CS106X
Autumn 2019

Handout 21S
October 16th, 2019

CS106X Practice Midterm Solution

Solution 1: Word Ladders, Take II

a.
```
static Vector<string> reconstruct(const string& start, const string& finish,
                                  const Map<string, string>& predecessors) {
   Stack<string> inverted; // could have been a Vector as well
   string rung = finish;

   while (true) {
      inverted.push(rung);
      if (rung == start) break;
      rung = predecessors[rung];
   }

   Vector<string> ladder;
   while (!inverted.isEmpty()) {
      ladder += inverted.pop();
   }
   return ladder;
}
```

b.
```
 static Vector<string> generateShortestWordLadder(const string& start,
                                                  const string& finish,
                                                  const Lexicon& english) {
   Map<string, string> predecessors;
   Queue<string> queue;
   queue.enqueue(start);
   predecessors[start] = "";
   while (!queue.isEmpty()) {
      string endpoint = queue.dequeue();
      if (endpoint == finish)
         return reconstruct(start, finish, predecessors);
      Vector<string> neighbors = generateAllNeighbors(endpoint, english);
      for (const string& neighbor: neighbors) {
         if (!predecessors.containsKey(neighbor)) {
            predecessors[neighbor] = endpoint;
            queue.enqueue(neighbor);
         }
      }
   }

   return Vector<string>();
 }
```

Solution 2: Autocorrect

```
static void ls(const string& prefix, const string& suffix,
               const Lexicon& english, const Map<char, string>& map, int max) {
   if (max < 0 || !english.containsPrefix(prefix)) return;
   if (suffix.empty()) {
      if (english.contains(prefix)) cout << prefix << endl;
      return;
   }

   string rest = suffix.substr(1);
   ls(prefix + suffix[0], rest, english, map, max);
   for (char ch: map[suffix[0]]) {
      ls(prefix + ch, rest, english, map, max - 1);
   }
}

static void ls(const string& str, const Lexicon& english,
               const Map<char, string>& alternatives, int maxChanges) {
   ls("", str, english, alternatives, maxChanges);
}
```

Solution 3: Valency

```
static bool findCircle(const Grid<int>& valencies, int& row, int& col) {
   for (row = 0; row < valencies.numRows(); row++) {
      for (col = 0; col < valencies.numCols(); col++) {
         if (valencies[row][col] > 0) return true;
      }
   }
   return false;
}

static bool solve(const Grid<int>& originals, Grid<int>& valencies,
                  Map<connection, int>& connections) {
   int row, col;
   if (!findCircle(valencies, row, col)) return true;

   coord location = {row, col};
   Set<coord> candidates = getCandidates(location, originals);
   valencies[row][col]--;
   for (const coord& candidate: candidates) {
      if (valencies[candidate.row][candidate.col] == 0) continue;
      valencies[candidate.row][candidate.col]--;
      if (solve(originals, valencies, connections)) {
         connection conn = { location, candidate };
         connections[conn]++;
         return true;
      }
      valencies[candidate.row][candidate.col]++;
   }
   valencies[row][col]++;
   return false;
}

static bool solve(Grid<int>& valencies, Map<connection, int>& connections) {
    Grid<int> originals = valencies;
    return solve(originals, valencies, connections);
}
```