CSE 100 Advanced Data Structures Fall 2018

Homework 5

Instructors: *Leo Porter* *(Sec. A)*
& *Debashis Sahoo (Sec. B)*

Due on: Wednesday Dec 5th *(40 points)*

Name: ShihHan Chan PID: A15677346

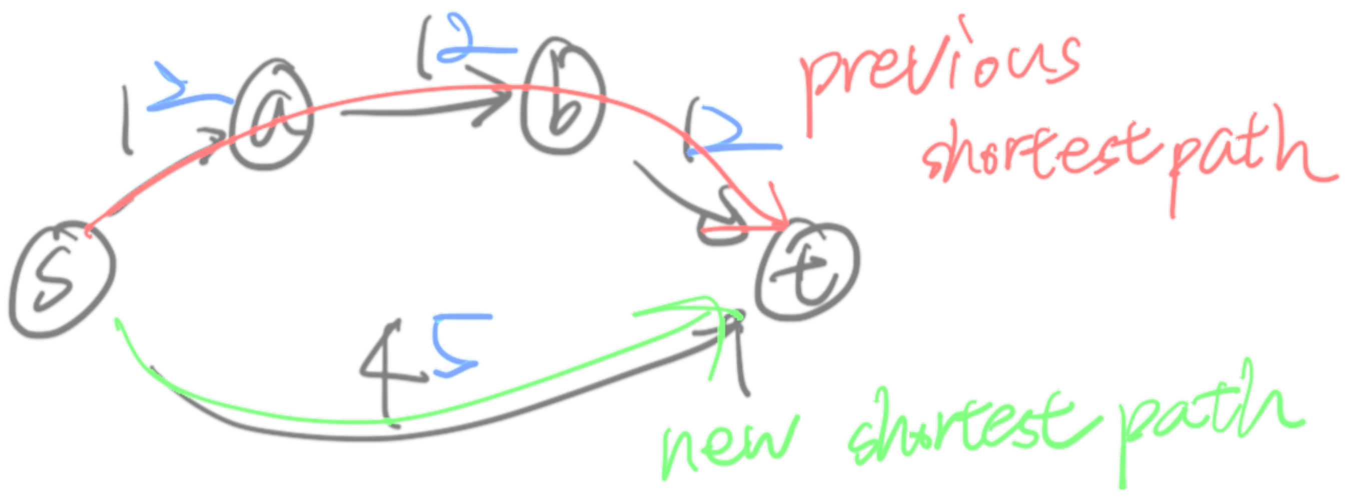
Instructions

1. Answer each problem in the boxes provided. Any writing outside of the boxes *will NOT be graded*. Do not turn in responses recorded on separate sheets.
2. The size of boxes does *not* necessarily correspond to the amount of space needed to solve the problem.
3. Handwritten or typed responses are accepted. In either case, make sure all answers are in the appropriate boxes.
4. All responses *must* be neat and legible. Illegible answers will result in zero points.
5. Make sure to *scan in portrait mode* and to *select the corresponding pages* on Gradescope for each question.
6. You may use code from any of the class resources, including Stepik. You may not use code from other sources.

