数据结构与算法实验报告-广州地铁线路查询

18324034 林天皓 linth5@mail2.sysu.du.cn

2020.12.26

摘要

本次数据结构与算法实验根据题目需求完成了一个广州地铁线路查询程序，具有展示地铁各条线路信息，根据汉字，编号或者模糊拼音选择起点与终点，通过 windows 窗口展示得到的具有最短途径站点数量以及最少换乘次数的地铁乘车线路信息。

引言

随着广州市的地铁站点数量和线路数量越来越多，乘客该如何乘坐地铁到达目的地的查询越来越重要，同时为了考虑较多站点的换乘，我们需要完成快速查找地铁乘车方式的程序，给与用户多种选择，完成用户查询乘坐地铁方式的目的。

解决方法

1. 数据输入的储存格式

在本次实验中，通过 data_in 文件储存地铁的所有线路信息，通过将输入文件与代码分离的方式，可以在地铁线更新后，只需要修改对应的输入文件即可，无需改动具体的程序逻辑。我储存地铁信息文

件的格式如下：

```
270 16
1   西塱    1   1号线   xilang
2   坑口    1   1号线   kangkou
3   花地湾  1   1号线   huadiwan
4   芳村    1   1号线   fangcun
5   黄沙    1   1号线   huangsha
6   长寿路  1   1号线   changshoulu
7   陈家祠  1   1号线   chenjiasi
8   西门口  1   1号线   ximenkou
9   公园前  1   1号线   gongyuanqian
10  农讲所  1   1号线   nongjiangsuo
11  烈士陵园    1   1号线   lieshilingyuan
12  东山口  1   1号线   dongshankou
13  杨箕    1   1号线   yangji
14  体育西路    1   1号线   tiyuxilu
15  体育中心    1   1号线   tiyuzhongxin
16  广州东站    1   1号线   guangzhoudongzhan
17  广州南站    2   2号线   guangzhounanzhan
18  石壁    2   2号线   shibi
19  会江    2   2号线   huijiang
```

图 1-数据储存格式

第一行为两个数字 n,m，分别代表地铁站点数量和地铁线路数量

后面 n 行，每一行分别存储了站点全局编号，站点中文名称，站点所属线路的全局编号，站点所属线路的中文名称，站点无声调的汉语拼音。

2. 换乘站点的处理

图 2-大学城南站-百度地图截图

可见，虽然是同一个站点，但是在百度地图上显示为两个地铁站。对应的，我在这里将一个站点分为多个伪站点，分别属于不同的地铁线路，通过这样的实现，在处理地铁边相连通的权值的时候更好处理，例如考虑站点数量最少的时候将这样的换乘站点的边权设置为零来计算最短路径，而考虑换乘数量最少的时候，将同一条线路上的站点的边权设置为零，换乘站点反而设置为一个正数，从而实现计算最短路就是换乘数量最少的功能。

3. 模糊拼音查询功能

在测试过程中，发现如果让用户使用输入站点的全局编号或者使用站点的全称来输入站点并不方便，输入麻烦而且很容易出错，同时采用鼠标点击的方式来选择站点在整个广州密密麻麻的地铁线路中选中想要的正确站点也不方便。因此，我实现了一种通过汉语拼音模糊查询的功能，可以通过输入站点的模糊拼音来选择站点。

如大学城南站点的汉语拼音为 daxuechengnan，采用模糊匹配算法，当用户输入 dxcn 或 daxuecn 等缩写字符串的时候，同样可以选择想要的地铁站点。通过展示所有模糊匹配的站点让用户进一步选择，可以达到更加方便的选择站点的效果。

4. 部分代码分析

本次实验中使用两个类，分别为 subway_query 类和 GUI 类。通过这种逻辑处理与显示分离的方式，可以更好的降低代码的耦合性，使得代码的可迁移性更强。所以是通过一开始实现的在命令行中运行的

