COMP 3804 — Assignment 3

Question 1: Daniil Kulik 101138752

Question 2: For this question we assume that DepthFirstSearch procedure studied on lectures sets pre- and post-visit numbers for each vertex while running.

Algorithm 1: Test if the given edge is an cycle edge

```
Input G, e
Result: Returns True if e is a cycle edge, False otherwise
DepthFirstSearch(G);
forall v ∈ G.V do
    forall (v, u) ∈ G.E do
        if not (pre(v) < pre(u) < post(u) < post(v)) and
         (pre(u) < pre(v) < post(v) < post(u)) then
            if (v, u) = e then
                return True;
            end
        end
    end
end
return False
```

Now let's find the running time of this procedure.

1. From lectures we now that running DepthFirstSearch routine takes $|V| + |E|$ time.
2. The worst case for the while loop appears when there is not a cycle edge in the graph. Thus, it will be executed $|V| + |E|$ times at most.

Therefore, the total running time of the algorithm:

$$T(n) = |V| + |E| + |V| + |E| = 2(|V| + |E|) = O(|V| + |E|)$$

Question 3: Solid edges are tree edges.

(1) See Figure 1
Tree edges: {(A,B),(B,C),(C,D),(D,H),(H,G),(G,F),(F,E)}
Forward edges: {(A,F),(B,E)}
Back edges: {(D,B),(E,D),(E,G),(F,G)}
Cross edge: $\emptyset$

(2) See Figure 2
Tree edges: {(H,G),(G,F),(F,E),(E,D),(D,B),(B,C)}
Forward edges: $\emptyset$

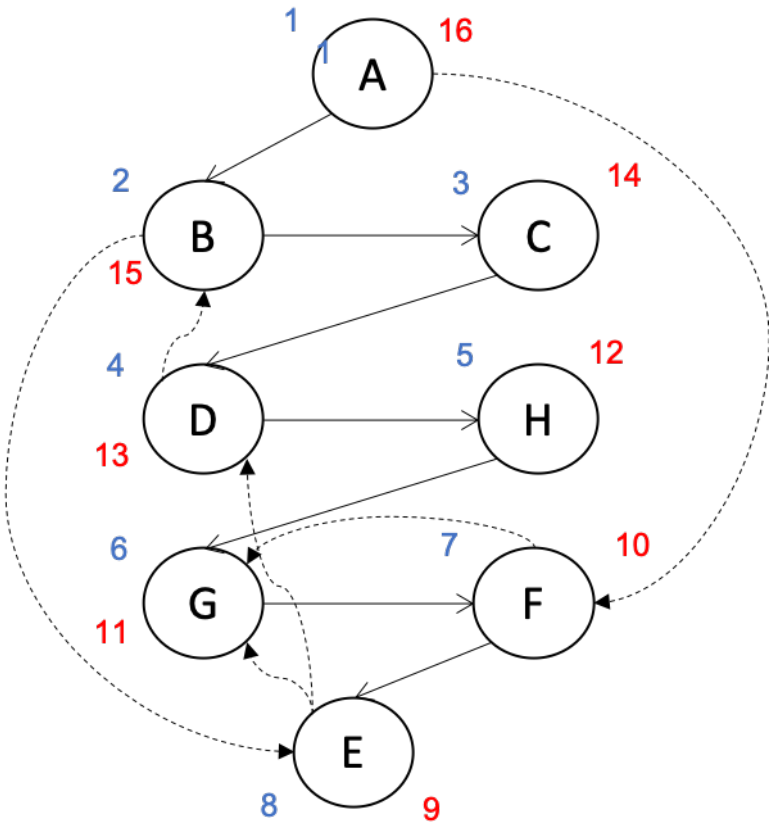

Figure 1: DFS-forest alphabetically first (blue is pre, red is post)

Back edges: {(F,G),(E,G),(D,H),(B,E),(C,D)}
Cross edge: {(A,F),(A,B)}

Question 4: We will test both algorithms on the graph 3 and path from 1 to 5.

(1) This algorithm will work correctly with negative weights. We can prove it by induction. However, first we need to show that three claims we saw on lectures still hold with negated weights.

