Vladosiya's blog

Hashing root trees

By **Vladosiya**, history, 39 hours ago, translation,

I found a small shortage of materials in English on this topic and I want to start with translated article of **rationalex**:

The problem

We want to learn how to compare root trees for isomorphism (equality up to the renumbering of vertices + the root of one tree necessarily goes to the root of another tree).

Hash of vertex

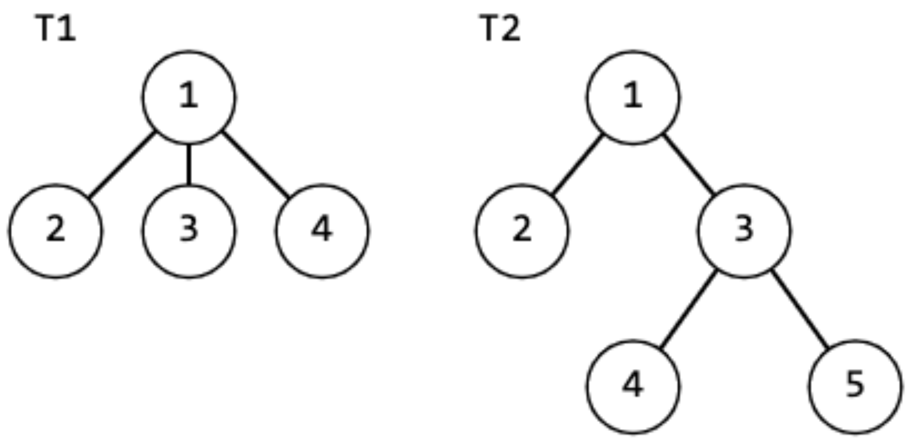
Note that since we cannot appeal to the vertex numbers, the only information we can operate on is the structure of our tree.

We will consider as a hash of a vertex without children some constant (for example, 179), and for a vertex with children we will use as a hash some function from the sorted list of hashes of children (since we do not know the true order in which the children should go, we need to bring them to the same form). The hash of the root tree will be considered the hash of the root.

By construction, the hashes of isomorphic root trees coincide (the author leaves the proof by induction on the number of levels in the tree to the reader as an exercise).

Polynomial hash is not suitable

Consider 2 trees:



If we calculate for them to take a polynomial hash as a function of children, we get:
$h(T1) = 179 + 179p + 179p^2 = 179 + p(179 + 179p) = h(T2)$

Which hash is suitable?

As a good hash function, for example, this is suitable

$$h(v) = 42 + \sum_{u \in sorted\_by\_hash(child(v))} log(h(u))$$

For this hash function, it may seem that it is possible not to sort the hashes of children, but this is not the case, because when calculating floating-point numbers, we have an error, and in order for this summation result to be the same for isomorphic trees, it is also necessary to sum the children in the same order.

An example of a more complicated hash function:

$$h(v) = \left[\sum_{u \in sorted\_by\_hash(child(v))} h(u)^2 + h(u)p^i + 42\right] \mod 2^{64}$$

Asymptotics

All we need to do at each level is sorting the vertices by hash value and summing, so the final complexity is: $O(|V|log(|V|))$

I want to continue on my own:

In the reality of Codeforces, these approaches have problems in the form of hacks (which can be seen, for example, by hacks of this task). Therefore, I want to talk about an approach in which there are no collisions.

What is this magic hash function?

Let's sort the hashes of the children for the vertex and match the number to this array, which we will consider the hash of the vertex (if the array is new, then we will assign it the minimum unoccupied number, otherwise we will take the one that has already been given).

Why does it work fast?

It is easy to notice that the total size of the arrays that we counted is $n - 1$ (each addition is a transition along the edge). Due to this, even using treemap for mapping, all accesses to it will require a total of $O(n \cdot log(n))$. Comparing a key of size sz with another key works in $O(sz)$ and such comparisons for each key will occur $O(\log(n))$ times, and the sum of all sz, as we remember, is $n - 1$, so it turns out a total of $O(n \cdot \log(n))$. (You might think that it is worth using hashmap, but this does not improve the asymptotics and causes the probability of a collision).

tree, hashing

+91 Vladosiya 39 hours ago 13

Comments (13)

Write comment?

