			A	B	C	B	D	A	D
		0	1	2	3	4	5	6	7
A	1	0	0	0	0	↖1	←1	↖1
B	2	0	↖1	←1	←1	←1	↖2	←2
C	3	0	↑1	←1	↖2	←2	←2	←2
B	4	0	↖1	←1	←1	←1	↖3	←3
D	5	0	↑1	↖2	←2	←2	←3	←3
A	6	0	↑1	←2	←2	↖3	←3	↖3
D	7	0	↑1	↖2	←2	←3	←3	←3

27/9/22

LENGTH-LCS (X, Y)

1. m ← length[X]
2. n ← length[Y]
3. for i ← 1 to m → can also be taken 0, but not taken as 0 for unnecessary initialization.
4. do c[i, 0] ← 0. // all the 1st row and 1st columns are made 0.

```
5. for j←0 to n.
6. do C[0,j]←0
7. for i←1 to m.
8. do for j←1 to n
9. if Xi = Yi
10.     do C[i,j]←c[i-1, j-1]+1
11.     b[i,j]←"↖"
12. else if c[i-1,j] ≥ c[i,j-1]
13.     c[i,j] ← c[i-1, j]
14.     b[i,j]←"↑"
15.     else c[i,j] ← c[i, j-1]
16.         b[i,j]← "←"
17. return b and c.
```

```
PRINT-LCS (b, x, i, j)
1. if i=0 or j=0.
2.     return
3. if b[i,j] ←"↖"
4.     do PRINT-LCS(b,X, i-1, j-1)
5.             print Xi
6. else if b[i,j]←"↑"
7.     do PRINT-LCS (b,X, i-1, j)
8.     else PRINT-LCS (b,X, C, j-1)
```

Greedy Methods

Dynamic programming contains certain steps, in which each steps has certain choices where all are executed and the best answer is found. But in greedy algorithms, only certain choices are considered which may or may not lead to a perfect answer. It uses the same approach as of dynamic programming.

Activity Selection Problem

Ex:

i	1	2	3	4	5	6	7	8	9	10	11	12
Si	1	3	0	5	3	5	6	8	8	2		12
Fi	4	5	6	7	8	9	10	11	12	13		17

according to greedy approach, the answer is $\langle a_1, a_4, a_8, a_{11} \rangle$

3/11/22

Algorithm for optimization problem go through a sequence of steps with set of choices at each step.

* A greedy algorithm always make a choice that looks best at that moment ~~that dis~~ i.e. it makes ~~equality~~ locally optimal choices in the hope that these choices will lead to a globally optimal solution.

Steps of greedy method

(i) Determine the optimal substructure of problem

(ii) Develop a recursive solution.

(iii) Prove that at any stage of recursion, one of the optimal choice is a greedy choice. So, it is always safe to make a greedy choice.

(iv) Show that one of the subproblem induced by having made greedy choice is empty.

Activity Selection problem

It is the problem of scheduling several competing activities that require exclusive use of common resource with a goal of selecting maximum size set of mutual compatible activities.

Algorithm

GREEDY- ACTIVITY - SELECTOR (s, f)

1. n ← length [s]
2. A ← {a1 }
3. i ← 1
4. for m ← 2 to n
5. if sm ≥ Fi
6. then A ← {A} ∪ {am}
7. i ← m.
8. Return A.

9/11/22

Fractional Knapsack. (Object chosen will be b/w 0 to 1).

$0 \le Obj \le 1$.

Objects	X:	X_1	X_2	X_3	X_4	X_5	X_6	X_7
Profit	P:	10	5	15	7	6	18	3
Weight	w:	2	3	5	7	1	4	1
	P/w	5	1.6	3	1	6	4.5	3

$\langle$ 6 (X5) , 5 (X1) , 4.5 (X6) , 3 (X7) , 3 (X3) , 1.6 (X2) , 1 (X4) $\rangle$

Objects	Profit $X_i P_i$	Weight $X_i w_i$	Remaining weight.
			15
$X_5(1)$	6	1	15 - 1 = 14.
$X_1(1)$	10	2	14 - 2 = 12.
$X_6(1)$	18	4	12 - 4 = 8.
$X_7(1)$	3	1	8 - 1 = 7
$X_3(1)$	15	5	7 - 5 = 2.
$X_2(2/3)$	$5 \times 2/3 = 3.3$	$2/3 \times 3 = 2$	2 - 2 = 0.
$X_4(0)$			
	$\sum X_i P_i = 55.3$	$\sum X_i = 15$.	

