rekt_n00b's blog

Mo's Algorithm on Trees [Tutorial]

By **rekt_n00b**, history, 9 months ago,

Introduction

Mo's Algorithm has become pretty popular in the past few years and is now considered as a pretty standard technique in the world of Competitive Programming. This blog will describe a method to generalize Mo's algorithm to maintain information about paths between nodes in a tree.

Prerequisites

Mo's Algorithm — If you do not know this yet, read this amazing article before continuing with this blog.

Preorder Traversal or DFS Order of the Tree.

Problem 1 — Handling Subtree Queries

Consider the following problem. You will be given a rooted Tree T of N nodes where each node is associated with a value $A[node]$. You need to handle Q queries, each comprising one integer u. In each query you must report the number of distinct values in the subtree rooted at u. In other words, if you store all the values in the subtree rooted at u in a set, what would be the size of this set?

Constraints

$1 \le N, Q \le 10^5$

$1 \le A[node] \le 10^9$

Solution(s)

Seems pretty simple, doesn't it? One easy way to solve this is to flatten the tree into an array by doing a Preorder traversal and then implement Mo's Algorithm. Maintain a lookup table which maintains the frequency of each value in the current window. By maintaining this, the answer can be updated easily. The complexity of this algorithm would be $O(Q\sqrt{N})$

Note that you can also solve this in $O(N\log^2 N)$ by maintaining a set in each node and merging the smaller set into the larger ones.

Problem 2 — Handling Path Queries

Now let's modify Problem 1 a little. Instead of computing the number of distinct values in a subtree, compute the number of distinct values in the unique path from u to v. I recommend you to pause here and try solving the problem for a while. The constraints of this problem are the same as Problem 1.

The Issue

An important reason why Problem (1) worked beautifully was because the dfs-order traversal made it possible to represent any subtree as a contiguous range in an array. Thus the problem was reduced to "finding number of distinct values in a subarray $[L, R]$ of $A[]$. Note

→ Pay attention

Before contest
Codeforces Round #379 (Div. 2)
7 days

Like 46 people like this.

→ gbuenoandrade

Rating: 1900
Contribution: 0

- Settings
- Blog
- Teams
- Submissions
- Groups
- Talks
- Contests

gbuenoandrade

→ Top rated

#	User	Rating
1	tourist	3580
2	Petr	3481
3	TooDifficult	3211
4	matthew99	3100
5	dotorya	3094
6	eatmore	3030
7	Endagorion	3020
7	W4yneb0t	3020
9	PavelKunyavskiy	3007
10	Um_nik	2999

Countries | Cities | Organizations View all →

→ Top contributors

#	User	Contrib.
1	gKseni	183
2	Errichto	173
3	Petr	164
4	rng_58	159
5	Zlobober	155
6	Swistakk	154
7	Edvard	150
8	csacademy	148
9	zscoder	145
10	Xellos	144

View all →

→ Find user

Handle:

Find

that it is not possible to do so for path queries, as nodes which are $O(N)$ distance apart in the tree might be $O(1)$ distance apart in the flattened tree (represented by Array $A[]$). So doing a normal dfs-order would not work out.

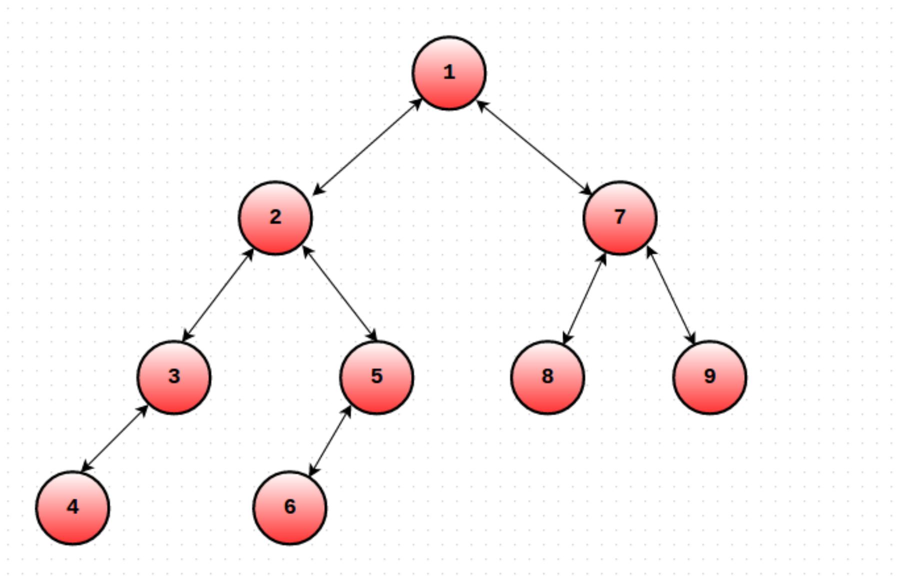
Observation(s)

Let a node u have k children. Let us number them as $v_1, v_2 \ldots v_k$. Let $S(u)$ denote the subtree rooted at u.

