**数据结构课程设计报告**

**地铁订票系统**

**学院： 网络空间安全学院**

**班级： 物联网工程1班**

**姓名： 冯远坤**

**学号： 202178040108**

**日期： 2023.1.8**

目录

[一、 课程设计目的 3](#_Toc123842402)

[二、 课程设计内容和要求 3](#_Toc123842403)

[(一) 设计内容 3](#_Toc123842404)

[(二) 设计要求 4](#_Toc123842405)

[三、 任务完成情况 4](#_Toc123842406)

[四、 设计报告 5](#_Toc123842407)

[(一) 系统开发平台 5](#_Toc123842408)

[(二) 设计要求 5](#_Toc123842409)

[1. 问题描述 5](#_Toc123842410)

[2. 输入要求 5](#_Toc123842411)

[3. 输出要求 5](#_Toc123842412)

[(三) 数据结构与算法描述 6](#_Toc123842413)

[1. 整体思路 6](#_Toc123842414)

[2. 所需数据结构及算法 6](#_Toc123842415)

[(四) 测试结果 13](#_Toc123842416)

[1. 测试输入及输出 13](#_Toc123842417)

[2. 测试中的问题及解决 17](#_Toc123842418)

[五、 分析与讨论 18](#_Toc123842419)

[(一) 测试结果分析 18](#_Toc123842420)

[(二) 算法复杂性分析 19](#_Toc123842421)

[(三) 探讨及改进 20](#_Toc123842422)

[六、 附录(源代码) 20](#_Toc123842423)

1. **课程设计目的**

数据结构课程设计是一项综合性设计活动，要求在教师的指导下，利用本课程内的以及到目前为止所学到的有关知识和技术解决一些不太复杂但却是综合性的问题。从规模来说，课程设计是在平时作业的基础上进一步扩大的大作业。在设计中，要求学生要全面考虑相互联系的各个方面及问题。

通过课程设计，使学生对整个课程的知识体系有较深入的理解，在运用本课程的知识解决实际问题方面得到锻炼，对锻炼学生的实践能力以及运用本课程的知识、方法解决更为复杂的实际问题有较好的启发和指导作用，从而为后续课程的学习、毕业设计环节以及将来的实际工作打好坚实的基础。

通过对给定问题的求解，使学生在运用《数据结构》、程序设计以及迄今为止所学课程中的各种基本技术和理论，在建立问题模型、构造求解算法、设计数据结构、编程及上机调试等方面得到全面的锻炼，从而能更深刻地理解《数据结构》的精髓，为后续软件课程的学习及软件设计能力的提高奠定良好的基础。