程序通过一些少许修改就迁移到了基于 windows 窗口的图形化界面上，而且保持了交互逻辑的一致性。

基础结构类型(station,edge,line 结构体较为简单已经添加注释，不再赘述）代码如下

```
struct station
{
    //站点全局编号
    int num;
    //站点所属线路编号
    int line;
    //站点中文名称
    string name;
    //站点汉语拼音 无声调
    string pinyin;
    //站点所属线路的名称
    string line_name;
};
struct subway_line
{
    //线路全局编号
    int num;
    //线路名称
    string name;
    //线路起始站点的全局编号
    int start_station_num;
    //线路终末站点的全局编号
    int end_station_num;
};
struct edge
{
    //边的起始站点编号
    int from;
    //边的终止站点编号
    int to;
    //边的权重， 根据情况有可能是站点的数量，有可能是站点的换乘数
    int weight;
};
```

subway_query 类:负责处理具体的查询问题，并且生成结果字符串传递给 GUI 类的来展示结果。具体定义代码如下，各种变量已经加注释

```
class subway_query
{
    //储存地铁图的邻接表
    vector<vector<edge>> g;
    //储存地铁总共有多少条地铁站点
    int stations_num;
    //储存地铁总共有多少条地铁线路
    int lines_num;
    //储存各个地铁站店的站点信息
    vector<station> stations;
    //储存各个地铁线路的线路信息
    vector<subway_line> lines;

public:
    //初始化
    void init_subway();
    //清除图中的边
    void clear_edge();
    //添加换乘的站点，权值计算方式为站点数量
    void add_cross_station();
    //添加同一条线路上的站点,权值计算方式为站点数量
    void add_normal_station();
    //添加换乘的站点，权值计算方式为换乘数量
    void add_huancheng_cross_station();
    //添加同一条线路上的站点,权值计算方式为换乘数量
    void add_huancheng_normal_station();
    //按照最小途径站点数量获取最短路径，返回站点序列
    vector<station> get_shortest_path(string station_a, string st
ation_b);
    //按照最小途径站点数量获取最短路径的一个重载
    vector<station> get_shortest_path(int station_a, int station_
b);
    //按照最小换乘数量获取最短路径，返回站点序列
    vector<station> get_huancheng_shortest_path(string station_a,
 string station_b);
    //按照最小换乘数量获取最短路径的一个重载
    vector<station> get_huancheng_shortest_path(int station_a, in
t station_b);
    //创建返回给 gui 类的最短路结果的字符串对象，逻辑与现实分离
    string creat_result_path(vector<station> path);
    //创建返回给 gui 类的线路信息查询字符串对象，逻辑与现实分离
    string get_line_message(int line_num);
    //根据 string 查询对应的站点全局编号接口
    int getIndexByString(string station_string);
```

```
    //根据模糊的汉语拼音查询对应的站点全局编号接口
    vector<station> sta_mohu_query(string py);
};
```

GUI 类:负责呈现处理结果，并接收用户的输入数据与页面状态机跳转。

在该类中，实现了如下不同的状态页面的跳转（这部分类似于一个状态机），以及通过 windows 对话框的形式读取用户输入的数字或字符串，并且将查询返回的字符串显示到屏幕上完成信息的展示处理。

