Problem 1

1. Given an algorithm taking inputs

    (a) $G = (V, E)$ a connected, undirected graph

    (b) $w : E \to \mathbb{Z}^+$ a weight function

    (c) $T \subseteq E$, a MST of G

    (d) $e_1 = \{u, v\} \notin E$, an edge no in G

    (e) $w_1 \in \mathbb{Z}^+$, a weight for e_1

    and outputs a MST T_1 for $G_1 = (V, E \cup \{e_1\})$ with $w(e_1) = w_1$. For full marks, your algorithm must be more efficient than computing a MST for G_1 from scratch. Justify that this is the case by analysing your algorithms worst-case running time. Finally, write a detailed proof that your algorithm is correct.

    Solution. □

    Algorithm

    (a) Let $G_1 = (V, E \cup \{e_1\})$, let $e_1 = (s, t)$ for some $s, t \in V$

    (b) Find the unique simple cycle c in G_1. Starting from s (or t, the choice is arbitrary), run DFS on G_1 with slight modification. When looking at vertex $u \in V$, Check if $v.color$ is $GRAY$. Break from DFS if true, otherwise continue DFS.

    (c) Find the largest weight edge e_2 in the cycle. Iterate through $G_1.E$, find $e_2 = (u, v) \in G_1.E$ such that $u.color$ and $v.color$ are both $GRAY$ (in the cycle) and $w(u, v)$ is maximum of all edges with $GRAY$ vertices (max weighted edge).

    (d) Return MST $T_1 = T \cup \{e_1\} \setminus \{e_2\}$

    Analysis

    (a) Since we are adding a constant time check at every while iteration and DFS itself has a run time of $O(V + E)$, the part of algorithm for finding a cycle (modified DFS) has a worst case running time of $O(V + E)$

    (b) The loop over all vertices to locate the maximum weight edge runs for $|E|$ iterations, each time doing a constant time operation. hence has a worst case running time of $O(E)$

    (c) Altogether the algorithm runs for $O(V+E)$, which is more efficient by computing MST from ground up which takes $O(E \lg V)$

```
Function DFS-Visit (G, u)
    u.color ← GRAY
    for v ∈ G_1.Adj[u] do
        if v.color is GRAY then
            Exit-DFS
        if v.color is WHITE then
            DFS-Visit(G, v)
    u.color ← BLACK
Function Find-MST-1 (G, w, T, e_1, w_1)
    G_1 ← (V, E = E ∪ {e_1}) with w : E → ℝ and w(e_1) = w_1
    (s, t) ← e_1
    for v ∈ V do
        v.color ← WHITE
    DFS-Visit(G_1, s)
    max_w ← 0
    e_2 ← NIL
    for (u, v) ∈ G_1.E do
        if u.color = GRAY and v.color = GRAY and w(u, v) > max_w then
            max_w ← w(u, v)
            e_2 ← (u, v)
    T_1 ← T ∪ {e_1} \ {e_2} return T_1
```

Lemma. *The graph $G_1 = (V, E = E \cup \{e_1\})$ has a unique simple cycle c, and e_1 is in c*

Proof. Let $T \subseteq E$, and $e_1 = (s, t)$. Since T is MST, s is reachable from t, i.e. exists a path p such that $t \overset{p}{\rightsquigarrow} s$. Consider $E_1 = E \cup \{e_1\}$, we have $c = s \to t \overset{p}{\rightsquigarrow} s = \langle v_0 = s, \cdots, v_k = s \rangle$. Therefore e_1 is in c. Now we prove c is unique. This is equivalent to proving that p is unique, since if adding e_1 to E produces more than 1 cycle, then there must exists p' such that $p' \neq p$. This is not possible since $s \overset{p}{\rightsquigarrow} t \overset{p'}{\rightsquigarrow} s$ forms a cycle, which is not possible in MST T. Also, c is simple because T is simple. $\square$

Proof of correctness

Proposition. *The output T_1 is a MST for $G_1 = (V, E = E \cup \{e_1\})$*

Proof. By correctness of DFS, when we first explored (u, v), if $v.color$ is $GRAY$, then (u, v) is a back edge, indicating we have found a cycle c in the graph. By the previous lemma and the fact that we started from s where $e_1 = (s, t)$, we will always find such a cycle (i.e. discover t such that $s \in G_1.Adj[t]$ and $s.color$ is $GRAY$). When Exit-DFS is called, the vertices $v \in V$ such that $v.color$ is $GRAY$ represents vertices constituting the cycle c. The claim holds since by the time EXIT-DFS is called, all ancestors of t has yet to finish (i.e. setting their color to $BLACK$). The subsequent loop over $G_1.E$ finds the maximum weighted edge e_2 in the cycle c. By the cycle property of MST, e_2 cannot be included in any MST of G_1. Now consider the return value $T_1 = T \cup \{e_1\} \setminus \{e_2\}$. Now we prove T_1 is a MST of G_1. Consider $A = T \setminus \{e_2\}$, note A breaks into 2 connected components as MST T is connected. Let $C = (P, Q)$ be the cut where $s \in P$ and $t \in Q$, $P \cup Q = A$ and $P \cap Q = \emptyset$. The fact that $e_2 \in T$ implies that e_2 is a light edge cross the cut C. Now we have $w(e_1) < w(e_2)$ since e_2 is the maximum weight edge in the cycle c, therefore e_1 is the light across the cut C. By corollary in CLRS, since P respects the cut C and e_2 is a light edge crossing the cut and $P \subseteq T$, therefore e_2 is safe for P. Therefore T_1 is some MST of G_0 □