1. **课程设计内容和要求**
2. 设计内容



【选题】

参考图中郑州地铁一号线部分线路图设计一个地铁订票系统。

【问题描述】

订票管理系统应实现地铁站的插入、删除、修改、查询、排序以及票价查询等工作，请设计一个计算机系统，实现上述功能。

【基本要求】

（1）使用合适的数据结构存储地铁站数据并将示意图中的数据存入你选择的数据结构中。

（2）由于地铁还在不断扩建，请实现地铁站的更新、删除与插入功能。

（3）用户购票时只需输入上车站与下车站，由系统自动计算出票价（两站之内 2 元，三到四站 3 元，以此类推） 。

（4）输出用户票价的同时输出用户经过的地铁站。

（5）使用合适的查找算法，依据用户的输入实现地铁站点查询功能。

（6）记录每个地铁站的人流量总数，在管理员查询时，使用至少 3 种排序算法按照人流量从大到小的顺序列出站点名称。

1. 设计要求

1.学生必须仔细阅读《数据结构》课程设计方案，认真主动完成课设的要求。有问题及时主动通过各种方式与教师联系沟通。

2.学生要发挥自主学习的能力，充分利用时间，安排好课设的时间计划，并在课设过程中不断检测自己的计划完成情况，及时向教师汇报。

3.课程设计按照教学要求需要4周时间完成，本学期第16周末上交所有资料。

1. **任务完成情况**

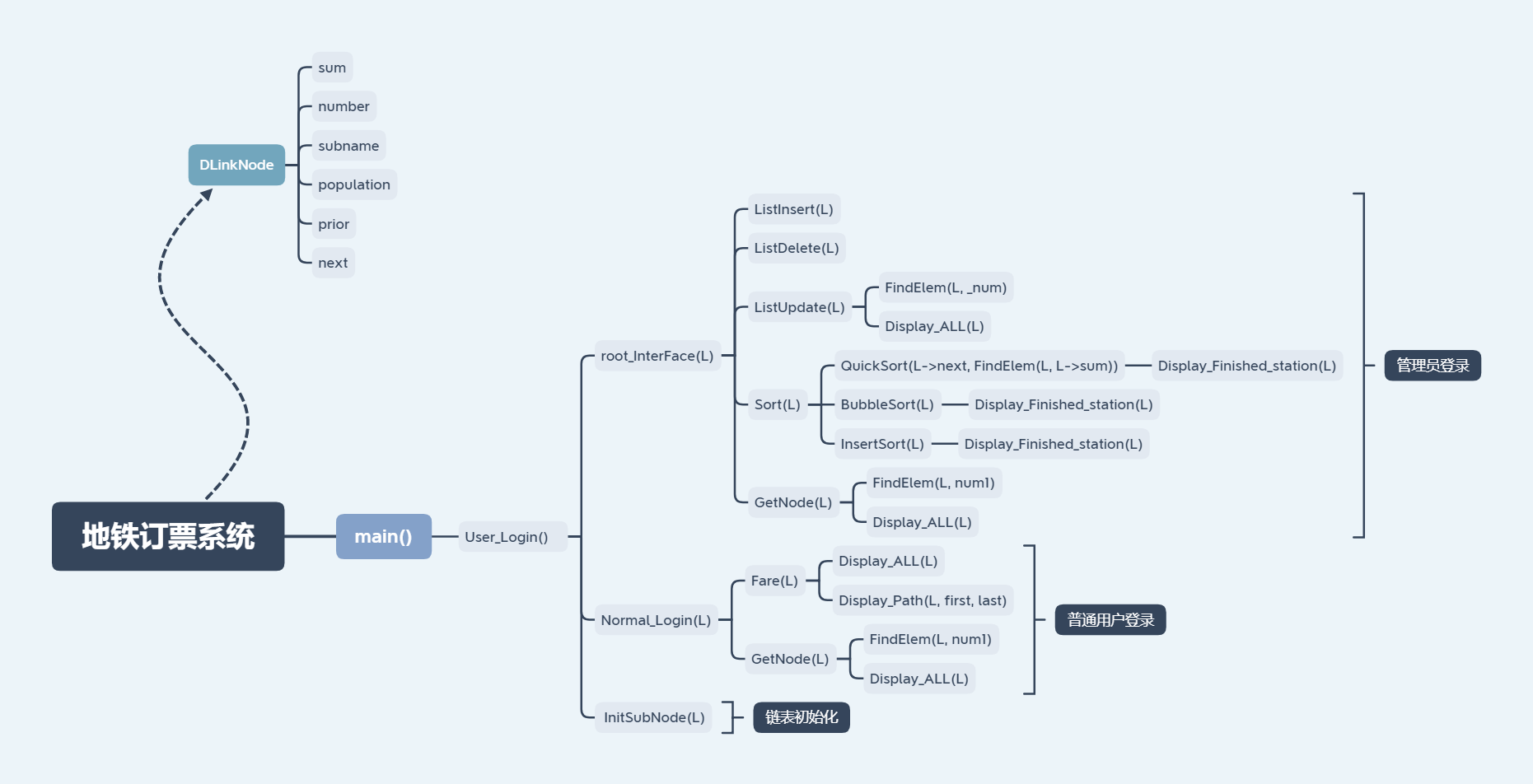
已完成选题的基本要求，包括地铁站的更新、删除与插入功能、系统自动计算出票价、输出用户票价的同时输出用户经过的地铁站、地铁站点查询功能、使用至少 3 种排序算法按照人流量从大到小的顺序列出站点名称。

