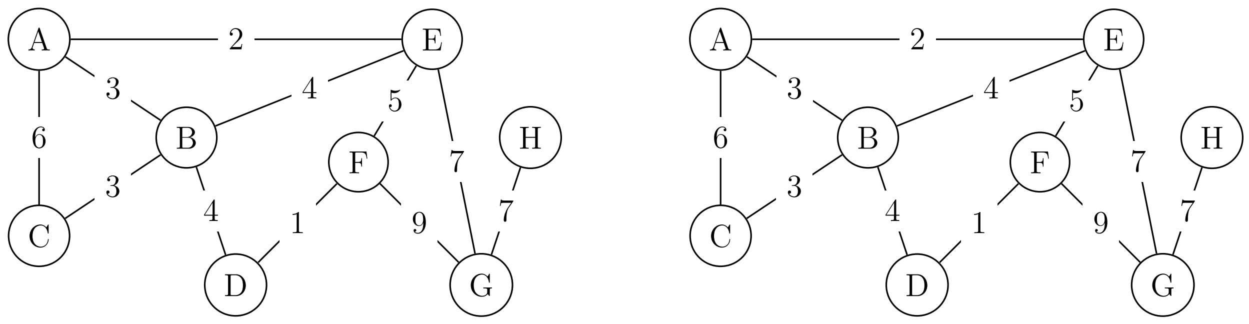
CSE 332 Winter 2020 **Quiz 7** Name: Evil Kevin
UW NetID: evilkevin

1 Minimum Spanning Tree

Consider this undirected graph, G, with 8 vertices and 11 weighted edges, for problems (a), (b), and (d). Denote each edge with alphabetical overbar notation $\overline{\text{AB}}$, which represents the edge from A to B. For your convenience, the graph is printed twice.



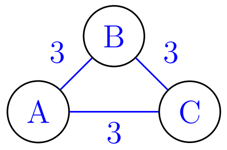
(a) What are the edges on the cut between ABCE and DFGH? You may not need all blanks.

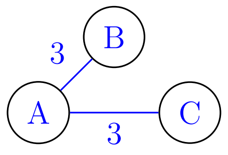
$\overline{\text{BD}}$ $\overline{\text{EF}}$ $\overline{\text{EG}}$ ___ ___ ___ ___ ___

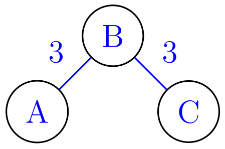
(b) What is the minimum edge on the cut between ABE and the rest of G? $\overline{\text{BC}}$

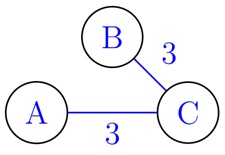
(c) **True** / [✓ **False:**] Given any undirected graph, G, G will always have a unique MST.

For example, this graph has more than one MST:

G:


1st MST:


2nd MST:


3rd MST:


Note that, if all edge weights in G are distinct, MST of G will be unique by the cut property.

(d) List the edges in the order that they're added to the MST by each algorithm. Assume ties are broken in alphabetical order (i.e. the edge $\overline{\text{BD}}$ would be considered before $\overline{\text{BE}}$). You may not need all blanks.

Prim's algorithm from A: $\overline{\text{AE}}$ $\overline{\text{AB}}$ $\overline{\text{BC}}$ $\overline{\text{BD}}$ $\overline{\text{DF}}$ $\overline{\text{EG}}$ $\overline{\text{GH}}$ ___ ___

Using the cut property approach, the order of Prim's algorithm starting from A comes from these steps:

1) cut between A and BCDEFGH: $(\overline{\text{AB}}, 3)$ $(\overline{\text{AC}}, 6)$ $(\overline{\text{AE}}, 2)$
choose $\overline{\text{AE}}$ (Now, A and E are in the same component.)

