CSCI-GA.1170-001/002 Fundamental Algorithms December 3, 2015

Solutions to Problem 1 of Homework 10 (8 points)

Name: GOWTHAM GOLI (N17656180) *Due: Tuesday, December 1*

You are given a directed graph $G = (V, E)$ on n nodes and m edges, where the node set $V = A \cup B$ consists of two disjoint subsets A and B of sizes n_1 and n_2 (so $n = n_1 + n_2$). Nodes in A are "healthy", while nodes in B are "infected". Given a source $s \in A$, your goal is to compute the shortest distance from s to every other healthy node which can pass through *at most one* infected node (i.e., if the path from s to v contains at most one infected u, this is OK, but if it contains two or more, this path is not allowed when computing the shortest distance).

Define the following *directed* graph $G' = (V', E')$ on $2n_1 + n_2$ nodes. The vertex set of V' of G' is $V' = A_1 \cup B \cup A_2$, where A_1 and A_2 are two copies of healthy nodes A. Two nodes in A_1 are connected in G' if and only if they are connected in G, and the same between two nodes in A_2. The nodes in B are more interesting. For every original incoming edge $(a, b) \in E$, where $a \in A$ and $b \in B$, we will put an edge (a_1, b) in E', where a_1 is the copy of a in A_1. Similarly, for every original outgoing edge $(b, a) \in E$, where $a \in A$ and $b \in B$, we will put an edge (b, a_2) in E', where a_2 is the copy of a in A_2.

(a) (2 pts) Let n', m' be the number of vertices and edges in G'. Show that $n' \leq 2n$ and $m' \leq 2m$.

Solution:

$n' = 2n_1 + n_2 = n + n_1 < n + n = 2n \implies n' < 2n$

Let number of edges within A be m_1 and the number of edges within B be m_2 and the number of edges between A and B be m_3. Hence $m = m_1 + m_2 + m_3$.

$\therefore |E'| = m' = m_1 + m_1 + m_3 < m_1 + m_1 + m_3 + (m_2 + m_2 + m_3) = 2m \implies m' < 2m$ $\square$

(b) (4 pts) Recall our original problem of computing the required shortest distance in G from s to every other healthy node $a \in A$ which can pass through *at most one* infected node $b \in B$. Call this distance $a[dis]$. Let s_1 and s_2 be the two copies of s in G'. Using one "appropriate" BFS call on G', show how to compute the values $a[dis]$. Specifically, say what is the starting node (call it s') of your BFS call in G'. Also, after your BFS call computed shortest distances $v'.d$ from s' to v', for every $v' \in V'$, show how to compute the desired values $a[dis]$ for the problem at hand (i.e., write an explicit formula for $a[dis]$ using appropriate $v'.d$ values). Justify your algorithm.

Solution:

Let s' be s_1 i.e the copy of s in A_1 and now call BFS on s' in G'. This gives us the shortest distance from s' to every other node in G' i.e $v'.d$, for every $v' \in V'$.

However we are only interested in $v' \in A_1 \cup A_2$ i.e the copy of all the vertices of A. Consider some vertex $a \in A$. There can be multiple paths from s to a in G i.e the paths containing zero infected nodes or one infected nodes or two infected nodes etc.

Now $a[dis]$ is the shortest distance from s to a containing atmost one infected node. Therefore this can be broken down into two parts i.e the minimum of the shortest distance from s to a containing 0 infected nodes, let it be $a[dis_0]$ or the shortest distance from s to a containing 1 infected nodes, let it be $a[dis_1]$.

$$a[dis] = min\{a[dis_0], a[dis_1]\}$$

Let $a_1 \in A_1$ and $a_2 \in A_2$ be the copies of the vertex $a \in A$. Note that the shortest path from s' to a_1 has 0 number of infected nodes in it as there are no edges from the B to A_1 and the path from s_2 to a_2 has exactly 1 infected node as we need to visit exactly one node in B to reach any node in A_2. Therefore substitute these values of $a_1.d = a[dis_0]$ and $a_2.d = a[dis_1]$ in the above equation

$$a[dis] = min\{a_1.d, a_2.d\}$$

where a_1 and a_2 are the copies of a in A_1 and A_2 □

(c) (2 pts) Show that the running time of your procedure is $O(m + n)$.

Solution:

In the above algorithm we called BFS on s' in G'. This step takes $O(|V'| + |E'|) = O(m + n)$ time and then we take minimum of $a_1.d$ and $a_2.d$ for every $a \in A$. This step takes $O(|A|) = O(|V|) = O(n)$ time. Therefore the total running time of the algorithm is $O(m + n)$ □

CSCI-GA.1170-001/002 Fundamental Algorithms December 3, 2015

Solutions to Problem 2 of Homework 10 (15 points)