Greedy-fractional-Knapsack (P, W, m).

1. for i ← 1 to n.
2. calculate (P/W).
3. sort the element in decreasing order of (P/W) ratio.
4. for i ← 1 to n.
5. if (m > 0 and $w_i \leq m$).
6. do $m = m - w_i$.
7. $P_r = P_r + P_i$
8. else break;
9. if (m > 0)
10. $P = P + P_i(\frac{m}{w_i})$
11. return P.

Time complexity - n log n.
(n + n + n log n)

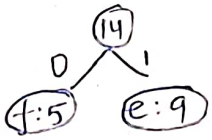
10/11/22:

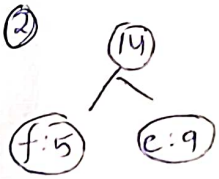
Huffman coding

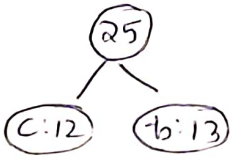
	a	b	c	d	e	f
Freauency	45	13	12	16	9	5
Fixed length code ward	000	001	010	011	100	101
Variable length code						

f:5 e:9 c:12 b:13 a:16 a:45

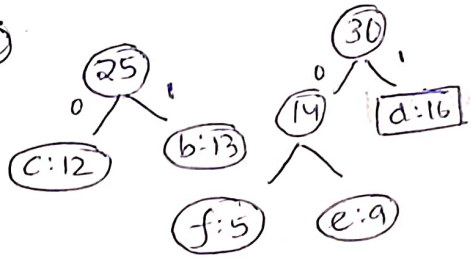
① c:12 b:13 (14) a:16 a:45



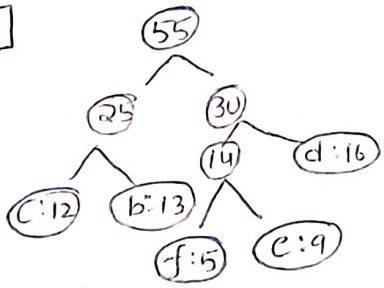
② 

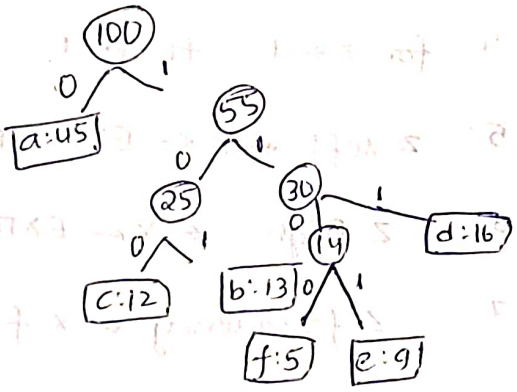
d:16 

a:45

③ 

a:45

④ a:45 

⑤ 

a: 0
b: 101
c: 100
d: 111
e: 1101
f: 1100

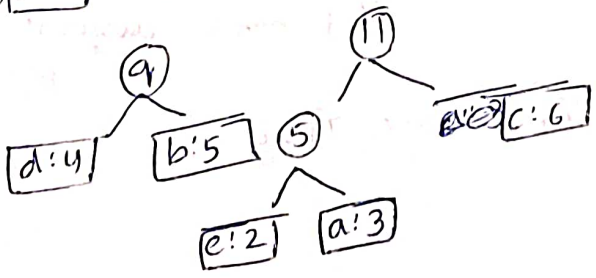
Que bcc abbddaeccbba eddcc

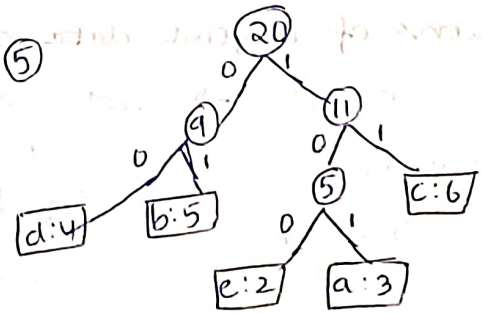
	a	b	c	d	e
Frequency	3	5	6	4	2

① e:2 a:3 d:4 b:5 c:6

② d:4 b:5 (5) c:6
e:2 a:3

③ (5) c:6 (9)
e:2 a:3 d:4 b:5

④ 

⑤ 

a:3 — 101
b:5 — 01
c:6 — 11
d:4 — 00
e:2 — 100

HUFFMAN(C)

