Design & Analysis Of Algorithms

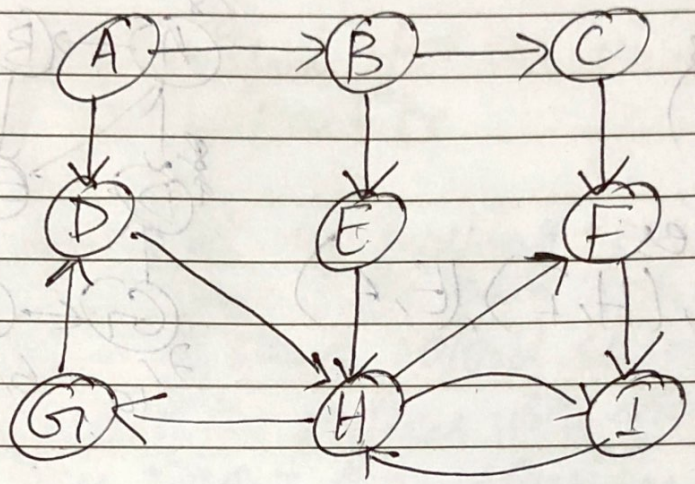
Date: 22-11-22.

M	T	W	T	F	S	S

K20-1052
S. M. Hassan Ali
BSE-5B.

Q.1

a)



DFS. from A

Stack

F
I
G
H
E
C
B
D
A

DFs (A) → DFS (B) or DFS (D)
DFS (B) → DFS (C) or DFS (E)
DFS (D) → DFS (H)
DFs (C) → DFs (F)
DFS (E) → DFS (H)
DFs (H) → DFs (G) or DFS (I)
DFs (F) → DFs (I)
DFs (G) → DFS (D)
DFs (I) → DFS (H)

DFs: {I, G, F, H, E, C, D, B, A}

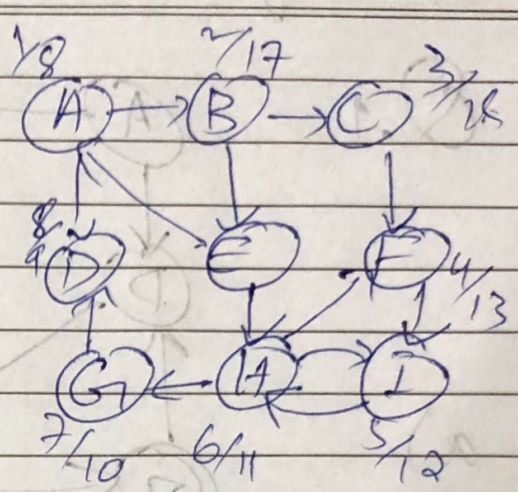
{D, G, H, I, F, C, E, B, A}

K20-1052

Date:

M	T	W	T	F	S	S

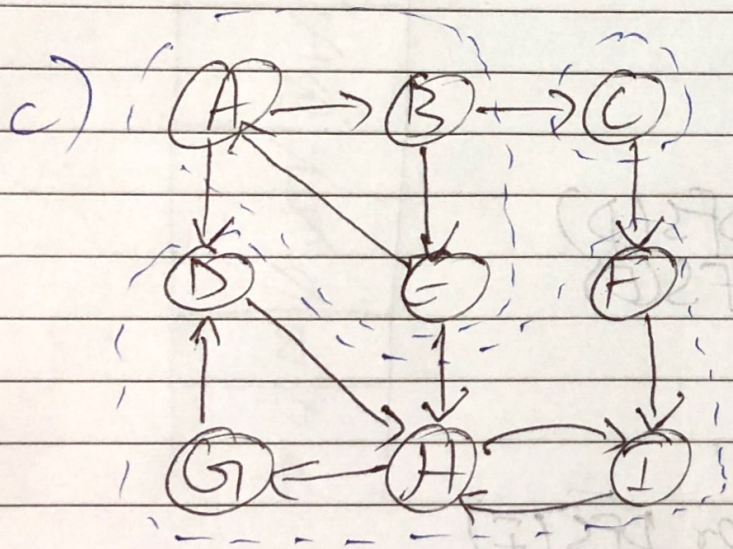
b) Forward edges
~~Null~~ (A, D)



Backward edges
(d, h), (H, f), (H, I), (E, A)

Cross Edge
(E, H)

c)



There are three strongly connected graphs.

