

*Ajay Ladsaria*

*CS473 Notes*

*9/11/2001*

## 1 Introduction

I used Lyx to typeset this, and XFig to draw the figures. This is my first time using both of these tools so I haven't done a very good job with formatting. But more importantly, I realized that I don't really understand the algorithm for *Replace()*. So these notes might not make sense until I ask to Jeff to clarify them for me on Tuesday, and then I can update these notes.

Jeff started by talking about things he didn't have time to cover during the semester.

Bentley-Saxe\* - general idea of turning a data structure that supports only delete into one that supports insert.

Dynamic Convex Hulls - Overmars\*-VanLeeuwen '83  $O(\log^2(n))$

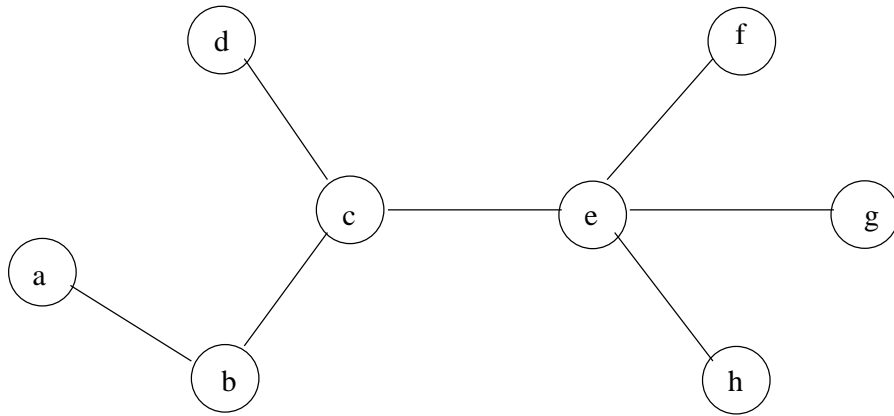
Chan'98 (graduated '96)  $O(\log^{1+\epsilon}(n))$

Brodal and \*'s '99 (graduated '95)  $O(\log(n) \log(\log(n)))$

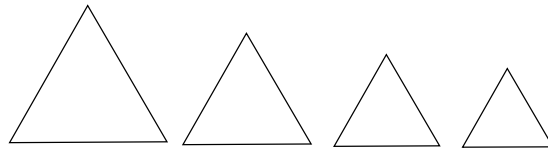
The above stuff is not going to be covered due to lack of time.

## 2 Dynamic Graph Algorithms

The problem is to keep the connectivity information of a graph in a data structure to be able to quickly determine if two nodes are connected by a chain of edges. The basic idea is to maintain a spanning forest of the nodes in the graph using a data structure that will show up again and again, namely Euler-Tour Tree (ET-Tree) Henzinger-King'96.



Every node in the ET-Tree corresponds to an edge. Every edge in the Euler-Tour of the graph is included in the ET-Tree as nodes. The Euler-Tour of the above graph is a-b-c-e-h-e-s-e-f-e-c-d-c-b-a. Thus each is included twice in the tree; for example, edge ce and ec are in the tree. Every vertex in the graph points to an ET-Tree node that represents an edge beginning at that vertex.



Spanning forest of ET-Trees

In the figure above, there is an ET-Tree for the edges in each connected portion of the graph. Connection queries can be quickly answered by looking to see whether the two nodes pointed to by the two vertices in question are in the same ET-Tree. This is done by walking up to the root from each node and seeing if the roots are the same. This of course takes  $O(\log(n))$  time assuming the trees are balanced.

Deletion can be handled with two splits and one concatenation of the ET-Tree. For example, let's observe what happens when edge ce is deleted. Both ce and ec are deleted from the ET-Tree, leaving us with three ET-Trees. Namely, a-b-c, e-h-e-s-e-f-e, and c-d-c-b-a. We now concatenate the first and third trees so we have a-b-c-d-c-b-a and e-h-e-s-e-f-e.

If an edge e connects two nodes in the graph that are already connected by another sequence of edges then when edge e is inserted, it does not affect the ET-Trees. Otherwise two ET-Trees are concatenated.

Now to get down to the more complicated analysis portion.

### 3 Analysis

Let each edge  $e$  have a level  $l(e)$  between 0 and  $\lceil \lg(n) \rceil = L$

Let  $G$  be the graph.  $G_i$  contains the edges of  $G$  with  $l(e) \geq i$

Thus  $G_0$  is the entire graph. Let  $F_i = F \cap G_i$

Thus,  $G = G_0 \supseteq G_1 \supseteq \dots \supseteq G_L$  and  $F = F_0 \supseteq F_1 \supseteq \dots \supseteq F_L$

Two invariants:

1.  $F$  is a maximum spanning forest of  $G$  with respect to level.
2. Each component of  $F_i$  has  $\leq \frac{n}{2^i}$  nodes.

#### 3.1 Initialize

For all edges,  $l(e) = 0$ , and  $F_0 \leftarrow$  some spanning forest of  $G_0$

#### 3.2 Insert( $u, v$ )

$l(u, v) = 0$ , if  $Connect?(u, v) = \text{False}$ , then add  $(u, v)$  to  $F_0$

Running time is  $O(\log^2(n))$  amortized time because we are paying for later promotions.

#### 3.3 Delete( $u, v$ )

Without loss of generality assume that  $(u, v) \in F$

$Replace(u, v, l(u, v))$

#### 3.4 Replace( $u, v, l(u, v)$ )

The algorithm runs in  $O(\log^2(n))$  amortized time.

Without loss of generality assume that  $l(u, v) \geq 0$

$T_u \leftarrow$  component of  $F_i$  containing  $u$ , and  $T_v \leftarrow$  component of  $F_i$  containing  $v$ .  
 $O(\log(n))$

Without loss of generality assume that  $|T_u| \leq |T_v|$ .  $O(1)$

For each edge  $e$  in  $T_u$  with  $l(e) = i$ ,  $l(e) = i + 1$ . Promoted edges need to be inserted into higher level, this cost is  $O(\log(n))$ .

For each edge  $e$  adjacent to  $T_u$  with  $l(e) = i$ . Number of iterations is  $O(\log(n))$

- If  $e$  joins  $T_u$  and  $T_v$ , done
- $l(e) = i + 1$

*Replace*( $u, v, i - 1$ )

### 3.5 ET-Tree Nodes

Each node stores the following three things:

1. #edges in this segment of the tour
2. 0/1 Any edges at level  $i$  ?
3. 0/1 Any non-tree edge at level  $i$  adjacent to segment?

Any vertex of  $G$  can return an edge with  $l(e) = i$  in  $O(\log(n))$  time. Connectivity query is  $O(\log(n))$ .