GIKI Map Using Graph and Vectors

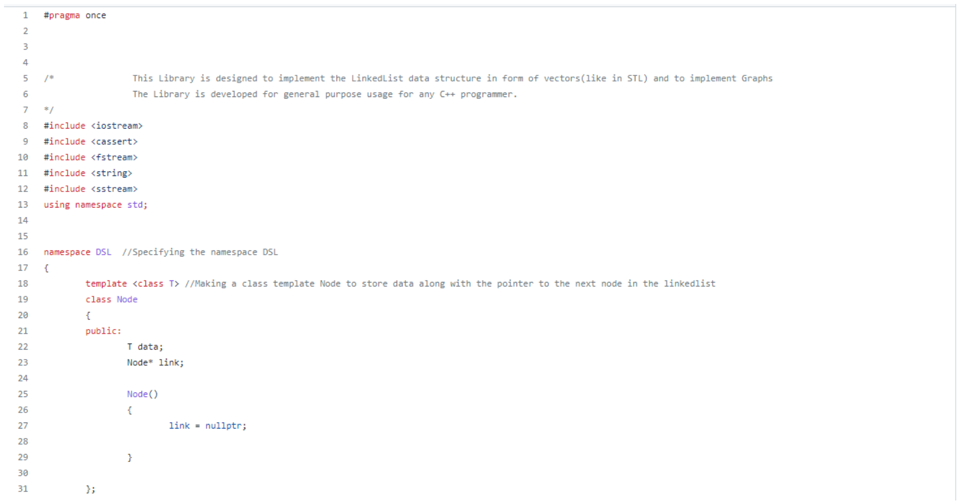
Project Proposal

The goal of this project is to implement the map of GIKI using vector and graphs data structures. Different locations of GIKI are stored in form of a graph, the nodes are connected with edges and cost is the distance between two points. We used Uniform Cost Search to find the path between two points with the lowest cost.

Methodology

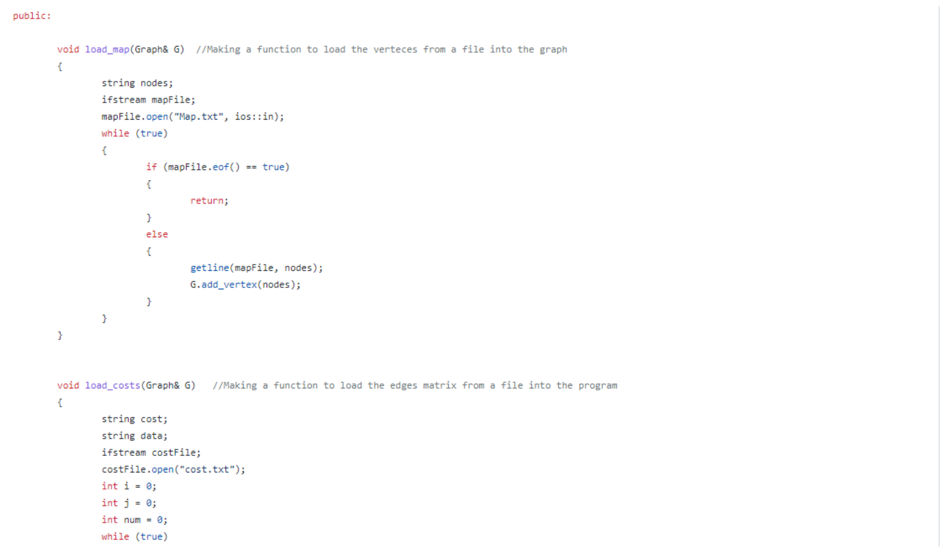
We created a header file DSL.h, that contain the data structures used in our application. Vector and Graph data structures are implemented in DSL.h that are used throughout the application. We created a file called cost.txt, that contains a matrix of costs calculated by measuring the distance between two locations given by Google Maps. We created a header file FileIO.h, that contain functions load_map() and load_cost() to initialize the location to the graph nodes and initialize the cost matrix in the file to a 2D array in our application. We created a header file PathFinder.h and implemented the Uniform Cost Search Algorithm that finds the shortest path between two locations. After that we created a header file called GIKIGuide.h to implement all the functionality in a single function that we called in main.cpp.

