CS369B: Problem Set #3

Due in class on Thursday, February 21, 2008

Instructions: Same as the first two problem sets.

Problem 11

Constant-Time Shortest-Path Queries. A *metric* on a finite set V can be thought of as a complete undirected graph on G with nonnegative edge lengths such that, for every $u, v \in V$, the shortest u-v path is just the one-hop path (u, v). We use $d(u, v)$ to denote the length of edge (u, v).

Solving a shortest-path problem on such a graph in the standard sense is obviously trivial (it looks like the output of such a problem, not the input). Our goal here is to compile this graph into a small (sub-quadratic-space) data structure that can answer queries of the form "what is $d(u, v)$?" in constant time. Intuition suggests (and it is true but not trivial to prove) that with sub-quadratic space we must settle for approximate answers.

Let k be an integer, at least 2. The data structure is based on successive random sampling. Define $A_0 = V$. For $i = 1, 2, \ldots, k-1$, obtain A_i from A_{i-1} be independently keeping each point with probability $n^{-1/k}$. Define $A_k = \emptyset$. Chernoff bounds show that $|A_i| \approx n^{1-i/k}$ for each i with very high probability. (you don't need to prove this, it is straightforward).

For each vertex $v \in V$, we maintain the following data. First, for each $i = 0, 1, \ldots, k-1$, v's nearest neighbor $p_i(v)$ among A_i (which is v itself, if $v \in A_i$), as well as the distance $\delta_i(v) = d(v, p_i(v))$. (Set $\delta_k(v) = +\infty$.) Second, we construct a set $B(v)$ of vertices w that meet the following property for some $i \in \{1, 2, \ldots, k\}$: w is included in A_{i-1}, is excluded from A_i, and w is closer to v than v's nearest neighbor in A_i (i.e., $d(v, w) < \delta_i(v)$). We explicitly store $d(v, w)$ for every $w \in B(v)$.

The idea for answering a query (u, v) is to identify a good "midpoint" vertex w, with distance roughly equal from u and v, and then return $d(u, w) + d(v, w)$ as an estimate for $d(u, v)$. (Since the Triangle Inequality holds in G, this can only be an overestimate. Of course, we better have both $d(u, w)$ and $d(v, w)$ stored to do this.) Our candidate w's are identified in iterations $i = 0, 1, 2, \ldots, k-1$ as follows: if i is even, we set $w_i = p_i(u)$ and halt with this w_i if $w_i \in B(v)$ (otherwise continuing to the next iteration); if i is odd, we set $w_i = p_i(v)$ and halt with this w_i if $w_i \in B(u)$ (otherwise continuing).

(a) Prove that, for every $v \in V$, the expected cardinality of the set $B(v)$ is at most $kn^{1/k}$. What upper bound does this imply on the expected size of the entire data structure (which should be decreasing in k and sub-quadratic for all $k \geq 2$).

[Hint: imagine first sorting vertices of A_{i-1} in order of distance from v, and then flip coins to determine membership in A_i.]

(b) Argue that, given a query, the algorithm above identifies a candidate w in $O(k)$ time. [Assume that given w, you can test membership in $B(u)$ or $B(v)$ in $O(1)$ time; this can be implemented via hashing, for example.]

(c) Prove that $\min\{d(u, w_i), d(v, w_i)\} \leq i \cdot d(u, v)$ for every $i = 0, 1, 2, \ldots, k-1$ throughout the query answering algorithm above (up to and including its final iteration).

(d) Prove that the query answering algorithm is guaranteed to return an estimate $\Delta := d(u, w_i) + d(v, w_i)$ satisfying $d(u, v) \leq \Delta \leq (2k-1) \cdot d(u, v)$.

(e) [Do not hand in.] Compare and contrast the goals and properties of this data structure with the distance-preserving subgraphs we studied in Problem #5.

