Dynamic Trees

Jonathan Turner

April 6, 2013

In an earlier section, we discussed how the dynamic trees data structure can be used to speed up Dinic's algorithm for the maximum flow problem. In this section, we will look at how dynamic trees can be represented by another data structure, called the *path set* data structure. Then, in the next section, we will see how path sets can be implemented using self-adjusting binary search trees.

First, recall that the *dynamic trees* data structure represents a collection of trees on n vertices, where each vertex has an associated *cost*. The data structure defines the following operations.

- *findroot*(v) returns the root of the tree containing vertex v.
- *findcost*(v) returns a pair $[w, x]$ where x the is minimum cost for any vertex on the tree path from v to *findroot*(v) and w is the last vertex on the path with cost x.
- *addcost*(v, x) adds x to the cost of every vertex on the path from v to *findroot*(v).
- link(v, w) joins the tree with root v and the tree containing w by adding the edge (v, w).
- *cut*(v) divides the tree containing v into two trees by deleting the edge between v and $p(v)$.

Notice that some of the operations on the dynamic trees data structure are defined with respect to a path in the tree from some node to the root. We'll see that these operations can be carried out efficiently if we isolate the vertices on the path from the rest of the tree. To facilitate this, we represent each tree in the dynamic trees data structure as collection of linked paths, as illustrated in Figure 1.

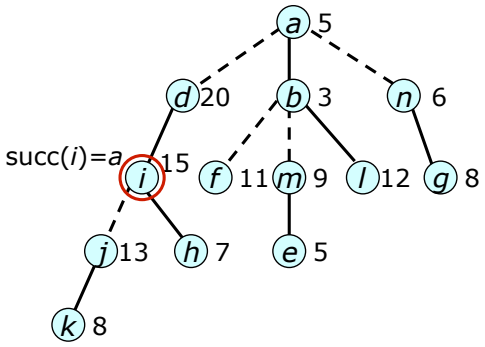

Figure 1: Representing a tree using paths

In the figure, the solid edges join nodes in a path and dashed edges are used to link the paths together. Paths are directed up the tree, so for example, h is the first node on its path and d is the last. These are also referred to as the *head* and *tail* of the the path, respectively. Each path has a designated *canonical node* that serves as the identifier for the path. The dashed edges in the diagram are implemented by defining a *successor* for each path, which is associated with the canonical node. So for example, if i is the canonical node of path $[h, i, d]$, then $succ(i) = a$.

Note that a given tree can be decomposed into paths in many different ways, and the decomposition may change over time. However, note that no node can have more than two solid edges, and if it has two, one of them must connect to its parent in the tree.

The paths in the dynamic trees data structure are represented using a *path set* data structure. Each path is identified by its *canonical node*. The operations on a path set are listed below.

- *findpath*(v) returns the path containing v (or more explicitly, the canonical node of the path containing v).
- *findtail*(p) returns the last node on the path p.
- *findpathcost*(p) returns a pair $[w, x]$ where x is the minimum cost for any node on the path and w is the last node on the path with cost x.
- *addpathcost*(p, x) adds x to the cost of every vertex on the path p.
- *join*(p, v, q) combines the path p, the single node v and the path q to form a new path that starts from the head of p and ends at the tail of q. Either p or q may be empty.

- *split*(v) divides the path containing v into a path that includes the nodes that come before v, another path that includes the nodes that come after v, and v itself. The pair of new paths is returned. If initially, v is the head or tail of its path, one or both of the returned paths may have length 0.

These definitions are illustrated in Figure 2.

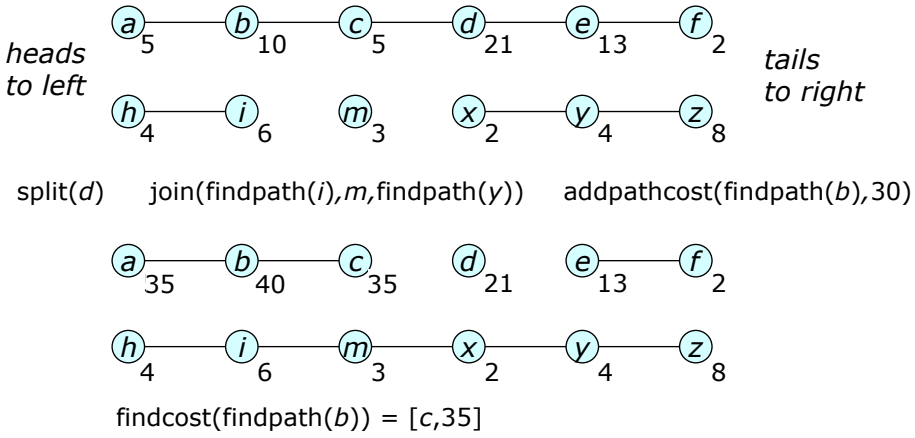

Figure 2: Path set example

The dynamic trees data structure is implemented in several levels. When first trying to understand the implementation, it's useful to focus on one level at a time, but it's also important to understand how the different levels relate to one another. Figure **??** illustrates these different levels. At the outermost

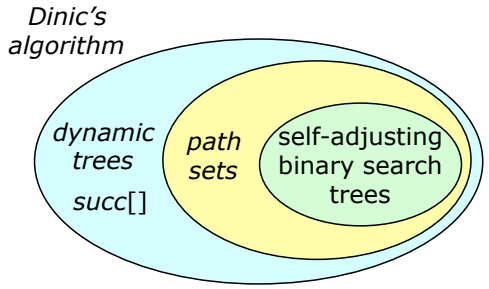

Figure 3: Levels of abstraction for dynamic trees

level, we have Dinic's algorithm itself. Dinic's algorithm uses the dynamic trees data structure in order to speed up its search for augmenting paths.

The dynamic trees data structure has two main components, the path sets data structure and the successor array $succ[]$, that defines how paths are linked together. The path sets data structure in turn, is represented as a set of binary search trees, where each search tree corresponds to one path. So, we're using trees to represent paths that implement trees that we use to find augmenting paths.

If you feel like your head is going to explode, just try to take it one layer at a time. You'll find that each layer is reasonably straightforward to understand on its own, and once you've gotten a handle on the individual layers, you'll find it easier to see how they relate to one another.