山东大学 软件 学院

数据结构课程设计 课程实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 学号：201922301282 | 姓名： 李润泽 | | 班级： 软件工程20.1 |
| 实验题目：公交线路优化路径的查询 | | | |
| 实验学时：32 | | 实验日期：2022.2.22-2022.2.25 | |
| 实验目的：  掌握和强化上学期数据结构课上所学，提高自身独立解决问题的能力，增强编程能力。 | | | |
| 硬件环境：  *Dell inspiron 7590* | | | |
| 软件环境：  *Visual Studio 2019 Qt* | | | |
| 实验步骤与内容：  **一.问题描述与分析**  **1.1 问题描述**  最短路径问题是图论中的一个经典问题，其中的Dijkstra算法一直被认为是图论中的好算法，但有的时候需要适当的调整Dijkstra算法才能完成多种不同的优化路径的查询。  对于某城市的公交线路，乘坐公交的顾客希望在这样的线路上实现各种优化路径的查询。设该城市的公交线路的输入格式为：  线路编号：起始站名(该站坐标)；经过的站点1名(该站坐标)；经过的站点2名(该站坐标)；……；经过的站点n名(该站坐标)；终点站名(该站坐标)。该线路的乘坐价钱。该线路平均经过多少时间来一辆。车速。  例如：63：A(32,45)；B(76,45)；C(76,90)；……；N(100,100)。1元。5分钟。1/每分钟。  假定线路的乘坐价钱与乘坐站数无关，假定不考虑公交线路在路上的交通堵塞。  对这样的公交线路，需要在其上进行的优化路径查询包括：任何两个站点之间最便宜的路径；任何两个站点之间最省时间的路径等等。  基本要求  ① 根据上述公交线路的输入格式，定义并建立合适的图模型。  ② 针对上述公交线路，能查询获得任何两个站点之间最便宜的路径，即输入站名S，T后，可以输出从S到T的最便宜的路径，输出格式为：线路x：站名S，…，站名M1；换乘线路x：站名M1，…，站名M2；…；换乘线路x：站名MK，…，站名T。共花费x元。  ③ 针对上述公交线路，能查询获得任何两个站点之间最省时间的路径（不考虑在中间站等下一辆线路的等待时间），即输入站名S，T后，可以输出从S到T的考虑在中间站等下一辆线路的等待时间的最省时间的路径，输出格式为：线路x：站名S，…，站名M1；换乘线路x：站名M1，…，站名M2；…；换乘线路x：站名MK，…，站名T。共花费x时间。  ④ 针对上述公交线路，能查询获得任何两个站点之间最省时间的路径（要考虑在中间站等下一辆线路的等待时间），即输入站名S，T后，可以输出从S到T的考虑在中间站等下一辆线路的等待时间的最省时间的路径，输出格式为：线路x：站名S，…，站名M1；换乘线路x：站名M1，…，站名M2；…；换乘线路x：站名MK，…，站名T。共花费x时间。  (4) 实现提示  需深入考虑，应根据不同的应用目标，即不同的优化查询来建立合适的图模型。  **1.2 问题分析**  （1）输入数据：总的路线数目，各路线的站点数目，各站点的名称、位置坐标以及各公交路线的速度、间隔时间及票价。  （2）输出数据：输入数据的预处理结果以及各种不同需求下公交线路的结果。  （3）功能：实现在不同查询条件下最优公交线路的输出。  （4）存储结构: 链表、二维数组、优先队列  （5）核心算法：Dijkstra算法、floyd算法、A\*算法  首先需要对输入的数据进行预处理。  一开始想要使用双向链表进行站点和线路的存储和操作，但是因为双向链表增删改查都太过困难，并且公交线路也是从始发站到终点站或者从终点站到始发站，不会存在半路回头的可能性，所以最终决定采用单向链表进行存储  运用BusNode链表结构来存储各站点的名称、横纵坐标。  然后用一个公交车BusChain链表类来存储各个线路的票价、速度及总线路。  通过生成三种邻接矩阵或邻接表：按距离、按时间、按价钱  在最终邻接矩阵处理中主要运用Dijkstra算法。  