Name: Sasha Farhat

ID: 105887541

CSCI 3104, Algorithms **Escobedo & Jahagirdar**

Homework 6 (100 points) **Summer 2020, CU-Boulder**

Advice 1: For every problem in this class, you must justify your answer: show how you arrived at it and why it is correct. If there are assumptions you need to make along the way, state those clearly.

Advice 2: Verbal reasoning is typically insufficient for full credit. Instead, write a logical argument, in the style of a mathematical proof.

Instructions for submitting your solution:

- The solutions **should be typed**, we cannot accept hand-written solutions. Here's a short intro to **Latex**.
- In this homework we denote the asymptomatic *Big-O* notation by $\mathcal{O}$ and *Small-O* notation is represented as o.
- We recommend using online Latex editor **Overleaf**. Download the **.tex** file from Canvas and upload it on overleaf to edit.
- You should submit your work through **Gradescope** only.
- If you don't have an account on it, sign up for one using your CU email. You should have gotten an email to sign up. If your name based CU email doesn't work, try the identikey@colorado.edu version.
- Gradescope will only accept **.pdf** files (except for code files that should be submitted separately on Canvas if a problem set has them) and **try to fit your work in the box provided**.
- You cannot submit a pdf which has less pages than what we provided you as Gradescope won't allow it.

Name: Sasha Farhat

ID: 105887541

CSCI 3104, Algorithms **Escobedo & Jahagirdar**

Homework 6 (100 points) **Summer 2020, CU-Boulder**

Piazza threads for hints and further discussion

Piazza Threads
Question 1
Question 2
Question 3
Question 4
Question 5
Question 6
Question 7
Question 8

Recommended reading:
Graph Algorithms Intro: Ch. 22 → 22.1, 22.2, 22.3
Graph Algorithms SSSPs: Ch. 24 → 24.3

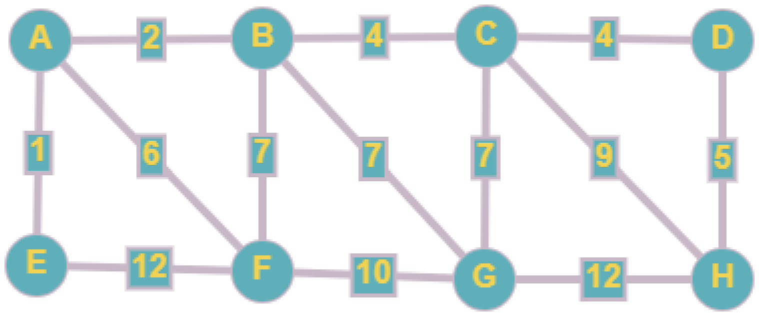
Name: Sasha Farhat

ID: 105887541

CSCI 3104, Algorithms **Escobedo & Jahagirdar**

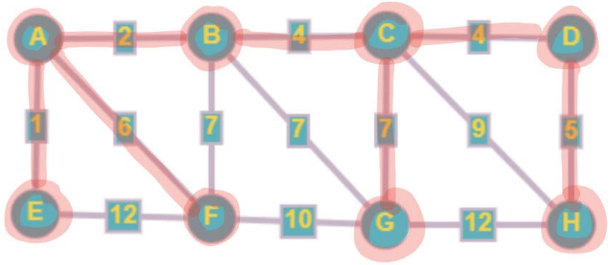
Homework 6 (100 points) **Summer 2020, CU-Boulder**

1. (5 pts) How many unique MSTs does the following graph have. Show the necessary work to justify your answer.

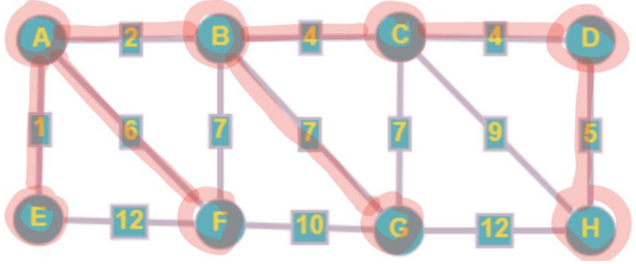


There are two unique MST's within this graph:

① [AE - AF - AB - BC - CG - CD - DH]