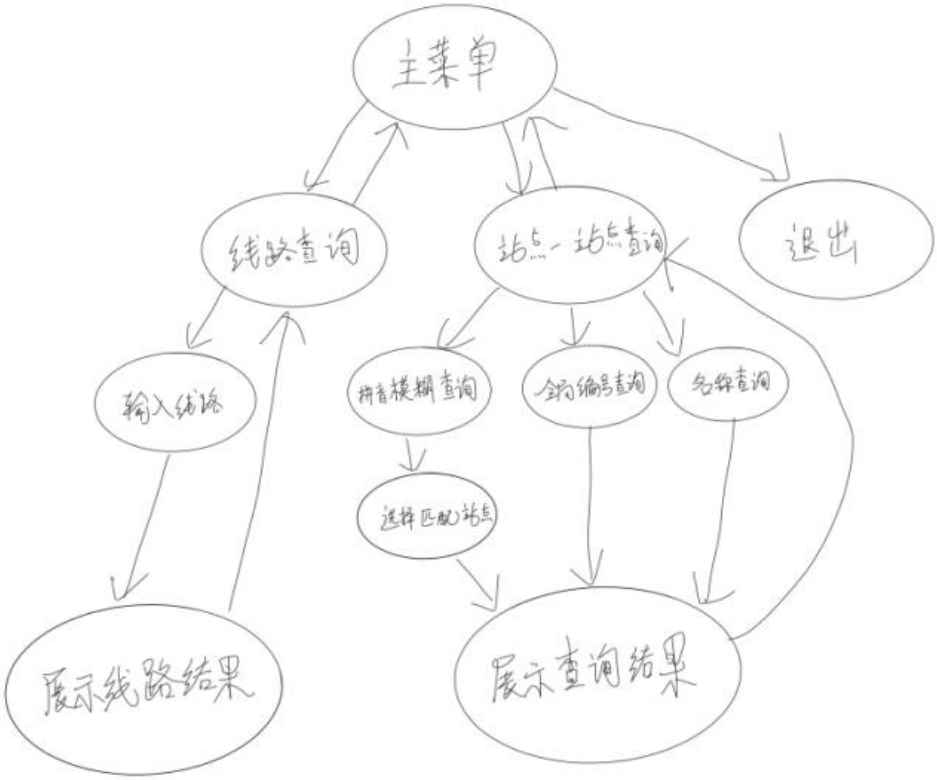

图 3-各个状态页面机的跳转关系

具体定义代码如下

```
class GUI
{
    //内部的查询结构
    subway_query t;

public:
```

```
    //初始化 gui
    void GUIinit();
    //gui 主循环
    void GUImain();
    //展示主页面菜单
    void showMainMenu();
    //展示线路信息查询页面菜单
    void show_line_queryMenu();
    //展示站点信息查询页面菜单
    void show_station_queryMenu();
    //清空当前页面显示的内容
    void clearMenu();
    //进入地铁线路信息查询页面
    void subway_line_query();
    //输出地铁线路信息查询
    void print_line_query_message(int line_num);
    //进入地铁站点-站点查询页面
    void subway_station_query();
    //暂停函数
    void delay();
    //打印错误信息
    void wrong_message();
    //根据站点的模糊汉语拼音查询
    void mohu_query();
    //根据站点编号查询
    void code_query();
    //根据线路的精确名称查询
    void name_query();
    //输出 string 信息到展示页面的接口
    void gui_print_msg(string);
    //从对话框获取输入的数字
    int get_int_input(string);
    //从对话框获取输入的字符串
    string get_string_input(string);
    //获取模糊选择的线路全局编号
    int mohu_select();
};
```

下面挑选其中部分函数具体分析：

站点最短路查找函数：

```
vector<station> subway_query::get_shortest_path(string station_a, strin
g station_b)
{
```

```
    clear_edge();
    add_cross_station();
    add_normal_station();
    vector<int> vis;
    vector<int> dis;
    vector<int> pre;
    int start = getIndexByString(station_a);
    int end = getIndexByString(station_b);
    //cerr << "start " << start << " end " << end << endl;
    vis.resize(stations_num + 1, 0);
    pre.resize(stations_num + 1, 0);
    dis.resize(stations_num + 1, 0x3f3f3f3f);
    vis[start] = 0;
    dis[start] = 0;
    pre[start] = -1;
    //int last_station=0;
    for (int i = 0; i < stations_num; i++)
    {
        //选取最近的点
        int now_station = 0;
        int now_mindis = 0x3f3f3f3f;
        for (int j = 1; j <= stations_num; j++)
        {
            if (vis[j] == 0 && dis[j] < now_mindis)
            {
                now_station = j;
                now_mindis = dis[j];
            }
        }
        if (now_station == 0)
        {
            //cerr << "now_station == 0 exit" << endl;
            break; //完成
        }
        //已经确认 优化
        vis[now_station] = 1;
        //cerr << "solve now_station = " << now_station << endl;
        for (int j = 0; j < g[now_station].size(); j++)
        {
            edge temp = g[now_station][j];
            if (dis[now_station] + temp.weight < dis[temp.to])
            {
                pre[temp.to] = now_station;
                dis[temp.to] = dis[now_station] + temp.weight;
```

```
            }
        }
    }
    int now_sta = end;
    vector<station> ret;
    while (now_sta != start)
    {
        ret.push_back(stations[now_sta]);
        now_sta = pre[now_sta];
    }
        ret.push_back(stations[now_sta]);

    // 反向
    for (int i = 0, j = ret.size() - 1; i < j; i++, j--)
    {
        swap(ret[i], ret[j]);
    }
    if(ret.size()>1 && ret[ret.size()-1].name == ret[ret.size()-
2].name){
        ret.pop_back();
    }
    return ret;
```

