Advanced Data Structures

Lecture 9

Lecture by Dr. Shay Mozes
Typeset by Steven Karas

2017-01-05
Last edited 22:41:27 2017-01-05

1 Agenda

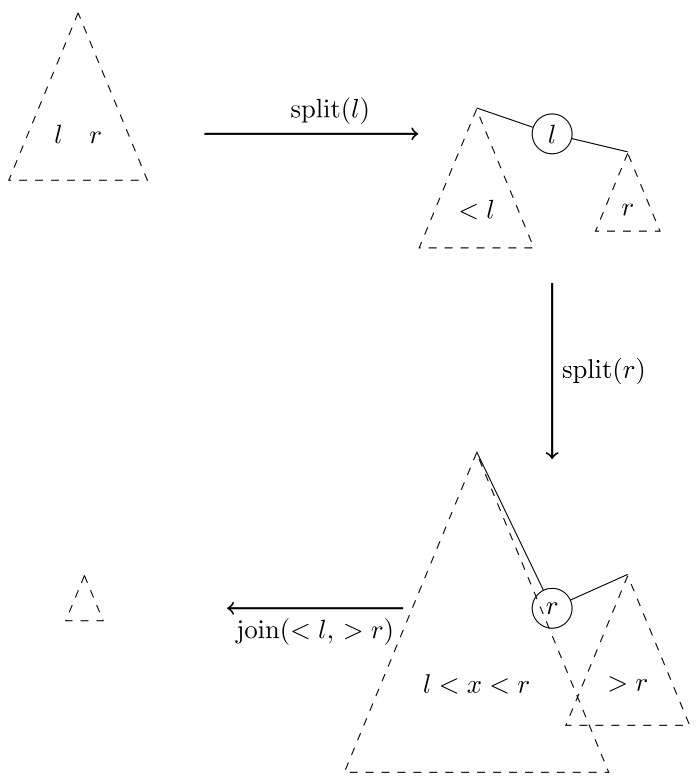
Tango trees wrapup. Link-cut trees.

2 Tango Trees

Built to conform to the lower bound on optimality (based on the number of changes to favored children in a static optimal tree). Tango trees are $O(\lg\lg n)$-competitive with the dynamic optimum.

As a reminder, tango trees are forests of trees of the favored paths.

2.1 Changing the tree



3 Link-cut Trees

Discovered by Sleator and Tarjan in 1983. Has applications for planar graphs. Represents trees/forests for some operations:

3.1 Topological Operations

MakeTree(v) Create a tree with a single node v.

FindRoot(v) Returns the root of the tree that contains v.

Cut(v) Assume that v is not a root. Deletes the edge from v to its parent.

Link(v, w) Given two nodes v, w. Assume that v is the root of its tree, and v, w are not in the same tree. Attach the tree of v as a child of w.

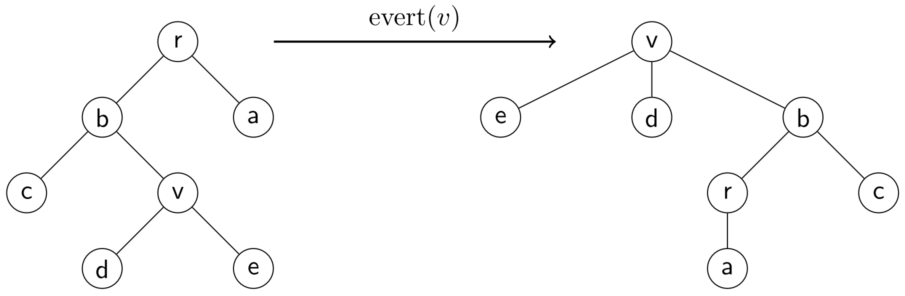
3.2

Weights can be assigned to either nodes or edges. Our definitions are for when the weights are defined on nodes.

FindMinCost(v) Find the first node with minimal cost on the path from v to the root.

AddCost(v, x) Add x to the weight of each node on the path from v to the root.

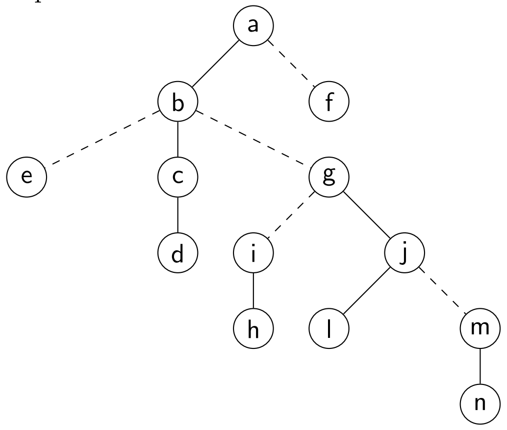
Evert(v) Make v the root of the



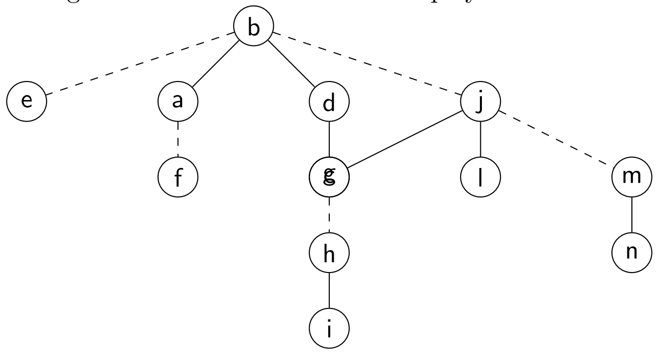
Complexity Amortized $O(\log n)$ time. Useful for planar graphs.