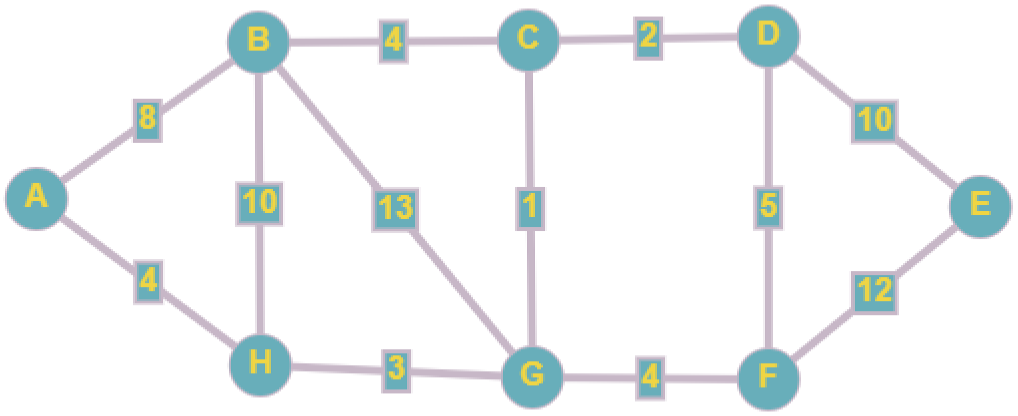


② [AE - AB - AF - BG - BC - CD - DH]



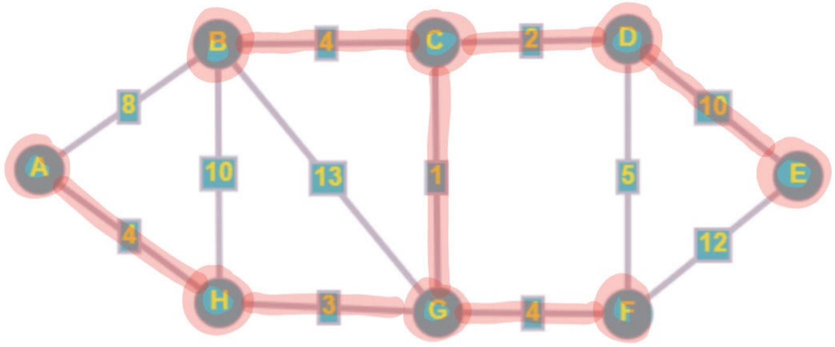
Name: Sasha Farhat
ID: 105887541

2. (20 pts) Based on the following graph :



(a) (10 pts) In what order would Prim's algorithm add edges to the MST if we start at vertex A?

$AH \rightarrow HG \rightarrow GC \rightarrow GC \rightarrow CD \rightarrow CB \rightarrow GF \rightarrow DE$



Name: Sasha Farhat

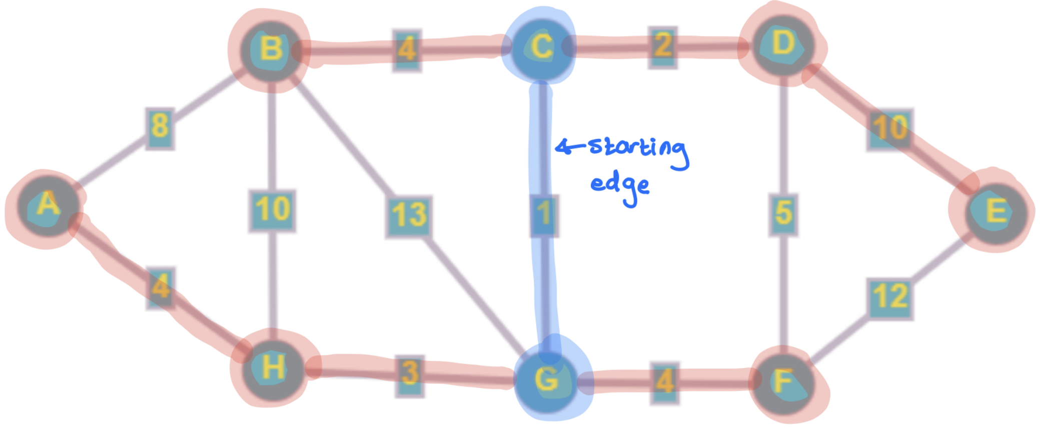
ID: 105887541

CSCI 3104, Algorithms
Homework 6 (100 points)

Escobedo & Jahagirdar
Summer 2020, CU-Boulder

(b) (10 pts) In what order Kruskal's would add the edges to the MST?

$CG \rightarrow CD \rightarrow HG \rightarrow HG \rightarrow AH \rightarrow BC \rightarrow GF \rightarrow DE$



Name: Sasha Farhat
ID: 105887541

CSCI 3104, Algorithms **Escobedo & Jahagirdar**
Homework 6 (100 points) **Summer 2020, CU-Boulder**

3. (10 pts) Suppose that you have calculated the MST of an undirected graph $G = (V, E)$ with positive edge weights. If you increase each edge weight by 5, will the MST change? Prove that it cannot change or give a counterexample if it changes. (Note: Your proof, if there is one, can be a simple logical argument.)

