```
...#includes

const string WIKI_PREFIX = "https://en.wikipedia.org";
const string PHILOSOPHY = "/wiki/Philosophy";

typedef Vector<Vertex *> Path;

void findPathDFS(string url);
void findPathBFS(string url);
void findPathDFSRecursive(string currentPage);
bool findPathDFSRecursive(Vertex *currentPage,
                          Vertex *end,
                          Path &currPath,
                          Vector<Vertex *> &allVertices,
                          Set<string> &seen);
Vector<string> getPageLinks(string url);
string removeParens(string s);

int main() {
    while (true) {
        int searchChoice = getInteger("Please select:\n\t1.DFS\n\t2.BFS\n\t3.DFS (recursive)\n\t",
                                      "Please enter an integer!");
        string startPage = getLine("Please enter a Wikipedia Page to start on (e.g., Stanford University)");
        // replace space with underscore (necessary for some searches)
        replace( startPage.begin(), startPage.end(), ' ', '_' );

        if (searchChoice == 1) {
            findPathDFS("/wiki/"+startPage);
        } else if (searchChoice == 2) {
            findPathBFS("/wiki/"+startPage);
        } else {
            findPathDFSRecursive("/wiki/"+startPage);
        }
        cout << endl;
    }
    return 0;
}

// Perform a DFS on Wikipedia
void findPathDFS(string currentPage) {
    Vector<Vertex *> allVertices; // for cleanup

    Vertex *start = new Vertex(currentPage);
    allVertices.add(start);

    Vertex *end = new Vertex(PHILOSOPHY);
    allVertices.add(end);

    Vector<string> wikiLinks;
    Stack<Path> fringe;
    Path startPath;
    startPath.add(start);
    fringe.push(startPath);
    Set<string> seen;

    while(!fringe.isEmpty()) {
        Path currPath = fringe.pop();
        Vertex * last = currPath[currPath.size() - 1];
        cout << "Looking at " << last->name << endl;
        if(last->name == end->name) {
            cout << endl << "Found a path from " << start->name.substr(6)
                 << " to " << PHILOSOPHY.substr(6) << "! ("
                 << currPath.size()-1 << " clicks)" << endl;
            for (Vertex *v : currPath) {
                cout << "\t" << v->name.substr(6) << endl;
            }
            return;
        }
        if (seen.contains(last->name)) {
            continue;
        }
        seen.add(last->name);
        // get neighbors from the Wikipedia page
        wikiLinks = getPageLinks(last->name);
        // do in reverse order so we always pop the first link first
        for (int i=wikiLinks.size()-1; i >= 0; i--) {
            string link = wikiLinks[i];
            if(!seen.contains(link)) {
                Path newPath = currPath;
                Vertex *neighbor = new Vertex(link);
                allVertices.add(neighbor);
                newPath.add(neighbor);
                fringe.add(newPath);
            }
        }
    }
    // clean up
    for (Vertex *v : allVertices) {
        delete v;
    }
}
```

```
void findPathBFS(string currentPage) {

    Vector<Vertex *> allVertices; // for cleanup

    Vertex *start = new Vertex(currentPage);
    allVertices.add(start);

    Vertex *end = new Vertex(PHILOSOPHY);
    allVertices.add(end);

    Vector<string> wikiLinks;
    Queue<Path> fringe;
    Path startPath;
    startPath.add(start);
    fringe.enqueue(startPath);
    Set<string> seen;

    while(!fringe.isEmpty()) {
        Path currPath = fringe.dequeue();
        Vertex * last = currPath[currPath.size() - 1];
        if(last->name == end->name) {
            cout << endl << "Found a path from " << start->name.substr(6)
                 << " to " << PHILOSOPHY.substr(6) << "! ("
                 << currPath.size()-1 << " clicks)" << endl;
            for (Vertex *v : currPath) {
                cout << "\t" << v->name.substr(6) << endl;
            }
            return;
        }
        if (seen.contains(last->name)) {
            continue;
        }
        seen.add(last->name);
        cout << "Looking at " << last->name << endl;
        // get neighbors from the Wikipedia page
        wikiLinks = getPageLinks(last->name);
        // do in reverse order so we always pop the first link first
        for (int i=wikiLinks.size()-1; i >= 0; i--) {
            string link = wikiLinks[i];
            if(!seen.contains(link)) {
                Path newPath = currPath;
                Vertex *neighbor = new Vertex(link);
                allVertices.add(neighbor);
                newPath.add(neighbor);
                fringe.add(newPath);
            }
        }
    }
    // clean up
    for (Vertex *v : allVertices) {
        delete v;
    }
}
```

```
// Perform a DFS on Wikipedia Recursively
void findPathDFSRecursive(string currentPage) {
    Vector<Vertex *> allVertices; // for cleanup

    Vertex *start = new Vertex(currentPage);
    allVertices.add(start);

    Vertex *end = new Vertex(PHILOSOPHY);
    allVertices.add(end);

    Path currentPath;
    currentPath.add(start);

    Set<string> seen;

    findPathDFSRecursive(start, end, currentPath, allVertices, seen);
    // clean up
    for (Vertex *v : allVertices) {
        delete v;
    }
}

// Recursive helper function for Recursive DFS
bool findPathDFSRecursive(Vertex *currentPage,
                          Vertex *end,
                          Path &currPath,
                          Vector<Vertex *> &allVertices,
                          Set<string> &seen){
    Vector<string> wikiLinks;
    // base case
    if (currentPage->name == PHILOSOPHY) {
        cout << endl << "Found a path from " << currentPage->name.substr(6)
             << " to " << PHILOSOPHY.substr(6) << "! ("
             << currPath.size()-1 << " clicks)" << endl;
        for (Vertex *v : currPath) {
            cout << "\t" << v->name.substr(6) << endl;
        }
        return true;
    }

    // another base case
    if (seen.contains(currentPage->name)) {
        return false;
    }

    cout << "Looking at " << currentPage->name << endl;
    Vertex * last = currPath[currPath.size() - 1];
    seen.add(last->name);
    // get neighbors from the Wikipedia page
    wikiLinks = getPageLinks(last->name);
    for (int i=0; i < wikiLinks.size(); i++) {
        string link = wikiLinks[i];
        if(!seen.contains(link)) {
            Path newPath = currPath;
            Vertex *neighbor = new Vertex(link);
            allVertices.add(neighbor);
            newPath.add(neighbor);
            if (findPathDFSRecursive(neighbor,end,newPath,allVertices,seen)) {
                return true;
            }
        }
    }
    return false;
}
```