3.3 Implementation

We represent each of the trees in the forest as a forest of splay trees.



This gives us the concrete forest of splay trees:

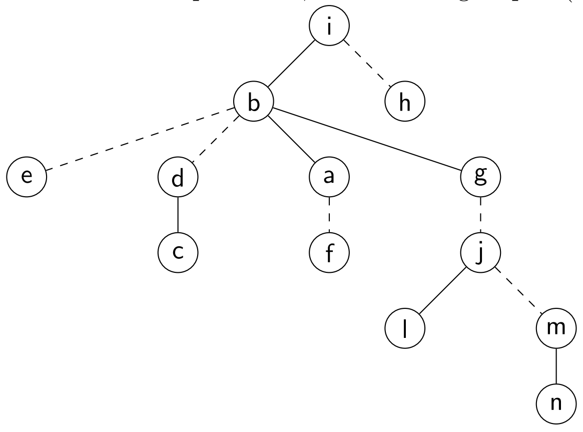


The nodes of the concrete tree are the nodes of the represented tree. Each node in the concrete tree keeps a pointer to its parent in the splay tree. The root of each splay tree keeps a pointer to the node from which the path the splay tree represents came from (the parent of the path in the represented tree). Each node tracks its leftmost and rightmost child in the splay tree.

Expose(v) This is the difficult primitive that will help us implement all the other operations efficiently. This operation makes the path from v to the root a favored path and marking all favored edges that are incident to the path to unfavored.

```
Expose(v):
  Splay(v)
  v.right = nil
  while v.parent != nil:
    w = v.parent
    Splay(w)
    w.right = v
    rotateUp(v)
```

So for our example above, after running Expose(i):



After running Expose(v), v is the root of the splay tree that represents the favored path from v to the root of the represented tree. In other words, v is the root of the splay forest.

Another property after is that v will have no right child.

```
FindRoot(v):
  Expose(v)
  r = min element in splay tree of v
  Splay(r)
  return r
```

```
Cut(v):
  Expose(v)
```

```
    w = v.left
    w.parent = nil
    v.left = nil
```

```
Link(v, w):
    Expose(v)
    Expose(w)
    w.right = v
    v.parent = w
```

3.4 Analysis

We want to show that this implementation is $O(\log n)$ amortized. Notably, we'll only prove this for Expose, since we take a constant number per other operation, and Splay has already been proven to be $O(\log n)$ amortized.

Expose $\underbrace{O(\log n)}_{\text{Splay}} \cdot \underbrace{\text{\# changes of favored children}}_{\text{\# iterations of the loop}}$

Lemma The number of changes of favored children in a sequence of m operations is $O(n + m \log n)$

Proof We will use a "heavy-light decomposition"[1]. Let $size(v)$ be the number of nodes in the subtree of v in the represented tree. The edge $(v, v.parent)$ is "heavy" if $size(v) > \frac{1}{2} size(v.parent)$. For each node, there is at most one heavy child. For any path from the root to any node, there is at most $\log n$ "light" edges. This is because each time we go down a "light" branch, the subtree is at least half as small. Each edge may be either favored/unfavored, and light/heavy. Each time a favored branch becomes unfavored, an unfavored branch must become favored. It's enough to count the amount of branches that become favored. In fact, we'll actually count: L - the light branches that become favored; H - the heavy branches that become favored.

Trivially, we can bound the light branches by $|L| \leq m \cdot \log n$.

In a tree, it holds that there are at most n favored branches.

If a branch e becomes favored more than i times as part of the sequence of operations, it must have become unfavored at least $i - 1$ times.[2]

Therefore, we can say that $|H| \leq n+$heavy branches that become unfavored.

Each time a heavy branch becomes unfavored, a light branch becomes favored. This happens at most $\log n$ times in each Expose. This gives us $|H| \leq n + m \log n$.

Therefore, the number of changes is $|L| + |H| = O(n + m \log n)$.

[1]This may have been the first use of this decomposition, but it is widely used for data structure analysis

[2]Alternative explanation: From within the heavy branches, all branches except maybe for n branches become unfavored before the end of the operation sequence.

We have shown a bound of $O((n + m \log n) \log n)$ on a sequence of m operations.

In a sequence of m $O(m \log^2 n) = O((n + (m - n) \log n) \log n)$.