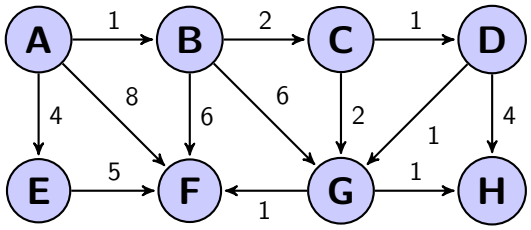
This homework is due by the start of class on Friday, February 26th. You must submit the homework via the course page on T-Square, and you may hand in a printed version with any graph drawings, if you choose not to include them in the online copy.

Homework Policies:

- Your work will be graded on correctness and clarity. Write complete and precise answers.
- You may collaborate with up to *three* other classmates on this problem set. However, you must *write your own solutions* in your own words and *list your collaborators*.
- You may portions of our two textbooks that we have covered. The purpose of these homeworks is NOT only to teach you the algorithms that we cover, but also to teach you how to problem-solve. Therefore using *ANY* outside resources like the internet, previous students etc. is *not* allowed!
- **Optional** questions are for you to practice and learn the basics before you advance. They may ask you to follow the steps of an algorithm covered in class, or review material from an earlier class. Solutions to these problems will not be graded, but you must be are able to do them, and you are responsible for material that is addressed by these optional questions.
- **Basic** questions *must be solved and written up alone*; you may not collaborate with others. These will help you build the framework needed to solve the harder questions. Please do these first, before meeting in groups, for both you and your group's benefit.
- **Group** questions may be solved in collaboration with up to ***three*** other classmates. However, you must still *write up your own solutions* alone and in your own words. These will form the bulk of your homework questions.
- **Bonus** questions are like group questions, but they will be more challenging and will be graded more rigorously. Bonus points are tallied separately from your normal points, and will be applied to your grade *after* the curve. Bonus points are worth more than normal points, and can have a significant positive affect on your grade. Most importantly, bonus problems are fun!

1. (optional) Suppose Dijkstra's algorithm is run on the following graph, starting at node A.



   (a) Draw a table showing the intermediate distance values of all the nodes at each iteration of the algorithm.

Solution:

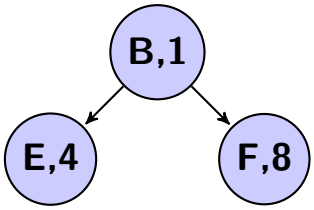
A	B	C	D	E	F	G	H
0	∞	∞	∞	∞	∞	∞	∞
0	1	∞	∞	4	8	∞	∞
0	**1**	3	∞	4	7	7	∞
0	1	**3**	4	4	7	5	∞
0	1	3	4	**4**	7	5	∞
0	1	3	**4**	4	7	5	8
0	1	3	4	4	6	**5**	6
0	1	3	4	4	**6**	5	6
0	1	3	4	4	6	5	**6**

(b) Show the min-heap (or priority heap) at each iteration of the algorithm.

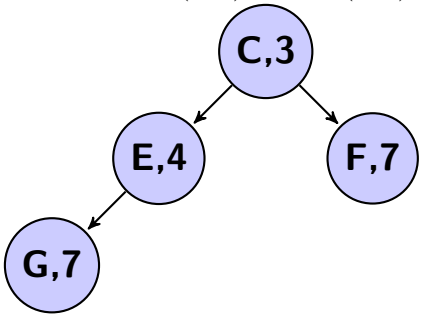
Solution: What's more important than the shape of the heap is understanding what the actions on the heap are at each step. Different implementations of priority queues could result in a different heap shape or order in which same-weight elements get popped, etc. based on the exact implementation.

