CS 5084 - Introduction to Algorithms: Design and Analysis

Section 3.4 Testing Bipartness: An Application of Breadth-First Search

Joe Johnson

Table of Contents

Table of Contents

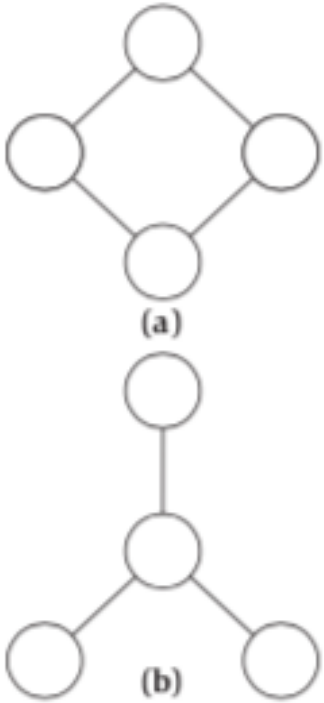
Biparte Graph

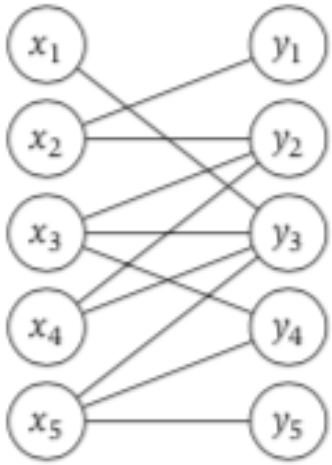
Definition: A *bipartite graph* is a graph $G = (V, E)$ in which the set of nodes, V, can be partitioned into two sets, X and Y such that every edge, $e \in E$ has one end in X and the other end in Y.

In order to understand this idea a bit better, imaging that the nodes in set X are all colored red, and those in Y are colored blue. We can say that a graph is bipartite if every edge has one node that is red and the other blue.

Examples of Bipartite Graphs

Some examples of bipartite graphs:





Example of Non-Bipartite Graph

Can a triangle be bipartite? No. Why? Suppose we color one node red, and another blue. What do we do with the third node? Whether we color it red or blue, we'll have an edge whose nodes are both the same color.

Another Example of Non-Bipartitie Graph

Consider a cycle, C of odd length, with nodes numbered $1, 2, \ldots, 2k, 2k+1$. If we color node 1 red, then we must color node 2 blue, and then node 3 red, and so on - coloring odd-numbered nodes red and even-numbered nodes blue. But finally, we'll get to node $2k+1$ and we'll color it red. But it has an edge to node 1, which is also red. So, there's no way to partition C into red and blue nodes as required.

(3.14) Bipartite Graph - No Odd Cycle

(3.14) If a graph G is bipartite, then it cannot contain an odd cycle.

Table of Contents

Problem: How do we determine whether any given graph, G is bipartite?

(3.14) tells us that if G contains an odd cycle, then it *cannot* be bipartite. This observation will be important in the algorithm we describe next for determining bipartness.

Table of Contents

Consider the following algorithm for testing for bipartness:

1. Assume G is connected. (Since if it is not, we simply take each connected component of G and consider each of these connected components separately.)
2. Take an arbitrary starting node, $s \in V$ and color it red.
3. Now, find all of the neighbors of s and color them blue.
4. Now, find all of the as yet un-colored neighbors of *these* nodes and color them red.
5. Continue with steps 2 and 3 until all of the nodes of G are colored.
6. Check each edge, $e \in E$ to verify whether one node is red and the other blue. If this is true for all $e \in E$, then G is bipartite. However, if there is at least one edge whose nodes are the same color, then G is *not* bipartite.

We can implement this algorithm using BFS, as follows:

- Add an extra array called, *Color* over the nodes.
- Whenever we get to a step in BFS where we are adding a node v to a list $L[i+1]$, we assign $Color[v] =$ red if $i+1$ is an even number and $Color[v] =$ blue if $i+1$ is odd.
- At the end of the procedure, we scan the edges and determine whether there's an edge for which both nodes received the same color.
- Note that this algorithm runs in $O(m+n)$ as it does for BFS.

