Segment Trees: Applications

HKU ACM Team

Static and Dynamic RMQ

- RMQ: Range Minimum Query.
- Static version: given an array A[1, 2, .., n], query the minimum in interval [x, y].
- Dynamic version: the array can be updated between queries. For example, the update is to set each A[i] to be c for i in [x, y].

Static RMQ: Sparse Table (ST)

- We introduce a O(n log n) preprocessing, O(1) query approach.
- Preprocessing. Build a table T[i, j], such that T[i, j] is the minimum between [j, j + 2^i -1]. That is, the minimum in the interval of length 2^i starting at j.
- T[0, j] = A[j]
- Recursion for i>0: T[i, j] = min(T[i-1, j], T[i-1, j + 2^(i-1)])
- We calculate the table for log(n) rows.
- Suppose the query is [x, y]. Define l to be such that 2^l <= len([x, y]) <2^(l+1).
- Fact: [x, y] is covered by [x, x + 2^l - 1] and [y – 2^l +1, y]
- Therefore, rmq[x, y] = min(T[l, x], T[l, y-2^l+1]).
- http://acm.timus.ru/problem.aspx?space=1&num=1126

Dynamic RMQ: Segment Trees

- Maintain filed “min” in Segment Tree.
- Push up: min = min(L.min, R.min).
- Preprocessing: O(n) space, O(n) time.
- Query & Update: O(log n) time.

Dynamic maximum subarray sum

- Maximum subarray sum: given array A[1, 2, …, n], what is the maximum sum of a continuous subarray.
- Dynamic version: modify some of the values, and query the maximum sum after each modification.
- Segment Tree solution: what to maintain?

Divide and Conquer

- Segment Tree uses the idea of divide and conquer. For example, min = min(L.min, R.min), sum = L.sum + R.sum.
- How about maximum subarray sum?
- Suppose we have solved the problem in L and R subintervals. How do we use them to get the answer for the whole interval in O(1) time?
- The maximum of the whole interval, is the maximum of:

1) The maximum of the left interval
2) The maximum of the right interval
3) The maximum between the left and right intervals.

- (1) and (2) are solved recursively (and known to us now). The maximum between the left and right intervals, is the maximum from the right of the left interval + maximum from the left of the right interval.

Maintaining the Fields

- Maintain: leftMax, rightMax, totalMax.
- leftMax is the maximum subarray sum over the intervals with starting point being the left endpoint.
- rightMax is defined similar, which denotes the maximum subarray sum over the intervals with starting point being the right endpoint.
- totalMax is the actual maximum subarraysum
- Then:

leftMax = max(L.leftMax, L.sum + R.leftMax)

rightMax = max(R.rightMax, R.sum + L.rightMax)

totalMax = max(L.totalMax, R.totalMax, L.rightMax+R.leftMax)

- http://www.spoj.com/problems/GSS3/

Composition of Updates

- Consider the following two kinds of updates:

1. Add c to A[i] for i in [x, y]
2. Set A[i] = c for i in [x, y]

- We have to maintain two fields in order to account for update 1 and update 2 in lazy update. In our case, we use “add” to record the updates 1, and use “set” to address update 2. We use a boolean variable “lazy” to denote the node needs lazy update.

Add & Set

- If it is update 1, we update add+=c, and lazy = true. If it is update 2, we record set = c, and lazy = true
- The above is incorrect, because it cannot distinguish the ordering.
- (a) add 10 to [1, 8]; set [1, 8] to be 5 (b) set [1, 8] to be 5; add 10 to [1, 8]
- The “add” and “set” fields are the same in the two different examples.
- Solution: at any time, we make only one of the two fields effective, and make the state consistent. The rule is the following:

1. For Add(x, y, c) update, if currently the add field is effective, then keep using it; otherwise, the set field is effective, we update set+=c.

2. For Set(x, y, c) update, if currently the add field is effective, then make it ineffective, clean the add field, and make set field effective while setting set = c. Otherwise, currently the set field is effective, keep using it and set = c.

