Exercises


Solution: Assume that each vertex has a member variable `assigned` and a second member variable called `set`. The first is a `bool` and the second is an `int`. Here’s the solution. Starting with a random vertex, each vertex is labeled as either set 0 or set 1. If a neighbor of a vertex is in one set, it must be placed in the other. If these constraints say that it must be in both set 0 and set 1, the graph is not bipartite.

```cpp
// return true if and only if the graph is bipartite
// if so, the label "set" for each vertex will be 0 or 1;
bool
Graph::Bipartite()
{
    for each vertex v do
        v->assigned = false;
    int current_set = 1;
    list<vertex*> ready;
    vertices[0]->assigned = true;
    vertices[0]->set = current_set;
    ready.push_back( vertices[0] );  // first vertex
    while ( ! ready.empty() ) do {
        v = ready.begin(); ready.pop_front();
        current_set = ( v->set + 1 ) % 2;  // 1->0 and 0->1
        for ( each vertex w adjacent to v ) do {
            if ( w->assigned ) {
                if ( w->set != current_set )
                    return false;  // w must be in both set 0 and set 1!
                    // so graph is not bipartite
            }
            else {
                w->set = current_set;
                w->assigned = true;
                ready.push_back( w );
            }
        }
    }
}
```
return true; // made it through without a conflict

The algorithm is clearly $O(|V| + |E|)$ because each vertex is considered once.