CSE 431/531: Algorithm Analysis and Design Fall 2022

Homework 2 Solutions

Instructor: Shi Li **Deadline: 10/9/2022**

Your Name: _____ Your Student ID: _____

Problems	1	2	3	Total
Max. Score	20	30	30	80
Your Score				

Problem 1 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:

Symbols	a	b	c	d	e	f	g
Frequencies	50	20	27	25	29	85	55

Also give the weighted length of the code (i.e, the sum over all symbols the frequency of the symbol times its encoding length).

Symbols	a	b	c	d	e	f	g
Frequencies	50	20	27	25	29	85	55
Encoding	001	0000	100	0001	101	01	11

Weighted length $= 50 \times 3 + 20 \times 4 + 27 \times 3 + 25 \times 4 + 29 \times 3 + 85 \times 2 + 55 \times 2 = 150 + 80 + 81 + 100 + 87 + 170 + 110 = 778$.

Problem 2 We are given an array A of length n. For every integer i in $\{1, 2, 3, \cdots, n\}$, let b_i be median of the sub-array $A[1..i]$. (If the sub-array has even length, its the median is defined as the lower of the two medians. That is, if i is even, b_i is the $i/2$-th smallest number in $A[1..i]$.) The goal of the problem is to output $b_1, b_2, b_3, \cdots, b_n$ in $O(n \log n)$ time.

For example, if $n = 10$ and $A = (110, 80, 10, 30, 90, 100, 20, 40, 35, 70)$. Then $b_1 = 110, b_2 = 80, b_3 = 80, b_4 = 30, b_5 = 80, b_6 = 80, b_7 = 80, b_8 = 40, b_9 = 40, b_{10} = 40$.

Hint: use the heap data structure.

We maintain two heaps, one max-heap and one min-heap. In the i-th iteration of our algorithm, the max-heap contains the $\lceil i/2 \rceil$ smallest numbers in $A[1..i]$, and the min-heap contains the $\lfloor i/2 \rfloor$ biggest numbers in $A[1..i]$. Then, the lower median of $A[1..i]$ is the maximum element in the maximum heap. The pseudo-code gives how to maintain the two heaps:

```
1: maxheap ← an empty max-heap
2: minheap ← an empty min-heap
3: for i ← 1 to n do
4:     if i is odd then
5:         if A[i] ≤ minheap.get-min() then
6:             maxheap.add(A[i], A[i])
7:         else
8:             t ← minheap.extract-min(), maxheap.add(t, t), minheap.add(A[i], A[i])
9:     else                                                    ▷ i is even
10:        if A[i] ≥ maxheap.get-max() then
11:            minheap.add(A[i], A[i])
12:        else
13:            t ← maxheap.extract-max(), minheap.add(t, t), maxheap.add(A[i], A[i])
14:    b_i ← maxheap.get-max()
15:    return b
```

In the pseudo-code, extract-min (extract-max) for the min-heap (max-heap) returns and removes the minimum (maximum) element. get-min (get-max) for the min-heap (max-heap) returns but does not remove the minimum (maximum) element. If the min-heap (max-heap) is empty, then get-min (get-max) returns ∞ ($-\infty$). The running time in each iteration of Loop 3 is $O(\log n)$ and thus the overall running time of the whole algorithm is $O(n \log n)$.

Problem 3 In the interval covering problem, we are given n intervals $(s_1, t_1], (s_2, t_2], \cdots, (s_n, t_n]$ such that $\bigcup_{i\in[n]}(s_i, t_i] = (0, T]$. The goal of the problem is to return a smallest-size set $S \subseteq \{1, 2, 3, \cdots, n\}$ such that $\bigcup_{i\in S}(s_i, t_i] = (0, T]$.

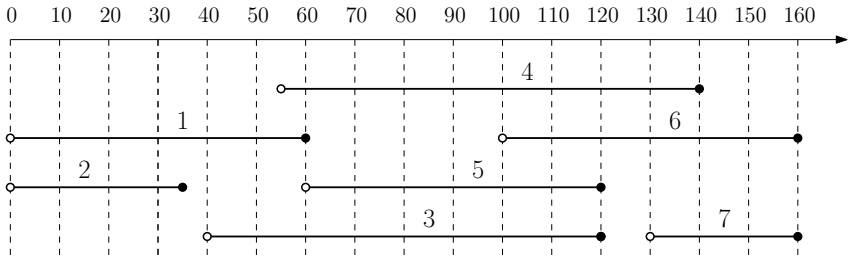

Figure 1: An instance of the interval covering problem.

For example, in the instance given by Figure 1. The intervals are $(0, 60], (0, 35], (40, 120], (55, 140], (60, 120], (100, 160], (130, 160]$. Then we can use 3 intervals indexed by 1, 4, 7 to cover the interval $(0, 160]$. This is optimum since no two intervals can cover $(0, 160]$.

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, it is recommended that you can follow the following steps:

- Design a simple strategy to make a decision for one step.

  Since $\bigcup_{i\in[n]}(s_i, t_i] = (0, T]$, there is an interval $(s_i, t_i]$ such that $t_i = T$. Consider the following greedy strategy: let $i^* = \arg\min_{i:t_i=T} s_i$ and add i^* to S. In words,

among all the intervals with right end point being T, we choose the one with the leftmost left end point, and add it to S.

- Prove that the decision is safe.

  We need to show that there is an optimum solution S^* to the instance such that $i^* \in S$. Take an arbitrary optimum solution S' to the instance. If $i^* \in S'$ then there is nothing to prove. So we assume $i^* \notin S'$. Since $\bigcup_{i \in S'}(s_i, t_i] = (0, T]$, there exists an $i \in S'$ such that $t_i = T$. Let $S^* = S' \setminus \{i\} \cup \{i^*\}$. By our choice of i^*, we have $(s_{i^*}, t_{i^*}] \supseteq (s_i, t_i]$. Thus, S^* is also a valid solution; since $|S^*| = |S'|$, it is also an optimum solution to the instance. Thus, we proved that the strategy is safe.

- Describe the reduced instance after you made the decision.

  After we selected $(s_{i^*}, t_{i^*} = T]$, the remaining task is to choose a minimum-size set S such that $\bigcup_{i \in S}(s_i, t_i] \supseteq (0, s_{i^*}]$. This is equivalent to the following updated interval covering instance. For every $i = 1, 2, \cdots, n$, if $(s_i, t_i] \subseteq (s_{i^*}, t_{i^*}]$, then we remove interval $(s_i, t_i]$; otherwise, we update the interval to $(s_i, \min\{t_i, s_{i^*}\}]$. Finally we update T to s_{i^*}.