COS 226 – Data Structures and Algorithms
Fall 2014 – Flipped Lecture Section
Group Worksheet week 7 – 10.23.14
30 minutes

Problem #1: A connected graph is one in which every vertex is reachable from every other vertex. Prove that every connected undirected graph has a vertex whose removal (including all adjacent edges) will not disconnect the graph. Design an algorithm to find a vertex whose removal will not disconnect the graph. You may find it easier to construct a proof around an algorithm.

Solution: Suppose we run DFS from some starting vertex v and let w be the first vertex that finishes. This vertex is obviously a dead end, and thus must either be a leaf, or it must only point back at some previously visited node. Thus, we can remove this vertex and the rest of the graph will remain connected.

Problem #2: DFS – from recursion to iteration

We implemented DFS using the simple recursive routine shown below (this version of DFS stores paths unlike the simpler version).

```
private void dfs(Graph G, int v) {
    marked[v] = true;
    for (int w : G.adj(v)) {
        if (!marked[w]) {
            edgeTo[w] = v;
            dfs(G, w);
        }
    }
}
```

Rewrite this version w/o using recursion.

```
private void dfs_iterative(Graph G, int v) {
  marked[v] = true;
  Stack S = new Stack();
  while (!S.empty()) {
    int v = S.peek();
    boolean allmarked = true;
    for (int w: G.adj(v)) {
      if (!marked[w]) {
        marked[w] = true;
        edgeTo[w] = v;
        allmarked = false;
        break;  /* from for loop */
      }
    }
    if (allmarked)  S.pop();
  }
```

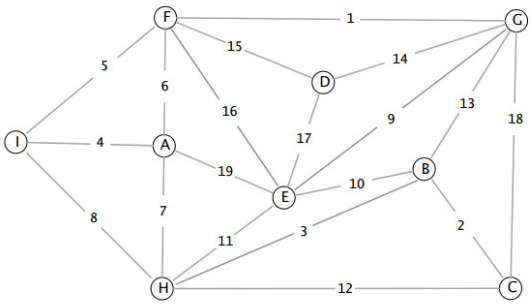
Problem #3: Odd length paths

Given a directed graph and a starting vertex u, give an algorithm for finding all vertices such that there is an odd-length path to those vertices from u. These paths may involve cycles. Your algorithm should complete in $O(E + V)$ time. If you're stuck, you might also try to come up with an algorithm that completes in $O(EV)$ time (which might be a little easier?)

Run BFS, but maintain an evenQueue and an oddQueue and give each vertex an even and odd marker. We start by marking the source as even and enqueuing it on the evenQueue. From there, we dequeue each vertex v from the evenQueue (only the source on the first iteration); then for each neighbor w of v, we then mark each w as odd and enqueue it on the oddQueue, unless w has already been marked as odd (in which case we ignore it). We next do the same thing with the oddQueue, dequeuing each vertex v, and marking and enqueuing each neighbor w as long as it is not even. We alternate between the two queues until both are empty.

Problem 4: Graph Problems

Consider the following weighted graph.



(a) Starting with vertex I, find the DFS sequence, if the heuristic "the smaller weight" is used to decide which node to process next.

I ➔ A ➔ F ➔ G ➔ E ➔ B ➔ H ➔ C ➔ G

(b) Device an algorithm, that finds the number of hops (minimum) from vertex 1 to any other connected vertex.

do BFS on the graph starting from vertex I. Mark I as 0. For all vertices that can be reached from I mark them as 1. Continue to perform the BFS by marking new vertices with sum of (distance to parent + 1) unless there is a smaller sum already.

(c) Think about a possible algorithm, where this idea can be extended to finding the shortest path from vertex 1 to any other vertex (we will discuss this in detail soon)

apply DFS, this time, marking nodes with the distance. Use the heuristic "shorter distance" to decide which node to go to next. Continue until all nodes are exhausted and distance adjusted (if a new min is found through another node, adjust it)