Homework 6

1. You are given as input an undirected graph $G = (V, E)$ with unit edge lengths (i.e., every edge has weight 1), and two vertices $s, t \in V$. Give a linear time algorithm which will compute the *number* of distinct shortest paths from s to t. Explain clearly why algorithm is correct and analyze its running time.

Solution: Think about this as a Dynamic Programming problem. We will find (together) both the shortest path from s to every node, as well as the total number of paths from s to that node. If the total path from s to a node x has distance d, then it must have passed through a node w at distance $d-1$ that neighbors x. Thus the total number of ways to get to x passing through w is the same as the number of ways to get to w. The total number of ways to get to x is this summed over all such values w.

We could do a two-pass algorithm, one that uses BFS to calculate all shortest path lengths, and a second pass to do the above dynamic programming idea, but we'll present a single-pass algorithm that does both simultaneously. We use a modified version of BFS that tracks the length of the shortest path to each vertex, and also the number of such shortest paths to that vertex.

If you assume (inductively) our algorithm works correctly (counts and computes the distance) for all nodes at distance d then when it processes each node at distance d it will update the count and distance for all adjacent nodes at distance $d+1$.

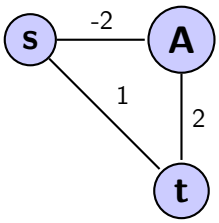
```
function ModBFS(G = (V,E),,s,t)
    for u ∈ V do
        dist(u) = ∞
        count(u) = 0
    dist(s) = 0
    count(s) = 1                                   ▷ Initialization of values
    Q = queue(s)
    while Q is not empty do
        u ← Q.dequeue()
        for all (u, v) ∈ u.outedges do
            if dist(v) = ∞ then                    ▷ The first time we visit a node
                dist(v) ← dist(u) + 1
                count(u) = count(v)
                Q.enqueue(v)
            else if dist(v) = dist(u) + 1 then     ▷ There are additional shortest paths
                count(u) = count(u) + count(v)
    return count(t)
```

The runtime is $O(V + E)$, by the same argument for the runtime as BFS (The total amount of work can be partitioned into a constant amount per vertex and per edge).

2. 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.

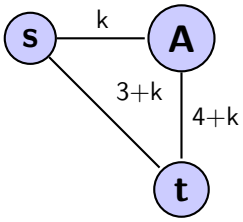
Is this a valid method (assuming the graph has no negative weight cycles- cycles whose total weight is negative)? Either explain why it works correctly, or give a counter-example (and explain clearly why your graph is a counter-example).

Solution: No! We give a counterexample.



In this original graph, the shortest path from s to t goes 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 $k \geq 0$.

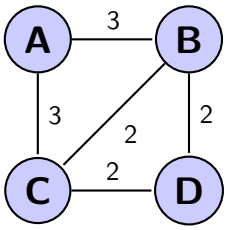


In this modified graph, the direct path from s to t will now be cheaper. This method fails!

3. **(PRACTICE)** For this problem, assume that the graph $G = (V, E)$ is undirected, connected and has at least 2 vertices. Do not assume that the edge weights are distinct. **PROVE** the following statements about minimum spanning trees.

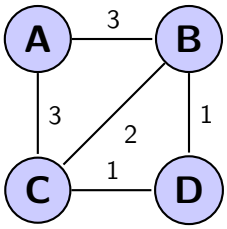
   (a) There exists a graph G which has a cycle C, and e is the **unique lightest** edge in C so that e is a part of **at least one but not all** MSTs of G.

Solution:



Consider the graph above, edge BC is the unique lightest edge on the cycle ABC. This edge is contained in the MST : $\{AB, BC, BD\}$. However, it is not contained in the MST: $\{AB, BD, CD\}$.

   (b) There exists a graph G which has a cycle C, and e is the **unique lightest** edge in C so that e is **never** a part of any of the MSTs of G.



Solution:

Consider the example above, edge BC is the unique lightest edge on the cycle ABC. However, this graph has only 2 MSTs: $\{AB, BD, CD\}$ and $\{AC, CD, BD\}$ neither of which contain BC.

(c) For any graph G, if G has a cycle C, and e is the **unique heaviest** edge in C then e is never a part of any MST on G.

Solution: 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.