Given the graph G increased all its edges by 2, the MST will not change since all edge weights increased by the same amount. If G has n vertices, G has n-1 edges, therefore when we add 2 to each edge, the cost of the spanning tree also increases.

Name: Sasha Farhat

ID: 105887541

CSCI 3104, Algorithms **Escobedo & Jahagirdar**

Homework 6 (100 points) **Summer 2020, CU-Boulder**

4. (15 pts) Suppose you are given the minimum spanning tree T of a given graph G (with n vertices and m edges) and a new edge $e = (u, v)$ of weight w that will be added to G. Give an efficient algorithm to find the MST of the graph $G \cup e$. Your algorithm should run in $O(n)$ time.

1. Examine the path edge $e = (u, v)$ with weight w. Determine if any vertex within (T) path has a larger weight than w.

2. If there is an edge with weight s, within T, that is larger than w, such that $s > w$, then remove the edge and add the new edge e to get the new MST.

3. If the new edge has larger weight than any of the edges within T, then do nothing to the current MST.

Since finding the path from $u \rightarrow v$ does loop iterations in n times, and all other operations are constat complexities, the algorithm therefore has $\mathcal{O}(n)$ time complexity.

Name: Sasha Farhat

ID: 105887541

CSCI 3104, Algorithms **Escobedo & Jahagirdar**

Homework 6 (100 points) **Summer 2020, CU-Boulder**

5. (20 pts) In a directed graph $G = (V, E)$ with positive edge weights, we define the maximum weight of a path from s to t to be the maximum of the edge weights along the path. For example, if the path from s to t has edges with weights $e_1 = 10, e_2 = 15, e_3 = 5$ then the maximum of the path is $max(e_1, e_2, e_3) = 15$. Give an algorithm to compute the smallest maximum weight paths from a source vertex s to all other vertices. (Hint: Your algorithm should be a modification of Dijkstra's algorithm)

```
smallestMexWeight(G, s)

    1. initialize the distance $s$ to 0
    2. We set all edges to $\infty$ within a struct $v$
    3. Create queue $Q$, and add all vertices $v$ to queue
    4. While $Q$ is not empty
        a. pop Q and store it in variable $u$
        b. for each vertices ($v$) adjacent to $u$
            i. set our current edge ($e_{u,v}$) to max
            ii. if max $<$ the maxedge within our struct $v$, then
                a. set maxedge(v) = to our currently explored edge (max)
                b. update $prev$ to the node connecting with the smaller edge
                   of weights within our struct $v$
    5. return $v(s)$
```

Name:	Sasha Farhat
ID:	105887541

CSCI 3104, Algorithms **Escobedo & Jahagirdar**
Homework 6 (100 points) **Summer 2020, CU-Boulder**

6. (2 pts) What are the two conditions that must be met for the flow on a graph to be valid?

Conditions for network flow:

1. for all edges, the flow on that edge must be at least 0, and no greater than the capacity of the edge
$0 \le f(u, v) \le c(u, v)$ for all edges (u, v)
2. for all vertices, (other than s and t), the flow into the vertex has to equal the flow out of the vertex

7. (3 pts) What do the edge weights in the residual graph G_f represent? Include the definition of both forward and backward edges.

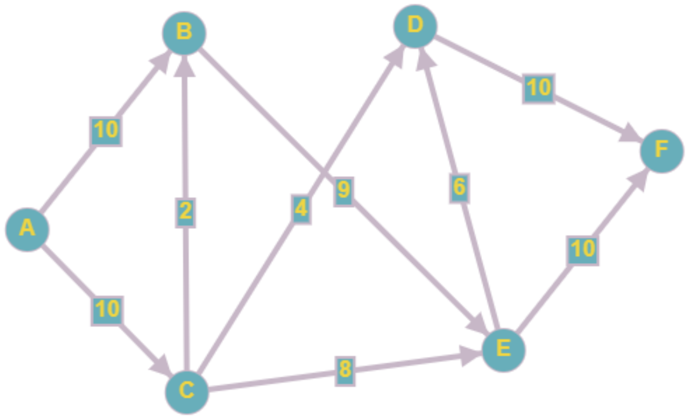
In the residual graph G_f, the edge weight represents the residual capacity which is the current capacity of the edge weight.
Forward edge: For each edge $e = (u, v)$ of graph G for which $f(e) < c_e$, include an edge $e' = (u, v)$ in G_f with capacity $c_e - f(e)$.
Backward edge: For each edge $e = (u, v)$ in graph G with $f(e) > 0$, we include an edge $e' = (u, v)$ in G_f with capacity of the edge.

