```
#include "d_graph.h"
#include "d_vector.h"
#include "d_util.h" // writeContainer()
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <list>

// Program written by Eric Wasiolek Copyright 2008 except ShortestPath function

bool sublist(list<string> lsubpath, list<string> lpath)
{
//	cout << "entered sublist" << endl;
	int subpathlength = lsubpath.size();
	int pathlength = lpath.size();
	int count = 0;

	vector<string> subpath;
	vector<string> path;
	list<string>::iterator spiter;
	spiter = lsubpath.begin();
	list<string>::iterator piter;
	piter = lpath.begin();

	for(int i=0; i < subpathlength; ++i)
	{
		subpath.push_back(*spiter++);
	}
```

```
	for(int i=0; i < pathlength; ++i)
	{
		path.push_back(*piter++);
	}
	for(int i = 1; i < pathlength-subpathlength; ++i)
	{
		for(int j = 0; j < subpathlength; ++j)
			if(subpath[j] == path[i+j])
				++count;
		if(count == subpathlength)
			return true;
		count = 0;
	}
	return false;
}

template<typename T>
int find(graph<T> &g, string x)
{
	int r = 0;

	for(r=0; r < g.numVertices; ++r)
		if((*(g.vInfo[r].vtxMapLoc)).first == x)
		{
			return r;
		}
		else if(r == g.numVertices - 1)
		{
			cout << "Couldn't find vertex." << endl;
```

```
                return -1;
            }
}


bool operator== (list<string> l1, list<string> l2)
{
    if(l1.size() != l2.size())
        return false;
    list<string>::iterator liter1, liter2;
    liter1 = l1.begin();
    liter2 = l2.begin();

    while(liter1 != l1.end())
    {
        if(*liter1 != *liter2)
            return false;
        else if (liter1 == l1.end())
            return true;
        ++liter1;
        ++liter2;
    }
}


bool find(list<string> l1, vector<list<string> >& subpaths)
{
```

```
	// cout << "Entered find list in vector of lists" << endl;
	if(subpaths.empty())
		return false;
	for(int i =0; i < subpaths.size(); ++i)
		if(subpaths[i] == l1)  // compare two lists using the operator==
		{
			cout << "find returned true" << endl;
			return true;
		}
		else if(i == subpaths.size()-1)
		{
			cout << "find returned false" << endl;
			return false;
		}
}

void nrinsert(list<string> x, vector<list<string> >& subpaths)
{
	if(find(x,subpaths))
	{
	//	cout << "Found subpath in subpath list" << endl;
		return;
	}
	else
	{
	//	cout << "Did not find subpath in subpath list, insert it." << endl;
		subpaths.push_back(x);
	}
}
```

```
void subpaths(list<string> path, vector<list<string> >& vsubpaths)
{
	vector<string> v;
	list<string>::iterator liter;
	liter = path.begin();
	for(int i = 0; i < path.size(); ++i)
		v.push_back(*liter++);

	list<string> subpath;
	int passes = v.size()-2;
	int number = v.size()-1;
	int length = 1;
	for(int pass = 0; pass < passes; ++pass)
	{
		for(int i=1; i < number; ++i)
		{
			for(int j=0; j<length; ++j)
				subpath.push_back(v[i+j]);
			cout << "subpath is: ";
			writeList(subpath, " ");
			cout << endl;
			nrinsert(subpath, vsubpaths);

			subpath.erase(subpath.begin(), subpath.end());
		}
		++length;
		--number;
	}
```

```
}

int main(int argc, char *argv[])
{

	// Open four files
	ifstream graphname;
	ifstream inputneurons;
	ifstream outputneurons;
	ofstream neuronpaths;

	if(argc != 5)
	{
		cout << "Usage: subpaths graphname inputneurons outputneurons neuronpathlist " <<
endl;
		return 0;
	}
	else
	{
		graphname.open(argv[1]);
		if(!graphname)
		{
			cout << "Could not open graph file!" << endl;
			return 0;
		}
		else
			cout << "Opened graph file " << argv[1] << " successfully." << endl;
		inputneurons.open(argv[2]);
```

```
	if(!inputneurons)
	{
		cout << "Could not open input neuron file!" << endl;
		return 0;
	}
	else
		cout << "Opened input file " << argv[2] << " successfully." << endl;
	outputneurons.open(argv[3]);
	if(!outputneurons)
	{
		cout << "Could not open output neuron file!" << endl;
		return 0;
	}
	else
		cout << "Opened input file " << argv[3] << " successfully." << endl;
	neuronpaths.open(argv[4]);
	if(!neuronpaths)
	{
		cout << "Could not open neuron path output file!" << endl;
		return 0;
	}
	else
		cout << "Opened output file " << argv[4] << " successfully." << endl;


}
```

```
    // Load the input neuron vector

    int numinputnodes = 0, numoutputnodes = 0, i = 0;
    string temp;
    inputneurons >> numinputnodes;
string vinputneurons[numinputnodes];
    for(i = 0; i < numinputnodes; ++i)
    {
            inputneurons >> temp;
            vinputneurons[i] = temp;
    }

    // load the output neuron vector

    outputneurons >> numoutputnodes;
string voutputneurons[numoutputnodes];
    for(i = 0; i < numoutputnodes; ++i)
    {
            outputneurons >> temp;
            voutputneurons[i] = temp;
    }

    // Open graph
    graph<string> g;
    g.clear();

    graphname >> g;
```

```
// calculate and write out paths from input neurons to output neurons


list<string> path;

vector<list<string> > vpaths;


int p=0;

int q=0;

for(p = 0; p < numinputnodes; ++p)

        for(q = 0; q < numoutputnodes; ++q)

        {

                cout << "Begining neuron is: " << vinputneurons[p] << endl;

                cout << "Ending neuron is: " << voutputneurons[q] << endl;

                 cout << "Path length is: " << shortestPath(g, (*(g.vInfo[find(g,
vinputneurons[p])].vtxMapLoc)).first, (*(g.vInfo[find(g, voutputneurons[q])].vtxMapLoc)).first, path) <<
endl;

                cout << "Path is: ";

                writeList(path, "  ");

                cout << endl;

                vpaths.push_back(path);

        }


vector<list<string> > vsubpaths;


// Generate all of the subpaths

int vpathsize = vpaths.size();

for(int i=0; i < vpathsize; ++i)

        subpaths(vpaths[i], vsubpaths); // subpath algorithm used here
```

```
	cout << "Contents of vsubpaths: " << endl;
	// foreach subpath, find all paths that go through that subpath
	int vsubpathsize = vsubpaths.size();
	for(int i = 0; i < vsubpathsize; ++i)
		writeList(vsubpaths[i], " ");

	for(int i = 0; i < vsubpathsize; ++i)
	{
		cout << "Subpath is: ";
		writeList(vsubpaths[i], " ");
		cout << endl;
		cout << "Paths going through this subpath are: " << endl;
		for(int j=0; j < vpathsize; ++j)
			if(sublist(vsubpaths[i], vpaths[j]))
				writeList(vpaths[j], " ");
		cout << endl << endl;
	}

	graphname.close();
	inputneurons.close();
	outputneurons.close();
	neuronpaths.close();


	return 0;
}
```