Design and Analysis of Computer Algorithms
CS 6363.005: Homework #3

Due on Wednesday November 9, 2016 at 11:59pm

Instructor: Benjamin Raichel

Hanlin He **(hxh160630)**
hanlin.he@utdallas.edu

Contents

Problem 1 Fractional Jewelry Collection

Part (a) Algorithm Description

Each time, take the whole jewelry with the largest $V[i]/S[i]$ value, until the bag's space is insufficient for the next whole jewelry. And then, cut the next largest $V[i]/S[i]$ value jewelry to fit the remaining bag's space and take it. As a result, the total value in the bag will be optimal.

Part (b) Proof of Correctness

Proof: Divide the bag into b units with each unit size of 1.

Assume each unit is the minimal size to fill, and each time we cut a piece of one jewelry of a unit size to put in the bag. Putting a whole jewelry into the bag equals to putting all its pieces with unit size into the bag.

Based on the description of part a, each time we put a piece into the bag, its value must be the largest among the remaining jewelry pieces, since the pieces with bigger value has already been taken.

Assume there is a optimal solution other than the greedy solution described in part a. Then, there is a unit size jewelry piece different in the choosing order, and its value should be larger than the piece chosen in the greedy solution. However, all jewelry with larger value has already been taken in the greedy solution. Thus, it cause a contradiction.

In conclusion, the greedy solution described in part a is optimal. □

Problem 2 Counting Paths

For any node s, let $PC(s,t)$ denote the paths count from s to t. If G is a dag, this function satisfies the recurrence:

$$PC(s,t) = \begin{cases} 1 & \text{if } s = t \\ 0 & \text{if } s \text{ is a sink.} \\ \sum_{s\to v} PC(v,t) & \text{otherwise.} \end{cases} \tag{2.1}$$

where $\sum_{s\to v} PC(v,t)$ is the sum of $PC(v,t)$ for all edges $s \to v$. In particular, if s is a sink but not equal to t, then $PC(s,t) = 0$, since there is no path from s to t.

The dependency graph for this recurrence is the input graph G itself: subproblem $PC(u,t)$ depends on subproblem $PC(v,t)$ if and only if $u \to v$ is en edge in G. Thus we can evaluate this recursive function in $\mathcal{O}(V + E)$ time by performing a depth-first-search of G, starting at s.

The algorithm is shown is algorithm 1, in which $PC[s]$ denote the paths count from s to t.

Problem 2 continued on next page...

Algorithm 1 Counting Paths Algorithm

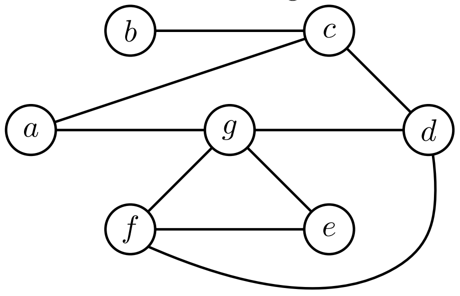
```
procedure CountingPaths($s, t$)
    if $s = t$ then
        return 1
    end if
    if $PC[s]$ is undefined then
        $PC[s] = 0$                                  ▷ Initially, whether $t$ is reachable from $s$ is unknown. ◁
        for each edge $s \to v$ do                   ▷ If $s$ is a sink, this For-Loop is skipped. ◁
            $PC[s] = PC[s] + $ CountingPaths $(v, t)$
        end for
    end if
    return $PC[s]$
end procedure
```

Problem 3 Running the Algorithms

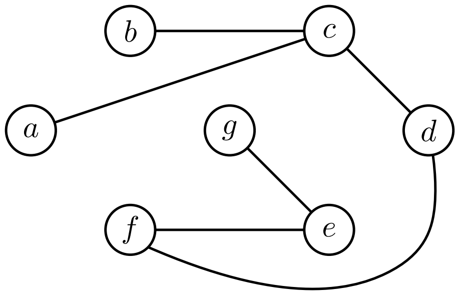
Part (a) DFS & BFS

The DFS&BFS from vertex a are shown in fig. 1 and fig. 2.

