CSE 431/531: Algorithm Analysis and Design Spring 2022

Homework 2

Instructor: Shi Li **Deadline: 3/12/2022**

Your Name: _____ Your Student ID: _____

Problems	1	2	3	4	Total
Max. Score	15	15	25	25	80
Your Score					

Problem 1 Consider the interval scheduling problem given by a set $[n] := \{1, 2, \cdots, n\}$ of activities, each activity $i \in [n]$ with a starting time s_i and finish time $f_i > s_i$. Decide whether each of the following two decisions is safe of not. If the answer is yes, give a proof using the exchanging argument; if the answer is no, give a counterexample.

(1a) Schedule the activity $i \in [n]$ with the latest starting time.

(1b) Pick the longest activity $i \in [n]$. If i conflicts with some other job in $[n]$, then we do not schedule i; otherwise we schedule i.

(1a) The decision is safe. Let S be an optimum schedule for the instance. If S contains i, then we are done. So assume S does not contain i. Let i' be the last job in S. Then $s_{i'} \le s_i$ as i is the job with the latest starting time. All the jobs in $S \setminus \{i'\}$ finish at or before $s_{i'} \le s_i$. So $S \setminus \{i'\} \cup \{i\}$ is a valid schedule, and is also optimum. Therefore in any case there is an optimum schedule that contains the job i.

(1b) The decision is not safe. Consider the instance with 3 jobs: $s_1 = 0, t_1 = 5, s_2 = 4, t_2 = 6, s_3 = 5, t_3 = 7$. Then the unique optimum solution is $\{1, 3\}$. However, job 1 is the longest job and it conflicts with job 2. By the decision, we will not schedule 1 and will miss the optimum solution. So the decision is not safe.

Problem 2 Construct the Huffman code (i.e, the optimum prefix code) for the alphabet $\{\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{e}, \mathsf{f}, \mathsf{g}\}$ with the following frequencies:

Letters	a	b	c	d	e	f	g
Frequencies	13	42	51	24	25	70	58

You need to give the codes (i.e, binary strings) for all the letters, and the weighted length of the codes (i.e, the sum over all letters the frequency of the letter times the length of its code).

Codes: a: 0000, b: 100, c:101, d: 0001, e: 001, f: 11, g: 01.

weighted length $= 4 \times 13 + 3 \times 42 + 3 \times 51 + 4 \times 24 + 3 \times 25 + 2 \times 70 + 2 \times 58 = 758$.

Problem 3 We are given an array A of length n and an integer k with $1 \leq k \leq n$. For every integer i in $\{k, k+1, \cdots, n\}$, let b_i be the sum of the k biggest numbers in the array $A[1..i]$. The goal of the problem is to output $b_k, b_{k+1}, b_{k+2}, \cdots, b_n$ in $O(n \log k)$ time, using the heap data structure.

You can assume that the heap data structure (min-heap or max-heap) and the procedures such as insert, get-min(or get-max), extract-min(or extract-max), increase-key, decrease-key are provided to you. If the running time and correctness of the algorithm are easy to see, a pseudo-code is sufficient.

For example, if $n = 10, k = 3$ and $A = (50, 80, 10, 30, 90, 100, 20, 40, 105, 95)$. Then you should output $140, 160, 220, 270, 270, 270, 295, 300$ as demonstrated by the following table:

i	3 largest numbers in $A[1..i]$	b_i
3	50, 80, 10	140
4	50, 80, 30	160
5	50, 80, 90	220
6	80, 90, 100	270
7	80, 90, 100	270
8	80, 90, 100	270
9	90, 100, 105	295
10	100, 105, 95	300

The pseudo-code is as follows, and the running time is $O(n \log k)$.

```
1: Q ← empty min-heap, sum ← 0
2: for i ← 1 to k do
3:     Q.insert(i, A[i]), sum ← sum + A[i]
4: b_k ← sum
5: for i ← k + 1 to n do
6:     Q.insert(i, A[i]), j ← Q.extract_min(), sum ← sum + A[i] − A[j]
7:     b_i ← sum
8: return (b_k, b_{k+1}, ··· , b_n)
```

Problem 4 Given a set of n points $X = \{x_1, x_2, \cdots, x_n\}$ on the real line, we want to use the smallest number of unit-length *closed* intervals to cover all the points in X. For example, the points X in Figure 1 can be covered by 3 unit-length intervals.

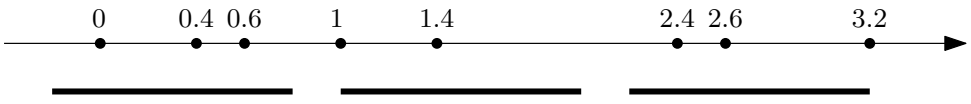

Figure 1: Using 3 unit-length intervals (denoted by thick lines) to cover points in X (denoted by the solid circles).

Design a greedy algorithm to solve the problem; it suffices for your algorithm to run in

polynomial time. To prove the correctness of the algorithm, you can follow the following steps:

- Design a simple strategy to make a decision.
- Prove that the decision is safe.
- Give the reduced instance after you made the decision.

- Assume the set of points is not empty. Let x_i be the smallest number in $\{x_1, x_2, \cdots, x_n\}$. Then, we include the interval $[x_i, x_i + 1]$ in the solution.
- Assume we do not include the interval $[x_i, x_i + 1]$ in the solution. Since the point x_i must be covered, there is some interval $[s, s + 1]$ that covers x_i; i.e, $s < x_i \leq s + 1$. Then the set of points in $\{x_1, x_2, \cdots, x_n\}$ covered by $[x_i, x_i + 1]$ is a super-set of the points covered by $[s, s + 1]$. Therefore, removing $[s, s + 1]$ from the solution and adding $[x_i, x_i + 1]$ still gives an optimum solution. So, we have an optimum solution that contains the interval $[x_i, x_i + 1]$.
- We remove all the points in $\{x_1, x_2, \cdots, x_n\}$ that are covered by $[x_i, x_i + 1]$ from the set. The residual instance is defined by the set of remaining points.