Will you want to pick up your worksheet later? Circle one: Yes / No
How confident do you feel with the material this week? Circle one: 1 / 2 / 3 / 4

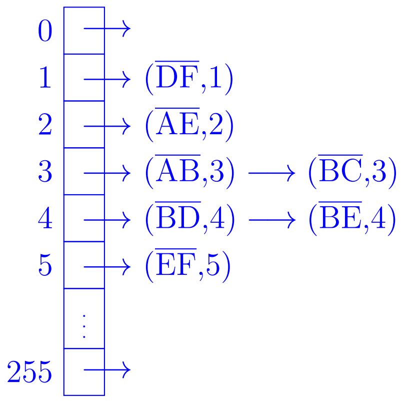
2) cut between AE and BCDFGH: $(\overline{AB}, 3)$ $(\overline{AC}, 6)$ $(\overline{BE}, 4)$ $(\overline{EF}, 5)$ $(\overline{EG}, 7)$
   choose $\overline{AB}$
3) cut between ABE and CDFGH: $(\overline{AC}, 6)$ $(\overline{BC}, 3)$ $(\overline{BD}, 4)$ $(\overline{EF}, 5)$ $(\overline{EG}, 7)$
   choose $\overline{BC}$
4) cut between ABCE and DFGH: $(\overline{BD}, 4)$ $(\overline{EF}, 5)$ $(\overline{EG}, 7)$
   choose $\overline{BD}$
5) cut between ABCDE and FGH: $(\overline{DF}, 1)$ $(\overline{EF}, 5)$ $(\overline{EG}, 7)$
   choose $\overline{DF}$
6) cut between ABCDEF and GH: $(\overline{EG}, 7)$ $(\overline{FG}, 9)$
   choose $\overline{EG}$
7) cut between ABCDEFG and H: $(\overline{GH}, 7)$
   choose $\overline{GH}$

Kruskal's algorithm: $\overline{DF}$ $\overline{AE}$ $\overline{AB}$ $\overline{BC}$ $\overline{BD}$ $\overline{EG}$ $\overline{GH}$ ___ ___

(e) We are trying to find the MST of a graph where the edge weights only range between 0 and 255. One TA suggests that it is possible to find the MST in a time faster than $O(|E|log|E|)$. Is this true? If so, briefly explain how this can be done. If not, briefly explain why.

It is possible by modifying Kruskal's algorithm! In Kruskal's, sorting edges takes the most time, $O(|E|log|E|)$. Since the edge weights only range between 0 and 255, we can have a hash table with 256 buckets to store edges. Edge's `hashCode` will just be the weight itself. By using this approach, edge weights are sorted naturally in $O(|E|)$. We can consider edges bucket by bucket starting at bucket zero and so on.

For example, the hash table could look something like:



Will you want to pick up your worksheet later? Circle one: Yes / No
How confident do you feel with the material this week? Circle one: 1 / 2 / 3 / 4

(f) **(Optional)** Give a scenario where Prim's algorithm is more preferred than Kruskal's algorithm.

Prim's algorithm runs in $O(|V|log|V| + |E|log|V|)$ and Kruskal's algorithm runs in $O(|E|log|E| + E \cdot f(|V|) + V \cdot g(|V|))$ where $f(|V|)$ is the runtime of `isConnected` and $g(|V|)$ is the runtime of `connect` from your choice of Union Find. For example, weighted quick-union by size and weighted quick-union by height have $O(log|V|)$ runtime for both `connect` and `isConnected`. Kruskal's will run slower in a dense graph where number of edges is very large due to the sorting. In that case, Prim's is more preferred than Kruskal's.

Note: the optimal union find with path compression would run very fast (in amortized $O(\alpha(n))$ where $\alpha(n) < 5$ for any value of n that can be written in this physical universe). More information in Wikipedia.

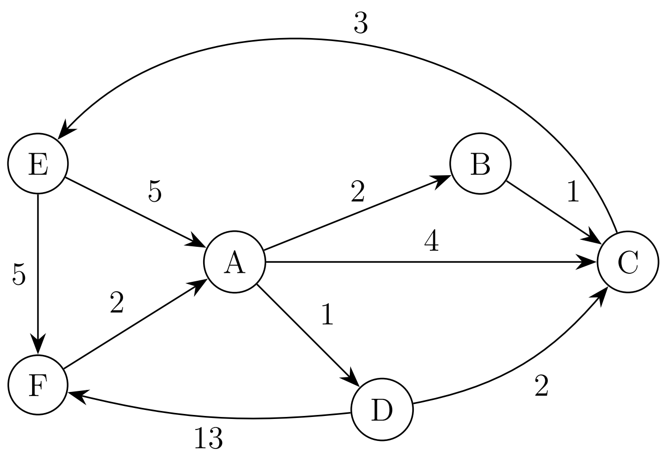
Pseudocode for Kruskal's algorithm:

```
Kruskal(Graph G):
    initialize MST to be an empty edge list
    sort G.edges() by weights              \\ O(|E|log|E|)
    for each edge e(u, v, w) in G.edges():    \\ loop for |E| times
        if isConnected(u, v) is true:      \\ runs in f(|V|)
            continue;    \\ because connecting u and v will cause a cycle
        \\ at this point, u and v are in different components
        connect(u, v)    \\ runs in g(|V|); called at most |V|-1 times
        MST.add(e)
    return MST
```