在时间最少（两种）和距离最短查询时直接使用Dijkstra算法即可  在票价最少时需要对算法进行调整。因为坐公交车时在一条线路上一直乘坐只需要交一次车票钱因此需要线路上的任意两点之间都要加入边  **题目中提到Dijkstra算法一直被认为是图论中的好算法，寻找网上类似问题的解答也都采用了该算法。但是在做这类问题中Dijkstra算法的好处是什么，其他算法的局限性是什么，为什么要选择Dijkstra算法做这类题目。这些都是我在分析问题的过程中产生的疑问，通过不断深入的查阅资料和学习，我发现了这些问题的解答。**  **二、算法设计**  **2.1 Floyd算法**  P8FU0[QXA4{0TL8$DR7}1%V  是一种基于动态规划的多源最短路算法。  优点：代码非常容易理解，而且很优美。  是一种在具有正或负（但没有负环）  加权图中找到最短路径的算法，容易理解，可以算出任意两个节点之间的最短距离，代码编写简单  缺点：时间复杂度比较高，不适合计算大量数据  空间复杂度:O(n^2)；时间复杂度:O(n^3)；  **2.2 Bellman-Ford算法**  DX314W5ON~7PBZE(QQLSWDH  算法思想：Bellman-Ford算法基于动态  规划，反复利用已有的边来更新最短  距离。  核心思想是松弛。如果dist[u]和dist[v]  满足dist[v]<=dist[u]+map[u][v]，  dist[v]就应该被更新为dist[u]+map[u][v]  反复地利用上式对dist数组进行松弛，  如果没有负权回路的话，  应当会在n-1次松弛之后结束。  时间复杂度O(nm),空间复杂度O(m)  适合于用邻接表建立的图模型（稀疏图）  **2.3 SPFA算法**  最短路径快速算法（Shortest Path Faster Algorithm (SPFA)），一般认为是队列优化的Bellman-Ford 算法，是一个用于求解有向带权图单源最短路径的算法。  这一算法被认为在随机的稀疏图上表现出色，并且适用于带有负边权的图  对于无向图，可通过将每个无向边视作两条有向边以采用 SPFA 算法。  SPFA的复杂度大约是O(kE),k是每个点的平均进队次数(一般的，k是一个常数，在稀疏图中小于2)  其优于狄杰斯特拉算法的方面是边的权值可以为负数、实现简单，缺点是时间复杂度过高，高达 O(VE)。  算法思路：  用数组dis记录每个结点的最短路径估计值，用邻接表或邻接矩阵来存储图G。  采取动态逼近法：设立一个先进先出的队列用来保存待优化的结点，优化时每次取出队首结点u，并且用u点当前的最短路径估计值对离开u点所指向的结点v进行松弛操作，如果v点的最短路径估计值有所调整，且v点不在当前的队列中，就将v点放入队尾。这样不断从队列中取出结点来进行松弛操作，直至队列空为止  BQ(3VINBVW3$BQDY4R%3(SH  C01Y3[U`LRFJN_L_SYS9M~V  **2.4 Dijkstra算法（也是本题推荐使用的算法）**  采用的是贪心算法的策略，只能用于正权图，不适用于含有负权边的图。  算法思想：  1、将所有顶点分为两部分：已知最短路程的顶点集合P和未知最短路程的顶点集合Q。用vis[i]表示，如果vis[i]=1则表示这个顶点在集合P中，反之顶点在集合Q中。  2、设置源点s到自己的最短路径为0。其余按照实际情况进行设置。  3、在集合Q的所有定点中选择一个离源点s最近的顶点加入到集合P。并考察所有以点u为起点的边，对每一条边进行松弛操作。  4、重复第三步，如果集合Q为空，算法结束。最终dis数组中的值就是源点到所有顶点的最短路径。  一般写法的Dijkstra算法时间复杂度为O（N^2），需要在遍历图中所有点和寻找最短距离的时候进行两次遍历，同样适用于用邻接矩阵和邻接表建立的图模型。  **邻接矩阵写法**  MR_EZC2NNI70DM0`Z(T}F9O  **邻接表写法**  3JD{LWOAW%LJW_T_GYIUKAQ  采用堆优化的Dijkstra算法时间复杂度为O（NlogN），在使用堆查找最短距离的时候，可以将时间缩短到logN从而减少时间复杂度  这也是本次课设中主要使用的最短路径算法    2.5 A\*算法  47L0TE[M$610ED1C[NBM8@Y    Q7TXGDCBM}_E9`N%98YFEX5  A\*算法是我在课设制作中，发现的一种比较dijkstra算法更为优异的算法，而且在人工智能和地图导航方面有很多相关应用  **2.6 反思和选择**  以上五种算法都能完成图的最短路径搜索，并且在Floyd算法、Bellman-Ford算法和SPFA算法既能解决本题目中正权图的搜索，还能完成负权图的搜索，但是在效率和时间空间复杂度上都有不小的劣势，更适合邻接表建立的稀疏图模型，在稠密图中效率会异常差。  Dijkstra算法和A\*算法则对邻接矩阵建立的稠密图模型搜索中更胜一筹，并且Dijkstra算法可以通过优先队列对其进行优化，从而使时间复杂度降低到O（NlogN），如果再通过二叉堆进行进一步优化，时间复杂度会变得更优秀，也是本题目中使用最多的算法。  **三、具体功能的实现**  **3.1查询最短时间（不考虑等待时间）**  函数中建立邻接链表部分：只在相邻两站点之间加入边，之后使用dijkstra（）方法  }D(H90624$J4]9RJ13[{_0W  **3.2 查询最短时间（考虑等待时间）**  函数中建立邻接链表部分：只在相邻两站点之间加入边，之后使用Astar（）方法  因为等待时间的随机性，在计算和比较路线具体时间的时候，加入Random（）方法计算随机等待时间，最终进行随机最短时间路线输出    **3.3 查找最少花费**  建立票价邻接矩阵：因为一条线路上任意两个站点之间的票价都是一样的，所以需要设置双层循环，线路上的任意两点之间都要加入边，然后调用dijkstra（）  }D(H90624$J4]9RJ13[{_0W  **3.4 输出耗费时间或花费结果**  使用一个数组进行计数，查找到最短路径之后返回逆序输出，从终点pb开始向前，每次向前寻找前驱，即可得到路径    **四、测试运行与优化**  IMG_0323_看图王Q7SE4FZY)Z6J])SBG}[WFS5    **五、心得感悟**  **5.1畅游算法的海洋**  在本次数据结构课设的设计中，为了寻找更好的实现方法，我查阅了大量资料并且请教了很多大佬，让我初窥数据结构和算法的奥秘。  **5.2 在不断试错中成长**  因为一开始对数据结构不太熟悉，能力仅仅停留在解决一个问题上，没有解决多个问题的能力，而且编程能力和水平偏弱，使我步步为艰  但是我没有放弃，寻找相关资料并且根据自己的想法不断完善借鉴，最终基本完成了题目相关要求  **5.3 未来加油**  在这次课程设计的实验中，我收获了许多知识，也培养了独立思考、动手操作的能力。我也学会了许多学习和解决实际问题的方法，让我受益匪浅。课程设计对我来说，趣味性强，不仅锻炼我们编写代码能力，而且可以学到很多新东西  **5.4 不足和缺陷**  使用的数据结构过于简陋，未能实现可视化输入和输出，输入的数据量还有限制。这些都需要在未来进行维护和完善，随着对数据结构和算法的不断学习，希望能一步步提高。 | | | |
| 结论分析与体会：  ①通过本次数据结构课设的学习和展示，我看到了同学们的优秀成果，并且学习了很多数据结构和算法的新知识，并且学习了同学们的学习态度和方法。  ②在本次数据结构课设的设计中，为了寻找更好的实现方法，我查阅了大量资料并且请教了很多大佬，让我初窥数据结构和算法的奥秘。并且对不同的图模型的建立和最短路径的搜索算法有了深层次的理解  ③因为一开始对数据结构不太熟悉，能力仅仅停留在解决一个问题上，没有解决多个问题的能力，而且编程能力和水平偏弱，使我步步为艰  但是我没有放弃，寻找相关资料并且根据自己的想法不断完善借鉴，最终基本完成了题目相关要求  ④在这次课程设计的实验中，我收获了许多知识，也培养了独立思考、动手操作的能力。我也学会了许多学习和解决实际问题的方法，让我受益匪浅。课程设计对我来说，趣味性强，不仅锻炼我们编写代码能力，而且可以学到很多新东西 | | | |

