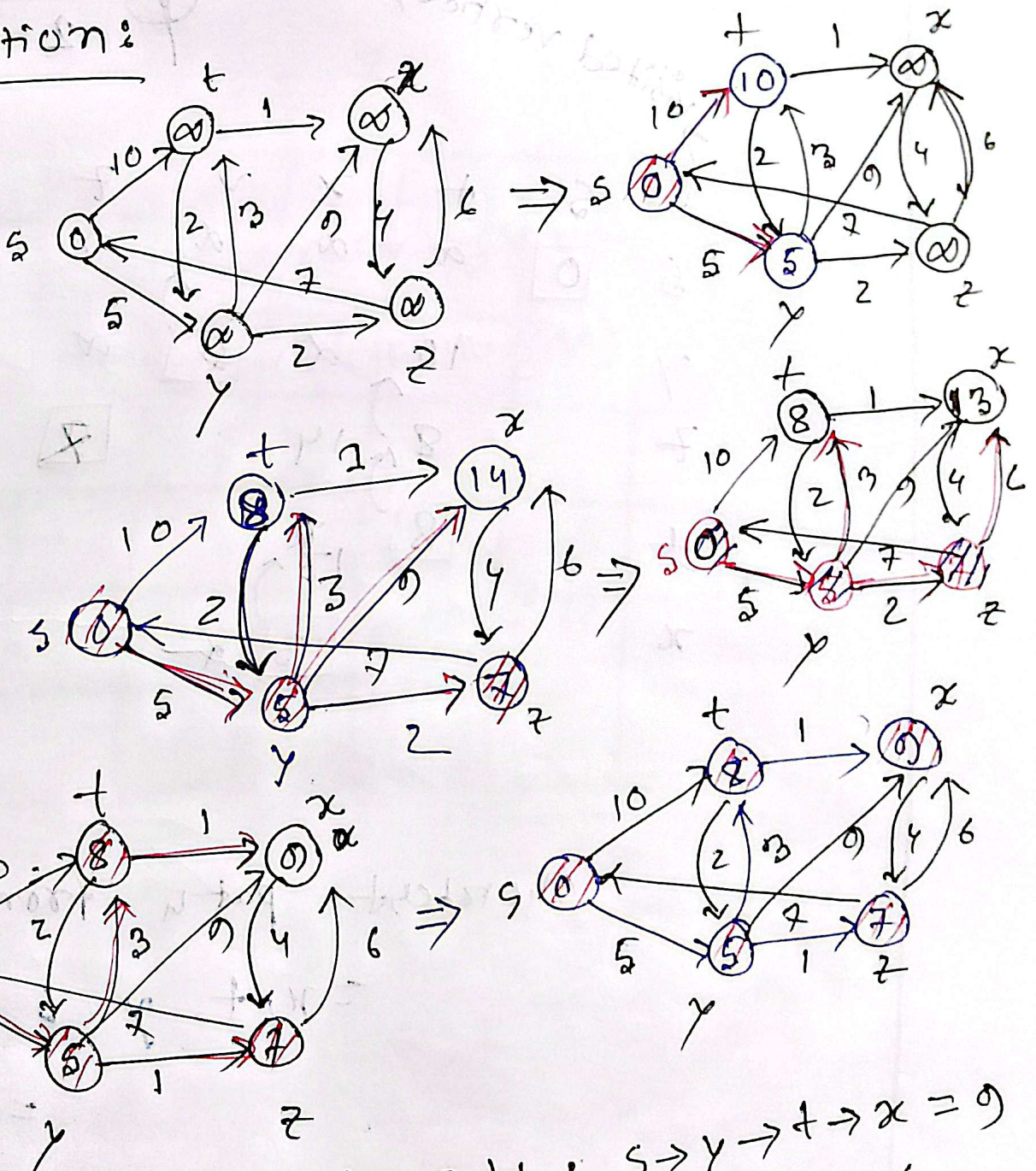
$u = \text{Extract-min}(Q) \rightarrow O(\log V)$

$S = S \cup \{u\}$

for each vertex $v \in G$. adjacent

Relax $(u, v, w) \rightarrow O(E) ; O(\log V)$

Simulation:

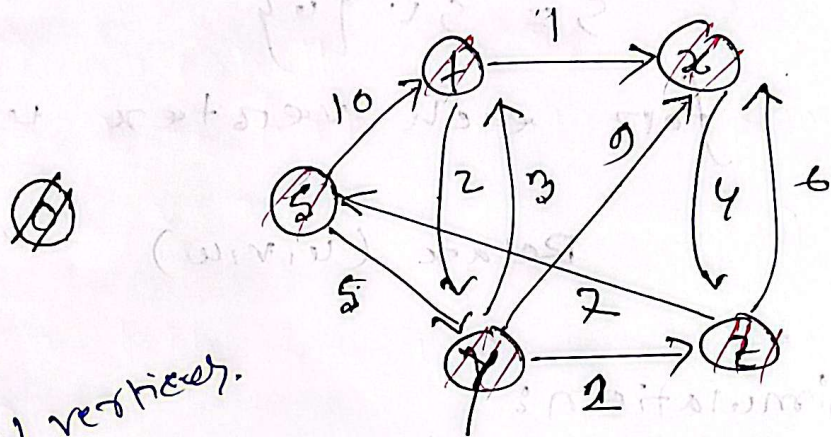


shortest path: $s \rightarrow y \rightarrow t \rightarrow x = 9$
 $s \rightarrow y \rightarrow z = 6$

Subject:.....