Name: GOWTHAM GOLI (N17656180) *Due: Tuesday, December 1*

A fellow Moe and his buddy Joe live in a city $G = (V, E)$ which is an undirected graph on n vertices and m edges, given in the adjacency list form. Moe lives in a vertex a and owns a crazy dog Mimi, while Joe lives at a vertex b and owns a crazy dog Kiki. This Sunday Moe wants to take Mimi to a veterinarian clinic located at vertex c, while Joe wants to take Kiki to the dance competition located at a vertex d. One problem, though: the dogs hate each other, and if one of them smells the other, all hell breaks loose. Luckily, they can smell each other only if within distance at most 15 in G, and you know that $dist_G(a, b), dist_G(c, d) > 15$. Moe and Joe would like to start at the same time (with Moe and Mimi at a and Joe and Kiki at b) and get both dogs to their respective destinations c and d in the smallest number of *steps* t. A *step* consists of both dogs going to their respective neighboring vertices, or one dog going to a neighboring vertex, and the other dog staying put, barking at the pedestrians. Of course, such a step is possible only if the dogs stay within distance 16 or more both before and after the step.

Your job is to design an algorithm for Moe and Joe to compute t (and the optimal route), if the route exists, and analyze its running time.

(a) (5 pts) Using one or more runs of the BFS algorithm on G, fill in the matrix $OK[x, y]$, where $OK[x, y] = 1$, if it is OK for Mimi to be at vertex x and Kiki to be at vertex y at the same time, and $OK[x, y] = 0$ otherwise. How long did it take you to fill this matrix $OK[x, y]$?

Solution:

For every node $x \in V$ compute $BFS(x)$
and $\forall y \in V$, if $\delta(x, y) > 15$ set $OK[x, y] = 1$ and 0 otherwise.

In the above algorithm BFS is called on every node of G. Therefore the running time is $O(n(m + n)) = O(mn)$ □

(b) (5 pts) Design a graph $H = (V', E')$ whose vertex set consists of possible "ok configurations" for Mimi and Kiki, and whose edge set represents the possible single steps of your algorithm. Be sure to formally define V' and E' as functions of V and E and the matrix OK from part (a). How long (in the worst case) did it take you to create an adjacency list for H (not counting what you did in part (a))? What is the maximum $|V'|$ and $|E'|$?

Solution:

If for any $x, y \in V$, $OK[x, y] = 1$ then we create a single node for both x and y. Let this node be called xy. Therefore $V' = \{xy$ such that $OK[x, y] = 1$ where $x, y \in V\}$. Therefore creating V' takes $O(n^2)$ time

Let x_1y_1 and x_2y_2 be any two vertices in V' then

- $(x_1y_1, x_2, y_2) \in E'$ if $OK[x_1, y_1] = 1$, $OK[x_2, y_2] = 1$ and $(x_1, x_2), (y_1, y_2) \in E$ i.e if Initially Mimi is at x_1 and Kiki is at y_1. Now each of them take one step, go to their respective neighbors x_2 and y_2 and they are still > 15 units apart from each other
- $(x_1y_1, x_2, y_2) \in E'$ if $OK[x_1, y_1] = 1$, $OK[x_1, y_2] = 1$ and $(y_1, y_2) \in E$ i.e if Initially Mimi is at x_1 and Kiki is at y_1. Now Mimi stays put at x_1 while Kiki goes to his neighboring vertex y_2 and they are still > 15 units apart from each other
- $(x_1y_1, x_2, y_1) \in E'$ if $OK[x_1, y_1] = 1$, $OK[x_2, y_1] = 1$ and $(x_1, x_2) \in E$ i.e if Initially Mimi is at x_1 and Kiki is at y_1. Now Kiki stays put at y_1 while Mimi to goes his neighboring vertex x_2 and they are still > 15 units apart from each other

Let $xy \in V'$. Now we need to analyze the number of outgoing edges from xy. Let m_x be the number of outgoing edges from x to it's neighboring vertices $x_1, \ldots, x_{m_x}$ and m_y be the number of outgoing edges from y to it's neighboring vertices $y_1, \ldots, y_{m_y}$ in G. Now for each $x_i \in \{x_1, \ldots, x_{m_x}\}$ and $y_j \in \{y_1, \ldots, y_{m_y}\}$, we check if $OK[x_i, y_j]$ is 1 and add an edge from xy to x_iy_j. Therefore the total time taken to add the number of outgoing edges from xy is m_xm_y.

Let $x_1, \ldots, x_n \in V$ and $y_1, \ldots, y_n \in V$ be all the vertices of G. In the worst case if $OK[x_1, y_i] = 1$ for each $i = 1, \ldots, n$ then the time taken to add all the outgoing edges from each vertex x_1y_i is $m_{x_1}(m_{y_1} + \ldots + m_{y_n}) = m_{x_1}m$. Similarly we can analyse for all $x_1, \ldots, x_n \in V$, Therefore the total time taken to create E' is $m(m_{x_1} + \ldots + m_{x_n}) = m^2$

Therefore the total time to create the adjacency list of H is the time to create V' and the time taken to $E' = O(m^2 + n^2)$.

In the worst case when all the vertices are more than 15 units from each other the entire OK matrix will be 1. Therefore $|V|_{max} = n^2$ and using the above analysis of how outgoing edges are created it is easy to see that $|E|_{max} = m^2$ □

(c) (5 pts) Describe the original problem as a shortest path computation on H. Finally, solve the original problem, and help Moe and Joe. Analyze the overall running time of your algorithm, as a function of n and m.

