

```

...#includes

14 const string WIKI_PREFIX = "https://en.wikipedia.org";
15 const string PHILOSOPHY = "/wiki/Philosophy";
16
17 typedef Vector<Vertex *> Path;
18
19 void findPathDFS(string url);
20 void findPathBFS(string url);
21 void findPathDFSRecursive(string currentPage);
22 bool findPathDFSRecursive(Vertex *currentPage,
23                             Vertex *end,
24                             Path &currPath,
25                             Vector<Vertex *> &allVertices,
26                             Set<string> &seen);
27 Vector<string> getPageLinks(string url);
28 string removeParens(string s);
29
30 int main() {
31     while (true) {
32         int searchChoice = getInteger("Please select:\n\t1.DFS\n\t2.BFS\n\t3.DFS
(recursive)\n\t",
33                                     "Please enter an integer!");
34         string startPage = getLine("Please enter a Wikipedia Page to start on
(e.g., Stanford University)");
35         // replace space with underscore (necessary for some searches)
36         replace( startPage.begin(), startPage.end(), ' ', '_' );
37
38         if (searchChoice == 1) {
39             findPathDFS("/wiki/"+startPage);
40         } else if (searchChoice == 2) {
41             findPathBFS("/wiki/"+startPage);
42         } else {
43             findPathDFSRecursive("/wiki/"+startPage);
44         }
45         cout << endl;
46     }
47     return 0;
48 }
49
50 // Perform a DFS on Wikipedia
51 void findPathDFS(string currentPage) {
52     Vector<Vertex *> allVertices; // for cleanup
53
54     Vertex *start = new Vertex(currentPage);
55     allVertices.add(start);
56
57     Vertex *end = new Vertex(PHILOSOPHY);
58     allVertices.add(end);
59
60     Vector<string> wikiLinks;
61     Stack<Path> fringe;
62     Path startPath;
63     startPath.add(start);
64     fringe.push(startPath);
65     Set<string> seen;
66

```

```

67 while(!fringe.isEmpty()) {
68     Path currPath = fringe.pop();
69     Vertex * last = currPath[currPath.size() - 1];
70     cout << "Looking at " << last->name << endl;
71     if(last->name == end->name) {
72         cout << endl << "Found a path from " << start->name.substr(6)
73             << " to " << PHILOSOPHY.substr(6) << "! ("
74             << currPath.size()-1 << " clicks)" << endl;
75         for (Vertex *v : currPath) {
76             cout << "\t" << v->name.substr(6) << endl;
77         }
78         return;
79     }
80     if (seen.contains(last->name)) {
81         continue;
82     }
83     seen.add(last->name);
84     // get neighbors from the Wikipedia page
85     wikiLinks = getPageLinks(last->name);
86     // do in reverse order so we always pop the first link first
87     for (int i=wikiLinks.size()-1; i >= 0; i--) {
88         string link = wikiLinks[i];
89         if(!seen.contains(link)) {
90             Path newPath = currPath;
91             Vertex *neighbor = new Vertex(link);
92             allVertices.add(neighbor);
93             newPath.add(neighbor);
94             fringe.add(newPath);
95         }
96     }
97 }
98 // clean up
99 for (Vertex *v : allVertices) {
100     delete v;
101 }
102 }
103

```

```

104 void findPathBFS(string currentPage) {
105
106     Vector<Vertex *> allVertices; // for cleanup
107
108     Vertex *start = new Vertex(currentPage);
109     allVertices.add(start);
110
111     Vertex *end = new Vertex(PHILOSOPHY);
112     allVertices.add(end);
113
114     Vector<string> wikiLinks;
115     Queue<Path> fringe;
116     Path startPath;
117     startPath.add(start);
118     fringe.enqueue(startPath);
119     Set<string> seen;
120
121     while(!fringe.isEmpty()) {
122         Path currPath = fringe.dequeue();
123         Vertex * last = currPath[currPath.size() - 1];
124         if(last->name == end->name) {
125             cout << endl << "Found a path from " << start->name.substr(6)
126                 << " to " << PHILOSOPHY.substr(6) << "! ("
127                 << currPath.size()-1 << " clicks)" << endl;
128             for (Vertex *v : currPath) {
129                 cout << "\t" << v->name.substr(6) << endl;
130             }
131             return;
132         }
133         if (seen.contains(last->name)) {
134             continue;
135         }
136         seen.add(last->name);
137         cout << "Looking at " << last->name << endl;
138         // get neighbors from the Wikipedia page
139         wikiLinks = getPageLinks(last->name);
140         // do in reverse order so we always pop the first link first
141         for (int i=wikiLinks.size()-1; i >= 0; i--) {
142             string link = wikiLinks[i];
143             if(!seen.contains(link)) {
144                 Path newPath = currPath;
145                 Vertex *neighbor = new Vertex(link);
146                 allVertices.add(neighbor);
147                 newPath.add(neighbor);
148                 fringe.add(newPath);
149             }
150         }
151     }
152     // clean up
153     for (Vertex *v : allVertices) {
154         delete v;
155     }
156 }
157

```

```

238 // Perform a DFS on Wikipedia Recursively
239 void findPathDFSRecursive(string currentPage) {
240     Vector<Vertex *> allVertices; // for cleanup
241
242     Vertex *start = new Vertex(currentPage);
243     allVertices.add(start);
244
245     Vertex *end = new Vertex(PHILOSOPHY);
246     allVertices.add(end);
247
248     Path currentPath;
249     currentPath.add(start);
250
251     Set<string> seen;
252
253     findPathDFSRecursive(start, end, currentPath, allVertices, seen);
254     // clean up
255     for (Vertex *v : allVertices) {
256         delete v;
257     }
258 }
259
260 // Recursive helper function for Recursive DFS
261 bool findPathDFSRecursive(Vertex *currentPage,
262                          Vertex *end,
263                          Path &currPath,
264                          Vector<Vertex *> &allVertices,
265                          Set<string> &seen){
266     Vector<string> wikiLinks;
267     // base case
268     if (currentPage->name == PHILOSOPHY) {
269         cout << endl << "Found a path from " << currentPage->name.substr(6)
270             << " to " << PHILOSOPHY.substr(6) << "! ("
271             << currPath.size()-1 << " clicks)" << endl;
272         for (Vertex *v : currPath) {
273             cout << "\t" << v->name.substr(6) << endl;
274         }
275         return true;
276     }
277
278     // another base case
279     if (seen.contains(currentPage->name)) {
280         return false;
281     }
282
283     cout << "Looking at " << currentPage->name << endl;
284     Vertex * last = currPath[currPath.size() - 1];
285     seen.add(last->name);
286     // get neighbors from the Wikipedia page
287     wikiLinks = getPageLinks(last->name);
288     for (int i=0; i < wikiLinks.size(); i++) {
289         string link = wikiLinks[i];
290         if(!seen.contains(link)) {
291             Path newPath = currPath;
292             Vertex *neighbor = new Vertex(link);
293             allVertices.add(neighbor);
294             newPath.add(neighbor);
295             if (findPathDFSRecursive(neighbor,end,newPath,allVertices,seen)) {
296                 return true;
297             }
298         }
299     }
300     return false;
301 }

```