March 13, 2019

Homework 3: IsBipartite Algorithm
50 Points

A $k-coloring$ of a graph $G = (V, E)$ is a mapping $color : V \rightarrow \{1, \ldots, k\}$ such that for any edge $e = [v, w] \in E$ $color(v) \neq color(w)$. G is said to be *bipartite* iff G has a 2-coloring.

The idea of a k-coloring is that the vertices can be “colored” using k distinct colors so that the vertices of any edge have different colors. A bipartite graph is one that can be colored with two colors.

Part 1. Invent an algorithm named **IsBipartite** with these properties:

(1) IsBipartite operates on any undirected graph $G = (V, E)$
(2) IsBipartite returns true iff G is bipartite
(3) If IsBipartite returns true then a supplied vector will be populated with a 2-coloring of the vertices of G
(4) The runtime of IsBipartite is $\leq \mathcal{O}(|V| + |E|)$

The Algorithm

```
bool IsBipartite
  Input:  Graph G(V,E)
  Return: true iff G is 2-colorable
  Output if true: mapping color: V -> {1,2} such that:
                  for any edge e = [x,y] in E color(x) != color(y)
  {
    1: Run breadth-first survey on G
    2: Assign colors based on distance parity:
       for each vertex v
       {
         if bfs.Distance(v) is even
           color(v) = 1
         else
           color(v) = 2
       }
    3: Check 2-color property:
       for each vertex v
       {
         for each neighbor w of v
         {
           if color(w) == color(v) return false;
         }
       }
    return true
  }
```

Part 2. Code up the algorithm in C++ conformant with the stub below. Test the implementation on small graphs that can be hand verified and on some large graphs (such as the “Kevin Bacon” actor-movie abstract graph) and some very large graphs generated at random. Include some random maze graphs, and report any discoveries.

The Implementation

```
template < class G >
bool IsBipartite ( const G& g , fsu::Vector <char>& color )
{
  fsu::BFSurvey<G> bfs(g);
  bfs.Search();
  color.SetSize(g.VrtxSize());
  // set color Red for even distance and Black for odd distance
  for (typename G::Vertex v = 0; v < g.VrtxSize(); ++v)
  {
    (0 == bfs.Distance()[v] % 2 ? color[v] = ’R’ : color[v] = ’B’);
  }
  // all vertices have been colored; check 2-color property
  for (typename G::Vertex  v = 0; v < g.VrtxSize(); ++v)
  {
    for (typename G::AdjIterator i = g.Begin(v); i != g.End(v); ++i)
    {
      if (color[*i] == color[v]) return 0;
    }
  }
  return 1;
}

template < class G >
bool IsBipartite ( const G& g )
{
  fsu::Vector<char> color (g.VrtxSize());
  return IsBipartite (g,color);
}
```

```
Sample Test Results
===================
graph            g.VrtxSize()    g.EdgeSize()  IsBipartite()  vertex color counts
-----            -----------     ------------  -----------    -------------------
graph1.10.10     10              10            true           Red = 5 , Black = 5
graph.20.25      20              25            false
rangraph1        100             100           false
rangraph2        100000          35000         true           Red = 75772 , Black = 24228
rangraph3        100000          350000        false
Kevin Bacon      119429          202927        true           Red = 4188 , Black = 115241
rangraph_bp      20000           100000        true           Red = 10000 , Black = 10000
ranmaze100x200   20000           18200         true           Red = 10674 , Black = 9326
```

Discovery. Maze graphs are bipartite? (True - any subgraph of the square lattice is bipartite.)

Part 3. Provide a proof that your algorithm is correct.

Lemma 1. If G has a 2-coloring then colors of vertices along any path in G must alternate.

Proof of Lemma 1. The vertices along a path are the ends of the connecting edges of the path. By definition, a k-coloring must assign different colors to the two ends of each edge. When $k = 2$ there are only 2 color choices, so they must alternate. $\square$

Lemma 2. If G is bipartite and connected then except for color names any two 2-colorings of G are the same.

Proof of Lemma 2. Suppose c_1 is a coloring of G. Choose any vertex v_0. Then the color of any other vertex x is determined by following a path from v_0 to x: by Lemma 1, if the path has an even number of edges then $c_1(x) = c_1(v_0)$ and if the path has an odd number of edges then $c_1(x)$ is the other color. So if c_2 is any other 2-coloring of G then either $c_2(x) = c_1(x)$ for all vertices x or $c_2(x) \neq c_1(x)$ for all vertices x: c_1 and c_2 either agree on all vertices or they supply the opposite color for all vertices. $\square$

Proof of correctness of algorithm. Assume without loss of generality that the graph is connected. (Otherwise, apply the argument to each component of G.)

Step 1 runs a BFSurvey on G which results in the distance calculated from a search root for each vertex. Since the distance is realized by a path, any 2-coloring must respect the parity along these paths - colors must alternate as we encounter vertices along any path. (Lemma 1.)

Step 2 assigns colors based on distance parity. By Lemma 1, any 2-coloring must satisfy this property.

Step 3 checks the 2-color property for every pair of adjacent vertices. If a failure is found, the algorithm terminates and returns false. If no failure is found, true is returned and the coloring is returned.

Note that when the check fails, there is no other possible 2-coloring of the graph, by Lemma 2. Thus when a check fails, the only possible 2-coloring has failed, and the graph is not bipartite. If all checks pass, the 2-coloring has been found. $\square$

Part 4. Provide a proof that your algorithm has runtime $\leq \mathcal{O}(|V| + |E|)$

Proof. We know that the runtime of BFSurvey done in Step 1 is $\Theta(|V| + |E|)$ by Theorem 2f of the graph notes. Step 2 consists of a simple loop over all vertices, so step 2 has runtime $= \Theta(|V|)$. Step 3 consists of a standard traversal "touching" each vertex and each edge (twice ...) and hence has runtime $\Theta(|V| + 2|E|) = \Theta(|V| + |E|)$. Since these three steps are executed consequtively, the runtime is the maximum of the three estimates. $\square$