The University of British Columbia

Department of Computer Science

Computer Science 420—Advanced Algorithm Design and Analysis

Homework Assignment 6/7

Due: 2015 March 19

Please review the class policy on collaboration before starting this assignment. You are free to discuss problems in groups of size at most three. However, your actual homework submission must be prepared on your own. At the top of the first page of every homework submission you must clearly acknowledge all sources (including books, websites and discussions with fellow students) that you have used in the preparation of your submission.

1) Let $G = (V, E)$ be an undirected graph, and let $n = |V|$ and $m = |E|$.

We denote by $G^{[t]}$ the graph $(V, E^{[t]})$, where $(v_i, v_j) \in E^{[t]}$ if and only if there exists a path consisting of exactly t edges from v_i to v_j in G. Similarly, we denote by $G^{\langle t \rangle}$ the graph $(V, E^{\langle t \rangle})$, where $(v_i, v_j) \in E^{\langle t \rangle}$ if and only if there exists a path consisting of at most t edges from v_i to v_j in G. Finally, we denote by G^* the graph (V, E^*), where $(v_i, v_j) \in E^*$ if and only if there exists a path consisting of some (arbitrary) number of edges from v_i to v_j in G.

Let A, $A^{[t]}$, $A^{\langle t \rangle}$ and A^* denote the adjacency matrix representations of G, $G^{[t]}$, $G^{\langle t \rangle}$ and G^*, respectively.

a) Show that $a^{[2]}_{i,j}$, the i, j-th entry of $A^{[2]}$, is given by $a^{[2]}_{i,j} = \bigvee_k (a_{i,k} \wedge a_{k,j})$. (Thus $A^{[2]} = (A^{[1]})^2$, the Boolean matrix product of $A^{[1]}$ with itself.) Similarly, show that $A^{\langle 2 \rangle} = (A^{\langle 1 \rangle})^2$.

b) Prove that $A^{[t]} = (A^{[1]})^t$ and $A^{\langle t \rangle} = (A^{\langle 1 \rangle})^t$, for all $t \geq 1$.

c) Show that $A^* = A^{\langle t \rangle}$, for all $t \geq n - 1$.

2) [From Cormen et al., Problem 24-4 (page 615): *Gabow's scaling algorithm for single-source minimum-cost paths*]

A *scaling* algorithm solves a problem by initially considering only the highest-order bit of each relevant input value (such as an edge weight). It then refines the initial solution by looking at the two highest-order bits. It progressively looks at more and more higher order bits, refining the solution each time, until all bits have been considered and the correct solution has been computed.

In this problem, we examine an algorithm for computing $\delta(s, v)$, the cost of the minimum-cost path from a single source s to vertex v, for all vertices $v \in V$, by scaling edge costs. We are given as input an n-node, m-edge directed graph $G = (V, E)$ with non-negative integer edge cost function c. Let $C = max_{(u,v) \in E}\{c(u, v)\}$. Our goal is to develop an algorithm that runs in $O(m\, lg(C + 1))$ time. We assume that all vertices are reachable from the source vertex s, so in particular $m \geq n - 1$.

The algorithm uncovers the bits in the binary representation of the edge costs one at a time, from the most significant bit to the least significant bit. Specifically, let $k = \lceil lg(C + 1) \rceil$ be the number of bits in the binary representation of C, and for $i = 1, 2, \ldots, k$, let $c_i(u, v) = \lfloor c(u, v)/2^{k-i} \rfloor$. That is, $c_i(u, v)$ is the "scaled-down" version of $c(u, v)$ given by the i most significant bits of $c(u, v)$. (Thus $c_k(u, v) = c(u, v)$, for all $(u, v) \in E$.) For example, if $k = 5$ and $c(u, v) = 25$, which has the binary representation $\langle 11001 \rangle$, then $c_3(u, v) = \langle 110 \rangle = 6$. As another example, with $k = 5$, if $c(u, v) = \langle 00100 \rangle = 4$, then $c_3(u, v) = \langle 001 \rangle = 1$.

Let us define $\delta_i(u,v)$ to be the cost of the minimum-cost path from vertex u to vertex v when we use cost function c_i. Thus $\delta_k(u,v) = \delta(u,v)$, for all $u, v \in V$. The scaling algorithm first computes the minimum-cost values $\delta_1(s,v)$ for all $v \in V$, then computes $\delta_2(s,v)$ for all $v \in V$, and so on, until it computes $\delta_k(s,v)$ for all $v \in V$. We shall see that computing δ_i values from δ_{i-1} values takes $O(m)$ time in total, from which we conclude that the entire algorithm takes $O(km) = O(m \lg C)$ time.

a) Suppose that for all vertices $v \in V$, we have $\delta(s,v) \leq m$. Show that we can compute $\delta(s,v)$ for all $v \in V$ in $O(m)$ time in total. (Hint: Use Dijkstra's algorithm, maintaining the priority queue of the d-values associated with elements of $V - S$ as a list structure $L[0 \ : \ m+1]$, where $L[i]$ points to a doubly-connected list containing all vertices $v \in V - S$ with $d[v] = i$, and $L[m+1]$ points to a doubly-connected list of vertices $v \in V - S$ with $d[v] > m$. Exploit the facts that(i) d-values only decrease over time, and (ii) the d-values extracted from the priority queue increase monotonically over time.)

b) Show that we can compute $\delta_1(s,v)$ for all $v \in V$ in $O(m)$ time in total.