**本次实验主要代码：**

1. main.cpp

// 定义控制台应用程序的入口点。

//

#include <time.h>

#include<stdio.h>

#include<stdlib.h>

#include <iostream>

#include "busChain.h"

#include "busRoute.h"

#include "adjMatrix.h"

#include <string.h>

#include <string>

#include <iomanip>

#include <cmath>

#define sn 30//站点总数

#define noEdge 9999

using namespace std;

//函数声明

void Welcome();

void Input();

void Menu();

void getRoutes();

int choose();

void Function();

void findMinCost();

void findMinTime1();

void findMinTime2();

void Dijkstra(int da);

int getIndex(string bs);

void outputPath(int pa, int pb);

void AstarSearch(int da);

int find\_(int Infomation);

//全局变量

int option;//进行功能选择

char flag = 'y';//每个功能完成后判断是否退出

busRoute\* bus = new busRoute[10];//公交车线路数组

int BUS\_SUM;//公交线路数

string in[sn] = { " ","A","B","C","D","E","F","G","H","I","J","K","L","M","N","O","P","Q","R","S","T","U","V","W","X","Y","Z"," "," "," " };

//已有对应关系，将站点名和站点标号对应

int vis[sn];//判断是否访问过

int pre[sn];//前驱

double dis[sn];//到源点的距离

adjMatrix matrix(30);//邻接矩阵

int main() {

Welcome();

Input();

getRoutes();

return 0;

}

//欢迎界面

void Welcome() {

cout << endl << endl;

cout << "\t\*\t\t欢迎使用公交线路优化查询系统\t\t\t\*\n";

cout << endl << endl;

cout << "------------------------------------------------------------------------------\n";

}

//输入线路信息

void Input() {

cout << "请输入线路总条数：";

cin >> BUS\_SUM;

int n, nn;

for (int i = 1; i <= BUS\_SUM; i++) {

cout << "请输入第" << i << "条线路的编号：";

cin >> nn;

cout << "请输入" << nn << "号公交车的站点个数：";

cin >> n;

bus[i].initi1(nn, n);

cout << "请输入" << nn << "号公交车的票价(元)、发车间隔(分钟)、速度(每分钟):";

int tt;

double xp, yp;

cin >> xp >> yp >> tt;

bus[i].initi2(xp, yp, tt);//设置票价、发车间隔、速度

cout << endl;

}

}

//显示已有公交线路

void getRoutes() {

cout << endl;

cout << "=------------------------------------------------------------------------------=\n";

cout << "目前已知公交线路相关信息如下：" << endl;

for (int i = 1; i <= BUS\_SUM; i++) {

cout << bus[i];

}

cout << "=------------------------------------------------------------------------------=\n";

cout << "是否确认当前信息？(y/n)";

char aaa;

cin >> aaa;

cout << endl;

if (aaa == 'y'|| aaa == 'Y') {

Menu();

return;