- Actions: insert (A,0)

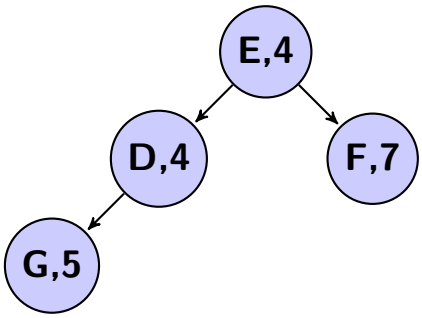
- Actions: pop (A,0), insert (B,1), (F,8), (E,4)



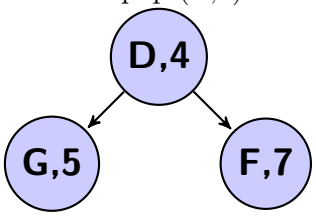
- Actions: pop (B,1), insert (C,3), (G,7), update (F,8) to (F,7)



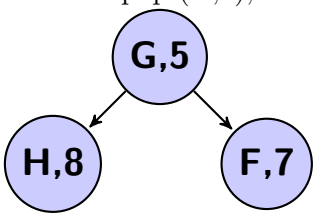
- Actions: pop (C,3), insert (D,4), update (G,7) to (G,5)



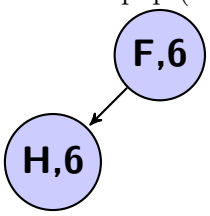
- Actions: pop (E,4)



- Actions: pop (D,4), insert (H,8)



- Actions: pop (G,5), update (H,8) to (H,6), (F,7) to (F,6)

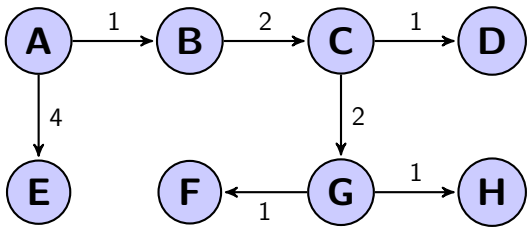


- Actions: pop (F,6)

- Actions: pop (H,6)

(c) Show the final shortest-path tree.

Solution:



2. (optional) Repeat Question 1 on the graph given, but run Prim's algorithm to find a Minimum Spanning Tree.

   (a) Draw a table showing the intermediate cost values of all the nodes at each iteration of the algorithm.

Solution:

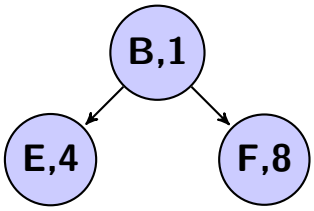
A	B	C	D	E	F	G	H
0	∞	∞	∞	∞	∞	∞	∞
0	1	∞	∞	4	8	∞	∞
0	**1**	2	∞	4	6	6	∞
0	1	**2**	1	4	6	2	∞
0	1	2	**1**	4	6	1	4
0	1	2	1	4	1	**1**	1
0	1	2	1	4	**1**	1	1
0	1	2	1	4	1	1	**1**
0	1	2	1	**4**	1	1	1

   (b) Show the min-heap (or priority heap) at each iteration of the algorithm.

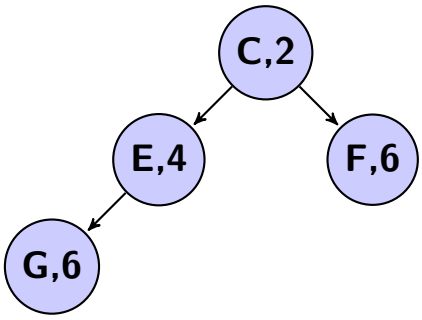
Solution:

- Actions: insert (A,0)

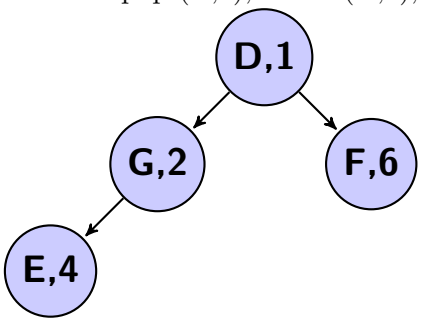
- Actions: pop (A,0), insert (B,1), (F,8), (E,4)



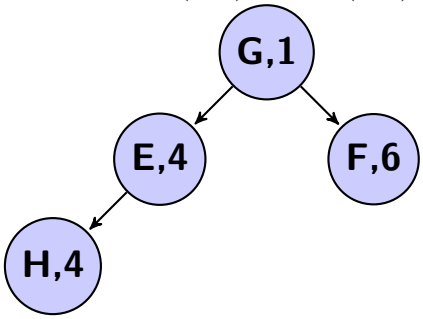
- Actions: pop (B,1), insert (C,2), (G,6), update (F,8) to (F,6)



