**指导老师：**

**班级：1621**

**学号：1621**

**姓名：**

**日期：2022-12-31**

**《数据结构》**

**课程设计报告**

目录

[第一部分 必做题 4](#_Toc127211715)

[1.1 菜鸟智慧系统 4](#_Toc127211716)

[1.1.1 数据结构： 4](#_Toc127211717)

[1.1.2 算法设计思想： 4](#_Toc127211718)

[1.1.3 测试数据和结果： 16](#_Toc127211719)

[1.1.4 时间复杂度： 22](#_Toc127211720)

[1.1.5 反思： 22](#_Toc127211721)

[1.1.6 改进： 22](#_Toc127211722)

[1.1.7 该题代码行： 22](#_Toc127211723)

[1.2 算术表达式求值 23](#_Toc127211724)

[1.2.1 数据结构： 23](#_Toc127211725)

[1.2.2 算法设计思想： 23](#_Toc127211726)

[1.2.3 源程序： 23](#_Toc127211727)

[1.2.4测试数据和结果 30](#_Toc127211728)

[1.2.5 时间复杂度： 32](#_Toc127211729)

[1.2.6 反思： 32](#_Toc127211730)

[1.2.7 改进： 33](#_Toc127211731)

[1.2.8 该题代码行： 33](#_Toc127211732)

[1.3 特殊路径统计 33](#_Toc127211733)

[1.3.1 数据结构： 33](#_Toc127211734)

[1.3.2 算法设计思想： 33](#_Toc127211735)

[1.3.3 源程序： 34](#_Toc127211736)

[1.3.4 测试数据和结果 38](#_Toc127211737)

[1.3.5 时间复杂度： 38](#_Toc127211738)

[1.3.6 改进： 39](#_Toc127211739)

[1.3.7 反思： 39](#_Toc127211740)

[1.3.8 该题代码行： 39](#_Toc127211741)

[1.4 公交路线提示 39](#_Toc127211742)

[1.4.1 数据结构： 39](#_Toc127211743)

[1.4.2 算法设计思想： 41](#_Toc127211744)

[1.4.3 源程序： 42](#_Toc127211745)

[1.4.4 测试数据和结果 57](#_Toc127211746)

[1.4.5 时间复杂度： 59](#_Toc127211747)

[1.4.6 反思： 60](#_Toc127211748)

[1.4.7 改进： 60](#_Toc127211749)

[1.4.8 该题代码行： 60](#_Toc127211750)

[1.5 霍夫曼编码 60](#_Toc127211751)

[1.5.1 数据结构： 60](#_Toc127211752)

[1.5.2 算法设计思想： 60](#_Toc127211753)

[1.5.3 源程序： 61](#_Toc127211754)

[1.5.4 测试数据和结果 65](#_Toc127211755)

[1.5.5 时间复杂度： 84](#_Toc127211756)

[1.5.6 反思： 84](#_Toc127211757)

[1.5.7 该题代码行： 84](#_Toc127211758)

[1.6 排序算法比较 84](#_Toc127211759)

[1.6.1 数据结构： 84](#_Toc127211760)

[1.6.2 算法设计思想： 84](#_Toc127211761)

[1.6.3源程序： 85](#_Toc127211762)

[1.6.4 测试数据和结果： 96](#_Toc127211763)

[1.6.5 时间复杂度： 99](#_Toc127211764)

[1.6.6 反思： 100](#_Toc127211765)

[1.6.7 改进 100](#_Toc127211766)

[1.6.8 该题代码行： 100](#_Toc127211767)

[第二部分 选做题 101](#_Toc127211768)

[2.1 -8：棋局评估【2】 101](#_Toc127211769)

[2.1.1数据结构： 101](#_Toc127211770)

[2.1.2 算法设计思想： 101](#_Toc127211771)

[2.1.3 源程序： 101](#_Toc127211772)

[2.1.4 测试数据和结果： 103](#_Toc127211773)

[2.1.5 时间复杂度： 104](#_Toc127211774)

[2.1.6 该题代码行： 104](#_Toc127211775)

[2.2 -10：URL映射【2】 104](#_Toc127211776)

[2.2.1 数据结构： 104](#_Toc127211777)

[2.2.2 算法设计思想： 104](#_Toc127211778)

[2.2.3 源程序： 105](#_Toc127211779)

