Name: Sasha Farhat

ID: 105887541

CSCI 3104, Algorithms **Escobedo & Jahagirdar**
Homework 5A (40 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 5A (40 points) **Summer 2020, CU-Boulder**

Piazza threads for hints and further discussion

Piazza Threads
Question 1
Question 2
Question 3
Question 4

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

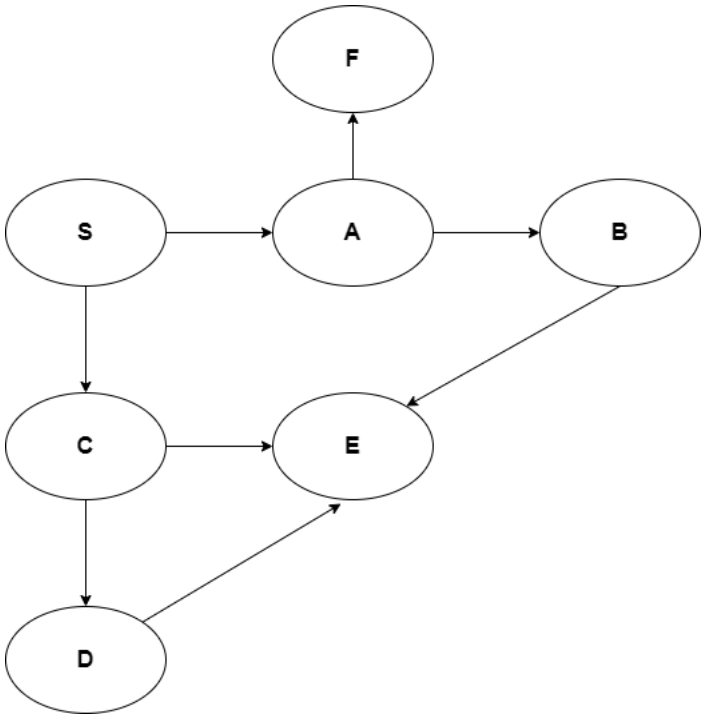
Name: Sasha Farhat

ID: 105887541

CSCI 3104, Algorithms **Escobedo & Jahagirdar**

Homework 5A (40 points) **Summer 2020, CU-Boulder**

1. (5 pts) Consider the following unweighted directed graph.



(a) (2.5 pts) Write down the order in which nodes are visited if Depth First Search (DFS) is called on the above graph with **S** as the source node.

S, A, B, E, F, C, D

(b) (2.5 pts) Write down the order in which nodes are visited if Breadth First Search (BFS) is called on the above graph with **S** as the source node.

