=============================================================================

CSC 263 Tutorial 9 Winter 2019

=============================================================================

Question 1: Bipartite Graph

An undirected graph G = (V,E) is called "bipartite" when the vertices can be

partitioned into two subsets V = V\_1 uion V\_2 (with V\_1 intersects V\_2 = {}) such that every edge of G has one endpoint in V\_1 and the other in V\_2 (equivalently, no edge of G has both endpoints in V\_1 or both endpoints in V\_2).

1. Draw two examples of undirected graphs that \_are\_ bipartite, one with at

least 7 vertices and 10 edges.

answer:

Example 1:  
        1: 2  
        2: 1  
  
    Example 2:  
        1: 2,4,6  
        2: 1,5,7  
        3: 4,6  
        4: 1,3,5,7  
        5: 2,4,6  
        6: 1,3,5  
        7: 2,4

2. Draw two examples of undirected graphs that are \_not\_ bipartite, one with

at least 7 vertices and no "triangle" (three vertices with all the edges

between them).

anwer:

Example 1:  
        1: 2,3  
        2: 1,3  
        3: 1,2  
(odd number of edge cycle, not bypartite)

    Example 2:  
        1: 2,5  
        2: 1,3  
        3: 2,4  
        4: 3,5  
        5: 1,4  
        6: 7  
        7: 6

（4，5 are in the same group, violates the property）

3. Modify the Depth-First Search algorithm so that:

(a) your algorithm runs on an undirected graph;

(b) your algorithm returns True **if G is bipartite**, False otherwise;

(c) when G is bipartite, your algorithm sets a new field side[v] equal to

1 or 2 for every vertex v, indicating which side of the bipartition v

belongs to -- side[v] could have any value when G is not bipartite.

Argue that your algorithm is correct and analyse its running time.

answer:

Idea: Instead of colour, assign each vertex a 'side' equal to 1 or 2  
 (initially set every vertex's side to 0). Each edge encountered should  
    have endpoints with different sides: set undiscovered vertices' side  
    accordingly; return False if any conflict is discovered.  
  
        BIPART(G=(V,E)):  
            # Precondition: G is an undirected graph (connected or not).  
            # Postcondition: return True if G is bipartite (and set  
            #   side[v] = 1 or 2 for every v in V, representing the  
            #   bipartition); return False otherwise.  
            for each v in V:  
                side[v] <- 0  
            for each v in V:#每个v都call一次BYPART-REC()看v有没有连到同group的  
                if side[v] = 0:  
                    side[v] <- 1  # arbitrarily pick a side for v  
                    if not BIPART-REC(G,v):  
                        return False  
            return True  
  
        BIPART-REC(G=(V,E),v):  
            # Precondition: side[v] = 1 or 2; v has not been examined yet.  
            # Postcondition: return True if the connected component  
            #   containing v is bipartite (and set sides for each vertex  
            #   encountered); return False otherwise.  
            for each (v,u) in E:#check v连着的所有edge是不是都跟v是相反group的  
                if side[u] = 0:  
                    side[u] <- 3 - side[v]  # opposite side from v  
                    if not BIPART-REC(G,u):#再recursion看u连着的每个edge  
                        return False  
                else if side[u] = side[v]:  # conflict  
                    return False  
            return True  
  
    Runtime? Exactly the same as DFS: BIPART-REC(G,v) called exactly once for  
    each vertex v in V, examines adjacency list for v exactly once, so total  
    Theta(n+m) over all calls to BIPART-REC.  
  
    Correctness? Every edge encountered cannot have both endpoints on the  
    same side: the algorithm assigns sides to endpoints of edges to try to  
    ensure this holds; it returns False if it finds an edge whose endpoints  
    are on the same side (meaning there is a sequence of edges forcing them  
    to be on the same side). If the algorithm returns True, it was able to  
    assign sides to each vertex so that no edge has both endpoints on the  
    same side.

\*4. Prove that an undirected graph is bipartite iff it contains no cycle whose length is odd (called simply an "odd cycle").所存在even number cycle。

answer:

Check out the proof here:  
    https://proofwiki.org/wiki/Graph\_is\_Bipartite\_iff\_No\_Odd\_Cycles

Question 2: Bug Genders

Dr. Amy Farrah Fowler is a biologist who is currently studying the reproductive

behavior of a rare species of bugs. She assumes that they feature two different

genders and that they only interact with bugs of the opposite gender. In her

experiment, individual bugs and their interactions were easy to identify,

because numbers were printed on their backs.

As the data analyst in Amy's lab, you are given the experiment data which is a

list of bug interactions, i.e., a list of pairs such as [(1, 9), (4, 5), (3,

5), (3, 4), ...]. Your job is to devise an algorithm that decides whether the

experiment supports Amy's assumption of two genders with no same-sex

interaction, or if it contains some bug interactions that falsify it. In other

words, in we find an evidence proving that Amy's assumption is false, return

"AMY IS WRONG"; otherwise return "AMY MIGHT BE RIGHT".

Devise an algorithm for this and analyze its runtime. Let N be the number of bugs

and K be the number of interactions in the data.

answer:

This question is basically the same as Q1. The algorithm would return "AMY MIGHT BE RIGHT" if the graph of bug interactions is bipartite; otherwise, it returns "AMY IS WRONG"  
  
We need to build the adjacency list first from the edge list, which takes  
O(N+K) time (N on creating the entries for each vertex and K for going through the edge list). The runtime of the DFS is O(N+K). So, overall, the runtime is O(N+K).