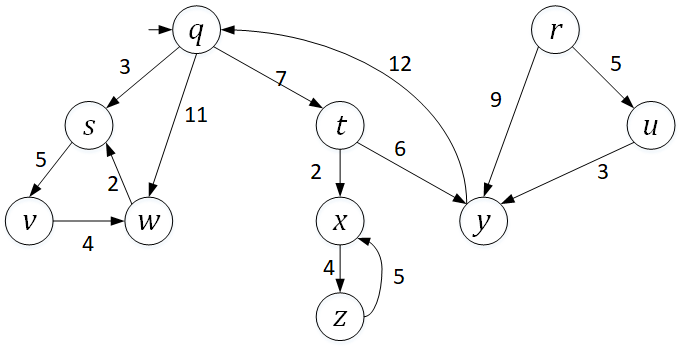
CSci 242: Algorithms and Data Structures  **Fall, 2019** Instructor: Dr. M..E. Kim Date: November 26 (Tue.), 2019

**Due: by the end of day, December 10th (Tue.)**

**Home Assignment 8: Graph Algorithms (200 + 100 optional)282/200**

**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. [25] Breadth First Search (BFS)**

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

1.1) [10] List the vertices in the order of traversal

q s t w v x y z

1.2) [9] Give a list of the discovery edges in your DFS tree

Q:S

Q:T

Q:W

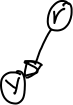
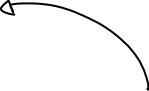
S:V

T:X  
 T:Y

X:Z

(q, s), (q, t), (q, w), (s, v), (t, x), (t, y), (x, z), (r, u)

1.3) [5] Mark the DFS tree with the discovery edges in red and the cross edges in blue, respectively in the given graph.



**Q2. [35] Depth First Search (DFS)**

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

2.1) [8] 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 |

(q, 1/16), (s, 2/7), (v, 3/6), (w, 4/5), (t, 8/15), (x, 9/12), (z, 10/11), (y, 13/14),

(r, 17/20), (u, 18/19)  
2.2) [10] 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) [8] 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:

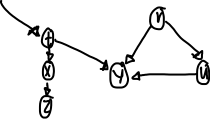
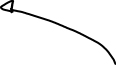
R:Y  
 R : U  
 U : Y

Back edge: (w, s), (z, x), (y, q)

Forward edge: (q, w),

Cross edge: (r, y), (u, y)

2.4) [5] Mark the DFS tree with the discovery edges in the given graph.



**Q3. [25] A Single-Source Shortest Path (SSSP)**

EITHER by applying *Dijkstra*’s algorithm

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

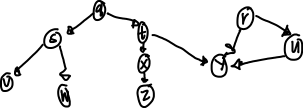
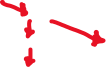
find the shortest path from *q* to each vertex, respectively.

3.1) [15] 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 | sq | 8 |
| w | q | 11 |
| x | tq | 9 |
| y | tq | 13 |
| z | xtq | 13 |

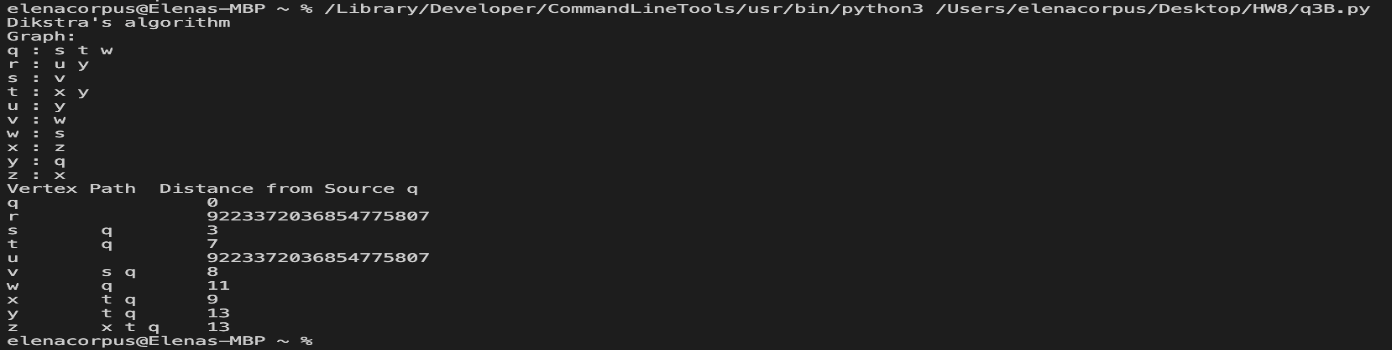
(q, s: 3), (s, v: 5), (q, w: 11), (q, t: 7), (t, x: 2), (x, z: 4), (t, y: 6)

3.2\_ [10] Mark the shortest path in the graph.



**Q3B. [25, optional]** Implementation in Python/Java. Print the outcomes of 3.1).

Specify which algorithm you’ve applied.



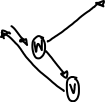
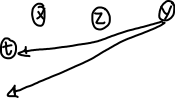
**Q4.** **[32]** **Strongly Connected Component (SCC)**

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

4.1) [10] Arrange the vertices in decreasing order of its finishing time,

w v s z x y t q u r

4.2) [3] Draw the transposed graph *G1T* of G1.



