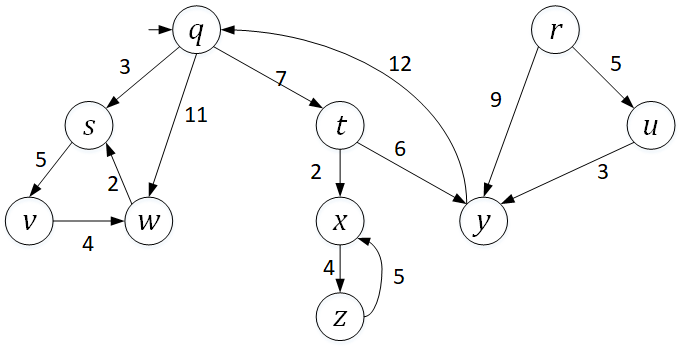
Algorithms and Data Structures

**Q1 – Q4.** For a given graph *G1=(V, E)* in the figure, perform the given tasks.

In DFS and BFS, a weight of edge is not considered and a priority for selection is given to the vertex whose alphabetic order is the lower: e.g.) s < w < y in DFS or in BFS from the starting vertex *q.*



**Q1. Breadth First Search (BFS)**

Traverse the graph *G1* from a start vertex *q* by *breadth first search (BFS)*.

* 1. List the vertices in the order of traversal.

q s t w v x y z

* 1. Give a list of the discovery edges in your BFS tree

q->s

q->t

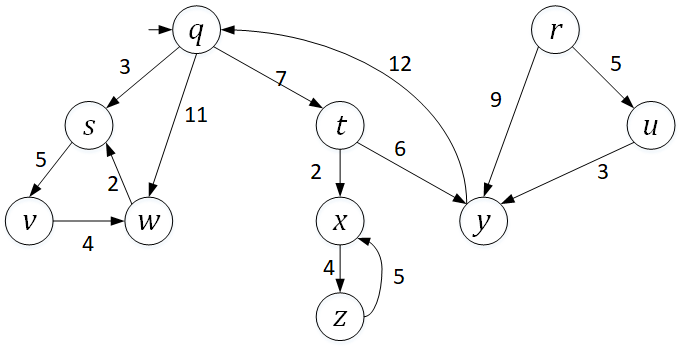
q->w

s ->v

t->x  
 t->y

x->z

* 1. Mark the BFS tree with the discovery edges in red and the cross edges in blue, respectively in the given graph.



**Q2. Depth First Search (DFS)**

Traverse the graph *G1* from a start vertex *q* by *depth first search (DFS)*.

2.1) List the vertices in the order of traversal with their start time & finish time.

vertex start finish

q 1 9

s 2 5

v 3 5

w 4 5

t 5 9

x 6 8

z 7 8

y 8 9

2.2) Give a list of the discovery edges in your DFS tree.

q->s

s->v

v->w

q->t

t->x

x->z

t->y

2.3) Give a list of back edge, forward edge and cross edge, respectively, if there were any.

Back edge:

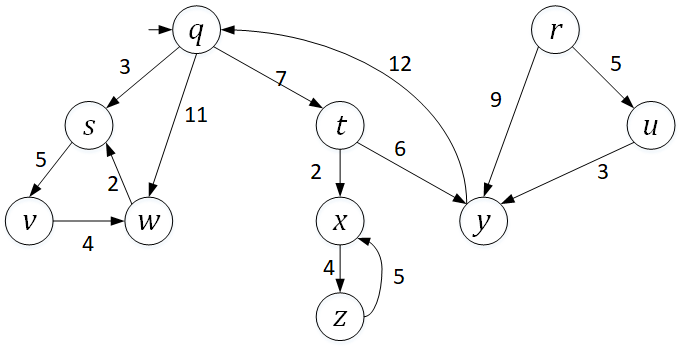
w->s  
 y->q

z -> x

Forward Edge:

r->y  
 r -> u  
 u->y

2.4) Mark the DFS tree with the discovery edges in the given graph.



**Q3. A Single-Source Shortest Path (SSSP)**

EITHER by applying *Dijkstra*’s algorithm

OR by applying *Bellman-Ford* algorithm to the directed graph G1,

find the shortest path from *q* to each vertex, respectively.

3.1) List the edges in the shortest path.

Vertex Path Distance from Source q

q 0

r INF

s q 3

t q 7

u INF

v s q 8

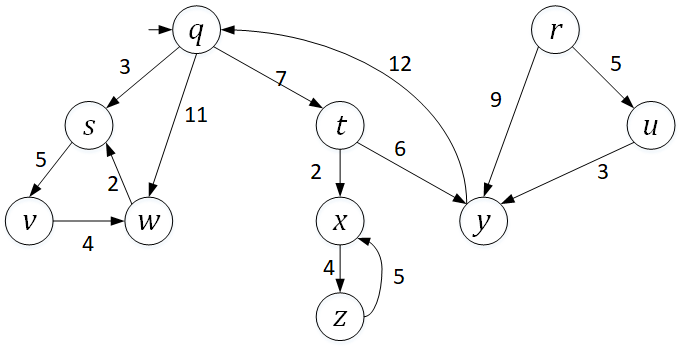
w q 11

x t q 9

y t q 13

z x t q 13

3.2) Mark the shortest path in the graph.