Golovanov399

28 hours ago, # | ← Rev. 3 +25

First of all, what do you call a "polynomial hash"? Is it something like sum of "take i-th child, multiply its hash by p^i"? If it is true, then this "hash" changes when we swap two children with different hashes, which shouldn't happen.

UPD: okay, I wasn't very observant and didn't read the whole paragraph with "hash of subtree is hash of sorted list of smth"

Second, I think that the last way you mentioned is the most popular and simple, but just out of theoretical interest one can read this ancient article by **rng_58**. It contains two pictures which seem to have expired, but I have the second of them:

the relevant pic

→ Reply

Formatci

28 hours ago, # | -110
→ Reply

The comment is hidden because of too negative feedback, click here to view it

27 hours ago, # | +31

There's a way to hash rooted trees which has a better time complexity and is also easy to implement, whose expected number of collisions doesn't exceed $O(\frac{n^2}{2^w})$.

It is mentioned here.

To avoid being hacked, you can use random parameter instead of 1237123 in the article when solving Codeforces problems.

chenshi028

→ Reply

25 hours ago, # | ← Rev. 2 +20

lemelisk

Polynomial hash works fine, you just need to represent tree's euler tour as a correct bracket sequence — `(` when you walk along an edge from parent to child and `)` when you go backward from child to parent. So T1 will be `()()()` and T2 will be `()(()())` . Then hash it as usual string (of course, with sorting childs' hashes).

→ Reply

Vladosiya

24 hours ago, # ^ | ← Rev. 2 0

Doesn't it look like it's easier to create a collision with such hashing? (In context of hacks)

→ Reply

lemelisk

23 hours ago, # ^ | 0

Every tree is just a string in such representation. If we use polynomial hashing for strings, why shouldn't we use it in this case?

→ Reply

Facelessman2.0

20 hours ago, # ^ | 0

I solved https://codeforces.com/contest/1800/problem/G using your idea. It's nice

→ Reply

sahaun

23 hours ago, # | ← Rev. 3 +3

Can anybody suggest some problems on the topic?

UPD:

1800G - Symmetree

763D - Timofey and a flat tree

Kattis — Two Charts Become One

→ Reply

Noble_Mushtak

11 hours ago, # ^ | ☆ +4

I recently solved a problem from an ICPC regionals using this technique: Two Charts Become One

→ Reply

MikeMirzayanov

19 hours ago, # | ☆ +57

It seems often a good idea to compute not hashes for subtrees, but simply numeric representations $val(u)$ — consecutive integers from 0 and beyond, where u is the root, which defines a subtree rooted at u. The values $val(u)$ and $val(v)$ will be equal if and only if the subtrees with roots u and v are equal (or isomorphic).

You can just use `map<vector<int>, int> vals` for memoization. Then if for u you need to calculate the value of $val(u)$, then take all children $w_1, w_2, \ldots, w_k$ of u and make a vector $[val(w_1), val(w_2), \ldots, val(w_k)]$. If it has a value in $vals$, then take the value from there, if not, enter the next integer (well, just use current $vals.size()$).

And just recursively calculate this for all subtrees. In total, it will work for $O(nlogn)$, apparently. And no issues with possible hash collisions. You can think of the resulting values as perfect hashes.

→ Reply

Uzumaki_TheLoser

7 hours ago, # ^ | ☆ 0

don't we need to sort the vector ([val(w1),val(w2),…,val(wk)]) ?

→ Reply

why_I

7 hours ago, # ^ | ☆ 0

Yeah i think we need to sort them.

→ Reply

why_I

7 hours ago, # ^ | ☆ 0

Thank you! i literally do not know about hashing at all. i just saw your comment and solved the problem 1800G - Symmetree .

My submission if anyone want :- 195934874

→ Reply