AI Assignment 1

Ritabroto Ganguly
001910501090
BCSE-III A3

```
#include <iostream>
#include <utility>
#include "bfs.cpp"
#include "dfs.cpp"
#include "utils.cpp"
using namespace std;

int main(){
	vector<pair<int, vector<int> > > graph;
	int inp = 0;
	cout<<"Enter the total number of vertices"<<endl;
	cin>>inp;
	cout<<"Vertices are numbered from 0"<<endl;
	int vertex = 0;
	while(vertex<inp){
		cout<<"Enter the value for vertex "<<vertex<<endl;
		int val;
		cin>>val;
		pair<int, vector<int> > p;
		p.first = val;
		p.second = vector<int>();
		graph.push_back(p);

		cout<<"Enter the number of edges for vertex
"<<vertex<<endl;
		int edges;
		cin>>edges;

		if(edges>0)
			cout<<"Enter the edgelist for vertex "<<vertex<<endl;
		for(int i=0;i<edges;i++){
			int edge;
			cin>>edge;
			graph[vertex].second.push_back(edge);
```

```
            }
            vertex++;
        }
        printGraph(graph);

        cout<<"Search value: ";
        int val;
        cin>>val;
        cout<<"1. BFS\n2. DFS\n3. Depth Limited Search\n4. Iterative
deepening search\n5. Iterative broadening search\n*. Exit"<<endl;
        int opt = 0;
        cin>>opt;
        vector<int>* traversal;
        while(opt>0 && opt<6){
            int limit;
            switch(opt){
                case 1: traversal = bfs(graph, val);break;
                case 2: traversal = dfs(graph, val);break;
                case 3: cout<<"Enter limit:";
                    cin>>limit;
                    traversal = dfs(graph, val, limit);break;
                case 4: cout<<"Enter max depth of the graph/tree(src/
root node depth is 0): ";
                   cin>>limit;
                    traversal = ids(graph, val, limit);break;
                case 5: cout<<"Enter max no. of children a node can
have in the graph/tree: ";
                    cin>>limit;
                    traversal = ibs(graph, val, limit);break;
            }

            if(traversal[0].empty())
                cout<<"Not found"<<endl;
            else{
                cout<<"Found at vertex "<<traversal[1]
[traversal[1].size()-1]<<endl;
                cout<<"Path:"<<endl;
                printVector(traversal[0]);
                cout<<"Order:"<<endl;
                printVector(traversal[1]);
            }
            delete[] traversal;//cleanup
```

```
            cout<<"Search value: ";
            cin>>val;
            cout<<"1. BFS\n2. DFS\n3. Depth Limited Search\n4.
Iterative deepening search\n5. Iterative broadening search\n*.
Exit"<<endl;
            cin>>opt;
        };

        return 0;
}
```

bfs.cpp

```
#include <vector>
#include <queue>
using namespace std;

vector<int>* bfs(vector<pair<int, vector<int> > >& graph, int val, int
breadth = -1){
        queue<int> q;
        q.push(0);
        vector<int>* traversal = new vector<int>[2];
        int order[graph.size()];
        order[0] = 0;
        bool visited[graph.size()];
        visited[0] = true;
        for(int i=1;i<graph.size();i++)
                visited[i] = false;
        bool flag = false;

        while(!q.empty()){
                int v = q.front();
                q.pop();
                traversal[1].push_back(v);

                if(graph[v].first == val){
                        flag = true;
                        break;
                }
```

```
            if(breadth==-1)
                breadth = graph[v].second.size();
            else
                breadth = breadth > graph[v].second.size() ?
graph[v].second.size() : breadth;

            for(int i=0;i<breadth;i++){
                int ind = graph[v].second[i];
                //cout<<v<<" "<<graph[v][i]<<" "<<visited[graph[v]
[i]]<<endl;
                if(visited[ind]==false){
                    visited[ind] = true;
                    order[ind] = v;
                    q.push(ind);
                }
            }
        }

        if(flag==false)
            traversal[0].clear();
        else{
            int ind = traversal[1][traversal[1].size()-1];
            while(ind!=0){
                traversal[0].push_back(ind);
                ind = order[ind];
            }
            traversal[0].push_back(0);
            reverse(traversal[0].begin(), traversal[0].end());
        }

        return traversal;
}

vector<int>* ibs(vector<pair<int, vector<int> > >& graph, int val, int
maxBreadth){
        vector<int>* traversal = new vector<int>[2];

        for(int i = 0; i <= maxBreadth; i++){
            auto temp = bfs(graph, val, i);
            if(!temp[0].empty()){
                traversal[0] = temp[0];
```

```
                    traversal[1].insert(traversal[1].end(), temp[1].begin(),
temp[1].end());
                    delete[] temp;
                    break;
            }else{
                    traversal[1].insert(traversal[1].end(), temp[1].begin(),
temp[1].end());
                    delete[] temp;
            }
        }

        return traversal;
}
```

