DSA-II Project
Range Minimum Queries

Ayush Garg, Ayush Garg, Rendla Aditya

IIT Gandhinagar

Abstract. Finding the range minimum query in a sub array has various practical applications. There are many improved algorithms that have been proposed through years. Some of the works on finding RMQs includes,

- Fischer and Heun introduced a new pre-processing scheme for RMQs that occupies 2n+o(n) bits in memory[1]
- Gabow et al.[2] reduced the RMQ problem to finding LCA (least common ancestor) in Cartesian tree
- Ferrada and Navarro implemented general tree made out of Cartesian tree and has made further optimizations to it[3]

In this project we had implemented Sparse Table algorithm and Optimized Sparse table algorithm.

1 Introduction

Index data structures are computed once and they are used to answer the queries without looking the whole data set again. In this project we are going implement two different algorithms which use index data structures.
Problem statement: Given an array A of size n. The range minimum query(RMQ) problem is to find an index structure that returns for any range A[i,,j] the position of the leftmost minimum.

$$RMQ(i, j) = argmin_{i \leq k \leq j} < A[k], k >$$

There are many straight forward algorithms to do this. One way is to precompute every possible query and store them in a table of size $O(n^2)$.

2 Algorithms

Definitions:

- Pre-processing time: It is the time taken to process the given data and to make required data structures that we use for answering further queries based on the given data.
- Query time: It is the elapsed between arrival and answering of query.

In this project we are going to implement Sparse table algorithm and Optimized Sparse table algorithm.

Sparse table algorithm

In this algorithm we make a table(M) of size $O(nlog(n))$. Here we precompute the table where the entry M[i][j] has $RMQ_A(i, i+2^j 1)$; i.e. for all queries with an interval size that is a power of two, the answers are stored. Any query $RMQ_A(i, j)$ can be calculating k, where k is the maximal number for which it satisfies the condition,

$$2^k \leq j - i + 1$$

Then after we compare $A[M[i, k]]$, $A[M[j - 2^k + 1, k]]$ and determine the result.
In this algorithm, pre-processing time is $O(nlog(n))$ and query time is $O(1)$.

Optimized Sparse table algorithm

Forward minimum array(F_A): It is an array of size equals the size(A). F[i] will contain the minimum of A[i,...,n-1] for all $i = 0, 1, 2, ..., n - 1$.

Backward minimum array(B_A): It is an array of size equals the size(A). B[i] will contain the minimum of B[1,..,i] for all $i = 0, 1, 2, ..., n - 1$.

Preprocessing:
In this algorithm we divide the given array in to blocks of size $\frac{log(n)}{2}$. For each block we will make both forward minimum and backward minimum arrays. Along with this we will make a binary tree over each block with the following properties.

- For each element we will make a node in the binary tree. These nodes will be the leaves of the tree. That is, this will be the last level of the tree
- Now each node in the binary tee will have the minimum of all its children nodes
- Each node will also store the indexes of the original array, for which it is storing the minimum. For example, the root will store the indexes (1,n). Since it stores the minimum of all its children which is essentially the whole array.

We will find minimum of each block and then make Sparse table over block minimals. This will take preprocessing time of $O(n)$, since preprocessing time is dominated for making forward and backward minimum arrays.

Query:
For a given query with indexes (i,j).

Case 1: If i and j lies in different blocks
Let b_i and b_j be the blocks in which i and j lies. Then using forward and backward minimum arrays with index i and j respectively, we will find the minimum elements in blocks b_i and b_j for the required range. The minimum over the blocks that are between the blocks b_i and b_j can be found using Sparse table. Then we will compare these three minimums and return the minimum of these three minimums. This takes $O(1)$ query time.

Case 2: If i and j lies in the same block
Then for the given indexes we will start from the root node and check

- if i and j lies to left of middle index then we will move to the left child of the node with same indexes i and j
- if i and j lies to right of middle index then we will move to the right child of the node with same indexes i and j
- if i and j lies on either side of the middle index. Then repeat the same process with the left node using indexes (i,middle index) and with the right node using indexes (middle index,j)
- if at any node i=left index storing in node and j=right index stored in node, then return the element in the node

In worst case query time is $O(log(logn))$.

Empirical evaluation plan

We are planning to make a data set of input arrays and corresponding queries using random number generator functions in C++. With respect to the input size of the array, we are going to compare the preprocessing time, query time for each array, to check and ensure the correctness of both the algorithms.

References

1. Fischer heun paper: https://arxiv.org/pdf/0812.2775.pdf
2. Gabow paper on LCA: H. N. Gabow, J. L. Bentley, R. E. Tarjan, Scaling and related techniques for geometry problems, in: Proc. 16th STOC, 1984, pp. 135143.
3. Ferrada and Nevarro paper: https://www.dcc.uchile.cl/ gnavarro/ps/jda16.pdf
4. http://drops.dagstuhl.de/opus/volltexte/ 2017/7615/pdf/LIPIcs-SEA-2017-12.pdf