Name: maahir kalbay (ert547)

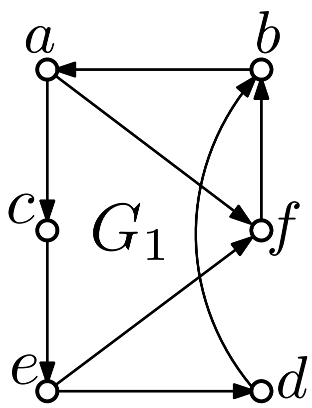
CS2123 Data Structures - Fall 2020

Assignment 5: Graph Algorithms

Due 12/3/20 by 11:59pm

1. DFS and BFS (6 points)

For the following problems, assume that vertices are ordered alphabetically in the adjacency lists (thus you will visit adjacent vertices in alphabetical order).



(1) (2 points) Execute a Breadth-First Search on the graph G_1, starting on vertex a. Specifiy the visit times for each node of the graph.

(2) (2 points) Execute a Depth-First Search on the graph G_1, starting on vertex a. Specifiy the visit and finish times for each node of the graph

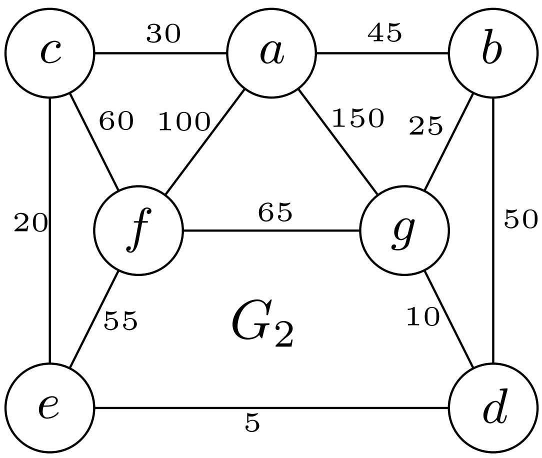
(a) (1 point) For each edge in your graph identify whether it is a tree edge, back edge, forward edge, or a cross edge.

(b) (1 point) Which edges would you remove to make your graph into a DAG? (hint: use your edge classification to justify your choice)

2. Prim's Algorithm (3 points)

(4 points) Run Prim's algorithm on the graph G_2, with start vertex a. Assume that vertices are ordered alphabetically.

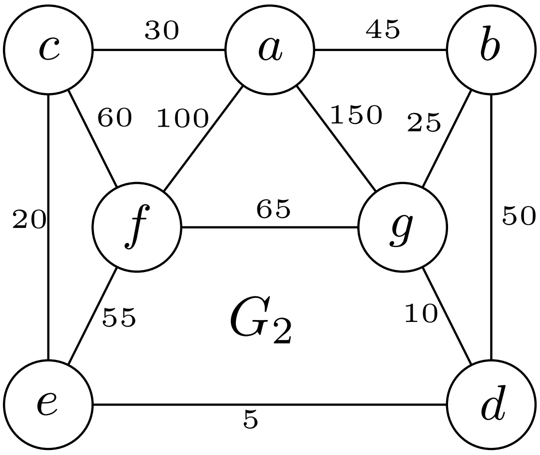
For each step of the algorithm specify the current vertex weights (you can use a table to represent this data). Draw the minimum spanning tree the algorithm finds.



RM	a	b	c	d	e	f	g
–	0	∞	∞	∞	∞	∞	∞
a	–	45	30	∞	∞	100	150
c	–	45	–	∞	20	60	150
b	–	45	–	50	–	60	25
d	–	45	–	50	–	60	10
e	–	45	–	5	–	55	–
f	–	45	–	–	–	–	–
g	–	25	–	–	–	–	–

3. Dijkstra's Algorithm (3 points)

Run Dijkstra's algorithm on the graph G_1 to compute the shortest paths from a to all of the other vertices.