1. $n \leftarrow |C|$
2. $Q \leftarrow C$
3. allocate a new node z
4. for $z \leftarrow 1$ to $n-1$
5. $z.left \leftarrow x \leftarrow$ EXTRA-MIN(Q)
6. $z.right \leftarrow y \leftarrow$ EXTRA-MIN(Q)
7. $z.frequency \leftarrow x.frequency + y.frequency$
8. insert (Q, z).
9. return extra-MIN(Q).

Time complexity - $(n \log n)$

14/11/22

Disjoint Data structure

$S = \{S_1, S_2, S_3 \cdots S_n\}$.

↳ S is a disjoint data structure.

disjoint $\begin{cases} S_1 = \{1, 2, 3, 4\} \\ S_2 = \{5, 6, 7\}. \end{cases}$

↳ no elements in common.

$S_1 \cap S_2 = \emptyset$

Operations of disjoint data structure.

① MAKE-SET(X) → Create a set X with only one element X.

② UNION(X, Y)

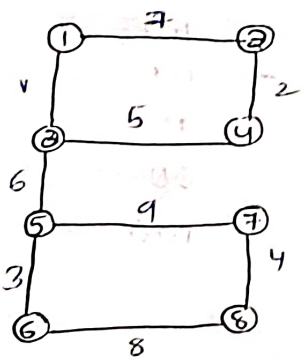
③ FIND-SET(X)

1, 2, 3, 4 ...

{1} {2} {3} {4}.

using the element X, we cannot make any other set (if made disjoint is not possible)

Q.