Problem 12

Planar Separators and Approximation Algorithms. Planar separators enable good approximation algorithms for a number of NP-hard problems in planar graphs. Consider, for example, the unweighted Independent Set (IS) problem: given an undirected graph $G = (V, E)$, find a max-cardinality subset of mutually non-adjacent vertices. In general graphs, there are essentially no non-trivial polynomial-time approximation algorithms for this problem (assuming $P \neq NP$, of course).

(a) Prove that in a k-vertex graph, you can optimally solve the Max IS problem in $O(k^2 2^k)$ time.

(b) Prove that by applying the planar separator theorem recursively, it is possible to remove a set S of $O(n/\sqrt{k})$ nodes from a planar graph such that all connected components of $G \setminus S$ have size at most k. (If you can't prove an $O(n/\sqrt{k})$ bound, try to at least prove a bound of $O((n/\sqrt{k}) \log n)$.)

(c) Explain why every instance of Max IS has a feasible solution of size at least $n/4$.

[Hint: this follows from a very famous theorem.]

(d) Use (a)–(c) to give a "PTAS" (polynomial-time approximation scheme) for the Max IS problem in planar graphs: for an arbitrarily small but fixed constant $\epsilon > 0$, your algorithm should run in time polynomial in the input size and be guaranteed to output an IS with cardinality at least $(1 - \epsilon)$ times that of the largest IS of the input graph.

Problem 13

Treewidth. Another famous class of graphs for which many NP-hard problems become poly-time solvable is *bounded-treewidth graphs.* A nice discussion of such graphs can be found in Section 10.4 of the Kleinberg & Tardos algorithms book. Briefly, a *tree decomposition* of an undirected graph $G = (V, E)$ is defined by a grouping of V into (typically overlapping) sets $\mathcal{V} = \{V_1, \ldots, V_k\}$ and an edge set T on $\mathcal{V}$ such that four properties hold: (i) every vertex of the original graph G lies in at least one group V_i; (ii) for every edge e of the original graph G, both of its endpoints lie in some common group V_i of $\mathcal{V}$; (iii) the edges T form a tree on $\mathcal{V}$; and (iv) if V_1, V_2, V_3 are three groups of $\mathcal{V}$ on a path in T (with V_2 in the middle), then whenever V_1 and V_3 both contain some vertex v of the original graph G, then V_2 also contains v.

Obviously every graph has a tree decomposition (with one group that contains all nodes). The *width* of a tree decomposition is defined to be one less than the maximum size of one of its groups. The *treewidth* of a graph is the minimum width of a tree decomposition for it.

(a) [Do not hand in.] Convince yourself that every tree graph G has treewidth 1 (with a single group for each vertex of G and a group of size two for each edge of G).

(b) Prove that every treewidth k graph has a balanced vertex separator (in the sense of Lipton-Tarjan) of size at most $k + 1$.

[Hint: You can assume that no group of the tree decomposition contains another. As a warm-up, prove that every tree has a balanced separator consisting of a single node. Next use property (iv) to argue that non-leaf nodes of tree decompositions yield candidate separators for the underlying graph.]

(c) Give a polynomial-time algorithm for the following problem (which is NP-hard in general graphs). You are given an undirected graph $G = (V, E)$ and a tree decomposition of width 2 for it (which you can assume has at most n groups), as well as k vertex pairs $(s_1, t_1), \ldots, (s_k, t_k)$. Your task is to decide whether or not G contains an s_i-t_i path P_i for every i such that the paths $P_1, \ldots, P_k$ are all vertex-disjoint.

[Hint: dynamic programming over the tree decomposition. Consider the structure of a feasible solution and argue that $O(k^3)$ subproblems per group should suffice. Use property (iv) of tree decompositions to justify solving different subproblems independently.]

Problem 14

Biconnected Components. Let $G = (V, E)$ be an undirected graph. Recall that depth-first search (DFS) is simple in such graphs (see e.g. CLRS Section 22.3 to review). Starting from a node s, DFS creates a spanning tree of edges directed out of s. This directed spanning tree T is isomorphic to the recursion tree corresponding to DFS's execution (i.e., the children of a vertex v in T are precisely the vertices on which DFS was recursively invoked from v). All other edges of G appear as back edges, which can only lead from a vertex v back to one of v's ancestors in T. (Recall that in undirected graphs, DFS never produces cross or forward edges.)

