```
#ifndef BFS_H_
#define BFS_H_

class bfsGraph
{
public:
    bfsGraph(int v, int e);
        //Constructor
    void addEdge(cities start, cities e);
        //Adds a new edge to the graph
    void bfsTraversal(cities start);
        //Breadth first traversal
    void printEdges();
        //Prints out the discovery and cross edges
private:
    int vNum; //number of vertices
    int eNum; //number of edges
    int** adjMat; //adjacency matrix
    vector<edge> backEdges; //vector of the back edges
    vector<edge> discoveryEdges; //vector of the discovery edges

    int findDistBtwn(cities v1, cities v2);
    vector<int> sortCity(cities v);
};

bfsGraph::bfsGraph(int v, int e)
{
    vNum = v;
    eNum = e;
    adjMat = new int*[vNum];
    for (int row = 0; row < v; row++)
    {
        adjMat[row] = new int[v];
        for(int column = 0; column < v; column++)
        {
            adjMat[row][column] = 0;
        }
    }

    addEdge(Seattle, Chicago);
    addEdge(Seattle, Denver);
    addEdge(Seattle, SanFrancisco);
    addEdge(SanFrancisco, Seattle);
    addEdge(SanFrancisco, Denver);
    addEdge(SanFrancisco, LosAngeles);
    addEdge(Denver, Seattle);
    addEdge(Denver, SanFrancisco);
    addEdge(Denver, LosAngeles);
    addEdge(Denver, KansasCity);
    addEdge(Denver, Chicago);
    addEdge(Chicago, Seattle);
    addEdge(Chicago, Denver);
    addEdge(Chicago, KansasCity);
    addEdge(Chicago, NewYork);
    addEdge(Chicago, Boston);
    addEdge(Boston, Chicago);
    addEdge(Boston, NewYork);
```

```
    addEdge(NewYork, Boston);
    addEdge(NewYork, Chicago);
    addEdge(NewYork, Atlanta);
    addEdge(NewYork, KansasCity);
    addEdge(LosAngeles, SanFrancisco);
    addEdge(LosAngeles, Denver);
    addEdge(LosAngeles, KansasCity);
    addEdge(LosAngeles, Dallas);
    addEdge(KansasCity, LosAngeles);
    addEdge(KansasCity, Denver);
    addEdge(KansasCity, Chicago);
    addEdge(KansasCity, NewYork);
    addEdge(KansasCity, Atlanta);
    addEdge(KansasCity, Dallas);
    addEdge(Atlanta, NewYork);
    addEdge(Atlanta, KansasCity);
    addEdge(Atlanta, Dallas);
    addEdge(Atlanta, Houston);
    addEdge(Atlanta, Miami);
    addEdge(Dallas, LosAngeles);
    addEdge(Dallas, KansasCity);
    addEdge(Dallas, Atlanta);
    addEdge(Dallas, Houston);
    addEdge(Houston, Dallas);
    addEdge(Houston, Atlanta);
    addEdge(Houston, Miami);
    addEdge(Miami, Atlanta);
    addEdge(Miami, Houston);
}

void bfsGraph::addEdge(cities start, cities e)
{
    adjMat[static_cast<int>(start)][static_cast<int>(e)] = 1;
    adjMat[static_cast<int>(e)][static_cast<int>(start)] = 1;
}

void bfsGraph::bfsTraversal(cities start)
{
    int totalDist = 0;
    vector<bool> visited(vNum, false);
    vector<int> q;
    q.push_back(static_cast<int>(start));

    visited[start] = true;

    int vis;
    while(!q.empty())
    {
        vis = q[0];

        cout << enumGet(static_cast<cities>(vis)) << endl;
        q.erase(q.begin());

        vector<int> vec = sortCity(static_cast<cities>(vis));

        vector<int>::iterator i;
        for(i = vec.begin(); i != vec.end(); i++)
```

```
115         {
116             if(adjMat[vis][*i] == 1 && (!visited[*i]))
117             {
118                 discoveryEdges.push_back(edge(static_cast<cities>(vis),
119                                               static_cast<cities>(*i)));
120                 q.push_back(*i);
121                 totalDist += findDistBtwn(static_cast<cities>(vis),
122                                           static_cast<cities>(*i));
123                 visited[*i] = true;
124             }
125             else if(adjMat[vis][*i] == 1 && (visited[*i]))
126             {
127                 backEdges.push_back(edge(static_cast<cities>(vis),
128                                          static_cast<cities>(*i)));
129             }
130         }
131     }
132     cout << "\nTotal Distance Traveled: " << totalDist << " miles.\n";
133 }
134
135 void bfsGraph::printEdges()
136 {
137     cout << "\nDiscovery Edges:";
138     vector<edge>::iterator i;
139     for(i = discoveryEdges.begin(); i != discoveryEdges.end(); i++)
140     {
141         cout << endl << enumGet(i->city1) << "->" << enumGet(i->city2);
142     }
143
144     cout << "\n\nBack Edges:";
145     for(i = backEdges.begin(); i != backEdges.end(); i++)
146     {
147         cout << endl << enumGet(i->city1) << "->" << enumGet(i->city2);
148     }
149 }
150
151 vector<int> bfsGraph::sortCity(cities v)
152 {
153     vector<int> vec;
154     for(int j = 0; j < vNum; j++)
155     {
156         vec.push_back(j);
157     }
158
159     bool swap = true;
160     while(swap)
161     {
162       swap = false;
163       int size = vec.size();
164       for (int i = 0; i < size -1; i++)
165       {
166           if (findDistBtwn(v, static_cast<cities>(vec[i])) > findDistBtwn(v,
167                           static_cast<cities>(vec[i+1])))
168           {
169               int temp = vec[i];
170               vec[i] = vec[i+1];
171               vec[i+1] = temp;
```

```
172                 swap = true;
173             }
174         }
175     }
176 
177     return vec;
178  }
179 
180 int bfsGraph::findDistBtwn(cities v1, cities v2)
181 {
182     int distBtwn;
183     int j = 0;
184     while (j < 23)
185     {
186         if((distances[j].city1 == v1 && distances[j].city2 == v2) ||
187            (distances[j].city2 == v1 && distances[j].city1 == v2))
188         {
189             distBtwn = distances[j].distance;
190             break;
191         }
192         else
193         {
194             j++;
195         }
196     }
197     return distBtwn;
198 }
199 
200 #endif /* BFS_H_ */
201 
```