COMP 3804 — Winter 2026
Solutions Problem Set 4

Some useful facts:

1. $1 + 2 + 3 + \cdots + n = n(n+1)/2$.
2. for any real number $x > 0$, $x = 2^{\log x}$.
3. For any real number $x \neq 1$ and any integer $k \geq 1$,
$$1 + x + x^2 + \cdots + x^{k-1} = \frac{x^k - 1}{x - 1}.$$
4. For any real number $0 < \alpha < 1$,
$$\sum_{i=0}^{\infty} \alpha^i = \frac{1}{1-\alpha}.$$

Master Theorem:

1. Let $a \geq 1$, $b > 1$, $d \geq 0$, and
$$T(n) = \begin{cases} 1 & \text{if } n = 1, \\ a \cdot T(n/b) + \Theta(n^d) & \text{if } n \geq 2. \end{cases}$$
2. If $d > \log_b a$, then $T(n) = \Theta(n^d)$.
3. If $d = \log_b a$, then $T(n) = \Theta(n^d \log n)$.
4. If $d < \log_b a$, then $T(n) = \Theta(n^{\log_b a})$.

Question 1: Let $G = (V, E)$ be a directed graph, in which each edge (u, v) has a positive weight $wt(u, v)$, let $n = |V|$ and $m = |E|$. This graph is given using adjacency lists: Each vertex u has a list that stores all vertices v such that (u, v) is an edge in E. In other words, this list stores all edges *going out* of u.

As in class, we denote the length of a shortest path from vertex u to vertex v by $\delta(u, v)$.

Give an algorithm that takes as input this graph G, together with two distinct vertices a and b. The output of the algorithm is the list of all vertices v such that

1. $\delta(v, a) \leq 20$ (i.e., the shortest path from v to a has length at most 20) *and*
2. $\delta(v, b) \leq 26$ (i.e., the shortest path from v to b has length at most 26).

The running time of your algorithm must be $O((n + m) \log n)$.

Describe your algorithm in plain English (Step 1: Do this. Step 2: Do that. Etc.) You may use any result that was proven in class or in the tutorials. As always, explain why your algorithm is correct and why its running time is $O((n + m) \log n)$.

Solution:

Step 1: Let $G' = (V, E')$ be the graph obtained from G by replacing each edge (u, v) by (v, u); the edge weight stays the same. In words, we obtain G' by reversing all arrows in G.

From the tutorials, the adjacency list representation of G' can be computed in $O(n+m)$ time.

We denote the length of a shortest path in G' from vertex v to vertex u by $\delta'(v, u)$. Note that $\delta(u, v) = \delta'(v, u)$.

Step 2: Run Dijkstra's algorithm on G' with source vertex a. This gives, for each vertex v, the value of $\delta'(a, v)$. By walking along these values, we find the list L_a of all vertices v for which $\delta'(a, v) = \delta(v, a) \leq 20$. All of this takes $O((n + m) \log n)$ time.

Step 3: Do the same as in Step 2, with a replaced by b. This gives the list L_b of all vertices v for which $\delta(v, b) \leq 26$. All of this takes $O((n + m) \log n)$ time.

Step 4: For the final output, we compute the intersection of the lists L_a and L_b.

Number the vertices of V as $v_1, v_2, \ldots, v_n$. Sort the vertices in L_a by their indices, and sort the vertices in L_b by their indices. As in the merge step of Merge-Sort, scan the sorted list, and find all vertices that occur in both. All of this takes $O(n \log n)$ time.

Question 2: Let $G = (V, E)$ be a directed graph, in which each edge (u, v) has a positive weight $wt(u, v)$, and let s be a source vertex in V. Let $n = |V|$ and $m = |E|$.

Recall that Dijkstra's algorithm computes for each vertex v, the length $\delta(s, v)$ of a shortest path from s to v, in total time $O((n + m) \log n)$. The pseudocode for this algorithm is given below.