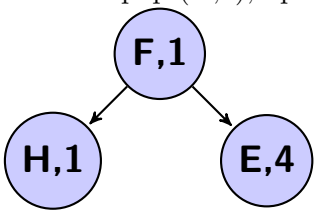
- Actions: pop (C,2), insert (D,1), update (G,6) to (G,2)



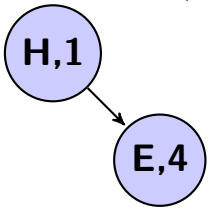
- Actions: pop (D,1), insert (H,4), update (G,2) to (G,1)



- Actions: pop (G,1), update (F,6) to (F,1), (H,4) to (H,1)



- Actions: pop (F,1)

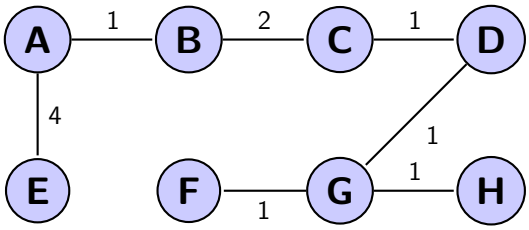


- Actions: pop (H,1)

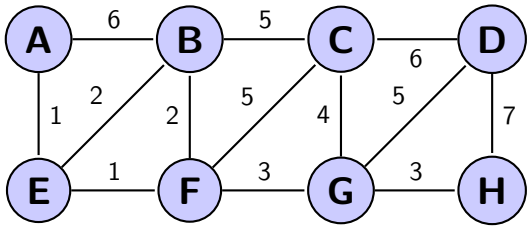
- Actions: pop (E,4)

(c) Show the final MST

Solution:

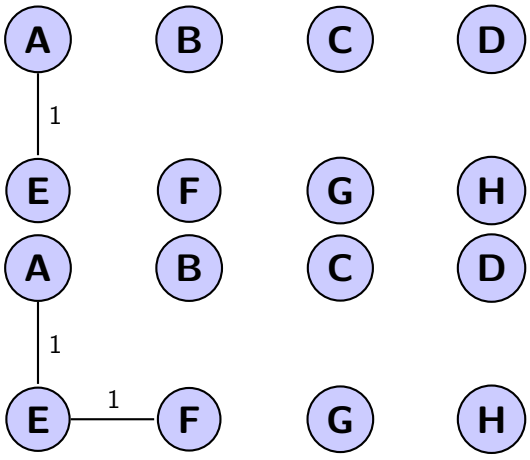


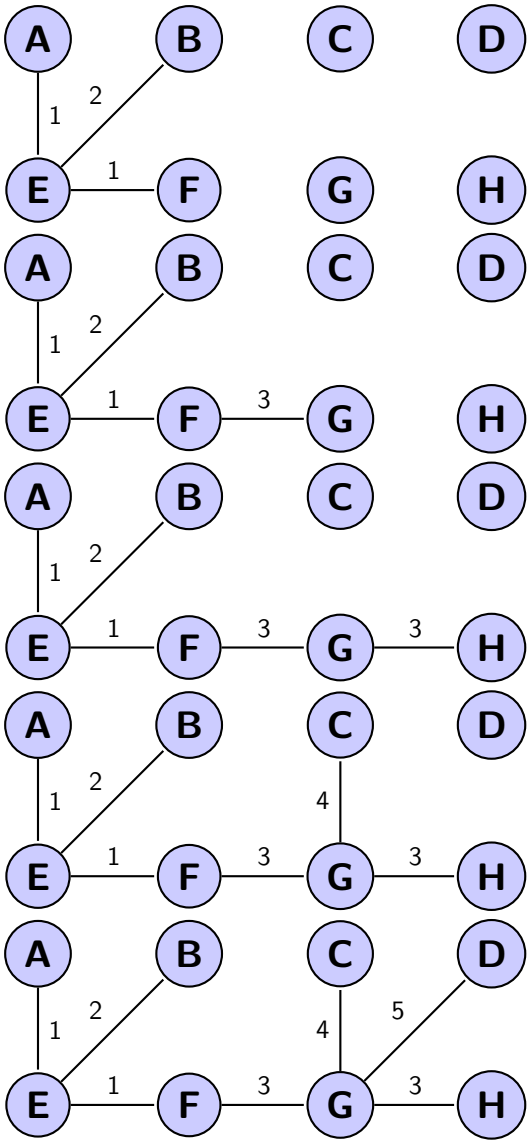
3. (optional) Consider the following graph.