2. Give an efficient algorithm that takes the following inputs:

   (a) $G = (V, E)$ a connected, undirected graph
   (b) $w : E \to \mathbb{Z}^+$ a weight function
   (c) $T \subseteq E$, a MST of G
   (d) $e_0 \in E$, an edge in G

   and that outputs a minimum spanning tree T_0 for the graph $G_0 = (V, E \setminus \{e_0\})$, if G_0 is still connected your algorithm should output the special value Nil if G_0 is disconnected. For full marks, your algorithm must be more efficient than computing a MST for G_0 from scratch. Justify that this is the case by analysing your algorithms worst-case running time. Finally, write a detailed proof that your algorithm is correct. (Note that this argument of correctness will be worth at least as much as the algorithm itself.)

   Solution. □

   Algorithm

   (a) Keep track of the cut $C = (P, Q)$ such that $P \cup Q = E \setminus \{e_0\}$
      i. Make set on every $v \in V$
      ii. Iterate over $T \subseteq E$, combine the sets if there is an edge connecting the two connected components
   (b) Find the set of edges $E' \subseteq E$ such that for all $e \in E'$, e crosses the cut C. This can be achieved by checking all $e = (u, v) \in E$ against cut C and see if the u and v are in different connected components
   (c) Find the light edge e_2 (lowest weight edge crossing the cut C) from the E'
   (d) Return MST $T_0 = T \setminus \{e_0\} \cup \{e_2\}$ if e_2 exists otherwise return Nil

```
Function Find-MST-2 (G, w, T, e_0)
    G_0 = (V, E \ {e_0})
    for v ∈ V do
        Make-Set(v)
    for (u, v) ∈ T do
        Union-Set(u, v)
    E' ← ∅
    for (u, v) ∈ G_0.E do
        if Find-Set(u) ≠ Find-Set(v) then
            E' ← E' ∪ {(u, v)}
    if E' = ∅ then
        return Nil
    min_w ← ∞
    e_2 ← Nil
    for e ∈ E' do
        if w(e) ≤ min_w then
            min_w ← w(e)
            e_2 ← e
    T_0 ← T \ {e_0} ∪ {e_2}
    return T_0
```

Proof of correctness

Proposition. *The algorithm returns T_0, a MST for $G_0 = (V, E \setminus \{e_0\})$ if exists,* NIL *otherwise*

Proof. Note $G = (V, E)$ has MST T implies G is connected. The removal of an edge e_0 disconnects E into 2 connected components. Let $C = (P, Q)$ be the cut such that $P \cup Q = E \setminus \{e_0\}$. $C = (P, Q)$ is kept track of in sets, where all $e_1 \in P$ belongs to one set and all $e_2 \in Q$ belongs to another. E' hence contains edges $e = (u, v) \in E$ such that $u \in P \wedge v \in Q$ or $v \in P \wedge u \in Q$. If there is no such edges other than e_0 that crosses the cut C, then E' is empty and G_0 is disconnected. In this case the algorithm returns NIL, which is correct. Otherwise we find a light edge e_2 and returns $T_0 = T \setminus \{e_0\} \cup \{e_2\}$. Here we claim T_0 is some MST of G_0. By corollary given in CLRS, since P respects the cut C and e_2 is a light edge crossing the cut and $P \subseteq T$, therefore e_2 is safe for P. Therefore T_0 is some MST of G_0 □

Problem 2

Consider the following MST with Fixed Leaves problem:

1. Input: A weighted graph $G = (V, E)$ with integer costs $c(e)$ for all edge $e \in E$, and a subset of vertices $L \subseteq V$

2. Output: A spanning tree T of G where every node of L is a leaf in T and T has the minimum total cost among all such spanning trees.

1. Does this problem always have a solution? In other words, are there inputs G, L for which there is no spanning tree T that satisfies the requirements? Either provide a counter-example (along with an explanation of why it is a counter-example), or give a detailed argument that there is always some solution.