[2.2.4 测试数据和结果： 109](#_Toc127211780)

[2.2.5 时间复杂度： 110](#_Toc127211781)

[2.2.6 该题代码行： 110](#_Toc127211782)

[2.3 -11：词梯【2】 110](#_Toc127211783)

[2.3.1 数据结构： 110](#_Toc127211784)

[2.3.2 算法设计思想： 110](#_Toc127211785)

[2.3.4 源程序： 111](#_Toc127211786)

[2.3.5 测试数据和结果 115](#_Toc127211787)

[2.3.6 时间复杂度： 116](#_Toc127211788)

[2.3.7 反思： 116](#_Toc127211789)

[2.3.8 改进： 116](#_Toc127211790)

[2.3.9 该题代码行： 116](#_Toc127211791)

[2.4-15：迷宫问题【2】 117](#_Toc127211792)

[2.4.1 数据结构： 117](#_Toc127211793)

[2.4.2 算法设计思想： 117](#_Toc127211794)

[2.4.3 源程序： 117](#_Toc127211795)

[2.4.4 测试数据和结果 123](#_Toc127211796)

[2.4.5 时间复杂度： 126](#_Toc127211797)

[2.4.6 反思： 126](#_Toc127211798)

[2.4.7 改进： 126](#_Toc127211799)

[2.4.8 该题代码行： 126](#_Toc127211800)

[2.5-20：平衡二叉树操作的演示【4】 126](#_Toc127211801)

[2.5.1 数据结构： 126](#_Toc127211802)

[2.5.2 算法设计思想： 126](#_Toc127211803)

[2.5.3 源程序： 127](#_Toc127211804)

[2.5.4 测试数据和结果 135](#_Toc127211805)

[2.5.5 时间复杂度： 139](#_Toc127211806)

[2.5.6 反思： 140](#_Toc127211807)

[2.5.7 改进： 140](#_Toc127211808)

[2.5.8 该题代码行： 140](#_Toc127211809)

[第三部分 心得体会 140](#_Toc127211810)

# 第一部分 必做题

## 1.1 菜鸟智慧系统

### 1.1.1 数据结构：

采用了结构体存储包裹的相关数据，如下：

typedef struct pakege{

string name;

string packCode;

string phone;

string size;

int date;

}pak;

双向链表节点的类如下：

class Node {

public:

pak data;

Node \*prev;

Node \*next;

Node(pak data) : data(data), prev(nullptr), next(nullptr) {}

};

### 1.1.2 算法设计思想：

将双向链表相关的数据成员（头尾指针，节点个数，节点最大个数）以及相关函数（添加删除节点和遍历链表等）写在一个类DoublyLinkedList的里面。方便系统的相关编写。大中小三个货架分别对应三个双向链表。用一个全局变量Date表示日期。

在主函数中，先使用loadPack函数读取data.txt文件里的数据到链表中。然后使用menu函数输出菜单并接受命令进行操作，当一天结束后会使用updatePack函数清除过期包裹。退出系统则会调用savePack函数将链表的数据写回至data.txt文件中。

在menu函数中输出菜单，程序使用者可选择进入管理员系统、用户系统、结束今天或退出主系统。管理员系统会调用admin函数，用户系统会调用users函数。

在admin函数中，管理员可以手动或自动存包裹，或者获取相关统计信息。手动存包裹调用packNotAuto函数，该函数在管理员输入相关信息后，生成取件码，并使用DoublyLinkedList类的添加节点（添加在链表尾部）insertAtTail函数添加包裹到相对应链表中。自动存包裹调用packAuto函数读取input.txt文件里的数据添加包裹，过程类似手动存包裹，统计则调用analyse函数展示相关信息。

在users函数中，用户可以按照电话号码取包裹或者用取件码取包裹。前者调用pickByPhone函数，后者调用pickByCode函数。在pickByPhone函数中，用户输入电话号码后，程序现在查找链表中是否存在对应电话号码，然后调用双向链表的删除节点成员函数。在pickByCode函数中，程序会根据输入的取件码查找对应的包裹的电话号码，然后再根据电话号码删除对应的所有包裹。如果输入的电话号码或者取件码找不到对应包裹会给出相关提示。

源程序：

#include <bits/stdc++.h>

using namespace std;