Claim 1 At any moment $\delta(v_i) \leq d(v_i)$, proof: at the start of the algorithm $d(v_i) = \infty$ or $d(v_i)$ is the length of some path from s to v_i

Claim 2 If we assume that at some point $\delta(v_i) = d(v_i)$ then $d(v_i)$ does not change. Proof: Claim 1 $d(v_i)$ only gets smaller until it becomes $\delta(v_i)$

Claim 3 Let's consider the shortest path from s to v_i. In order to find $\delta(s, v_i)$ we assume that we correctly found $\delta(s, v_j)$ such that at the of iteration i we will have $d(v_i) = \delta(s, v_i)$. By the property of the shortest path, the path from s to v_i goes through v_j and $\delta(s, v_i) = \delta(s, v_j) + wt'(v_j, v_i)$

$$d(v_i) \leq d(v_i) + wt'(v_j, v_i) = \delta(s, v_j) + wt'(v_j, v_i) = \delta(s, v_i)$$

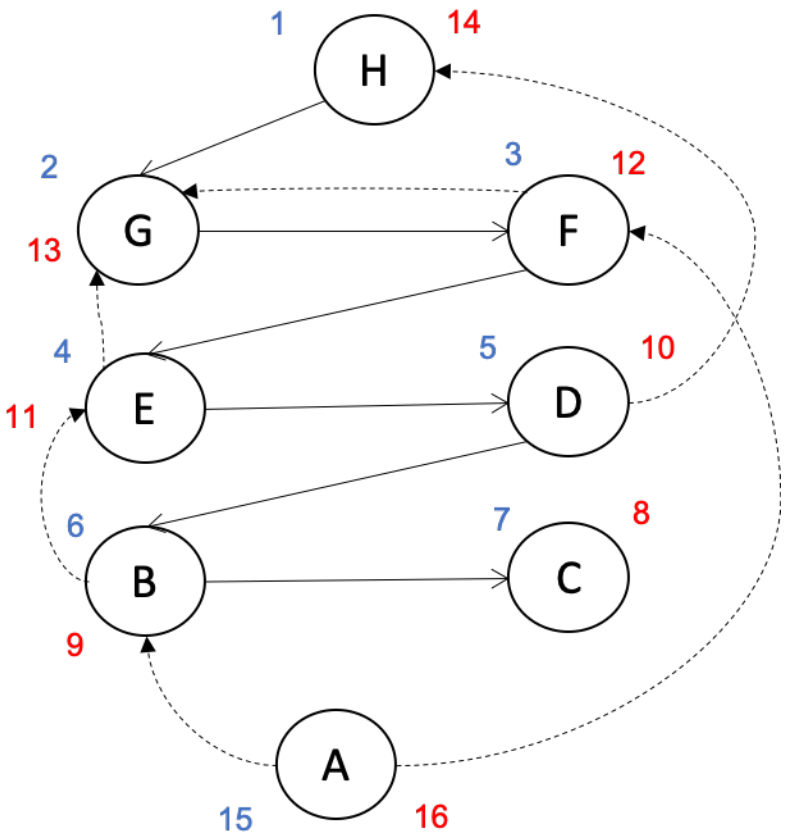

Figure 2: DFS-forest alphabetically last (blue is pre, red is post)

---

since $wt'(v_j, v_i)$ is negative

$$\delta(s, v_i) \leq d(v_i)$$

Therefore, $\delta(s, v_i) = d(v_i)$

Basis step: $d(s) = 0 = \delta(s, s)$, by claim 2 it will not change.
Inductive Hypothesis: Assume at the end of iteration k $d(v_k) = \delta(v_k) \rightarrow$ at the end of iteration $k+1$ $d(v_{k+1}) = \delta(v_{k+1})$. Consider iteration $k+1$ such that the last edge from s to v_{k+1} is (v_k, v_{k+1})
1. $d(v_k) = \delta(v_k)$ by assumption
2. $d(v_{k+1}) = \delta(s, v_k) + wt'(v_k, v_{k+1}) = \delta(v_{k+1})$ by the property of the shortest path
3. $d(v_{k+1})$ will not change by claim 2

Let's look at the example graph in Figure 3 and run the **ShortestPathAcyclic**. We expect to get this path: $[(1,2),(2,3),(3,4),(4,5)]$ with total weight -17.

