GSourceCode

GSOURCECODE

INTRODUCTION TO SEGMENT TREES (RANGE MINIMUM QUERY)

12 Mar 2014 · *by* MGhareeb · *in Algorithms, Data Structure* · *Leave a comment*

1 0 Rate This

Range Minimum Query

Given an array A of length N, answer $getMin(i, j)$ qeuries. $getMin(i, j)$ should return the index m, where $A[m] = min_{i<k<j}\{A[k]\}$.

There are many ways to do this. In the case where A's element values are allowed to change, and yet we have to maintain $getMin(i, j)$ to work correctly, a segment tree is our best option. Follows is a brief explanation of segment trees for this specific example. I'll put a more generalized explanation in another post.

Segment Trees

The idea behind a segment tree is to build a **binary tree** where each node represents a segment of A. For Range Minimum Query, each tree node will contain the value of $getMin(segment)$ (the index of the minimum value in the node's segment of the array).

Now, how are nodes and segments assigned? For simplicity, let's assume that A's length is a power of two:

- Starting at the lowest level of the tree, we'll have N leaf nodes each representing an element of A. As shown in the following figure, if we label leaf nodes from 0 to $N - 1$ from left to right, each node i will contain the value i.
- For each internal node $node$ starting from the bottom, we compute $node.value$ as follows:

$node.value = m, A[m] = min\{A[node.left.value], A[node.right.value]\}$

ENTER YOUR EMAIL TO SUBSCRIBE

Enter your email address

Follow!

SEARCH

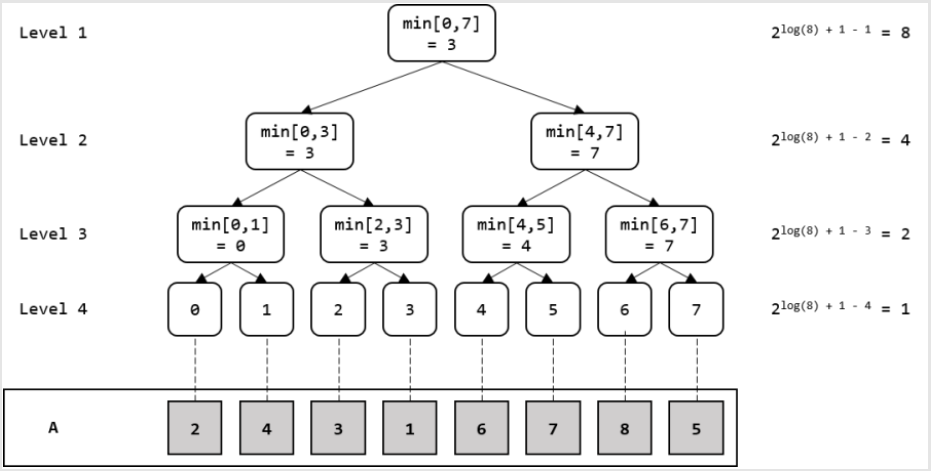
Search this site...

BROWSE A CATEGORY

- Algorithms
  - Ad-hoc
  - Binary Search
    - Problems
  - CompleteSearch
    - Problems
    - Utilities
  - Dynamic Programming
    - Problems
    - Utilities
  - Geometry
    - Problems
    - Utilities
  - Graph
    - Problems
    - Utilities
  - Heuristic Search
    - Utilities
  - Math
    - Problems
    - Utilities

Follow

. Meaning, a parent node $node$ will g
children such that $A[node.value]$ is
- Since this is a binary tree with the las
tree will have the height of $log(N)$ +
node, the root, which holds $getMin($
each node on a level L will represent
$2^{log(N)+1-L}$.



The implementation is going to be straight forward from here on. We'll represent the tree as an array `tree` of length `2 << ceil(log2(N + 1))`. Node `i` has a left child `2 * i` and a right child `2 * i + 1`. On an `update(i)`, we'll traverse the tree from top to bottom, change array value at when we reach the leaf `i`, and update the tree as we go back up. On a `getMin(range)`, we'll recurse from root until we visit all maximal sub-segments of the `range` and return the required value. Here's an implementation in `C++` with this tutorial as a reference:

```
#include <iostream>
#include <vector>
#include <cstring>
#include <cmath>
using namespace std;

class segTree {
    // O(n)
    int *array, *tree;
    int arrayLen, treeLen;

    // O(n)
    void initialize(int node, int b, int e) {
        if (b == e)
            tree[node] = b;
        else {
            // recurse
            initialize(2 * node, b, (b + e) / 2);
            initialize(2 * node + 1, (b + e) / 2
            // update value
            if (array[tree[2 * node]] <= array[tr
                tree[node] = tree[2 * node];
            else
                tree[node] = tree[2 * node + 1];
        }
```

```
    }
public:
    segTree(int *array, int arrayLen) {
        this->arrayLen = arrayLen;
        this->array = array;
        this->treeLen = 2 << (int)ceil(log2(array
        cout << "treeLen=" << treeLen << endl;
        this->tree = new int[treeLen];
        memset(tree, -1, sizeof(int) * treeLen);
        initialize(1, 0, arrayLen - 1);
    }

    // O(log n)
    void update(int i, int v, int node = 1, int b
        e = arrayLen - 1 - e;
        if (b == e) {
            array[i] = v;
        } else {
            int mid = (b + e) / 2;
            if (i <= mid)
                update(i, v, 2 * node, b, arrayLe
            else
                update(i, v, 2 * node + 1, mid +
            if (array[tree[2 * node]] <= array[tr
                tree[node] = tree[2 * node];
            else
                tree[node] = tree[2 * node + 1];
        }
    }

    // O(log n)
    int query(int i, int j, int node = 1, int b =
        e = arrayLen - 1 - e;
        // bad interval
        if (i > e || j < b)
            return -1;
        // good interval
        if (b >= i && e <= j)
            return tree[node];
        // partial interval
        int left = query(i, j, 2 * node, b, array
        int right = query(i, j, 2 * node + 1, (b
        if (left == -1)
            return tree[node] = right;
        if (right == -1)
            return tree[node] = left;
        if (array[left] <= array[right])
            return tree[node] = left;
        return tree[node] = right;
    }
};

int main() {
    int A[10] = { 2, 4, 3, 1, 6, 7, 8, 9, 1, 7 };
    segTree t(A, 10);
    cout << "getMin(0, 4) = " << t.query(0, 4) <<
    t.update(1, 0);
    cout << "getMin(0, 4) = " << t.query(0, 4) <<
    t.update(0, -1);
    cout << "getMin(0, 4) = " << t.query(0, 4) <<
    return 0;
}
```

You May Like

- 1.

Share

Related

TopCoder - SRM570
In "Ad-hoc"

Introduction to Binary Trees
In "Data Structure"

Heuristic Search & AI - 3
In "Algorithms"

Tags: ancestor, binary tree, common, data, dp, dynamic, dynamic programming, lca, lowest, max, maximum, min, minimum, programming, query, range, rmq, segment, segtree, structure, tree

std::cout <<

Enter your comment here...

← 315 – Network

- Introduction to Dynamic Programming - Cutting Rods
- 272 - TEX Quotes
- 10684 - The jackpot
- gsourcecode

AUTHORS

MGhareeb

Algorithms

Android Binary Search CompleteSearch Data Structure Desktop Applications Dynamic Programming Geometry Graph Heuristic Search J2ME Math Operations Research Problems Utilities

- Log in
- Entries RSS
- Comments RSS
- Create a free website or blog at WordPress.com.