CS222 Homework 2

Algorithm Analysis & Deadline: 2020-9-28 Monday 24:00

Exercises for Algorithm Design and Analysis by Li Jiang, 2020 Autumn Semester

1. Consider the following job scheduling problem. We are given all at once N jobs with positive lengths l_1,l_2,...,l_n. We can schedule only one job at a time and once we start a job we must run it to completion. A schedule for a set of jobs is then the starting time for each job s_1,s_2,...,s_n. Find an efficient algorithm to schedule the jobs to minimize the total wait time, where the wait time for a job is the difference between the time the job arrived and when it finished. Since all jobs arrive at the same time $t = 0$, the wait time for job i simplifies to $l_i + s_i$ and so the problem is to schedule the jobs to minimize $\sum_{i=1}^{i=n}(l_i + s_i)$. Please design an algorithm based on greedy strategy to solve the above problem(Write a pseudocode) and prove that your algorithm is correct.

Solution. *We need to minimize $\sum_{i=1}^{n} s_i$ because l_i are fixed values. And a natural strategy is to schedule jobs with short lengths first.*

Algorithm 1: Greedy-short-length-first

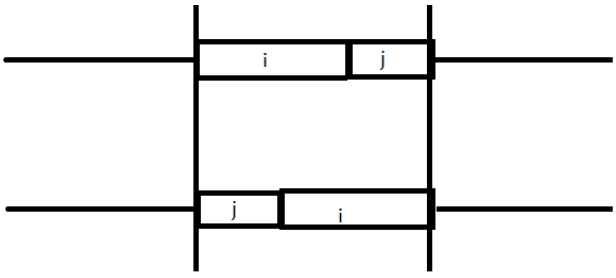
Input: $l_1, l_2, \cdots, l_n$
Output: $s_1, s_2, \cdots, s_n$

```
Sort n jobs so that l_1 ≤ l_2 ≤ ··· ≤ l_n
t ← 0
for j ← 1 to n do
    Schedule job j to start at s_j = t
    s_j ← t
    t ← t + l_j
end
return s_1, s_2, ··· , s_n
```

(a) Correct: $s_{j+1} - s_j = l_j$ according to the algorithm. So no two jobs run at the same time.

(b) Optimal: Obviously the optimal solution must have no idle time. Assume there is an optimal solution S^* different to the solution got from our greedy algorithm above. There must be such adjacent i, j that $s_i < s_j, l_i > l_j$ in S^*. We exchange job i with job j. Then the total wait time decreases by $\delta_t = l_i - l_j > 0$. So solution S^* cannot be optimal. Contradict! So the optimal solution must have a ascending order of l_i, hence our greedy algorithm above will get the optimal solution.

Figure 1: sketch map for problem 1



(c) Efficient: $T(n) = \Theta(n \log n)$.

Consider the above problem, but associate with the jobs positive values of importance $w_1, , w_n$, and minimize $\sum_{i=1}^{i=n} w_i(l_i + s_i)$. Find an efficient algorithm to determine an optimal schedule in this case. Please design an algorithm based on greedy strategy to solve the above problem(Write a pseudocode). You need not prove that your algorithm is correct.

Solution. *Sort by* $\frac{l_i}{w_i}$.

Algorithm 2: Greedy-smallest-l/w-first

Input: $l_1, l_2, \cdots, l_n$
Output: $s_1, s_2, \cdots, s_n$

Sort n jobs so that $\frac{l_1}{w_1} \leq \frac{l_2}{w_2} \leq \cdots \leq \frac{l_n}{w_n}$
$t \leftarrow 0$
for $j \leftarrow 1$ **to** n **do**
  Schedule job j to start at $s_j = t$
  $s_j \leftarrow t$
  $t \leftarrow t + l_j$
end
return $s_1, s_2, \cdots, s_n$

2. Considering the following Problem, and show your algorithm with pseudocode. Please ensure that the complexity of your algorithm is $O(n)$.
You have an array "prices" for which the *ith* element is the price of a given stock on day i. Design an algorithm to find the maximum profit. You may complete as many transactions as you like (i.e., buy one and sell one share of the stock multiple times).

Note: You may not engage in multiple transactions at the same time (i.e., you must sell the stock before you buy again).

Example1: the input array is : [7,1,5,3,6,4]. The output of the maximum profit is 7.

Solution. *Assume* p_i *is monotone increasing in intervals* $[l_1, r_1], [l_2, r_2], \cdots, [l_k, r_k]$. *The optimal solution is* $profit = \sum_{i=1}^{k} p_{r_i} - p_{l_i} = \sum_{i=2}^{n} \max\{0, p_i - p_{i-1}\}$, *because it can be checked easily that buying at* l_i *and selling at* r_i *is better than buying at* $l_i + \epsilon_{l_i}$ *and selling at* $r_i + \epsilon_{r_i}$.

Algorithm 3: Greedy-low-buy-high-sell

Input: $p_1, p_2, \cdots, p_n$
Output: $profit$

$profit \leftarrow 0$
for $i \leftarrow 2$ **to** n **do**
  $profit \leftarrow profit + \max\{0, p_i - p_{i-1}\}$
end
return $profit$

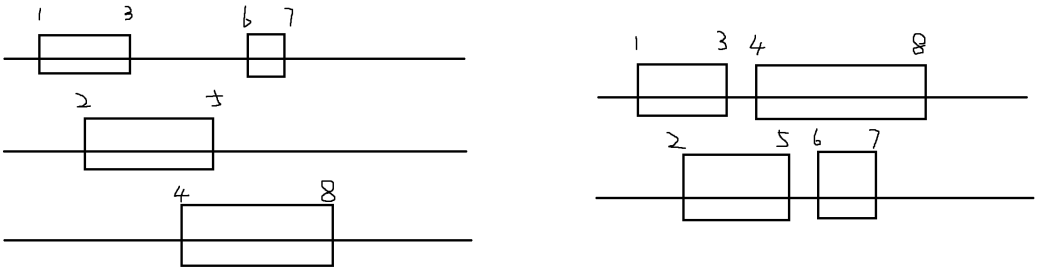
$T(n) = O(n)$

3. Consider the following Interval Problem.
INPUT:A set $S = \{(x_i, y_i) | 1 \leq i \leq n\}$ of intervals over the real line. Think of interval (x_i, y_i) as being a request for a room for a class that meets from time x_i to time y_i.
OUTPUT: Find an assignment of classes to rooms that uses the fewest number of rooms.
Note: that every room request must be honored and that no two classes can use a room at the same time.

