CS545 Spring, 2015

Homework Assignment #3
Due: March 26
Student: Shuo Yang

---

1. (a) Suppose we have 6 activities, for activity i, s_i is the start time, f_i is the finish time and d_i is the duration. And we sort them in order of increasing duration:

i	1	2	3	4	5	6
s_i	8	3	1	4	4	0
f_i	9	5	4	8	9	6
d_i	1	2	3	4	5	6

The greedy procedure when the activities are considered in order of increasing duration would be: 1) select activity 1 since it is the local best; 2) select activity 2 since it is compatible with activity 1; 3) no other activities can be selected since they are not compatible with either activity 1 or 2. So the procedure yields the solution $\{1, 2\}$, but the optimal solution should be $\{3, 4, 1\}$. Therefore the greedy procedure is not correct.

(b) With the same input, now let us sort them in order of increasing start-time:

i	1	2	3	4	5	6
s_i	0	1	3	4	4	8
f_i	6	4	5	8	9	9

The greedy procedure when the activities are considered in order of increasing duration is:

Function $GreedyActivitySelection(S, F, n)$
$A := \{1\}$ // activity with the earliest start time
$j := 1$ // indicates the activity with greatest start time in A
for $i := 2$ to n
  if $S[i] \geq F[j]$
    $A := A \cup \{i\}$
    $j := i$
return A

Running the above procedure with the same input yields the solution $\{1, 6\}$ which is certainly not an optimal solution. Therefore the greedy procedure is not correct.

(c) Consider the following 9 activities ordered by the number of overlaps which is denoted by o_i:

i	1	2	3	4	5	6	7	8	9
s_i	4	0	7	1	2	3	5	6	6
f_i	6	3	10	4	4	5	7	8	9
o_i	2	2	2	3	3	3	3	3	3

When considering the order of increasing number of overlaps, the greedy procedure would first pick activity 1, then pick activity 2, and finally pick activity 3. Thus the solution derived by the greedy procedure is $\{1, 2, 3\}$, but the optimal solution should be $\{2, 6, 7, 3\}$. Therefore the greedy procedure is not correct.

2. Label n gas stations as $g_1, g_2, \cdots, g_n$ and let d_i be the distance between gas stations g_i and g_{i+1}, specially, d_0 denotes the distance between g_1 and city A, and d_n denotes the distance between city B and g_n. Let D be the total distance between city A and B, where $D = \sum_{i=0}^{n} d_i$.

Greedy algorithm

The greedy strategy is to refuel at the gas station whose distance from the current stop is closest to m but less than m.

Function $GreedyTripRefuel()$
$A := \emptyset$ // the set of refueling stops
$i := 0$ // indicates the farthest gas station one can make.
$T_1 := 0$ // distance traveled before making a refuel
$T_2 := 0$ // total distance traveled
while $T_2 < D$
if $(T_2 - T_1) \leq m$
$T_2 := T_2 + d_i$
$i := i + 1$
else
$i := i - 1$ // cannot make it to station g_i, so must refuel at station g_{i-1}
$A := A \cup \{g_i\}$
$T_2 := T_2 - d_i$
$T_1 := T_2$
return A

The algorithm runs in $O(n)$ time since there are at most n gas stations along the way.

Correctness

Lemma: Suppose A is a subset of an optimal solution where the latest refuel stop is station g_k. Let g_i be the gas station after g_k whose distance from g_k is closest to m but less than m. Then $A \cup \{g_i\}$ is also a subset of an optimal solution.

Proof. Since A is a subset of an optimal solution, let A^* be an optimal solution with $A \subseteq A^*$. Let g_j be the gas station in $A^* - A$ that is next to g_k. We have two cases:

- case-1: $i = j$. Then $A \cup \{g_i\} \subseteq A^*$, the lemma holds.
- case-2: $i \neq j$. Since the distance from g_k to g_i is closest to m but less than m, this distance must be greater than the distance from g_k to g_j. Thus we can replace g_j with g_i and still yields an optimal solution since g_i is closer to the future stops than g_j.

□

Theorem: $GreedyTripRefuel()$ finds an optimal solution.

Proof. Initially, A is an empty set, which is a subset of an optimal solution and g_k in this case would be city A. By the lemma, $A \cup g_i$ is a subset of an optimal solution where g_i is the gas station whose distance from city A is closest to m but less than m.

By induction on the number of iterations, when the function terminates, A is a subset of an optimal solution.

Since from the starting point, the greedy algorithm makes the least possible stops, there is no better solution with fewer stops than the one produced by the greedy algorithm. Therefore A is optimal. □

3. **Greedy algorithm**

Let the set of n tasks be $\{a_i | 1 \leq i \leq n\}$.

(a) Merge sort the n tasks in the order of increasing execution time such that $t_1 \leq t_2 \leq \cdots \leq t_n$.

(b) Schedule the tasks in the order of $\{a_1, a_2, \cdots, a_n\}$.

Step (a) takes $O(n \log n)$ time and step (b) takes $O(n)$ time, thus the overall runtime is $O(n \log n)$.

Correctness

First, we define containment relationship.
We define $\{a_1, a_2, \cdots, a_m\}$ as a schedule sequence such that a_i is scheduled after a_{i-1} where $1 < i \leq m$. For a schedule sequence $\{a_1, a_2, \cdots, a_n\}$, we define $\{a_1, a_2, \cdots, a_k\}$ as a prefix-subsequence of it where $k \leq n$.