DSL.h



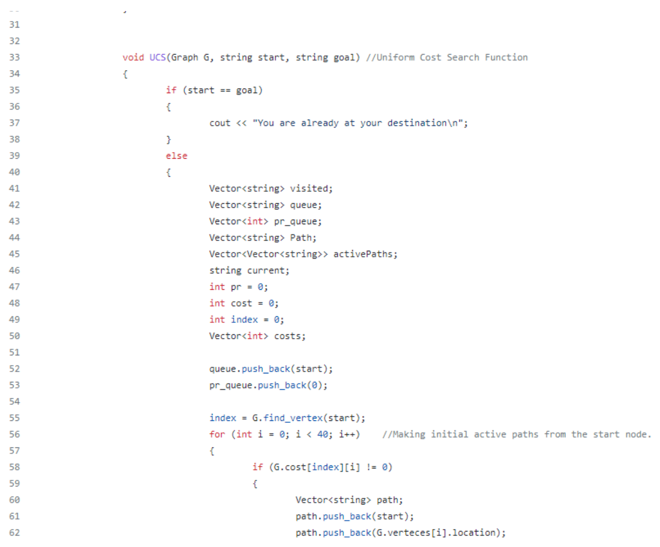
```
#pragma once



/*              This Library is designed to implement the LinkedList data structure in form of vectors(like in STL) and to implement Graphs
                The Library is developed for general purpose usage for any C++ programmer.
*/
#include <iostream>
#include <cassert>
#include <fstream>
#include <string>
#include <sstream>
using namespace std;


namespace DSL  //Specifying the namespace DSL
{
        template <class T> //Making a class template Node to store data along with the pointer to the next node in the linkedlist
        class Node
        {
        public:
                T data;
                Node* link;

                Node()
                {
                        link = nullptr;

                }

        };
```

FileIO.h



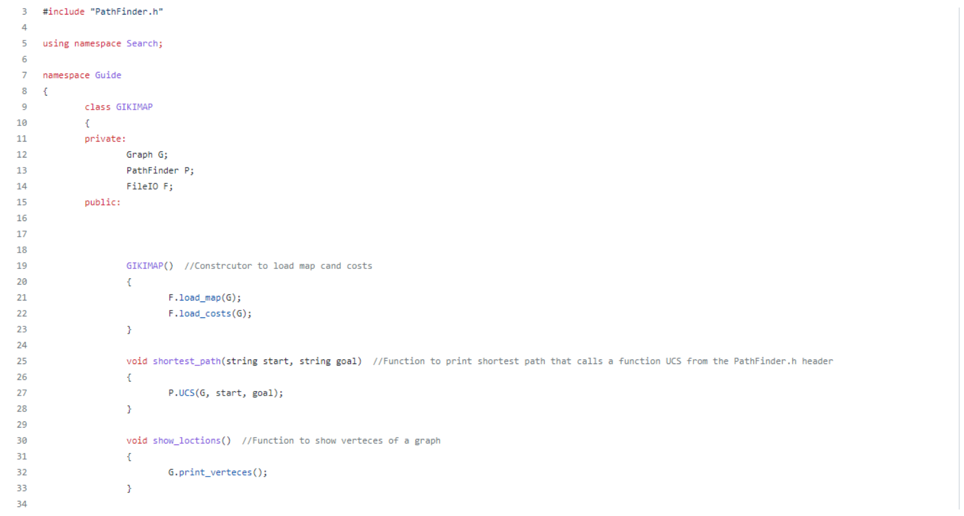
```
public:

    void load_map(Graph& G)  //Making a function to load the verteces from a file into the graph
    {
        string nodes;
        ifstream mapFile;
        mapFile.open("Map.txt", ios::in);
        while (true)
        {
            if (mapFile.eof() == true)
            {
                return;
            }
            else
            {
                getline(mapFile, nodes);
                G.add_vertex(nodes);
            }
        }
    }


    void load_costs(Graph& G)   //Making a function to load the edges matrix from a file into the program
    {
        string cost;
        string data;
        ifstream costFile;
        costFile.open("cost.txt");
        int i = 0;
        int j = 0;
        int num = 0;
        while (true)
```

