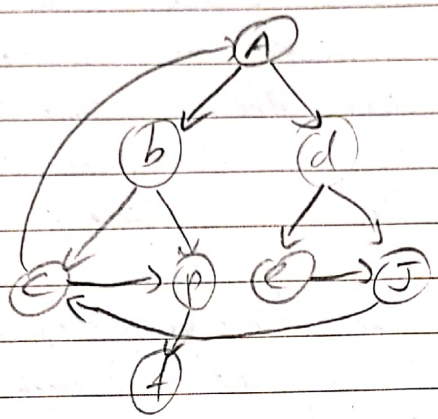
Shehryar Azam

Section-5 Algo Assignment No 4 k17-0295

Q1(a) tree with starting node 'a'

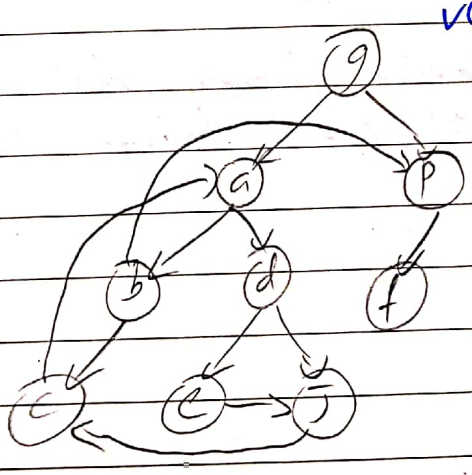
j
e
d
f
p
c
b
a



subset 2: visit order => j, e, d, f, p, c, b, a
thru dfs.

tree with starting node 'g', cause we had 4 unvisited vertices left (g, h, s, k) and selecting g cause alphabetic.

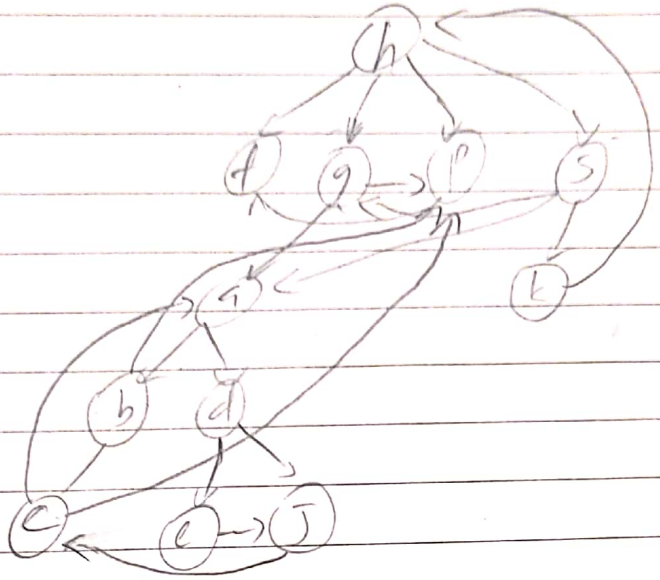
f
p
j
e
d
c
b
a
g



subset 2: visit order: f, p, j, e, d, c, b, a, g
thru dfs

k
s
p
j
e
d
c
b
a
g
f
h

tree with starting node 'h'



subject 3 visit order : k, s, p, j, e, d, c, b, a, g, f, h
three dots.

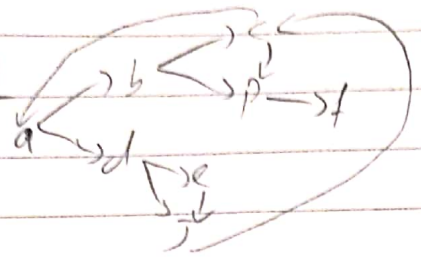
(b). for subject 3: forward edge: none
backward edges: k → h, c → a, c → p,
cross edges: g → p, s → g, e → j, j → c

for subject 2: // edge : none
backward edge: b → p, c → a
cross edge : e → j, j → c

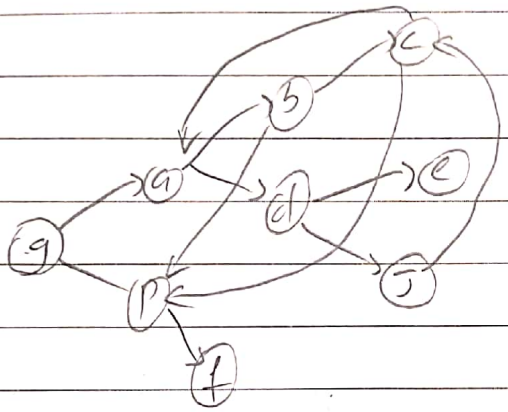
for subject 1: forward edge : none
backward edge: c → a
cross edge : e → j, c → p

strongly connected graphs.