Step 1 Topologically sort see Figure 4
Step 2 $d(1) = 0, d(2) = d(3) = d(4) = \infty$
Step 3 $d(1) = 0, d(2) = -3, d(3) = -9, d(5) = -12, d(4) = \infty$
Step 4 $d(1) = 0, d(2) = -3, d(3) = -10, d(4) = -8, d(5) = -12$

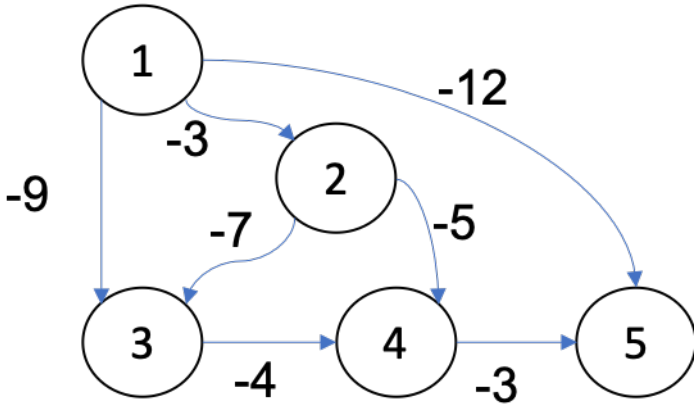

Figure 3: Example graph with negated weights

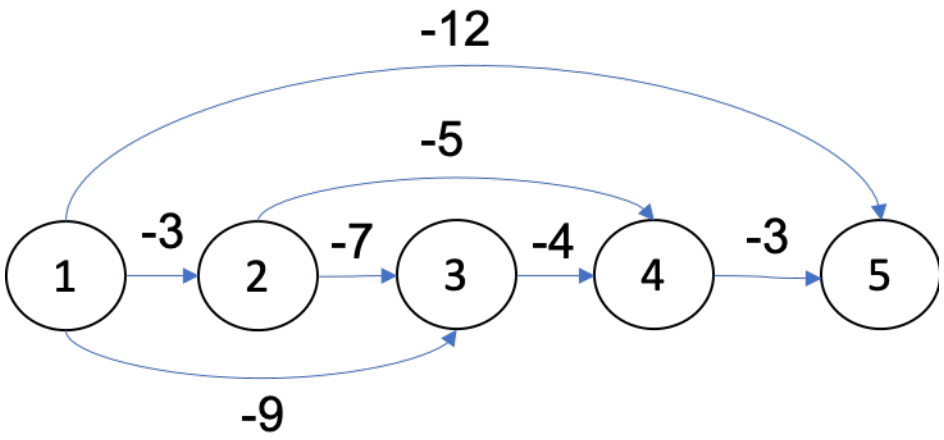

Figure 4: Topologically sorted

Step 5 $d(1) = 0, d(2) = -3, d(3) = -10, d(4) = -14, d(5) = -12$
Step 6 $d(1) = 0, d(2) = -3, d(3) = -10, d(4) = -14, d(5) = -17$
The algorithm give us the correct answer $d(1, 5) = -17$ and it is the longest possible path if we negate the weights back.

(2) Dijkstra's algorithm will not work properly with negative weights. We proved its correctness with a claim (Claim 4 for Dijkstra's algorithm) the minimum value in Q never changes. However, if we use negative weights then it's possible that during some iteration $d(v)$ for $v \in Q$ will decrease. Thus, the d values will be inconsistent and will not reflect the correct weights for shortest path, because on the next iteration where will be $d(u) < d(v)$
Let's look at Figure 3 and find the shortest path from 1 to 5 with Dijkstra's algorithm.