Name: Sasha Farhat

ID: 105887541

CSCI 3104, Algorithms **Escobedo & Jahagirdar**
Homework 6 (100 points) **Summer 2020, CU-Boulder**

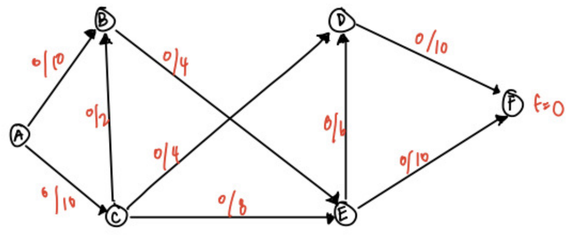
8. (25 pts) Based on the following network and the given edge capacities answer the following.



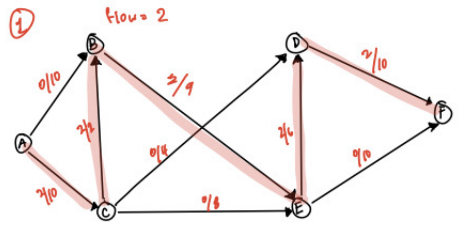
(a) (15 pts) Suppose we start the Ford-Fulkerson algorithm with **A** being the source vertex and **F** being the sink and **select the path** $A- > C- > B- > E- > D- > F$ **in the first iteration (Do not chose the first A-F path on your own).** Complete all the iterations of Ford-Fulkerson to find the Max-Flow (including the first round that is incomplete). Clearly show each round with

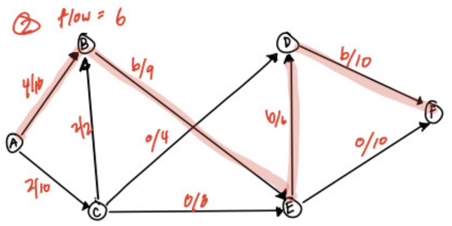
i. The path that you are selecting in that round.
ii. The bottleneck edge on this path.
iii. The additional flow that you push from the source by augmenting (pushing maximum allowed flow along) this selected augmenting path.
iv. The residual graph with the residual capacities (on both the forward and backward) edges.

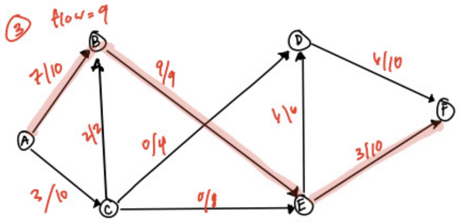
Also, report the Max-Flow after the algorithm terminates.

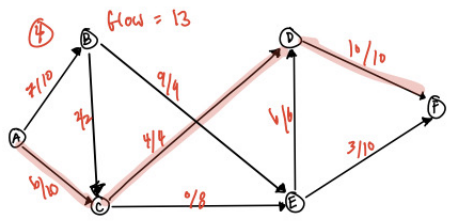


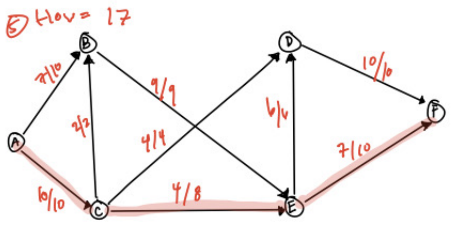
Path	bottleneck cap
① A-C-B-E-D-F	2
② A-B-E-D-F	4
③ A-B-E-F	3
④ A-C-D-F	4
⑤ A-C-E-F	2