S, A, C, F, B, D, E

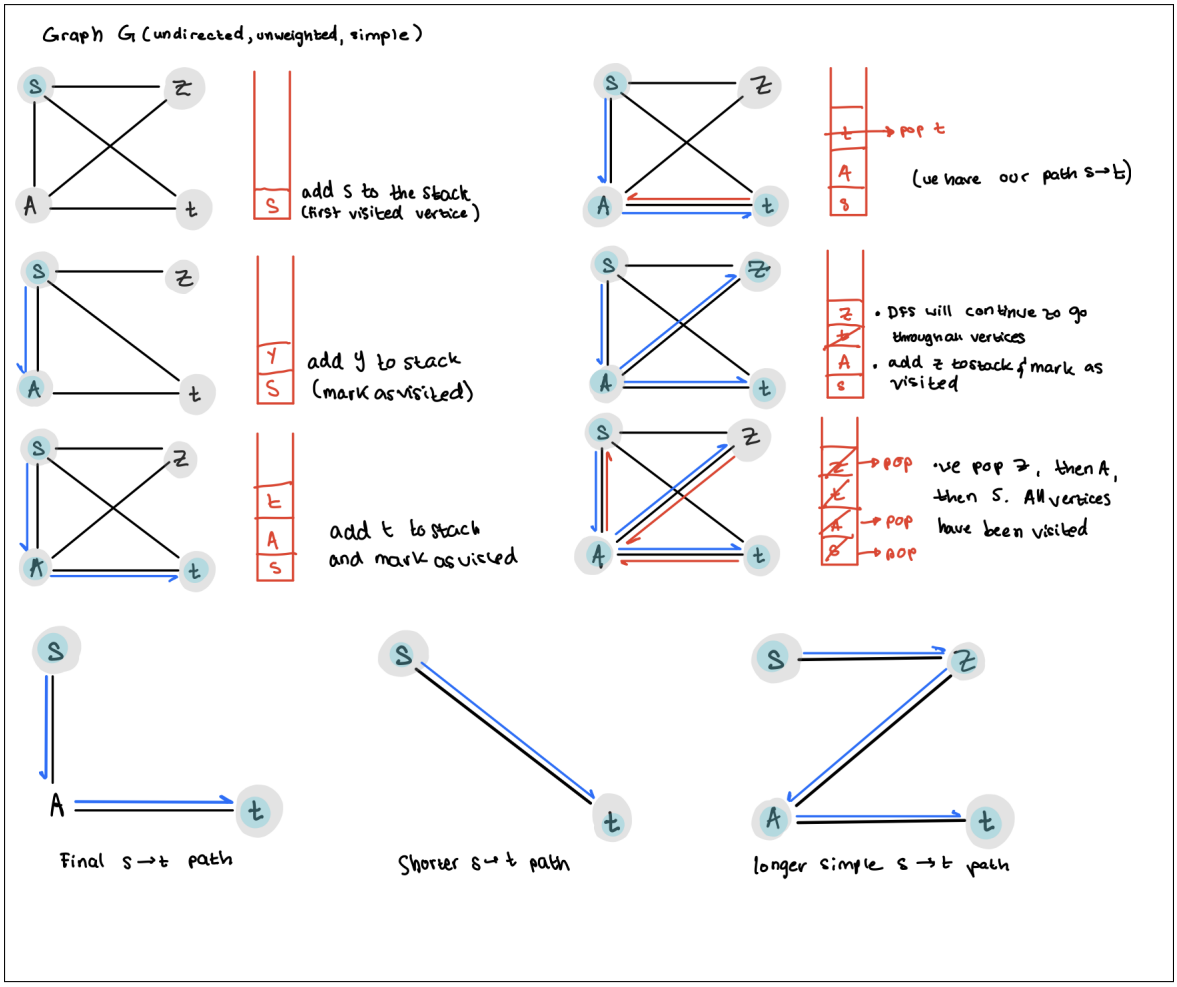
Name: Sasha Farhat

ID: 105887541

CSCI 3104, Algorithms **Escobedo & Jahagirdar**

Homework 5A (40 points) **Summer 2020, CU-Boulder**

2. (5 pts) A simple path $s \to t$ in a graph G is a path starting at s, ending at t, and never visiting the same vertex twice. Give an example graph (simple, undirected, unweighted) $G = (V; E)$ and vertices $s, t \in V$ such that DFS finds a path from s to t which is neither a shortest path nor a longest simple path. Detail the execution of DFS (list the contents of the stack at each step, and which vertex it pops off the stack), show the final $s \to t$ path it finds, show a shorter $s \to t$ path, and a longer simple $s \to t$ path.



Name: Sasha Farhat

ID: 105887541

CSCI 3104, Algorithms **Escobedo & Jahagirdar**
Homework 5A (40 points) **Summer 2020, CU-Boulder**

3. (20 pts) Assume you're given an integer matrix that representing islands in an ocean. A value of 0 in a particular cell indicates water and a value of 1 indicated land. An island is region of land connected vertically, horizontally, or diagonally. The size of an island is the total number of connected land cells. Write an algorithm to compute the sizes of all islands in the matrix. Example:

```
1 0 0 1
0 1 0 1
0 1 0 0
0 1 0 1
```

would output 1, 2, 4.

