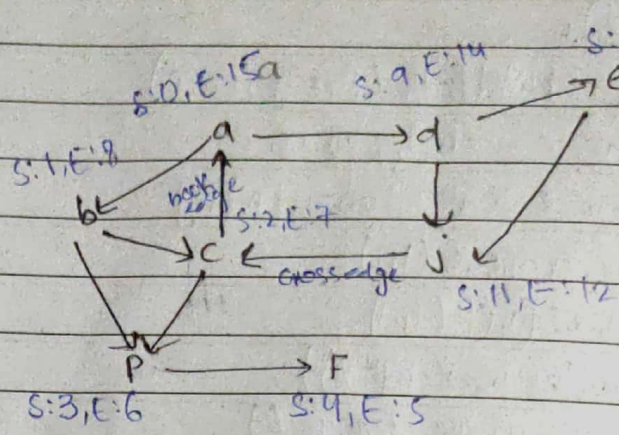
Asmad Anis
19K-0305- H

ALGO ASSIGNMENT #04

Date: 14 Dec 2021

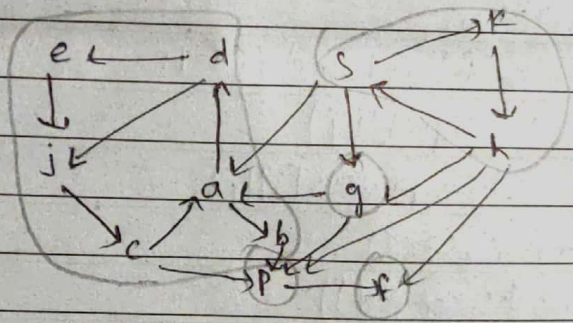
Question #1 (a)

DFS(a) → DFS(b) or DFS(d)
DFS(b) → DFS(c) or DFS(p)
DFS(d) → DFS(e) or DFS(j)
DFS(c) → DFS(p)
DFS(p) → DFS(f)
DFS(e) → DFS(j)
DFS(j) → dead
DFS(f) → dead
DFS: [f, j, e, p, c, d, b, a]



(b) cross edge [(j, c)]
Forward edge [(a, d), (a, b), (b, p), (p, f), (d, e), (d, j), (b, c), (c, p), (e, j)]
Back edge [(c, a)]

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



Sandal

Date:

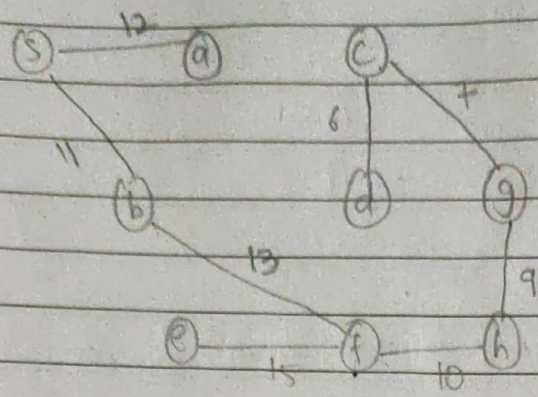
Question # 02).

```
Algorithm shortestPath (Array, n, m).
    queue. ← empty queue.
    previous ← new array[n][m]
    queue. enqueue (1,1)
    previous[1,1]=(-1,-1)
    while(!queue. isempty()):
        i,j = queue. dequeue ()
        find(queue, previous, i,j, i-1, j, n, m)
        find (queue, previous, i, j, i+1, j, n, m)
        find(queue, previous, i,j, i, j-1, n, m)
        find(queue, previous, i, j, i, j+1, n, m)

    ss ← new stack of int
    i,j = l,m
    while i != -1 and j != -1:
        ss. push(i,j)
        i,j = previous[i][j]
    dist = ss. size()
    while(!ss. isempty()):
        print(ss. pop())
    return dist

function find(queue, previous, i,j, a,b, n,m):
    if((a>=1 && a<=n) && (b>=1 and b<=m) && array[a][b] != blocked()):
        queue. enqueue(a,b)
        previous[a][b] = i,j
```

Pg No.

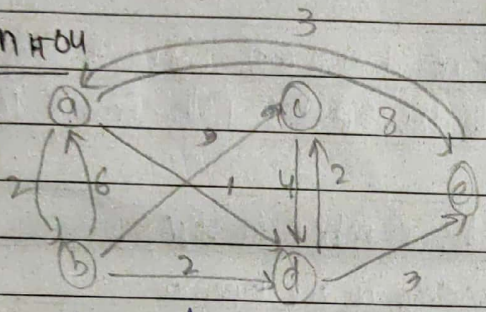
Date:

Question #03



Step#1: min cost edge from S
(S,b)
Step #2: (S,a)
Step#3: (b,f)
Step#4: (f,h)
Step#5: (h,g)
Step#6: (g,c)
Step#7: (c,d)
Step#8: (f,e)
cost = 11+12+13+15+10+6+7+9 = 83

Question #04



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

D_1 =

	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

D_2 =

	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_4 =

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

D_5 =

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

Time complexity: $O(n^3)$ due to three loops each loop has complexity of $O(n)$

Sandal

Date:

D_5 =

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

Question #05

To verify the Dijkstra's Algorithm's correctness, we need to ensure that the relaxation step is safe

Lemma: the relaxation operation maintains the invariant

$d[v] \geq \delta(s,v)$ for all $v \in V$

where $d[v]$ depicts the length of current shortest path from starting s to vertex v and $\delta(s,v)$ is the length of the shortest path from s to v.