$W = \{1, 2, 3, 4, 5, 6, 7, 8\}$

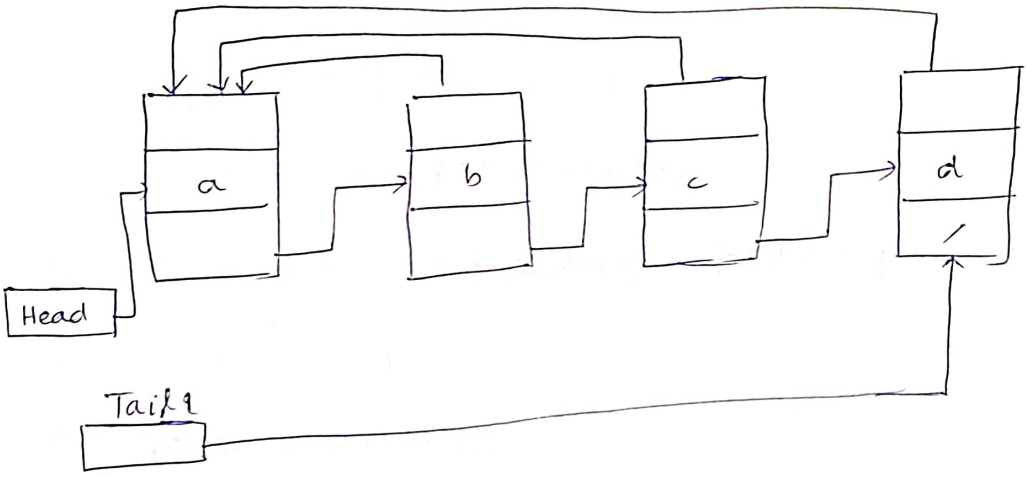
15/11/22

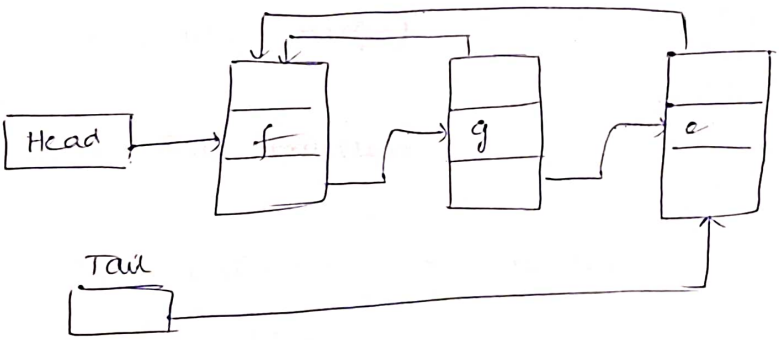
Linked list representation

$S_1 = \{a, b, c, d\}$

$S_2 = \{e, f, g\}$

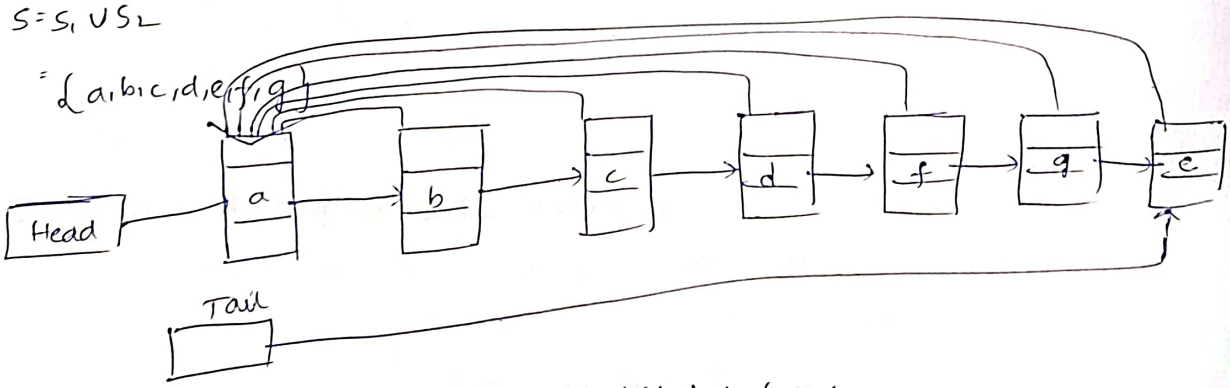
→ representative elements (any elements can be selected)





$S = S_1 \cup S_2$

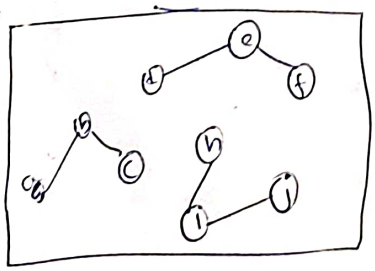
$= \{a, b, c, d, e, f, g\}$

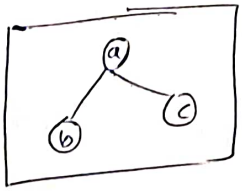


Forest: It is a collection of disjoint trees.

Every tree is a forest but every forest is not a tree.

Ex:



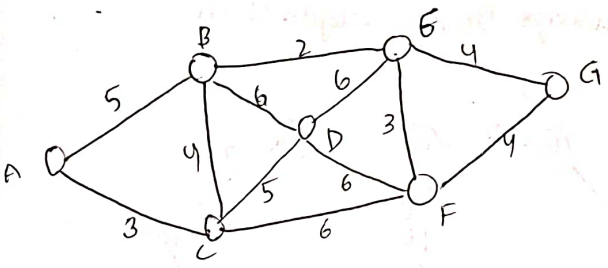


Spanning tree: A connected subgraph S of graph G that consists of G(V, E) is said to be spanning iff

(i) The subgraph contains all the vertices of graph

(ii) the no of edges in subgraph S should be (|V|-1)

Find the spanning tree using Kruskal algorithm.

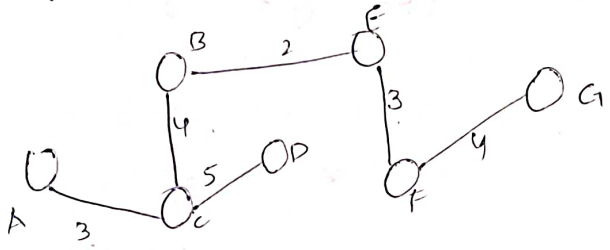


Edges = 7 | = 6

Weight of tree = $\sum w(u, v)$

= 21

Sol



Algorithm

MST-KRUSKAL(G, W)

① A ← ∅

② for each vertices V ∈ V[G]

③ do MAKE-SET(V) (undergoes V times so TC = V log V)
this will have time complexity log V

④ Sort the edges E in non-decreasing order by weight w. (E log E)

⑤ For each edge (u, v) ∈ E, taken in non-decreasing order (V)

⑥ if FIND-SET(u) ≠ FIND-SET(v)

⑦ A ← A ∪ {(u, v)}.

