

# Algorithms Notes

Bliss Of Comprehension

July 17, 2019

# Contents

<b>1</b>	<b>Stack</b>	<b>3</b>
1.1	Balanced Parantheses . . . . .	3
<b>2</b>	<b>XOR</b>	<b>5</b>
2.1	Game Theory . . . . .	6
<b>3</b>	<b>Counting</b>	<b>7</b>
<b>4</b>	<b>Graphs</b>	<b>8</b>
4.1	Trees . . . . .	8
<b>5</b>	<b>C++ tricks</b>	<b>10</b>
<b>6</b>	<b>Arrays</b>	<b>11</b>
6.1	Permutation . . . . .	12
<b>7</b>	<b>Numbers</b>	<b>13</b>

# **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  $\geq 0$  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 pair up 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  $i$ th and  $j$ th 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  $sum[i..l-1] > 0 \Rightarrow sum[l..j] < 0$  since  $sum[l..l] = 0$  there exists  $r < j$  S.t  $sum[l..r] = 0$ . We used the property that  $sum[i..j]$  is continuous 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
- Gaussian elimination is used to find subset with maximum XOR.
- XOR of two numbers is just bitwise addition (mod 2).

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 contiguous  $\forall x$ .
- $a + b = a \oplus b + 2 * (a \& b)$

## 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.

- In a nim game with custom moves (examp. we can remove 1,2,5 stones at once) there will always be a periodicity of mexes with number of stones.

# 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).

There are certain kind of problems where number of edges are huge. They are not given in the input but told to us using certain properties of vertices. (Example. There are  $n$  nodes and each has a value  $a_i$ . There is an edge between  $i$  and  $j$  iff  $a[i] \oplus a[j] == k$ ).

- Sometimes these problems can be solved by adding some more nodes to the graph and connection the real vertices with new vertices. The property for the graph(ex.number of components) asked in the question may remain same for the new graph and now edges may be less.

### 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  $O(n)$  through out the procedure. Since we are inserting smaller maps in to the big map(which is already present).
- This can be implemented using dfs.

In DSU, while performing dsu one of the three types of techniques is used.

- Path Compression

```
Find(x)
  if x.parent != x
    x.parent := Find(x.parent)
  return x.parent
```

- Path Halving

```
Find(x)
  while x.parent != x
    x.parent := x.parent.parent
    x := x.parent
  return x
```

- Path Splitting.

In the problems where we need a pointer to an ancestor in a tree (or similarly in the array) and update operations makes you add more links then Path compression is useful. Example

Given a tree with each node having a value  $a_i$ . There are two kind of operations

Update : update  $a_i$  with  $c$  where  $c < a_i$ .

Query : Given  $u$ . Find the deepest node with value  $\geq k$  which is ancestor of  $u$ .

# Chapter 5

## C++ tricks

- In a normal set finding the *ith* element is not possible in  $O(\lg n)$  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 3 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.
  - (online algo) Consider  $\sqrt{q}$  operations at a time. Since there can be atmost  $O(\sqrt{q})$  update operations. The array will be changed at atmost  $\sqrt{q}$  indices hence if we find a way to answer the query operations (in this  $\sqrt{q}$  block) with some preprocessing each in  $O(\sqrt{q})$ .  
(Example. Find answer for this query without considering any updates in  $O(1 \text{ or } \lg n)$  and update this answer according to changed indices in this range(which are only  $\sqrt{q}$  in number). The time complexity of each block will be  $O(q)$  hence total complexity will be  $O(q\sqrt{q})$ .(PreProcessing is done for each block seperately just before considering the block). Example Problem Link.
- Let  $p$  be permutation and  $next_i$  be the smallest  $j > i$  s.t  $p[j] > p[i]$ . Then when we draw edges(curves) from  $i \rightarrow next_i$  on top of the array then no two edges intersect. (i.e the condition  $i < j < next_i < next_j$  is never satisfied for any pair of indices  $i, j$ ).
- You can find number of inversions in an array using policy data structure (balance BST). Infact we can do many variations of inversion problem. (Example. Find number of inversions when equal elements are present, Which is not possible using merge sort.)

## 6.1 Permutation

- If we have a directed graph of  $n$  nodes, and connect  $i$ th to  $j$ th iff  $p[i] = j$ ; then we get a bunch of disjoint cycles.

# Chapter 7

## Numbers

- $\lfloor a/(bc) \rfloor = \lfloor \lfloor a/b \rfloor / c \rfloor$