Sqrt Decomposition

In this article, we will describe a technique for solving a wide range of problems pertaining to the update-query model. For the purposes of this article, all arrays are indexed from 1 unless stated otherwise.

Let us consider the following problem. You are given an array A of n integers and you would be asked to perform Q operations on the array where each operation can be of any one of the two types:

1. Given an index i and a value v, increase the value of a[i] by v. This we will denote by Update(i,v)
2. Given two indices l and r, print the sum $\sum_{i=l}^{r} a[i]$. This we will denote by Query(l,r)

Let us call the first operation as the update operation and the second one as the query operation.

First,let us try the obvious solution and see how that turns out.
For the first operation, we can just add an update rule: $a[i] += v$. This takes $O(1)$ time.
For the second operation, we can iterate over all the elements from indices l to r and calculate the sum. This takes $O(n)$ time.

In the worst case, every query can be of the form Query(1,n) and so for Q queries, the worst case running time would be $O(Qn)$.

Let us consider another solution. Suppose in addition to the array A, we also maintain another array $pre[1, \cdots, n]$, where

$$pre[1] = a[1]$$
$$pre[i] = pre[i-1] + a[i], \quad 2 \leq i \leq n$$

This array is usually called the prefix sum array. Let us see if the prefix sum array reduces the overall running time complexity.

For the second operation, we can answer Query(l,r) as follows.

If $l = 1$, then the answer is just $pre[r]$.

If $l > 1$, then the answer is

$$a[l] + \cdots + a[r] = a[1] + \cdots + a[r] - a[1] - \cdots - a[l-1] = pre[r] - pre[l-1]$$

Hence, each query operation takes $O(1)$ time.

But now Update(i,v) requires that we do $pre[j] += v, \forall j$, such that $i \leq j \leq n$. Hence, this operation would take $O(n)$ time.

In the worst case, every operation might be of the form Query(1,v) and hence overall we might end up spending $O(Qn)$ time.

We will now present an idea which can be used to reduce the worst case running time significantly. We will assume that n which is the size of the array A is a perfect square. (The same solution can be modified to work in the case where n is not a perfect square as well).

Instead of working on the array A which is a one-dimensional array, we keep an array B which is two dimensional of size $\sqrt{n} \times \sqrt{n}$, where

$$B[i][j] = A[((i-1) \cdot \sqrt{n}) + j]$$

This array can be filled in $O(n)$ time.

We also maintain another array pre which is of size $\sqrt{n}$, where

$$pre[i] = \sum_{j=1}^{\sqrt{n}} B[i][j]$$

Let i be an index for the original array A. Set row = $\lfloor i/\sqrt{n} \rfloor + 1$. Further

1. If (i mod $\sqrt{n}$) = 0, then set col = $\sqrt{n}$
2. Else set col = (i mod $\sqrt{n}$)

The reader can verify that B[row][col] is indeed the same as A[i].

Let Update(i,v) be an update operation. By the previous method, we can find the row number (row) and column number (col) for the index i. Updating pre[row] by the value v, does the required operation.

Let Query(l,r) be a query operation. Let row1 and col1 be the corresponding row and column numbers of the index l. Let row2 and col2 be the corresponding row and column numbers of the index r.

1. If row1 = row2, then we can iterate from $B[row1][col1]$ to $B[row1][col2]$ and sum up all the entries and report the answer. This step takes $O(\sqrt{n})$ time.
2. If row1 $\neq$ row2, then, initialize a variable count to 0 and do the following:
   (a) count = count + $\sum_{i=col1}^{\sqrt{n}} B[row1][i]$.
   (b) count = count + $\sum_{i=row1+1}^{row2-1} pre[i]$.
   (c) count = count + $\sum_{i=1}^{col2} B[row2][i]$.

   After the computation, print the variable count.

   Clearly, all the three computation steps take $O(\sqrt{n})$ time.

Hence, we see that by the sqrt-decomposition method, we can perform the Update operation in constant time and the Query operation in $O(\sqrt{n})$ time.

Hence over Q queries, the overall worst case time complexity would be $O(Q\sqrt{n})$.

Exercise

1. Figure out how to modify the method in the case where n is not a perfect square.
2. Suppose along with the update and the query operations, we can also remove certain elements from the array. Modify the method to work in this case as well.

It also turns out that one can perform both the Update and the Query operations in $O(logn)$ time. This involves maintaining the elements in a special data structure called the Binary Indexed Tree. We postpone the discussion of Binary Indexed Trees to another day.

References:

1. http://www.iarcs.org.in/inoi/contests/dec2004/Advanced-1-solution.php