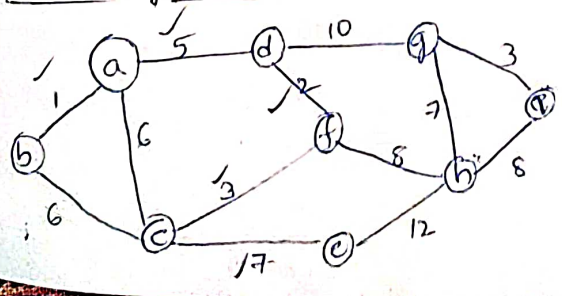
⑧ UNION(x, y)

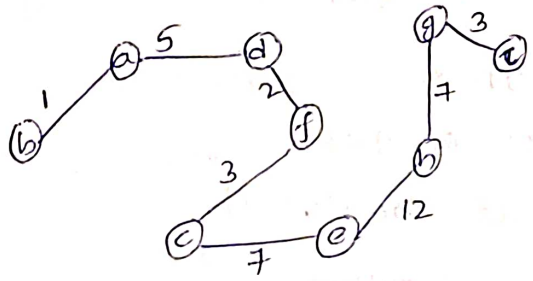
⑨ Return A.

Time complexity = |E log E| (or) E log E not given in MCQ
go for E log V.

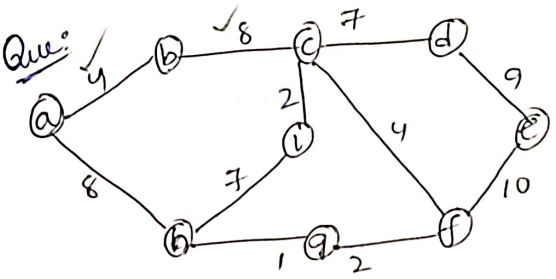
17/11/22

Prim's algorithm

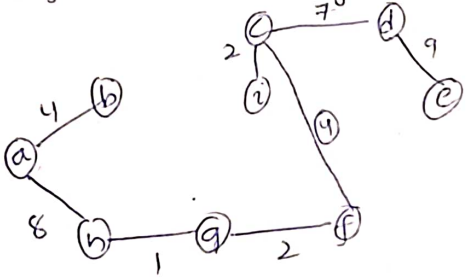




Que:

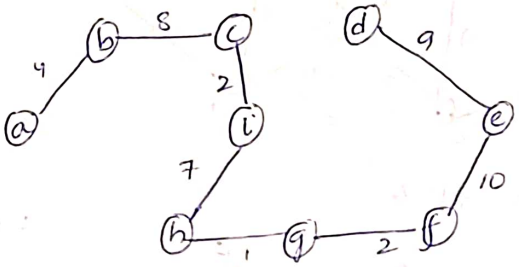


Using Kruskal's algorithm.



Ans: 37

Using Prim's algorithm.



PRIM's Algorithm

MST-PRIM(G, w, r).

1. For each vertex $u \in V[G]$ — $|V|$
2. do key[u] ← ∞ // key[u] → min weight of any edge that connects u to any other edge.
3. π[v] ← NIL
4. key[r] ← 0.
5. Q ← V[G] // Heap (log V)
6. while Q ≠ ∅ (V)
7. do u ← EXTRACT-MIN(Q) (V log V).
8. for each v ∈ adj[u]
9. do if v ∈ Q and w(u,v) < key[v].
10. π[v] ← u
11. key[v] ← w(u,v) (E log V)

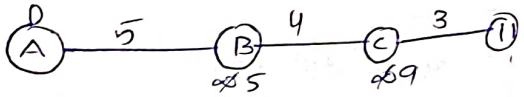
Time complexity = $E \log V$. $(V + V\log V + E\log V)$.

21/11/22

Single source shortest path problem

This problem is dealt using Dijkstra's algorithm

Relaxation: (relaxation of the edges).



