**Project 4**

**导游咨询与最短路径**

班级： 2015级教务三班

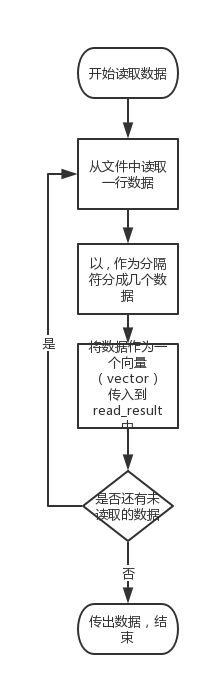
姓名： 罗剑杰 学号：15331229

姓名： 卢健彬 学号：15331225

姓名： 刘俊君 学号：15331214

**【题目要求】**

用图形结构实现一个为游客提供中山大学东校区的地图

**【数据结构与算法】**

1. **dataHelper:实现对储存的地点的数据的读取。在此前的几次项目中，我们小组发现基本上每次都需要用到文件流的读写操作，因此我们尝试实现一个基础类，可以方便实现以后的Project的数据文件流读写（数据使用csv格式储存）。结构如下：**

**#ifndef DATAHELPER\_HPP\_**

**#define DATAHELPER\_HPP\_**

**#include <fstream>**

**#include <string>**

**#include <vector>**

**class dataHelper {**

**public:**

**dataHelper() : m\_data\_path("./data/data.csv") {}**

**dataHelper(const std::string t\_data\_path) : m\_data\_path(t\_data\_path) {}**

**~dataHelper() {}**

**/\*\***

**\* get the data from the data.csv file**

**\* return a vector with each spot info in a line**

**\*/**

**std::vector<std::string> getData();**

**bool writeSpotData(const std::vector<std::string> t\_info);**

**std::vector<std::string> splitEachInfo(std::string t\_spotInfo);**

**private:**

**const std::string m\_data\_path;**

**};**

**#endif**

1. **Node:是图中的节点，有id、名字、推荐指数、展示信息等功能和信息。结构如下：**

**struct Node {**

**int id;**

**std::string name;**

**int score;**

**std::string description;**

**Node(int \_id = 0, std::string \_n = "", int \_s = 0, std::string \_d = ""): id(\_id), name(\_n), score(\_s), description(\_d) {}**

**void showInfo() const;**

**};**

1. **graph：是Project的核心，实现了对地图的数据的管理、查找地点、推荐路线功能。图采用了邻接矩阵的方式储存。查找地点功能中，除了会根据用户输入的地点返回到达查询地点的路途时间，还会输出查询地点的推荐指数和简短介绍。推荐路线中有所创新，实现的结果是把所有路径输出，然后第一条是最短路径，用户可以根据自己想要浏览的景点数目选择路线，并且在两个功能中都实现了对无效输入的处理功能。结构如下：**

**#ifndef GRAPH\_HPP\_**

**#define GRAPH\_HPP\_**

**#include <string>**

**#define MAX 50**

**class Graph {**

**private:**

**int m\_vertexNum;**

**int m\_edgeNum;**

**Vertex vertex[MAX];**

**int edges[MAX][MAX];**

**// bool visited[MAX];**

**public:**

**Graph(std::string spotInfoPath, std::string edgeInfoPath);**

**~Graph();**

**bool querySpot(const std::string t\_spot\_name) const;**

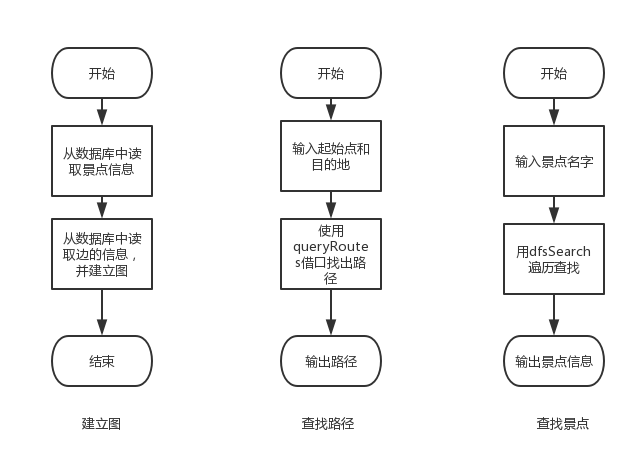
**void queryRoutes(const std::string t\_start, const std::string t\_end) const;**