Consider the following iterative algorithm. Assign as many classes as possible to the first room(sorted by end time), then assign as many classes as possible to the second room, then assign as many classes as possible to the third room, etc. Is this algorithm correct? Prove the correctness of such idea, or else provide a counter-example, and design your algorithm.

Solution. *This algorithm described above (sort by end time in ascending order) is not optimal. Counterexample: Consider intervals* $(1,3), (2,5), (6,7), (4,8)$. *The solution got by the algorithm above is classroom1:* $(1,3), (6,7)$, *classroom2:* $(2,5)$, *classroom3:* $(4,8)$. *Need 3 rooms. However, a better solution is classroom1:* $(1,3), (4,8)$, *classroom2:* $(2,5), (6,7)$. *Need 2 rooms.*

Figure 2: Counterexample for problem 3



My algorithm: sort the classes by start time. If a class can be add to a classroom, then add it. If a class conflicts with all the classroom allocated, then allocate a new classroom and add this class into it.

Algorithm 4: Greedy-start-time-early-first

Input: $(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)$
Output: $class_1, class_2, \cdots, class_n$

```
1  number ← 0
2  for i ← 1 to n do
3  |  assigned ← false.
4  |  Try to assign a room.
5  |  for j ← 1 to number do
6  |  |  if end_j ≤ x_i then
7  |  |  |  class_i ← j
8  |  |  |  end_j ← y_i
9  |  |  |  assigned ← true
10 |  |  |  break
11 |  |  end
12 |  end
13 |  if assigned = false then
14 |  |  Allocate a new room.
15 |  |  number = number + 1
16 |  |  class_i ← number
17 |  |  end_number ← y_i
18 |  end
19 end
20 return class_1, class_2, ..., class_n
```

The correctness and the optimality are proved in the class. The main proving strategy is to show there are $number lessons cover each other in certain interval, hence the number of rooms needed cannot be less than $number. $T(n) = \Theta(n^2)$, using binary heap can reduce it to $\Theta(n \log n)$.

However, if sort by end time in descending order, and assign as many classes as possible, then it is optimal. The proving strategy is similar to the one above. Assume that we use d classrooms. And there is a class $t : (x_t, y_t)$ in classroom d. Class t is incompatible with a class t_i in classroom i, $1 \leq i < d$. We claim that $\forall 1 \leq i < d, x_{t_i} < y_t$ and $y_{t_i} \geq y_t$, because class t is incompatible with class t_i and t is assigned after t_i. So there is at least d classes cover each other in the non-empty interval $(\max\{\max\{x_{t_i}\}, x_t\}, y_t)$. So d is the least number of classrooms needed, the algorithm is optimal.

4. Considering the following Problem, and show your algorithm with pseudocode. Please ensure that the complexity of your algorithm is $O(n)$.
Given an array of non-negative integers, you are initially positioned at the first index of the array. Each element in the array represents your maximum jump length at that position. Your goal is to reach the last index in the minimum number of jumps.

Note:You can assume that you can always reach the last index.

Example1: Input: [2,3,1,1,4]; Output: 2 Explanation: The minimum number of jumps to reach the last index is 2. Jump 1 step from index 0 to 1 (in index 0 you can choose to jump to index1 or index2), then 3 steps to the last index(in index 1 you can jump to index2, index3 or index4).

Solution. *Assume now we are at position i, and we can jump to the position $i+1, \cdots, i+a_i$. Which target position is the best? If $i+1 \leq s, t \leq i+a_i$ and $s+a_s < t+a_t$, then the target t is better than s, because any position can be reached from s by 1 step can be also reached from t by 1 step. So our strategy is to choose the target $i < j \leq i+a_i$ with the largest jump range $j+a_j$.*

Algorithm 5: greedy-largest-jump-range-first

Input: $a_1, a_2, \cdots, a_n$
Output: *minjump*

```
goal ← n
pos ← 1
i ← pos + 1
tojump ← i
minjump ← 0
while pos < goal do
    if i = goal then
        pos ← i
        minjump ← minjump + 1
        break
    end
    if i + a_i > tojump + a_tojump then
        tojump ← i
    end
    if i = pos + a_pos then
        pos ← tojump
        minjump ← minjump + 1
    end
    i ← i + 1
end
return minjump
```

Some property:

(a) *If jump from i to j, then $i + a_i < j + a_j$. Reason: First we say $i + a_i \leq (i + a_i) + a_{i+a_i} \leq j + a_j$; Second if $i + a_i = j + a_j$ we cannot jump to the goal in any jumping strategy.*

(b) *To find the best target* t *to jump from* j*, we only consider the positions after* $i + a_i$*, because all the positions* t' *before* $i + a_i$ *satisfy* $t' + a_{t'} \leq j + a_j \leq (j + a_j) + a_{j+a_j} \leq t + a_t$.

(c) $T(n) = \Theta(n)$. *In every iteration, variable* i *increases by* 1. *When* i *reaches goal* $= n$*, the loop finishes. Hence it iterates at most* n *times.*