#define MAX\_PERSON 30

int Date = 0;

typedef struct pakege{

string name;

string packCode;

string phone;

string size;

int date;

}pak;

class Node {

public:

pak data;

Node \*prev;

Node \*next;

Node(pak data) : data(data), prev(nullptr), next(nullptr) {}

};

class DoublyLinkedList {

public:

Node \*head;

Node \*tail;

int MaxOfList;

int NumOfList = 0;

DoublyLinkedList(int M) : MaxOfList(M), head(nullptr), tail(nullptr) {}

void traverse() {

Node \*temp = head;

while (temp) {

cout << "姓名："<<temp->data.name << " 电话："<<temp->data.phone<<" 日期（第几天）："<<temp->data.date<<" 取件码"<<temp->data.packCode<<" 货架编号："<<temp->data.size<<endl;

temp = temp->next;

}

}

void insertAtHead(pak data) {

if (NumOfList+1 <= MaxOfList){

Node \*newNode = new Node(data);

NumOfList++;

if (!head) {

head = newNode;

tail = newNode;

return;

}

head->prev = newNode;

newNode->next = head;

head = newNode;

}

else{

cout<<"货架已满"<<endl;

}

}

void insertAtTail(pak data) {

if (NumOfList+1 <= MaxOfList){

Node \*newNode = new Node(data);

NumOfList++;

if (!tail) {

head = newNode;

tail = newNode;

return;

}

tail->next = newNode;

newNode->prev = tail;

tail = newNode;

}

else {

cout<<"货架已满"<<endl;

}

}

void deleteAtHead() {

if (!head) {

return;

}

if (head == tail) {

NumOfList=0;

delete head;

head = nullptr;

tail = nullptr;

return;

}

Node \*temp = head;

head = head->next;

head->prev = nullptr;

NumOfList--;

delete temp;

}

void deleteAtTail() {

if (!tail) {

return;

}

if (head == tail) {

NumOfList=0;

delete tail;

head = nullptr;

tail = nullptr;

return;

}

Node \*temp = tail;

tail = tail->prev;

tail->next = nullptr;

NumOfList--;

delete temp;

}

/\*

bool search(pak data) {

Node \*temp = head;

while (temp) {

if (temp->data == data) {

return true;

}

temp = temp->next;

}

return false;

}

\*/

bool searchPhone(string phoneNeedToSearch) {

Node \*temp = head;

while (temp) {

if (temp->data.phone == phoneNeedToSearch) {

return true;

}

temp = temp->next;

}

return false;

}

bool searchCode(string codeNeedToSearch) {

Node \*temp = head;

while (temp) {

if (temp->data.packCode == codeNeedToSearch) {

return true;

}

temp = temp->next;

}

return false;

}

string findPhoneByCode(string codeNeedToSearch){

Node \*temp = head;

string result;

while (temp) {

if (temp->data.packCode == codeNeedToSearch) {

result = temp->data.phone;

return result;

}

temp = temp->next;

}

}

void deleteNode(pak data) {

Node \*temp = head;

while (temp) {

if (temp->data.phone == data.phone) {

if (temp == head) {

deleteAtHead();

return;

} else if (temp == tail) {

deleteAtTail();

return;

} else {

temp->prev->next = temp->next;

temp->next->prev = temp->prev;

delete temp;

NumOfList--;

return;

}

}

temp = temp->next;

}

}

void deleteNodeByPhone(string phoneDel) {

Node \*temp = head;

while (temp) {

if (temp->data.phone == phoneDel) {

cout << "取到取件码为"<<temp->data.packCode<<"的包裹\n";

if (temp == head) {

deleteAtHead();

return;

} else if (temp == tail) {

deleteAtTail();

return;

} else {

temp->prev->next = temp->next;

temp->next->prev = temp->prev;

delete temp;

NumOfList--;

return;

}

}

temp = temp->next;

}

}

void deleteOutdate(int date) {

Node \*temp = head;

while (temp) {

if (temp->data.date + 2 <= date) {

if (temp == head) {

deleteAtHead();

return;

} else if (temp == tail) {

deleteAtTail();

return;

} else {

temp->prev->next = temp->next;

temp->next->prev = temp->prev;

delete temp;

NumOfList--;

return;

}

}

temp = temp->next;

}

}

};

DoublyLinkedList Max(50);