Date:.....

Another way:



visited vertices.

	s	t	x	y	z
s	0	∞	∞	∞	∞
y		10	∞	5	∞
z		8	14		2
t		8	13		
x			13		

shortest path from s to x

$$= x, t, y, s = s, y, t, x$$

$$= 5 + 3 + 1$$

$$= 9$$

Subject:.....

Date:.....

similarly

2 to 2

$$= 2, 1, 5$$

$$= 2, 1, 2$$

$$= 0 + 5 + 2$$

$$= 7$$

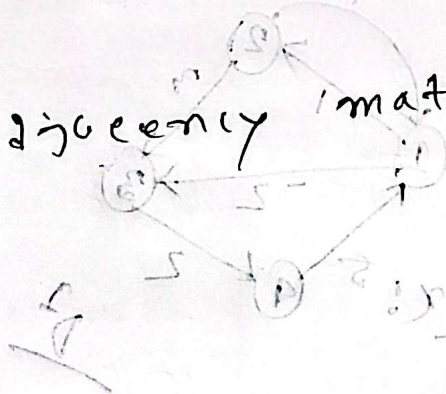
complexity

$$\text{total time} = O(\sqrt{V} \log V + E \log V)$$

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

$$\text{space} = O(V + E)$$

For use adjacency matrix, $T_c = O(V^2)$



Subject:.....

Date:.....

Floyd - Warshall Algorithm all pair shortest path

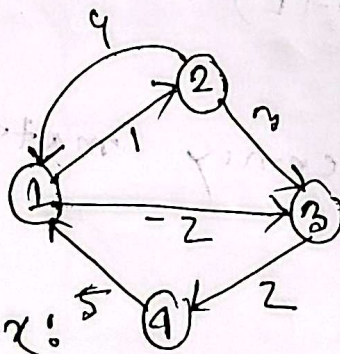
The Floyd Warshall algorithm is a DP algorithm used to find shortest paths between all pairs of vertices in a weighted graph.

→ works for both directed & undirected graph.

→ can handle positive and negative edge weights (but no negative cycle)

→ solve all pairs shortest path problem.

Ex:



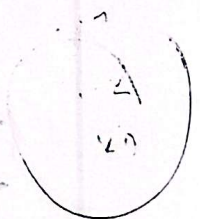
Need matrix = no. of vertices.

(4) matrix

distance matrix:

	1	2	3	4
1	0	4	-2	5 ∞
2	4	0	3	∞
3	∞	∞	0	2
4	5	∞	∞	0

	1	2	3	4
1	0	1	-2	∞
2	4	0	2	∞
3	∞	∞	0	2
4	5	6	3	0



Subject:

Date:

Formula:

$$D^k[i, j] = \min \{ D^{k-1}[i, j], D^{k-1}[i, k] + D^{k-1}[k, j] \}$$

Now

$k=1$

$$\begin{aligned} D^1[2, 3] &= \min \{ D^0[2, 3], D^0[2, 1] + D^0[1, 3] \} \\ &= \min \{ 3, \overset{4-2}{\cancel{4+0}} \} = \min \{ 3, 2 \} \\ &= 2 \end{aligned}$$

$$D^1[2, 4] = \min \{ \infty, 4 + \infty \} = \infty$$

$$D^1[3, 2] = \min \{ \infty, \infty + 1 \} = \infty$$

$$\begin{aligned} D^1[3, 4] &= \min \{ D^0[3, 4], D^0[3, 1] + D^0[1, 4] \} \\ &= \min \{ 2, \infty + 5 \} = 2 \end{aligned}$$

$$\begin{aligned} D^1[4, 2] &= \min \{ D^0[4, 2], D^0[4, 1] + D^0[1, 2] \} \\ &= \min \{ \infty, 5 + 1 \} = 6 \end{aligned}$$

$$\begin{aligned} D^1[4, 3] &= \min \{ D^0[4, 3], D^0[4, 1] + D^0[1, 3] \} \\ &= \min \{ \infty, 5 + \cancel{2} \} = 3 \end{aligned}$$

Subject:.....

Date:.....