1. (5 points **- Correctness**) Shortest Paths

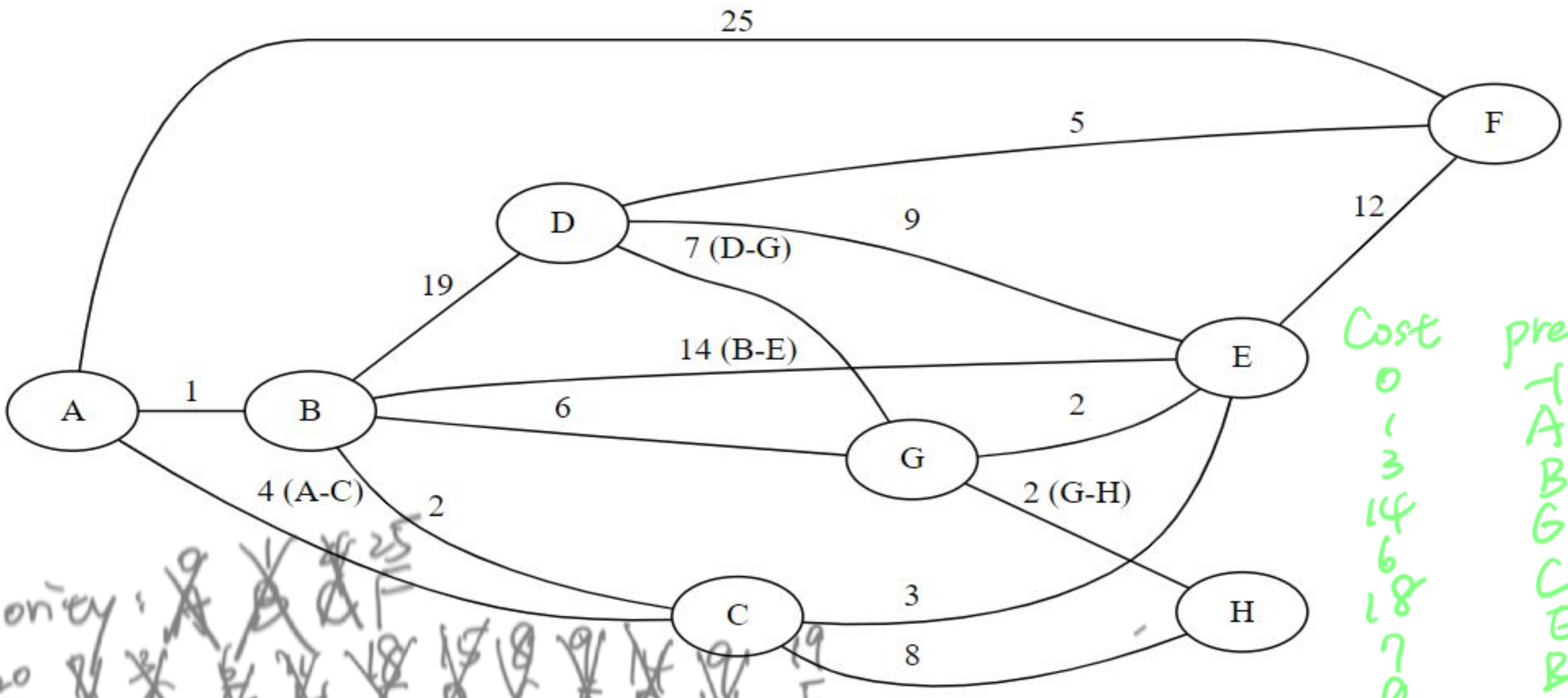
Let A be the shortest path from some vertex s to some other vertex t in a directed graph with nonnegatively weighted edges. If the weight of each edge in the graph is increased by one, will A still be the shortest path from s to t? If true, prove it (remember, the proof should be comprehensive - i.e it should exhaust all scenarios). If false, provide a counterexample.

False



2. (2 points - **Completeness**) **Dijkstra's**

Walk-through Dijkstra's algorithm to compute the shortest paths from A to every other node in the given graph. Show your steps in the table below. Do this by crossing out old values and writing in new ones as the algorithm proceeds.



priority: ~~A~~ ~~B~~ ~~C~~ 25 F
20 D ~~A~~ ~~C~~ ~~E~~ ~~H~~ ~~E~~ ~~D~~ ~~G~~ ~~E~~ ~~D~~ ~~H~~ 19 F

Cost	prev
0	-
1	A
3	B
14	G
6	C
18	E
7	B
9	G

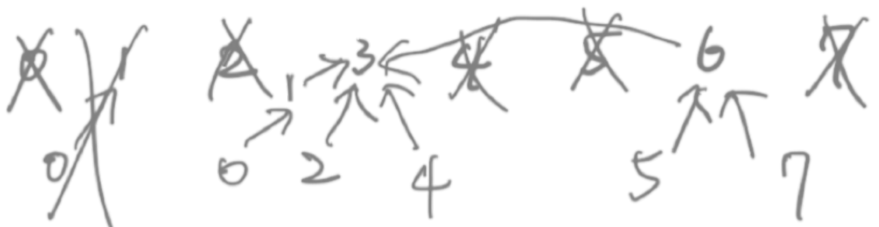
Vertex	**Visited**	**Cost**	**Previous**
A			
B			
C			
D			
E			
F			
G			
H			

(4 points - **Correctness**) All Vertices, In Order Visited: (Visited == Found the Shortest Path to)

A, B, C, E, G, H, D, F

(4 points - **Correctness**) Shortest Path from A to F:

A, B, C, E, F



3. **Disjoint Sets // Union-Find**

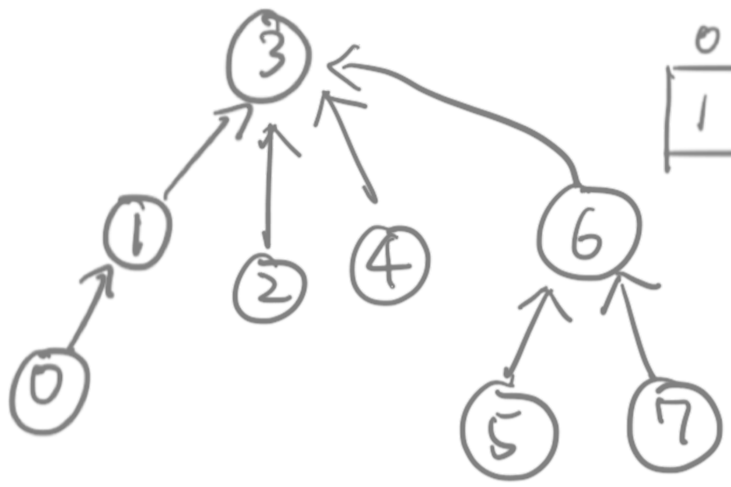
Union(x,y) := Make the sentinel of x point to the sentinel of y. Does not perform the weighting optimization.