(a) Run Kruskal's algorithm; show the forest at each step.

Solution: Here it is, step by step.

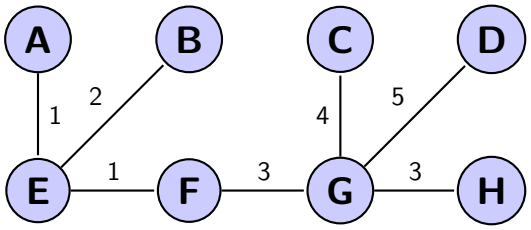


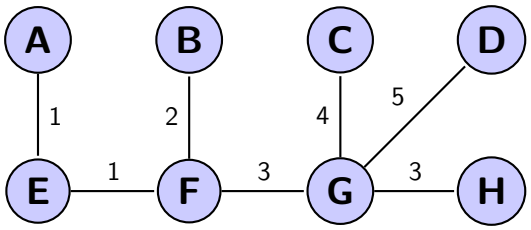


In this particular case, the edges are being added in about the same order as if we were doing Prim's algorithm as well, starting at node A! (This is not the case in general).

(b) How many minimum spanning trees does the graph have? Show each one.

Solution: Only two!





You can see that we really only had one choice - which node to connect B to.

4. (20 points) (group) Given an unweighted, undirected graph $G = (V, E)$ and a edge $e \in E$, design a linear time algorithm that finds the shortest cycle in G that includes e. Describe the algorithm in words or with pseudocode, give a short proof of correctness and analyze the runtime.

Solution: Let $e = (u, v)$ be the edge in question. Consider the graph $G \backslash \{e\}$ (in other words, remove e from the graph). Notice that there is a cycle in G containing e if and only if u is connected to v in $G \backslash \{e\}$.

Our algorithm merely runs BFS from u on $G \backslash \{e\}$ and as soon as v is reached for the first time, outputs the path from u to v plus the edge e. If v is never reached, the algorithm states that e is not in a cycle.

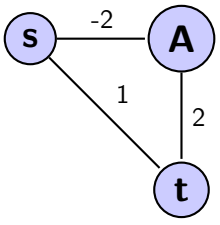
This algorithm runs in linear time as BFS runs in linear time.

Correctness: Any cycle containing e can be broken into the edge e and a path between u and v (the endpoints of e). If u is connected to v in $G \backslash \{e\}$ by some path p, we can add edge e to p to form a cycle. We know we have found the smallest cycle containing e because of the shortest path property of BFS. If v is not reachable from u in $G \backslash \{e\}$, the removal of e from G disconnects u and v, so there is no cycle in G containing e.

5. (20 points) (group) Consider the following algorithm for finding the shortest path from node s to node t in a directed graph with some negative edges: add a large constant to each edge weight so that all the weights become positive, then run Dijkstra's algorithm starting at node s, and return the shortest path found to node t.

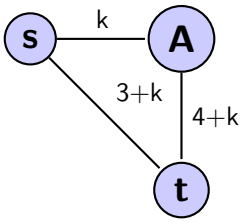
   Is this a valid method? Either prove that it works correctly, or give a counterexample graph where it fails.

Solution: No! We give a counterexample.



In this original graph, the shortest path from s to t goeas through A and has total weight 0.

No matter which number we add to each edge (your solutions could have used just one example), the edge weights will then be in the following form, for some constant $k \geq 0$.



In this modified graph, the direct path from s to t will now be cheaper. This method fails!

6. (60 points) (group) Read this question carefully. In each of the below parts, you will be given some information about a graph $G = (V, E)$ and an edge $e \in E$. You need to decide which of the following three statements (it could be more than one) are possible for G and e by either **giving an example** for that scenario, or by **proving it impossible**.