- UVA 11992

Dual Fields

- Consider the following problem:
- Update(x, y): neglect A[i] for i in [x, y] (A[i] = -A[i]).
- Query(x, y): return the minimum of A[i] for i in [x, y].
- Obviously, we have to maintain “min” field.
- However, for some interval, after neglecting, we don’t know what is min.
- Observation: actually, the old min now is the new max, and the new min is the old max!

Neglect & min

- Maintain min and max simultaneously.
- When neglecting, swap min and max.
- Min and max are dual variables. We need dual variables, if the update is some kind of "neglecting".
- Similar situation: we are maintaining a Boolean array, and the update is to flip the values in an interval (i.e., 1 -> 0, 0 -> 1).
- http://acm.hdu.edu.cn/showproblem.php?pid=3911

Dynamic Subtree Sum

- Segment Tree can be applied to other problems that require fast interval operation.
- In dynamic subtree sum problem, we are given a tree with n nodes, and each node has a value. We are required to do the following:

1. Update(x, c): add c to the value of nodes that are decedents of x.
2. Query(x): return the sum of the value of nodes that are decedents of x.

- We show that this problem can be converted to interval modification problem that can be solved by using segment tree.

Preorder Sequence

- Preorder transversal for general tree: visit root first, and then visit the children in any order. That is:

```
Preorder(root):
  if root !=null:
    visit(root)
    for c in children of root:
      visit(c)
```

We call the sequence of the visited nodes as preorder sequence. Note that the sequence is a permutation of [n], where n is the number of nodes in the tree.

- Property: for any x, the nodes in the subtree rooted at x is a continuous interval in the preorder sequence.
- This property is true because of the depth-first ordering.
- The position of the two endpoints for each node can be preprocessed in linear time also.
- Therefore, the dynamic subtree sum is converted to a standard interval problem, by preprocessing the preorder sequence.
- http://acm.timus.ru/problem.aspx?space=1&num=1890

The k-th smallest element

- Suppose we are maintaining a permutation of [n].
- Del(x): delete element A[x]
- Query(k): return the k-th smallest elements left.
- We build a segment tree with n leaf nodes. We maintain the sum of intervals.
- The sum of a leaf [x, x] is 1, if there exists i such that A[i] = x, otherwise it is 0.
- Then, for some interval [x, y] in the Segment Tree, the sum means how many elements in A that have value between x and y.
- So, to find the k-th smallest, it is equivalent to find the smallest value x, such that sum[1, x] = k, and x is the k-th smallest value.
- This can be done by using binary search over x, and sum[1, x] can be calculated in O(log n) using Segment Tree.
- In total, we update in time O(log n), and query in time O(log^2(n)).
- http://acm.timus.ru/problem.aspx?space=1&num=1521

The k-th smallest element - O(log n) approach

- If we use Segment Tree as a black box, then it will be O(log^ n) for query.
- Modify the query: do not use binary search, do not query the sum of the interval. But we still have to maintain sum. If the left child has sum no less than k, then we go left; otherwise we go right. Then, the k-th smallest element is always in the current interval. When we stop at a leaf, we find the k-th smallest element.

Query(r, k):

- push-down(r);

```
if r is leaf:
    return r
if r.leftChild.sum >= k:
    return Query(r.leftChild, k)
else:
    return Query(r.rightChild, k-r.leftChild.sum)
```

Number of Unordered Pairs

- We are given an array A that is a permutation of n, and we are to calculate the number of unordered pairs. A pair (i, j) is unordered, if i<j and A[i]>A[j].
- Naïve approach is O(n^2).
- A classical divide and conquer approach using the idea of merge sort gives O(n log n) running time.
- O(n log n) running time can also be obtained by using Segment Tree:

```
for i from 1 to n:
    pairs += tree.sum(A[i]+1, n)
    tree.set(A[i], 1)
```

- Here, the sum of an interval [x, y] is the number of element in A that are of the value between x and y.
- In i's iteration, all the A[j]'s for j<i are set to 1 in the tree. By querying the sum[A[i]+1, n], we know the number elements A[j] such that j<i and A[j]>A[i].
- http://acm.hdu.edu.cn/showproblem.php?pid=2492