COS 528 notes

Janet Suzie Yoon

10/02/2006

1 Problems

Problem 1 (due 10/18): Prove the segmented path compression takes $O((n+m)log*n)$ time for up to n links and m intermixed evals.

Research Problem: Ge a better upperbound for segmented path compression or a lowerbound bigger than α per eval.

2 Segmented Paths continued

if $rank(x) = rank(root)$ AND some node on path to root has higher steppe, then new parent of x is the node on path of highest steppe (this is going to be unique).

every node on a compressed path ends up at most two steppes away from the root

$eval(x)$: computes value of node all the way to the root and does path compression along the way

$scompress(x)$: segmented path compression. Around half the nodes have their pointers changed (instead of all the nodes like in regular path compression)

eval(x):
begin
scompress(x);
if $p(x) = null$ then return $val(x)$
else if $p(p(x)) = null$ then return $val(x) \oplus val(p(x))$
else return $val(x) \oplus val(p(x)) \oplus val(p(x) \oplus val(p(p(x))))$
end

scompress(x):
begin
if $p(p(x)) \neq null$ then $scompress(p(x))$

if $p(p(x)) \neq null$ and $(r(p(p(x))) > r(x)$ OR $t(p(x)) < t(x))$
then $val(x) := val(x) \oplus val(p(x))$
$p(x) := p(p(x))$
if $p(p(x)) \neq null$ AND $(r(p(p(x)) > r(x)$ OR $t(p(x)) < t(x))$
then $val(x) := val(x) \oplus val(p(x))$
$p(x) := p(p(x))$
end

In path compression, the pointers end up threading btw the implicit binary tree

For top down compression analysis, the inequality $cost(c) < cost(c_t) + cost(c_b) + |x_b| + |c_t|$ works for regular compression. However since we have downward pointers in segmented path compression, this inequality breaks down.

3 Maximum Flow

1. Classical algorithms:
   (a) Augmenting Path Algorithms
   (b) Preflow Push Algorithms
   (c) Balancing Algorithms
2. Min cost flow
3. 0-1 case (max cardinality matching)

Given: Directed graph G, source s, sink t and positive capacities along each edge.

Problem: Get as much material possible from s to t.

Flow: non-negatibe function on edges obeying *capacity constraints* and *conservation constraints*.

Capacity constraint: $0 \leq f(v,w) \leq c(v,w)$ and either $f(v,w) = 0$ or $f(w,v) = 0$

Let *excess* at $v = e(v) = \sum_{(u,v)} f(u,v) - \sum_{(v,w)} f(v,w)$

Residual capacity of an edge (v,w): $C(v,q) - f(v,w)$ (forward) OR $f(w,v)$ (backward).

Ford Fulerson depends on the number of capacity since the number of augmentations correlate with the capacity. Can we get a flow dependent on the number of bits needed to represent the nodes or even better trongly polynomial (dependent to edges and nodes)? YES. Edmonds and Karp algorithm is strongly polynomial.

3.1 Edmonds Karp

Overview concept: Augment on "shortest" path (shortest in terms of the number of edges). Then the number of augmentations is polynomial to n and m. The running time for Edmonds Karp is $O(nm^2)$ Why? The length of the shortest augmented path is non-decreasing and the number of augmented paths of a given length is $\leq m$.

Define $d(x)$ = length of shortest path from s to x

We create a level graph of the residual graph, where each level is defined by $d(x)$. If (x, y) is a residual graph then $d(y) \leq d(x) + 1$. Any shortest augmenting path walks along edges that go from one level to the next. None of the backwards edges can participate in shortest path until $d(t)$ is increased. Every time we do an augmentation, we are "throwing" away one of the forward edges. We end up getting pieces without saturated flow, and thus end up with a forest of rooted trees.

3.2 o-1 case

Assume all capacities are 1. Total running time is $O(mk)$ where k is the number of distince augmented path flows. $k \leq n$ so uninterestingly enough the running time is O(mn)

Let $\hat{V}$ be the maximum flow value Let V be the current flow value

Claim: In the residual grpah, there is a flow of value $\hat{V} - V$. Flow breaks down into L edge disjoint paths. Therefore some augmenting path has $\leq m$Ł edges. So for the 0-1 case $\hat{V} - V = L$ and therefore $k \leq 2\sqrt{m}$ so the running time is $O(m^{3/2})$.