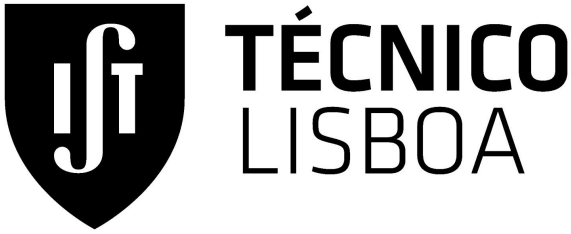


MEIC-A - AAva 2016/2017
João Franco Nº 75219
Fábio Almeida Nº 76959

1. Link cut trees - introduction

Link cut trees are data structures used to represent a forest of rooted, unordered trees. This structure should support the operations:

- **link(v, w)** - Adds an edge linking node v to node w. If such an edge already exists or this insertion would create a cycle then the operation has no effect.
- **cut(v, w)** - Removes the edge linking the node v to the node w, if such an edge exists. If the edge does not exist this operation has no effect.
- **connected(v, w)** - Returns true if there is a connection from v to w. If such a connection does not exist it returns false. A connection may consist of a single edge, or a sequence of edges, provided that it links v to w.

2. Implementation details

In this section we will describe the implementation details of the aforementioned operations, but before we will detail some auxiliary functions used.
In order to use the link operation we have to ensure one of the nodes is the root of its represented tree. Therefore we had the need to implement the **re_root(Node *n)** operation.

2.1 re-root

This operation works simply by accessing the node (making it the root of it's splay tree and having no right child) and then swapping the left child with the right child and propagating those changes below. In order to make it fast, instead of actually flipping everything in memory, we just set a bit indicating that the tree is now flipped. This will later be used by splay to make sure everything is consistent.

2.2 Link

Having re-root defined we can now implement link.
To do this we **re-root(v)**, making **v** the root of the represented tree and also the root of the splay tree it is currently in, followed by an **access(w)**, making **w** the root of it's splay tree, and the root of its the tree of trees.
If connecting **v** and **w** would not cause a loop we could just link them by making **v** a right child of **w**.
In order to determine if a loop exists, we simply check if **v** stopped being the root of the tree of trees after we accessed **w**. If that is the case it means **v** and **w** are already connected and so we do nothing.

2.3 Connected

To implement **connected(v, w)** we do the same as in link, the only difference being that we don't connect them in the case they are connected (obviously) . Otherwise, everything is the same.

2.4 Cut

To implement the cut operation, we could have used re_root, but I decided against it seeing it wasn't necessary.
We first start by accessing **v**, making it the root of the tree of trees. Now we want to find if **w** is connected to **v**. This can happen in two situations:

1. **w** is the **parent of v** in the represented tree. To check this we follow **v's** left child to the right, and compare it to **w**. If they match we can cut it, by removing the edge between **v** and its left child..
2. **w** is a **child of v** in the represented tree, in which case it has to point to **v** via a parent pointer. Removing that parent pointer does the cut.

3. Practical analysis

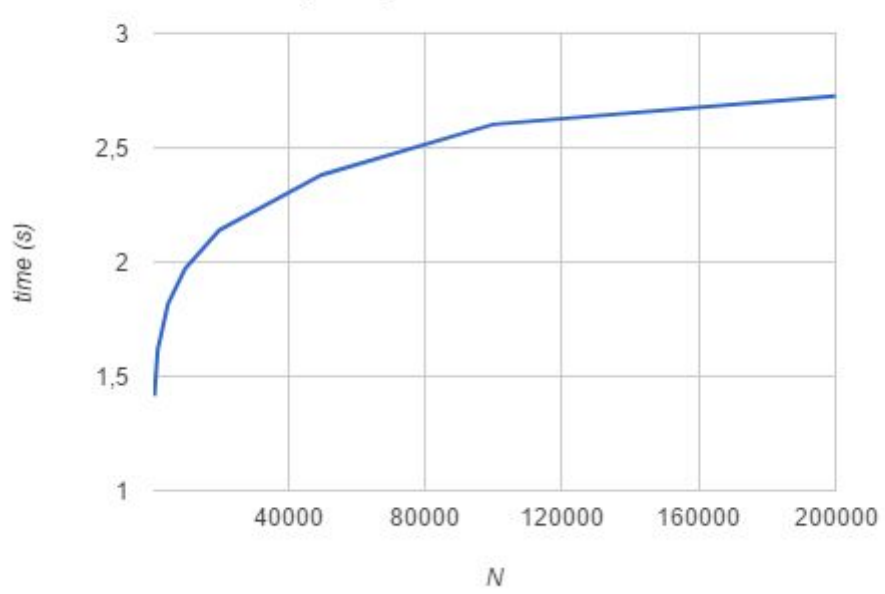
3.1 Method

The tests were generated so that at the start **all the nodes were linked in order**, and after that all input sequences were of the form **<cut, connected, connected, link>** repeated until the end.

The time values in the graph below were obtained by **averaging out 5 different tests** with new randomly generated inputs every time. We observed that even though the input files were different, the time difference was never larger than 0.15s.
The **number of operations** was **fixed to 1M** in all tests. The only **variable** changing was the number of nodes inserted **(N)**, starting from 1000 nodes all the way to 200k nodes.

3.2 Experimental results

The graph below represents the tests mentioned above:



This was the table of results used to generate this graph:

N	time
1000	1,414
2000	1,618
5000	1,818
10000	1,97
20000	2,14
50000	2,382
100000	2,602
200000	2,726

3.3 Theoretical vs experimental comparison

Amortized analysis tells us that the operations cut, link and connected are time bounded by O(logN). The experimental results agree with that.
Regarding size, it is obviously O(N), since we need to allocate 1 and only 1 more node when N increases by 1.

4. Conclusion

In conclusion we were able to implement a link cut tree structure that is efficient and could be used to solve real problems with little changes to fit the specifications of each specific problem.

5. References

[1] Sleator, D.D., and R.E. Tarjan, ‘Self-adjusting binary trees,” Proc. 15th Ann. ACM Symp. Theory Comput. STOC-83, pp. 235–245; also in final form in J. ACM 32 (1985), pp. 652–686.
[2] MIT 6.851 Advanced Data Structures, Spring 2012 - Dynamic Graphs I (https://www.youtube.com/watch?v=XZLN6NxEQWo)