CS577 HW4

10/08/2015

Keith Funkhouser

Haruki Yamaguchi

Algorithm

Following greedy algorithm G is used for this homework:

Assume that the sequence of arrival times of kayakers is sorted chronologically. Split kayakers into a group with size k in descending order; for example, kayakers from $n - k + 1$th earliest arrival time up to n th earliest arrival time form one group. A size of the group that contains a kayaker with the most earliest time may be less than k.

Make a roundtrip for each group at their earliest possible time; either when their last member arrives or when the bus becomes available for them after that.

Claim: For every valid solution S, G gets the finish time of the last bus no later than S does.

Proof: The claim have to follow for all possible outputs with valid S.

Base case: $n = 0$.

Simply, both S and G gets 0.

Case 1: G gets the finish time of the last bus $= a_n + m$.

It means that either $n \le k$ or $x \le a_n$, where x is a time that the previous (second last) bus that carries $(n - k)$th kayaker returns, so that all remaining kayakers can be taken by the next (last) bus. Since this is the earliest possible solution that any S can get, no S gets better time than G does for this case.

Case 2: G gets the finish time of the last bus $> a_n + m$.

It means that $n > k$ and $x > a_n$, where x is a time that the second last bus that carries $(n - k)$th kayaker returns. If there is any S that has the second bus returns before or at a_n, then the claim fails. Following sub cases determines it with two different conditions of a_{n-k}.

Case 2a: $a_{n-k} \le a_n - m$

Combining this case and the fact that $x > a_n$ in G indicates that $n - k > k$. If $n - k \le k$, then G must get $x \le a_n$ since all kayakers up to a_{n-k} can be taken with the bus that leaves at a_{n-k}, which is $\le a_n - m$ in this case.

At here, the conditions of this sub case are similar to the case 2. Actually, this sub case can be rephrased as "G gets the finish time of the second last bus $> a_{n-k} + m$," and recursively for its sub case as "G gets the finish time of the third last bus $> a_{n-2k} + m$..." and so on, until it hits the base case, where $n - k \leq k$, which indicates that no valid S exists that have any bus schedules earlier than those in G (as the condition explained in the first part of this case).

Case 2b: $a_{n-k} > a_n - m$

In order to guarantee the next bus to be the last one, the second last bus has to carry the $(n - k)$th kayaker. If $n - k \leq k$, it can leave at the earliest possible time a_{n-k}, and returns after a_n, so that the last bus immediately follows it. As a result, the finish time of the last bus becomes $a_{n-k} + 2m$. In fact, this is exactly the same result that G gets when $n - k \leq k$ and $a_{n-k} > a_n - m$. It indicates that the best output that S can get is the same as what G gets for this case.

If $n - k > k$, then the second bus may be only able to leave later than a_{n-k} (at least cannot leave earlier). At here again, this case can recursively be applied to the case 2 as "G gets the finish time of the second last bus $> a_{n-k} + m$" until it hits the base case, where $n - k \leq k$, which again indicates that S cannot be better than G.

Case 3: G gets the finish time of the last bus $< a_n + m$.

This case is actually invalid for both S and G since this finish time is not possible unless leaving nth kayaker or traveling faster than m.

The claim is proven since it follows all possible outputs from valid S.

Pseudo code

Input: $k \in \mathbb{Z}^+$ is a capacity of kayakers that the bus can take at one ride, $m \in \mathbb{R}^+$ is a duration of a round trip of one ride in minutes, a sequence $a = \{a_1, a_2, \ldots a_n\} \in \mathbb{R}^+$ in alphabetical order (not chronological) indicates an arrival time of each kayakers from the initial time. For example, $a_i = 40$ means that ith kayaker arrives 40 minutes after the initial starting time.

Output: The minutes spent to carry all kayakers since the initial time.

Procedure $ScheduleBusRides(k, m, a)$

$a \leftarrow MergeSort(a)$ // Assume that a is sorted in ascending chronological order.

$t \leftarrow 0$ // Indicates time in minutes since the initial time.

$r \leftarrow size\ of\ a\ \mathbf{mod}\ k$ // Indicates how many kayakers left before the next trip. Setting this initial value allows the procedure to correctly pick a member of the group described in G.

If $r = 0$ **then**

$r \leftarrow k$

For $i \leftarrow 1$ **to** n **do**

$r \leftarrow r - 1$

If $r = 0$ **then**

$t \leftarrow Max(t, a_i) + m$ // Assume Max returns a larger input. Basically, it picks a later one of the finish time of the last trip or an arrival time of the last member in the group as a start time.

$r \leftarrow k$ // Reset the count for the next group after each trip.

Return t

Termination & **Runtime**

Termination and runtime analyses can be done together.

The procedure can be separated into two sections:

- At the beginning of the procedure, a sequence of kayakers is sorted in chronological order by the merge sort. It takes $\boldsymbol{O(nlogn)}$ time (proof is omitted for this part).
- Next, a for-loop runs with a counter from 1 to n. Since the counter variable i isn't modified inside the loop, it simply iterates each a_i exactly once and terminates after nth iteration with $\boldsymbol{O(n)}$ time.

  Overall complexity is $O(nlogn) + O(n) = \boldsymbol{O(nlogn)}$ time.