(a) (6 pts) State if the graph data structure that your algorithm will use for this problem will be i) be weighted or unweighted and ii) directed or undirected. In addition, iii) what makes two nodes have an edge/connection? Give an explanation for each of your answers.

i) The algorithm we will use will be an unweighted graph. This is because we are counting the number of connected components and not the weight between the components.
ii) The graph will be undirected. This is because we are counting the number of 1's connected to each other regardless the direction of the graph.
iii) A graph has a node when two vertices or nodes are connected to each other by a path or paths which are connected to no other vertices outside of that subgraph.Let's denote M as a given integer within the matrix G. Suppose v is some cell of M where $v \in M$. We can then say that the corresponding value is denoted by M_v. In our matrix, let's denote rows as i and columns as j, thus the corresponding row and column for the integer M_v as i_v and j_v respectively.The graph G has a node v if and only if $v \in M$ and $M_v = 1$. Thus we can say that the edge is denoted by $v, u \in G$ if and only if $|i_v - i_u| \leq 1$ and $|j_v - j_u| \leq 1$.

(b) (4 pts) Provide a 3-4 sentence description of how your algorithm works, including how the matrix is converted to the graph, how adjacent vertices are identified, and how the algorithm traverses the graph to identify connected vertices.

My algorithm will have two functions, a function that checks that the row and column that is called on the cell if it is in range of the matrix, and a recursive call function which will be a modified version of dfs. The algorithm will first create a boolean array to store the visited the cells which will be used to check if the cells in the matrix in the graph have been visited. The algorithm will then traverse through all the cells in the graph by going through each row i and each row j. If the cell has a value 1 and is not yet stored in the visited array, the dfs function will be called on that cell and count the island size every time dfs is called recursively for neighboring cells with value 1. The dfs function will be performed on a 2D matrix. The algorithm can only be performed on a given 2D matrix. When dfs comes across no more 1's within 8 direction neighbors, the algorithm returns to the main for loop which continues to check cell and calls dfs on the unvisited cell. The counted island size is stored in an array and when all the cells have been checked and all the cells with 1 have been visited, the main for loop exits and array d is returned with the island size values.

Name: Sasha Farhat

ID: 105887541

CSCI 3104, Algorithms **Escobedo & Jahagirdar**
Homework 5A (40 points) **Summer 2020, CU-Boulder**

(c) (10 pts) Write well commented pseudo or real code to solve this problem.

```
class Graph:

	def __init__(self, row, col, g):
		self.ROW = row
		self.COL = col
		self.graph = g

	# a function so see if the column and row can be used in the DFS function
	def isSafe(self, i, j, visited):
		# row and column are in range and contain value 1
		return (i >= 0 and i < self.ROW and
				j >= 0 and j < self.COL and
				not visited[i][j] and self.graph[i][j])


	# DFS function that looks at the cells 8 adjacent neighbors
	#the dfs function will recursivly call its self for every 1 found
	def DFS(self, i, j, visited):

		#these arrays are used to find the neighbors
		#in 8 directions
		rowNbr = [-1, -1, -1, 0, 0, 1, 1, 1];
		colNbr = [-1, 0, 1, -1, 1, -1, 0, 1];

		#create counter to count the island sizes
		count = 1
		# Mark this cell as visited
		visited[i][j] = True

		# recursively iterates through each neighbor (8 directions)
		for k in range(8):
			if self.isSafe(i + rowNbr[k], j + colNbr[k], visited):
				#count the recursive calls for when there is a 1 neighbor
				count += self.DFS(i + rowNbr[k], j + colNbr[k], visited)
		#return the island size
		return count

	 def islandsize(self):

		 # Initially all cells are unvisited
		 visited = [[False for j in range(self.COL)]for i in range(self.ROW)]
		 #create empty array that stores the island sizes in the graph
		 d = []
		 #travese through the all cells of the graph
		 for i in range(self.ROW):
			 for j in range(self.COL):
				 # if cell with value 1 is not visited yet
				 if visited[i][j] == False and self.graph[i][j] == 1:
					 # call dfs to count the island size around the new
					 #found island
					 count = self.DFS(i, j, visited)
					 #add the value to the array
					 d.append(count)

		 return d


graph = [[1, 0, 0, 1],
		 [0, 1, 0, 1],
		 [0, 1, 0, 0],
		 [0, 1, 0, 1]]


row = len(graph)
col = len(graph[0])

g = Graph(row, col, graph)

print "Island Sizes are:"
print g.islandsize()
```