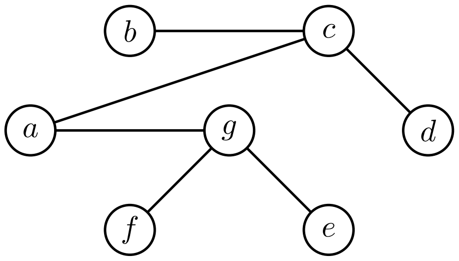
Figure 1: DFS & BFS for First Graph



Original Graph



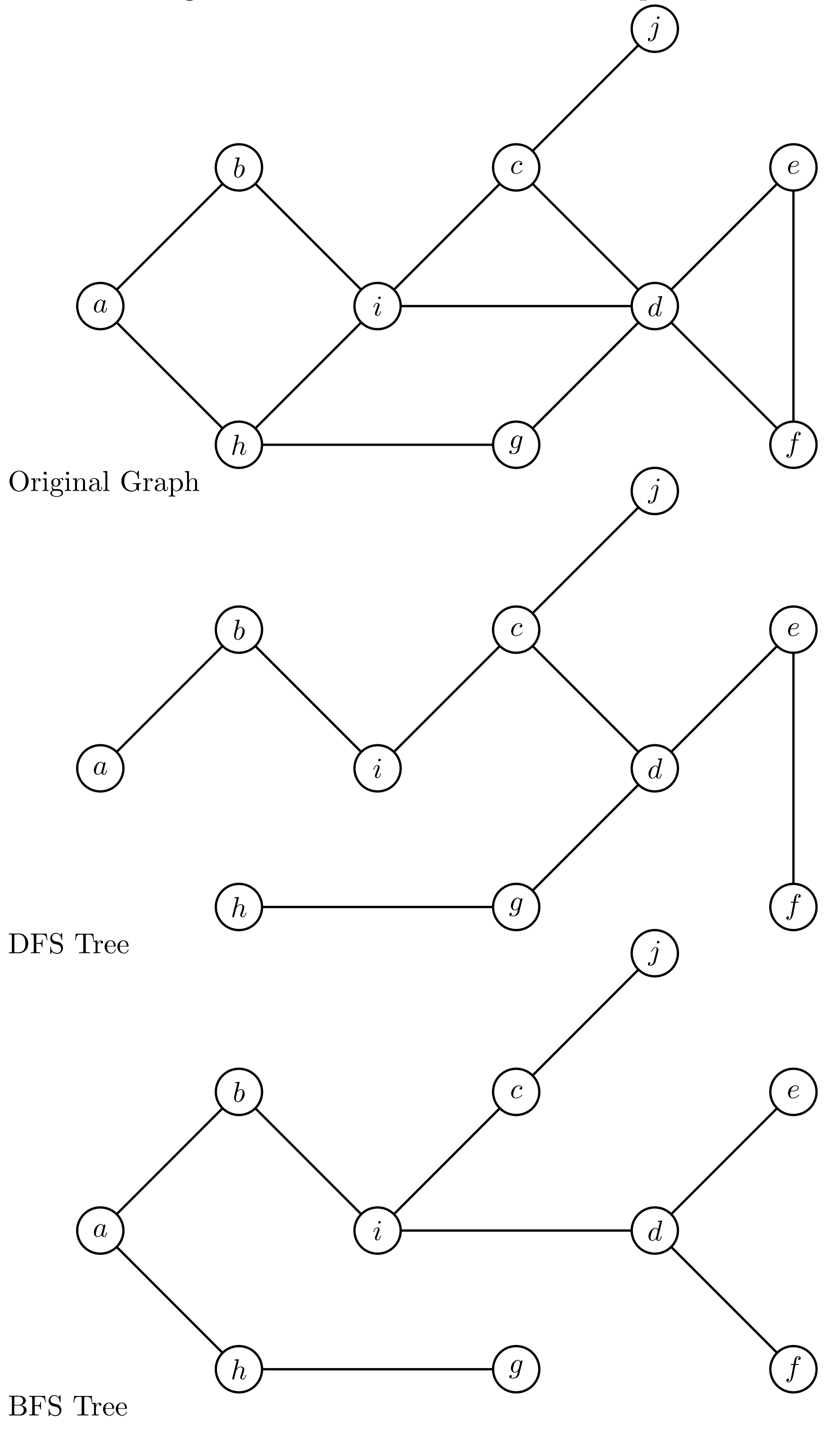
DFS Tree



BFS Tree

Problem 3 continued on next page. . .