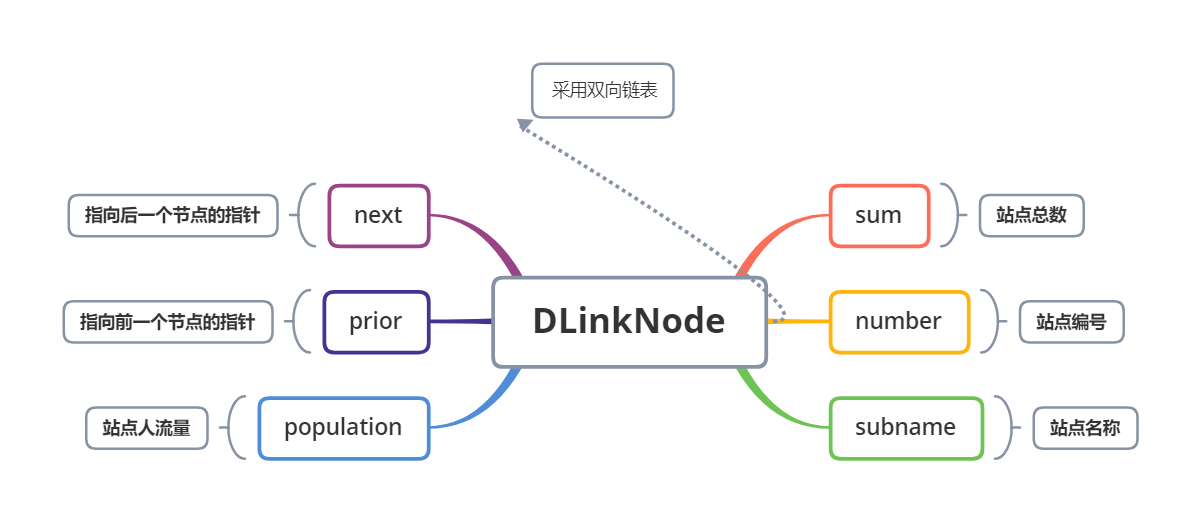
1. **设计报告**
2. 系统开发平台

Visual Studio 2022

1. 设计要求
2. **问题描述**

订票管理系统应实现地铁站的插入、删除、修改、查询、排序以及票价查询等工作，请设计一个计算机系统，实现上述功能。

1. **输入要求**
2. 选择以普通用户或者管理员身份登录
3. 选择实现地铁站的插入、删除、更新或者排序功能
4. 用户输入起始站点和终点站和需购买的票数
5. 输入要查询的站点
6. 用户选择一种排序方式进行排序
7. **输出要求**
8. 根据用户选择的身份输出相应的界面
9. 检测用户输入是否合法
10. 检根据用户输入计算票价并输入途径站点
11. 输出用户查询站点的信息
12. 输出更新后的地铁站
13. 输出排好序的结果
14. 数据结构与算法描述
15. **整体思路**
16. **所需数据结构及算法**
17. 地铁站节点



1. 用车站信息初始化双链表——InitSubNode(DLinkNode\*& L)

建立头结点

建立新结点

输入数据

sum++

Sum<=5

Y N

N

结点的next域置空

1. 插入站点—void ListInsert(DLinkNode\*& L)

建立新结点

sum++

j < i - 1 && p != NULL

输入数据

p = p->next

j++

输入i,p=L,j=0

Y

N

1. 删除站点—void ListDelete(DLinkNode\*& L)

sum--

删除该结点

输入i,p=L

p = p->next,j++

j<i-1&&p!=NULL

Y

N

1. 查找双链表元素——DLinkNode\* FindElem(DLinkNode\* L, int elem)

p->number==elem

return p

p!=NULL

p=p->next

输入elem,p=L

N

Y

Y N

1. 计算费用—void Fare(DLinkNode\* L)

money = (abs(last - first) - 1) \* num

输出money

Display\_Path(L, first, last)

abs(last - first) >= 1 && abs(last - first) <= 2

first<1||last>L->sum||first==last||num <= 0

money = 2 \* num

输入first,last,num

Y

N

N

Y

1. 地铁站的更新——void ListUpdate(DLinkNode\*& L)

Display\_ALL(L)

修改信息

p = FindElem(L, \_num)

输入\_name,\_name

1. 站点查询——void GetNode(DLinkNode\* L)

输出站点信息

p = FindElem(L, num1)

输入num1

Display\_ALL(L)

1. 快速排序——void QuickSort(DLinkNode\* low, DLinkNode\* high)

low != high && low->prior != high

Y N

base->population >= high->population

low != high && low->prior != high

high->population = low->population

high->subname = low->subname

low != high && low->prior != high

low = low->next

base->population <= low->population

high = high->prior

low->population = high->population

low->subname = high->subname;

Y N

以low为基准点，递归调用QuickSort,分别对low左右两边进行排序

1. 冒泡排序—void BubbleSort(DLinkNode\* L)

**N**  **Y**

开始

初始化交换次数为元素个数

flag=0,交换次数比上一轮减少一次

Y

flag==0

N

Y

Display\_Finished\_station(L)

flag==0

本轮交换次数是否用完

swap(), flag=1

相邻两个数据符合降序

从数据起始端开始

1. 插入排序——void InsertSort(DLinkNode\* L)

**N**   **Y**

p = L->next

temp->prior->next = NULL

将temp放入有序区

Display\_Finished\_station(L)