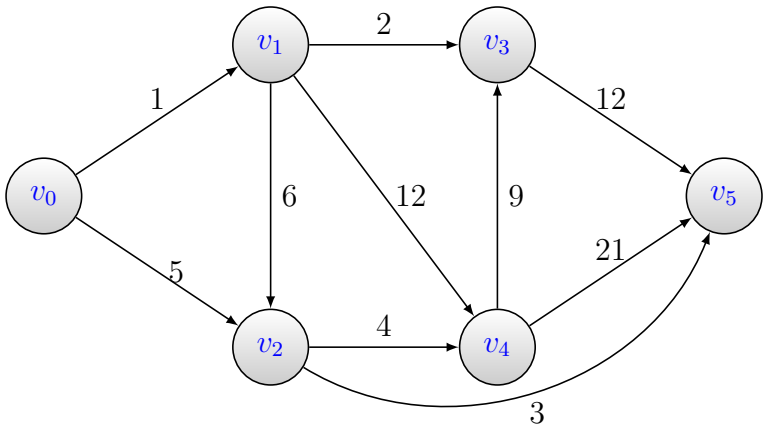
Name: Sasha Farhat

ID: 105887541

CSCI 3104, Algorithms **Escobedo & Jahagirdar**

Homework 5A (40 points) **Summer 2020, CU-Boulder**

4. (10 pts) Using Dijkstra's algorithm, determine the length of the shortest path from v_0 to each of the other vertices in the graph. Clearly specify the distances from v_0 to each vertex **after each iteration** of the algorithm.



Starting at v_0 we update the distance at vertex v_1 and v_2 as 1, and 5 respectively since both values are less than infinity. We visit v_1 since it has the lower tentative distance. Since v_1 is neighbors to 3 unvisted nodes, v_2, v_3, v_4, we must calculate new values for these nodes. From v_1 to v_2 we get a value of 7 which is greater than 5, so we do not update the value of v_2. From v_1 to v_3 we get a value 3. Since 3 is less than infinity we update its value. From v_1 to v_4 we get a value of 13. Since 13 is less than infinity we update the value of v_4 to 13. Now we visit the next lowest valued vertex from v_0, which is v_2. From v_2 we only re-calculate the distance to v_4 since 9 is less than 13. We update v_4 to 13. From v_2 to v_5 the value is 8. Since 8 is less than ∞ we update the value of v_5 to 8. We evaluate the next lowest node which is v_3. From v_3 to v_5, it has a tentative value of 13. Since 13 is greater than 8, we ignore it. From v_4 to v_3 the tentative value 18 which is greater than 3 and thus we ignore it. From v_4 to v_5 the tentative value 30 is which is greater than 8, and thus we ignore it. Using Dijkstra's algorithm, we determined the length of the shortest path from v0 to each of the other vertices in the graph. **We can say using Dijktra's Algorithm that the shortest path from v_0 to v_5 is $[v_0, v_2,$ $v_5]$.**

Name: Sasha Farhat

ID: 105887541

CSCI 3104, Algorithms **Escobedo & Jahagirdar**
Homework 5A (40 points) **Summer 2020, CU-Boulder**

5. ***Extra Credit (5% of total homework grade)*** *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/keys-and-rooms/

```
class Solution(object):
    def canVisitAllRooms(self, rooms):
        N = len(rooms)
        #create boolean array of size N
        visited = [False for i in range(N) ]
        #using dfs we will keep track of the visited rooms
        def dfs(i):
            # if the room is visited we return, this is the initial room 0
            if visited[i] :
                return
            #set the visited room to true
            visited[i] = True
            #we look at each key, and add those
            #keys to the available_room only IF they are not visited yet.
            for key in rooms[i]:
                if not visited[key]:
                    dfs(key)
        #the first value is 0, and we recurse through the rest
        dfs(0)
        #if a room is already visited but a key cannot access
        # the room, we return false on the rom
        for room in visited:
            if room == False:
                return False
        return True
```