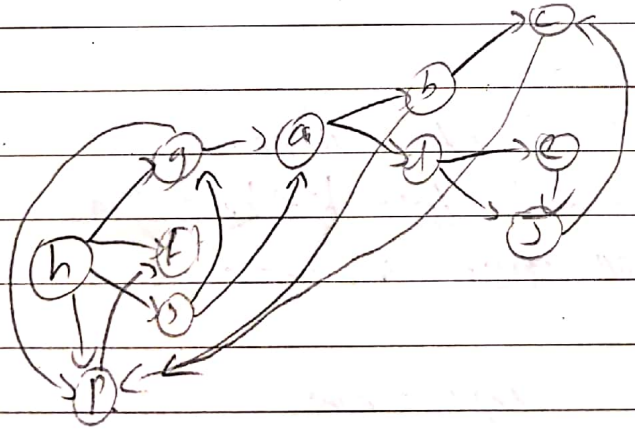
Q1(c) for subpart 2



for subpart 2.



for subpart 3



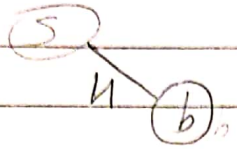
Q2. Algorithm for Maze problem.

```
start_position = board[0][0]
rows = columns = 8
pathLength = 1;

find_shortest_Path(Row, Column, PathLength)
{
  // base condition
  if(row == 7 && column == 7)
      return pathLength;
  // checking validity of our move
  else{
      if(if row >= 0 & row < 8 And col >= 0 && col < 8
          And board position is not blocked or visited)
      {
          board[row][col] <- visited
          pathLength ++;
      }
      else
          return 'infinity/∞';

      // 4 steps/moves through recursive call
// moving down :  find_shortest_path(row++, column, pathLength)
// moving right   //   //   (row, column++, pathLength)
// moving left    //   //   (row--, column, pathLength)
// moving up      //   //   (row, column--, pathLength)
  }

// caller function   find_shortest_path(0,0,1)
                                  initial position
                                        pathLength
```

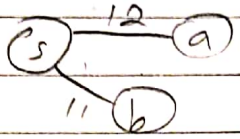
prim's algorithm.

Q)

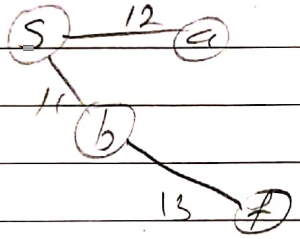
step (1) initial node = (s)



step (2)

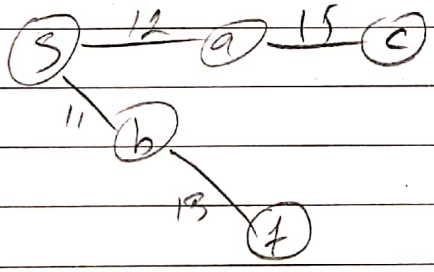


step (3)



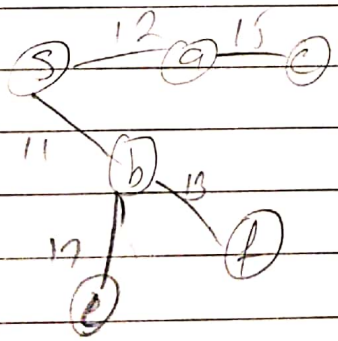
cost of going to
c = 27
d = 29
e = 28
~~f = 24~~

step (4)

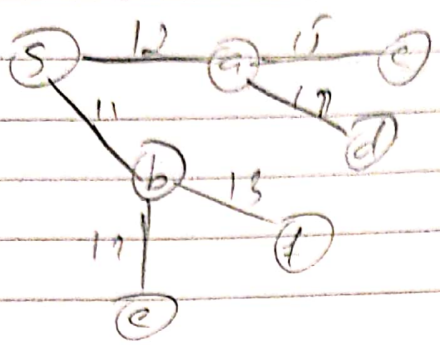


cost of going to
c = 27
d = 29
e = 28
g = 35/5 = 34
h = 34

step (5)