**// just for test apoi**

**// const Vertex\* getVertexs() const { return vertex; }**

**private:**

**void createGraph(std::string spotInfoPath, std::string edgeInfoPath);**

 **void dfsSearch(const std::string t\_spot\_name, int index, bool\* visisted, bool& flag) const;**

**};**

**#endif**

**【测试数据，结果及分析】**

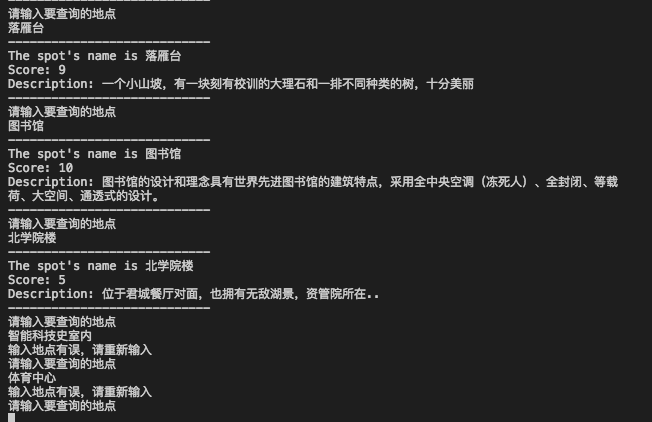
测试数据及分析：经测试得图的地点查询和路径查询功能正常

展示景点：



图[1]

实现地点查询功能:



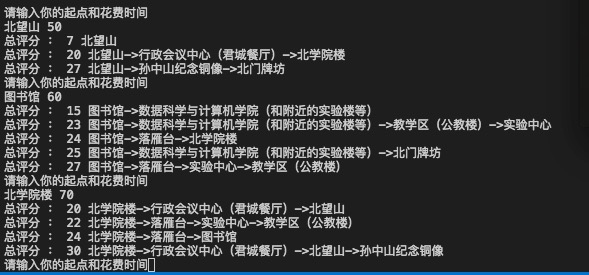
图[2]

实现树的打印



图[3]

通过背包问题的算法，输入起始点和想要游玩的时间，给出可以实现的推荐的路径：



图[4]

**【分工、贡献%、自我评分】**

刘俊君：queryRoutes , queryAllRoute方法的实现与优化， 实验报告书写

卢健彬：景点数据，图边数据的构造，ui的部分实现

罗剑杰：负责项目框架构成，dataHelper类的实现，Graph类的实现（出去查询路径的函数实现），makefile构造，ui实现

**贡献%： 刘俊君35%，卢健彬30%，罗剑杰：35%**

自我评分：90

**【项目总结】**

**完成情况：实现了功能查询的基本功能**

**开发过程中的优点及收获：**

1. 这次的思路跟想法比上次要清晰，所以做起来所花的时间和精力也更为轻松
2. 对项目的管理尤其是github的使用比项目刚开始的时候好了很多，引入了unittest文件夹，不同的类之间分开开发，每个人负责好自己的部分，互不影响
3. 小组内部拓展了一个以时间作为限制的路线推荐功能queryAllRoute
4. 分工相当明确

**开发过程中的不足及以后要改进的：**

1. Ui太丑了= =，自身实力不够不能够做成web或者图形化界面的外表。应当争取基础功能的代码更为简洁、高效。
2. 在算法选择方面，使用的dfs遍历的方法，这个算法在数据量巨大的时候查询的速度就会降低，而且输出的路径信息就会很大。然而小组通过实际情况考虑，因为一开始这个景点信息已经确定，所以查询速度不会很慢，而且输出的路径数不多，全部展示出来有利于用户自己去选择，给予了按照路途时间的排序给用户进行参照。

**【程序清单】**

(因为开发的时候小组内部分开并行对各个功能开发，所以有好几个unittest文件)

├── 20121112045416099140.jpg

├── Makefile

├── Readme.md

├── data

│   ├── edgeCar.csv

│   ├── edgePace.csv

│   └── spot.csv

├── include

│   ├── dataHelper.hpp

│   └── graph.hpp

├── src

│   ├── dataHelper.cpp

│   └── graph.cpp

└── unittest

├── dataHelper\_unittest

│   ├── Makefile

│   ├── data

│   │   └── data.csv

│   ├── include

│   │   └── dataHelper.hpp

│   └── src

│   ├── dataHelper.cpp

│   └── main.cpp

├── data\_

│   ├── edgeCar.csv

│   ├── edgePace.csv

│   └── spot.csv

├── graph\_unittest

│   ├── Makefile

│   ├── data

│   │   ├── edgeCar.csv

│   │   ├── edgePace.csv

│   │   └── spot.csv

│   ├── include

│   │   ├── dataHelper.hpp

│   │   └── graph.hpp

│   └── src

│   ├── dataHelper.cpp

│   ├── graph.cpp

│   └── unittest.cpp

└── queryRoute\_unittest

├── Makefile

├── bin

│   └── unittest

├── build

│   ├── dataHelper.o

│   ├── graph.o

│   └── unittest.o

├── data

│   ├── edgeCar.csv

│   ├── edgePace.csv

│   └── spot.csv

├── include

│   ├── dataHelper.hpp

│   └── graph.hpp

└── src

├── dataHelper.cpp

├── graph.cpp

└── unittest.cpp