}

else {

cout << "请重新输入公交线路信息！" << endl;

cout << "=------------------------------------------------------------------------------=\n";

Input();

getRoutes();

}

}

//菜单

void Menu() {

while (flag == 'y') {

cout << "\n\n";

cout << "\t\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*\t\n";

cout << "\t\*\t\t\t\t\t\t\t\t\*\n";

cout << "\t\*\t\t欢迎使用公交线路优化查询系统\t\t\t\*\n";

cout << "\t\* \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \*\t\n";

cout << "\t\*\t\t\t\t\t\t\t\t\*\n";

cout << "\t\*\t\t1-查询两站点最省钱方案；\t\t\t\*\t\n";

cout << "\t\*\t\t\t\t\t\t\t\t\*\n";

cout << "\t\*\t\t2-查询两站点最省时方案（不考虑等待时间）；\t\*\t\n";

cout << "\t\*\t\t\t\t\t\t\t\t\*\n";

cout << "\t\*\t\t3-查询两站点最省时方案（考虑等待时间）；\t\*\t\n";

cout << "\t\*\t\t\t\t\t\t\t\t\*\n";

cout << "\t\*\t\t4-退出。\t\t\t\t\t\*\t\n";

cout << "\t\*\t\t\t\t\t\t\t\t\*\n";

cout << "\t\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*\t\n";

cout << endl << endl;

Function();//选择功能

}

}

//功能

void Function() {

cout << "=------------------------------------------------------------------------------=\n";

cout << "\t请输入你的选择：";

cin >> option;

cout << "=------------------------------------------------------------------------------=\n";

if (option == 1)

findMinCost();//查找最少花费

else if (option == 2)

findMinTime1();//查找最少时间（不考虑等待时间）

else if (option == 3)

findMinTime2();//查找最少时间（考虑等待时间）

else if (option == 4) {

cout << "\t已退出公交路径查询系统！" << endl;

flag = 'n';

return;//返回，结束

}

else {

cout << "\t请输入（1 - 4）！\n\n\n";

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

cout << "-----------------------------------------------------------------------------------\n\n";

//Menu();

}//错误情况-容错性？

}

//查找最少花费

void findMinCost() {

cout << "正在查询两站点最省钱方案；" << endl;

matrix.ini();

routeInfo rI;

//建立票价邻接矩阵

for (int i = 1; i <= BUS\_SUM; i++) {

rI.Info = bus[i].getNum();

rI.data = bus[i].getPrice();

busNode\* p = bus[i].getLink().getFirst();

for (p; p != NULL; p = p->next) {

busNode\* pp = p->next;

int ta = getIndex(p->sta\_name);

for (pp; pp != NULL; pp = pp->next) {

int tb = getIndex(pp->sta\_name);

matrix.insert(ta, tb, i, rI);

matrix.insert(tb, ta, i, rI);

}

}

}

string sa, sb;

cout << "请输入出发站点和目的站点：";

cin >> sa >> sb;

matrix.Dijkstra(dis, pre, getIndex(sa), BUS\_SUM);

if (dis[getIndex(sb)] == noEdge) {

cout << "无法利用目前已有线路从" << sa << "到达" << sb << "！" << endl;

cout << "=------------------------------------------------------------------------------=\n";

}

else {

matrix.outputPath(getIndex(sa), getIndex(sb), in, pre, BUS\_SUM);

cout << "共花费" << setiosflags(ios::fixed) << setprecision(0) << dis[getIndex(sb)] << "元。" << endl << endl;

}

cout << "=------------------------------------------------------------------------------=\n";

return;

}

//查找最少时间（不考虑等待时间）

void findMinTime1() {

cout << "正在查询两站点最省时方案（不考虑等待时间）；" << endl;

matrix.ini();

routeInfo rI;

