Homework 2

CS 4104 (Spring 2021)

Assigned on February 8, 2021.
Submit a PDF file containing your solutions on Canvas by 11:59pm on February 15, 2021.

My team-mate is: Joseph McAlister

Problem 1 (25 points) This problem is in three parts.

1. (10 points) If G is an undirected graph and v is a node in G, let us use $G-\{v\}$ to denote the graph formed by deleting v and all the incident edges on v from G. In general, even if G is connected, it is possible that $G-\{v\}$ may contain two or more connected components, e.g., if G is a tree and v is a node with degree greater than one. However, this fragmentation cannot happen for every node v. Prove this fact, i.e., prove that in every connected, undirected graph G, there is at least one node v such that the graph $G-\{v\}$ is connected. *Hint:* In the special case that G is a tree, any leaf can serve as v. How will you find a "leaf" in a general undirected graph?

   Solution: For any undirected, connected graph G, there exists a sub-graph T which connects all of it's nodes without any loops called a spanning tree (which we can attain from a BFS).

   As mentioned in problem, we can delete any of T's leaf nodes (nodes with degree of 1) which will leave the rest of the graph connected.

2. (10 points) Now prove that if G is an undirected, connected graph on n nodes that G must contain at least $n-1$ edges. *Hint:* Use the statement in part (a) in combination with mathematical induction. Your proof must clearly state the quantity over which you are inducting (e.g., the number of nodes n or the number of edges m or something else that may be appropriate to the problem) and prove the base case, state the inductive hypothesis, and then prove the inductive step.

   Solution: Let n be the number of nodes on a connected graph G.

   Base case:

   For $n = 2$, the number of edges e must be at least 1 to connect the two nodes, so $e \geq 1$ which, for the base case, is the same as having at least $n-1$ edges.

   Inductive Hypothesis: Assume that for any connected graph G with n nodes, the number of edges is at least $n-1$.

   Inductive Step:

   Suppose a connected graph has $n+1$ nodes, then the number of edges must be at least $(n+1)-1$ which is at least n edges.

   So, for any connected graph G, the number of edges e is *at least* $n-1$. ■

3. (5 points) Finally, prove that if G is an undirected, connected graph on n nodes that contains exactly $n-1$ edges, then G does not contain a cycle. *Hint:* The statement you proved in part (b) may be helpful.

   Solution:

   Proof by contradiction: Assume that G is an undirected, connected graph on n nodes that contains exactly $n-1$ edges, and that G does contain a cycle.

   If a connected graph G has exactly $n-1$ edges, then it must be a tree.

   Trees by definition do not contain cycles, contradicting our assumption that G contains a cycle.

   Therefore an undirected, connected graph on n nodes that contains exactly $n-1$ edges does not contain a cycle. ■

Problem 2 (10 points) For each node u in an undirected graph $G = (V, E)$, let $q(u)$ be the sum of the squares of the degrees of u's neighbors, i.e.,

$$q(u) = \sum_{(u,v)\in E} (d(v))^2,$$

where $d(v)$ denotes the degree of node v. In this definition of $q(u)$, think of u as being fixed. Then the sum runs over all nodes v such that (u, v) is an edge in the graph. In other words, the sum runs over all neighbours of u. Describe an algorithm that computes the $q(u)$ value for every node u in G in linear time in the size of the graph. To be clear, the algorithm should take linear time over all the nodes in G and not linear time for each node in G.

Solution:

Input: the Adjacency List of some undirected graph G
Output: a list of $q(u)$ values for each node $u \in G$

```
1 q_values ← [0, ..., 0] // initialize an empty list of size |V| = n to store q values
  // for each node u ∈ G = n
2 foreach u ∈ n do
3 |   foreach v ∈ u do
  |   |   // for each node neighboring the current node
  |   |   // d(v) is the length of v's adjacency sub-list
4 |   |   q_values[u] += d(v)^2
5 |   end
6 end
7 return q_values
```

---

proof of correctness:

We begin by iterating over each node u in the graph of n nodes.

For each neighbor v of the current node u, we increment the resulting q_values entry corresponding to the current node u by the square of the degrees of its neighbor v. Since the adjacency list of the neighbor stores the indices or labels of the nodes that it is adjacent to, the length of this sub list is its degree.

By performing this doubly nested iteration for each node u in the graph of size n, we calculate the $q(u)$ value of each node in the graph. ■

proof of linear runtime complexity over the graph of size $m + n$:

The loop on line 2 iterates n times, once for each node in the graph.

The inner loop iterates m_i times, once for each edge connected to the current node u. The sum of all the m_i edges will be $2m$ since each node counts the edge twice, where m is the total number of edges in the graph G.

We can assume that all the other operations (initialization on line 1, summation on line 4, and return statement on line 7) are constant time, leaving us with a runtime complexity of $O(n+2m) = O(n+m)$ which is linear time over all the nodes in G. ■

Problem 3 (15 points) Given an undirected graph G and an edge e in this graph, develop a linear time algorithm to determine if there is a cycle in G that contains e.

Solution: Let e be the edge connecting two nodes (u, v). If we remove e from G, and perform a Breadth First Search starting at u, targeting v on $G - \{e\}$ which returns true (i.e. there is a path from $u \to v$), then that means that G has a cycle containing e since e connects the BFS path $u \to v$ back to u: $u \to v \to u$. Since BFS is linear $O(n + m)$ on a graph G our algorithm modified algorithm is also linear.

