

MSc in Computer Science and Engineering

**Advanced Algorithms 2016-2017**

**Connectivity in Forests:**

In this project students had to implement an API to maintain the connectivity information of and underlying forest. Maintaining the connectivity of information is a well know problem for computing nearest common ancestor, solving various network flow problems including finding maximum flows, blocking flows, and acyclic flows, computing certain kinds of constrained minimum spanning trees and implementing the network simplex algorithm for minimum-cost flows.

1. **INTRODUCTION:**

The goal of this project is to maintain the connectivity information of an underlying forest. To store this information, we are going to use the Link/Cut tree data structure (with some minor changes that we will explain in section 2). The motivation behind this is that if we represent the forest as a regular collection of parent pointer trees, it might take us a long time to find the root of a given node, however if we represent each tree in the forest as a Link/Cut tree we can do all operations in amortized time where n is the number of nodes in the structure.

The forest in this project consists of a set of vertexes linked by edges, such that no set of edge forms a cycle.

The API developed to resolve this project consists in 3 operations: *link*, *cut* and *connected* which we will explain better in sections 3.

1. **BACKGROUND**:

In order to better understand the operations of the project API it’s required to understand 2 functions: splay and access.

This functions are responsible for the logarithmic time bound on the desired operations and they keep our data structure updated.

1. Splay(u):

This function is similar to a move-to-root where it does rotations bottom-up along the accessed path and moves the node u until it is the root of the tree. These rotations are done in pairs, according to the structure of the accessed path, with the auxiliary function called splaying step.

The splaying step auxiliary function receives a node and evaluates 3 main cases:

Case 1 (zig):

if the parent of u is the tree root, rotates the edge joining u with his parent (terminal case:

Case 2 (zig-zig):

if the parent of u is not the root and u and his parent are both left or both right childs. Rotate the edge joining the parent of u with its grandparent, and then rotate the edge joining u with his parent.

Case 3 (zig-zig):

if the parent of u is not the root and u is a left child and his parent is a right child, or vice-versa, rotate the edge joining u with his parent and then rotate the edge joining u with his new parent.

1. Access(u):

The access operation changes the represented tree in order to make the path from the root to node u the preferred path. To do this we start by splaying u, which brings u to the root of the auxiliary tree. We then disconnect the right sub-tree. The root of the disconnected tree will have a hook (of type path-parent pointer) to u.

**DATA STRUCTURE AND API:**

* 1. **Link/Cut Tree Structure:**

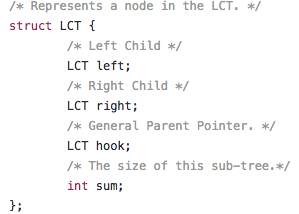


Figure 1 - Link/Cut tree node C structure

The Link/Cut tree underlying structure is shown in figure 1.

The left and right pointers are used to represent the pointers in the auxiliary splay tree. The hook contains the parent of that node. If the parent points back to the node either by a right pointer or a left pointer than it represents a parent otherwise the hook is representing a path pointer and the node pointed by the hook is a path parent. The sum field should represent the depth of the node but in the context of this project it is only used as a bit. If the sum field is -1 than the tree is conceptually reversed, meaning that the order of the nodes in the sub-tree is from right to left instead of left to right.

* 1. **ConnectedQ(v,w) Operation:**

The ConnectedQ(v,w) operation asserts True or False if two nodes v and w are connected through some path or not, respectively. We Access(node) both v and w, make v a root and then see if we can reach v from w by jumping through w's parents.

* 1. **Link(v,w) Operation:**

The Link(v,w) operation combines 2 trees into 1 by adding an edge between nodes v and w. For this operation to be performed we have to assure that there was no edge already linking v and w, and also that they are not connected through some other path, because this would create a cycle. Inside this operation we perform an Access(node) on both v and w to make sure upcoming queries are optimized, then make w the left child of v and v the parent of w.

* 1. **Cut Operation:**

The Cut(v,w) operation divides 1 tree into 2 by deleting an edge. For this operation to be sucessful we must first guarantee that there is a link between v and w, otherwise the operation does nothing.

1. **EXPERIMENTAL ANALYSIS:**
2. **DISCUSSION:**

Blab la bla

1. **REFERENCES:**

# Algorithms on Strings, Trees, and Sequences: Dan Gusfield 1997 Cambridge University Press

* Introduction to Algorithms: Thomas H. Cormen, Charles E. Leiserson, Rolnald L.Rivest and Clifford Stain 2009 MIT Press