//建立耗费时间邻接表

for (int i = 1; i <= BUS\_SUM; i++) {

rI.Info = bus[i].getNum();

busNode\* p = bus[i].getLink().getFirst();

for (p; p != NULL; p = p->next) {

busNode\* pp = p->next;

int ta = getIndex(p->sta\_name);

if (pp != NULL) {

double ddis = p->dx;

rI.data = ddis / bus[i].getSpeed();//时间

//cout<<rI.data<<endl;

int tb = getIndex(pp->sta\_name);

matrix.insert(ta, tb, i, rI);

matrix.insert(tb, ta, i, rI);

}

}

}

string sa, sb;

cout << "请输入出发站点和目的站点（以空格分隔）：";

cin >> sa >> sb;

matrix.Dijkstra(dis, pre, getIndex(sa), BUS\_SUM);

if (dis[getIndex(sb)] == noEdge) {

cout << "无法利用目前已有线路从" << sa << "到达" << sb << "！" << endl;

cout << "=------------------------------------------------------------------------------=\n";

}

else {

matrix.outputPath(getIndex(sa), getIndex(sb), in, pre, BUS\_SUM);

//outputPath(getIndex(sa),getIndex(sb));

cout << "共花费" << setiosflags(ios::fixed) << setprecision(2) << dis[getIndex(sb)] << "分钟时间。" << endl << endl;

}

cout << "=------------------------------------------------------------------------------=\n";

return;

}

int Random(int start, int end) {

int dis = end - start;

return rand() % dis + start;

}

//查找最少时间（考虑等待时间）

void findMinTime2() {

cout << "正在查询两站点最省时方案（考虑等待时间）；" << endl;

cout << "=------------------------------------------------------------------------------=\n";

matrix.ini();//初始化

routeInfo rI;

//建立耗费时间(包括等待时间)邻接矩阵

for (int i = 1; i <= BUS\_SUM; i++) {

rI.Info = bus[i].getNum();

busNode\* p = bus[i].getLink().getFirst();

for (p; p != NULL; p = p->next) {

busNode\* pp = p->next;

int ta = getIndex(p->sta\_name);

if (pp != NULL) {

double ddis = p->dx;

rI.data = ddis / bus[i].getSpeed();//时间

//cout<<rI.data<<endl;

int tb = getIndex(pp->sta\_name);

matrix.insert(ta, tb, i, rI);

matrix.insert(tb, ta, i, rI);

}

}

}

string sa, sb;

cout << "请输入出发站点和目的站点（以空格分隔）：";

cin >> sa >> sb;

matrix.AstarSearch(dis, pre, getIndex(sa), BUS\_SUM, bus);

if (dis[getIndex(sb)] == noEdge) {

cout << "无法利用目前已有线路从" << sa << "到达" << sb << "！" << endl;

cout << "=------------------------------------------------------------------------------=\n";

}

else {

matrix.outputPath(getIndex(sa), getIndex(sb), in, pre, BUS\_SUM);

//outputPath(getIndex(sa),getIndex(sb));

cout << "共花费" << setiosflags(ios::fixed) << setprecision(2) << dis[getIndex(sb)] << "分钟时间。" << endl << endl;

}

cout << "=------------------------------------------------------------------------------=\n";

return;

}

//得到各站点索引

int getIndex(string bs) {

for (int i = 1; i <= 26; i++) {

if (in[i] == bs)

return i;

}

return -1;

}

1. BusRoute.h

#ifndef busRoute\_

#define busRoute\_

#include "busChain.h"

#include <iostream>

using namespace std;

class busRoute {

public:

busRoute() {

num = 0;

sum = 0;

speed = 0;

price = 0;

time\_ = 0;

}

~busRoute();

void initi1(int nn, int ss);//设置线路编号、站点总数

void initi2(double sp, double pp, double tt);//设置票价、发车间隔、速度

double getPrice() { return price; }

double getSpeed() { return speed; }

int getNum() { return num; }

int getSum() { return sum; }

double getTime\_() { return time\_; }

busChain& getLink() { return link; }

void output(ostream& out) {//输出本线路停经站点及其坐标

cout << num << "：";

cout << link;

link.get\_distance();

cout << price << "元。" << time\_ << "分钟。" << speed << "/分钟" << endl;

