Solutions to Homework Two

CSE 101

1. *Abstraction using graphs.*

   (a) Create graph $G = (V, E)$ where

   $$V = \{\text{all airports in US}\}$$
   $$E = \{\{u, v\} : \text{there are direct flights between } u \text{ and } v\}$$

   Let z denote San Diego.
   The problem we want to solve is FURTHEST NODE: given a graph $G = (V, E)$ and a specific node $z \in V$, find the node in V furthest from z (that requires the maximum number of edges to reach).

   (a) Create a graph with

   $$V = \{1, 2, \ldots, n\} \quad \text{(possible locations)}$$
   $$E = \{\{i, j\} : d_{ij} \leq D\}$$

   The problem we need to solve is DOMINATING SET: given a graph $G = (V, E)$, find the size of the smallest subset $S \subseteq V$ such that every node is either in S or has a neighbor in S.

   (c) Here the graph has one node for each possible configuration of the Rubik's cube and there is an edge between any two neighboring configurations (that can be reached in a single move). The problem we want to solve is once again FURTHEST NODE, where z is the solved configuration.

   (d) Create a graph with a node for each person and an edge between any two people who know each other. The problem to be solved is again DOMINATING SET.

2. *Finding the sources of a directed graph.*

   We will keep an array `in`$[u]$ which holds the indegree (number of incoming edges) of each node. For a source, this value is zero.

```
function sources(G)
Input:  Directed graph G = (V, E)
Output:  A list of G's source nodes
for all nodes u:   in[u] = 0
for all nodes u:
   for all neighbors w of u:
      in[w] = in[w] + 1

L = empty linked list
for all nodes u:
   if in[u] is 0:  add u to L
return L
```

   The running time is $O(|V|)$ for the first loop, $O(|E|)$ for the second loop, and $O(|V|)$ for the third loop, for a grand total of $O(|V| + |E|)$.

3. **Claim.** Any connected undirected graph with n nodes has at least $n - 1$ edges.

   Proof. Let's use induction on n. The base case is $n = 1$, and follows immediately.

   Now suppose the statement is true for graphs of up to $n-1$ nodes. Take any connected graph $G = (V, E)$ with n nodes. Let's consider two cases.

   Case 1: If every node has degree ≥ 2, then by the formula

   $$\sum_{u \in V} \text{degree}(u) = 2|E|,$$

we see that $|E| \geq n$, and we're done.

Case 2: Otherwise there must be at least one node of degree 1. Pick any such node, and call it u. Now remove u and its adjacent edge from the graph. This leaves a connected graph, call it $G_{\setminus u}$, with $n-1$ nodes. By the inductive hypothesis, $G_{\setminus u}$ has at least $n-2$ edges. Since we removed one edge from G, it follows that G has at least $n-1$ edges.

4. *Textbook problem 3.5.* Computing the reverse G^R of a directed graph G:

```
create G^R = (V, E^R), with edge-set E^R initially empty
for each node u in V:
   for each neighbor w of u in E:
      add edge (w, u) to E^R
```

For each edge $(u, w) \in E$, adding (w, u) to E^R takes $O(1)$ time: the edge is simply added to the front of the adjacency list for node w. Thus the total running time is $O(|V| + |E|)$, linear.

5. *Textbook problem 3.9.* First, the degree of each node can be determined by going through the adjacency list. The array `twodegree` can then be set in a second pass through the adjacency list.

```
for all nodes u:
   degree[u] = 0
   for all neighbors w of u:
      degree[u] = degree[u] + 1
for all nodes u:
   twodegree[u] = 0
   for all neighbors w of u:
      twodegree[u] = twodegree[u] + degree[w]
```

This algorithm directly implements the definition of `degree` and `twodegree`. Its running time is $O(|V|)$ for the initializations, plus the time for the two inner loops; in each inner loop, over the entire execution, a single pass is made through the adjacency list. Thus the total running time is $O(|V| + |E|)$, linear.

6. *Textbook problem 3.16.* Create a directed graph $G = (V, E)$ with a node for each course and an edge (u, v) whenever course u is a prerequisite for course v. The courses we can take in the first semester are precisely the sources of this graph; we might as well take them all. We can then remove them from the graph, and in the subsequent semester, take the sources of the remaining graph; and so on.

```
Compute indegrees in[·] and let currL be the list of source nodes (Problem 1)

time = 0
repeat until currL is empty:
  time = time + 1
  nextL = empty list
  for all u in currL:
    for all neighbors w of u in E:
      in[w] = in[w] - 1
      if in[w] is 0:
        add w to nextL
  currL = nextL
Output time
```

The first part is the linear time indegree computation and source identification of Problem 1. Then the main loop begins, with the integer variable `time` containing the current semester number and `currL` containing the current list of courses. When these courses are removed from the graph, the indegrees of

the nodes they point to are reduced; the main body of the loop performs the corresponding reductions and identifies the next set of source nodes, `nextL`.

Each course u is on `currL` during exactly one semester; at that point, its indegree is already zero and thus we never again consider edges into u. While u is on the the current list, all the edges out of it are examined. Thus in the repeat loop, over the entire execution of the algorithm, all edges in the graph are examined exactly once. Therefore, the total time taken by the loop is $O(|V| + |E|)$.