Let us assume that $dfs()$ will visit $u's$ children in the order $v_1, v_2 \ldots v_k$. Let x be any node in $S(v_i)$ and y be any node in $S(v_j)$ and let $i < j$. Notice that $dfs(y)$ will be called only after $dfs(x)$ has been completed and $S(x)$ has been explored. Thus, before we call $dfs(y)$, we would have entered and exited $S(x)$. We will exploit this seemingly obvious property of $dfs()$ to modify our existing algorithm and try to represent each query as a contiguous range in a flattened array.

Modified DFS-Order

Let us modify the dfs order as follows. For each node u, maintain the Start and End time of $S(u)$. Let's call them $ST(u)$ and $EN(u)$. The only change you need to make is that you must increment the global timekeeping variable even when you finish traversing some subtree ($EN(u) = ++cur$). In short, we will maintain 2 values for each node u. One will denote the time when you entered $S(u)$ and the other would denote the time when you exited $S(u)$. Consider the tree in the picture. Given below are the $ST()$ and $EN()$ values of the nodes.



$ST(1) = 1$ $EN(1) = 18$

$ST(2) = 2$ $EN(2) = 11$

$ST(3) = 3$ $EN(3) = 6$

$ST(4) = 4$ $EN(4) = 5$

$ST(5) = 7$ $EN(5) = 10$

$ST(6) = 8$ $EN(6) = 9$

$ST(7) = 12$ $EN(7) = 17$

$ST(8) = 13$ $EN(8) = 14$

$ST(9) = 15$ $EN(9) = 16$

$A[] = \{1, 2, 3, 4, 4, 3, 5, 6, 6, 5, 2, 7, 8, 8, 9, 9, 7, 1\}$

The Algorithm

→ Recent actions

gKseni → Codeforces announces contest for the best motto for laptop stickers

gninrael → TLE when using C++11 compiler

prime → TLE

lperovskaya → Three Subregionals Cup 2016

Xiejiadong → About 2016 Multi-University Training Contest 9(HDU)

r3gz3n → HackerEarth Collegiate Cup Final Round Results

fivt → Moscow International Workshop ACM ICPC 2016: Second wave of registration

now_you_see_me → would you please help me to find out the error in my code for lightoj 1220

EG0R → Codeforces Round #235 (Div. 2) — Tutorial

temp_test → Programming community

Salitanloo → What happened to acm.sgu.ru?

shashank21j → Sears Dots and Arrows Coding Contest[HackerRank]

Swift → Infinity Hats Problem

AnasAbbas → Why not always use long long

manishbisht → Google APAC 2017 — Round E

MikeMirzayanov → All-Siberian Olympiad 2016 (XVII Open Cup, GP of Siberia)

Petr → A Fourier combination week

minimario → Problems that are Dijkstra with a Twist

anand_20 → HackerEarth CodeMonk [Searching] Contest

abdelkarim → Egyptian Collegiate Programming Contest 2016

KaiZeR → Editorial Codeforces Round #247 (Div. 2)

theLifter → C++: What is the best way to read a line containing of string followed by numbers ?

barish2003 → Problem Chords (geometry)

zemen → Week of Code 25 on HackerRank

atatomir → Fast Fourier Transform and variations of it

Detailed →

Now that we're equipped with the necessary weapons, let's understand how to process the queries.

Let a query be (u, v). We will try to map each query to a range in the flattened array. Let $ST(u) \leq ST(v)$ where $ST(u)$ denotes visit time of node u in T. Let $P = LCA(u, v)$ denote the lowest common ancestor of nodes u and v. There are 2 possible cases:

$Case\ 1: P = u$

In this case, our query range would be $[ST(u), ST(v)]$. Why will this work?

Consider any node x that does not lie in the (u, v) path.
Notice that x occurs twice or zero times in our specified query range.
Therefore, the nodes which occur exactly once in this range are precisely those that are on the (u, v) path! (Try to convince yourself of why this is true : It's all because of $dfs()$ properties.)

This forms the crux of our algorithm. While implementing Mo's, our add/remove function needs to check the number of times a particular node appears in a range. If it occurs twice (or zero times), then we don't take it's value into account! This can be easily implemented while moving the left and right pointers.

$Case\ 2: P \neq u$

In this case, our query range would be $[EN(u), ST(v)] + [ST(P), ST(P)]$.

The same logic as Case 1 applies here as well. The only difference is that we need to consider the value of P i.e the LCA separately, as it would not be counted in the query range.

This same problem is available on SPOJ.

If you aren't sure about some elements of this algorithm, take a look at this neat code.

Conclusion

We have effectively managed to reduce problem (2) to number of distinct values in a subarray by doing some careful bookkeeping. Now we can solve the problem in $O(Q\sqrt{N})$. This modified DFS order works brilliantly to handle any type path queries and works well with Mo's algo. We can use a similar approach to solve many types of path query problems.

For example, consider the question of finding number of inversions in a (u, v) path in a Tree T, where each node has a value associated with it. This can now be solved in $O(Q\sqrt{N}\log N)$ by using the above technique and maintaining a BIT or Segment Tree.

This is my first blog and I apologize for any mistakes that I may have made. I would like to thank **sidhant** for helping me understand this technique.

Sample Problems

1) Count on a Tree II
2) Frank Sinatra — Problem F
3) Vasya and Little Bear

Thanks a lot for reading!

Original Post
Related Blog

+406 rekt_n00b 9 months ago 73

Comments (73)

Write comment?