Section #7

Ashley Taylor and Shreya Shankar (Based on handouts by previous CS106B instructors and TAs)

Graphs: Solutions

1. Graph Properties

1. Directed
2. Weighted
3. Connected
4. Cyclic
5. A: (1 2) B: (2 2) C: (0 2) E: (1 2) F: (4 0) G: (1 2) H: (1 1) I: (1 2) J: (2 0)
6. A, B, F, E, H, I
7. {A, E, H, I}
8. B, G, A, F, J, E, H
9. {C, B, A, E, H}

2. Task Orderings

Note that there may be more than one valid ordering for each list. The following are example solutions.

1. `[ C, D, B, A ]`
2. `[ 4, 5, 2, 3, 1, 0 ]`
3. `[ A, B, F, C, D, E ]`

3. kth Level Friends

```
Set<string> kthLevelFriends(Map<string, Set<string>>& adjacencyList, string v, int k) {
    Set<string> result;
    HashMap<string, int> distances;
    Queue<string> q;

    // initialize BFS
    string curr = v;
    q.enqueue(curr);
    distances[curr] = 0;

    while(!q.isEmpty()) {
        curr = q.dequeue();
        if (distances[curr] == k) {
            result.add(curr);
        }
        if (distances[curr] > k) {
```

```
            break;
        }

        for (string buddy : adjacencyList[curr]) {
            if (!distances.containsKey(buddy)) {
                distances[buddy] = distances[curr] + 1;
                q.enqueue(buddy);
            }
        }
    }
    return result;
}
```

4. Minimum Vertex Cover

```
void coverHelper(Map<string, Set<string>>& graph, Set<string>& allEdges, Set<string>& chosen,
        Set<Edge>& coveredEdges, Vector<string>& allVertexes, int index, Set<string>& best) {
    if (chosen.size() >= best.size()) return; // base case: current cover too large
    else if (coveredEdges.size() == allEdges.size()) {
        // base case: found a new smaller cover that uses all edges; remember it
        best = chosen; return;
    } else if (index == graph.keys().size()) return; // base case: exhausted all vertices to explore
    else {
        // choose not to include this vertex; explore
        coverHelper(graph, allEdges, chosen, coveredEdges, allVertexes, index + 1, best);

        // choose to include this vertex; explore
        chosen += allVertexes[index];

        // remember which new edges are added here (so that we can un-choose later)
        Set<Edge> newEdges;
        for (string neighbor : graph[allVertexes[index]]) {
            Edge e = new Edge(allVertexes[index], neighbor);
            if (!coveredEdges.contains(e)) {
                // must add this edge and its inverse (A -> B and B -> A)
                Edge inverse = new Edge(neighbor, allVertexes[index]);
                newEdges += e, inverse;
                coveredEdges += e, inverse;
            }
        }
        coverHelper(graph, allEdges, chosen, coveredEdges, allVertexes, index + 1, best);

        // unchoose
        chosen -= allVertexes[index]; coveredEdges -= newEdges;
    }
}

Set<string> findMinimumVertexCover(Map<string, Set<string>>& graph, Set<string>& allEdges) {
    Set<string> best = graph.keys();   // worst case solution
    Set<string> chosen; Set<Edge*> coveredEdges; Vector<Vertex*> allVertexes;
    for (Vertex* v : graph.keys()) {
        allVertexes += v;
    }
    coverHelper(graph, allEdges, chosen, coveredEdges, allVertexes, 0, best);
    return best;
}
```