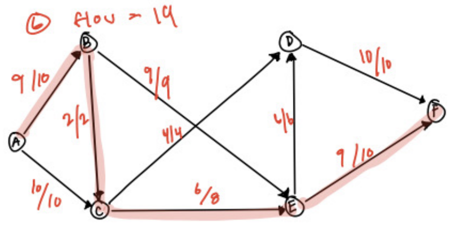












max flow = 19

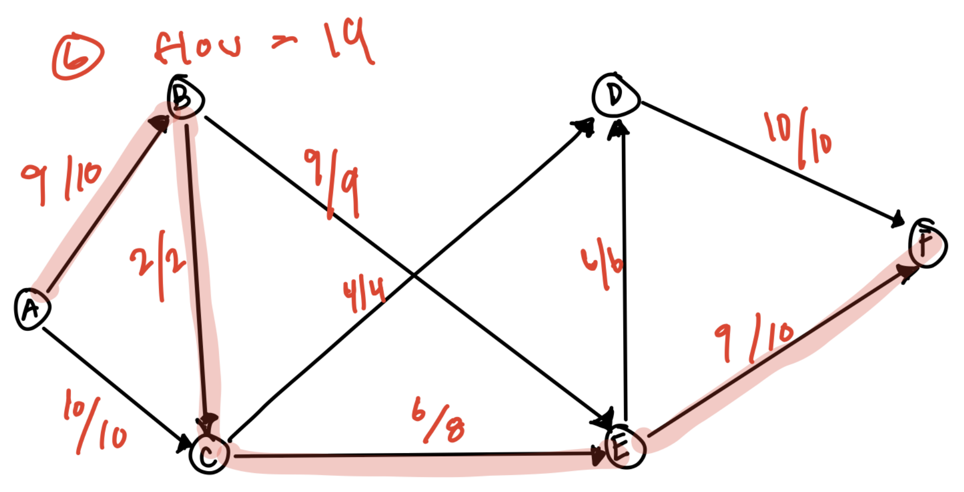
Name: Sasha Farhat

ID: 105887541

CSCI 3104, Algorithms **Escobedo & Jahagirdar**

Homework 6 (100 points) **Summer 2020, CU-Boulder**

(b) (5 pts) Draw the graph and show the final flow f(e) for the edges of the original graph when the Ford-Fulkerson algorithm terminates.



(c) (5 pts) Find the minimum capacity cut with respect to the capacities on the original graph. Is this minimum capacity equal to the Max-Flow that you earlier identified? Justify your answer in a sentence. Also, list the edges that are part of the min-cut and are saturated (can't carry any more flow).

The minimum capacity is equal to the max flow identified. By cutting all possible flows from source to target vertex, we find that the minimum cut is 19 which also is equal to our max flow. The edges that are part of the min cut are $[AC, CB, BE]$

Name: Sasha Farhat

ID: 105887541

CSCI 3104, Algorithms **Escobedo & Jahagirdar**
Homework 6 (100 points) **Summer 2020, CU-Boulder**

9. ***Extra Credit Question 1 (5 pts)*** *For this extra credit question, please refer the leetcode link provided below or click here. Multiple solutions exist to this question ranging from brute force to the most optimal one. Points will be provided based on Time and Space Complexities relative to that of the most optimal solution.*

   Please provide your solution with proper comments which carries points as well.

   https://leetcode.com/problems/cheapest-flights-within-k-stops/

   ```
   Replace this text with your source code inside of the .tex document
   ```

10. ***Extra Credit Question 2 (5 pts)***

    https://leetcode.com/problems/pacific-atlantic-water-flow/

```
class Solution(object):
    def bfs(self, matrix, ocean, m, n):
            #create the 4 directions we will be using when looking at each cell
            direction = zip((1, 0, -1, 0), (0, 1, 0, -1))
            #create a queue with our set from the specific ocean
            queue = list(ocean)

            #while there are elements in our queue
            while queue:
                #create tuple for the initial pop
                hx, hy = queue.pop(0)

                #for elements within our range
                for dx, dy in direction:
                    #i is the grid cord in x
                    i = hx + dx
                    #j is the grid cord in y
                    j = hy + dy

                    if 0 <= i < m and 0 <= j < n:
                        #check if the grid cord is greater than or
                        #equal to the previous pop
                        if matrix[i][j] >= matrix[hx][hy]:
                            #check if not in the visted set
```

```
                            if (i, j) not in ocean:
                                queue.append((i, j))
                                #add to our visited set
                                ocean.add((i, j))

    def pacificAtlantic(self, matrix):
        """
        :type matrix: List[List[int]]
        :rtype: List[List[int]]
        """
        #check to see if we are given a valid matrix
        if not matrix:
            return []

        #get the m x n dimmensions of the graph given
        m = len(matrix)
        n = len(matrix[0])

        #set the top left edges for pacific
        topEdge = [(0, y) for y in range(n)]
        leftEdge = [(x, 0) for x in range(m)]
        pacific = set(topEdge + leftEdge)
        #use bfs to traverse through the set created in pacific
        self.bfs(matrix, pacific, m, n)

        bottomEdge = [(m - 1, y) for y in range(n)]
        rightEdge = [(x, n - 1) for x in range(m)]
        atlantic = set(bottomEdge + rightEdge)
        #use bfs to traverse through the set created in atlantic
        self.bfs(matrix, atlantic, m, n)

        #after bfs is called on both sets
        #add pacific and atlantic to ourfinal results
        result = pacific & atlantic

        return result
```