D^2

	1	2	3	4
1	0	1	-2	∞
2	4	0	2	∞
3	∞	∞	0	2
4	5	6	3	0

D^3

	1	2	3	4
1	0	1	-2	0
2	4	0	2	4
3	∞	∞	0	2
4	5	6	3	0

D^4

	1	2	3	4
1	0	1	-2	0
2	4	0	2	∞
3	7	8	0	2
4	5	6	3	0

Subject:

Date:

All Pair shortest path

Given:

- Directed graph $G = (V, E)$
- weighted function $w: E \rightarrow \mathbb{R}$

compute:

- The shortest path between all possible pairs of vertices in a graph
- representation of the graph result, an $n \times n$ matrix of shortest path distances $\delta(u, v)$

Run Bellman-Ford once from each vertex.

but now the complexity. $O(V^2 E)$, which is $O(V^4)$ if the graph is dense ($E = \Theta(V^2)$) dense ($E = \Theta(V^2)$)

→ If no negative edge weights edges, could run Dijkstra algorithm once from each vertex.

~~Here complexity comes $O(V^2 E \log V)$ with~~

where time complexity $O(VE \log V)$ with binary heap, $O(V^3 \log V)$ if the graph is dense ($E = O(V^2)$)

Now, we can solve the problem in $O(V^3)$ with no elaborate data structures

Assume the graph G is given as adjacency matrix of weights.

→ $W = (w_{ij})$, $n \times n$ matrix $|V| = n$

→ vertices number 1 to n

$$w_{ij} = \begin{cases} 0 & \text{if } i=j \\ \text{weight of } (i,j) & \text{if } i \neq j, (i,j) \in E \\ \infty & \text{if } i \neq j, (i,j) \notin E \end{cases}$$

Output the result in an $n \times n$ matrix

$$D = (d_{ij}), \text{ where } d_{ij} = \delta(i,j)$$

Subject:.....

Date:.....

pseudocode:

Extend-shortest-path (L, w)

create L' , an $n \times n$ matrix

for $i = 1$ to n

for $j = 1$ to n

$L'_{ij} = \infty$

for $k = 1$ to n

$L'_{ij} = \min(L'_{ij}, L_{ik} + w_{kj})$

return L'

Running time: $O(n^3)$

Show-all-pairs-shortest-path (w)

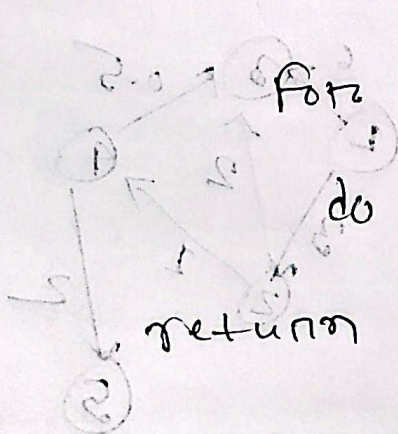
$L \leftarrow w$

for $m \leftarrow 2$ to $n-1$

$L^{(m)} \leftarrow \text{extend}(L^{(m-1)}, w)$

return $L^{(n-1)}$

running time: $O(n^4)$



Subject:.....

Date:.....

Floyd - Warshall

Given:

~~~~~

— Directed, weighted graph  $G=(V,E)$

— negative ~~edge~~ weight edges may be present

— no negative weight cycle could be present in the graph.

• the structure of shortest path.

→ vertices in  $G$  are given by  $v = \{1, 2, \dots, n\}$

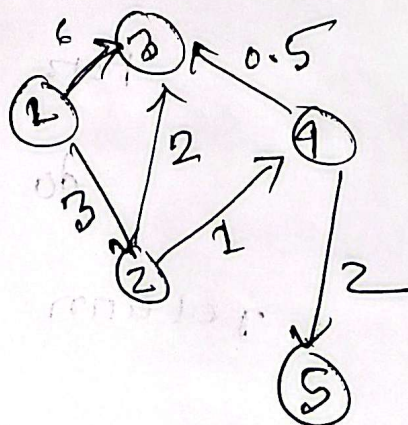
→ consider a path  $p = \{v_1, v_2, \dots, v_k\}$

an intermediate vertex of  $p$  is

any vertex in the set  $\{v_2, v_3, \dots, v_{k-1}\}$

Ex:  $p = \{1, 2, 4, 5\}$

$\therefore \{2, 4\}$





Subject:.....

Date:.....

Pseudocode:

Floyd-Warshall (w)

$D \leftarrow W$  //  $D^{(0)}$

for  $k \leftarrow 1$  to  $n$  // compute  $D^{(k)}$  ...  $O(V)$   
let  $D^{(k)} = (d_{ij}^{(k)})$  be a new  $n \times n$  matrix  
for  $i \leftarrow 1$  to  $n$  ...  $O(V)$

for  $j \leftarrow 1$  to  $n$  ...  $O(V)$   
 $d_{ij}^{(k)} = \min \left( d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \right)$

return  $D^{(n)}$

$O(V) \cdot O(V) \cdot O(V)$   
 $= O(V^3)$

Time needed:  $O(n^3), O(V^3)$

space needed:  $O(V^2), O(n^2)$

store the distance matrix

distance  $[v][u] \rightarrow O(V^2)$



Subject:.....

Date:.....

### Application:

- All pair shortest path
- Negative cycle detection
- Network routing
- Traffic system
- Social networks
- GIS application
- Transitive closure
- AI path planning

graph exists