If $d(v) > d(u) + w(u,v)$, then $d(v) = d(u) + w(u,v)$

Algorithm

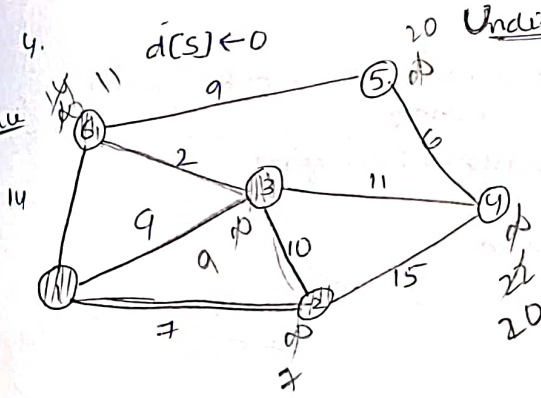
1. RELAX (u, v, w)
2. If $d[v] > d[u] + w(u,v)$
3. do $d[u] \leftarrow d[v] + w(u,v)$.
4. $\pi[v] \leftarrow u$.

INITIALIZE-SINGLE SOURCE (G, s).

1. For each vertex $v \in G[v]$
2. $d[v] \leftarrow \infty$
3. $\pi[v] \leftarrow NIL$
4. $d[s] \leftarrow 0$

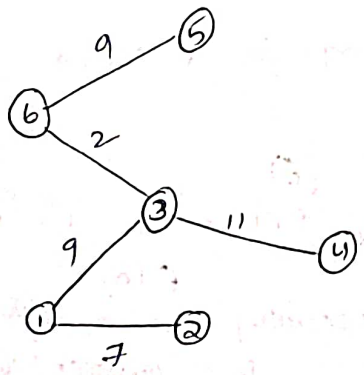
Undirected graph

Que

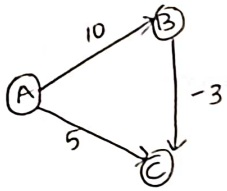


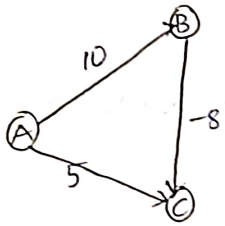
Source	Destination				
1	2	3	4	5	6
0	∞	∞	∞	∞	∞
	7	9			14
(1,2)	7	9	22	∞	14
(1,2,3)	7	9	20	∞	11
(1,2,3,6)	7	9	20	20	11
(1,2,3,6,4)	7	9	20	20	11
(1,2,3,6,4,5)					

Que



Directed graph



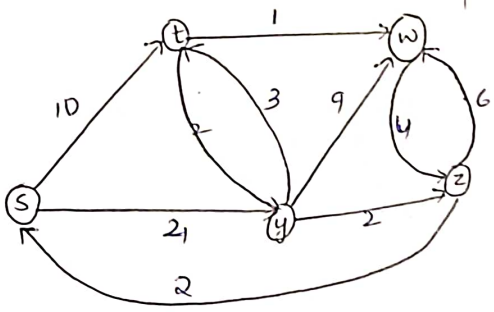


A	B	C
0	∞	∞
A	10	⑤
(A,C)	10	⑤

A	B	C
A	∞	∞
A	10	⑤
(A,C)	⑩	⑤
(A,C,B)		

Here djikstra's algorithm doesn't work.

Que:



s t y z & w

(s, t, x) = s

Problem with djikstra's algorithm is it does not work with negative numbers (weight) (may)

In this case Bellman Ford's algorithm is used.

22/11/22.

Djikstra(G, W, S).

1. INITIALISE-SINGLE-SOURCE (G, S) // (V)
2. S ← ∅
3. Q ← V[G]. // O(V)
4. while Q ≠ ∅ (V)
5. do u ← EXTRA-MIN(Q). // V log V
   (log V running for V times)
6. S ← S ∪ {u}. //(V)
7. For each v ∈ add [u]. // add → adjacent. (E)
8. do RELAX (u, v, w). // In this case only RELAX has t.c of E log V.

