CS 170 Algorithms Fall 2013 Satish Rao HW 5

Due October 4, 6:00pm

1. (2 + 6 + 6 + 6 pts.) Even Faster DFS

Throughout this problem, you may assume that there are no self-edges.

A *one-way connected graph* is a directed graph $G = (V, E)$ with a special vertex $s \in V$ such that every other vertex is reachable from s. (There may be other such vertices, but you are only given s.)

Consider a one-way connected graph $G = (V, E)$ and a DFS from s. G stores the **tree edges** from this DFS in a data structure L. Note that the edges in L only need to correspond to the tree edges for *some* DFS started from s, so you can choose how the DFS breaks ties.

For $v \in V$, define $d_L(v)$ to be the number of children of v stored in L.

You may make the following assumptions:

- For any vertex v, we can find its parent edge (u, v) in L in $O(1)$ time.
- We can find a list of its children edges (v, u') in L in $O(1)$ time. This list is sorted in the order that they would be visited by a particular DFS from s.
- Insertion and removal of an edge (v, u) in L takes $O(d_L(v))$ time. For insertion, you must specify where in the list of children edges of v the edge should go - for example, if the edge (v, u) would be the last edge visited during a DFS, it should be inserted at the end of the list of children edges of v.
- We can find the pre number of a vertex in $O(1)$ time.

(a) Explain how we can now run a DFS starting from s in $O(|V|)$ time, assuming that L is given.

We want to modify the graph and maintain the data structure L efficiently, so that we can still run fast DFS.

(b) Suppose we want to add a new edge $e = (u, v)$ to G, where $u, v \in V$. In the following incomplete algorithm, $C(G, e)$ is some boolean function that returns true or false without modifying G or e.

1. Add e to G normally without updating L.
2. If $C(G, e)$ is true, find the parent edge of v given by (u', v) in L. Remove (u', v) from L, and add (u, v) to L. Otherwise, do nothing.

Show that this algorithm will not work. (Hint: Find (G, s, L, e) such that no matter whether C returns true or false, when the algorithm terminates we end up with a version of L such that *no matter how ties are broken,* there is no DFS on G started at s such that L will contain the tree edges of that DFS.)

(c) Suppose we have two vertices u and v such that $\texttt{pre}(u) > \texttt{pre}(v)$. Give and justify an algorithm to add the edge $e = (u, v)$ to G. Make sure that you maintain the invariant that L contains only tree edges from some DFS. Assume that adding the edge to G without updating L takes $O(1)$ time.

(d) Give and justify an $O(|V|)$ algorithm to add a **new** vertex v and **incoming** edges $(u_1, v), (u_2, v), \ldots (u_d, v)$ to G. Make sure that you maintain the invariant that L contains only tree edges from some DFS. Assume that adding v and the corresponding edges in G without updating L takes $O(d)$ time. (Hint: Remember that you can choose how the DFS breaks ties.)

(e) **This part will not be graded. You do not need to solve it. It is included for the sake of completeness.** Show how to implement the data structure L.

2. (2 + 8 pts.) Constructing the Graph for Fast DFS

Throughout this problem, assume that there are no self-edges.

Consider the following algorithm to construct a one-way connected graph $G = (V, E)$:

1. Start with an empty graph.
2. Repeat until G has been constructed:
   a. Add a new vertex v and **incoming** edges to that vertex v to the graph. Note that the incoming edges may only come from the nodes that have been added to the graph previously.

(Why would we want to do this? If we use the algorithm from Problem 1d, we can construct the graph in $O(|V|^2)$ time and then run DFS from Problem 1a in $O(|V|)$ time.)

(a) Give a one-way connected graph G which **cannot** be constructed using the algorithm above, no matter in what order the vertices are added to the graph.

(b) Consider a one-way connected graph G. We want to make the following statement:

G can be constructed using the algorithm above if and only if G has property P.

P should be a statement about the structure of the graph. What should the property P be? Prove your answer. (Answers like "P is that G can be constructed using the algorithm above" will receive no credit.)

3. (10 pts.) Problem 3.22

Give an efficient algorithm which takes as input a directed graph $G = (V, E)$ and determines whether or not there is a vertex $s \in V$ from which all other vertices are reachable.

(Note: This just checks whether G is one-way connected.)

4. (2 + 8 pts.) DFS edge types in BFS

Consider a directed graph $G = (V, E)$. Depth first search allows us to label edges as tree, forward, back and cross edges. We can make similar definitions for breadth first search on a directed graph:

i. A BFS tree edge is an edge that is present in the BFS tree.
ii. A BFS forward edge leads from a node to a nonchild descendant in the BFS tree.
iii. A BFS back edge leads to an ancestor in the BFS tree.
iv. All other edges are BFS cross edges.

(a) Explain why it is impossible to have BFS forward edges.

(b) Give an efficient algorithm that classifies all edges in G as BFS tree edges, back edges, or cross edges.

5. (10 pts.) Modified Problem 4.9

This is slightly different from problem 4.9 in the book.

Consider a directed graph in which the only negative edges are those that leave s; all other edges are positive. **In addition, s has no incoming edges.** Can Dijkstra's algorithm, started at s, fail on such a graph? Prove your answer.

Week 5 Fun Fact

Breadth first search may seem like a simple algorithm, but even now there is active research in making it faster. In fact, in 2012, UC Berkeley researchers came up with a new method of doing a breadth-first search. When combined with the conventional method, this can yield significant improvements.

The idea is simple - normally, to compute which vertices to use in layer $d+1$, BFS looks at all of the children in layer d. We can instead look at all of the unvisited vertices, and see if they have an edge to a vertex in layer d. This can avoid looking at all edges which gives it an advantage over the conventional approach in some cases.

You can read the paper at http://www.cs.berkeley.edu/~sbeamer/beamer-sc2012.pdf