$S = \emptyset$ and $Q = \{1, 2, 3, 4, 5\}$. $d(1) = 0$, $d(2) = d(3) = d(4) = d(5) = \infty$

Step 1 $S = \{1\}$, $Q = \{2, 3, 4, 5\}$, $d(1) = 0, d(2) = -3, d(3) = -9, d(5) = -12, d(4) = \infty$

Step 2 $S = \{1, 5\}$, $Q = \{2, 3, 4\}$, $d(1) = 0, d(2) = -3, d(3) = -9, d(5) = -12, d(4) = \infty$

Step 3 $S = \{1, 5, 3\}$, $Q = \{2, 4\}$, $d(1) = 0, d(2) = -3, d(3) = -9, d(5) = -12, d(4) = -13$

Step 4 $S = \{1, 5, 3, 4\}$, $Q = \{2\}$, $d(1) = 0, d(2) = -3, d(3) = -9, d(5) = -16, d(4) = -13$

Step 5 At this step we will update $d(3)$ because $d(2) + (-7) < d(3)$ so $d(3) = -10$ and the path is $1 \to 2 \to 3 \to 4 \to 5$ giving us the total cost of $d(5) = -16$ while it should be -17.

Therefore, Dijkstra's algorithm will not work properly with negative weights.

Question 5:

(1) Let's assume that G is a connected undirected graph with $|V| \geq 3 \Rightarrow |E| \geq |V| - 1$. Otherwise, we have a trivial case when there is only one edge and the second smallest edge does not exist. If the graph meets the condition $|V| \geq 3 \Rightarrow |E| \geq |V| - 1$ than we can apply Kruskal's algorithm to construct a MST. We demonstrated the algorithm's correctness on lectures.

Kruskal's algorithm constructs an MST by repeatedly adding the lightest edge that does not produce a cycle. Since the given graph is connected undirected graph we can apply the algorithm. Therefore, the second smallest edge does not produce a cycle between connected components it will be picked by Kruskal's algorithm and will be an edge in the resulting MST. We also notice that two edges and three vertices cannot produce a cycle in a connected undirected graph (e.g they create an MST). Thus, we can guarantee that the second smallest edge will be in the MST.

(2) Let's assume that G is a connected undirected graph with $|V| \geq 4 \Rightarrow |E| \geq |V| - 1$. Otherwise, we have a trivial case when there is only two edges and the third smallest edge does not exist. In this case picking up the third smallest edge is not guaranteed because three edges and four vertices can produce a cycle in a graph. Let's look at Figure 5 where the third smallest element produces a cycle and by definition an MST does not contain cycles.

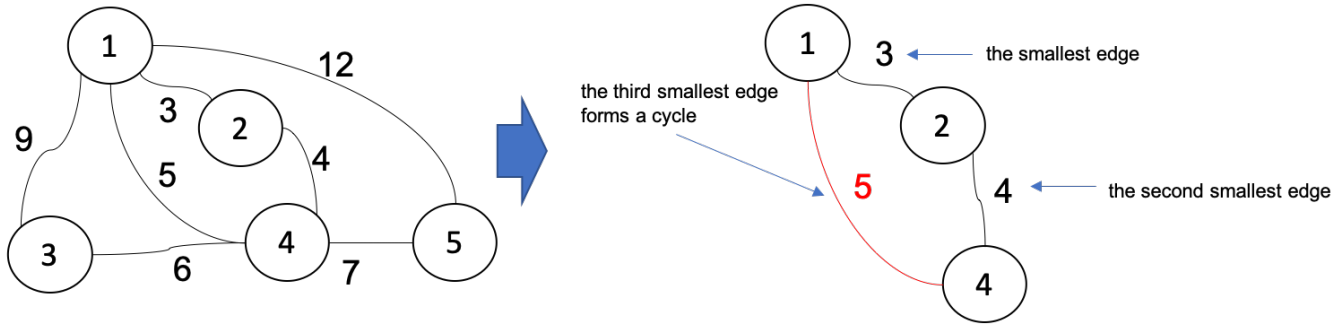

Figure 5: Example graph

This counterexample demonstrates that the edge with the third smallest weight is not an edge in the minimum spanning tree of G.

