Data Structures and Algorithms Homework 9

Due Wednesday Oct 30; Joseph Sepich (jps6444)

1 Problem 1

Collaborators: None

First let's prove that **if** there is perfect matching, **then** there is any subset where the condition holds.

If you look at the base case with only 1 boy and 1 girl, then with perfect matching the condition must hold true, because the empty set, trivially holds true, and the set with 1 boy and 1 girl is the full set of perfect matching we started with. Suppose then it is true that with perfect matching, a subset of k boys is connected to at least k girls. If we look at k+1 boys, then if we have perfect matching we know that the k+first boy must be matched to at least one more girl, so you must have at least k+1 girls.

Then if you look at **if** any subset of boys the condition holds, **then** there is perfect matching.

At the base case you have 1 boy and 1 girl (or more). Since the boy is connected to at least 1 girl, there must be perfect matching. Let's assume then if you have k boys, who are connected to at least k girls, you have perfect matching. If you then have k+1 boys, who are connected to at least k+1 girls, then that implies the additional boy is connected to a new girl, or one that was not matched with only k boys, so then you must have perfect matching.

Since the conditional holds both ways then the given if and only if statement must be true.

2 Problem 2

2.1 Part 1

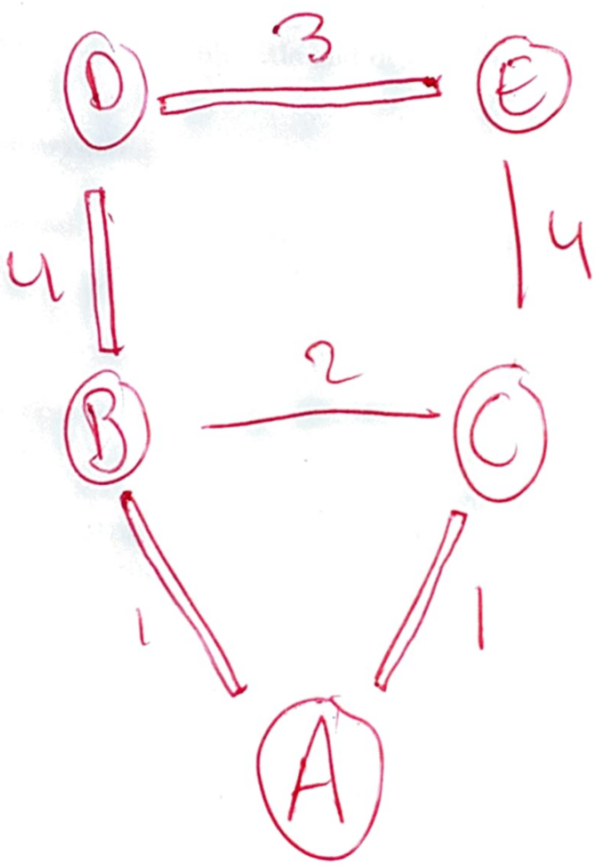
The minimum edge weight in G must be a part of some MST. You can see this via the minimum cut property that we had proven. This property states that the edge with the smallest weight across any cut of G must be in some MST. Any cut that has the minimum weighted edge that crosses it wouled therefore have to have that edge as the minimum weighted edge across the cut, and therefore be in the MST.

2.2 Part 2

This is the opposite conditional of the minimum cut property that we had proven in class that states the lightest edge of some cut must be in some MST. If e is the lightest edge across some cut, then it must be in the MST, which you can see by running an MST algorithm. For example Prim's algorithm creates a tree and has a priority queue of nodes with their distance from the current tree. If you wanted to look at some cut as the current MST being created, then the lightest edge across the cut would be the closest node to the current tree, therefore if e is the lightest edge across some cut, it must in the MST.

2.3 Part 3

This conditional is not true if it must hold for every cycle in G. The example below has the cycle BCED, but the unique lightest edge BC would not be in the MST, because of the edges AB and AC.