Table of Contents

(3.15) Analysis of the Algorithm

(3.15) Let G be a connected graph, and let $L_0, L_1, L_2, L_3, \ldots$ be the layers produced by BFS starting at node s. Then, exactly one of the following two statements must hold:

1. There is *no* edge of G joining two nodes of the same layer. In this case, G is a bipartite graph in which the nodes in even-numbered layers can be colored red, and the nodes in odd-numbered layers can be colored blue.
2. There *is* an edge of G joining two nodes of the same layer. In this case, G contains an odd-length cycle, and thus, it cannot be bipartite.

(3.15) Proof

Before getting to the actual proof of (3.15), recall (3.4):

(3.4) Let T be a BFS Tree, let x and y be nodes in T belonging to layers L_i and L_j respectively, and let (x, y) be an edge e in graph, G. Then i and j differ by at most 1.

(3.15) Proof

Proof:

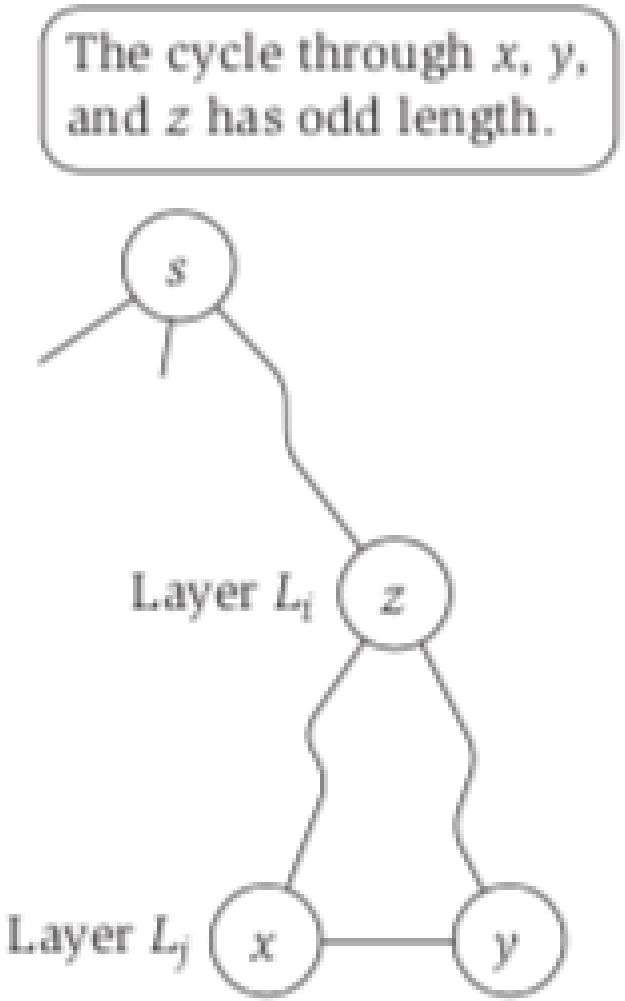
1. Suppose there is *no* edge joining two nodes of the same layer. By (3.4), we know that every edge of G joins nodes either in the same layer of in adjacent layers. Our assumption here is that the same-layer scenario *never* happens. Thus, it must be the case that *every* edge joins nodes in adjacent layers. But our coloring procedure gives nodes in adjacent layers opposite colors. So, every edge has ends with opposite colors. Thus, this coloring establishes that G is bipartite.

(3.15) Proof (contd.)

Proof (contd.):

2. Suppose there exists an edge of G containing two nodes of the same layer. (We need to show that this implies there's an odd-length cycle.) Suppose this is edge is $e = (x, y)$ with $x, y \in L_j$. Also, for notational reasons, recall that L_0 is the set consisting of just s. Now, consider the BFS tree, T, produced by our algorithm, and let z be the node whose layer number, i, (and thus, $z \in L_i$, where $i < j$) is as *large* as possible, subject to the constraint that z is an ancestor of *both* x and y in T. We can call z the *lowest common ancestor* of x and y. Consider the diagram on the following slide...

(3.15) Proof (contd.)



(3.15) Proof (contd.)

Proof (contd.):

2. (contd.) Consider the cycle C which follows the path from z to x, the edge from x to y, and then from y to z. The length of this cycle is given by: $(j - i) + 1 + (j - i)$, adding the three parts separately. This expression is equal to $2(j - i) + 1$, which is an odd number. Thus, C is an odd-length cycle, and G is not bipartite.