

```

1 #ifndef DFS_H_
2 #define DFS_H_
3
4 class dfsGraph
5 {
6 public:
7     dfsGraph(int vN);
8         //Constructor
9     void addEdge(cities v, cities w);
10        //Adds a new edge to the graph
11     void dfsTraversal(cities start);
12        //Depth first traversal
13     void printEdges();
14        //Prints out the discovery and cross edges
15 private:
16     int vNum; //number of vertices
17     vector<cities> *adjList; //adjacency list
18     vector<edge> backEdges; //vector of the back edges
19     vector<edge> discoveryEdges; //vector of the discovery edges
20
21     void dfsUtil(cities v, bool visited[], int& tDist);
22        //Recursive function for the Depth first traversal
23     int findDistBtw(cities v1, cities v2);
24        //Find the distance between 2 vertices
25        //returns int of the distance
26     void bubbleSort(cities v, vector<cities>& a);
27        //Vector bubble sort (least to greatest)
28 };
29
30 dfsGraph::dfsGraph(int vN)
31 {
32     vNum = vN;
33     adjList = new vector<cities>[vNum];
34
35     addEdge(Seattle, Chicago);
36     addEdge(Seattle, Denver);
37     addEdge(Seattle, SanFrancisco);
38     addEdge(SanFrancisco, Seattle);
39     addEdge(SanFrancisco, Denver);
40     addEdge(SanFrancisco, LosAngeles);
41     addEdge(Denver, Seattle);
42     addEdge(Denver, SanFrancisco);
43     addEdge(Denver, LosAngeles);
44     addEdge(Denver, KansasCity);
45     addEdge(Denver, Chicago);
46     addEdge(Chicago, Seattle);
47     addEdge(Chicago, Denver);
48     addEdge(Chicago, KansasCity);
49     addEdge(Chicago, NewYork);
50     addEdge(Chicago, Boston);
51     addEdge(Boston, Chicago);
52     addEdge(Boston, NewYork);
53     addEdge(NewYork, Boston);
54     addEdge(NewYork, Chicago);
55     addEdge(NewYork, Atlanta);
56     addEdge(NewYork, KansasCity);
57     addEdge(LosAngeles, SanFrancisco);

```

```

58     addEdge(LosAngeles, Denver);
59     addEdge(LosAngeles, KansasCity);
60     addEdge(LosAngeles, Dallas);
61     addEdge(KansasCity, LosAngeles);
62     addEdge(KansasCity, Denver);
63     addEdge(KansasCity, Chicago);
64     addEdge(KansasCity, NewYork);
65     addEdge(KansasCity, Atlanta);
66     addEdge(KansasCity, Dallas);
67     addEdge(Atlanta, NewYork);
68     addEdge(Atlanta, KansasCity);
69     addEdge(Atlanta, Dallas);
70     addEdge(Atlanta, Houston);
71     addEdge(Atlanta, Miami);
72     addEdge(Dallas, LosAngeles);
73     addEdge(Dallas, KansasCity);
74     addEdge(Dallas, Atlanta);
75     addEdge(Dallas, Houston);
76     addEdge(Houston, Dallas);
77     addEdge(Houston, Atlanta);
78     addEdge(Houston, Miami);
79     addEdge(Miami, Atlanta);
80     addEdge(Miami, Houston);
81 }
82
83 void dfsGraph::addEdge(cities v, cities w)
84 {
85     adjList[v].push_back(w);
86 }
87
88 void dfsGraph::dfsTraversal(cities start)
89 {
90     int totalDist = 0;
91     bool *visited = new bool[vNum];
92     for(int i = 0; i < vNum; i++)
93     {
94         visited[i] = false;
95     }
96
97     dfsUtil(start, visited, totalDist);
98     cout << "\nTotal Distance Traveled: " << totalDist << " miles.\n";
99 }
100
101 void dfsGraph::dfsUtil(cities v, bool visited[], int& tDist)
102 {
103     visited[v] = true;
104     cout << enumGet(v) << endl;
105
106     bubbleSort(v, adjList[v]);
107
108     vector<cities>::iterator i;
109     for(i = adjList[v].begin(); i != adjList[v].end(); i++)
110     {
111         if(!visited[*i])
112         {
113             discoveryEdges.push_back(edge(v, *i));
114             tDist += findDistBtw(v, *i);

```

```

115         dfsUtil(*i, visited, tDist);
116     }
117     else
118     {
119         backEdges.push_back(edge(v, *i));
120     }
121 }
122 }
123
124 int dfsGraph::findDistBtwn(cities v1, cities v2)
125 {
126     int distBtwn;
127     int j = 0;
128     while (j < 23)
129     {
130         if((distances[j].city1 == v1 && distances[j].city2 == v2) ||
131            (distances[j].city2 == v1 && distances[j].city1 == v2))
132         {
133             distBtwn = distances[j].distance;
134             break;
135         }
136         else
137         {
138             j++;
139         }
140     }
141     return distBtwn;
142 }
143
144 void dfsGraph::bubbleSort(cities v, vector<cities>& a)
145 {
146     bool swap = true;
147     while(swap)
148     {
149         swap = false;
150         int size = a.size();
151         for (int i = 0; i < size - 1; i++)
152         {
153             if (findDistBtwn(v, a[i]) > findDistBtwn(v, a[i+1]))
154             {
155                 cities temp = a[i];
156                 a[i] = a[i+1];
157                 a[i+1] = temp;
158                 swap = true;
159             }
160         }
161     }
162 }
163
164 void dfsGraph::printEdges()
165 {
166     cout << "\nDiscovery Edges:";
167     vector<edge>::iterator i;
168     for(i = discoveryEdges.begin(); i != discoveryEdges.end(); i++)
169     {
170         cout << endl << enumGet(i->city1) << "->" << enumGet(i->city2);
171     }

```

dfs.h

```
172
173     cout << "\n\nBack Edges:";
174     for(i = backEdges.begin(); i != backEdges.end(); i++)
175     {
176         cout << endl << enumGet(i->city1) << "->" << enumGet(i->city2);
177     }
178 }
179
180 #endif /* DFS_H_ */
181
```