Advanced Data Structures

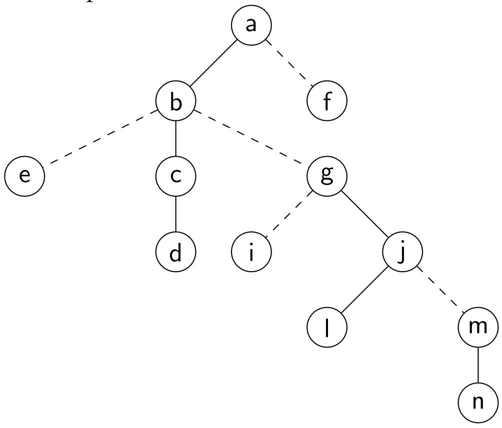
Lecture 10

Lecture by Dr. Shay Mozes
Typeset by Steven Karas

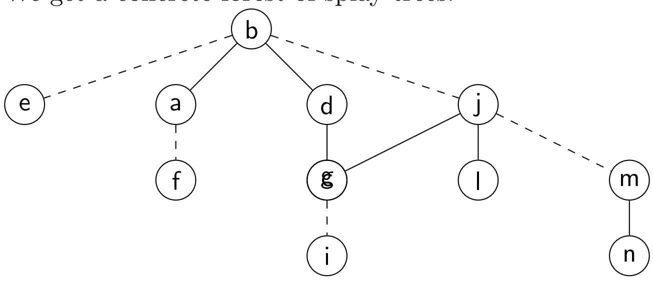
2017-01-12
Last edited 21:01:55 2017-01-12

1 Review of Dynamic trees

For this represented tree:



We get a concrete forest of splay trees:



1.1 Topological operations

1.1.1 MakeTree(v)

Create a tree with a single node v.

1.1.2 FindRoot(v)

Returns the root of the tree that contains v.

1.1.3 Cut(v)

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

1.1.4 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.

1.1.5 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. It has the side effect of making v the root of the concrete tree.

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

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

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

A full proof including a heavy-light decomposition was in the previous lecture.

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

In a sequence of m operations, we get a bound of: $O(n \log n + m \log^2 n)$ that gives us an amortized cost of $O(\log^2 n)$.

Reminder: Splay tree complexity Each node v has a weight $w(v)$. Let $S(x) = \sum_{u \text{ descendent of } v} w(u)$. Let $\phi_{splay}(T) = \sum_{v \in T} \log(S(v)$. $amort(splay\, v) \leq 3(\log S(root) - \log S(v))$

Amortized complexity using potential function Define $S'(v) =$ the number of nodes in the subtree of v in the concrete tree.

$$\phi_{LC} = \sum_{v \in T_c} \log S'(v)$$

Changing the favored child does not change the potential function. Therefore the contribution of changing the favored child to the amortized cost of Expose is the number of changes: $O(n + m \log n)$.

Changes of Splay to the potential function Let $w'(v) = 1 + \sum_{u \text{ unfavored children of } v} S'(u)$.

For example, in our example above, $S'(j) = 5$, yet $w'(v) = 2$.

Therefore: $S(v) = \sum w'(u) = S'(v)$

The contribution of each splay operation to the amortized cost is $\leq 3(\log S(root_{splay}) - \log S(v))$. Because the node v_{i+1} that is splayed in the $i+1$ step is the parent of the $root_i$ in the i^{th} iteration $S(v_{i+1}) \geq S(root_i)$, the series is telescopic.

Therefore the contribution of each splay of an Expose(v) is at most $\log S'(root) - \log S'(v) = O(\log n)$.