   Solution. □

   The problem does not always have a solution. Consider $G = (V, E)$ where $V = \{1, 2, 3, 4\}$ and $E = \{(1, 2), (1, 3), (1, 4)\}$ with $L = \{1\}$ and arbitrary weights. Since there is only 3 edges which happen to connect to all 4 vertices, there is only one MST $T = E$ for G. A non-root node in a acyclic graph (tree) implies that that there is only one parent. Since vertex 1 connects to 3 other vertices, it cannot be non-root, implying it must be root. Therefore we have a counter example where the only MST possible $T = E$ do not have $v \in L$ as leaves.

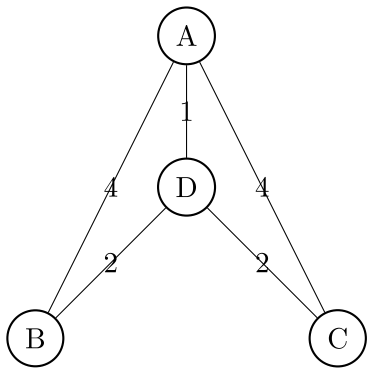
2. Let G, L be an input for the MST with Fixed Leaves problem for which there is a solution.

   (a) Is every MST of G an optimal solution to the MST with Fixed Leaves problem? Justify.

   (b) Is every optimal solution to the MST with Fixed Leaves problem necessarily a MST of G (if we remove the constraint that every node of L must be a leaf)? Justify.

   Solution. □

   Both claim are incorrect. Here we provide a counter example. Consider $G = (V, E)$ where $V = \{A, B, C, D\}$ and $E = \{(A, C), (A, B), (D, A), (D, B), (D, C)\}$ and weights labelled in the graph. Let $L = \{D\}$



There is only one MST $T = \{(D, A), (D, B), (D, C)\}$ for G with $w(T) = 4$. There is only one MST $T' = \{(A, C), (A, B), (D, A)\}$ given fixed leaves L for G with $w(T) = 9$. Note $T \neq T'$ and $w(T) < w(T')$.

(a) The existence of T, a MST of G but not the a solution to MST with fixed leaves problem, disproves this claim

(b) The existence of T', a MST of G with fixed leaves problem but not an MST for G, disproves this claim

3. Write a greedy algorithm to solve the MST with Fixed Leaves problem. Give a detailed pseudo-code implementation of your algorithm, as well as a high-level English description of the main steps in your algorithm. What is the worst-case running time of your algorithm? Justify briefly.

Algorithm

(a) Let $S = V \setminus L$

(b) Find MST for graph induced by S, let T_S be the ouput MST for S

(c) For each (u, v) where $u \in L$ and $v \in S$, adds the lowest weight edge for each u to T_S. In other words, find the light edge crossing the cut $(\{u\}, T_S)$ and adds it to T_S

(d) Return the augmented T, consists of T_S and edges added to it in the previous step

Analysis

(a) Assume sets S and T membership check can be done in $O(1)$ time, this is possible if keep an extra bit in adjacency list representation of the graph. Assume insertion to array T takes $O(1)$ time.

(b) MST-Kruskal has worst case running time of $O(E \lg V)$

(c) The for loop executes $O(E)$ times altogether, since sum of lengths of all adjacency list is $2|E|$ for undirected graphs and $|L| \leq |V|$. In each iteration, membership check and potential assignment operation takes $O(1)$.

(d) The entire algorithm has a worst time running time of $O(E \lg V)$

```
Function Find-MST-Fixed-Leaves (G, w, L)
    S ← V \ L
    G′ ← induced by S
    T ← MST-Kruskal(G′, w)
    for u ∈ L do
        e_light ← Nil
        w_light ← ∞
        for v ∈ G.Adj[u] do
            if v ∈ S and w(u, v) < w_light then
                w_light ← w(u, v)
                e_light ← (u, v)
        T ← T ∪ {e_light}
    return T
```

4. Write a detailed proof that your algorithm always produces an optimal solution.

Lemma. *Given condition provided by the algorithm, provided that there is a valid solution for fixed leaves problem, then for any $u \in L$, there exists some $v \in S = V \setminus L$ such that $(u, v) \in E$.*

Proof. Prove by contradiciton. Assume there exists $u \in L$ such that for all $(u, v) \in E' = E$, we have $v \in L$. Pick any appropriate edge $(u, v) \in E'$ to be in some valid solution MST T. Now vertex u and v has one edge incident on it. We cannot include any other edge such that they incident on u or v since otherwise u or v would not be leaves (i.e. have degree of more than 1). Therefore $\{u, v\}$ is isolated from the rest of the vertices, hence no viable MST exists possibly as a solution. Since we are given there is some solution to the problem, the claim thus holds □

Proposition. *Given $G = (V, E)$ with cost function $c : E \to \mathbb{R}$ and a subset $L \subseteq V$, the algorithm returns MST T where every $u \in L$ is a leaf and T has the minimum cost amongst such spanning trees*

Proof. By correctness of Kruskal's algorithm, MST-Kruskal terminates and returns T_S, which is a MST for $S = V \setminus L$.