//发车时间间隔：速度：票价：

}

private:

int num;//线路编号

int sum;//站点总数

double speed;//速度

double price;//票价

double time\_;//发车时间间隔

busChain link;//本公交线路链表

};

busRoute::~busRoute() {

num = 0;

sum = 0;

speed = 0;

price = 0;

time\_ = 0;

link.~busChain();

}

void busRoute::initi1(int nn, int ss) {

num = nn;

sum = ss;

string str;

double xx, yy;

cout << "请依次输入各个站点的名称和坐标：" << endl;

for (int i = 1; i <= sum; i++) {

cout << "\t站点" << i << "：";

cin >> str >> xx >> yy;

link.insert(str, xx, yy);

}

}

//sp表示价格，pp表示发车间隔，tt表示速度

void busRoute::initi2(double sp, double pp, double tt) {

price = sp;

time\_ = pp;

speed = tt;

}

ostream& operator<<(ostream& out, busRoute& theRoute) {

theRoute.output(out);

return out;

}

#endif

#pragma once

1. BusChain.h

#ifndef busChain\_

#define busChain\_

#include<iostream>

#include<string>

#include<cmath>

#define noEdge 9999

using namespace std;

struct busNode {

string sta\_name;//站点名称

double dx;//站点横坐标

double dy;//站点纵坐标

busNode\* next;//指向下一个站点的指针

};

//公交车链表

class busChain {

friend class busRoute;//友元

public:

busChain() {//构造函数

firstNode = NULL;

listSize = 0;

}

~busChain();//析构函数

bool isEmpty() { return listSize == 0; }//判断是否为空

int size() { return listSize; }//站点数目

busNode\* getFirst() { return firstNode; }//得到首节点

string getFirstStation() { return firstNode->sta\_name; }//得到始发站

void insert(const string& s, const double& x, const double& y);//在链表最后插入一个节点

void get\_distance();//计算每两个站点之间的距离，存入dx

void output(ostream& out) const;//输出

private:

busNode\* firstNode;//指向首节点的指针

int listSize;//节点个数

};

//析构函数

busChain::~busChain() {

cout << "111" << endl;

busNode\* p = firstNode, \* pp;

while (p != NULL) {

pp = p->next;

delete p;

p = pp;

}

}

//在最后插入一个站点

void busChain::insert(const string& s, const double& x, const double& y) {

if (firstNode == NULL) {

firstNode = new busNode;

firstNode->sta\_name = s;

firstNode->dx = x;

firstNode->dy = y;

firstNode->next = NULL;

}

else {

busNode\* p = firstNode;

while (p->next != NULL) {

p = p->next;

}

busNode\* pp = new busNode;

pp->sta\_name = s;

pp->dx = x;

pp->dy = y;

p->next = pp;

pp->next = NULL;

}

listSize++;

//return \*this;

}

void busChain::get\_distance() {

busNode\* p = firstNode, \* pp;

for (pp = p->next; pp != NULL; pp = pp->next) {

p->dx = sqrt(abs(p->dx - pp->dx) \* abs(p->dx - pp->dx) + abs(p->dy - pp->dy) \* abs(p->dy - pp->dy));

p = pp;

}

p->dx = noEdge;

}

void busChain::output(ostream& out) const {

busNode\* p;

for (p = firstNode; p->next != NULL; p = p->next) {

out << p->sta\_name << "(" << p->dx << "," << p->dy << ")" << "->";

}

out << p->sta\_name << "(" << p->dx << "," << p->dy << ")" << "。";

}

ostream& operator<<(ostream& out, const busChain& theChain) {

theChain.output(out);

return out;