Assume that each edge weight $wt(u, v)$ is an integer in the set $\{1, 2, \ldots, 2026\}$. Prove that Dijkstra's algorithm can be implemented such that the running time is $O(m + n)$.

Hint: A *priority queue* is a data structure that stores any finite sequence of numbers and supports the operations INSERT, EXTRACT_MIN and DECREASE_KEY. An example of a priority queue is a min-heap.

A priority queue is called *monotone* if, during any sequence of operations, the smallest element never decreases. Thus, if at some moment, the smallest element is 45, then later on, the smallest value will always be at least 45. (Note that, even though this was not proven in class, the sequence of operations during Dijkstra's algorithm has this property. You may use this fact.)

Assume that at any moment, any number stored in a monotone priority queue is an integer belonging to the set $\{0, 1, 2, \ldots, k\}$. Start by showing that any sequence consisting of n INSERT-operations, n EXTRACT_MIN-operations and m DECREASE_KEY-operations (these operations may appear in any order) can be processed in total time $O(m + n + k)$.

```
Algorithm DIJKSTRA(G, s):
for each v ∈ V
do d(v) = ∞
endfor;
d(s) = 0;
S = ∅;
Q = V;
while Q ≠ ∅
do u = vertex in Q for which d(u) is minimum;
    delete u from Q;
    insert u into S;
    for each edge (u, v)
    do if d(u) + wt(u, v) < d(v)
        then d(v) = d(u) + wt(u, v)
        endif
    endfor
endwhile
```

Solution: As the hint suggests, we start by describing a data structure that implements a monotone priority queue.

- We have a set Q of items, where each item v has a key $key(v)$, which is an integer belonging to the set $\{0, 1, 2, \ldots, k\}$.

- We store the set Q in an array $A[0, \ldots, k]$. Each entry $A[i]$ is a doubly-linked list storing all items v in Q for which $key(v) = i$.
- We have a variable *current_min*, whose value is the smallest integer i for which the list $A[i]$ is non-empty.

  Note: Since the priority queue is monotone, this variable never decreases.

- INSERT(v): Add item v at the end of the list $A[key(v)]$.
  - This takes $O(1)$ time.
- DECREASE_KEY(v, x): This operation gets a pointer to the node in the list $A[key(v)]$ that stores the item v. The real number x satisfies $x < key(v)$. Note that since the priority queue is monotone, we have $x \geq key(w)$, where w is an arbitrary item in the list $A[current\_min]$.

  To process this operation DECREASE_KEY(v, x), we do the following:
  - Delete v from the list $A[key(v)]$.
  - Set $key(v) = x$.
  - Add item v at the end of the list $A[key(v)]$.
  - Note: We do not have to update the variable *current_min*.
  - The entire operation takes $O(1)$ time.
- EXTRACT_MIN: We do the following:
  - Let v be an arbitrary item in the list $A[current\_min]$.
  - Delete v from the list $A[current\_min]$.
  - If the list $A[current\_min]$ is empty: keep on increasing *current_min* until the list $A[current\_min]$ is non-empty.
  - Return v.
  - The entire operation takes time proportional to 1 plus the number of times the variable *current_min* is increased.

Consider an arbitrary sequence consisting of n INSERT-operations, n EXTRACT_MIN-operations and m DECREASE_KEY-operations (these operations may appear in any order).

- It takes $O(k)$ time to initialize the array $A[0, \ldots, k]$.
- The total time for the n INSERT-operations is $O(n)$.
- The total time for the m DECREASE_KEY-operations is $O(m)$
- What is the total time for the n EXTRACT_MIN-operations:

  - It is $O(n)$ plus the total number of times the variable *current_min* is increased.
  - The variable *current_min* can be increased only k times.
  - Thus, the total time for the n EXTRACT_MIN-operations is $O(n+k)$.