dfs.cpp

```
#include <vector>
#include <queue>
using namespace std;

void dfs(vector<pair<int, vector<int> > >& graph, int root, bool* v,
vector<int>* traversal, int val, bool* flag, int limit = -1, int level = 0){
        if(limit>-1 && level>limit)
                return;

        v[root] = true;
        traversal[1].push_back(root);

        if(graph[root].first != val){
                for(int& i : graph[root].second){
                        if(v[i]==false){
                                dfs(graph, i, v, traversal, val, flag, limit, level+1);
                                if(*flag==true){
                                        traversal[0].push_back(i);
                                        return;
                                }
                        }
                }
        }else
                *flag = true;
}
```

```
vector<int>* dfs(vector<pair<int, vector<int> > >& graph, int val, int limit
= -1){
	vector<int>* traversal = new vector<int>[2];
	bool v[graph.size()];
	for(int i = 0; i < graph.size(); i++)
		v[i] = false;

	bool* flag = new bool();
	*flag = false;
	dfs(graph, 0, v, traversal, val, flag, limit);
	if(*flag==false){
		traversal[0].clear();
	}
	else{
		traversal[0].push_back(0);
		reverse(traversal[0].begin(), traversal[0].end());
	}
	delete flag;

	return traversal;
}

vector<int>* ids(vector<pair<int, vector<int> > >& graph, int val, int
maxDepth){
	vector<int>* traversal = new vector<int>[2];

	for(int i = 0; i <= maxDepth; i++){//max depth can be graph.size() if
maxDepth passing is not allowed
		auto temp = dfs(graph, val, i);
		if(!temp[0].empty()){
			traversal[0] = temp[0];
			traversal[1].insert(traversal[1].end(), temp[1].begin(),
temp[1].end());
			delete[] temp;
			break;
		}else{
			traversal[1].insert(traversal[1].end(), temp[1].begin(),
temp[1].end());
			delete[] temp;
```

```
            }
        }

        return traversal;
}
```

utils.cpp

```
#include<iostream>
using namespace std;

void printVector(vector<int>& v){
        for(int i : v)
                cout<<i<<" ";
        cout<<endl;
}

void printGraph(vector<pair<int, vector<int> > >& v){
        for(int i=0;i<v.size();i++){
                cout<<i<<" val:"<<v[i].first<<endl;
                printVector(v[i].second);
        }
}
```

```
Enter the total number of vertices
4
Vertices are numbered from 0
Enter the value for vertex 0
0
Enter the number of edges for vertex 0
2
Enter the edgelist for vertex 0
1 2
Enter the value for vertex 1
1
Enter the number of edges for vertex 1
1
Enter the edgelist for vertex 1
3
Enter the value for vertex 2
2
Enter the number of edges for vertex 2
1
Enter the edgelist for vertex 2
3
Enter the value for vertex 3
3
Enter the number of edges for vertex 3
0
0 val:0
1 2
1 val:1
3
2 val:2
3
3 val:3
```

```
Search value: 3
1. BFS
2. DFS
3. Depth Limited Search
4. Iterative deepening search
5. Iterative broadening search
*. Exit
1
Found at vertex 3
Path:
0 1 3
Order:
0 1 2 3
Search value: 3
1. BFS
2. DFS
3. Depth Limited Search
4. Iterative deepening search
5. Iterative broadening search
*. Exit
2
Found at vertex 3
Path:
0 1 3
Order:
0 1 3
Search value: 3
1. BFS
2. DFS
3. Depth Limited Search
4. Iterative deepening search
5. Iterative broadening search
*. Exit
```

```
3
Enter limit:2
Found at vertex 3
Path:
0 1 3
Order:
0 1 3
Search value: 3
1. BFS
2. DFS
3. Depth Limited Search
4. Iterative deepening search
5. Iterative broadening search
*. Exit
4
Enter max depth of the graph/tree(src/root node depth is 0): 3
Found at vertex 3
Path:
0 1 3
Order:
0 0 1 2 0 1 3
Search value: 3
1. BFS
2. DFS
3. Depth Limited Search
4. Iterative deepening search
5. Iterative broadening search
*. Exit
5
Enter max no. of children a node can have in the graph/tree: 2
Found at vertex 3
Path:
0 1 3
Order:
0 0 1 3
Search value: 2
```

```
1. BFS
2. DFS
3. Depth Limited Search
4. Iterative deepening search
5. Iterative broadening search
*. Exit
0
```