**Q3B.** Implementation in Python. Print the outcomes of 3.1).

Specify which algorithm you’ve applied.

Dikstra’s algorithm see q3b.py

**Q4.** **Strongly Connected Component (SCC)**

From the Depth-First Search(DFS) in Q2, showing the finishing times of the vertices,

4.1) Arrange the vertices in decreasing order of its finishing time,

w v s z x y t q u r

4.2) Draw the transposed graph *G1T* of G1.

q : y

r :

s : q w

t : q

u : r

v : s

w : q v

x : t z

y : r t u

z : x

4.3) Perform DFS on *G1T*. Show the DFS tree(s) in the G1T in the map.

r: r

u: u

q: q y t

t:

y:

x: x z

z:

s: s w v

v:

w:

4.4) Show each *SCC* of *G1*.: e.g.) SCC1 = {*q, s, t*}, SCC2 = {*x, y, z*}

q y t SCC1

x z SCC2

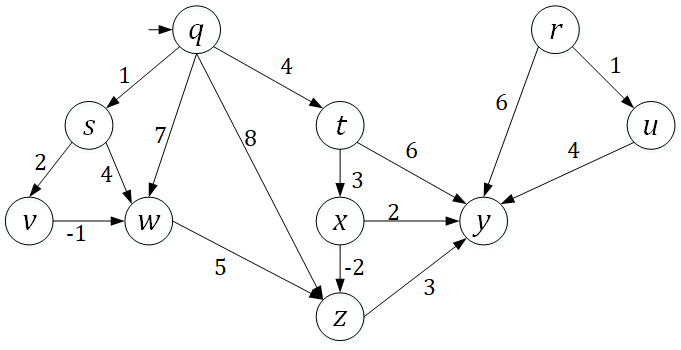
s w v SCC3

4.5) draw the acyclic *component graph* *GSCC* .

Assume that the loop of DFS considers vertices in alphabetical order.

**Q4B.** Implementation in Python. Print the outcomes of 4.1) and 4.4)

**Q5.** In the given modified Directed Acyclic Graph (DAG) G2,



**Q5. Single Source Shortest Path in the DAG**

5.1) Sort the vertices in the ***topological order*** starting from *q* and give its list.

q t x s v w z y

5.2) Redraw the graph by arranging the vertices in the sorted order.

5.3) Find the shortest path from a vertex *q* to each vertex. You have to show the proper

steps of edge relaxations, updating a key, D[*v*] of each vertex *v, v* ∈ V(G2).

Vertex Path Distance from Source q

q 0

r INF

s q 1

t q 4

u INF

v s q 3

w v s q 2

x t q 7

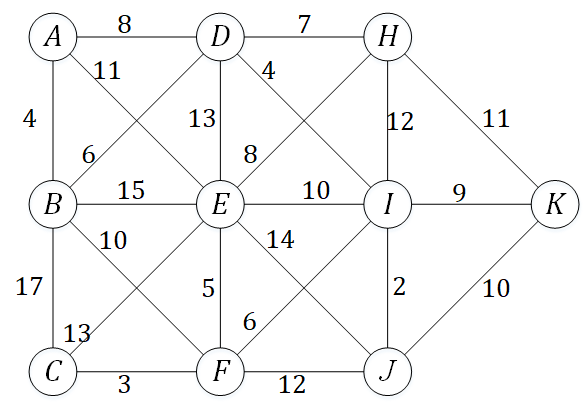
y z x t q 8

z x t q 5

**Q5B.** Implementation in Python. Print the outcomes of 5.1) and 5.3): the vertices in the topological order and the list of edges in the shortest path from *q*, respectively.

See program q5b.py

**Q6.**  In the given undirected graph G3 below:



**Q6. Minimum Spanning Tree**

EITHER by applying *Prim*’s algorithm

OR by applying Kruskal’s algorithm

find the Minimum Spanning Tree (MST) of G3.

6.1) List the edges in the MST.

Edge Weight

a - b 4

f - c 3

b - d 6

f - e 5

i - f 6

d - h 7

d - i 4

i - j 2

i - k 9

6.2) Mark the edges of the MST in the graph. (See above graph)

**Q6B.** Implementation in Python. Print the outcomes of 6.1) or 6.2).

Specify which algorithm you’ve applied.

Prims algorithm

See program q6b.py