通过 dijkstar 算法查找最短路径,通过返回一个 vector<station> 以供后续判断是否有重复站点以及换乘处理,其他方法查找最短路径的只有边权不一样以及参数不同，此处不再赘述,。

通过站点路径生成结果字符串函数:

```
string subway_query::creat_result_path(vector<station> path)
{
    string ret;
    if(path.size()==0){
        return ret;
    }
    ret+="乘坐 ";
    ret+=path[0].line_name;
    ret+="\n";
    for(int i=0;i<path.size();i++){
        if(i!=0 && path[i].line != path[i-1].line ){
            ret+="\n";
```

```
            ret+="换乘到 ";
            ret+=path[i].line_name;
            ret+="\n";
        }

        ret+=path[i].name;
        if(i!=path.size()-1){
            ret+=" -> ";
        }
    }
    ret+='\n';
    return ret;
}
```

其中需要判断是否是同一个站点的换乘站，如果是换乘站则需要添加”换乘到:”字符串。

拼音模糊匹配函数

```
vector<station> subway_query::sta_mohu_query(string py)
{
    vector<station> ret;
    for(auto &sta:stations){
        int match_count=0;
        int py_cur=0;
        int sta_cur=0;
        //统计拼音匹配程度
        while(sta_cur<sta.pinyin.size() && py_cur<py.size()){
            if(py[py_cur]==sta.pinyin[sta_cur]){
                match_count++;
                py_cur++;
            }
            sta_cur++;
        }

        if(py.size() <3 && match_count>=py.size()-
1 || match_count==py.size()){
            ret.push_back(sta);
        }
    }
    return ret;
}
```

通过两个指针分别指向输入的模糊拼音和站点拼音，站点拼音按照顺序匹配模糊拼音，统计匹配程度，长度为 3 以下的要求完全匹配，长度为 3 以上的允许有一个匹配不到。

页面状态函数：

```
void GUI::GUImain()
{
    int opcode = 0;
    while (opcode != -1)
    {
        showMainMenu();
        int t;
        // cin >> t;
        t = get_int_input("请输入选择编号:");
        if (t == 1)
        {
            subway_line_query();
        }
        if (t == 2)
        {
            subway_station_query();
        }
        if (t == 3)
        {
            opcode = -1;
            string msg="再见！欢迎下次使用";
            gui_print_msg(msg);
        }
        else
        {
            opcode = 0;
        }
    }
}
```

上文为主菜单展示函数，通过控制 opcode 来决定是否结束循环退出程序，其他的页面状态类似，不再赘述。

其他函数请查看源代码。

程序使用和测试说明

在本次程序中，使用了 visual studio 平台，添加了 easyx 图形库（https://easyx.cn/），才能编译运行，所以为了方便读者想自行编译运行，提供了一个命令行版本的程序源代码，交互逻辑与显示结果与 gui 版本仅有较小差异。以下以 gui 版本作为使用演示的例子。