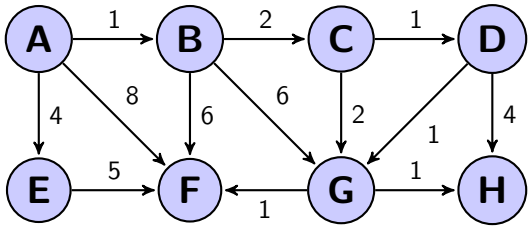
But that edge has cost less than e, and adding that edge to the two subtrees would create a new spanning tree with lower cost than T. This contradicts the fact that T is an MST.

Therefore, e can never be part of any MST of G.

(d) For any graph G, let e be an edge of **minimal** weight then there is some MST which includes e.

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

4. **(PRACTICE)** 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 after each iteration of the while loop.

Solution: Note that I've bolded the node that will get removed from the heap at the next step. Also, since it's dist value will never change I've stopped putting it in the

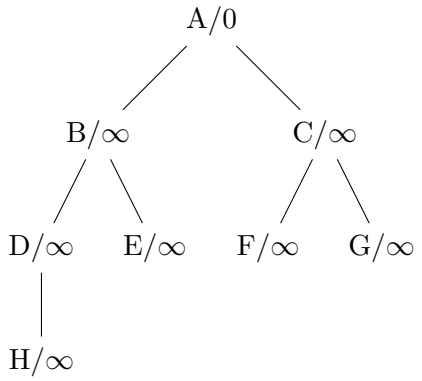
table to make it a bit clearer.

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

(b) Show the min-heap (or priority heap) after each iteration of the while loop. List the heap operations that were performed (e.g. DeleteMin(Q), DecreaseKey(Q, v, 5)).

Solution:

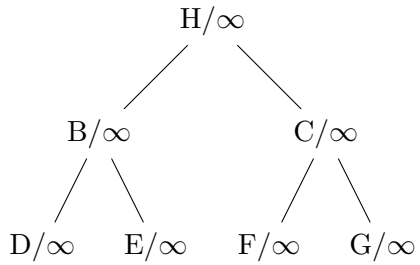
- Actions: Insert (Q,A,0), Insert(Q,B,∞), Insert(Q,C,∞), Insert(Q,D, ∞), Insert(Q,E,∞),Insert(Q,F,∞),Insert(Q,G,∞),Insert(Q,H,∞)



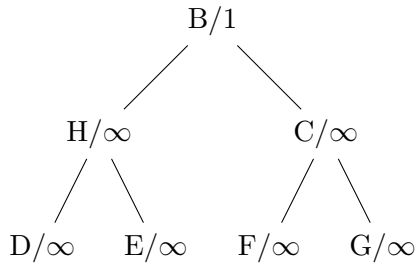
Min Heap: $[A/0, B/\infty, C/\infty, D/\infty, E/\infty, F/\infty, G/\infty, H/\infty]$

- Actions: DeleteMin(Q), DecreaseKey(Q,B,1), DecreaseKey(Q,F,8), DecreaseKey(Q,E,4)

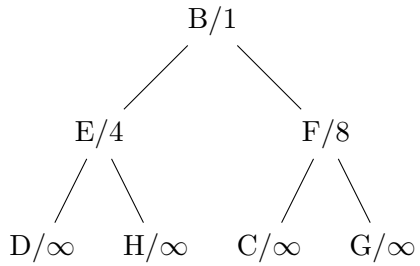
(after the DeleteMin(Q))



(after DecreaseKey(Q,B,1))



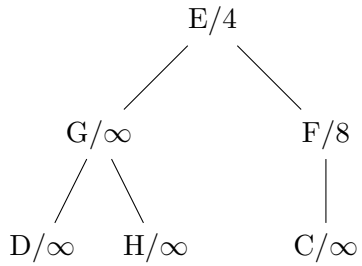
(after DecreaseKey(Q,F,8), DecreaseKey(Q,E,4))



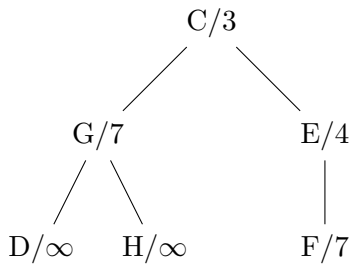
Min Heap: $[B/1, E/4, F/8, D/\infty, H/\infty, C/\infty, G/\infty]$

- Actions: DeleteMin(Q), DecreaseKey(Q,C,3), DecreaseKey(Q,F,7), DecreaseKey(Q,G,7)

