Dinic's Algorithm with Dynamic Trees

Jonathan Turner

March 30, 2013

Dinic's algorithm for the maximum flow problem has a worst-case running time of $O(mn^2)$. The highest label preflow-push algorithm beats this, with a running time of $O(m^{1/2}n^2)$. In this section we'll see how the introduction of a special data structure improves the worst-case running time of Dinic's algorithm to $O(mn \log n)$ which improves on the preflow-push algorithm by a factor of $O(n/m^{1/2} \log n)$ which can be a significant advantage if $m \ll (n/\log n)^2$.

The primary source of inefficiency in Dinic's algorithm is that successive augmenting searches may re-discover paths with positive residual capacity that were already found in previous searches. What if, instead of starting each search from scratch, we could remember portions of the graph that still have positive residual capacity? Then, instead of searching for an augmenting path edge-by-edge, we could try to combine existing path segments to produce the desired path. This idea is illustrated in Figure 1.

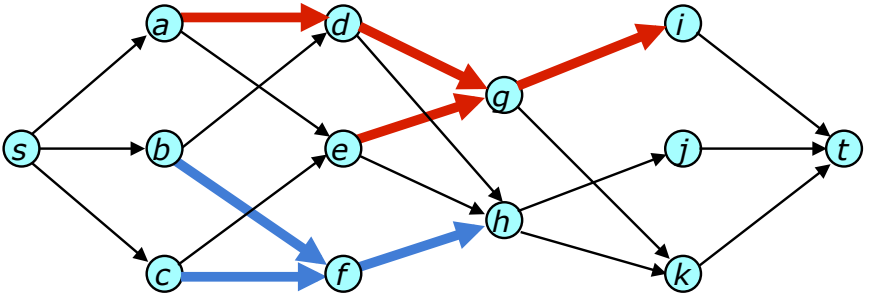

Figure 1: Dinic's algorithm with dynamic trees

The figure shows two directed subtrees of the residual graph, one with

root i and one with root h. These subtrees consist of edges (u, v) in the residual graph with $res(u, v) > 0$ and $level(v) = level(u) + 1$. Suppose we start a path search from s using the edge (a, s). Because a is part of the subtree with root i, we can use this information to avoid explicitly searching the edges on the path from a to i. We'll see that with the appropriate representation of the subtrees, we can jump to the root of a subtree in $O(\log n)$ time, potentially reducing the time needed for path searches by a factor of $O(n/\log n)$.

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 $p(v)$.

The data structure an be implemented so that any sequence of $m \geq n$ operations takes $O(m \log n)$ time. Figure 2 shows an instance of the data structure with two trees. The numbers next to the vertices are the costs.

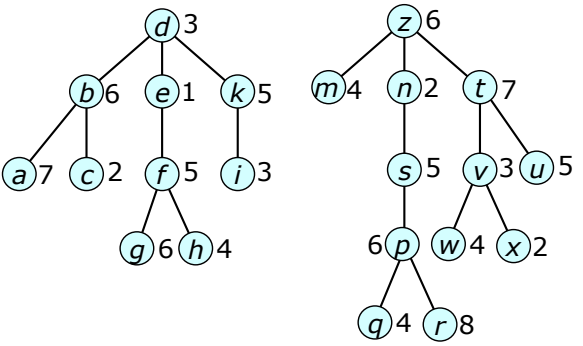

Figure 2: Dynamic trees example

Note that $\mathit{findroot}(x) = z$ and $\mathit{findcost}(i) = [d, 3]$. Figure 3 shows how the data structure changes following the operations $\mathit{cut}(k)$, $\mathit{link}(k, m)$ and $\mathit{addcost}(p, 3)$.

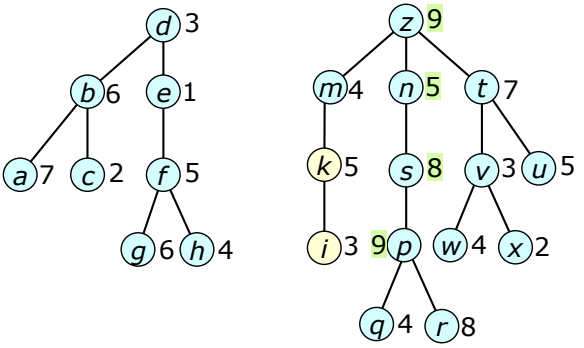

Figure 3: Dynamic trees example (continued)

When used with Dinic's algorithm, the costs are used to represent residual capacities. Specifically, if (u, v) is an edge in the tree, then $\mathit{cost}(u)$ is set equal to $\mathit{res}(u, v)$ in the flow graph. The *addcost* operation allows us to change the costs at all vertices on a tree path. This is used by Dinic's algorithm to effectively add flow to all edges on the tree path.