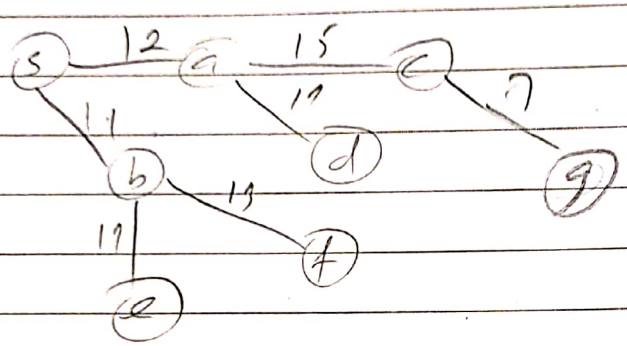
step ⑥



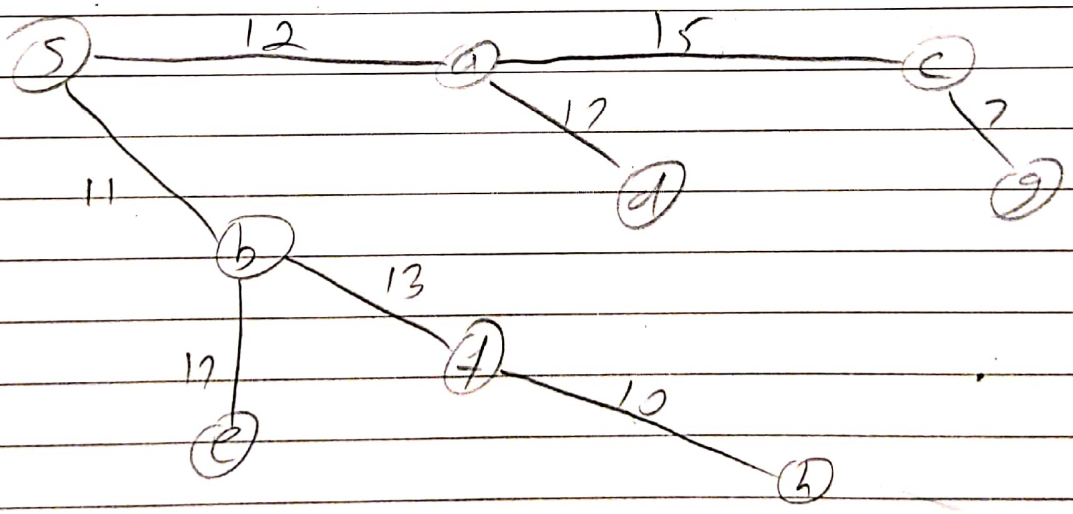
cost of going to:

- d = 29
- f = 34
- h = 54

step ⑦



step ⑧ Final graph



Q4

D^0

	a	b	c	d	e
a	0	2	∞	1	8
b	6	0	3	2	∞
c	∞	∞	0	4	∞
d	∞	∞	2	0	3
e	3	∞	∞	∞	0

c → b = ∞

c → a → b = ∞ + 2

c → d = 4 c → e = ∞

c → a → d = ∞ c → a → e ∞

D^a

	a	b	c	d	e
a	0	2	∞	1	8
b	6	0	3	2	14
c	∞	∞	0	4	∞
d	∞	∞	2	0	3
e	3	5	∞	4	0

b → c 3

b → a → c 6 + ∞

b → d 2 b → c = ∞

b → a → d ~~b → a → c~~

6 + 8 = 14

e → d → b = ∞

d → a → b = ∞

e → b = ∞ d → c = 2

e → a → b d → a = ∞ e → c = ∞

e → d = ∞ e → a → d = 3+

e → a = 3

~~e → b →~~

~~5 + 6 = 11~~ e → c ∞ a → d 1

a → b → c 2 + 3 c → b

e → c → ∞ a → e 8

c → b → a = ∞ a → b → e =

2 + 14 = 16 ✗

D^b

	a	b	c	d	e
a	0	2	5	1	8
b	6	0	3	2	14
c	∞	∞	0	4	∞
d	∞	∞	2	0	3
e	3	5	8	4	0

d → c = 2 e → d = 4

e → c = ∞ d → b ∞ c → d ∞ c → e ∞

e → b → c = 6 + 3 d → e = 5 d → a ∞ c → b ∞

d → b → e → 9 d → b → a

e → d → 4

e → b → d → 5

PAPERWORK

$b \to d = 2$ $b \to a = 6$ ✓

$b \to c$ ✗ $b \to e \to a$ 3 + ∞

$b \to c = 14$

$b \to c \to e = 3 + \infty$ $a \to b = 2$

$a \to c \to b$ 5 + ∞ ✗

$a \to d = 1$

$d \to a = \infty$ $a \to c \to d = 5 + 2$ ✓

$d \to c \to a$ $a \to e = 8$

$2 + \infty$ $a \to c \to e = 5 + \infty$

$e \to a = 3$ $d \to b = \infty$

$e \to c = \infty$ $d \to c \to b = 2 + \infty$ $d \to e = 3$

$e \to b = 5$ $d \to e \to c =$

$2 + \infty$

$a \to c$ $e \to d = 4$

$e \to$

D²

	a	b	c	d	e
a	0	2	5	1	8
b	6	0	3	2	14
c	∞	∞	0	4	∞
d	∞	∞	2	0	3
e	3	5	8	4	0

D¹

	a	b	c	d	e
a	0	2	3	1	4
b	6	0	3	2	5
c	∞	∞	0	4	7
d	∞	∞	2	0	3
e	3	5	6	4	0

$b \to e = 14$ $a \to b = 2$

$b \to d \to c$ $a \to d \to b$ $1 + \infty$

$2 + 3 = 5$ $a \to c = 5$

$b \to c = 3$ $a \to d \to c = 1 + 2 = 3$

$b \to d \to c$ $a \to e = 8$ $a \to d \to c =$

$2 + 2 = 4$ $1 + 3 = 4$

$b \to a = 6$

$b \to d \to a = 2 + \infty$

$c \to a = \infty$ $c \to d \to a = 2 + \infty$

$c \to b = \infty$

$e \to a = 3$ $c \to d \to b \to 2 + \infty = \infty$

$e \to d \to a$

$4 + \infty$ $c \to e = \infty$

$c \to d \to e = 7$

$e \to b = 5$ $4 + 3$

$e \to d \to b = 4 + \infty$ $c \to c = 8$

$e \to d \to c =$

$e \to d = 4$ $4 + 2 = 6$

e

D^e

	a	b	c	d	e
a	0	2	3	1	4
b	6	0	3	2	5
c	10	12	0	4	7
d	6	8	2	0	3
e	3	5	6	4	0

a -> b => 2

a -> d -> b => ~~4~~

b -> a -> c

b -> e -> a => 5 + 3 = 8

b -> c => 13

b -> e 5 -

a -> a -> ∞

c -> e -> a 7 + 3

d -> a => ∞

d -> e -> a =>

5 7 + 3

d -> b => ∞

d -> e -> b =>

3 + 5 = 8

e -> b => ∞

c -> e -> b = ~~4~~ 5 - 2

c -> d => 2

c -> e -> ~~4~~ 7

c -> e -> e

d -> c => 2

d -> e -> c => 3 + c

The floyd warshall Algorithm consists of three loops over all the nodes. Hence the asymptotic complexity of Floyd Warshall algorithm is $O(n^3)$. This Algorithm is best suited for dense graphs. For sparse graphs any other Algorithm is suited.

Q5 This lecture talks about 2 special cases of graphs. One with negative edges and cycles and the other without any negative edges. Algorithms were discussed for such graph traversals with Djikstra's being one of them. As an introduction to Djikstra we will first discuss it's Algorithm as discussed in the Lecture followed by the proof of it's correctness.

Algo: D(G, w, s)
initialize (G, s)
$S \leftarrow \phi$
$Q \leftarrow V[G]$
$d[s] = 0$
while $Q \neq \phi$
$u \leftarrow$ extract min (Q)
$S \leftarrow S \cup \{u\}$
for each vertex $v \in$ Adj $[u]$ {relax (u, v, w)}

S# is first initialized as 0 because there is no shortest path in the beginning.