1. 主界面

图 4-主界面

具有选择进入线路查询页面，站点-站点查询页面，退出三种选择。

2. 线路查询

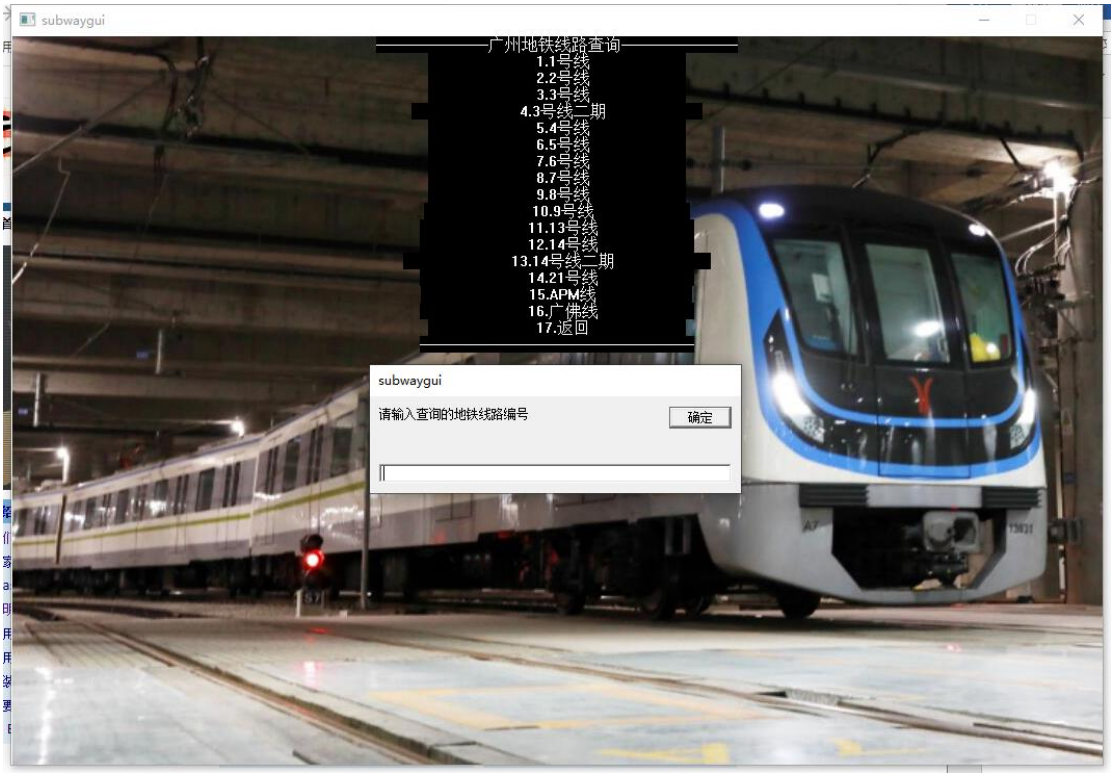

图 5-线路查询-选择线路页面

弹出对话框输入线路，具有选择线路和返回的功能页面。

图 6-线路查询-显示线路信息页面

显示线路信息页面，按任意键返回。

3. 站点站点查询

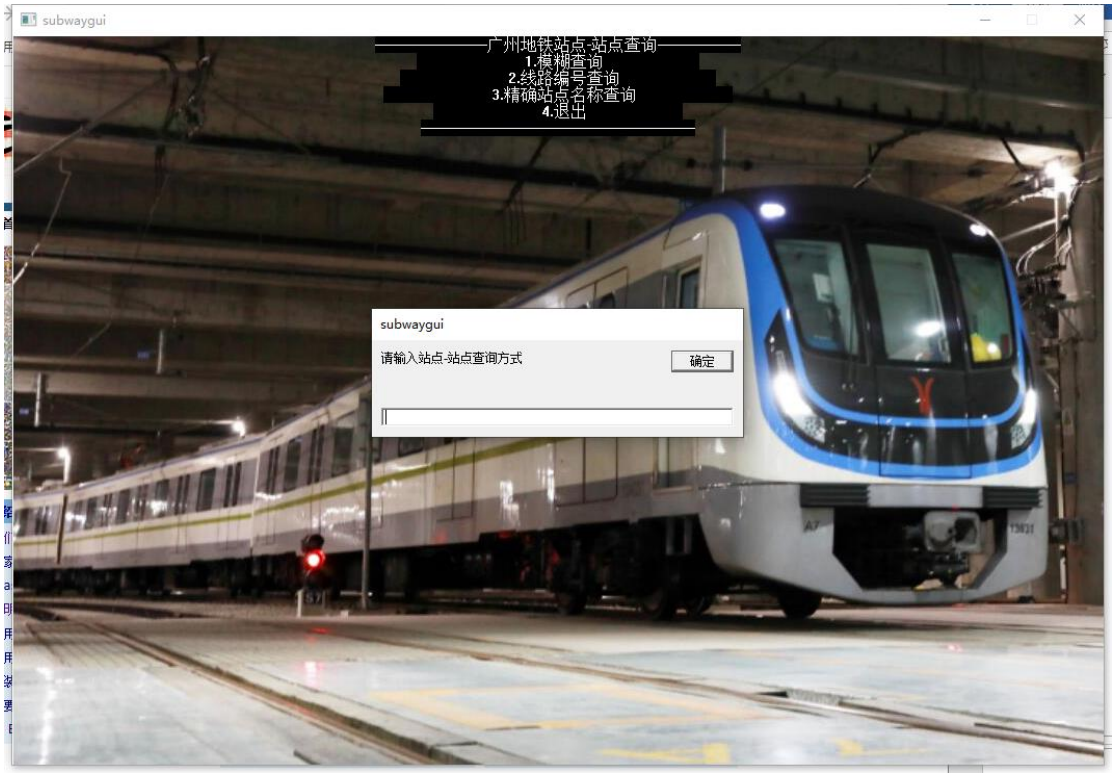

图 7-选择查询方式页面

方式 1：模糊拼音查询。

例如查询天平架-车陂南

