CSC11F: Advanced Data Structures and Algorithms

June 8, 2020 Final Report

Hoang Tuan Linh (m5232108@u-aizu.ac.jp)

1 Problem A: Sliding Minimum Element

Problem description

For a given array $a_1, a_2, a_3, ..., a_N$ of N elements and an integer L, find the minimum of each possible sub-arrays with size L and print them from the beginning. For example, for an array $\{1, 7, 7, 4, 8, 1, 6\}$ and $L = 3$, the possible sub-arrays with size $L = 3$ includes $\{1, 7, 7\}, \{7, 7, 4\}, \{7, 4, 8\}, \{4, 8, 1\}, \{8, 1, 6\}$ and the minimum of each sub-array is $1, 4, 4, 1, 1$ respectively.

Constraints and critical cases of input

- $1 \leq N \leq 10^6$
- $1 \leq L \leq 10^6$
- $1 \leq a_i \leq 10^9$
- $L \leq N$

Description of data structure and algorithm

A segment tree (ST) is employed to solve this problem because this data structure allows answering queries on a range of a given array efficiently. The ST is designed with two operations:

- update(i, x): change a_i to x.
- findMin(i, j): report the minimum value in the range $a_i, a_{i+1}, ...a_j$.

To find the minimum value of a sub-array with length L starting from a_i, i.e. the sub-array $a_i, a_{i+1}, ...a_{i+L-1}$, we call the operation find$(i, i + L - 1)$.

Time and space complexity

- Time complexity: $O(N \log N)$. The update(i, x) query takes $O(\log N)$, we call it N times when initiating N elements of the array. To report the minimum values of all possible sub-arrays, we call the operation find(i, j) $(N - L + 1)$ times, and each takes $O(\log N)$. In conclusion, the total time to solve the problem thus is $O(N \log N)$.
- Space complexity: the ST is of the size $(2N' - 1)$ where N' is the smallest power of 2 that is greater than or equal to N, i.e. $N' = 2^{\lceil \log_2 N \rceil}$.

Source code

The source code is in the file "W6A-m5232108.cpp" submitted along with this report.

2 Problem B: Range Query on a Tree

Problem description

Write a program which manipulates a weighted rooted tree T with the following operations:

- $add(v, w)$: add w to the edge which connects node v and its parent
- $getSum(u)$: report the sum of weights of all edges from the root to node u

The given tree T consists of n nodes and every node has a unique ID from 0 to $n - 1$ respectively where ID of the root is 0. Note that all weights are initialized to zero.

Constraints and critical cases of input

- All the inputs are given in integers
- $2 \leq n \leq 100000$
- $c_j < c_{j+1}$ $(1 \leq j \leq k - 1)$
- $2 \leq q \leq 200000$
- $1 \leq u, v \leq n - 1$
- $1 \leq w \leq 10000$

Description of data structure and algorithm

The data structure to solve this problem is the segment tree. The algorithm is as follows:

1. First, we convert the rooted weighted tree into an array of indexes of nodes, called arr[], based on a pre-order traversal on the given tree. In particular, nodes are indexed in arr[] based on the order of appearance when we deploy a depth-first-search (DFS) operation on the given tree, starting from the root. Besides, during the DFS operation, for each node, say node v, we track the index in arr[] of v and the right-most node of the sub-tree rooted at v; then save them in arrays left[] and right[], respectively. Let's say the index of node v in the array arr[] is i, i.e. arr$[i] = v$, we have left$[v] = i$, and right$[v]$ saves the index of the right-most node of the sub-tree rooted at v. By doing so, we can represent a sub-tree of the given tree as a sub-array of the array arr[]. In particular, the sub-tree rooted at node v is represented by the sub-array arr[left$[v]$], arr[left$[v]+1$], ... arr[right$[v]$]. Fig. 1 illustrates the weighted rooted tree and the corresponding arrays arr[], left[], and right[] for the Sample Input 1.

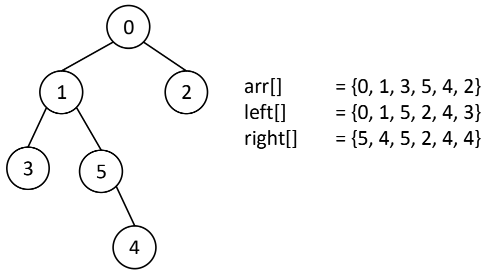

Figure 1: A weighted rooted tree with six nodes and the corresponding arrays arr[], left[], and right[].

2. Second, we initiate an array called dis[] to save the sum of weights of all edges to a node, i.e. the total distance from the root to that node. The i-th element of dis[], i.e. dis$[i]$, will save the total distance corresponding to the node with index saved at the i-th element of arr[], i.e. arr$[i]$. After adding w to an edge, say, the edge pointing to node v, we must update the total distance of node v and all of its children nodes. To do so, we simply add w to all elements of dis[] ranging from left$[v]$ to right$[v]$, i.e. add w to dis[left$[v]$], dis[left$[v]$+1], ... dis[right$[v]$]. To answer that range query on dis[], we employ a segment tree (ST) with two operations:

   - update(i, j, w): add w to dis$[i]$, dis$[i+1]$, ... dis$[j]$
   - read(i): report the value of dis$[i]$

   In particular, for $add(v, w)$ operation, we call update(left$[v]$, right$[v]$, w) query; for $getSum(u)$ operation, we call read(left$[u]$) query. Note: since all weights are initialized to zero, all elements in dis[] are also initialized as zero.

3. To further improve the efficiency of the segment tree, we might employ the lazy evaluation technique. In particular, an update related to a node, say node v, will be executed immediately at node v but not for children nodes of v. Updates related to children nodes of v are saved in an array called lazy[] and only executed when needed (i.e., when the update is needed to be done before executing another query).

Time and space complexity

- Time complexity: For each $getSum$ or add query, it takes $O(\log n)$, where n denotes the number of nodes in the given tree. We must execute q queries, the total time of the algorithm thus is $O(q \log n)$.
- Space complexity: For arrays arr[], left[], right[], and dis[], each of them is of the size n. The segment tree is of the size $(2n' - 1)$ where n' is the smallest power of 2 that is greater than or equal to n, i.e. $n' = 2^{\lceil \log_2 n \rceil}$. The size of the array lazy[] is equal to that of dis[], i.e. $(2n' - 1)$.

Source code

The source code is in the file "W6B-m5232108.cpp" submitted along with this report.