temp->next != NULL

q->next != p

q->population >= temp->population

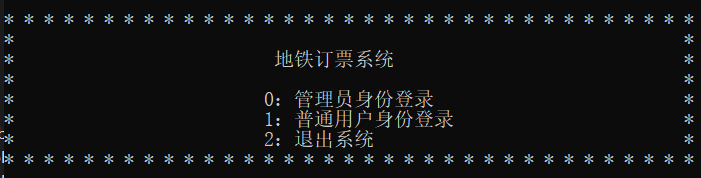
q = p->prior; q != L; q = q->prior

p = p->next,p!=NULL, p = p->next

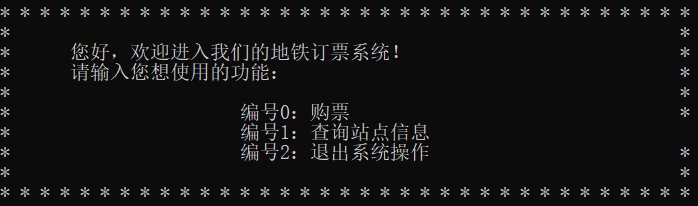
p != NULL

1. 测试结果
2. **测试输入及输出**

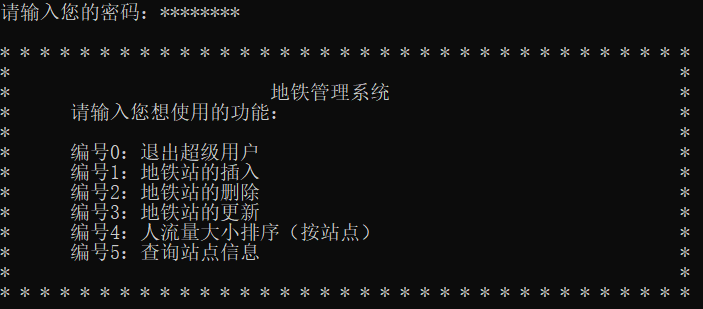
主页面：



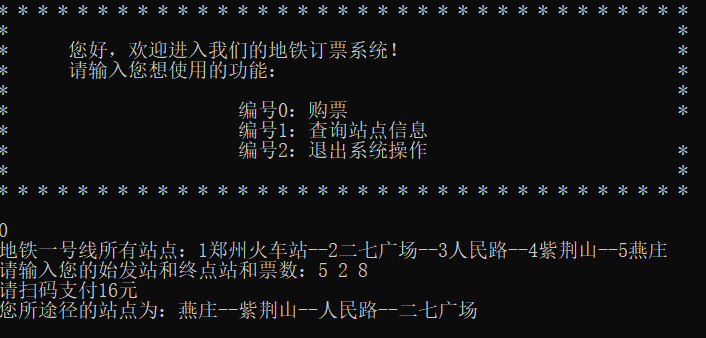
普通用户界面：



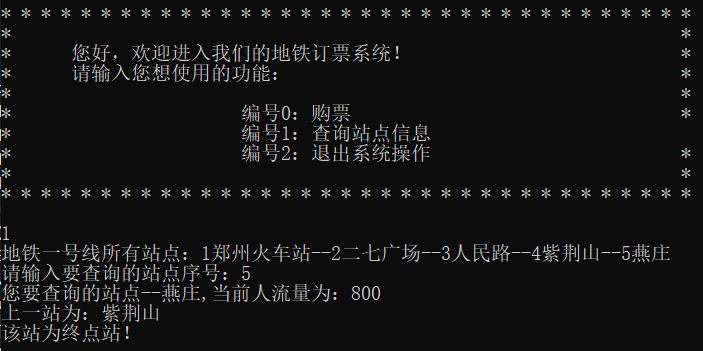
管理员界面：



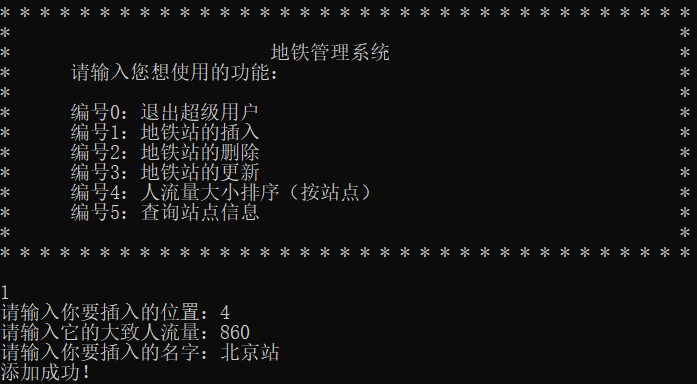
输出用户票价的同时输出用户经过的地铁站：



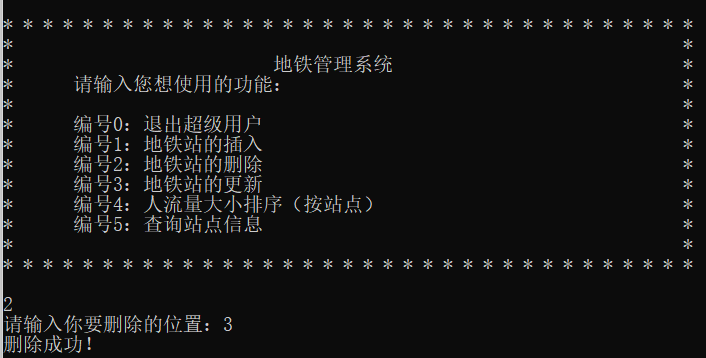
依据用户的输入实现地铁站点查询功能：



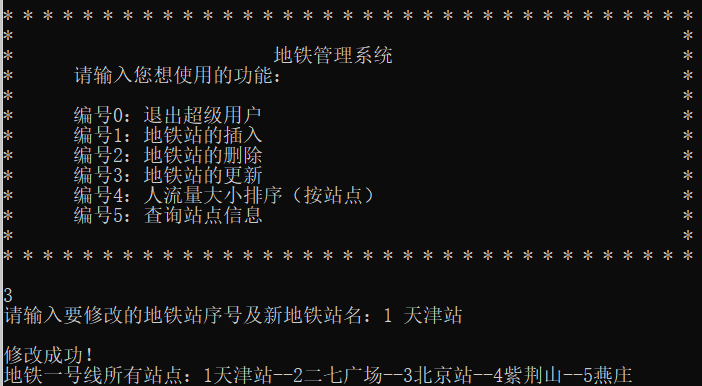
地铁站的插入：



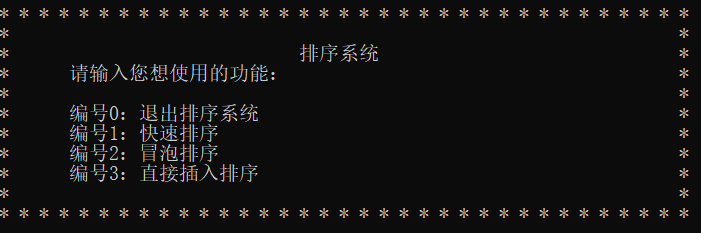
地铁站的删除：



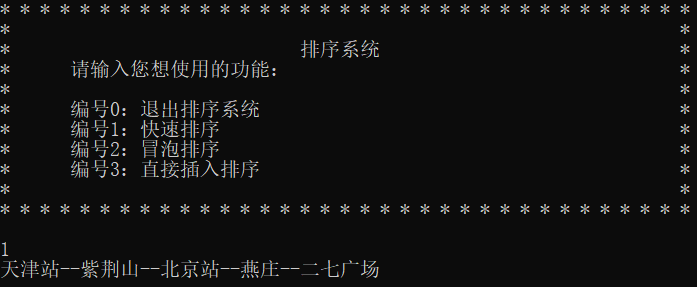
地铁站的更新：



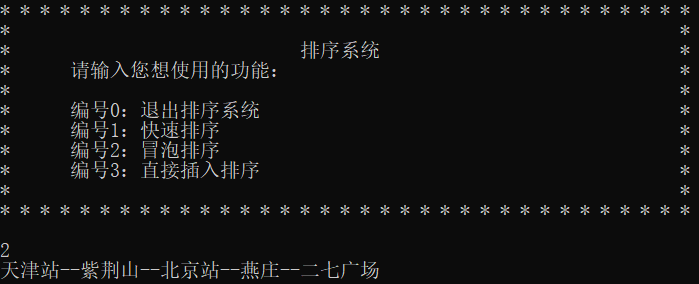
排序界面：



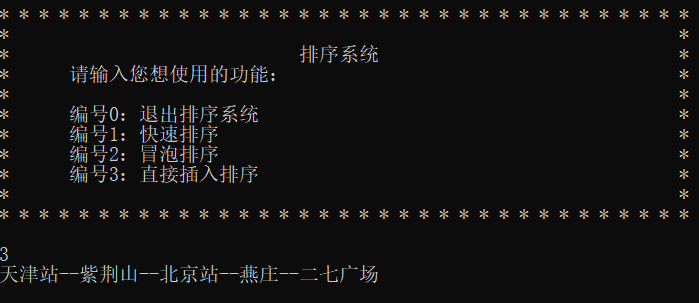
快速排序：



冒泡排序：



直接插入排序：



1. **测试中的问题及解决**

退出界面不成功：调试过程中，无法退出当前界面，外层采用一个do-while循环，内层套用switch进行选择,当输入0退出系统时还是在当前界面。

解决方法：外层再设置一个while循环，设标记标量作为循环结束的条件，为flag赋初值1，当选择退出时，对应的case语句中令flag=0,结束循环，返回上一级。

初始化双向链表时，使用malloc()函数开空间创建结点报错

解决方法：使用new开辟空间，创建结点

1. **分析与讨论**
2. 测试结果分析