(a) [Do not hand in.] A *cut vertex* of an undirected graph G is a vertex whose removal disconnects G. A graph with at least two vertices is *biconnected*, also known as *2-vertex-connected*, if it has no cut vertices. Convince yourself that an equivalent definition of biconnectivity is the following: for every pair e_1, e_2 of edges, G contains a cycle C including both e_1 and e_2.

Define a relation on edges by $e \equiv e'$ if and only if: (i) e, e' lie on a common cycle of G; or (ii) $e = e'$. Convince yourself that this is an equivalence relation (note transitivity requires a proof). The equivalence classes are called the *biconnected components* of G. The singleton classes are the *bridges* of G; convince yourself that these are precisely the edges whose removal will disconnect the graph.

(b) Prove that the cut vertices of G are precisely the vertices whose incident edges belong to more than one biconnected component.

(c) Let T be the DFS tree of G. Prove that every edge of T incident to the root lies in a different biconnected component.

(d) Rename the nodes in the order in which DFS first visits them. Define $\ell(v)$ to be the lowest-indexed (i.e., closest to the root) proper ancestor of v reachable by a back edge from the subtree T_v of T rooted at v (i.e., the lowest-indexed w such that there is a descendant u of v in T, possibly with $u = v$, with (u, w) a back edge). If no such back edges exist, define $\ell(v) = v$.

Consider consecutive tree edges (u, v) and (v, w) in T (with (v, w) further from the root). Prove that (u, v) and (v, w) lie on a common cycle in G if and only if $\ell(w) < v$ (i.e., $\ell(w)$ is a proper ancestor of v).

(e) Show how to compute all of the ℓ-labels in linear time.

(f) Show how to compute the biconnected components of G (i.e., labels for all edges such that two edges have the same label if and only if they are in the same biconnected component) in linear time. Note that given such labels one can easily list all of the cut vertices and bridges of the graph in linear time.

(g) [Do not hand in.] Convince yourself that this algorithm reduces planarity testing for general graphs to planarity testing for biconnected graphs (with only linear-time overhead).

Problem 15

Tree Covers. A *tree cover* of an unweighted, undirected graph is a collection $T_1, \ldots, T_k$ of spanning trees of G (not necessarily disjoint) so that, for every $u, v \in V$, a shortest u-v path in G is completely contained in one of the T_i's. Tree covers are useful in various routing problems. For dense graphs tree covers require $\Omega(n)$ trees. This problem shows that graphs with good separators admit relatively small tree covers.

(a) Show that the $\sqrt{n} \times \sqrt{n}$ grid admits a tree cover with only $O(\log n)$ trees.

(b) Consider a family of graphs that is closed under induced subgraphs and admits balanced separators (in the sense of Lipton-Tarjan) of size $f(n)$. (For example, $f(n) = O(\sqrt{n})$ for planar graphs and $f(n) = O(1)$ for graphs with bounded treewidth.) Prove that every graph from such a family admits a tree cover with only $O(f(n) \log n)$ trees. Can you sharpen the bound to $O(\sqrt{n})$ for planar graphs?

[Hint: Recursively apply separators as in Problem 12(b). Use shortest-path trees rooted at separator vertices.]

(c) **Extra Credit:** We can obtain radically smaller families of approximate tree covers in planar graphs. In a *3-approximate* tree cover, for every $u, v \in V$, there is a tree T_i such that its u-v path is no more than three times as long as the shortest u-v path in G. Prove that every planar graph has a 3-approximate tree cover with only $O(\log n)$ trees.

[Hint: this is not quite as difficult as previous extra credit problems. Review the proof of the planar separator theorem, and suppose you skipped the first step (the reduction to the low-diameter case). Then you get a (possibly large) balanced cyclic separator comprising two paths through the BFS (i.e., shortest-path) tree from s. Does this additional separator structure seem useful in this context?]