COMP3027 – Assignment 2

Problem 1 – Star Wars (Again)

Part A – Algorithm

Preprocessing

Sort stormtroopers by increasing perishing times (f_i).

Algorithm

To determine the minimum cost subset, S, we:

- Initialize an empty set $T = \emptyset$
  - Where T will store the indices of the selected non-overlapping stormtroopers, i.e., the ones we can exclude to reduce cost.
- Find the maximum cost that can be saved by selecting non-overlapping stormtrooper intervals
  - Let $OPT(i) =$ the maximum cost that can be saved by selecting non-overlapping stormtrooper intervals from the first i stormtroopers.
  - Using dynamic programming, we get two main cases:
    - OPT selects stormtrooper i. Giving → $c_i + OPT\big(p(i)\big)$
      - Where $p(i)$ is the stormtrooper before i that does not overlap with i
    - OPT does not select job i. Giving → $OPT(i-1)$
  - Resulting in the following definition of the recurrence relation:

$$OPT(i) = \begin{cases} 0 & if\ \ i = 0 \\ \max\Big(c_i + OPT\big(p(i)\big), OPT(i-1)\Big) & otherwise \end{cases}$$

  - During the computation for $OPT(n)$ using the recurrence relation, if:

$$c_i + OPT\big(p(i)\big) > OPT(i-1)$$

    We add stormtrooper i to the set T

- Once we have the full set T (stormtroopers to exclude), the minimum cost subset, S, of stormtroopers is:

$$S = \{1,2,3,\ldots,n\} \setminus T$$

Part B – Correctness

Reduction to Weighted Interval Scheduling

Rather than selecting the minimum-cost set S to cover all overlaps, we can find the maximum-cost set T of non-overlapping stormtroopers to exclude (i.e., a subset we can safely remove).

To do this, we use the following idea shown by this equation:

$$Minimum\ Cost = \sum_{i=1}^{n} c_i - OPT(n)$$

Where n is the number of stormtroopers, $\sum_{i=1}^{n} c_i$ is the sum of loyalty costs for all stormtroopers and $OPT(n)$ is the maximum cost that can be saved by selecting non-overlapping stormtrooper intervals from the n stormtroopers.

Since any two overlapping stormtroopers must have at least one included in the final set S, removing a maximum-cost set of non-overlapping ones ensures we minimize cost while satisfying the coverage constraint.

Thus, we reduce the problem to the Weighted Interval Scheduling problem:

- Finding the maximum-cost subset of non-overlapping intervals (stormtroopers), which we denote as T.
- Then the minimum-cost subset we actually want is:
$$S = \{1,2,3, \dots, n\} \setminus T$$

Proof by Induction

Defining Optimal Solution

Let $OPT(i)$ the maximum cost that can be saved by selecting non-overlapping stormtrooper intervals from the first i stormtroopers.

Proving the Base Case

$OPT(0) = 0$: with no stormtroopers, there's nothing to cover. Therefore, the cost is 0.

Inductive Hypothesis

Assume $OPT(i)$ is correct for all $i < j$. That is, for each smaller subproblem, $OPT(i)$ correctly gives the maximum cost of non-overlapping intervals that can be excluded among the first i stormtroopers.

Inductive Step

We consider the i^{th} stormtrooper and have two choices:

1. Include stormtrooper i:
   - This adds cost c_i, and we must combine it with a compatible solution for earlier intervals, i.e., $OPT\big(p(i)\big)$
   - Total cost: $c_i + OPT\big(p(i)\big)$
2. Exclude stormtrooper i:
   - Just take the optimal solution for $i - 1$: $OPT(i - 1)$

By taking the maximum of these two, we ensure $OPT(i)$ is the maximum cost saved by a valid subset of non-overlapping intervals.

Therefore, by the inductive hypothesis, both $OPT(i - 1)$ and $OPT\big(p(i)\big)$ are correctly computed, so the recurrence is correct.

Conclusion

By the principle of mathematical induction, the recurrence correctly computes $OPT(n)$, the maximum cost of a subset of non-overlapping stormtroopers that can be excluded

Thus:

- The cost of the optimal subset to keep is:
$$\sum_{i=1}^{n} c_i - OPT(n)$$
- And the subset $S = \{1,2,3, \dots, n\} \setminus T$, where T is the trace-back reconstruction of the selected non-overlapping intervals, satisfies the problem constraints.

Part C – Time Complexity

Sorting the list of stormtroopers by perishing times runs in $O(n \log n)$ time.

Precomputing $p(i)$ runs in $O(\log n)$ time (achieved via binary search). But it is done for all i, in other words: for each of the n stormtroopers, therefore it runs in $O(n \log n)$ time.

Filling out $OPT[0..n]$ iteratively using the recurrence relation and each computation of $OPT(i)$ takes constant time if $p(i)$ is already known. Therefore, it runs in $O(n)$ time.

Reconstructing the minimum-cost set $S = \{1,2,3, \dots, n\} \setminus T$ runs in $O(n)$ time as T is a set.

