Shaastra OPC 2010 - Solution Sketches

26th September, 2010

---

1 Fibonacci Sums)

Problem Setter: B.Srivatsan
Rating: Very Easy

The simplest problem of all. Compute and store the fibonacci numbers in an array. Carry out the summing and print the answer.
Complexity: $O(y - x + 1)$ **for each query**

---

2 Painful Prof

Problem Setter: B.Srivatsan
Rating: Easy

Find $region[i]$ = length of longest contiguous segment containing i where $a[i]$ is the smallest element.
This can be found either by using *disjoint set datastrucure* or by using *segment trees* or with the use of *STL set*
Sort the pairs $region[i], a[i]$ according to decreasing order of $region[i]$.
Store the prefix maximums of $a[i]$ in this sorted list.
To answer a query k, one can do a binary search to find the largest i in this sorted list, where $region[i] >= k$ and print the corresponding prefix maximum.
Complexity: $O(NlogN)$ **or** $O(N\alpha(N))$ **preprocessing and** $O(logN)$ **for each query**

---

3 K4 Counting

Problem Setters: B.Srivatsan and V.Prashant
Rating: Medium

A fact about planar graphs: There exists atleast one vertex which has atmost 5 neighbours.
For every such vertex u, we count the number of unordered triplets (x, y, z) such that x, y and z are neighbours of u and the edges (x, y), (y, z), (x, z) exist. We then delete u from the graph.
This can be implemented in $O(MlogN)$ time using sets to store adjacency lists and maintaining a queue for the vertices which have less than 6 neighbours.
Complexity: $O(MlogN)$

---

4 Giving Chocolates

Problem Setter: B.Srivatsan
Rating: Hard

At the final level, the solution is recursive backtracking method. We first establish a bijection which will help us to count faster.
In this array, we can write down the sequence of pairs $(number, position)$ where *number* is non-zero and is between A and B; in such a way that the sequence is sorted first according to *number* and ties are broken according to *position*.
(The dictionary order in positions is given by sorting first according to row index, and breaking ties by column index).
It is easy to see that there is a bijection between such sequences and the feasible solutions to the original question.

Let the ith element of the sequence be given by (n_i, p_i) and let there be m non-zero numbers, then we have

$$n_1 >= n_2 >= ... >= n_m \tag{1}$$

$$if\ p_i > p_{i+1},\ then\ n_i > n_{i+1} \tag{2}$$

Equation (2) follows from the fact that ties are broken according to position, incase of equal number. It tells us the dependance between n and p.

We wish to minimise this dependance to such an extent that we can count the sequences n and p separately and combine them to give the answer.

Some definitions:
i is said to be an "interesting" position if $p_i > p_{i+1}$.
$index(p)$ is defined as $\sum i$, where i is interesting.
$count(p)$ is defined as the number of interesting positions.

We define the "decr" operation as follows:
For every interesting position i, decrement each of n_1, n_2, ... , n_i.
Notice that, after this operation, the sequence n sums up to $N - index(p)$.
Also, every non-increasing sequence n with m parts that sums up to $N - index(p)$ can be made to obey (2) by doing the inverse of "decr" operation.
Further, note that the smallest number is unaltered by the "decr" operation and the largest number reduces by the $count(p)$.

Hence we count the sequences p stored according to the length m, $index(p)$ and $count(p)$ using a standard backtracking procedure.
We can also count the linear partitions of an integer of a given length where the smallest and the largest parts are specified (using a simple DP).

This bijection will then help us calculate the answer fast.
The central idea of this problem, the "decr" operation was taken from a paper by *D.E.Knuth* titled *"A note on solid partitions"*.

---

5 Longest Constrained Difference Subsequence

Problem Setter: V.Prashant
Rating: Medium

Define $f[x]$ = the length of the longest such sequence "seen so far" where the ending number is x.
Maintain a Range Maximum Query data structure on f.

Define $l[i]$ = length of the longest sequence ending in a_i.

As i goes from 1 to N, we have the following:

$$l[i] = 1 + MAX_{x=a_i-D}^{a_i+D} \ f[x] \tag{3}$$

$$f[a[i]] = MAX(f[a[i]], l[i]) \tag{4}$$

Note that (4) actually corresponds to an update in a Range Maximum Query datastrucure.

One issue was that the range of values of a_i was 32 bit signed integers. Hence the datastrucure(segment tree) had to be maintained dynamically, where we create new nodes as an when we require them. Its easy to see that this segment tree will have atmost 2*32*N nodes. **Complexity:** $O(NlogN)$

---

6 Best Managers

Problem Setter: V.Prashant
Rating: Medium

This is a typical tree-DP problem.
Root the tree at some node, say 1. Let
$a[i]$ = the maximum possible in the subtree rooted at i assuming that the pair $(i, parent[i])$ was already chosen. $b[i]$ = the maximum possible in the subtree rooted at i assuming that some pair of the form $(parent[i], x)$ was already chosen where $x \neq i$. $c[i]$ = the maximum possible in the subtree rooted at i assuming that both i and $parent[i]$ have not yet been chosen.

We can write the following recurrences:

$$a[i] = \sum_j b[j] \tag{5}$$

$$b[i] = \sum_j c[j] \tag{6}$$

$$c[i] = MAX(\sum_j c[i] \ , \ \ 2 + MAX_k \ (a[k] + \sum_{j \neq k} b[j])) \tag{7}$$

where j, k, range over the children of i.

The answer is $MAX(a[1], b[1], c[1])$. **Complexity:** $O(N)$

---

7 Cheapest Network

Problem Setter: B.Srivatsan
Rating: Medium-Hard

We ask the question: Assuming we use edges of length $\leq r$, is it possible to connect the computers? If we can answer this question, we can then do a binary search to find the final answer to required precision.

Given a maximum distance r, we look at all the edges between fixed computers which have a length $\leq r$ and form connected components using only

these edges. Now, we ask the question: Does there exist a point which is at a distance atmost r from some vertex in each connected component?
To answer this question, we draw radius r circles centred at every fixed computer. For every component, define "region" of the component as the union of the regions covered by the circles of the component. Now, we have to check if the intersetion of these "regions" is not null-set.
To do this, we consider only the intersetion points of two circles (of radius r) and check if there is an intersection point which lies in all regions.
Complexity: $O(N^3 logN)$
