CSL 356: Assignment #1

Due on Sunday, August 25, 2013

S. Iyengar

Harsimran Singh

Contents

Problem 1

Give an algorithm to detect whether a given undirected graph contains a cycle. If the graph contains a cycle then your algorithm should output one.(It should not output all the cycles of the graph it should output just one of them). The running time of your algorithm should be *O(m+n)* for a graph with n nodes and m edges.

Solution

Depth first search can be used to check whether an undirected graph contains a cycle or not. Initially all the node are marked not visited. Start from a node, say x_i and mark it visited. Then call the DFS on children of this node recursively and make x_i (the node which called the DFS on them) their parent. While doing this if you encounter a visited node than you have found a cycle (because both nodes were already connected in DFS tree and now we have found an edge between them, hence a cycle). Now take these nodes and by backtracing their parent nodes we will get that cycle.
Because, we are performing DFS which takes *O(m+n)* to complete, our algorithm's time complexity is also *O(m+n)*.

Problem 2

Inspired by the example of that great Cornellian, Vladimir Nabokov, some of your friends have become amateur lepidopterists (they study butterflies). Often when they return from a trip with specimens of butterflies, it is very difficult for them to tell how many distinct species they've caught-thanks to the fact that many species look very similar to one another. One day they return with n butterflies, and they believe that each belongs to one of two different species, which we'll call A and B for purposes of this discussion. They'd like to divide the n specimens into two groups-those that belong to A and those that belong to B-but it's very hard for them to directly label anyone specimen. So they decide to adopt the following approach. For each pair of specimens i and j, they study them carefully side by side. If they're confident enough in their judgment, then they label the pair (i,j) either "same" (meaning they believe them both to come from the same species) or "different" (meaning they believe them to come from different species). They also have the option of rendering no judgment on a given pair, in which case we'll call the pair ambiguous. So now they have the collection of n specimens, as well as a collection of m judgements (either "same" or "different") for the pairs that were not declared to be ambiguous. They'd like to know if this data is consistent with the idea that each butterfly is from one of species A or B. So more concretely, we'll declare the m judgements to be consistent if it is possible to label each specimen either A or B in such a way that for each pair (i,j) labeled "same", it is the case that i and j have the same label; and for each pair (i,j) labeled "different", it is the case that i and j have different labels. They're in the middle of tediously working out whether their judgements are consistent, when one of them realizes that you probably have an algorithm that would answer this question right away. Give an algorithm with running time *O(m + n)* that determines whether the m judgements are consistent.

Problem 2 continued on next page...

Solution

Let G=(V,E) be a graph with all the butterflies as their vertices and judgements as edges between the vertices. Initially all vertices are white. Start with DFS on a node, say x_r and turn it grey. Now call DFS on children, say x_i, x_{i+1}, ..., x_{i+j} recursively. Check the label of all the pairs generated at each call (a parent and a child) (x_a, x_b). Let us say x_a is grey in color. Color x_b grey (same as x_a) if label is same, otherwise color x_b black (different from x_a).
We say two nodes are consistent, if there is an edge between them and
label is "same" ⇒ both are same in color and
label is "different" ⇒ both are different in color

Now, if during this algorithm we encounter a node x_n which is already colored. If node which called DFS on x_n, say x_m is consistent with x_n then we carry on otherwise we say there is inconsistency in their judgements.

Because we are using DFS, therefore order of our algorithm is *O(m+n).*

Problem 3

We have a connected graph *G = (V, E)* and a specific vertex U ∈ V. Suppose we compute a depth-first search tree rooted at u, and obtain a tree T that includes all nodes of G. Suppose we then compute a breadth-first search tree rooted at u, and obtain the same tree T. Prove that G = T. (In other words, if T is both a depth-first search tree and a breadth-first search tree rooted at u, then G cannot contain any edges that do not belong to T.)

Solution

Let us assume that ∃ an edge (s,t) in *G = (V,E)* which is not present in Tree T.
If s and t has an edge between them in the graph then either of s or t must be ancester of other in DFS tree T.
⇒ ∃ another path between s and t in graph that is included in tree T, because edge (s,t) is not in the tree. Let this path be s, x_1, x_2,...,x_j, t. Let L_s be the layer of s in tree T and similarly, L_t is the layer of t in tree T. Because ∃ at least one vertex in the path from s and t in DFS tree T ⇒ difference between L_s and L_t is at least 2.