RM	a	b	c	d	e	f	g
–	0	∞	∞	∞	∞	∞	∞
a	–	45	30	∞	∞	100	150
c	–	-	–	∞	50	90	150
b	–	–	–	95	-	-	70
e	–	-	–	55	-	–	–
f	–	–	–	–	-	-	-
g	–	–	–	-	–	-	-
d	–	–	–	–	–	–	–

4. Coding Graphs (20 pts)

For all of these functions you'll be passed an char[][] array which represents a maze (the maze is square and you will also be passed the side length as an int). The symbol 'X' represents a space that is impassable. The symbol ' ' represents a space that is passable. The symbols 'S' and 'F' represent the starting point and ending point of the maze respectively. You may only travel up, down, left, and right (i.e, no diagonals).

You can create a graph to represent the maze where nodes/vertices represent locations in the char[][] array maze (e.g, the pair (i, j) represents the $maze[i][j]$ location) .

Here is a short description of the functions you should complete:

hasPath (10pts): Detect whether there is a path from 'S' to 'F'. Return 1 if a path exists and -1 otherwise.

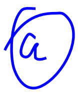
findNearestFinish (5pts): The maze contains one 'S' and multiple 'F's (1 or more). Find the length of the shortest path to any 'F' from 'S' and return it. If no 'F' is reachable return -1.

findLongestSimplePath (5pts): The maze contains one 'S' and one 'F'. Find the length of the longest simple path to 'F' from 'S' and return it (simple paths are those that do not visit any location twice). If 'F' is not reachable return -1. Solving this problem will involve recursively checking all possible paths from ‘S’ to ‘F’.

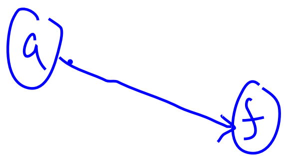
To receive full credit, the total run time of your program should be $<$ 20 seconds on the UTSA CS lab machines. Note your run time will probably be much faster than that if all of your functions are well chosen and correctly implemented.

1. (1)