(generally has a constant time complexity)

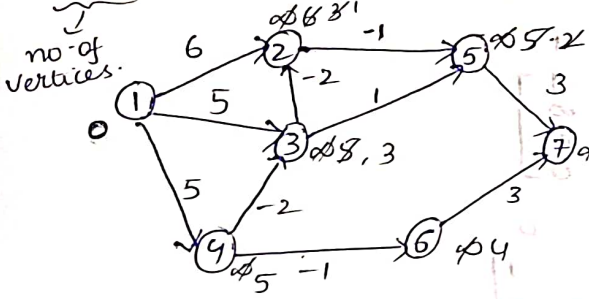
Time complexity = $O(V) + E\log V + E\log V$.

$= E\log V$.

Bellman Ford Algorithm

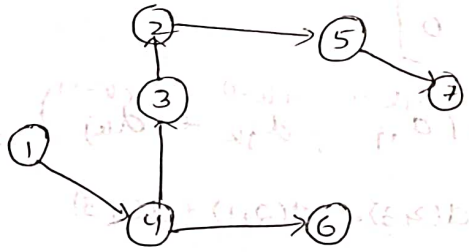
This algorithm is used when there are negative edges.

$|V[G]| - 1$ times → no of times relaxing has to be done.

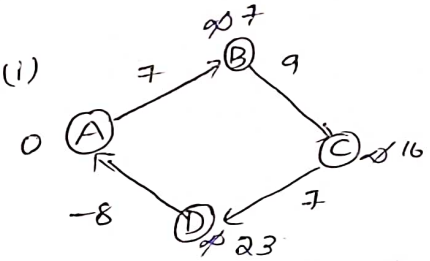


Here each edge need to be relaxed 6 times.

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

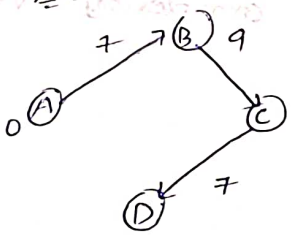


Que (i)

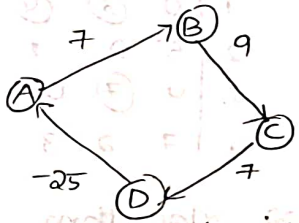


(A,B) (B,C), (C,D), (D,A)

Ans



(ii)



Here the cycle is negative weight cycle

It doesnot have any practical solution.

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

23/11/22 Algorithm.

BELLMAN-FORD(G, w, s).

1. INITIALISE-SINGLE-SOURCE (G,S)
2. For i ← 1 to $|V[G]| - 1$
3. do for each edge $(u,v) \in E[G]$
4. do RELAX (u,v,w).
5. For each edge $(u,v) \in E[G]$.
6. do if $d[v] > d[u] + w(u,v)$
7. return false.
8. return true.

All pair shortest paths

We can apply both Djikstra and Bellman-Ford Here we find out shortest path from each vertex to other vertices

Djikstra - $E\log V$.

$= VE\log V$ (For V vertices)

$V(V^2)\log V$ $(V^2 \geq E)$.

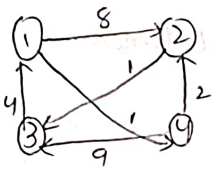
$= V^3 \log V$ (Time C).

Bellman-Ford. VE

$= V \cdot VE$

$= V^2 \cdot V^2 = V^4$.

Floyd - Warshall's algorithm.



$D^0 =$

	1	2	3	4
1	0	8	∞	1
2	∞	0	1	∞
3	4	∞	0	∞
4	∞	2	9	0

$D^1 =$

	1	2	3	4
1	0	8	∞	1
2	∞	0	1	∞
3	4	12	0	5
4	∞	2	9	0

$D^2 =$

	1	2	3	4
1	0	8	9	1
2	∞	0	1	∞
3	4	12	0	5
4	∞	2	3	0

$D^3 =$

	1	2	3	4
1	0	8	9	1
2	5	0	1	6
3	4	12	0	5
4	7	2	3	0

$D^4 =$

	1	2	3	4
1	0	3	4	1
2	5	0	1	6
3	4	7	0	5
4	7	2	3	0

$d_{ij}^{k} = \min\left(d_{ij}^{(k-1)} , d_{jk}^{(k-1)} + d_{kj}^{(k-1)}\right)$ Formula.

Eg: $d(2,3) = d(2,1) + d(1,3)$.

complexity $= V^3$.

24/11/22. Algorithm

FLOYD - WARSHALL(W).

