Chase The Red

Chase The Red: Algorithms and programming competitions

Order exercises // My problem in Functional Programming Contest - October'14

You can find the problem here: https://www.hackerrank.com/contests/lambda-calculi-oct14/challenges/order-exercises

Problem statement

You are given an array of n integers, $a = \{a_1, a_2, \ldots, a_n\}$ along with the following definitions:

- a **subarray** is a contiguous segment of an array. For example $a[l, r] = \{a_l, a_{l+1}, \ldots, a_r\}$ is a subarray, where $1 \leq l \leq r \leq n$

- We say that a **sum of a subarray** is a sum of elements in this subarray

- We say that subarray $X = \{a_i, a_{i+1}, \ldots, a_j\}$ is greater than subarray $Y = \{a_k, a_{k+1}, \ldots, a_l\}$ if and only if one of the following statements is true:
- X has a greater sum than Y
- X and Y has the same sum and X begins earlier
- X and Y has the same sum, they start in the same place and the length of X is smaller than the length of Y

It is guaranteed that there is no 0 in the array a.

You are given also an integer k. The task is to print as many as possible, but not more than k, subarrays with a **positive sum** in the following order.

- The first subarray is the greatest subarray in the array according to above definition.

- The i^{th} subarray is the greatest subarray disjoint to any of the j^{th} subarray, where $j < i$ (disjoint means that they have no common elements).

Solution

If you are familiar with the Maximum subarray problem you may notice that our problem is a natural extension of it.

Simple, but not enough solution

Probably the first solution which comes to mind is to run Kadane algorithm or similar to get the value of the greatest subarray, then assign $-\infty$ to every element of that subarray, find the second greatest subarray and so on. We iterate that process while there exists a subarray with a positive sum.

The time complexity of this method is $O(n \cdot k)$ which gives you some points, but it is too slow to pass all testcases even if it is hardly optimized.

Faster solution

The general pattern of above solution is good, but we have to find the next subarray faster. A segment tree can help us here.

The first thing here is to implement a segment tree which supports the following query in $O(\log n)$ time:

$query(i,j)$ returns the greatest subarray in $\{a_i, a_{i+1}, \ldots, a_j\}$

Let T_v be a subtree represented by a node v and a_v be an array corresponding to elements covered by T_v. We can implement that query maintaining in every node v several values:

- sum of the greatest subarray in a_v
- greatest prefix sum of a_v
- greatest suffix sum of a_v
- sum of values of elements in a_v

You can compute these values of any v based on values in its children. Values for leaves are obvious. After that, if you are familiar with segment trees, you can implement the $query(i,j)$ easily.

In order to do the next step we need a priority queue, let's call it Q. We store in the queue triplets $\langle R, V, P \rangle$ where:

- R is a range in which we search for subarrays
- V is the value of the greatest subarray in R
- P represents indices of the greatest subarray in R

Ordering of elements in Q is determined by values V and P.

First we compute the value and indices of the greatest subarray in the whole array. We do it using $query(1,n)$. Let's assume that it is $a[i,j]$ and has a sum V. We put into the queue a triplet $\langle (1,n), V, (i,j) \rangle$.

While Q is not empty and we have found less than k subarrays so far, we select the greatest triplet from the queue and remove it from there. Let's assume that the triplet is $\langle (k,l), V, (i,j) \rangle$. We print V as a result and if a range $(k, i-1)$ is not empty, we compute $query(k, i-1)$ and if it returns a subarray with a positive sum S

with indices (c,d), we put into the queue a triplet $\langle (k,i-1),S,(c,d)\rangle$. We do the same thing with a range $(j+1,l)$.

Time complexity

Time needed to construct the tree is $O(n\cdot\log n)$ and we do at most $O(k)$ queries, so the total complexity is $O(n\cdot\log n)$, because $k\leq n$.

This entry was posted in Hacker Rank, Problem setter, Programming Contests and tagged arrays, HackerRank, priority-queue, segment-tree on October 20, 2014 [http://chasethered.com/2014/10/order-exercises-my-problem-in-functional-programming-contest-october14/] .