输出功能选择界面

主界面 调用root\_InterFace(L)进入管理员界面

调用Normal\_Login(L)进入普通用户界面

输出功能选择界面

普通用户界面 调用Fare(L)计算票价并输出途径站点

调用GetNode(L)查询地铁站点信息

输入密码 进入功能选择界面

调用ListInsert(L)进行地铁站的插入

调用ListDelete(L)进行地铁站的删除

管理员界面

调用ListUpdate(L)进行地铁站的更新

调用Sort(L)进入排序系统

调用GetNode(L)查询地铁站点信息

调用QuickSort(DLinkNode\* low, DLinkNode\* high)进行快速排序

排序系统界面 调用BubbleSort(DLinkNode\* L)进行冒泡排序

调用InsertSort(DLinkNode\* L)进行直接插入排序

1. 算法复杂性分析
2. 地铁站点的插入：需要查找插入位置，找到后改变指针的指向进行插入，时间复杂度o(n),空间复杂度o(1)
3. 地铁站点的删除：需要查找删除的位置，找到后改变指针的指向，时间复杂度o(n),空间复杂度o(1)
4. 查找指定地铁站点—DLinkNode\* FindElem(DLinkNode\* L, int elem)

采用顺序查找，从头结点开始遍历查找，直到找到对应节点遍历结束。时间复杂度o(n),空间复杂度o(1)

1. 快速排序：快速排序的平均时间复杂度为o(nlog2n)。快速排序涉及递归，造成栈空间的使用，最好的情况，递归树的深度为logn,其空间复杂度也为o(logn)；最坏的情况，需要进行n-1次递归调用，其空间复杂度为o(n),则平均空间复杂度为o(logn)
2. 冒泡排序：冒泡排序的平均时间复杂度为o(n2),排序过程需要交换，需要一个中间变量，空间复杂度为o(1)
3. 直接插入排序：直接插入排序，当最好的情况，及排序本身就是有序的，共需比较n-1次，无需移动，时间复杂度为o(n);最坏的情况，即排序表是逆序的情况，时间复杂度为o(n2),平均时间复杂度为o(n2)。空间复杂度为o(1)
4. 探讨及改进
5. 可以设置一个人流量监测系统，通过检测当前购票人数估计当前地铁人流量，实时更新系统。
6. 设置一个阶梯式票价速查表，方便用户查看票价及管理员调整票价
7. 设置一个站点跟踪功能，当用户每到一个站点就输出该站点的名称
8. **附录(源代码)**

