CSI 604 – Spring 2016

Solutions to Homework – VII

Problem 1:

Part (a): We need to prove the following result.

Lemma 1: Let $G(V, E)$ be a connected undirected graph with $|V| = n$. Assume that n is even. Suppose G has two nodes s and t such that every simple path between s and t has strictly greater than $n/2$ edges. Then G has a node w, which is different from s and t, such that deleting w from G destroys *all* the paths between s and t.

Proof: Consider any breadth-first spanning tree T of G with s as the root. In T, each node v of G has a *level*, which is the length of (i.e., number of edges in) a shortest path from s to v. Since every path between s and t has length at least $\ell = (n/2) + 1$, node t occurs at a level $\geq \ell$. Consider levels 1 through $n/2$. The total number of nodes in levels 1 through $n/2$ is at most $n - 2$ (since nodes s and t don't appear in any of these levels). If each of these $n/2$ levels has 2 or more nodes, then total number of nodes in G will exceed n. Therefore, there must be a level in the range 1 through $n/2$ containing just one node, say w. Clearly, if we delete w, the resulting graph has no path between s and t. ■

Part (b): An algorithm for the problem follows directly from the proof of Lemma 1.

High-Level Description:

1. Construct a breadth-first search of G, starting with node s. For each level i, construct the list $L[i]$ of the nodes at level i.
2. Find a level j, where $1 \leq j \leq n/2$, such that $L[j]$ has only one node, say w. (The proof of Lemma 1 shows that such a level must exist.)
3. Output node w.

Correctness and Running Time: The correctness of the above algorithm follows from the proof of Lemma 1. For the running time, note that Step 1 can be done in $O(m + n)$ time since it involves just a breadth-first search of G. Steps 2 and 3 take $O(n)$ and $O(1)$ time respectively. So, the algorithm runs in $O(m + n)$ time.

Problem 2:

The dynamic programming approach used for this problem is very similar to the Floyd-Warshall algorithm for computing all pairs shortest paths.

Preliminaries: We assume that the nodes of the undirected graph $G(V, E)$ are numbered 1 through n, where $n = |V|$. We use $W = [w_{ij}]$ to represent the $n \times n$ weight matrix, where $w_{ij} = w_{ji}$ gives the weight of the undirected edge $\{i, j\}$.

As usual, we assume that $w_{ii} = 0$, $1 \leq i \leq n$. Also, if G does not have the edge $\{i, j\}$, we assume that $w_{ij} = \infty$.

Since all the edge weights are non-negative, only simple paths (i.e., paths with no repeated vertices) need to be considered to find minimum bottleneck cost paths.

(a) Information about the table used for Dynamic Programming: Let $d_{ij}^{(k)}$ represent the minimum bottleneck weight along paths between nodes i and j such that in each such path, intermediate vertices used are from the set $\{1, \ldots, k\}$, $0 \leq k \leq n$.

(b) Computing the entries in Dynamic Programming table:

1. Consider the value $k = 0$. The set of intermediate nodes $\{1, \ldots, 0\}$ is the empty set; that is, this step must consider only paths between nodes i and j with no intermediate vertices. Therefore, for all i and j, $d_{ij}^{(0)} = w_{ij}$.

2. Suppose we have computed all the entries $d_{ij}^{(k-1)}$ for some $k \geq 1$. We can compute the entries $d_{ij}^{(k)}$ by considering the following two possibilities.

   (i) There is a minimum bottleneck cost path between i and j such that all the intermediate nodes used in the path are from $\{1, 2, \ldots, k-1\}$. In this case, $d_{ij}^{(k)} = d_{ij}^{(k-1)}$.

   (ii) Every minimum bottleneck cost path between i and j uses k as an intermediate vertex. Consider any such path P. Since P is simple, its two sub-paths, one between i and k and the other between k and j, use only nodes from $\{1, \ldots, k-1\}$ as intermediate vertices. Therefore,

   $$d_{ij}^{(k)} = \min\{d_{ij}^{(k-1)},\ \max\{d_{ik}^{(k-1)},\ d_{kj}^{(k-1)}\}\}$$

   (We use "max" in the above expression since we are computing bottleneck costs.)