(a) Prove every $u \in L$ is a leaf in T. The algorithm finds and adds the least weight edge connecting u to some $v \in S$. Since T_S spans S and the fact that the previous lemma holds, such operation is always possible. Since the algorithm only adds one, specifically e_{light}, to T for each $u \in L$, u has degree of one, hence are leaves in MST T

(b) Prove by contradiction that the algorithm returns a MST amongst such spanning trees. Assume there is some other spanning tree T' such that $w(T) > w(T')$. We can decompose $T' = T'_S \cup T'_O$ where T'_S is a subset of T' and spans S and T'_O be the rest of edges in T', specifically T'_O consists of edges that crosses the cut (L, S).

i. By correctness of MST-KRUSKAL we have

$$w(T_S) \leq w(T'_S)$$

ii. After finding T_S, the algorithm then construct the solution MST T by adding (u, v) to T_S where $v \in S$ for each $u \in L$ (Note this is always possible with previous lemma) such that $w(u, v)$ is minimized. Denote E' be such set of edges added to T_S Therefore

$$\sum_{e \in E'} w(e) \leq w(T'_O)$$

Althgether we have

$$w(T) = w(T_S) + \sum_{e \in E'} w(e) \leq w(T'_S) + w(T'_O) = w(T')$$

which contradicts the assumption that $w(T) > w(T')$. Therefore the claim holds

This completes the proof □

Problem 3

An edge in a flow network is called critical if decreasing the capacity of this edge reduces the maximum possible flow in the network. Give an efficient algorithm that finds a critical edge in a network. Give a rigorous argument that your algorithm is correct and analyse its running time.

Solution. □

Algorithm

1. Find a maximum flow f for G, let G_f be the residual flow network for G given f
2. Return the set of edges $E' \subseteq E$ for which $f(u, v) = c(u, v)$, $(u, v) \in E'$

Analysis

1. FORD-FULKERSON runs in $O(VE)$
2. The for loop iterates $|E|$ times, doing a constant $O(1)$ operation to update E'. Suppose E" is implemented with an array, the operation to insert an element at the end takes $O(1)$ time. Therefore the loop has a worst time running time of $O(E)$

3. Altogether the algorithm has a worst case running time of $O(VE)$

```
Function Find-Critical-Edge (G, s, t)
    f ← Ford-Fulkerson(G, s, t)
    E′ ← ∅
    for (u, v) ∈ G.E do
        if f(u, v) = c(u, v) then
            E′ ← E′ ∪ {(u, v)}
    return E′
```

Proof of Correctness

Lemma. *Given a maximum flow f for flow network G and its residual flow network G_f. If any $(u,v) \in E$ such that $f(u,v) = c(u,v)$, then (u,v) is in the cut-set of some cut $C = (S,T)$, where $s \in S$, $t \in T$, and such that $|f| = c(S,T)$.*

Proof. Prove by contradiction. Assume there is no such cut C where (u,v) crosses $|f| = c(S,T)$. Then it must be that for all cut $C' = (S',T')$ that (u,v) crosses, $|f| < c(S,T)$ by the upper-bound property, therefore

$$|f| = f(S',T') = \sum_{u \in S'} \sum_{v \in T'} f(u,v) - \sum_{u \in S'} \sum_{v \in T'} f(v,u) < c(S,T) = \sum_{u \in S'} \sum_{v \in T'} c(u,v)$$

This implies that there is some other edge (x,y) in the cut-set of C' such that $f(x,y) < c(x,y)$. Since C' is an arbitrary and the above claim holds for all cuts, we can construct an augmenting path $p = \langle v_0, \cdots, v_k \rangle$ in G_f by picking appropriate cuts such that $f(v_{i-1}, v_i) > 0$ for all $i = 1, \cdots, k$. This contradicts with the assumption that the algorithm for finding max flow terminates, specifically when there is no augmenting paths left. Hence the claim holds. □

Proposition. *Given a maximum flow f for flow network G and its residual flow network G_f. the set of edges $E' \subseteq E$ for which $f(u,v) = c(u,v)$, $(u,v) \in E'$ is the set of critical edges*

Proof. Given assumption of the proposition and previous lemma, we have (u,v) in some cut $C = (S,T)$ such that $|f| = c(S,T)$. Since algorithm for finding the max flow f terminates, by the max-flow min-cut theorem, the cut C has minimal capacity, which bounds the value of the flow $|f|$. Hence, a reduction in $c(u,v)$ causes reduction in $C(S,T)$, and ultimately the maximum value a flow can achieve. Therefore (u,v) is a critical edge. Since (u,v) is an arbitrary edge in E', E' is the set of critical edges. □