[After DeleteMin(Q)]



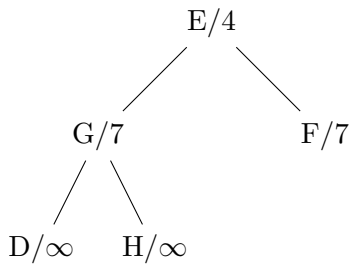
[After DecreaseKey(Q,C,3),DecreaseKey(Q,F,7), DecreaseKey(Q,G,7)]



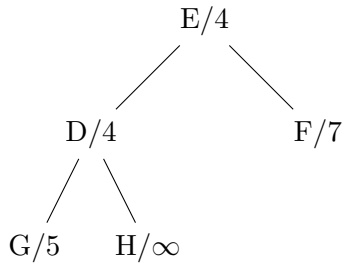
Min Heap: $[C/3, G/7, E/4, D/\infty, H/\infty, F/7]$

- Actions: DeleteMin(Q), DecreaseKey(Q,D,4), DecreaseKey(Q,G,5)

[After DeleteMin(Q)]

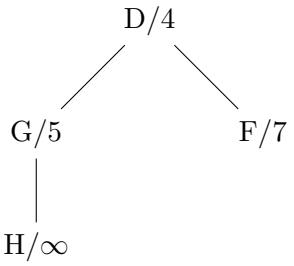


[After DecreaseKey(Q,D,4),DecreaseKey(Q,G,5)



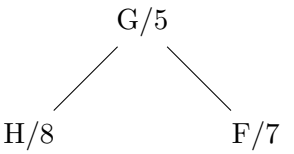
Min Heap: $[E/4, D/4, F/7, G/5, H/\infty]$

- Actions: DeleteMin(Q)



Min Heap: $[D/4, G/5, F/7, H/\infty]$

- Actions: DeleteMin(Q), DecreaseKey(Q,H,8)



Min Heap: $[G/5, H/8, F/7]$

- Actions: DeleteMin(Q), DecreaseKey(Q,F,6),DecreaseKey(Q,H,6),

Min Heap: $[F/6, H/6]$

- Actions: DeleteMin(Q)

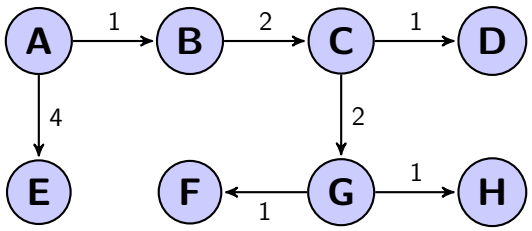
H/6

Min Heap: $[H/6]$

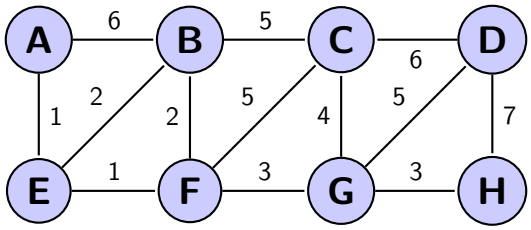
- Actions: DeleteMin(Q)
  Min Heap: []

(c) Give the final shortest path tree.

Solution:



5. **(PRACTICE)** Run **Prim's Algorithm** on the following graph.



(a) Show the tree at each step. Indicate the final spanning tree.

Solution: See part (c) below.