Figure 2: DFS & BFS for Second Graph

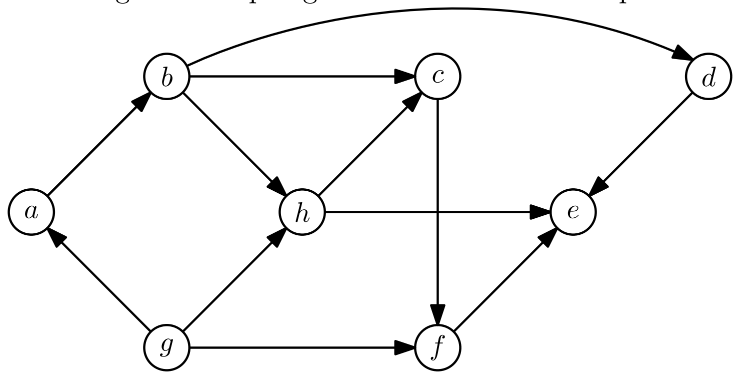


Problem 3 continued on next page. . .

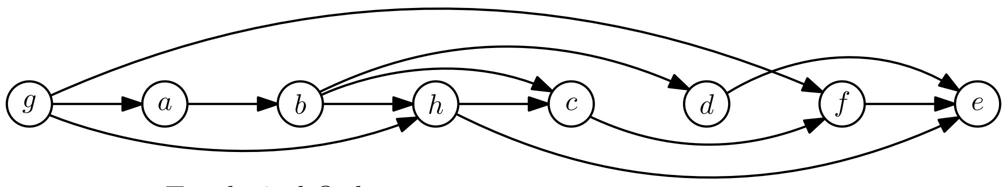
Part (b) Topological Ordering

The topological ordering of the vertices for each graph is shown in fig. 3 and fig. 4.