   - Can there be a graph G so that e is ALWAYS a part of EVERY MST of G?
   - Can there be a graph G so that e is a part of AT LEAST ONE but NOT ALL MSTs of G?
   - Can there be a graph G so that e is NEVER a part of ANY of the MSTs of G?

   Keep in mind that more than one of these statements may be possible. If you can prove that one of the three cases MUST be true for ALL such graphs G, then you don't need to separately disprove the other two options.

   Always assume that the graph $G = (V, E)$ is undirected and connected, and has at least 2 vertices. Do not assume that edge weights are distinct unless this is specifically stated.

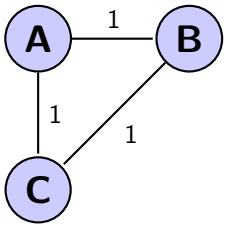
   (a) The edge e is of *minimal* but non-unique weight in all of G.

      i. Is there a graph G so that e is ALWAYS a part of EVERY MST of G?

      Solution: In a path with two edges of weight 1, both edges are minimal and must obviously be in the MST. TRUE!

      ii. Is there a graph G so that e is a part of AT LEAST ONE but NOT ALL MSTs of G?

      Solution: In a cycle, we can choose which edge to omit - it can be e, or it might not be.

      TRUE!

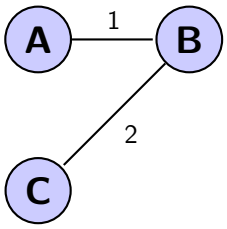
      iii. Is there a graph G so that e is NEVER a part of ANY of the MSTs of G?

Solution: We consider Kruskal's algorithm. If e has minimal weight in G, Kruskal's algorithm has the option to add it as the very first edge of the MST. Thus e must be in *some* MST of G - it is impossible for it to be in *no* MST of G. FALSE!

(b) The edge e is the unique heaviest edge in all of G, and G is not a tree.

i. Is there a graph G so that e is ALWAYS a part of EVERY MST of G?

Solution: True! Consider an edge where the max weight edge isn't in any cycle.

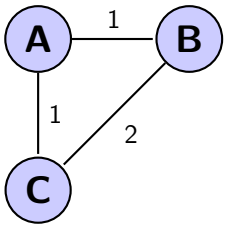


ii. Is there a graph G so that e is a part of AT LEAST ONE but NOT ALL MSTs of G?

Solution: This one is FALSE. Say there is one MST T that includes e and one MST T' that doesn't. Remove the edge e from T to form two connected trees T_1, T_2. In T', there is at least one edge e' that bridges the cut between T_1 and T_2. Add e' to T_1 and T_2 to obtain a spanning tree with less weight than T. This is a contradiction.

iii. Is there a graph G so that e is NEVER a part of ANY of the MSTs of G?

Solution: True! Consider the graph below, the edge with weight 2 isn't in an MST.



(c) Say G has a cycle C, and e is the unique heaviest edge in the cycle C.

i. Is there a graph G so that e is ALWAYS a part of EVERY MST of G?

Solution: By part c), this is impossible

ii. Is there a graph G so that e is a part of AT LEAST ONE but NOT ALL MSTs of G?

Solution: By part c), this is impossible

iii. Is there a graph G so that e is NEVER a part of ANY of the MSTs of G?

Solution: We argue that e can never part of any MST of any such G.
We do a proof by contradiction: Assume e were in an MST T of G. IF we remove e from the MST, then we are left with two disconnected subtrees on either side. Since e was part of a cycle, there is another path in G from the end points of e to each other - some edge in that cycle must be between the two subtrees separated by e.
But that edge has cost less than e, and adding that edge to the two subtrees would create a new spanning tree with lower cost than T. This contradicts the fact that T is an MST.
Therefore, e can never be part of any MST of G.

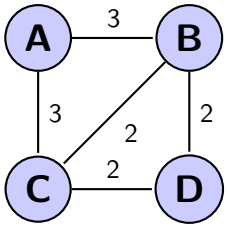
(d) Say G has a cycle C, and e is the unique lightest edge in the cycle C.

i. Is there a graph G so that e is ALWAYS a part of EVERY MST of G?