}

#endif

#pragma once

1. outputPath方法

//输出从pa到pb的最短路径

void outputPath(int pa, int pb, string\* in, int\* pre, int su) {

if (pa == pb)

return;

int temp[sn];

int co = 1;//计数器

temp[co] = pb;//终点

co++;

int te = pre[pb];//pb的前驱

while (te != pa) {//将路径上的点存到temp数组中

temp[co] = te;

co++;

te = pre[te];

}

temp[co] = pa;

int info = theMatrix[temp[co]][temp[co - 1]].ro[search(temp[co], temp[co - 1], su)].Info;

cout << "线路" << info << "：";

for (int i = co; i >= 1; i--) {//倒序输出

if (i != 1) {

if (theMatrix[temp[i]][temp[i - 1]].ro[search(temp[i], temp[i - 1], su)].Info != info) {

//cout<<"测试："<<in[temp[i]]<<" "<<in[temp[i-1]]<<endl;

cout << "站点" << in[temp[i]] << "。";

cout << "换乘线路" << theMatrix[temp[i]][temp[i - 1]].ro[search(temp[i], temp[i - 1], su)].Info << "：";

cout << "站点" << in[temp[i]] << ",";

}

else {

cout << "站点" << in[temp[i]] << ",";

}

info = theMatrix[temp[i]][temp[i - 1]].ro[search(temp[i], temp[i - 1], su)].Info;//更新当前线路信息

}

else {//最后一个站点

cout << "站点" << in[temp[i]] << "。" << endl;

}

}

}

1. Dijkstra方法

//求da到其他所有点的最短路径长度

void Dijkstra(double\* dis, int\* pre, int da, int su) {

int\* vis = new int[30];//标记数组

for (int i = 1; i <= n; i++) {//初始化为未访问过

vis[i] = 0;

}

for (int i = 1; i <= 26; i++) {

dis[i] = theMatrix[da][i].ro[search(da, i, su)].data;

//cout<<"测试："<<this->search(da,i,su)<<" "<<dis[i]<<endl;

if (dis[i] == noEdge)

pre[i] = 0;

else

pre[i] = da;

}

dis[da] = 0;//da到da距离为0

vis[da] = 1;//标记为访问过

for (int i = 1; i <= 25; i++) {

double temp = noEdge;

int v = 0;

for (int j = 1; j <= 26; j++) {//找出距离 点集 最近的点

if (vis[j] == 0 && dis[j] < temp) {

//cout << j << endl;

v = j;

temp = dis[j];

}

}

vis[v] = 1;

for (int j = 1; j <= 26; j++) {

if (vis[j] == 0 && theMatrix[v][j].ro[search(v, j, su)].data < noEdge) {//更新

double dd = dis[v] + theMatrix[v][j].ro[search(v, j, su)].data;

if (dd < dis[j]) {

dis[j] = dd;

pre[j] = v;

}

}

}

}

}

1. AstarSearch方法

void AstarSearch(Node\* startPos, Node\* endPos)

{

if (startPos->x < 0 || startPos->x > row || startPos->y < 0 || startPos->y >col

||

endPos->x < 0 || endPos->x > row || endPos->y < 0 || endPos->y > col)

return;

Node\* current;

this->startPos = startPos;

this->endPos = endPos;

openList.push\_back(startPos);

//把开始的节点放入openlist后开始查找旁边的8个节点，如果坐标超长范围或在closelist就return 如果已经存在openlist就对比当前节点到遍历到的那个节点的G值和当前节点到原来父节点的G值 如果原来的G值比较大 不用管 否则重新赋值G值 父节点 和f 如果是新节点 加入到openlist直到opellist为空或找到终点

while (openList.size() > 0)

{

current = openList[0];

if (current->x == endPos->x && current->y == endPos->y)

{

cout << "find the path" << endl;

printMap();

printPath(current);

openList.clear();

closeList.clear();

break;

}

NextStep(current);

closeList.push\_back(current);

openList.erase(openList.begin());

sort(openList.begin(), openList.end(), compare);

}

}