Figure 3: Topological Order for First Graph

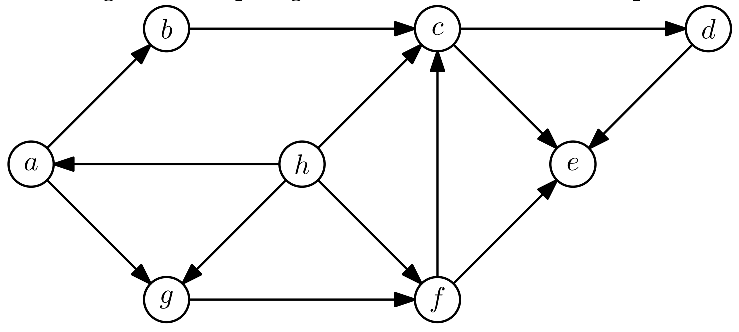


Original Graph

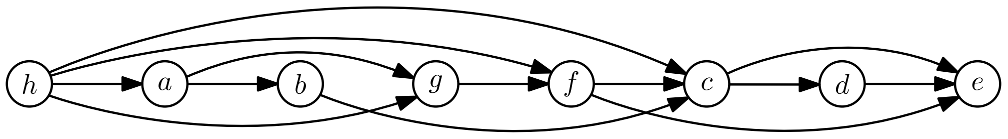


Topological Order

Figure 4: Topological Order for Second Graph



Original Graph



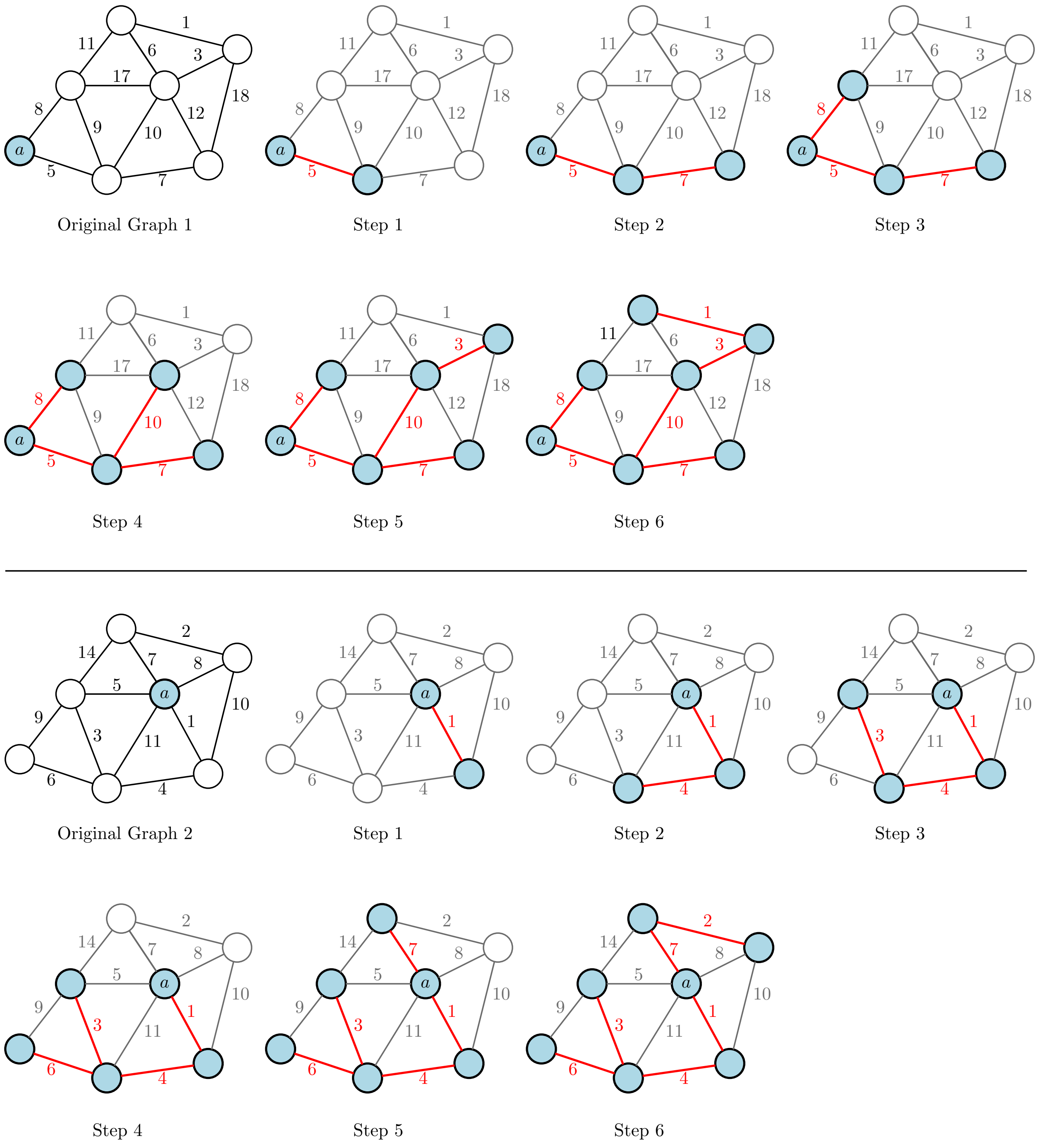
Topological Order

Problem 3 continued on next page. . .

Part (c) MST from Prim's Algorithm

The MST generation order by Prim's Algorithm is shown in fig. 5, from which we know that the weight of the fifth edge added for graph 1 is 3, for graph 2 is 7.