#include <bits/stdc++.h>

#include <conio.h>

using namespace std;

typedef int ElemType;

typedef struct DLinkNode

{

int sum = 0;//车站总数

int number = 0;//新添的序列号

string subname;//新添的地铁名

int population = 0;

struct DLinkNode\* prior;

struct DLinkNode\* next;

}DLinkNode;

//用车站信息初始化双链表

void InitSubNode(DLinkNode\*& L);

//插入站点

void ListInsert(DLinkNode\*& L);

//删除站点

void ListDelete(DLinkNode\*& L);

//查找双链表元素

DLinkNode\* FindElem(DLinkNode\* L, int elem);

//输出途径站点

void Display\_Path(DLinkNode\* L, int first, int last);

//输出全部站点

void Display\_ALL(DLinkNode\* L);

//计算费用

void Fare(DLinkNode\* L);

//地铁站的更新

void ListUpdate(DLinkNode\*& L);

//站点查询

void GetNode(DLinkNode\* L);

//输出排好序的站点

void Display\_Finished\_station(DLinkNode\* L);

//快速排序

void QuickSort(DLinkNode\* low, DLinkNode\* high);

//冒泡排序

void BubbleSort(DLinkNode\* L);

//插入排序

void InsertSort(DLinkNode\* L);

//排序方式

void Sort(DLinkNode\* L);

//管理员界面

void root\_InterFace(DLinkNode\* L);

//普通用户登录

void Normal\_Login(DLinkNode\* L);

//某个用户身份登录

void User\_Login();

int main()

{

User\_Login();

return 0;

}

void InitSubNode(DLinkNode\*& L)

{

L = new DLinkNode;//建立头结点

L->number = 0;

L->prior = NULL;

//尾插法建立数据结点

DLinkNode\* s1, \* s2, \* s3, \* s4, \* s5;

s1 = new DLinkNode;

L->next = s1; s1->prior = L;

s1->number = 1;

s1->population = 2000;

s1->subname = "郑州火车站";

s2 = new DLinkNode;

s1->next = s2; s2->prior = s1;

s2->number = 2;

s2->population = 600;

s2->subname = "二七广场";

s3 = new DLinkNode;

s2->next = s3; s3->prior = s2;

s3->number = 3;

s3->population = 500;

s3->subname = "人民路";

s4 = new DLinkNode;

s3->next = s4; s4->prior = s3;

s4->number = 4;

s4->population = 1500;

s4->subname = "紫荆山";

s5 = new DLinkNode;

s4->next = s5; s5->prior = s4; s5->next = NULL;

s5->number = 5;

s5->population = 800;

s5->subname = "燕庄";

L->sum = 5;//站点总数

}

void ListInsert(DLinkNode\*& L)

{

int i;

cout << "请输入你要插入的位置：";

cin >> i;

ElemType e;

cout << "请输入它的大致人流量：";

cin >> e;

string name;

cout << "请输入你要插入的名字：";

cin >> name;

DLinkNode\* s, \* p, \* q;

p = L;

int j = 0;

while (j < i - 1 && p != NULL)

{

p = p->next;

j++;

}

if (p == NULL)

return;

else

{

s = new DLinkNode;

s->population = e;

s->number = i;

s->subname = name;

s->next = p->next;

if (p->next != NULL)

p->next->prior = s;

p->next = s;

s->prior = p;

L->sum++;//地铁站总数加1

q = s->next;

//后续站点的序号加1

while (q != NULL)

{

q->number++;

q = q->next;

}

cout << "添加成功！" << endl;

}

}

void ListDelete(DLinkNode\*& L)

{

int i;

cout << "请输入你要删除的位置：";

cin >> i;

DLinkNode\* s, \* q, \* p, \* t;

int j = 0;

p = L;

while (j < i - 1 && p != NULL)

{

p = p->next;

j++;

}

if (p == NULL)

return;

else

{

q = p->next;

if (q == NULL)

{

return;

}

p->next = q->next;

if (q->next != NULL)

q->next->prior = p;

free(q);

L->sum--;//地铁站总数减1

t = p->next;

//后续地铁站序号依次减1

while (t != NULL)

{

t->number--;

t = t->next;

}

cout << "删除成功！" << endl;

}

}

DLinkNode\* FindElem(DLinkNode\* L, int elem)

{

DLinkNode\* p = L;

while (p)

{

if (p->number == elem)

break;

p = p->next;

}

return p;

}

void Display\_Path(DLinkNode\* L, int first, int last)

{

DLinkNode\* p, \* q;

DLinkNode\* first\_node = FindElem(L, first);

DLinkNode\* last\_node = FindElem(L, last);

p = first\_node;

q = p->next;

if (last > first)

{

while (p != last\_node)

{

cout << p->subname << "--";

p = p->next;

}

cout << p->subname << endl;