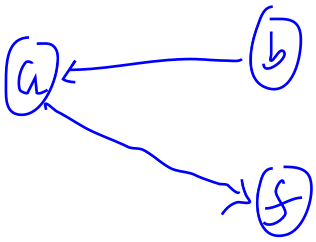
Step 1



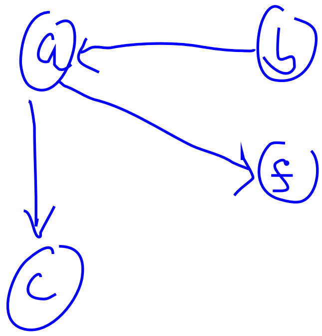
Step 2



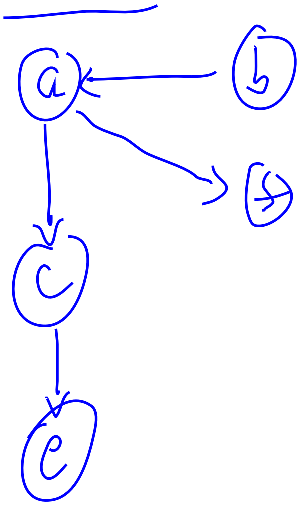
Step 3



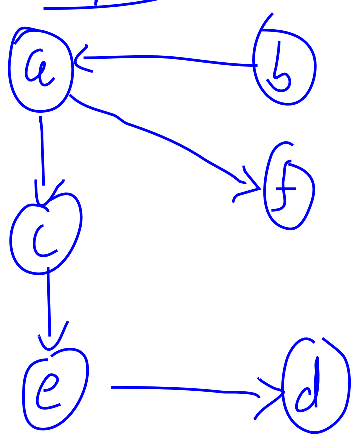
Step 4



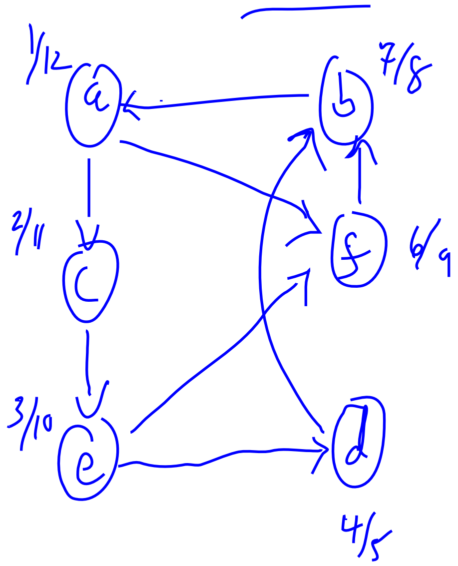
Step 5



Step 6

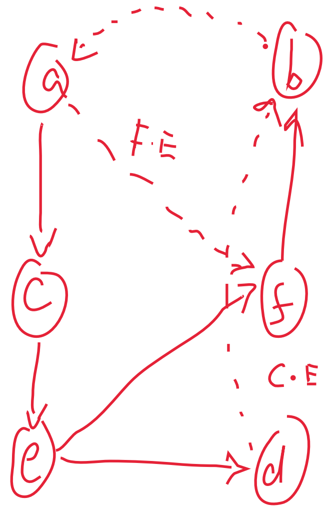


(2) D.F.S



=>

back edge



a) Tree edges:

<ac>, <ce>, <ed>, <ef>, <fb>

forward edge:

<af>

back edge:

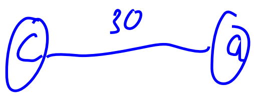
<ba>

Cross edge

<db>

b) <fb>, <db> Remove to make DAG

2. **Step 1:** Start at vertex a find all vertices conected to a

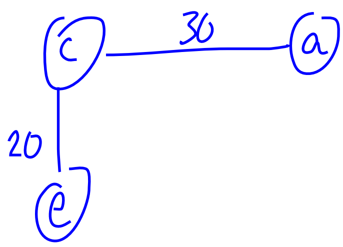


Edge	cost
a - c	30 → minimum cost
a - b	45
a - f	100
a - g	150

Step 2: All vertices connected to a and c

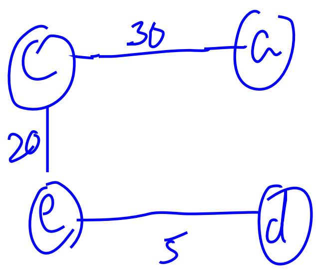
Edge	cost
a - b	45
a - f	100
a - g	150
c - f	60
c - e	20 → minimum cost

Step 3: All vertices connected to a, c and e



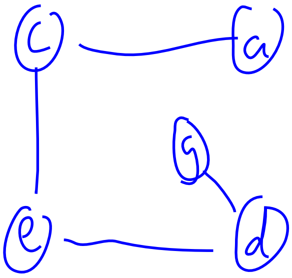
Edge	cost
a - b	45
a - f	100
a - g	150
c - f	60
e - f	55
e - d	5 → minimum cost

Step 4: All vertices connected to a, c, e and d



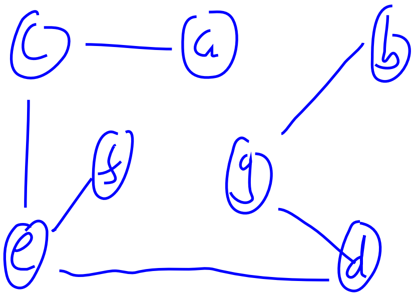
Edge	cost
a - b	45
a - f	100
a - g	150
c - f	60
e - f	55
d - g	10 → minimum cost
d - b	50

Step 5: All vertices connected to a, b, e, d and g



Edges	cost
a-b	45
g-f	100
a-g	150
c-f	60
e-f	55
g-f	65
g-b	25 → minimum cost
d-b	50

Step 6: All vertices connected to a, b, c, e, d, g and b



Edges	cost
a-b	45 → min but in loop.
a-f	100
a-g	150
c-f	60
e-f	55 → minimum cost
g-f	65
d-b	50

Which is the required MST.