- We conclude that the total time to process the sequence of n INSERT-operations, n EXTRACT_MIN-operations and m DECREASE_KEY-operations is $O(m+n+k)$.

- Monotone priority queues are Awesome Eh!

Now we are going to use this result to implement Dijkstra's algorithm for the case when all edge weights are integers in the set $\{1, 2, \ldots, 2026\}$.

- To implement Dijkstra's algorithm, it is enough to have a monotone priority queue; see property 4 on page 104 of my handwritten notes.

- Any path between two vertices has at most $n-1$ edges and, thus, length at most $2026(n-1)$.

- Thus, any value $d(v)$ is either ∞ or an element of the set $\{0, 1, \ldots, 2026(n-1)\}$.

- We store the set Q in our monotone priority queue, where $k = 2026n$ plays the role of ∞.

- The running time of Dijkstra's algorithm is equal to the total time to process a sequence of n INSERT-operations, n EXTRACT_MIN-operations and m DECREASE_KEY-operations. We have seen above that this is $O(m+n+k)$, which is $O(m+n)$.

Question 3: Let $G = (V, E)$ be a connected undirected graph, in which each edge has a weight. Assume that all edge weights are distinct.

Professor Justin Bieber claims that G has only one minimum spanning tree.

Is Professor Bieber's claim correct? As always, justify your answer.

Solution: Justin is correct. The proof is by contradiction. Assume that T_1 and T_2 are different minimum spanning trees of G. Note that T_1 and T_2 may share edges. However, there is at least one edge that is in exactly one of T_1 and T_2.

Let e be the shortest edge that is in exactly one of T_1 and T_2. We may assume that e is an edge in T_1. (Otherwise, we swap T_1 and T_2.)

Consider the graph $T_2 + e$, i.e., the graph obtained by adding the edge e to T_2. This graph contains a cycle C and e is an edge of C. Note that some other edges on C may be in T_1. However, C contains an edge f that is not in T_1. (Why: otherwise, T_1 contains a cycle.)

Since the edge f is in exactly one of T_1 and T_2, our choice of e implies that the weight of e is strictly smaller than the weight of f.

The graph $T_2 - f + e$, i.e., the graph obtained by replacing, in T_2, the edge f by e. This graph is a spanning tree. Since its weight is smaller than the weight of T_2, it follows that T_2 is not a minimum spanning tree. This is a contradiction.

Good job Justin!

Question 4: Let $G = (V, E)$ be a connected undirected graph, in which each edge has a weight. An edge $\{u, v\}$ is called *annoying* if the graph $G' = (V, E \setminus \{\{u, v\}\})$ is not connected.

Assume there is a unique edge in E with largest weight; denote this edge by e.

Prove that e is an edge in every minimum spanning tree of G if and only the edge e is annoying.

Solution: Let u and v be the two vertices of e.

We first assume that e is an edge in every minimum spanning tree of G. We have to prove that e is an annoying edge.

Let T be an arbitrary minimum spanning tree of G. By removing the edge e from T, we obtain two trees T_1 and T_2, where u is a vertex of T_1 and v is a vertex of T_2. Let A be the vertex set of T_1 and let B be the vertex set of T_2. Let $e' = \{u', v'\}$ be an edge in G of minimum weight with $u' \in A$ and $v' \in B$. Let T' be the spanning tree obtained from T by replacing e by e'. We have seen in class that the weight of T' is equal to the weight of T. This implies that e and e' have the same weight. Since the largest edge weight in G is unique, it follows that $e = e'$. Thus, there is only one edge in G between the sets A and B. Therefore, the edge e is annoying.

To prove the converse, assume that the edge e is annoying. If we remove e from G, we obtain two connected subgraphs, say G_1 containing u, and G_2 containing v. Let A be the vertex set of G_1 and let B be the vertex set of G_2. Observe that e is the only edge between A and B. As a result, every spanning tree of G contains the edge e. In particular, this is true for every minimum spanning tree.