You are given a set of unconnected nodes 0 through 7. You now perform the following operations in the given order on the nodes.

Union (0,1), Union (2,3), Union (0,2), Union (4,0), Union(5,6), Union(7,5), Union (5,4), Find (5)

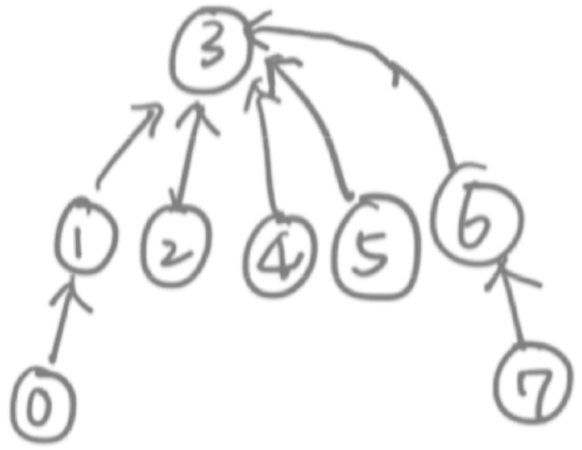
Give the array representation of this disjoint set data structure if the above operations use ... find operation

(a) (5 points **- Correctness**) ... no path compressions path compression works only in



0	1	2	3	4	5	6	7
1	3	3	-1	3	6	3	6

(b) (5 points **- Correctness**) ... path compressions



0	1	2	3	4	5	6	7
1	3	3	-1	3	3	3	6

4. **P vs NP**

(a) (5 points **- Correctness**) Suppose you have a problem that you believe is in NP-Complete. What is the most common way to go about showing that your intuition is correct? Explain what steps you would take, and why those steps show that the problem is in NP-Complete.

If you want to show a problem is NP-complete, we need to show it is both NP and it's at least as hard as any other problem in NP.
And I will explain my steps and tell why those steps show the problem is in NP-complete.
step1: show it is a NP problem.
We first need to argue that there is an efficient verifier for this problem. On the other hand, If we take a verifier that satisfy the following: for any yes instance of this problem, there exists a certificate that the verifier will accept, and for any no instance of this problem, certificate doesn't exist.Run time of the verifier or the size of certificate must be polynomial. And also, if we can find a solution to the given problem, then it is a sufficient certificate, and we finish this step.
step2: pick a known NP-complete problem.
We pick a known NP-complete problem and try to reduce it to the problem we want to solve.
step3: Build an algorithm to solve NP-complete problem given an algorithm to solve the problem we want to solve:
We need to show how to solve NP-complete problem by building one input to the problem we want to solve, and show our algorithm will output the same answer as is given by the problem we want to solve. And we need to pay attention to that this is not always the case Sometimes, we define the correspondence between the output of the problem we want to solve and the desired solution for NP-complete problem.
step4: prove the correctness of the algorithm
We need to prove the if and only if statement, and we want something that can satisfy two conditions below:
1. prove the solution we find is correct
2. prove we find all the solutions
step5: wrap up and run time
Check our construction for the problem we want to solve is polynomial space and polynomial time. And finally, we can conclude our algorithm is in polynomial time.

(b) (5 points **- Correctness**) Suppose you have a problem that you believe is in P. What is the most common way to go about showing that your intuition is correct? Explain what steps you would take, and why those steps show that the problem is in P.

A problem is of class P means this problem can be solved in polynomial time or better. So if
we can find and prove an algorithm can solve the problem in polynomial time or better, we
said that this problem is in P.
And I will explain my steps and tell why those steps show the problem is in P.
Step1: Give a direct polynomial-time algorithm for computing the problem
Step2: pick a known P problem.
Step3: Reduce the problem we want to solve to the known P problem that can be computed in
polynomial time
And then we can know this problem is in P.

5. (5 points **- Completeness**) **Islands**

You are given a 2D array consisting of '1's (land) and '0's (water). Write pseudocode to count the number of islands in the 2D array. An island is surrounded by water and is formed by lands adjacent to each other either horizontally or vertically, not diagonally. You may assume all four edges of the grid are all surrounded by water.

```
//assume all four edges of the grid are all surrounded by water
const int a;
const int b;
char array[a][b];
bool visit[a][b];
void dfs(int y,int x){
    visit[y][x]=true;
    //right
    if(x+1<b){dfs(y,x+1);}
    if(y-1>=0){dfs(y-1,x);}
    if(x-1>=0){dfs(y,x-1);}
    if(y+1<a){dfs(y+1,x);}
}
int countland(){
    int ans=0;
    for(int i=0;i<a;i++){for(int j=0;j<b;j++){visit[i][j]=false;}}
    for(int i=0;i<a;i++){for(int j=0;j<b;j++){if(!visit[i][j]){ans++;dfs(i,j);}}}
    return ans;
}
```