Now while building BFS tree (which should be same as DFS tree T), because s and t has edge between them, so they will either belong to same layer L_s or two different layers L_s and L_t such that difference between layers is 1.
But in DFS tree T, L_s and L_t are at least two layers apart. ⇒ DFS tree T is not same as BFS tree, but we are given that both trees are same.
Hence, Contradiction.
Therefore, ∃ no edge (s,t) which is present in G but not in Tree T ⇒ G=T.

Problem 4

Some friends of yours work on wireless networks, and they're currently studying the properties of a network of n mobile devices. As the devices move around (actually, as their human owners move around), they define a graph at any point in time as follows: there is a node representing

Problem 4 continued on next page...

each of the n devices, and there is an edge between device i and device j if the physical locations of i and j are no more than 500 meters apart. (If so, we say that i and j are "in range" of each other.) They'd like it to be the case that the network of devices is connected at all times, and so they've constrained the motion of the devices to satisfy the following property: at all times, each device i is within 500 meters of at least n/2 of the other devices. (We'll assume n is an even number.) What they'd like to know is: Does this property by itself guarantee that the network will remain connected? Here's a concrete way to formulate the question as a claim about graphs.

Claim: Let G be a graph on n nodes, where n is an even number. If every node of G has degree at least n/2, then G is connected.

Decide whether you think the claim is true or false, and give a proof of either the claim or its negation.

Solution

Let us assume that graph G = (V,E) with n nodes (n is even) is not connected ⇒ ∃ two nodes s and t in the graph such that there is no path between them.
Given: s and t both have degree at least n/2.
Let say s has degree i and has edge with vertices V_s = { s_1, s_2,...,s_i} and t has degree j and has edge with vertices V_t = { t_1, t_2,...,t_j}. Both i and j are greater than n/2.
If there is no path between them ⇒ V_s and V_t are disjoint. This will result in number of nodes ≥ n+2
Number of nodes = (|{ s }| + |{ t }| + |{ V_s }| + |{ V_t }|
Number of nodes ≥ 1+1+n/2+n/2
Number of nodes ≥ n+2
But our graph G has only n nodes.
Hence contradiction.
Thus, our assumption is false.
Therefore, ∃ no s and t such that there is no path from s to t.
Hence, graph G is connected.

Problem 5

There's a natural intuition that two nodes that are far apart in a communication network-separated by many hops-have a more tenuous connection than two nodes that are close together. There are a number of algorithmic results that are based to some extent on different ways of making this notion precise. Here's one that involves the susceptibility of paths to the deletion of nodes.
Suppose that an n-node undirected graph G = (V, E) contains two nodes s and t such that the distance between s and t is strictly greater than n/2. Show that there must exist some node v, not equal to either s or t, such that deleting v from G destroys all s-t paths. (In other words, the graph obtained from G by deleting v contains no path from s to t.) Give an algorithm with running, time O(m + n) to find such a node v

Solution

Distance between s and t is strictly greater than n/2 $\Rightarrow$ there are at least n/2 nodes between s and t, let say a_1, a_2,..., a_m and m $\geq$ n/2.
If $\exists$ a vertex a_i which is not included in any cycle formation in the graph than this vertex can be removed to disconnect s from t, because every path from s to t must go through this vertex a_i.
Now, We will prove that such vertex always exists under given conditions.
Let say $\exists$ a path between two vertices a_i and a_j other than a_i, a_{i+1},...,a_j. We call this path a_i, x_1, x_2,..., x_d, a_j. Length of this path must be greater than the length of former path, otherwise it will reduce the distance between s and t. Therefore we need at least n/2 vertices to involve all a_i 's in the cycle, but we have less than (n/2)-2 vertices at our disposal. Therefore, there will always be a vertex a_i such that it is not included in any cycle formation.

Algorithm: Perform BFS on the given graph starting from s. There will be a layer in the BFS tree which will contain only one vertex.
Explanation: BFS Tree will have at least n/2 layers , because distance between s and t is atleast n/2. If we have atleast 2 vertices in each layer than it will result in nodes more than n.

This is the vertex we need to remove to disconnect s from t.
Explanation: There will be no edge that connects two components divided by this vertex because if there is an edge between two vertices then they are atmost 1 layer apart.

Because we are using BFS algorithm only, so time complexity of our algorithm is *O(m+n)*