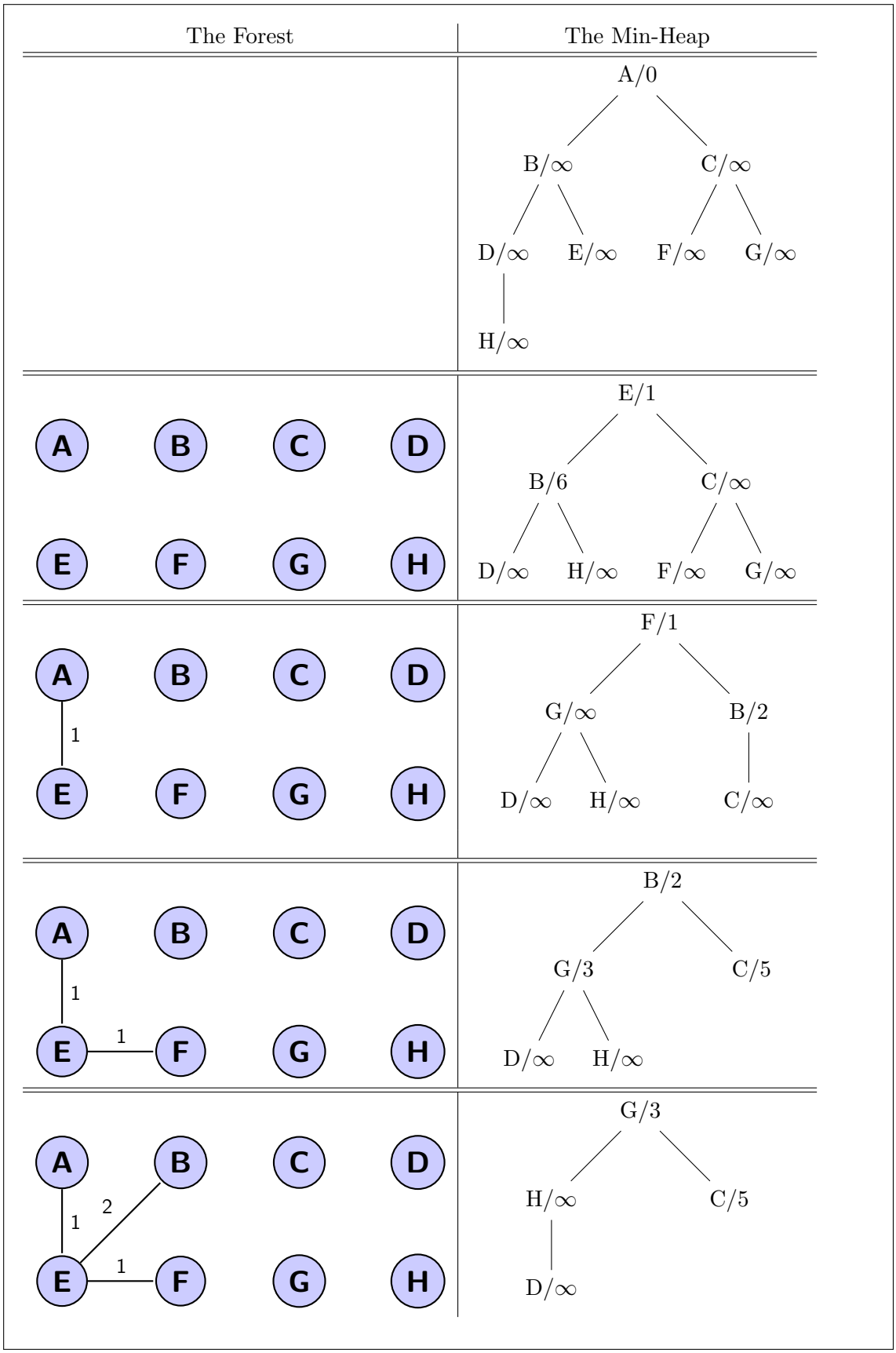
(b) Draw a table showing the intermediate cost values of all the nodes at each iteration of the algorithm.