Figure 5: Prim's Algorithm Edge Added Order



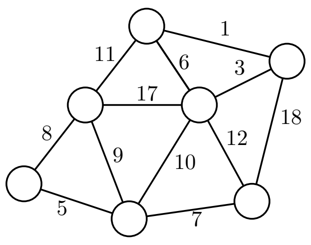
Note: 1. Circles in color means that the circle is marked in the MST.
2. Edges and Weights in RED is the edge weight taken in the MST.

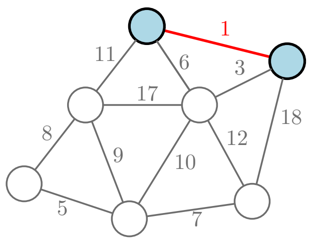
Problem 3 continued on next page. . .

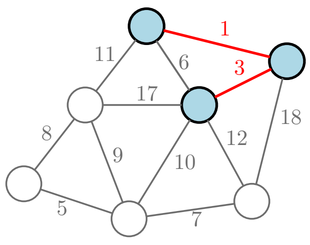
Part (d) MST by Kruskal's Algorithm

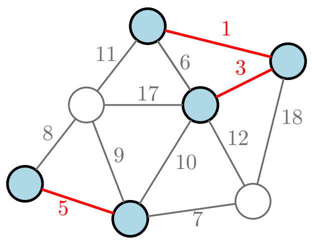
The MST generation order by Kruskal's Algorithm is shown in fig. 6, from which we know that the weight of the fifth edge added for graph 1 is 8, for graph 2 is 6.

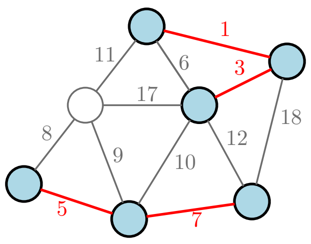
Figure 6: Kruskal's Algorithm Edge Added Order

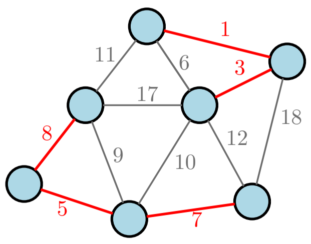

Original Graph 1

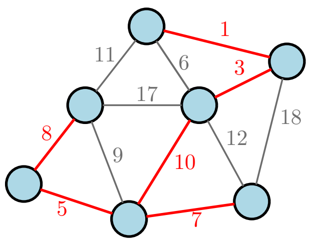

Step 1

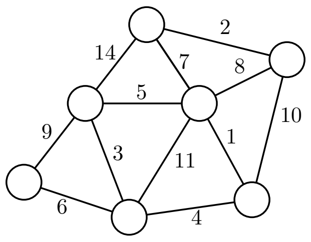

Step 2

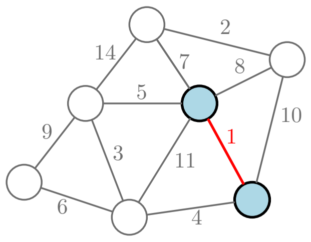

Step 3

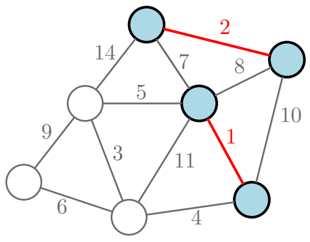

Step 4

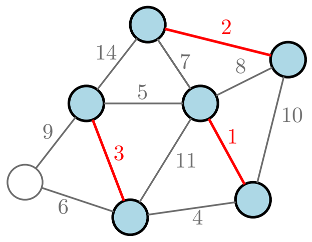

Step 5

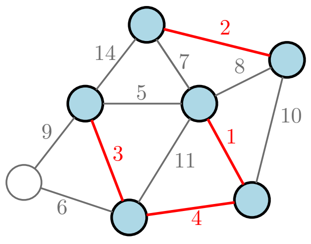

Step 6

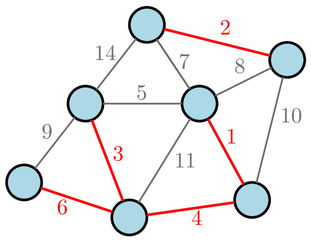

Original Graph 2

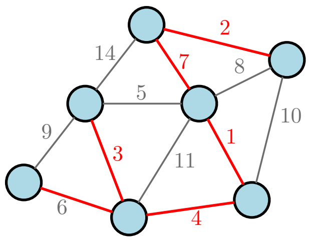

Step 1


Step 2


Step 3


Step 4


Step 5


Step 6

Note: Circles and edges in thick color means that the circle is marked and the edge is taken in the MST.

