CS310: Homework 8

Scott Fenton

May 2, 2017

Exercise (1). Let d be the maximum degree of the vertices in a graph G. Prove that we can color G with d + 1 colors. Hint: by induction.

Solution (1). This problem can be solved by induction:

Base Case:
While $n = 1$
P(1) is true
A 1-vertex graph has maximum degree 0, and is 1-colorable, so P(1) is true.

Inductive Step:
1. We assume that $P(n)$ is true, and let G be an (n+1) vertex graph with maximum degree at most k.

2. Remove a vertex v and all edges incident to i., leaving an n-vertex subgraph, W. The maximum degree of W is at most k, and so W is (k+1) colorable by our assumption $P(n)$.

3. Now add back vertex v. We can assign v a color from the set of k + 1 colors that is different from all its adjacent vertices, since there are at most k vertices adjacent to v and so at least one of the k + 1 colors is still available.

Conclusion:
Therefore, G is (d + 1) colorable. The above follows by induction.

Exercise (2). Consider this statement: In any directed graph G = (V,E), when DFS visits a vertex u ∈ V, then every undiscovered vertex v such that u has a path to v must be discovered before DFS returns from u. Is this statement true or false? If true, give a proof; if false, give a counterexample.

Solution (2). Consider a directed graph G = (V, E). After a DFS of graph G we can put each edge into one of three identities:

1. A tree edge is an edge in a DFS-tree.
2. A back edge connects a vertex to an ancestor in a DFS-tree.
3. A forward edge is a non-tree edge that connects a vertex to a descendent in a DFS-tree.

proof:
An Edge (u, v) is a back edge if and only if $d[v] < d[u] < f[u] < f[v]$.

(Forward(=>) direction) From the definition of a back edge, it connects vertex u to an ancestor vertex v in a DFS-tree. Hence, vertex u is a descendent of vertex v. In number (3) above it states that vertex u is a proper descendent of vertex v if and only if $d[v] < d[u] < f[u] < f[v]$. Thus we have proved forward direction.

(Backward(<=) direction) We can see from above (2) that a, vertex u is a proper descendent of vertex

v. if an edge (u, v) exists from u to v then it is an edge connecting a descendent vertex u to its ancestor vertex v. thus, it is a back edge, and we can prove backward direction.

Conclusion:
As shown above we can prove both forward and backward direction, meaning that each undiscoverted vertex v such that u has a path to v must be discovered before DFS returns from u.

Exercise (3). Consider this statement: In any undirected graph G = (V,E), there must be an even number of vertices whose degree is odd. Is this statement true or false? If true, give a proof; if false, give a counterexample.

Solution (3). We can solve this using a Direct Proof:

Let d_v denote the degree of vertex v(so $d_v = N_v$ where N_v is the set of neighbors of v). We see

$$\sum_{v \in V} d_v = 2m$$

that the sum of d_v is 2m. Because every edge is counted exactly twice when we sum the degrees of all the vertices. Now partition V into the odd degree vertices V_{odd}(G) and the even degree vertices $V_{odd}(G)^C$, we can write the sum of d_v minus the non off vertices.

$$\sum_{v \in V_{odd}(G)} d_v = 2m - \sum_{v \in V} d_v$$

Both terms in the righthand side above are even (2m is even, and each term d_v is even because we are summing over even degree vertices V not memeber V_{odd}). So for the lefthand side to be even we must have an even number of terms. since each term in the summation is odd. Therefore, there must be an even number of off-degree vertices, namely, $|V_{odd}|$ is even.

Exercise (4). Consider the problem of determining whether an undirected graph G = (V,E) contains a triangle (cycle of length three).

(a) Give an $O(|V|^3)$ to find a triangle if one exists.
(b) Improve your algorithm to run in time $O(|V||E|)$. You may assume $|V|$ is less than or equal to $|E|$.

Solution (4a). A naive solution would be to convert the adjacency list to a matrix. Then Loop through the matrix to determine if a triangle exists. Below is the psudo-code for the brute force algorithm.

```
for (int i = 0; i < n; i++)
     for(int j = i + 1; j < n; j++)
     if (matrix[i][j] == 1)
         for(int k = j + 1; k < n; k++)
                 if (matrix[i][k] == 1 && matrix[j][k] == 1)
                        return true;
return false;
```

Solution (4b). The $O(|V| * |E|)$ solution is similar to the brute-force in (a) but it checks all edges, and not all vertices pairs. Knowing the edges is enough to know if the edge and vertex form a feasible triangle.

```
for each edge (u, v):
     for each vertex w:
     if (v, w) is an edge and (w, u) is an edge:
         return true
return false
```

Exercise (5). You are given a weighted graph and its minimum spanning tree. There are n vertices and m edges in the graph.

(a) Design a polynomial-time algorithm that finds the smallest change in the weight of a non-MST edge that would cause a change in the MST.
(b) Analyze the runtime of your algorithm in terms of n and m.

Solution (5a). There are four steps to this algorithm:
1. Traverse the edge of the graph, passing the edge of the MST when it encounters the non-MST edge E(i, j)

2. Use the parent in MST to traverse all the edges of vertex i to j and get the edge of the maximum weight MST_{max}

3. Calculate the difference between E (i, j) and MST_{max}.

4. Traversing all non-MST edges, getting the minimum difference, which is the minimum value of the change.

In conclusion, the total number of edges m, algorithm complexity O(m^2).

Solution (5b). In terms of n and m.