Input: Any representation of some undirected graph G, and an edge connecting two nodes $(u, v) \in G$
Output: A boolean value representing whether a cycle exists in G

```
1 G' ← G − {e}
  // Perform BFS on G' starting from u, targeting v
2 R ← BFS(G', u, v)
  // R is the set of all nodes to which u has a path
3 return v ∈ R
```

proof of correctness:

By performing a Breadth First Search on the modified graph G' using an optimal $O(m+n)$ implementation, we are guaranteed that R is the set of all nodes for which u has a path to reach, including u itself.

If v is in R, $then there is still a path from u \to v \in G'$.

Therefore, by restoring the edge $e = (u, v)$, we are guaranteed *another* path from $u \to v$, thus extending the existing path and making a cycle.

Similarly, if v is not in R, then the only path from $u \to v \in G$ is the edge $e = (u, v)$ which we initially removed. ■

proof of linear runtime complexity over the graph of size $m + n$:

If we take an optimal implementation of BFS on Graph structure which supports constant time removal of an edge in the graph, then the running time of the presented algorithm is the same as the runtime of BFS: $O(m+n)$. ■

Problem 4 (20 points) 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 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?

> *Claim: Let G be an undirected 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:

Suppose by way of contradiction that G is not connected.

Let H be the smallest connected portion of the disconnected graph G such that the degree of any node i in H is less than or equal to half of all the nodes in G:

$\deg(i) \leq \frac{n}{2}, \quad \forall i \in H$. This is the same as stating $|H| \leq \frac{n}{2}$.

Take some node $i \in H$ and examine all the incident nodes of i. Since they neighbor i, they must also be in H.

So, the degree of i must be less than the number of other nodes in H minus one to account for i itself: $\deg(i) \leq |H| - 1$.

Substituting our upper bound of $|H|$ for $\frac{n}{2}$, we get: $\deg(i) \leq \frac{n}{2} - 1$.

But the problem states that every node has at least $\deg(i) \geq \frac{n}{2}$, contradicting our prior statement.

Therefore the claim is true by proof of contradiction. ■

Problem 5 (30 points) Solve exercise 9 in Chapter 3 (page 110) of your textbook. Suppose that an n-node undirected graph G contains two nodes s and t such that the distance between s and t is strictly greater than $n/2$. Show that G must contain some node v, not equal to s or t, such that deleting v from the graph 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. *Hint:* At least one of the layers in the BFS tree rooted at s has a special property.

Solution: In order for the claim to be true, there must be some layer L_i that contains only one node such that by removing it, there is no longer a path from $s \to t$.

If we run a Breadth First Search starting from s, targeting t then let the layer at which we find t be L_d, with $d > \frac{n}{2}$ according to the problem statement.

In the layers $L_1, \ldots, L_{d-1}$ between L_0 containing s and L_d containing t, we must have the L_i bottleneck containing only one node v.

We know this because if all the layers $L_1, \ldots, L_{d-1}$ had at least 2 nodes in them, the we would have $2(\frac{n}{2}) = n$ nodes, but the graph contains only n nodes. Therefore, we know that there has to be some bottleneck layer L_i with a single node v.

If we delete this node $v \in L_i$, then we remove all $s \to t$ paths where $d(s,t) > \frac{n}{2}$.

Input: A graph G, and two nodes $s, t \in G, d(s,t) > n/2$
Output: A node $v \in G$ where there is no path from $s \to t \in G - \{v\}$

```
// BFS' is a modified BFS which returns a set of tuples of the form (L_i, u, w) of
   layer distance and adjacencies from the root node s representing the BFS tree.
1 T ← BFS'(G, s, t)
// A two dimensional list to keep track of which nodes are in each layer
2 layers ← []
// Sort the set by layer index, storing only the nodes comprising the connection
3 foreach tuple (L_i, u, w) ∈ T do
4 |   layers[L_i].append((u, w))
5 end
6 foreach layer ∈ layers do
7 |   if len(layer) == 1 then
  |   |   // The layer containing only one entry is the bottleneck
8 |   |   L_i ← layer
  |   |   // Of the (w, v) edge tuple in L_i the second entry is the node of the
  |   |      bottleneck
9 |   |   v ← L_i[1]
10|   |   return v
11|   end
12 end
13
```

---

proof of correctness:

The presented algorithm relies on the modification of the BFS search shown in class which keeps track of the distance from the root node s. This additional piece of information allows for finding a bottleneck.

From the problem statement, we are guaranteed that s and t are connected since the distance $d(s,t) > n/2$. This means that the set T returned on line 1 contains all paths from $s \to t$.

As noted above, we know that there must exist some layer between $L_1, \ldots, L_{d-1}$ which contains only one node from the stipulation that the distance is greater than $n/2$ and otherwise there would be $n+2$ nodes in the graph, contradicting its definition of having n nodes.

Therefore, by sorting each tuple in T into sub lists indexed by their layer index L_i on lines 3-5, we can iterate over those lists, checking how many (w, v) edge tuples are contained in that layer's list on line 6.

We are guaranteed to find our layer which contains only one edge tuple, and the second node in the tuple is the desired v. ■

proof of linear runtime complexity over the graph of size $m + n$:

The queue BFS implementation from the lecture notes, modified to also store the layer distance L_i in the resulting tree T is linear over the size of the graph with a total time of $O(n + m)$.

The first pass over the resultant set T on lines 3-5 iterates over every node in the graph a single time, which is $O(n)$ time.

Then, the second pass over the "sorted" layers on line 6 iterates over each of the layers which, in the worst case of a "line graph" where each node has a degree of strictly less than 3, this pass would additionally take $O(n)$ time.

The remaining list append, assignment, and return statements on lines 4, 8, 9, 10 are constant time operations, so the sum of each of the iterative portions of the algorithm is $O(n + m) + O(n) + O(n) = O(3n + m)O(n + m)$, which satisfies the requirement. ■