Problem 4 Minimum Spanning Tree Problem

Part (a) Maximum Weight Spanning Tree

Modify the MST Algorithms that when consider safe edge, take the maximum-weight edge with exactly one endpoint in that component. The result will be the maximum weight spanning tree. Take Kruskal's Algorithm as an example:

> Modified KRUSKAL: Scan all edges in decreasing weight order; if an edge is safe (which means the maximum-weight edge with exactly one endpoint in that component), add it to F.

Since the algorithm examine the edges in order from heaviest to lightest, any edge been examined is safe if and only if its endpoints are in different components of the forest F. To prove this, suppose the edge e joins two components A and B but is not safe. Then there must be a heavier edge e' with the same endpoint in A, which should have been examined in the previous steps. Hence result in contradiction, since previously examined edge always has both endpoints in the same component of F.

By using the Union Find data structure, the modified Kruskal's algorithm can be defined as algorithm 2:

Algorithm 2 Modified Kruskal's Algorithm to Compute Maximum Weight Spanning Tree

```
procedure MaxKruskal($V, E$)
    sort $E$ by decreasing weight order
    $F = (V, \emptyset)$
    for each vertex $v \in V$ do MakeSet ($v$)
    end for
    for $i = 1$ to $|E|$ do
        $uv$ = $i$th heaviest edge in $E$
        if Find ($u$) $\neq$ Find ($v$) then
            Union ($u, v$)
            add $uv$ to $F$
        end if
    end for
    return $F$
end procedure
```

The modified Kruskal's algorithm performs $\mathcal{O}(E)$ FIND operations, two for each edge in the graph, and $\mathcal{O}(V)$ UNION operations, one for each edge in the maximum weight spanning tree. Based on the operation time of Union Find, other than sorting, this algorithm's running is approximately $\mathcal{O}(E\alpha(V))$. Plus $\mathcal{O}(E\log V)$ sorting time, the total algorithm running time is $\mathcal{O}(E\log V)$, i.e. $\mathcal{O}(m\log n)$.

Problem 4 continued on next page...

Part (b) Minimum Feedback Edge Set

Based on the definition, if a spanning tree T is generated from the graph G, then, all the remaining edges and the remaining edges only will be in the feedback edge set F, i.e. $F = G \setminus T$ is the feedback edge set.

Claim: If T is a Maximum Weight Spanning Tree, then $F = G \setminus T$ is the minimum weight feedback edge set.

Proof: Take the modified Kruskal's algorithm as an example to compute the Maximum Weight Spanning Tree.

Let e be an edge in F. Suppose e is not the minimum choice in the minimum feedback edge set. Then there would be an edge t, which join two components A and B, and with a lighter weight than e, that if we add e to T, e and t will be in the same cycle. In another word, e should join the same two components A and B. Hence result in contradiction, since previously examined edge always has the heaviest weight, unless marked useless. If only e and t is possible edges to choose when connecting components A and B, the heavier edge e will be chosen.