4.3) [7] 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:

(r, 1/2). (u, 3/4), (q, 5/10), (y, 6/9), (t, 7/8). (x, 11/14), (z, 12/13).

(s, 15/20 ), (w, 16/19) (v, 17/18)

4.4) [8] 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

SCCs are shown in the green circles.

SCCs: {r}, {u}, {q, t, y}, {x, z}, {s, w, v)

4.5) [4] draw the acyclic *component graph* *GSCC* .

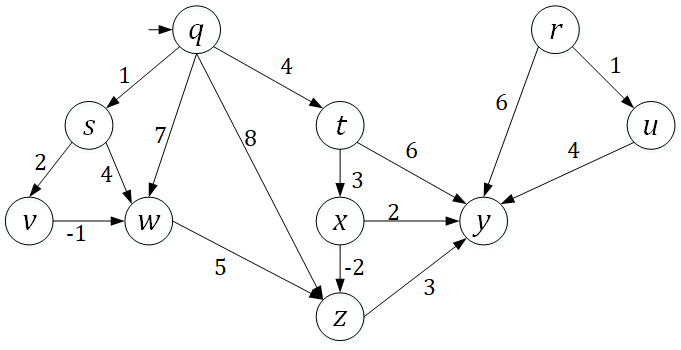


Assume that the loop of DFS considers vertices in alphabetical order.

**Q4B. [25, optional]** Implementation in Python/Java. Print the outcomes of 4.1) and 4.4)



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

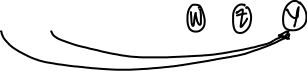


**Q5. [40] Single Source Shortest Path in the DAG**

5.1) [15] Sort the vertices in the ***topological order*** starting from *q* and give its list.

q t x s v w z y

5.2) [10] Redraw the graph by arranging the vertices in the sorted order.



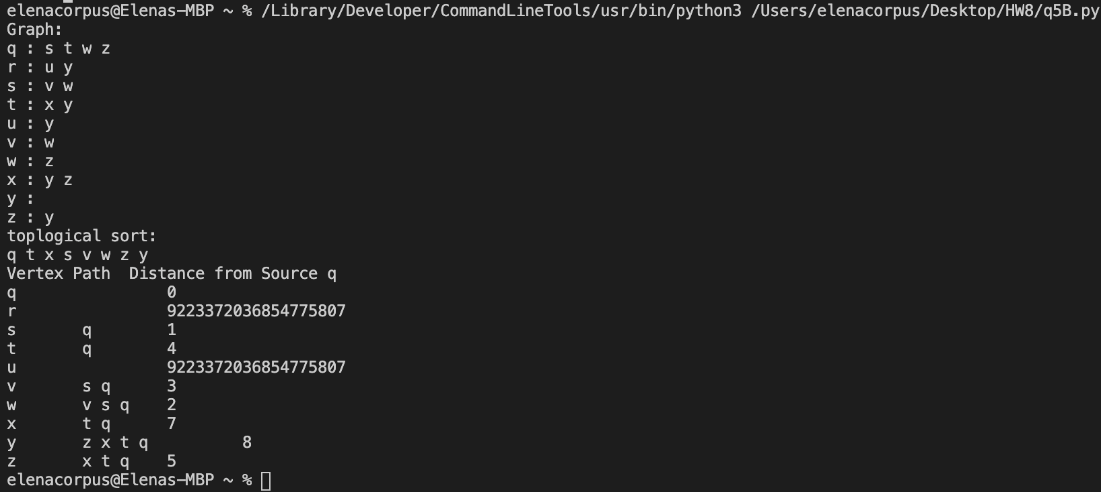
5.3) [15] Find the shortest path from a vertex *q* to each vertex. You have to show the proper

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

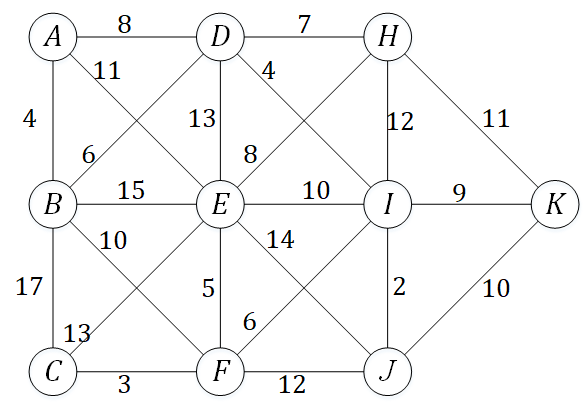


|  |  |  |
| --- | --- | --- |
| Vertex | Path | Distance from the Source Q |
| q |  | 0 |
| r |  | INF |
| s | q | 1 |
| t | q | 4 |
| u |  | INF |
| v | sq | 3 |
| w | vsq | 2 |
| x | tq | 7 |
| y | zxtq | 8 |
| z | xtq | 5 |
|  |  |  |

**Q5B. [25, optional]** Implementation in Python/Java. 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.



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





**Q6. [30] 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) [20] 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) [10] Mark the edges of the MST in the graph.

**Q6B. [25, optional]** Implementation in Python/Java. Print the outcomes of 6.1) or 6.2).

Specify which algorithm you’ve applied.