Solution:

All the vertices and edges of H are of OK configurations. So simply call BFS on ab and get the shortest distance from ab to cd and return $\delta(ab, cd)$. Running time will be $O(|V'|+|E'|) = O(m^2 + n^2)$ □

CSCI-GA.1170-001/002 Fundamental Algorithms December 3, 2015

Solutions to Problem 3 of Homework 10 (12 points)

Name: GOWTHAM GOLI (N17656180) *Due: Tuesday, December 1*

The diameter of an undirected tree $T = (V, E)$ on n vertices V (and $(n-1)$ edges E) is the largest of all shortest paths distances in the tree: $D = \max_{x,y \in V} \delta(x, y)$. You will design an $O(n)$ algorithm to compute D and will prove its correctness as follow.

(a) (7 pts) Let r be the root of T. Let b is the furthest node from r in T. Show that the diameter path in T either ends or starts at b.

Solution:

Let $\delta(a, b)$ be the diameter of T and b is the farthest node from r. Note that the root r may or may not lie in the path between a and b.

Consider that r lies in the path between a and b i.e r is the lowest common ancestor of a and b. Therefore $\delta(a, b) = \delta(a, r) + \delta(r, b)$. Suppose that $\delta(a, b)$ is not the diameter of T and let $\delta(a, c)$ be the diameter of T. Therefore $\delta(a, c) = \delta(a, r) + \delta(r, c)$. We know that $\delta(r, c) < \delta(r, b)$ as b is the farthest node from r. Hence $\delta(a, c) < \delta(a, b)$. Hence our assumption is wrong i.e c has to be the farthest node from r. Hence $c = b$

Now consider that r doesn't lie in the path between a and b and let p be the lower common ancestor of a and b. Therefore $\delta(a, b) = \delta(a, p) + \delta(p, b)$. Suppose that $\delta(a, b)$ is not the diameter of T and let $\delta(a, c)$ be the diameter of T. Therefore $\delta(a, c) = \delta(a, p) + \delta(p, c)$. Since b is the farthest node from r we have, $\delta(r, c) < \delta(r, b) \implies \delta(r, p) + \delta(p, c) < \delta(r, p) + \delta(p, b) \implies \delta(p, c) < \delta(p, b)$. Hence $\delta(a, c) < \delta(a, b)$. Hence our assumption is wrong i.e c has to be the farthest node from r. Hence $c = b$

Note that in the above two cases we considered that the diameter ends at b. We can similarly argue when the diameter starts at b. Therefore we can conclude that the diameter path in T starts or ends at b where b is the farthest node from the root of the tree □

(b) (5 pts) Assuming part (a), irrespective of whether or not you solved it, design an $O(n)$ algorithm to compute D. For partial credit, give a slower algorithm.

Solution:

- Pick the root node r and perform BFS on it
- Let b be the farthest node from r. Therefore from part (a), the diameter of T either starts or ends at b
- Perform BFS on b. Let a be the the farthest node from b
- $\delta(a, b)$ is the the diameter of T

In the above algorithm, BFS is called twice so the running time is $O(m + n)$ but we know that $m = n - 1$. Therefore the running time is $O(n)$ □

CSCI-GA.1170-001/002 Fundamental Algorithms December 3, 2015

Solutions to Problem 4 of Homework 10 (6 points)

Name: GOWTHAM GOLI (N17656180) *Due: Tuesday, December 1*

Give an algorithm that determines whether or not given a undirected graph $G = (V, E)$ contains a cycle. Your algorithm should run in $O(V)$ time, independent of $|E|$. Make sure you prove the correctness of your algorithm.

Solution:

Algorithm

There will be a cycle in G if in the DFS of G, any unexplored edge visits a node that has already been visited earlier i.e this unexplored edge is a backward edge in the DFS forest of G. Therefore if the DFS yields no back edges i.e it contains only tree edges then G is acyclic or else G contains a cycle

Correctness and Running Time

If the given graph G is acyclic then the DFS forest will not contain any egde (u, v) such that v is an ancestor of u. Suppose there exist such an edge (u, v) then $DFS(v)$ will visit u as v is the ancestor of u and then $DFS(u)$ visits v which is already visited thus resulting in a cycle. Therefore it contradicts our assumption and hence the edge (u, v) can't exist i.e DFS yields no back edges and contains only tree edges if G is acyclic

Therefore if G is acyclic, then maximum number of tree edges a graph can have is at most $|V| - 1$. Therefore a single run of DFS is sufficient to check for back edges in this case. Therefore the running time will be $O(m + n) = O(2n) = O(n)$

Suppose that G contains a cycle, then DFS of G will contain at least one back edge. Note that while running DFS this back edge can be found before seeing $|V|$ edges. Therefore there are only $O(|V|)$ number of operations and the back edge will be found before that. Hence the running time will be $O(n)$. Therefore the algorithm will run in $O(|V|)$, independent of $|E|$

□