Summing this up:

$$O(n \log n) + \; O(n \log n) + O(n) + O(n) = O(n \log n)$$

We get the final time complexity of $O(n \log n)$

Problem 2 – QDijkstra

Part A – Algorithm

Let $OPT[v][t]$ = the maximum scenery score achievable at vertex v with total time t.

We define a $V \times (T+1)$ table, called OPT, where:

- Each row corresponds to a vertex $v \in E$
- Each column corresponds to a time value $t \in [0, T]$
- The entry $OPT[v][t]$ stores the maximum scenery score achievable at vertex v with total time t

And we initialize the table as follows:

- Set all $OPT[v][t] = -\infty$ in the dynamic programming table initially (not reachable yet)
- Set all $OPT[u][0] = 0$ → initially (meaning starting at vertex u with 0 time and 0 scenery)

Using dynamic programming, we consider the following two main cases for reaching vertex b at total time t:

- We reach vertex b via an edge (a, b):
    - This means we previously reached vertex a at time $t - t(a, b)$
    - The total scenery score is $OPT[a][t - t(a, b)] + s(a, b)$
- We do not reach vertex b at time t:
    - In this case, $OPT[b][t]$ retains its previous value (i.e., no improvement)

Hence, we update each entry using the following recurrence:

$$OPT[b][t] = \max\big(OPT[b][t], OPT[a][t - t(a, b)] + s(a, b)\big)$$

Resulting in the following definition of the recurrence relation:

$$OPT[i][t] = \begin{cases} 0 & if\ \ i = u\ \ and\ \ t = 0 \\ -\infty & if\ \ t = 0\ \ and\ \ i \neq u \\ \max\limits_{(a,i)\,\in\,E} \Big(OPT(a)\big(t - t(a, i)\big) + s(a, i)\Big) & if\ \ t > 0 \end{cases}$$

Using the recurrence relationship, we fill in the above table by:

- Iterating over time from 0 to T
- For each edge, $(a, b) \in E$, if $t(a, b) \leq t$, update $OPT[b][t]$ based on the other vertex's previous state

After filling the table, we examine the values of $OPT[v][t]$ for all $t \in [0, T]$ (in simpler terms, we check the values in row v):

- If any of them satisfies $OPT[v][t] \geq S$, then return `True` - the goal is achievable.
- If none satisfy the condition, return `False`.

Part B – Correctness

Proof by Induction

Defining Optimal Solution

Let $OPT[v][t]$ be the maximum scenery score achievable at vertex v with total time t.

Proving the Base Case

By initialization, $OPT[u][0] = 0$, starting point, no travel, no scenery.
For all $v \neq u$, $OPT[v][0] = -\infty$, no way to reach other nodes in 0 time.

Inductive Hypothesis

Assume that for all $t < T$, and for all vertices v, $OPT[v][t]$ correctly stores the maximum scenery score for paths from u to v taking exactly t time.

Inductive Step

We consider each vertex $b \in V$. The recurrence is:

$$OPT[b][t] = \max\left(OPT[b][t], \max_{(a,b)\in E, t(a,b)\leq t}\left(OPT[a][t - t(a,b)] + s(a,b)\right)\right)$$

For each edge $(a,b) \in E$ such that travel time $t(a,b) \leq t$:

- By inductive hypothesis, $OPT[a][t - t(a,b)]$ is correct as it stores the best scenery score to reach a in $t - t(a,b)$ time.
- Adding edge (a,b) adds $s(a,b)$ scenery and takes $t(a,b)$ time.
- Therefore, total time is t, and total scenery is $OPT[a][t - t(a,b)] + s(a,b)$.
- Taking the maximum over all such predecessors a ensures we compute the best possible score to reach b in exactly time t.

Hence, $OPT[b][t]$ is correctly updated using only previously proven correct values from smaller t.

Conclusion

After computing all values in the table, we check $\exists\ t \in [0, T]$ such that $OPT[v][t] \geq S$:

- If such a value exists, it means there is a path from u to v in time $\leq\ T$ with scenery $\geq\ S$.
- Otherwise, no such path exists.

Thus, guaranteeing correct output.

Part C – Time Complexity

If we let:

- $n =$ total number of vertices in graph G
- $m =$ total number of edges in graph G

Setting up the DP table $OPT[n][t] = -\infty$ for all $n \in V$, $t \in [0, T]$, and initializing $OPT[u][0] = 0$ takes $O(V \times T)$ time.

Iterating over time from $t = 0$ to T, and for each time unit, we examine all edges $m \in E$ to fill in the DP table. Hence, running in $O(m \times T)$ time.

To determine whether any entry in $OPT[v][t] \geq S$ for some vertex v and $t \in [0, T]$, we check: T entries in total, running in $O(T)$ time.

Summing everything:

$$\begin{aligned} O(n \times T) + O(m \times T) + O(T) &= O\big((n \times T) + (m \times T)\big) \\ &= O((n + m) \times T) \\ &= O((n + m)T) \end{aligned}$$