Thus the total cost of a sequence of m operations is:

$$O(n + m \log n) + O(m \cdot \log n) = O(n + m \log n)$$

1.2 Decorative Operations

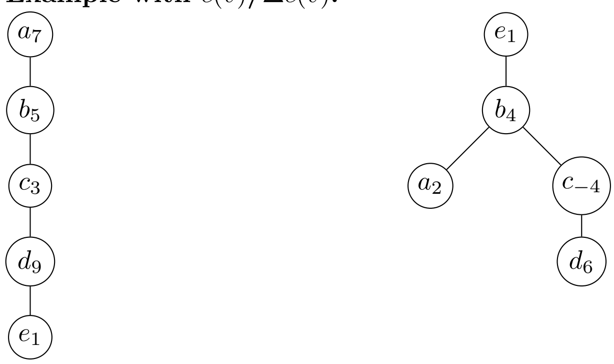
Instead of storing the cost of each node from the represented tree, we store the Δ:

$$\Delta c(v) = \begin{cases} c(v) & v \text{ is a root} \\ c(v) - c(parent(v)) & \text{else} \end{cases}$$

Which gives us that $c(u) = \Delta c(u)$ if u is a root, and $c(u) = \Delta c(u) + c(v)$ if u is a child of v.

This guarantees us that for any node u, $c(u) = \sum_{v \text{ ancestor of } u} \Delta c(v)$. [1]

Example with $c(v)/\Delta c(v)$:



1.2.1 AddCost(v, x)

Add x to the weight of each node on the path from v to the root.

```
AddCost(v, x):
  Expose(v)
  v.Δc += x
```

[1]We will prove in the homework that the rotation updates the costs correctly

1.2.2 FindMinCost(v)

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

Define the following: Let $\min c(v) =$ the minimal $c(v)$ in the subtree of the concrete splay tree of v.

$$\Delta c(v) = c(v) - \min c(v)$$

$$\Delta \min c(v) = \begin{cases} \min c(v) & v \text{ is a root} \\ \min c(v) - \min c(parent(v)) & \text{else} \end{cases}$$

$$\min c(u) = \sum_{v \text{ ancestor of } u} \Delta \min c(v)$$

$$c(u) = \Delta c(u) + \min c(u)$$

```
FindMinCost(v):
  Expose(v)
  Repeat:
    if v.left.Δminc=0:
      v = v.left
    else if Δc(v) > 0:
      v = v.right
    else:
      break
  Splay(v)
  return v
```

1.2.3 Evert

[2]

History In the original presentation of Link-Cut trees, splay trees were not yet discovered (or rather, were being discovered in parallel). As a result, the original complexity was much worse, and only after they combined them the complexity went down.

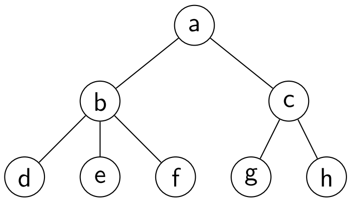
2 Euler tour trees

These trees are used to solve the problem of dynamic connectivity in graphs. For simplicity, we will assume that our link-cut trees are rooted, and the root does not change.

[2]There was a hint that this may appear in the next homework

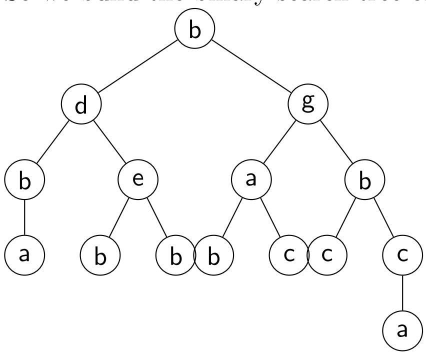
History King and Henzinger 1999

Note that the euler tour of the edges within this tree appear as such:



$$a \quad \underbrace{bdbebfb}_{\text{euler tour of subtree of b}} \quad acgchca$$

So we build the binary search tree of the euler tour:



Each edge points to both its nodes in the BST. Each node points to some instance of itself in the BST. Some implementations add an extra k sentry node at the root.

2.1 FindRoot

Simply return the minimum in the BST.

2.2 DeleteEdge

Split and join around interval that is being removed.

2.3 Reroot

Rotate the euler tour to make the splice point the root (with cosmetic changes).

2.4 AddEdge

Reroot both trees and $O(1)$ splits and joins.

All of these are $O(\log n)$.