⇒ (A, B, E), (C), (F, I, H, G, D)

K20-1052.

Date:

M	T	W	T	F	S	S

Q.2

a) Linear algorithm for an undirected graph that is bipartite.

A BFS would be run making each node with different colour than its parent. If any of the neighbour vertex is found to be of same colour, so graph is not bipartite. If we can mark the vertexes with different colours with respect to their parent, then consider the graph as bipartite.

Coloured BFS(V)

- Starting with vertex V, making adjacent node with opposite colour, Place V in queue.
- While queue is not empty

(a) x = pop(queue).

b) for every neighbor n of x.

if(n! coloured){colour it opposite then put in queue}

if (n== coloured && same coloured as x)

{ // not bipartite }.

return Graph is Bipartite.

O(V+E)

Page No.

Signature

K20-1052.

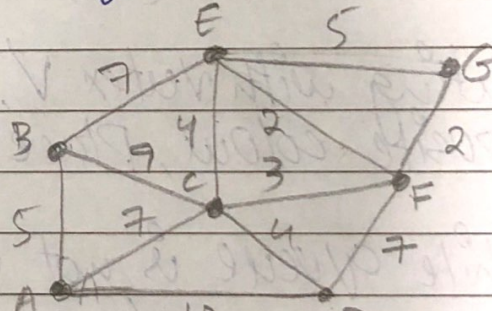
Date:

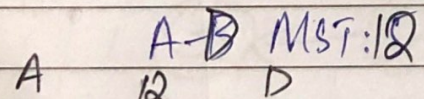
M	T	W	T	F	S	S

b) For exactly one odd length cycle in a graph atmost 3 colors are needed. If an edge is removed from graph then it results in two coloured graph with no odd length cycle. This edge can again be added in graph with third colour only if one odd length of cycle forms in graph.

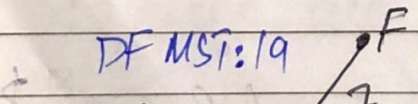
Q.3 Prims edges if n vertices use of n-1 ~~vertices~~ with no cycle.

Starting with vertex A, marked next vertex with alphabets.