Question 6: If we make the data structure to store only name it will not affect $find(x)$ implementation and its running time. It still be $O(1)$.

Now let's look how we can make $Union(A, B, C)$ still have $O(min(|A|, |B|)$ running time without storing size. The code below shows how it can be achieved with one loop where we decide which set is smaller.

Let's assume that **Merge(C, SmallerList, BiggerList)** makes the smaller list pointing to the head of the bigger list, exactly like the original **Union** operation but now we know which list is smaller.

```
UnionNew(A,B,C):
    NextA = A.Head
    NextB = B.Head
    while NextA != A.Tail and NextB != B.Tail do
        NextA = NextA.Next
        NextB = NextB.Next

    if NextB = B.Tail then
        Temp = A
        A = B
        B = Temp

    Merge(C, A, B)
    return C
```

Let's analyse the running time of the updated algorithm. By default, we assume that A has smaller size.

1) The while loop follows each linked list's pointers. The loop stops when any list reaches the last element. When the loop stops we know that one list contains k elements in total and the other one has $n-k$, where n is the number of elements . Thus, we can state that the list with k elements is smaller or equal in size. The following if statements checks if B reaches the end then it has smaller (or equivalent) size than A, otherwise A is smaller. Therefore, the running time of the while loop is $min(|A|, |B|)$
2) Swapping variables takes constant time c
3) From lectures we know that it takes $O(min(|A|, |B|)$ to union two lists given that A is smaller than B. Thus, **Merge(C, A, B)** will run in $O(min(|A|, |B|)$

Therefore, the total running time of the algorithm is

$$T(n) = min(|A|, |B|) + c + min(|A|, |B|) \leq 3min(|A|, |B|) = O(min(|A|, |B|))$$

Question 7:

```
Algorithm(A,B):
    A_length = Length(A)
    B_length = Length(B)
    Store = new [A_length][B_length];

    for i in 1..A_length+1 do
        for j in 1..B_length+1 do
            Store[i][j] = 0

    for i in 2..A_length+1 do
        for j in 2..B_length+1 do
            if A[i-1] == B[j-1] then
                Store[i][j] = Store[i-1][j-1] + 1
            else then
                Store[i][j] = 0

    max_i = 1
    max_j = 1
    max_len = 0
    for i in 2..A_length+1 do
        for j in 2..B_length+1 do
            if Store[i][j] > Store[max_i][max_j] then
                max_i = i
                max_j = j
                max_len = Store[i][j]

    LCS = SubString(A, (max_i-max_len), max_i)
    return LCS
```

We assume that $SubString(Str, Start, End)$ routine returns a sub-sequence for the given string in $O(abs(Start - End))$ time.

Let's analyse the running time for this algorithm.

1 First we get lengths for both strings it will take $m + n$ times.

2 On the second step we initialize the table that will store results for subproblems. Since we create a $(Length(A) + 1) \times (Length(B) + 1)$ grid it will take $O((m + 1)(n + 1))$, where $m = Length(A)$ and $n = Length(B)$

3 On the next step we follow the buttom-up approach to find longest common sequence lengths for $2 \leq i \leq Length(A) + 1$ and $2 \leq j \leq Length(B) + 1$. Since we need to fill

every cell in each row and column in the grid, except the first row and the first column, the running time will be $O(mn)$

4 The third loop traverses the grid and finds the maximum value and corresponding indices. Since we need to check all cells, except the first row and the first column, the running time will be $O(mn)$

5 The last step takes a substring, we assumed that is takes $O(abs(Start - End))$. In case when $A = B$ this operations will take at most $O(k)$, where $k = m = n$, but in general case $k \leq m$ and $k \leq n$

Therefore, the total running time will be:

$$T(n) = m + n + (m + 1)(n + 1) + mn + mn + k$$

$$= m + n + mn + m + n + 1 + mn + mn + k = 3mn + 2m + 2n + 1 + k$$

$$\leq 3mn + 2m + 2n + mn + k \leq 3mn + 2mn + 2mn + mn + k$$

$$\leq 8mn + k \leq 8mn + mn = 9mn = O(mn)$$