   (iii) Once we compute all the entries $d_{ij}^{(n)}$, we will have the required bottleneck costs.

(c) High-level description of the algorithm: This is obtained by a slight modification of the Floyd-Warshall algorithm. (We use the matrix $D^{(k)}$ to represent the entries $d_{ij}^{(k)}$, $0 \leq k \leq n$.)

1. $D^{(0)} = W$.

2. **for** $k = 1$ **to** n **do**

   for $i = 1$ **to** n **do**

   for $j = 1$ **to** n **do**

   $$d_{ij}^{(k)} = \min\left\{d_{ij}^{(k-1)},\ \max\{d_{ik}^{(k-1)},\ d_{kj}^{(k-1)}\}\right\}.$$

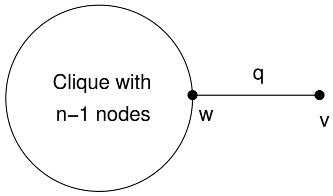
3. Return $D^{(n)} = \left[d_{ij}^{(n)}\right]$.

Note: We can implement the above algorithm using just two $n \times n$ matrices, say X and Y (in addition to the W matrix). We start with X as $D^{(0)}$ and compute $Y = D^{(1)}$. For the next step, we can use Y as the input and use X to store the new matrix $D^{(2)}$, and so on.

(d) Running time analysis: Steps 1 and 3 of the above algorithm use $O(n^2)$ time. In Step 2, there are three nested **for** loops, each of which iterates n times. Thus. the total number of iterations of these loops is $O(n^3)$. In each of these iterations, the time used to compute $d_{ij}^{(k)}$ is $O(1)$. Therefore, Step 2, and hence the whole algorithm, runs in $O(n^3)$ time.

Problem 3: The statement is *false*.

The following counterexample shows that one can construct graphs G for which ratio $\delta(G)/D(G)$ is *not* bounded by any constant c; it can be made arbitrarily large.



Let n be a sufficiently large integer. Consider the graph G on the left which consists of a clique on $n-1$ nodes (with node w being part of this clique) along with an edge to a node v (of degree 1). The weight of the edge $\{w, v\}$ is $q = n(n-1)$. All the other edges have weight 1.

The diameter $\delta(G)$ of the above graph is $q+1$ (which is the length of a shortest path between v and any node of the $n-1$ clique other than w). The sum $\sigma(G)$ of the shortest path distances between pairs of nodes in G can be decomposed into two parts as follows.

(a) The sum of the shortest distances between all pairs of nodes in the $n-1$ clique is given by $(n-1)(n-2)/2$ (since there are $(n-1)(n-2)/2$ edges in that clique and each edge has weight $= 1$).

(b) The sum of the shortest path distances between v and the other nodes is given by $q+(n-2)(q+1)$ (since the shortest path between v and w has length q and that between v and any other node of the $n-1$ clique has length $q+1$).

Thus, $\sigma(G) = (n-1)(n-2)/2 + q + (n-2)(q+1)$. Therefore, the average pairwise distance $D(G)$ is given by

$$\begin{aligned} D(G) &= \frac{\sigma(G)}{n(n-1)/2} \\ &= \frac{[(n-1)(n-2)/2] + q + (n-2)(q+1)}{n(n-1)/2} \\ &= \frac{n-2}{n} + \frac{2q}{n(n-1)} + \frac{2(n-2)(q+1)}{n(n-1)} \end{aligned}$$

Since $q = n(n-1)$, we have from the last equation,

$$D(G) \ < \ 1 + 2 + 2(n-2) \ < \ 2n$$

Since $\delta(G) = q+1 \ > \ n(n-1)$ and $D(G) < 2n$, the ratio $\delta(G)/D(G)$ is greater than $(n-1)/2$; this value can be made arbitrarily large by choosing an appropriate value of n.