PathFinder.h



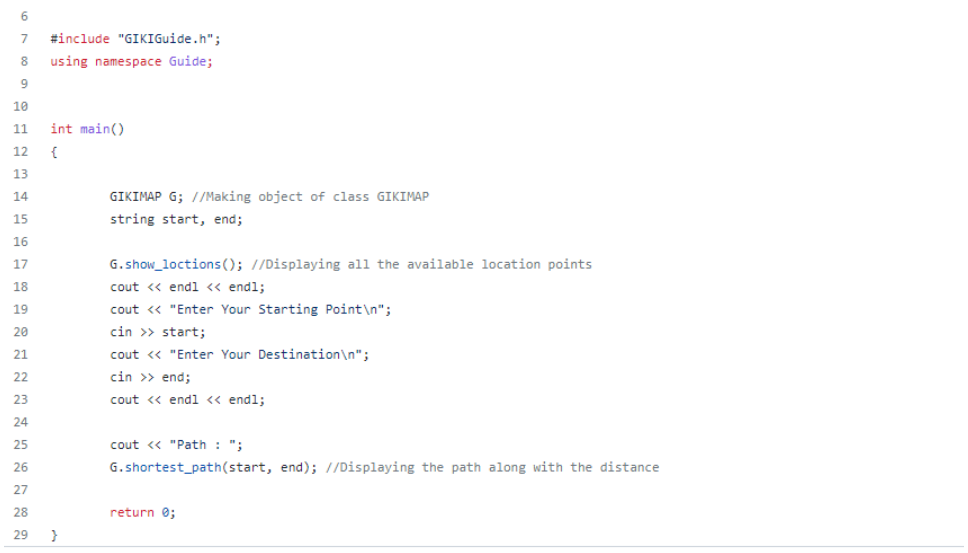
```


    void UCS(Graph G, string start, string goal) //Uniform Cost Search Function
    {
        if (start == goal)
        {
            cout << "You are already at your destination\n";
        }
        else
        {
            Vector<string> visited;
            Vector<string> queue;
            Vector<int> pr_queue;
            Vector<string> Path;
            Vector<Vector<string>> activePaths;
            string current;
            int pr = 0;
            int cost = 0;
            int index = 0;
            Vector<int> costs;

            queue.push_back(start);
            pr_queue.push_back(0);

            index = G.find_vertex(start);
            for (int i = 0; i < 40; i++)    //Making initial active paths from the start node.
            {
                if (G.cost[index][i] != 0)
                {
                    Vector<string> path;
                    path.push_back(start);
                    path.push_back(G.verteces[i].location);
```

GIKImap.h



```cpp
#include "PathFinder.h"

using namespace Search;

namespace Guide
{
	class GIKIMAP
	{
	private:
		Graph G;
		PathFinder P;
		FileIO F;
	public:



		GIKIMAP()  //Constrcutor to load map cand costs
		{
			F.load_map(G);
			F.load_costs(G);
		}

		void shortest_path(string start, string goal)  //Function to print shortest path that calls a function UCS from the PathFinder.h header
		{
			P.UCS(G, start, goal);
		}

		void show_loctions()  //Function to show verteces of a graph
		{
			G.print_verteces();
		}

```

Main.cpp



```cpp

#include "GIKIGuide.h";
using namespace Guide;


int main()
{

	GIKIMAP G; //Making object of class GIKIMAP
	string start, end;

	G.show_loctions(); //Displaying all the available location points
	cout << endl << endl;
	cout << "Enter Your Starting Point\n";
	cin >> start;
	cout << "Enter Your Destination\n";
	cin >> end;
	cout << endl << endl;

	cout << "Path : ";
	G.shortest_path(start, end); //Displaying the path along with the distance

	return 0;
}
```