输入 tpinj

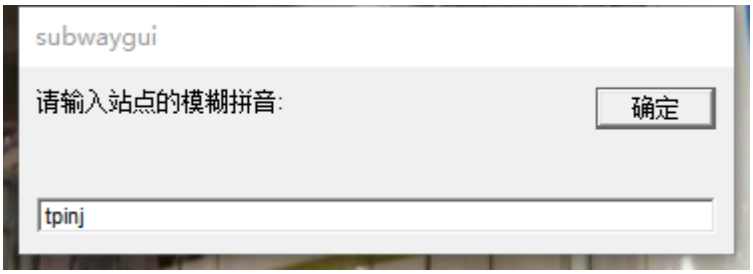

图 8-输入起始站模糊拼音

输入后出现候选站点页面，选择 0

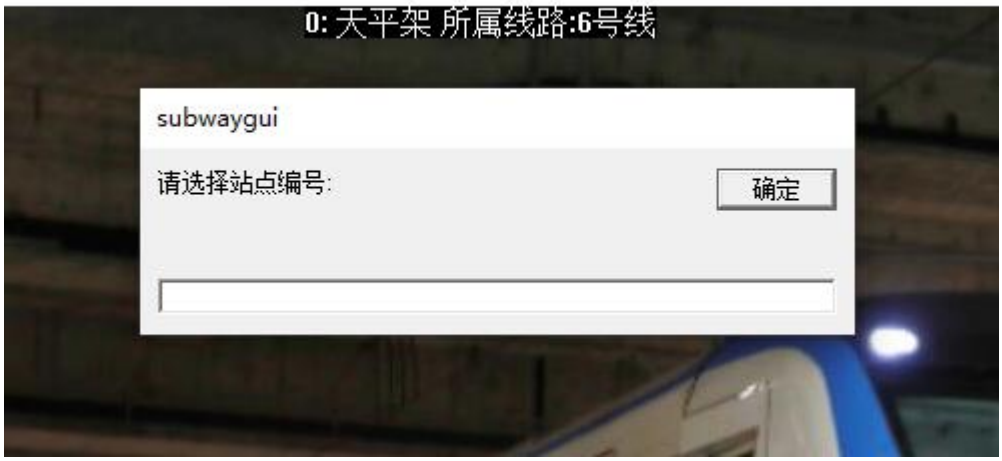

图 9-从备选列表中选择起始站

输入 cbn

图 10-输入起始站模糊拼音

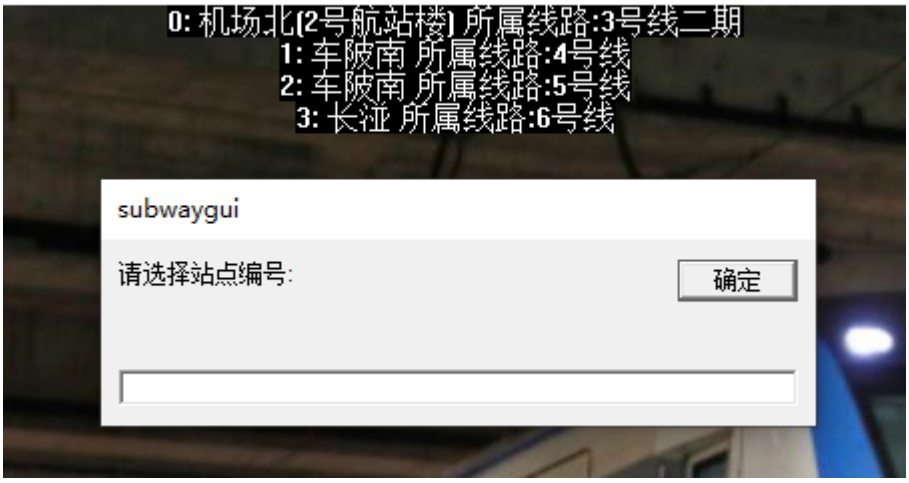

图 11-从备选列表中选择终点站

由于车陂南是换乘站点，所以根据我们的站点储存方式有两个站点,乘客选择距离自己最近的一条线路的站点即可,这里我们选择2。