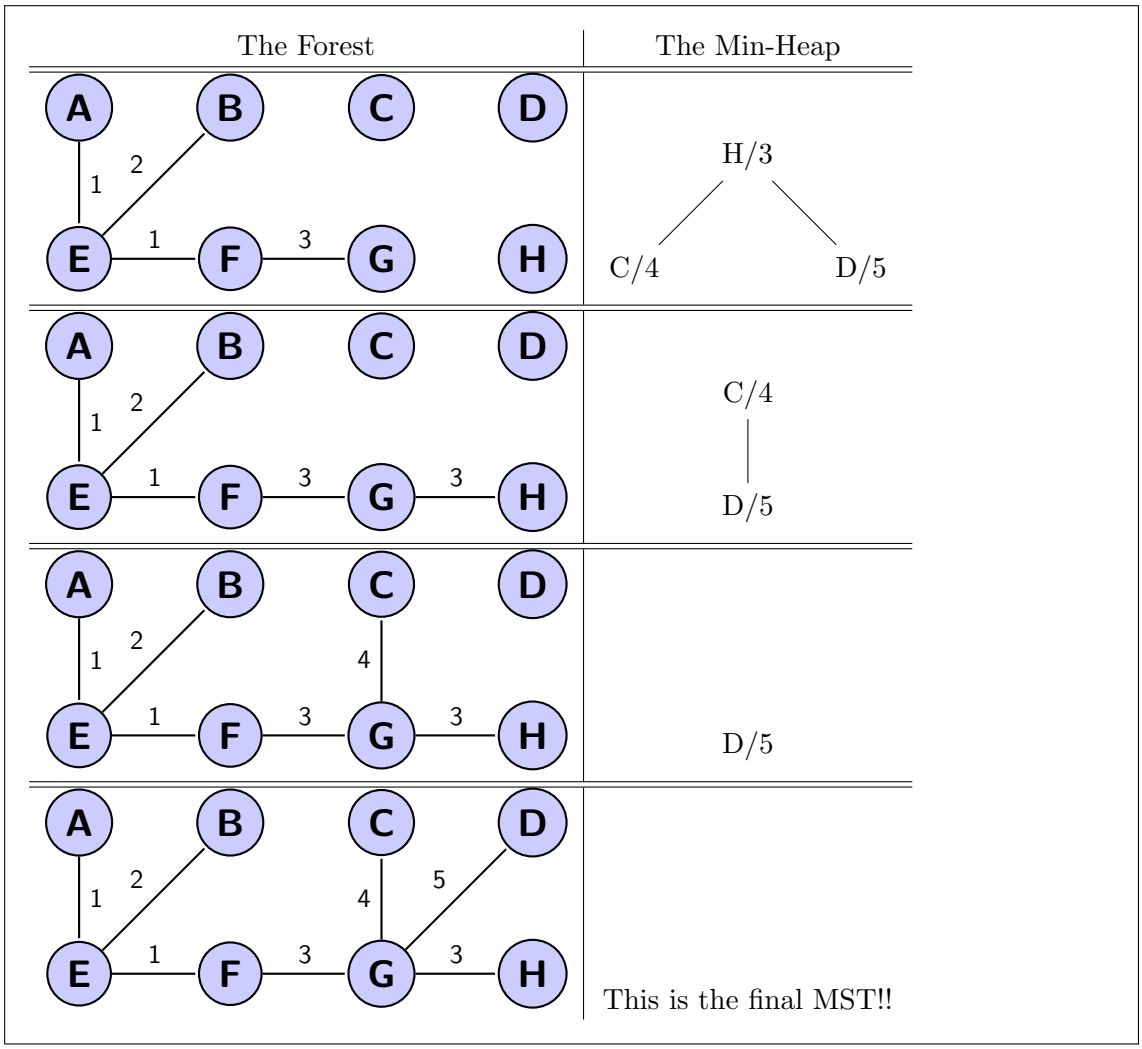
Solution: Note that I've marked the node that gets pulled off the min heap in bold. (Since this node is pulled off the heap next it's cost will never change again and thus I've left it blank to make the table more readable.)

A	B	C	D	E	F	G	H
0	∞	∞	∞	∞	∞	∞	∞
	6	∞	∞	**1**	∞	∞	∞
	2	∞	∞		**1**	∞	∞
	2	5	∞			3	∞
		5	∞			**3**	∞
		4	5				**3**
		4	5				
			5				

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

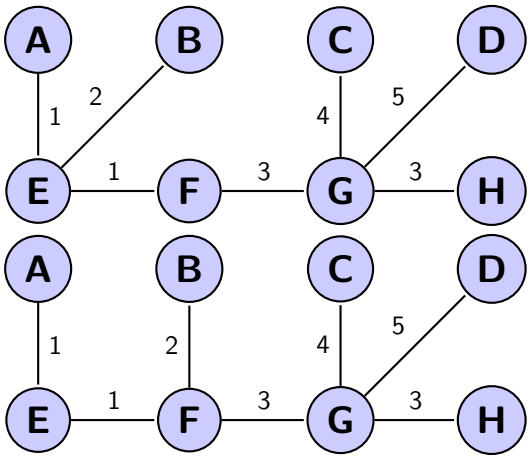
Solution: Here it is, step by step.


The Forest
The Min-Heap
A/0
B/∞
C/∞
D/∞
E/∞
F/∞
G/∞
H/∞
A
B
C
D
E
F
G
H
E/1
B/6
C/∞
D/∞
H/∞
F/∞
G/∞
1
F/1
G/∞
B/2
D/∞
H/∞
C/∞
B/2
G/3
C/5
D/∞
H/∞
G/3
H/∞
C/5
D/∞
2



(d) 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.