The correctness of Djikstra is based on the two loops that is while and for each. we will first discuss the relax function and prove that it is safe which will in turn prove the correctness of the algorithm. The terminologies are

$\pi[v]$ → predecessor of v
$\delta(s, u)$ → length of a shortest path
$d[v]$: length of current shortest path from S to v.

By relaxing we mean going to other nodes on the graph and updating the distance from ∞ to a value.

The Pseudocode for relax is

```
Relax(u,v,w)
    if (d[v] > d[u] + w(u,v)){
        d[v] = d[u] + w(u,v)
        π[v] = u
    }
```

The Lemma is prove that relaxation is safe if
Lemma: Relaxation operation maintains the invariant that
$d[v] \geq \delta(s,v)$ for all $v \in V$.
proof By induction on the number of steps.

By induction:
$d[u] \geq \delta(s,u)$ By Δ inequality $\delta(s,v) \leq \delta(s,u) + \delta(u,v)$
$\therefore$
$\delta(s,v) \leq (d[u] + w(u,v) = d[v])$
hence proved that since the length of any shortest path will always be lesser than or equal to the length of the current node's shortest path from Source to destination.
Since relaxation is safe the
~~$ $ $\leftarrow$ $ u[u]~~
in the while loop will always have shortest paths in every iteration thereby maintaining the loop invariance of Djikstra's. The extract-min function is a deciding factor of the running time of Djikstra. The relaxing function takes
$O(V+E)$ time as it logically sorts all the vertices and relaxes each edge. However if extract-min is implemented via Array the running time becomes $O(V^2)$. implementing it via min heap gives $O(V \lg V + E \lg V)$. The best running time of Djikstra is achieved by Fibonacci heap.

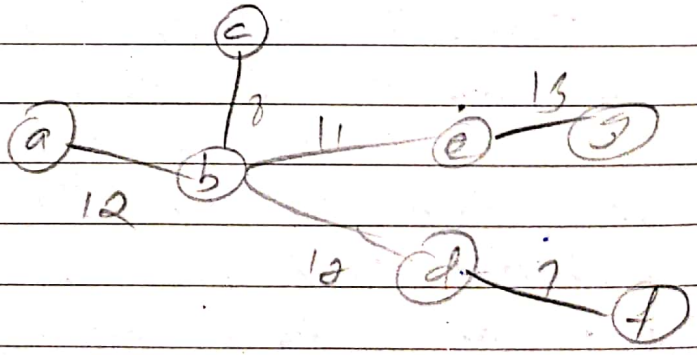
Q6 (a)

```
widest_Path ( G, S)
    priority queue temp;
    temp.push (0, source)
    widest[src] = ∞
    while (temp ≠ ϕ) {
     curr = temp.pop()
    for each v ∈ G
    dist = max (widest[v], min (widest[curr, v])

    if (dist > widest[v])
        widest[v] = dist
          temp.push (curr.next)
```

(b)

a	b	c	d	e	f	z
0	∞	∞	∞	∞	∞	∞
0	(12)	5	∞	∞	∞	∞
		8	(12)	11	∞	∞
		8		(11)	7	
						11

a → b = 12
a → b → c = 8
a → b → e = 11
a → b → d = 12
a → b → d → f = 7
a → b → c → g = 11



Q7. BFS: The complexity of BFS through Adjacency Matrix will be $O(V^2)$ and through Adjacency list is $O(V+E)$ The complexity difference occurs due to the edges being immediately available and taking proportional time to the number of vertices Hence giving us $O(V+E)$. whereas in Adjacency Matrix to determine which nodes are adjacent to a vertex we take $O(V)$ time.

DFS: Like BFS the time complexity through Adjacency Matrix is $O(V^2)$ and $O(V+E)$ through Adjacency list. This is due to the fact for each node we traverse through the adjacency list just once to visit the neighbours and sum of sizes of all nodes = E hence $O(V+E)$, while in the Matrix each row corresponds to a node and that row stores info about edges emerging from the node. Hence complexity is $O(V^2)$.

Prim's algorithm: The complexity through Adjacency Matrix is $O(V^2)$ and is $O(E \log V)$ through Adjacency list.

Kruskal's algorithm: The complexity through Adjacency list is $O(V^2 + E \log E)$. This is because it takes $O(V^2)$ to find the Edges and $O(E \log E)$ to sort them. And it's the same for Adjacency Matrix.