}

else if (last < first)

{

while (p != last\_node)

{

cout << p->subname << "--";

p = p->prior;

}

cout << p->subname << endl;

}

}

void Display\_ALL(DLinkNode\* L)

{

cout << "地铁一号线所有站点：";

DLinkNode\* p;

p = L->next;

while (p->next != NULL)

{

cout << p->number << p->subname << "--";

p = p->next;

}

cout << p->number << p->subname << endl;

}

void Fare(DLinkNode\* L)

{

int first, last;//始发站和终点站

int money;//票价

int num;//票数

Display\_ALL(L);

while (1)

{

cout << "请输入您的始发站和终点站和票数：";

cin >> first >> last >> num;

if (first<1 || last>L->sum || first == last || num <= 0)

{

cout << "输入错误，请重新输入：" << endl;

}

else

break;

}

if (abs(last - first) >= 1 && abs(last - first) <= 2)

money = 2 \* num;

else money = (abs(last - first) - 1) \* num;

cout << "请扫码支付" << money << "元" << endl;

cout << "您所途径的站点为：";

Display\_Path(L, first, last);

}

void ListUpdate(DLinkNode\*& L)

{

int \_num;

string \_name;

DLinkNode\* p;

cout << "请输入要修改的地铁站序号及新地铁站名：";

cin >> \_num >> \_name;

p = FindElem(L, \_num);

p->subname = \_name;

cout << endl;

cout << "修改成功！" << endl;

Display\_ALL(L);

}

void GetNode(DLinkNode\* L)

{

int num1;

DLinkNode\* p;

Display\_ALL(L);

cout << "请输入要查询的站点序号：";

cin >> num1;

p = FindElem(L, num1);

cout << "您要查询的站点--" << p->subname << ",当前人流量为：" << p->population << endl;

if (p->prior != L)

cout << "上一站为：" << p->prior->subname << endl;

else

cout << "该站为始发站！" << endl;

if (p->next != NULL)

cout << "下一站为：" << p->next->subname << endl;

else

cout << "该站为终点站！" << endl;

}

void Display\_Finished\_station(DLinkNode\* L)

{

DLinkNode\* p;

p = L->next;

while (p->next != NULL)

{

cout << p->subname << "--";

p = p->next;

}

cout << p->subname << endl;

}

void QuickSort(DLinkNode\* low, DLinkNode\* high)

{

DLinkNode\* i = low;

DLinkNode\* j = high;

DLinkNode\* base = new DLinkNode;//基准结点

if (low == NULL)

return;

base->population = low->population;

base->subname = low->subname;

if (low == high || low->prior == high)

return;

while (low != high && low->prior != high)

{

//从右向左扫描，直到找到一个大于基准的结点

for (; low != high && low->prior != high && base->population >= high->population; high = high->prior);

if (base->population < high->population)

{

low->population = high->population;

low->subname = high->subname;

}

//从左向右扫描，直到找到一个小于基准地结点

for (; low != high && low->prior != high && base->population <= low->population; low = low->next);

if (base->population > low->population)

{

high->population = low->population;

high->subname = low->subname;

}

}

//将基准结点归位

low->population = base->population;

low->subname = base->subname;

QuickSort(i, low->prior);

QuickSort(low->next, j);

}

void BubbleSort(DLinkNode\* L)

{

int flag = 1;//用来标记是否发生交换

DLinkNode\* head = L;

DLinkNode\* tail = NULL;

DLinkNode\* p, \* q;

while (flag)

{

flag = 0;

p = head->next;

while (p->next != tail)

{

//交换节点

if (p->population < p->next->population)

{

flag = 1;

q = p->next;

p->next = q->next;

if (q->next != NULL)

{

q->next->prior = p;

}

p->prior->next = q;

q->prior = p->prior;

p->prior = q;

q->next = p;

}

else

{

p = p->next;

}

}

//从尾部开始起泡

tail = p;

p = tail->prior;

while (flag && p->prior != head)

{

if (p->population > p->prior->population)

{

q = p->prior;

p->prior = q->prior;

if (q->prior != NULL)

{

q->prior->next = p;

}

p->next->prior = q;

q->next = p->next;

p->next = q;

q->prior = p;

}

else

{

p = p->prior;

}

}

head = p;

}

Display\_Finished\_station(L);

}

void InsertSort(DLinkNode\* L)

{

DLinkNode\* p, \* q, \* temp;

p = L->next;

if (p != NULL)

{

for (p = p->next; p; p = p->next)//从第二个数开始排序，第一个数默认有序

{

temp = p;

for (q = p->prior; q != L; q = q->prior)

{

if (q->population >= temp->population)//降序,结束循环

{

break;

}

}

if (q->next != p)//若q前移，插入temp

{

if (temp->next != NULL)//temp不是最后一个结点

{

temp->prior->next = temp->next;

temp->next->prior = temp->prior;

temp->next = q->next;

q->next->prior = temp;

temp->prior = q;

q->next = temp;

}

else

{

temp->prior->next = NULL;

temp->next = q->next;

q->next->prior = temp;

temp->prior = q;

q->next = temp;