We now focus on computing δ_i values from δ_{i-1} values.

c) Prove that for $i = 2, 3, \ldots, k$, we have either $c_i(u,v) = 2c_{i-1}(u,v)$ or $c_i(u,v) = 2c_{i-1}(u,v) + 1$. Then, prove that

$$2\delta_{i-1}(s,v) \leq \delta_i(s,v) \leq 2\delta_{i-1}(s,v) + |V| - 1$$

for all $v \in V$.

d) For $i = 2, 3, \ldots, k$ and for all $(u,v) \in E$, define $\widehat{c}_i(u,v)$ as:

$$\widehat{c}_i(u,v) = c_i(u,v) + 2\delta_{i-1}(s,u) - 2\delta_{i-1}(s,v).$$

Prove that for $i = 2, 3, \ldots, k$ and for all $(u,v) \in E$, the "reweighted" value $\widehat{c}_i(u,v)$ of edge (u,v) is a non-negative integer.

e) Now define $\widehat{\delta}_i(s,v)$ as the weight of the shortest path from s to v using the weight function $\widehat{c}_i$. Prove that for $i = 2, 3, \ldots, k$ and for all $v \in V$,

$$\delta_i(s,v) = \widehat{\delta}_i(s,v) + 2\delta_{i-1}(s,v)$$

and that $\widehat{\delta}_i(s,v) \leq |E|$.

f) Show how to compute $\delta_i(s,v)$ from $\delta_{i-1}(s,v)$ for all $v \in V$ in $O(m)$ time, and conclude that $\delta(s,v)$ can be computed for all $v \in V$ in $O(m \lg(C+1))$ time.

3) The *vertex cover* problem takes as input an undirected graph $G = (V_G, E_G)$ and an integer k and asks if there exists a subset V' of V_G of size k such that every edge $(u,v) \in E_G$ is *covered* by V', that is either $u \in V'$ or $v \in V'$ (or both). The vertex cover problem is known to be **NP**-hard.

Suppose that $V_G = \{v_1, v_2, \ldots, v_n\}$ and $E_G = \{e_1, e_2, \ldots, e_m\}$. Consider the edge-coloured graph $H = (V_H, E_H)$, where $V_H = \{v'_0, v'_1, \ldots v'_m\}$ and E_H contains an edge from v'_{i-1} to v'_i, with colour c_j exactly when vertex v_j is an endpoint of edge e_i in G.

a) The *minimum colour $s-t$-path problem* (recall Assignment 7, Question 1b) takes as input an edge-coloured graph H with two specified vertices s and t, and an integer r, and asks if there exists a path from s to t in H that uses at most r edge colours. Show how to use the transformation from G to H above to provide a reduction from the vertex cover problem to the coloured $s-t$-path problem.

b) Argue that your reduction can be carried out in time bounded by some polynomial in the size of the input graph G.

c) Is the coloured $s-t$-path problem **NP**-hard? Is it **NP**-complete? Explain your answers

4) A Boolean expression over a set $\{x_1, x_2, \ldots, x_n\}$ of Boolean variables is in 2-CNF (conjunctive normal form) if it is the conjunction (logical *and*) of a set of disjuncts, each of which is the disjunction (logical *or*) of two literals, each of which is either a variable x_i or its negation $\overline{x}_i$. For example, here an expression in 2-CNF:

$$(x_1 \vee \overline{x}_2) \wedge (\overline{x}_1 \vee \overline{x}_3) \wedge (x_1 \vee x_2) \wedge (\overline{x}_2 \vee x_4) \wedge (\overline{x}_1 \vee x_4)$$

In the 2-SAT problem, you are given an expression $\mathcal{E}$ in 2-CNF and you want to find an assignment of `true` or `false` to each of the variables appearing in $\mathcal{E}$ so that the expression is satisfied – that is, at least one literal in each disjunct is assigned the value `true`. (The expression above has a satisfying assignment: set x_1, x_2, x_3, and x_4 to `true`, `false`, `false`, and `true`, respectively.)

The purpose of this problem is to lead you to a way of solving the 2-SAT problem efficiently by reducing it to a question about paths in an associated graph. Given an expression $\mathcal{E}$ with n variables and m disjuncts, construct a directed graph $G = (V, E)$ as follows: (i) V consists of $2n$ vertices, one for each variable and its negation; (ii) E consists of $2m$ edges: for each disjunct $(\alpha \vee \beta)$ of $\mathcal{E}$ (where α and β are literals) G has an edge from the negation of α to β, and an edge from the negation of β to α.

a) Show that if there is a path from a literal α to a literal β in G, then any satisfying truth assignment of $\mathcal{E}$ that assigns `true` to α must also assign `true` to β.

b) Show that if there is a path from a literal α to its negation, and a path from the negation of α to α, then $\mathcal{E}$ is not satisfiable.

c) If there is no pair of paths of the form described in part (b) then $\mathcal{E}$ is satisfiable. Prove that this is so by describing an efficient procedure to construct a satisfying truth assignment, in this case. (Hint: form the transitive closure of G.)