1. $n \leftarrow rows[W]$
2. $D^0 \leftarrow W$.
3. For $k \leftarrow 1$ to n.
4. do for $i \leftarrow 1$ to n.
5. do for $j \leftarrow 1$ to n.
6. do $d_{ij}^{k} = \min\left(d_{ij}^{k-1} , d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right)$
7. return D^n.

Time complexity $= n^3$.

4 Queens question

Q_1, Q_2, Q_3, Q_4

Possibility 1

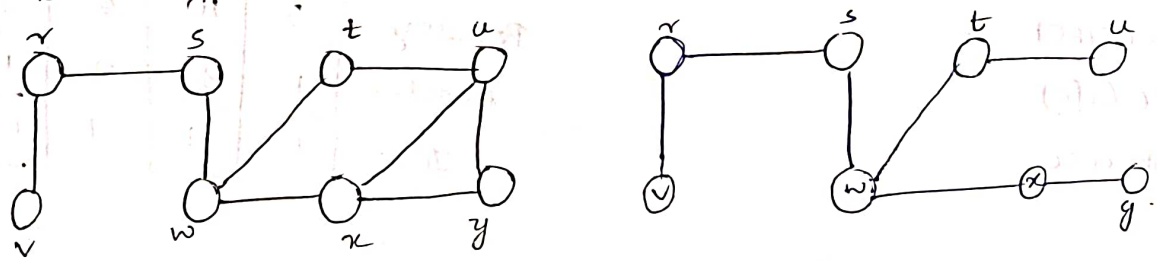
	Q_1		
			Q_2
Q_3			
		Q_4	

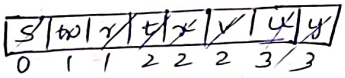
Possibility 2

		Q_1	
Q_3			
			Q_2
	Q_4		

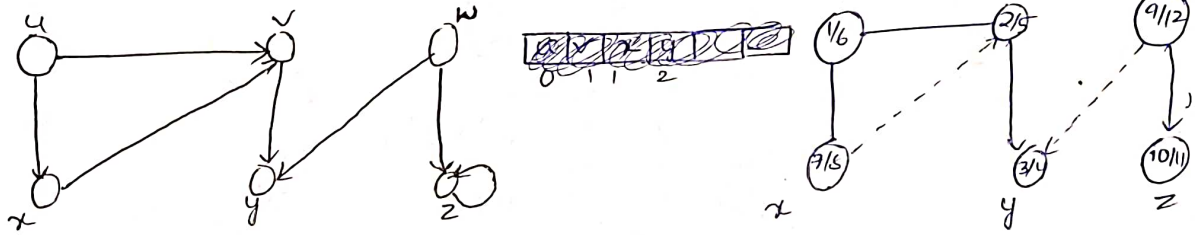
30/11/22

BFS (Breadth first search)



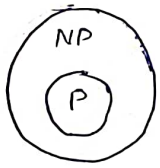


DFS (Depth first search)



12/12/22

Polynomial time algorithm	Exponential time algorithm
Linear search - n	0/1 Knapsack
Binary search - $\log n$	Travelling salesman
Selection sort - n^2	Sum of subset
Merge sort - $n \log n$	graph colouring
Matrix multiplication - n^3	



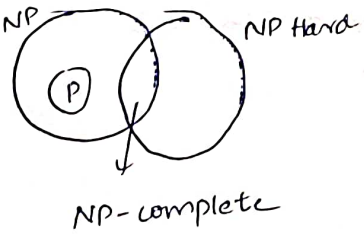
P: Deterministic alg. can solve in polynomial time.

NP: Non deterministic alg take polynomial time.

Satisfiability problem

3-CNF (Conjuctive normal form).

$CNF = (x_1 \cup \bar{x}_2 \cup x_3) \cap (\bar{x}_1 \cup x_2 \cup \bar{x}_3)$



Satisfiability $\propto$ L

If satisfiability reduces to any type of language (set of syntax)

$\Rightarrow$ NP Hard

Rabin-Karp Pattern matching algorithm

String - 

IIIT KIIT ITER.

Pattern - IIIT

$9 \times 10^3 + 9 \times 10^2 + 9 \times 10^1 + 20 \times 10^0$

$= 9000 + 900 + 90 + 20 =$ 10010.