CS2040S: Data Structures and Algorithms

Tutorial Problems for Week 10 Monday: Graphs and Traversal 1

For: 17 October 2022, Tutorial 8

Problem 1. Bipartite Graph Detection

A *bipartite graph* is a graph whose vertices can be divided into two disjoint sets U and V such that every edge connects a vertex in U to one in V; but there is no edge between vertices in U and also no edge between vertices in V.

Given an undirected graph with n vertices and m edges, we wish to check if it is bipartite.

Describe the most efficient algorithm you can think of to check whether a graph is bipartite. What is the running time of your algorithm?

run a DFS, to colour the vertices w alternating colours. if theres a neighbour with colour and that colour is same, it is not bipartite else if we can assign colour to every vertex, it is bipartite
O(n + m)

Problem 2. Cycle Detection

Given a graph with n vertices and m edges, we wish to check if the graph contains a cycle.

Problem 2.a. First, consider the case of an undirected graph. Describe an algorithm to check if the graph contains a cycle.

if no cycle, it is must be a forest -> set of connected components where every component is a tree. property of trees: |E| = |V| - 1
for each tree component, do a dfs to count no of vertices, and check the property

Both algo run in O(n + m) time

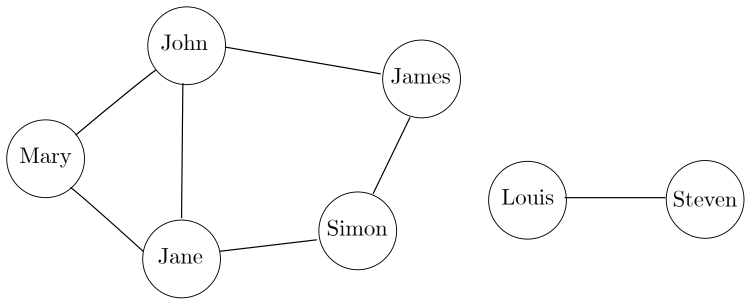
another method: modify DFS to check if a vertex is visited again(and not parent) if yes, theres a cycle

Problem 2.b. Next, consider the case of a directed graph (with no bi-directed edges). Describe an algorithm to check if the graph contains a cycle.

directed graph with no cycles: Directed Acyclic Graph(DAG) has topological ordering -> so if theres no such ordering, it has a cycle
use any topo sort algo to check for valid ordering

Problem 3. Friends Network (CS2010 AY17/18 Sem 1 Final Exam)

Peter is a very friendly person who has a lot of friends. In fact, he can construct a graph with n vertices and m edges, where the vertices represent his friends and the edges represent friends who know each other directly. An example is given below.



First, Peter wants to find out if a given pair of friends X and Y know each other directly (e.g John and Jane in the example).

Problem 3.a. What is the most appropriate graph data structure to store his friends graph in this scenario?

adj matrix

Problem 3.b. How would he answer his query using the graph data structure you have proposed in Problem 3a?

check if row x, col y of adj matrix is set to true -> O(1)

Next, Peter wants to know if a given pair of friends X and Y are related to each other indirectly, that is, there is no edge from X to Y but there is at least one path from X to Y with more than 1 edge (e.g. Mary is related indirectly to Simon through Jane among other possibilities in the example).

Problem 3.c. What is the most appropriate graph data structure to store his friends graph in this scenario?

adj list cause we wanna perform graph traversal and it works best on adj list

Problem 3.d. Describe an algorithm that answers his query using the graph data structure you have proposed in Problem 3c. What is the running time of your algorithm?

Finally, Peter wants to answer k queries of whether two given friends X and Y are related to each other *indirectly*.

Problem 3.e. Describe the most efficient algorithm you can think of to answer the k queries. What is the running time of *each* query?

3.d.

- Scan through the neighbours of X in the adjacency list and check if Y is connected directly to X. If so, we report that X is not indirectly connected to Y. This scan takes O(n) time, since X can be connected to at most n - 1 other vertices.

- Otherwise, we perform a DFS starting from X to check if Y is in the same connected component as X. This takes O(n + m) time. Overall, our algorithm takes O(n + m) time.

3.e.

By extending the idea in 3d, we perform a DFS on the adjacency list to label each connected component in the graph. Each vertex is labeled with the component number of the component it belongs to, and we can store the component numbers in an array.

This preprocessing step takes O(n + m) time.

To answer each query of whether two given friends X and Y are indirectly related, we first check if X and Y are directly related in O(1) time using the adjacency matrix as done in 3b. If they are directly related, then they cannot be indirectly related.

Otherwise, we can check if the component numbers of X and Y are the same in O(1) time. If their component numbers are the same, they are in the same connected component and thus indirectly related, else they are not related at all.

Thus, each query will take O(1) time.