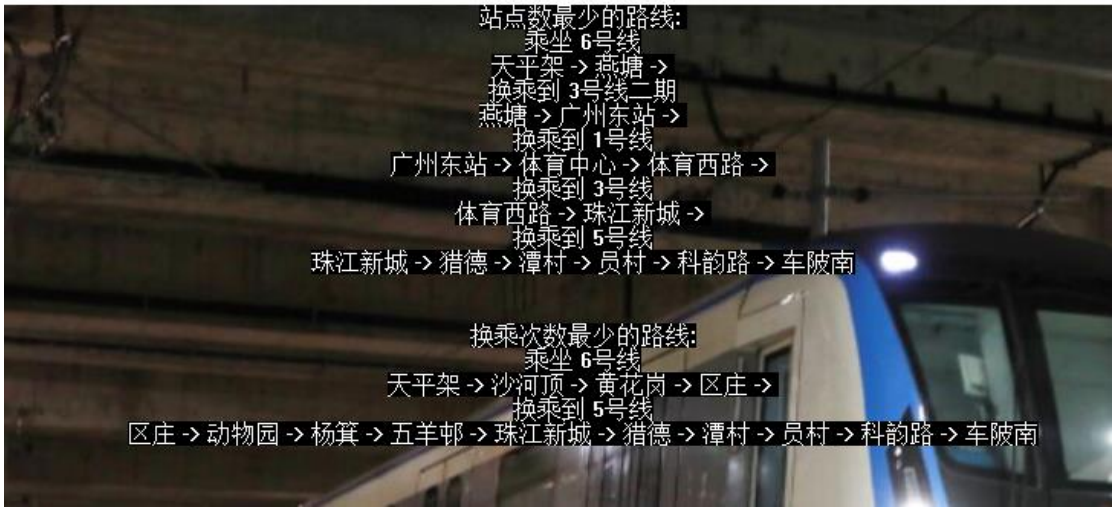

图 12-结果显示页面

可见站点最少的路线途径了 10 个站点，但是有 4 次换乘

而换乘数最少的路线途径了 12 个站点，但只是有 1 次换乘

此页面按任意键退出。

方式 2 与方式 3：全局编号查询与站点全称查询仅仅在输入的部分有不同，展示的结果相同，此处不再赘述。

4. 退出页

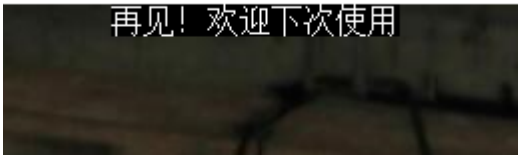

图 13-关闭页面

退出完成，按任意键关闭窗口。

总结与讨论:

本次实验中，实现了一个查询接口与显示分离，数据与逻辑分离的程序结构，为后续地铁信息的更改图形界面的改变提供了迁移的可能。在向窗口写入信息的时候，由于对 windows 窗口 API 不了解，在转换 C++语言中常见的 string 为 windos 窗口 API 中的 LPCWSTR 的过程中遇到了问题。通过查找资料发现这是各种编码格式的要求，采用 unicode 的时候 string 向 LPCWSTR 直接强制转换会发生错误，最终通过重新设置项目的编码解决了这个强制转换错误的问题。

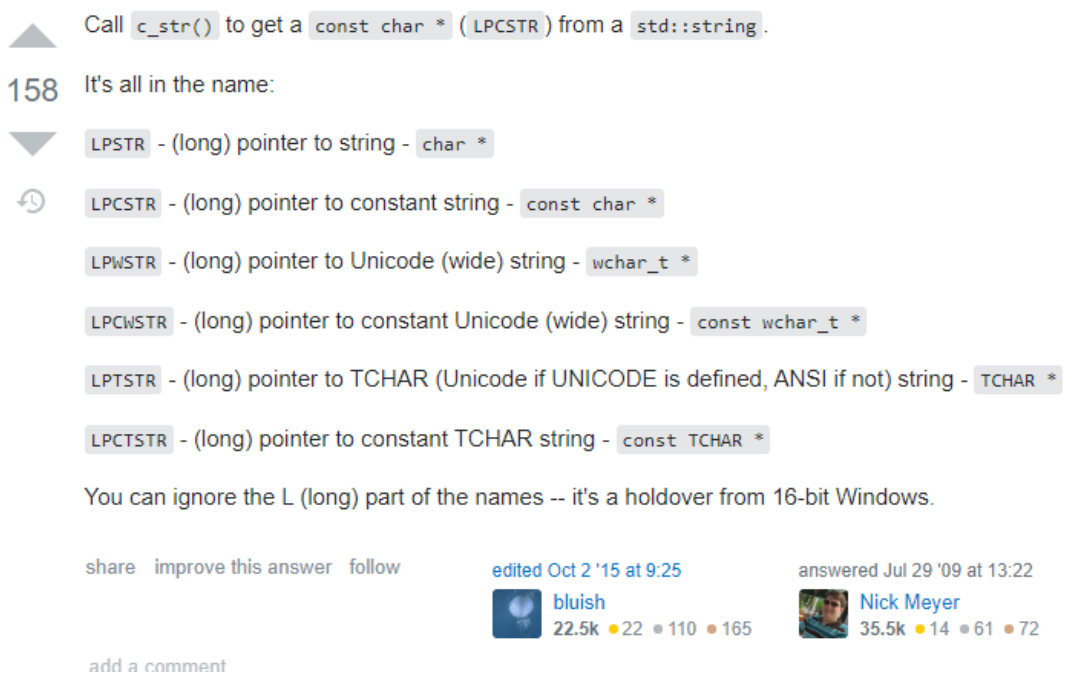

图 14-查找 windows 各种字符串的区别

来源:(https://stackoverflow.com/questions/1200188/how-to-convert-stdstring-to-lpcstr)

由于一开始实现的是命令行的版本，后来再进一步实现 gui 的版本，在这个从命令行模式迁移到 GUI 的实现中，并没有采用传统的鼠标点击的交互方式，因为地铁线路较多，站点密集，对于信息文字

的排布和显示要求较高，同时鼠标点击还需要设置按钮以及测量每个站点的所属范围，需要较大的工作量做一些重复的工作，最终没有使用鼠标点击的方式输入站点，而是添加了一个汉语拼音模糊选择的方式，这种方式帮助了用户快速的选择站点，查询乘车路线。

同时，在结果的展现方式上，还有很大的提升空间，文字的展现方式不如通过图形化的动态展示方式，将来可以进一步优化。

参考文献:

[1]乔海燕、蒋爱军，高集荣、刘晓铭 数据结构与算法实验实践教程 清华大学出版社，2012.

附录:

代码较多，代码与二进制均打包在压缩包内，不在此处列出。