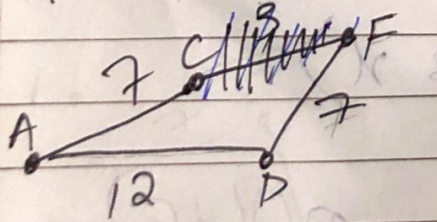




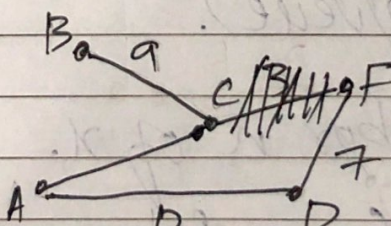
A-B MST: 12



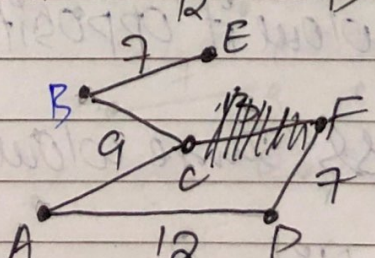
DF MST: 19



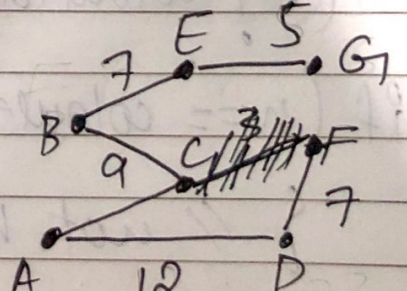
AC MST: 26



CB MST: 35



BE MST: 48



EG MST: 47

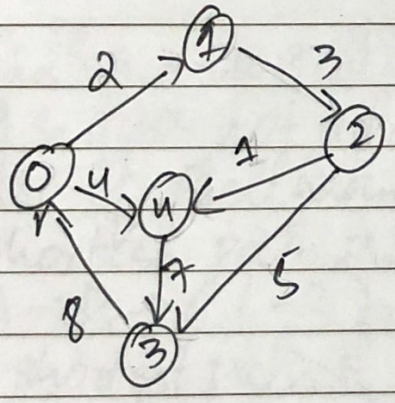
K20-1052.

Date:

M	T	W	T	F	S	S

b) Prim's and Kruskal's algorithm can yield different maximum spanning trees if the weights of the edges are not distinct. At the end the cost remains same.

Q.4



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

A_0:

	0	1	2	3	4
0	0	2	∞	∞	4
1	∞	0	3	∞	∞
2	∞	∞	0	5	1
3	8	10	∞	0	12
4	∞	∞	∞	7	0

A_1:

	0	1	2	3	4
0	0	2	5	8	4
1	∞	0	3	∞	∞
2	∞	∞	0	5	1
3	8	10	13	0	12
4	∞	∞	∞	7	0

K20-1052.

Date:

M	T	W	T	F	S	S

$A_2 =$

	0	1	2	3	4
0	0	2	5	8	4
1	2	0	3	8	4
2	2	2	0	5	1
3	8	10	13	0	12
4	2	2	2	7	

$A_3 =$

	0	1	2	3	4
0	0	2	5	8	4
1	16	0	3	8	4
2	13	15	0	5	1
3	8	10	13	0	12
4	15	17	20	7	0

$A_4 =$

	0	1	2	3	4
0	0	2	5	10	4
1	16	0	3	8	4
2	13	15	0	5	1
3	8	10	13	0	12
4	15	17	20	7	0

Dynamic Algorithm with time complexity $O(V^3)$

space complexity $O(|V|^2)$

Page No.

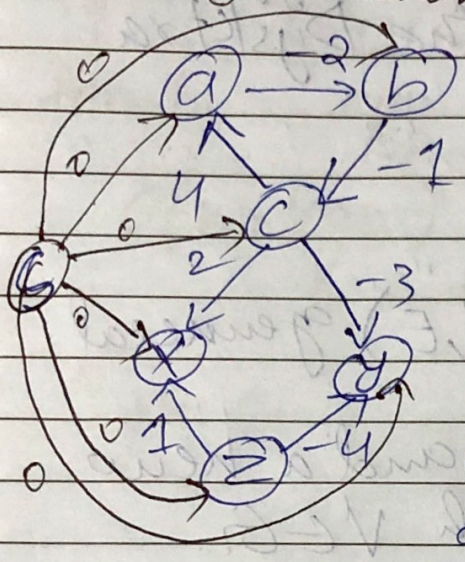
Signature

K20-1052.

Date:

M	T	W	T	F	S	S

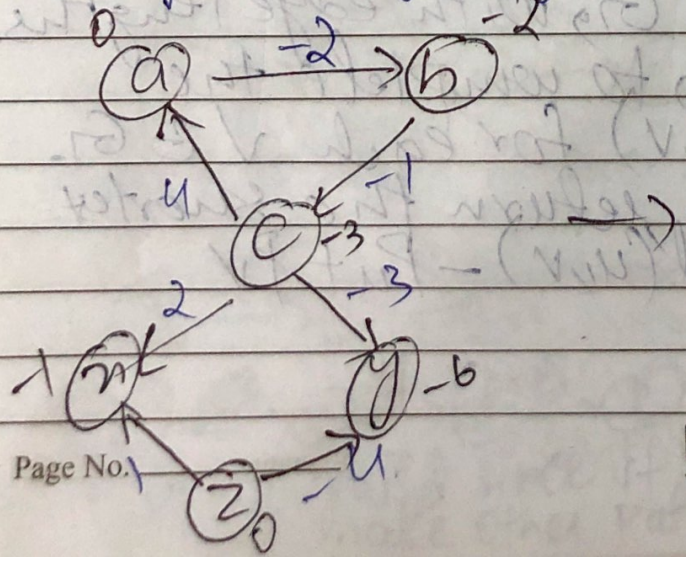
Q.5 Johnson's Algorithm part 1

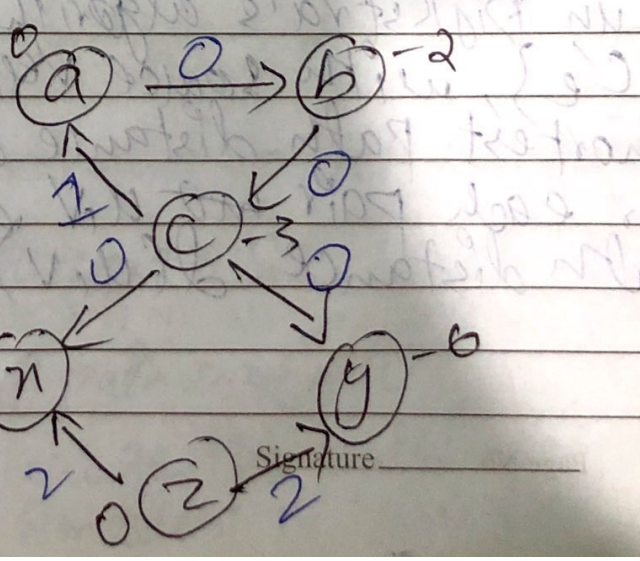


cannot use Dijsktra since we have -ve edges but can use Bellman Ford algo since we have a cycle ABC with cost 1.

In Johnson's algo we need to choose a magical vertex that can reach to all other vertex. Adding a new vertex with direct edges to every vertex with distance 0.

Now finding a shortest path from S to all other vertex. Adding S does not add any new paths for any u, v and G. Now use of Bellman Ford. S→A shortest path becomes 0. S→B shortest path = S→A→B (-2). S→C shortest path = S→A→B→C (-3). S→Z shortest path (0). S→Y shortest path = S→A→B→C→Y (-6). S→X shortest path = S→A→B→C→X (-1). We have calculated all magical weights for each vertex. Apply re-weighting technique $c'_e = c_e + P_v - P_v$





Page No.

Signature

K20-1052.

Date:

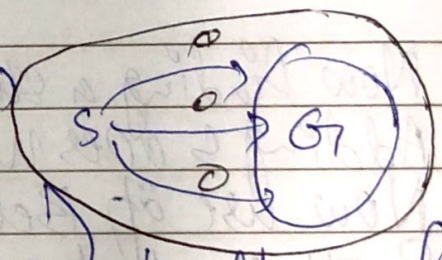
M	T	W	T	F	S	S

using re-weighting all edges are now +ve. It also preserved shortest path. Now the Dijkstra can be applied.

Johnson's Algorithm part 2.

Input is a directed graph $G = (V, E)$. genereal edge lengths c_e.

Form G' by adding a new vertex s and a new edge (s, v) with length 0 for each $v \in G$.



Run Bellman-Ford on G' with source vertex s.
(If B-F detects any -ve cycle cost in G' (which would also be present in G), halt and report this).

For each $v \in G$, define p_v = length of a shortest $s \to v$ path in G'. For each edge $e = (u, v) \in G$, define $c'_e = c_e + p_u - p_v$

For each vertex u of G:
Run Dijkstra's algorithm in G, with edge lengths $\{c'_e\}$ with source vertex u, to complete the shortest path distance $d'(u, v)$ for each $v \in G$.

For each pair ~~not~~ $u, v \in G$, return the shortest path distance $d(u, v) = d'(u, v) - p_u + p_v$

K20-1052.