2.4 Part 4

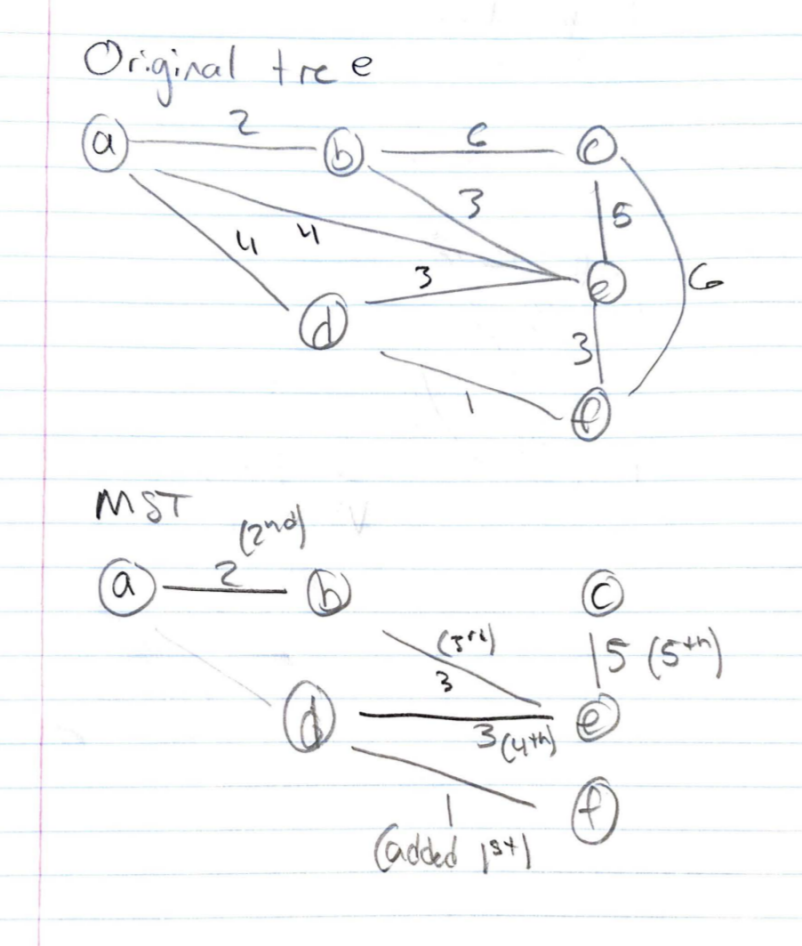
The given conditional must be true. If there is an r-path in G from s to t, then the MST, must have an r-path from s to t, because an r-path is **any** path from s-t that has edge weights **less than** r. We can prove this by contradiction. Say we have an r-path in G, but none in the MST. The MST by definition has some path from s to t, since it is spanning. This path also by definition is said to have minimum values, but if G has an r-path and the MST doesn't then the MST is including a path with larger edge weights than G (who has e $<$ r versus the MST with e $>=$ r). This would not meet the minimum definition requirement, and therefore the original claim must be true.

3 Problem 3 Kruskal's Algorithm

Algorithm by iteration:

1. FD (Cut (S = {F}, V - S))
2. AB (Cut (S = {F, D, A}, V - S))
3. BE (Cut (S = {F, D, A, B}, V - S))
4. DE (Cut (S = {F, D}, V - S))
5. CE (Cut (S = {A, B, D, E, F}, {C}))

Final MST:



4 Problem 4 Prim's Algorithm

1) Add DF D 0
F 0

2) DF F 0
E 3
A 4

3) DF E 3
A 4
C 6

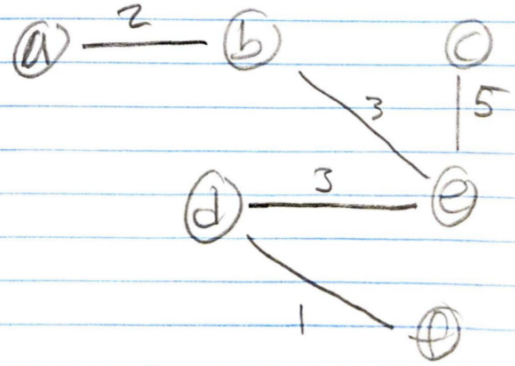
4) DF DE B 3
Add DE A 4
C 5

5) DF A 2
DE C 5
Add EB

6) DF C 5
DE
EB
Add BA

7) DF
DE
EB
BA
Add EC

Final MST



5 Problem 5

To compute a maximum spanning tree what we could do is take a minimum spanning tree algorithm such as Kruskal's Algorithm and run it on the graph with all edge weights multiplied by negative one. This would reverse any comparisons of edge weights that are made, so for example if you have two edges of weight 1 and 5, then in the standard minimum spanning tree you would choose the 1 weighted edge first, but when multiplied by negative 1, you get -1, and -5, which compels the minimum algorithm to select the -5 weight. The steps for the algoritm would be as follows:

1. Multiply each edge weight by -1, which would run in O(|E|) time.
2. Run Kruskal's on the graph using negative edge weights, which takes O(|E|log|V|).
3. THe output is the output of Kruskal's but you must multiply the edge weights by negative one again.

You could also ignore this method and just modify Kruskal's algorithm:

```
def Kruskal_Max(G, w) {
  A = empty set
  for all v in V {
    make_set(v)
  }
  G.E = G.E sorted in decreasing order by w
  for all (u, v) in G.E {
    if find_set(u) != find_set(v) {
      A = A union {(u,v)}
      union(u,v)
    }
  }
  return A
}
```

You know this algorithm must be correct, because you merely reverse the comparison operator from being small to large.