Lemma: Suppose A is a prefix-subsequence of an optimal schedule sequence. Let a_i be the task that has the shortest execution time among those tasks that are not in A. Let A' be the prefix-subsequence by appending a_i to the end of A. Then A' is also a prefix-subsequence of an optimal schedule sequence.

Proof. Since A is a prefix-subsequence of an optimal schedule sequence, let A^* be the optimal schedule sequence that contains A as a prefix-subsequence. Let a_i be the task that has the shortest execution time among those tasks in $A^* - A$. By appending a_i to the end of the A, we get another prefix-subsequence A'. There are two cases:

(a) A' is a prefix-subsequence of A^*, the lemma holds.

(b) A' is not a prefix-subsequence of A^*, Let a_j be the task that appears first in the subsequence $A^* - A$. In this case we must have: $t_i = t_j$. We prove this by contradiction:
Suppose $t_i \neq t_j$, since a_i is the task with the shortest execution time in $A^* - A$, we must have: $t_j > t_i$. Let the last task in A be a_m. Without loss of generality, assume $A^* - A$ is ordered as $a_j, a_{x_1}, \cdots, a_{x_s}, a_i, a_{y_1}, \cdots, a_{y_r}$ such that there are s tasks scheduled after a_j and before a_i, and there are r tasks scheduled after a_i. Let c_{x_k} denotes the completion time for the tasks scheduled in between a_j and a_i, and let c_{y_k} the completion time for the tasks scheduled after a_i. We have:

$$c_j = c_m + t_j \tag{1}$$

$$c_{x_k} = c_j + \sum_{1 \leq i \leq k}^{k \leq s} t_{x_i} \tag{2}$$

$$c_i = c_j + \sum_{1 \leq i \leq s} t_{x_i} + t_i \tag{3}$$

$$c_{y_k} = c_i + \sum_{1 \leq i \leq k}^{k \leq r} t_{y_i} \tag{4}$$

Now suppose we exchange the positions of a_i and a_j to produce another subsequence $(A^* - A)'$ $a_i, a_{x_1}, \cdots, a_{x_s}, a_j, a_{y_1}, \cdots, a_{y_r}$ and together with A, this produces another schedule sequence $A^{*'}$. In this case, let c'_j be the completion time for a_j, c'_i be the completion time for a_i, c'_{x_k} be the completion time for the tasks scheduled in between a_j and a_i, and let c'_{y_k} be the completion

time for the tasks scheduled after a_j. We have:

$$c'_i = c_m + t_i \tag{5}$$

$$c'_{x_k} = c'_i + \sum_{1 \le i \le k}^{k \le s} t_{x_i} \tag{6}$$

$$c'_j = c'_i + \sum_{1 \le i \le s} t_{x_i} + t_j \tag{7}$$

$$c'_{y_k} = c'_j + \sum_{1 \le i \le k}^{k \le r} t_{y_i} \tag{8}$$

Now we compare completion times for the two subsequences:

$$c_j - c'_i = (c_m + t_j) - (c_m + t_i) \tag{9}$$

$$= t_j - t_i \tag{10}$$

$$> 0 \tag{11}$$

$$c_{x_k} - c'_{x_k} = (c_j + \sum_{1 \le i \le k}^{k \le s} t_{x_i}) - (c'_i + \sum_{1 \le i \le k}^{k \le s} t_{x_i}) \tag{12}$$

$$= c_j - c'_i \tag{13}$$

$$> 0 \tag{14}$$

$$c_i - c'_j = (c_j + \sum_{1 \le i \le s} t_{x_i} + t_i) - (c'_i + \sum_{1 \le i \le s} t_{x_i} + t_j) \tag{15}$$

$$= c_j - c'_i + t_i - t_j \tag{16}$$

$$= t_j - t'_i + t_i - t_j \tag{17}$$

$$= 0 \tag{18}$$

$$c_{y_k} - c'_{y_k} = (c_i + \sum_{1 \le i \le k}^{k \le r} t_{y_i}) - (c'_j + \sum_{1 \le i \le k}^{k \le r} t_{y_i}) \tag{19}$$

$$= c_i - c'_j \tag{20}$$

$$= 0 \tag{21}$$

Therefore, the sum of the completion time for $A^* - A$ is absolutely greater than the sum of the completion time for $(A^* - A)'$. So the average completion time of A^* must be greater than that of $A^{*'}$, we have found a better solution than A^*. This contradicts the fact that A^* is an optimal solution. Thus we must have $t_i = t_j$.

So we can get another optimal schedule sequence by exchanging the order of a_i and a_j because since $t_i = t_j$, the exchange will not affect the overall average completion time. And this new optimal schedule sequence has A' as a prefix-subsequence. The lemma still holds.

□

Theorem: The greedy algorithm finds an optimal schedule sequence.

Proof. Initially, A is a empty sequence, which is always a prefix-subsequence of any schedule sequence. Suppose all n tasks $a_1, a_2, \cdots, a_n$ have been sorted in the order of increasing execution time. By the lemma, $\{a_1\}$ is a prefix-subsequence of an optimal schedule sequence.

By induction on the number of iterations, when the algorithm terminates, A is still a prefix-subsequence of an optimal schedule sequence. And when the algorithm terminates, A would contain all n tasks, no other schedule sequence can properly contain A, thus A is the optimal schedule sequence. □

4.