Date:

M	T	W	T	F	S	S

Running time:

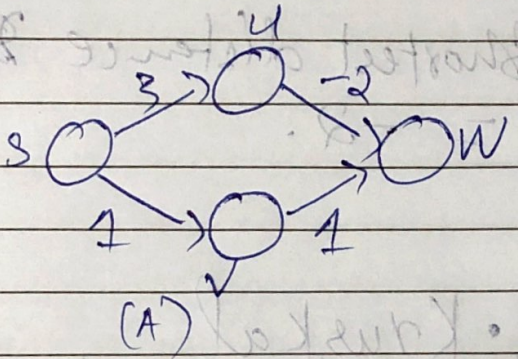
$O(n) + O(mn) + O(m) + O(nm\log n) + O(n^2)$

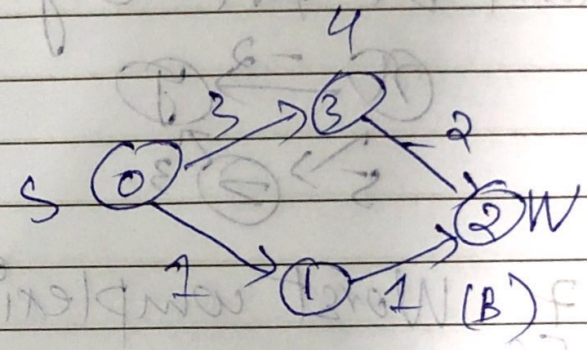
form G' using BP — form c' — n-Dijkstra — for every pair shortest path

$= O(mn\log n)$ much better than Floyd-Warshall for sparse graphs.

Q.6

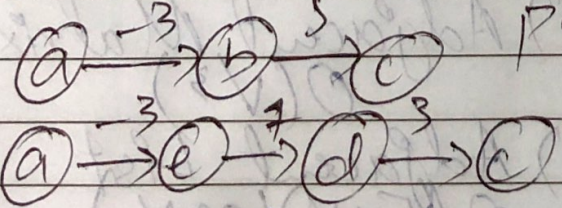
a)



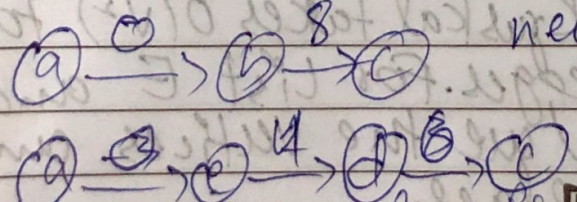


Applying Dijkstra's algo to (A) with S vertex as source then we have s, v, w, u. where the path s to w is not shortest as it can be seen in (B).

b)



add 3 to make it the cost +ve and to all the edges



It is not valid since it will change the path cost and make other path look smallest or largest.

Page No.

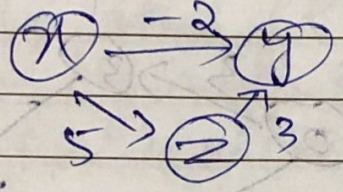
Signature

K20-1052.

Date:

M	T	W	T	F	S	S

c) In most cases Dijkstra fail as it greedily chooses node to close at each step. But If the directed acyclic graph has only the edges that leaves the source node that are -ve, then we can successfully use Dijkstra Algorithm. And remaining edges must be +ve. E.g



Shortest distance x to y = -2.

Q.7 Worst complexity

- BFS
  - i) Adjacent Matrix $O(|V|^2)$
  - ii) Adjacent List $O(V+E)$

- DFS
  - i) Adjacent Matrix $O(V^2)$
  - ii) Adjacent List $O(V+E)$

- BFS and DFS have same worst complexities as in the worst case whole array/adjacent vertex with edges are visited.

- Kruskal
  - i) Adjacent Matrix $O(V^2)$
  - ii) Adjacent List $O(E \log V)$

- Prims
  - i) Adjacent Matrix $O(V^3)$
  - ii) Adjacent List $O((E+V)\log V)$

- Kruskal takes $O(V^2)$ to find edges. For List E and V are the vertices and edges.

Signature