}

}

}

}

Display\_Finished\_station(L);

}

void Sort(DLinkNode\* L)

{

int t4 = 1;

while (t4)

{

cout << endl;

cout << "\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*" << endl;

cout << "\* \*" << endl;

cout << "\* 排序系统 \*" << endl;

cout << "\* 请输入您想使用的功能： \*" << endl;

cout << "\* \*" << endl;

cout << "\* 编号0：退出排序系统 \*" << endl;

cout << "\* 编号1：快速排序 \*" << endl;

cout << "\* 编号2：冒泡排序 \*" << endl;

cout << "\* 编号3：直接插入排序 \*" << endl;

cout << "\* \*" << endl;

cout << "\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*" << endl;

cout << endl;

int choice;

do

{

cin >> choice;

switch (choice)

{

case 0:

t4 = 0;

break;

case 1:

QuickSort(L->next, FindElem(L, L->sum));

Display\_Finished\_station(L);

break;

case 2:

BubbleSort(L);

break;

case 3:

InsertSort(L);

break;

default:

cout << "输入有误，请重新输入：";

break;

}

} while (choice < 0 || choice > 3);

}

}

void root\_InterFace(DLinkNode\* L)

{

int t3 = 1;

int cnt = 5;

int flag;//用于标记密码是否输入正确以提前结束循环

int index;

char password[100];

char str[] = { "88888888" };

cout << "请输入您的密码：";

for (int i = 1; i <= 5 && cnt > 0; i++)

{

index = 0;

flag = 0;

while (1)

{

char ch;

ch = \_getch();

if (ch == 8) //退格键

{

if (index != 0)

{

cout << char(8) << " " << char(8);

index--;

}

}

else if (ch == '\r') //回车键

{

password[index] = '\0';

cout << endl;

break;

}

else

{

cout << "\*";

password[index++] = ch;

}

}

for (int j = 0; j <= strlen(password); j++)

{

if (str[j] != password[j])

{

flag = 1;

cnt--;

if (cnt > 0)

cout << "密码错误，你还有" << cnt << "次机会，请重新输入：";

break;

}

}

//密码输入正确

if (flag == 0)

{

while (t3)

{

cout << endl;

cout << "\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*" << endl;

cout << "\* \*" << endl;

cout << "\* 地铁管理系统 \*" << endl;

cout << "\* 请输入您想使用的功能： \*" << endl;

cout << "\* \*" << endl;

cout << "\* 编号0：退出超级用户 \*" << endl;

cout << "\* 编号1：地铁站的插入 \*" << endl;

cout << "\* 编号2：地铁站的删除 \*" << endl;

cout << "\* 编号3：地铁站的更新 \*" << endl;

cout << "\* 编号4：人流量大小排序（按站点） \*" << endl;

cout << "\* 编号5：查询站点信息 \*" << endl;

cout << "\* \*" << endl;

cout << "\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*" << endl;

cout << endl;

int choice;

do

{

cin >> choice;

switch (choice)

{

case 0:

t3 = 0;

break;

case 1:

ListInsert(L);

break;

case 2:

ListDelete(L);

break;

case 3:

ListUpdate(L);

break;

case 4:

Sort(L);

break;

case 5:

GetNode(L);

break;

default:

cout << "输入有误，请重新输入：";

break;

}

} while (choice < 0 || choice > 5);

}

break;

}

}

}

void Normal\_Login(DLinkNode\* L)

{

int t2 = 1;

while (t2)

{

cout << endl;

cout << "\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*" << endl;

cout << "\* \*" << endl;

cout << "\* 您好，欢迎进入我们的地铁订票系统！ \*" << endl;

cout << "\* 请输入您想使用的功能： \*" << endl;

cout << "\* \*" << endl;

cout << "\* 编号0：购票 \*" << endl;

cout << "\* 编号1：查询站点信息 \*" << endl;

cout << "\* 编号2：退出系统操作 \*" << endl;

cout << "\* \*" << endl;

cout << "\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*" << endl;

cout << endl;

int choice;

do

{

cin >> choice;

switch (choice)

{

case 0:

Fare(L);

break;

case 1:

GetNode(L);

break;

case 2:

t2 = 0;

break;

default:

cout << "输入有误，请重新输入：";

break;

}

} while (choice < 0 || choice > 2);

}

}

void User\_Login()

{

int t1 = 1;

DLinkNode\* L;

InitSubNode(L);

while (t1)

{

cout << endl;

cout << "\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*" << endl;

cout << "\* \*" << endl;

cout << "\* 地铁订票系统 \*" << endl;

cout << "\* \*" << endl;

cout << "\* 0：管理员身份登录 \*" << endl;

cout << "\* 1：普通用户身份登录 \*" << endl;

cout << "\* 2：退出系统 \*" << endl;

cout << "\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*" << endl;

cout << endl;

int choice;

do

{

cin >> choice;

switch (choice)

{

case 0:

root\_InterFace(L);

break;

case 1:

Normal\_Login(L);

break;

case 2:

t1 = 0;

break;

default:

cout << "输入有误，请重新输入：";

break;

}

} while (choice < 0 || choice > 2);

}

}