Assignment #5

Due on Thursday 3/14/2019

1. *(20 points)* For the graph shown in the figure below.

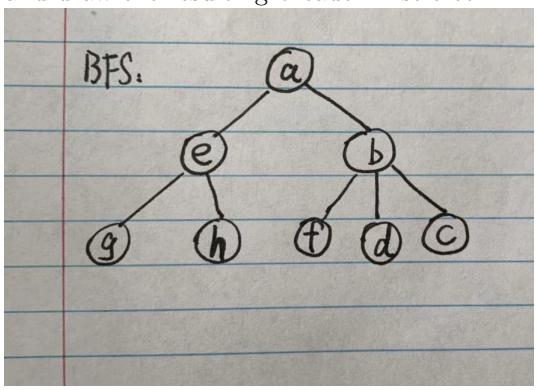
   (a) Classify the graph as either a *directed* or *undirected* graph.
   Answer: the graph is a undirected graph.

   (b) Represent the graph using an adjacency list. Assume that adjacent vertices in each neighbors list are ordered alphabetically.
   Answer: a $\rightarrow b \rightarrow e$
   $b \rightarrow a \rightarrow c \rightarrow d \rightarrow e \rightarrow f$
   $c \rightarrow b \rightarrow g \rightarrow h$
   $d \rightarrow b \rightarrow f$
   $e \rightarrow a \rightarrow b \rightarrow f \rightarrow g \rightarrow h$
   $f \rightarrow b \rightarrow d \rightarrow e$
   $g \rightarrow c \rightarrow e \rightarrow h$
   $h \rightarrow c \rightarrow e \rightarrow g$

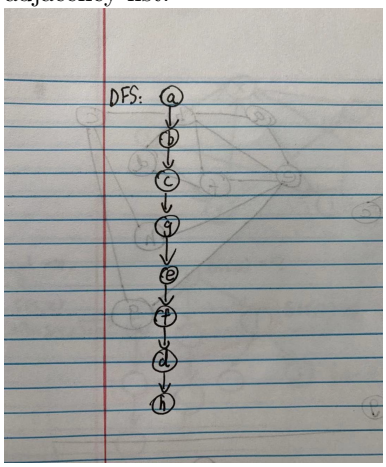
   (c) Represent the graph using an adjacency matrix.

	a	b	c	d	e	f	g	h
a	0	1	0	0	1	0	0	0
b	1	0	1	1	1	1	0	0
c	0	1	0	0	0	0	1	1
d	0	1	0	0	0	1	0	0
e	1	1	0	0	0	1	1	1
f	0	1	0	1	1	0	0	0
g	0	0	1	0	1	0	0	1
h	0	0	1	0	1	0	1	0

   (d) Perform a breadth-first search (BFS) traversal on the graph starts at node a and draw the resulting breadth-first tree.



(e) Perform a depth-first search (DFS) traversal on the graph starts at node a and draw the resulting depth-first tree. Follow the same order that you have in your adjacency list.



2. Show the optimality of Prim's minimum-spanning tree algorithm by relating it to the generic proof given in class. In other words, define the cut in each step and show how this cut respects the partial answer that was found so far.
   Answer:
   In regard to the generic proof, we first define **A** as a subset of some minimum spanning tree for a graph **G**. And we have a cut (S, V-S) which respect **A**. Then we would pick the light edge and add it to the subset **A**. Through this process, we managed to maintain **A** as the minimum-spanning tree. That is how we prove the optimality of generic proof. For Prim's algorithm, it first set the key of each vertex to ∞ except for the root r, set the parent of each vertex to NIL, and initialize the min-priority queue **Q** to contain all the vertices. After this, the algorithm pick the source node and update its adjacent nodes' key using the edge weight. Then it select the node with least key in set **Q** to be the next node in the minimum-spanning tree. First we can assume there is a subset **A** which only include source node. In this process, we can assume there is a cut between the source node and the rest graph. The Prim's algorithm essentially pick the node connecting the light edge and add it to the set **A**. We can know that **A** is always a subset of **Q**. The **A** is becoming larger and larger and eventually it would be the same set as **Q**. And we can know that the way set **A** grow is same as the generic MST. Since generic MST is optimal, Prim's MST is also optimal

3. Repeat the previous question for Kruskal's minimum spanning tree algorithm.
   Answer:
   Kruskal's MST algorithm first initializes the set A to the empty set and

create $|V|$ trees, one containing each vertex. Then, it sorts all edges by weight from small to large. After that, starting with the least weight edge, as long as two vertices connecting to that edge are not in the same set, it combine two vertices into one set and add that edge to **A**. We know that at first each vetex is a set. For the process of combining two sets, we can imagine there is a cut respect those two sets. And among all the crossing edges, we pick the light edge to connect these two sets and we add that light edge into the minimum spanning tree. This process is the same as generic MST. Through this process, the Kruskal's algorithm can maintain the minimum spanning tree. Therefore, we can prove that Kruskal's algorithm is optimal.

4. For the graph given below, apply the Kruskal's minimum spanning tree algorithm. Explain the steps in which the algorithm runs by ordering the edges in the order they are inspected and comment on each step whether that edge was added to the MST or not and why.

   Answer: notation: edge bc $\Rightarrow$ e(b,c), vertex a $\Rightarrow$ $v(a)$

1. sort all edges by weight from small to large
2. create a set for each vertex
3. pick e(b,c) because it has the least weight and check v(b) and v(c) are in the same set
4. union v(b) and v(c) becuase they are not in the same set, now v(b) and v(c) are in the same set
5. pick the next least weighted edge e(d,f), and check if v(d) and v(f) are in same set.
6. union v(d) and v(f) because they are not in the same set, now v(d) and v(f) are in the same set
7. pick e(a,b) and union v(a) and v(b), v(a) and v(b) and v(c) are in the same set
8. pick e(e,h) and union v(e) and v(h), v(e) and v(h) are in the same set
9. pick e(c,g) and union v(c) and v(g), v(a) and v(b) and v(c) and v(g) are in the same set
10. pick e(g,h) and union v(g) and v(h), v(a) and v(b) and v(c) and v(g) and v(e) and v(h) are in the same set
11. skip e(e,g) because v(e) and v(g) are in the same set
12. skip e(a,e) because v(a) and v(e) are in the same set
13. skip e(b,e) because v(b) and v(e) are in the same set
14. pick e(b,d) and union v(b) and v(d), v(a) and v(b) and v(c) and v(g) and v(e) and v(h) and v(d) and v(f) are in the same set
15 because the tree has all the vertex, we end the program