Algorithms Notes

Bliss Of Comprehension

July 14, 2019

Contents

preface

This book is largely concerned with algorithms helpful for solving OJ problems.

Chapter 1

Stack

Stack is the suprising Data Structure that comes in handy in unexpected of times.

1.1 Balanced Parantheses

A bracket sequence is balanced if either of the below are true. As all of them are equivalent.

- Every element of prefix sum array is ≥ 0 and last element is 0.
- When open bracket is encountered push it to the stack and pop the top open bracket otherwise. If there is no open bracket to pop or the stack is not empty at the end, the sequence is not balanced.
- Base Case : Empty sequence is balanced.
  Constructor Case : If s,t (sequences) are balanced then s(t) is balanced.
  Now check if our sequence can be produced this way using recursion.

Overlapping two balanced sequences produces a balanced sequence. Prove it using the 1st definition.

C_i (catalan number) counts the number of sequences containing n pairs of parentheses which are correctly matched. Using this we can calculate in O(n) unlike other DP solutions.

Let s[0..n-1] be a balanced sequence

- s[0..i] is balanced iff s[i+1..n-1] is balanced.

When sequence is balanced, we can pairup open and closed brackets. This is done during the 2nd definition. When we are popping out the open bracket from stack(when closed br is encountered) we pair those two.

The pairing can also be done like this. For open bracket at i, find the minimum index j S.t sum[i..j]=0. (i,j) are paired

Let ith and jth index (i<j) are paired.

- s[i..k] $i < k < j$ is not balanced.
- If $i < l < j$ then the partner of l (named as r), $i < r < j$.
  Proof: We know $sum[i..l-1] > 0 => sum[l..j] < 0$ since $sum[l..l] = 0$ there exists $r < j$ S.t $sum[l..r] = 0$. We used the property that $sum[i..j]$ is continous over j.

Chapter 2

XOR

Competitive problem setters love XOR (never understood the physical significance of it).

Different techniques to solve problems on XOR

- Bit Trie.
- Solving for each individual bit and combining them at the end.
- Using Lexicographic property. If the most significant bit in binary form a number is 1 and the other number has 0 then first number is greater (no matter what the other bits are).
- In problems concerning XOR converting the array into prefix array can be useful.
- In tougher problems, divide and conquer is also used (i.e solve the problem for the numbers with 0 at considered bit, and solve for numbers with 1 at considered bit and combine them for the answer).
  Problem link

Some tricks in solving XOR problems

- $4k \oplus (4k+1) \oplus (4k+2) \oplus (4k+3) = 0$ (used in finding XOR of numbers from 1..1e18)
- If $a \oplus b = 0$ and $b \neq c \implies c \oplus b \neq 0$. This trick is mostly used in number theory to prove theorems on nim.
- If the array is sorted, all the elements with the same first x bits will be contigous $\forall$x.
- $a + b = a \oplus b + 2 * (a \ \& \ b)$

2.1 Game Theory

There are two type of operations involved in game theory

- MEX
- XOR

Sprague-Grundy Theorem :

If we know the MEX function of different games (G1, G2..) then the MEX function of combined games is XOR of all the functions.

Chapter 3

Counting

- Given a binary string, find number of 3-tuple (indices $i, j, k \ni i < j < k$) and exactly one of $s[i], s[j], s[k]$ is 1.

Chapter 4

Graphs

- In OJ's most of the problems concerning MST's can be solved with Kruskal's algorithm.(others are Prim's and Boruvka's)

4.1 Trees

While performing tree dp we can keep a map (of values in the subtree) at each node. This is a very generic trick and seems to have high potential.
we will traverse the tree from the down most level (level wise).

- Have maps at each of the lowest level nodes.(each map has only one entry)
- Now when we go up to produce map of node 'u' in this level.
  Take the largest map among children of u and add remaining children maps in to this big map. Now add the value corresponding to 'u' to the map. Now this map is the final map for 'u'.
- Since when each node moves from one map m1 to other m2, $sizeof(m2) \geq sizeof(m1)$. Final size of m2 after addition will be $\geq 2 * sizeof(m1)$. So any node will change maps only $lg(n)$ times. So the time complexity is $O(nlg(n))$.
- The memory is $O(n)$ throught out the procedure. Since we are inserting smaller maps in to the big map(which is already present).
- This can be implemented using dfs.

Chapter 5

C++ tricks

- In a normal set finding the ith element is not possible in o(lgn) and it is also not possible to know number of elements less than a given value.
- As the above operations are theoretically possible for a balanced BST, we use policy data structure to support those two additional operations along with normal set operations. Blog link
- But for using the above structure as a multiset, there is no easy way to do it. We will use a shortcut by inserting pair (value, time of insertion) to distinguish equal elements. And do operations around that.

Chapter 6

Arrays

- There are 2 types(tentative) of sqrt decomp I know.
  - MO's algorithm(offline algo) : When there is no query of update form then we can take all the queries and sort them in a certain way to make naive computation faster.
  - (online algo)Divide the array in to blocks each of size ($\sqrt{n}$) and process both kind of queries in O($\sqrt{n}$) each.