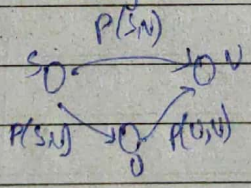
Proof: By the process of induction on the number of steps:

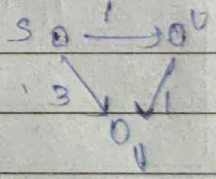
$d[v] \geq \delta(s,v)$, as per depending on our lemma.

Now, as per the triangle inequality

$\delta(s,v) \leq \delta(s,u) + s(u,v)$

Now to explain the triangle inequality, we use the example.





There seems to be have no contradiction have with a weighted digraph - As we can see from the graph, the directed path $s \rightarrow u \rightarrow v$ is the shortest path from s to v. Now this contradicts with the Δ inequality, but after the process of relaxation $\delta(s,v)$ becomes the shortest path; it is derived from $\delta(s,u) + \delta(u,v)$

Equation of Δ inequality:

$\delta(s,v) \leq \delta(s,u) + \delta(s,v)$

$\because \delta(s,u) = d[u]$, $\delta(u,v) = w(u,v)$

$\Rightarrow \delta(s,v) \leq d[u] + w(u,v)$

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

$\Rightarrow \delta(s,v) \leq d[v]$ which is in accordance with our lemma

Pg No.

Sandi

Date:

Question # 06 a):-

```
Algorithm widestPath(Graph, source):
    previous[] = array of size V
        previous = -1 // whole array to -1
    width[] = array of size V
        width = 0 // whole array to 0
    Vertices = number of vertices in graph
    queue = min heap of int pairs, 1st int is priority
    queue.push(0, source)
    width[source] = MAX
    while (queue is not empty):
        p, u = queue.pop()
        for v in neighbours of u:
            dis = max(width[v], min(width[u], distance(u, v)))
            if dis > width[v]:
                width[v] = dis
                previous[v] = u
                queue.push(dis, v)
            end if
        end for
    end while
    return width, previous
end widest_path
```

(b)

	a	b	c	d	e	f	g
previous	-1	-1	-1	-1	-1	-1	-1
width	∞	0	0	0	0	0	0

heap: [(∞, a)]

	a	b	c	d	e	f	g
previous	-1	a	a	-1	-1	-1	-1
width	∞	12	5	0	0	0	0

heap: [(5, c), (12, b)]

Sandal

Date:

	a	b	c*	d	e	f	g
previous	-1	a	a	c	-1	c	-1
width	∞	12	5	3	0	5	0

Heap: [(3,d),(5,f),(12,b)]

	a	b	c	d*	e	f	g
previous	-1	a	a	c	d	c	d
width	∞	12	5	3	3	5	3

Heap: [(3,d),(3,g),(5,f),(12,b)] No change when processing e and g

	a	b	c	d	e	f*	g
previous	-1	a	a	c	f	c	f
width	∞	12	5	3	5	5	5

Heap: [(5,e),(5,g),(12,b)]

	a	b	c	d	e*	f	g
previous	-1	a	a	e	f	c	f
width	∞	12	5	5	5	5	5

Heap: [(5,d),(5,g),(12,b)]

No change still while processing d and g.

	a	b*	c	d	e	f	g
previous	-1	a	b	b	b	c	f
width	∞	12	8	12	11	5	5

Heap: [(8,c),(11,e),(12,d)]

	a	b	c*	d	e	f	g
previous	-1	a	b	b	b	c	f
width	∞	12	8	12	11	8	5

Heap: [(8,g),(11,e),(12,d)]

	a	b	c	d	e	f*	g
previous	-1	a	b	b	b	c	f
width	∞	12	8	12	11	8	8

Heap: [(8,g),(11,c),(12,d)]

No change when processing g.

Pg No.

Date:

	a	b	c	d	e	f	g
previous	-1	a	b	b	b	e	e
width	∞	12	8	12	11	11	11

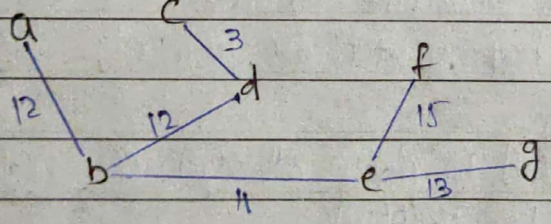
Heap: [(11, f), (11, g), (12, d)]

	a	b	c	d	e	f	g
previous	-1	a	f	b	b	e	e
width	∞	12	9	12	11	11	11

Heap: [(9, c), (11, g), (12, d)]

NB change when processing remaining

Target	path	Width
b	a→b	12
c	a→b→e→f→c	9
d	a→b→d	12
e	a→b→e	11
f	a→b→e→f	11
g	a→b→e→g	11



widest path tree

Date:

Question # 07:

BFS and DFS:

Both traversals processes a total of V vertices.
for an adjacency list, it takes O(N) time to iterate over every vertices neighbours, leading to a complexity of O(V+VN) :: where V*N is equal to the number of edges in the graph, so the overall complexity is O(V+E)
Using an adjacency matrix, neighbour are iterated over in O(V) time, leading to a time complexity of total $O(V^2)$.

Kruskal's:

It is necessary to sort all edges, which takes O(ElogE) time
In adjacency list, finding all edges takes O(E) time, so overall time complexity is O(ElogE +E) or O(ElogE)
In adjacency matrix, finding edges takes $O(V^2)$ time and overall time complexity becomes $O(V^2 + ElogE)$ in the worst taking case ($E=V^2$), this becomes $O(V^2 \log V)$

Prims:

using an adjacency list and priority queue, the algorithm runs O(ElogV), due to processing E edges and heap operation for each edge taking logV time on average.
with an adjacency matrix, it takes O(V) to process all edges of a single vertex, and for V vertices, the total time of $O(V^2)$.