Thus, edges in F will be the minimum choice. F is the minimum weight feedback edge set. □

Based on the argument above, the algorithm to compute minimum weight feedback edge set can be defined as algorithm 3:

Algorithm 3 Algorithm for Computing Minimum Weight Feedback Edge Set

```
1: procedure MinFES($V, E$)
2:     $T$ = MaxSpanningTree ($V, E$)          ▷ Use algorithm to compute the MaxST $T$ ◁
3:     return $F = E \setminus T$   ▷ The set-theoretic difference between $E$ and the MaxST $T$ is the result ◁
4: end procedure
5: procedure MaxSpanningTree($V, E$)
6:         ▷ This algorithm to compute Maximum Spanning Tree could be any algorithm that go the job. ◁
7:     return MaxKruskal ($V, E$)          ▷ For example, MaxKruskal ($V, E$) will do. ◁
8: end procedure
9: procedure MaxKruskal($V, E$)          ▷ This procedure is the same as algorithm 2 ◁
10:    sort $E$ by decreasing weight order
11:    $F = (V, \emptyset)$
12:    for each vertex $v \in V$ do MakeSet ($v$)
13:    end for
14:    for $i = 1$ to $|E|$ do
15:        $uv$ = $i$th heaviest edge in $E$
16:        if Find ($u$) $\neq$ Find ($v$) then
17:            Union ($u, v$)
18:            add $uv$ to $F$
19:        end if
20:    end for
21:    return $F$
22: end procedure
```

Part (c) Update MST

Removing e in the MST T induces a bi-partition, and e would be the lightest edge that joins the two components. Let A and B be those two components.

If the weight of e is increased to e', then

STEP 1 Remove e' from T.

STEP 2 Find the lightest edge l that connect A and B among e' and the rest undecided edges.

STEP 3 Add l to T, the new tree T' is the MST for G'.

We could modify the BORVKA'S algorithm, so that in the beginning, set F as $T \setminus \{e\}$. The rest of the algorithm remain the same.

The modified Borvka's algorithm is defined as algorithm 4:

Algorithm 4 Modified Borvka's Algorithm

```
1: procedure MODIFIEDBORVKA($V, E, T'$)
2:     $F = T'$                                  ▷ $T'$ is the parameter passed in as $T \setminus \{e\}$ ◁
3:     $count$ = COUNTANDLABEL ($F$)            ▷ The rest part is the same as original algorithm. ◁
4:     while $count > 1$ do
5:         ADDALLSAFEEDGES ($E, F, count$)
6:         $count$ = COUNTANDLABEL ($F$)
7:     end while
8:     return $F$
9: end procedure
10:
11: procedure ADDALLSAFEEDGES($E, F, count$)
12:     for $i = 1$ to $count$ do
13:         $S[i] = NULL$
14:     end for
15:     for each edge $uv \in E$ do
16:         if $label(u) \neq label(v)$ then
17:             if $weight(uv) < w(S[label(u)])$ then
18:                 $S[label(u)] = uv$
19:             end if
20:             if $weight(uv) < w(S[label(v)])$ then
21:                 $S[label(v)] = uv$
22:             end if
23:         end if
24:     end for
25:     for $i = 1$ to $count$ do
26:         if $S[i] \neq NULL$ then
27:             add $S[i]$ to $F$
28:         end if
29:     end for
30: end procedure
```

Problem 4 continued on next page. . .

To use the modified Borvka's algorithm, the wrapper algorithm can be defined as algorithm 5.

Algorithm 5 Algorithm to Compute MST for G' Using Modified Borvka's Algorithm

```
1: procedure FindNewMST($e, newWeight$)
2:     $T' = T \setminus e$                              ▷ Remove $e$ from $T$. ◁
3:     $weight(e) = newWeight$                           ▷ Set $e$ to its new weight. ◁
4:     return ModifiedBorvka ($V, E, T'$)                ▷ Pass $T'$ to the modified Borvka's algorithm. ◁
5: end procedure
```

Running Time Analysis:

- CountAndLabel (F) takes $\mathcal{O}(|E|)$ time, since F has $\mathcal{O}(|E|)$ edge.
- AddAllSafeEdges ($E, F, count$) takes $\mathcal{O}(|E| + |V|)$ time.
- While-loop's actual iterations count is 1, since only one edge e was removed from T, there are only two component in F, i.e. $count = 2$.

In conclusion, the total running time is $\mathcal{O}(|E| + |V|)$.

TA's Feedback

P4-c's algorithm has some mistake. Final score: 99.