Note that, if `MST.size() == G.vertices().size() - 1`, in other word, number of edges in the MST equals to $|V| - 1$, we can also break the loop because the MST is sucessfully built.

2 Single-Source & Single-Pair Shortest Path

(a) For the graph below, run Dijkstra's algorithm to find **single-source** shortest paths from A to every other vertices. You must show your work at every step and give the vertex order visited by Dijkstra's algorithm, assuming that we always visit alphabetically earlier vertex first if there are multiple valid choices. The first step is provided for you.



Fringe:

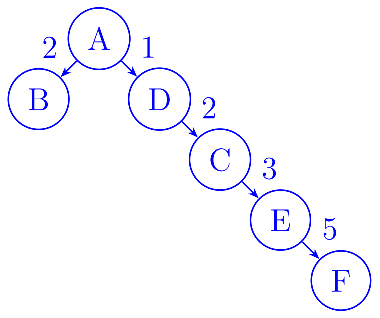
Vertex	Priority	Removed?
A	0	Yes
B	~~∞~~ 2	Yes
C	~~∞~~ ~~4~~ 3	Yes
D	~~∞~~ 1	Yes
E	~~∞~~ 6	Yes
F	~~∞~~ ~~14~~ 11	Yes

Vertex	distTo	edgeToVertex
A	0	–
B	~~∞~~ 2	$\overline{AB}$
C	~~∞~~ ~~4~~ 3	~~$\overline{AC}$~~ $\overline{DC}$
D	~~∞~~ 1	$\overline{AD}$
E	~~∞~~ 6	$\overline{CE}$
F	~~∞~~ ~~14~~ 11	~~$\overline{DF}$~~ $\overline{EF}$

Visiting order:

A D B C E F ____

(b) **(Optional)** Draw the shortest paths tree (SPT) resulting from the shortest paths found by Dijkstra's algorithm in problem 2(a).



(c) Using the graph from problem 2(a), suppose we are trying to find a **single-pair** shortest path from A to F. Specify your heuristic function for A* Search by **circling any possible numbers for** $h(E)$ that returns the **wrong shortest paths tree**.

$h(A) = 0$
$h(B) = 10$
$h(C) = 9$
$h(D) = 15$
$h(E) = -3$ 0 5 9 [✓ 11] [✓ 14]
$h(F) = 0$

Will you want to pick up your worksheet later? Circle one: Yes / No
How confident do you feel with the material this week? Circle one: 1 / 2 / 3 / 4

Consider this state after exploring A, B and C respectively:

Fringe:

Vertex	Priority	Removed?
A	0	Yes
B	~~∞~~ 12	Yes
C	~~∞~~ ~~13~~ 12	Yes
D	~~∞~~ 16	
E	~~∞~~ $6 + h(E)$	
F	∞	

Vertex	distTo	edgeToVertex
A	0	–
B	~~∞~~ 2	$\overline{AB}$
C	~~∞~~ ~~4~~ 3	~~$\overline{AC}$~~ $\overline{BC}$
D	~~∞~~ 1	$\overline{AD}$
E	~~∞~~ 6	$\overline{CE}$
F	∞	

We know that we have to go to F from E to make the shortest path. Then, the priority value of E must be lower than D at this point, so that we can pop E out from the fringe before D. That means, to make the correct shortest paths tree, we need to have $6 + h(E) < 16$ or $h(E) < 10$. Then, any value of $h(E) > 10$ will make A* Search resulting in the wrong shortest paths tree.

Will you want to pick up your worksheet later? Circle one: Yes / No
How confident do you feel with the material this week? Circle one: 1 / 2 / 3 / 4