Assignment 1

Siwei Tang (siweitang) `sw.tang@mail.utoronto.ca`

Changyan Xu (xuchangy) `changyan.xu@mail.utoronto.ca`

April 15, 2021

Reference

- `https://en.wikipedia.org/wiki/Adjacency_list`
- `https://www.geeksforgeeks.org/graph-and-its-representations/`

Question 2

Let n be the number of the vertices in the graph. Since for an **adjacency list**, an array of lists is used, where the outer array is generally an arraylist of length n, and the inner list in generally either an arraylist of fixed length with a sentinel or a linked list.

(a) Suppose we are given the adjacency list representation of an undirected graph with $2n$ nodes. We design an algorithm with $\mathcal{O}(n)$ runtime to determine if the given graph is a sun.

1. Looping over the adjacency list, check for each vertex if it has exactly 1 or 3 neighbouring vertex/vertices. At the meantime, record the number of vertices that has exactly 1 neighbouring vertex (i.e. `countOneNeighbour`) and the number of vertices that has exactly 3 neighbouring vertices (i.e. `countThreeNeighbours`). If any of the following cases occurs:
   - any vertex does not have either 1 or 3 neighbouring vertex/vertices
   - `countOneNeighbour` != n
   - `countThreeNeighbours` != n

   , we can terminate the algorithm and declare that the graph is not a sun. Otherwise, we can continue to the next step.
2. Direct access index 0 of the adjacency list, and obtain the corresponding vertex, v_0. Implement DFS on the given graph starting from v_0. (The depth of v_0 is 0.)

   At the meantime, we record/track the following during DFS:
   - total_num_nodes: total number of nodes visited
   - max_depth: maximum depth the DFS can reach
   - node of depth 0
   - node of depth 1

   If v_0 has only 1 neighbour vertex, we expect that the DFS would obtain a maximum depth of $n+2$ and it should have a chance to form a cycle at depth of $n+1$ with the node at the depth of 1. If v_0 has exactly 3 neighbour vertices, we expect that the DFS would obtain a maximum depth of $n+1$ and it should have a chance to form a cycle at depth of n with the node at the depth of 0. Additionally, we expect that the total_num_nodes equals to $2n$. Otherwise, we would terminate the algorithm and declare that the graph is not a sun.

(b) Justification of correctness:

A sun of $2n$ node has the following features:

1. n nodes forming a cycle
2. the rest n node are the out-reached nodes of each nodes in the cycle
3. n nodes have 3 neighbours
4. n nodes have only 1 neighbour

The step 1 (i.e. looping over the outer array of the adjacency list) checks if the feature 3 and 4 are abiding by.

The step 2 (i.e. DFS) checks if a cycle of n nodes exists in the graph which is in accordance with feature 1. Examining the `max_depth` is actually able to test if the counter-example may exist when disjoint subgraphs (including suns) occur in the given graph.

Among the both steps, any counter examples of feature 2 would be rule out. Because if n distinct nodes can form a cyclic graph and there is no disjoint subgraphs in the given graph, plus satisfying feature 3 and 4, there is no way that any other n nodes are not connected where each to a cycle vertex.

(c) Justification of $\mathcal{O}(n)$ runtime:

The worst case scenario is the given graph is a sun. Otherwise, the program should terminates earlier while it is still in the middle of the algorithm.

- Step 1: Looping over the adjacency list requires $2n$ times of accessing each vertex. Counting the number of neighbouring vertices in a linked list or an array corresponding to each vertex shall need 1 or 3 times of access. Other comparisons and recording along the looping, we can regard as constant time costs. In total, step 1 needs at most $4n$ course of actions $\in \mathcal{O}(n)$.
- Step 2: The direct access to the index 0 of the adjacency list takes constant time. For every visit to the node during DFS, it takes constant time to update initialized variables or record anythin. The implementation of DFS would take at most $O(\text{number of edges} + \text{number of vertices}) = \mathcal{O}(2n + 2n) = \mathcal{O}(n)$ runtime (where number of edges $= 2n$ for a legitimate sun with number of vertices $= 2n$), since every node is visited at most once.
- Thereby, the algorithm takes at most a running time of $\mathcal{O}(n)$.