Solution: In a graph which is only a single cycle C, an MST is any path that removes a maximal weighted edge from the cycle. Thus, any MST must include e, since it is not a maximal weighted edge. TRUE!

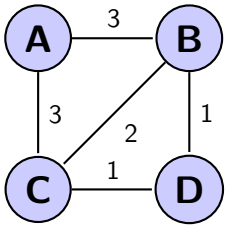
ii. Is there a graph G so that e is a part of AT LEAST ONE but NOT ALL MSTs of G?

Solution: If G has matching edges outside the cycle $C = (A, B, C)$, then $e = (B, C)$ might not be used in any MST, but it also might be.



iii. Is there a graph G so that e is NEVER a part of ANY of the MSTs of G?

Solution: If G has even lighter edges outside the cycle $C = (A, B, C)$, then $e = (B, C)$ might not be used in any MST.



7. (bonus) The **Traveling Salesman Problem (TSP)** is a famous problem in Graph Theory. The problem asks: given weighted graph $G = (V, E)$, with weights $w(e) > 0$ on each edge e, find a cycle C that visits every vertex of G with minimum total weight.

   For now, we'll deal only with **Metric TSP**, where the weight function w satisfies the *triangle inequality*. This means that for all vertices a, b, c, $w(a, c) \leq w(a, b) + w(b, c)$.

There are no known polynomial time algorithm to solve even the metric version of TSP, but we will give an *approximation algorithm*, one that finds a cycle of cost at least twice the weight of the optimum cycle C. Remember that even though we don't know what the weight of the optimum cycle is, we can prove that the cycle we return will be less than twice that weight.

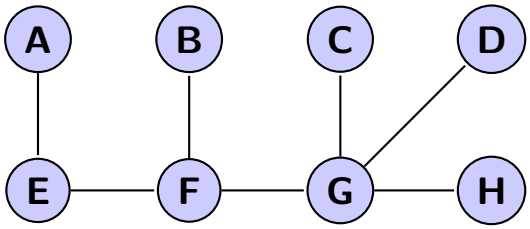
(a) Prove that the MST of the graph has weight strictly less than the optimal cycle C.

Solution: If we remove an edge from the optimal cycle C, we get a path P that covers every vertex, and has strictly less weight than the cycle. This path is also a spanning tree, thus an MST has at most as much weight as this path. Thus the cost of the cycle C is strictly more than the cost of the MST.

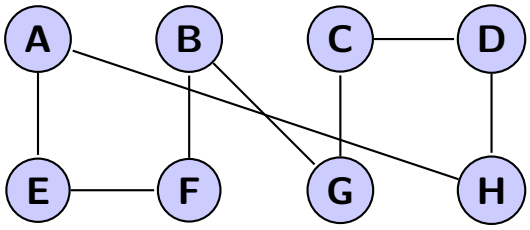
(b) Give an algorithm that can use the MST to find a cycle in the graph with total weight less than $2C$. What is the total runtime of your algorithm?

Hint: Perform a DFS on the MST. Think of the nodes you visit as a sort of improper cycle. Turn that improper cycle into a proper cycle. Show that this proper cycle has weight at most $2C$.

Solution: First, start by creating an MST. Starting at any vertex, if do a DFS of the MST, counting every descent to a vertex as an edge of the cycle and every return also as an edge of the cycle, we have an improper cycle that traverses each edge of the MST exactly twice - Non leaf edges will be counted more than once. For example, in the below MST, our improper cycle could consist of (A,E,F,B,F,G,C,G,D,G,H,G,F,E). This has length $2(|V| - 1)$.



We can now "short circuit" our improper cycle so that we only use each vertex once. We can do this in linear time by iterating through our improper cycle, marking the vertices we've already used, and only adding unmarked vertices to our new cycle. This would result in the cycle (A,E,F,B,G,C,D,H).



Can you finish the proof from here? Why does this short circuited cycle, which uses some completely new edges that aren't a part of the MST, have at most as much weight as our double traversal of the MST? Hint: it has to do with the traingle inequality that we get from the metric.