CS345 : Assignment 2

Dhruv Gupta (220361)

August 2024

Dhruv Gupta: 220361

Question 1

Algorithm and its Proof of Correctness :

We will simply sort the jobs in increasing order of their time/weight ratios, i.e., for each job i, calculate $x_i = t_i/w_i$ and sort them according to their x_i values.
We claim that this is an optimal ordering for the given problem (by optimal ordering we mean an ordering for which the sum given in problem is minimum). We will prove our claim by contradiction. Consider any ordering $\Pi = [j_1, ..j_n]$ which is not sorted, i.e., we have two consecutive jobs j_k, j_{k+1} such that $t_{j_k}/w_{j_k} > t_{j_{k+1}}/w_{j_{k+1}}$. Let us define cost of any ordering Π as $cost(\Pi) = \sum_{i=1}^{n} w_i C_i = \sum_{i=1}^{n} w_{j_i} C_{j_i}$. We will create another ordering $\Pi' = [j_1, ..., j_{k-1}, j_{k+1}, j_k, ..., j_n]$ by swapping j_k, k_{k+1} and we will prove that $cost(\Pi') < cost(\Pi)$. This would imply, that any ordering which is not sorted can not be an optimal ordering.
Note that for job j_i, $C_{j_i} = \sum_{r=1}^{i} t_{j_r}$. Observe that, C_{j_i} will remain same for Π and Π' for all $i < k$ and $i > k+1$, since the sum of t_{j_i} before k and after $k+1$ will not be affected by the swap. Also in Π', $C'_{j_{k+1}} = C'_{j_{k-1}} + t_{j_{k+1}} = C_{j_{k-1}} + t_{j_{k+1}}$ and $C'_{j_k} = C'_{j_{k+1}} + t_{j_k} = C_{j_{k-1}} + t_{j_{k+1}} + t_{j_k} = C_{j_k} + t_{j_{k+1}}$.

$$cost(\Pi') - cost(\Pi) = \sum_{i=1}^{n} w_{j_i} C'_{j_i} - \sum_{i=1}^{n} w_{j_i} C_{j_i}$$

$$= (\sum_{i=1}^{k-1} w_{j_i} C_{j_i} + \sum_{i=k+2}^{n} w_{j_i} C_{j_i} + w_{j_{k+1}}(C_{j_{k-1}} + t_{j_{k+1}}) + w_{j_k}(C_{j_k} + t_{j_{k+1}}))$$

$$-(\sum_{i=1}^{k-1} w_{j_i} C_{j_i} + \sum_{i=k+2}^{n} w_{j_i} C_{j_i} + w_{j_{k+1}} C_{j_{k+1}} + w_{j_k} C_{j_k})$$

$$= w_{j_{k+1}}(C_{j_{k-1}} + t_{j_{k+1}} - C_{j_{k+1}}) + w_{j_k} t_{j_{k+1}} = w_{j_{k+1}}(C_{j_{k-1}} + t_{j_{k+1}} - C_{j_{k-1}} - t_{j_{k+1}} - t_{j_k}) + w_{j_k} t_{j_{k+1}}$$

$$= w_{j_k} t_{j_{k+1}} - w_{j_{k+1}} t_{j_k} = w_{j_k} w_{j_{k+1}} (t_{j_{k+1}}/w_{j_{k+1}} - t_{j_k}/w_{j_k}) < 0$$

Thus $cost(\Pi') - cost(\Pi) < 0 \Rightarrow cost(\Pi') < cost(\Pi)$.

So, any unsorted ordering can not be optimal. Also, if all the t_i/w_i values are distinct then, there would be a unique sorted ordering. Since, no other ordering can be optimal, the sorted ordering must be optimal. In the case when t_i/w_i are not distinct, we can prove that all sorted orderings will give the same *cost* using the same equation we proved above. Any ordering can be converted to another using a series of swaps of consecutive elements (same idea which is used in insertion sort). Also, in any sorted ordering same value elements appear as a contiguous block, so converting a sorted ordering to another sorted ordering will consist of a series of swaps of consecutive elements, each of which will involve swapping two elements with the same value. As shown above $cost(\Pi') - cost(\Pi) = w_{j_k} w_{j_{k+1}} (t_{j_{k+1}}/w_{j_{k+1}} - t_{j_k}/w_{j_k}) = 0$ (since $t_{j_{k+1}}/w_{j_{k+1}} = t_{j_k}/w_{j_k}$), i.e., swapping two elements with same value does not affect the *cost*. Hence, all sorted ordering will have the same *cost*. Therefore we have proven that sorting the jobs according to their time/weight ratio will give us an optimal ordering. This proves the correctness of our algorithm.

Time Complexity :
Since sorting can be done in $O(nlog(n))$ time, time complexity of our algorithm is $O(nlog(n))$.

Dhruv Gupta: 220361

Question 2

Idea of the Algorithm & Pseudocode:

Since we have a DAG, we will use the idea of topological ordering to solve this problem. We know that in a topological ordering, for any node u, all the nodes reachable from u must lie after u. This is because, if v comes before u, then for a path to exist from u to v, we must have x, y such that $u \to ... \to x \to y \to ... \to v$ is a path and x comes after y in the topological ordering. Since (x, y) is an edge, this is not possible. We will assume that we have a function **Topo_order** which takes a graph $G = (V, E)$ as input and returns an array $T[1, ...n]$, $(n = |V|)$, such that $T[i]$ is the i^{th} node in the topological ordering of G in $O(m + n)$ time.
Let $N(u)$ denote the set consisting of all the nodes w, such that $(u, w) \in E$.
Note that for any node u,

$$cost(u) = \min(price(u), (\min_{w \in N(u)} cost(w))) \tag{1}$$

Idea behind the above equation is that for any node $v \neq u$, which is reachable from u, there must be some w in $N(u)$, such that v is reachable from w, because we would need to take atleast one edge from u to reach v. Therefore, if we have calculated the *cost* for all nodes in $N(u)$, then we can easily calculate the *cost* for u. This is where topological sorting helps us. Since all the nodes in $N(u)$ occur after u, we can traverse the topological order T in reverse and calculate *cost* for each node. In this way, when we need to calculate the *cost* for u, inductively we can say that the *cost* for all nodes in $N(u)$ will already be calculated. The base case would be the last node in T, which can't have any outgoing edge, so for that node, *cost* would be the same as its *price*.

```
CalculateCosts($G = (V, E)$) {
    $T \leftarrow$ Topo_order($G$);
    $n \leftarrow |V|$;
    for $i = n$ to $1$ {
        $u \leftarrow T[i]$;
        $cost[u] \leftarrow price[u]$;
        for every edge $(u, w) \in E$ {
            $cost[u] \leftarrow \min(cost[u], cost[w])$;
        }
    }
    return $cost[1, ..., n]$;
}
```

Time Complexity Analysis :

Let $n = |V|$ and $m = |E|$. Time taken by **Topo_order** is $O(m+n)$. In the outer **for** loop, we iterate over each node exactly once, and for each node u, the inner **for** loop will run for exactly $outdegree(u)$ iterations. Therefore time taken by these nested **for** loops $= \sum_{u \in V}(C + outdegree(v)) = Cn + \sum_{u \in V} outdegree(v) = Cn + m = O(m + n)$. Here C is some constant, since each iteration of outer **for** also involves some $O(1)$ time operations. An alternate way to calculate this time (instead of using *outdegree*), is to observe that each edge will be considered exactly once over all iterations of the **for** loops. Therefore, overall time complexity is $O(m + n)$.