DoublyLinkedList Mid(100);

DoublyLinkedList Min(500);

void packNotAuto(){

pak packTemp;

string codePart;

cout<<"请输入要存放的货架编号: 1(大) 2(中) 3(小)\n";

cin >> packTemp.size;

if (packTemp.size=="1"){

codePart = std::to\_string(Max.NumOfList+1);

}

else if (packTemp.size=="2"){

codePart = std::to\_string(Mid.NumOfList+1);

}

else if (packTemp.size=="3"){

codePart = std::to\_string(Min.NumOfList+1);

}

cout<<"请输入包裹主人电话号码\n";

cin >> packTemp.phone;

cout<<"请输入包裹主人姓名\n";

cin >> packTemp.name;

packTemp.date = Date;

packTemp.packCode = packTemp.size + codePart;

if (packTemp.size=="1"){

Max.insertAtTail(packTemp);

}

else if (packTemp.size=="2"){

Mid.insertAtTail(packTemp);

}

else if (packTemp.size=="3"){

Min.insertAtTail(packTemp);

}

cout<<"存放包裹完成\n";

}

void packAuto(){

ifstream inputPack;

inputPack.open("input.txt");

while (!inputPack.eof()){

pak packTemp;

string codePart;

inputPack >> packTemp.size;

if (packTemp.size=="1"){

codePart = std::to\_string(Max.NumOfList+1);

}

else if (packTemp.size=="2"){

codePart = std::to\_string(Mid.NumOfList+1);

}

else if (packTemp.size=="3"){

codePart = std::to\_string(Min.NumOfList+1);

}

//cout<<"请输入包裹主人电话号码\n";

inputPack >> packTemp.phone;

//cout<<"请输入包裹主人姓名\n";

inputPack >> packTemp.name;

packTemp.date = Date;

packTemp.packCode = packTemp.size + codePart;

if (packTemp.size=="1"){

Max.insertAtTail(packTemp);

}

else if (packTemp.size=="2"){

Mid.insertAtTail(packTemp);

}

else if (packTemp.size=="3"){

Min.insertAtTail(packTemp);

}

}

cout<<"存放包裹完成\n";

inputPack.close();

}

void analyse(){

cout<<"以下为包裹总信息的相关数据分析\n";

cout<<"现在大货架包裹总数为："<<Max.NumOfList<<endl;

cout<<"现在中货架包裹总数为："<<Mid.NumOfList<<endl;

cout<<"现在小货架包裹总数为："<<Min.NumOfList<<endl;

cout<<"按y显示详细信息\n";

string command;

cin >> command;

if (command == "y"||command == "Y") {

cout<<"大货架包裹具体信息如下：\n";

Max.traverse();

cout<<"中货架包裹具体信息如下：\n";

Mid.traverse();

cout<<"小货架包裹具体信息如下：\n";

Min.traverse();

}

}

void admin(){

while(1){

char command;

cout<<"输入相关数字进行相关操作\n";

cout<<"\*\*\*\*\*\*\*1.手动存包裹\*\*\*\*\*\n";

cout<<"\*\*\*\*\*\*\*2.自动存包裹\*\*\*\*\*\n";

cout<<"\*\*\*\*\*\*\*3.获取分析信息\*\*\*\n";

cout<<"\*\*\*\*\*\*\*4.退出管理员系统\*\n";

cin>> command;

if (command == '1') {

packNotAuto();

}

else if (command =='2') {

packAuto();

}

else if (command =='3') {

analyse();

}

else if (command =='4') break;

else cout<<"输入指令无效，请重新输入\n";

}

}

void pickByPhone(){

string phoneOfUser;

cout<<"请输入电话号码取件\n";

while (1){

cin >> phoneOfUser;

if (Max.searchPhone(phoneOfUser)||Mid.searchPhone(phoneOfUser)||Min.searchPhone(phoneOfUser)){

while (Max.searchPhone(phoneOfUser)){

Max.deleteNodeByPhone(phoneOfUser);

}

while (Mid.searchPhone(phoneOfUser)){

Mid.deleteNodeByPhone(phoneOfUser);

}

while (Min.searchPhone(phoneOfUser)){

Min.deleteNodeByPhone(phoneOfUser);

}

cout<<"相关的所有包裹已经取出\n";

break;

}

else{

cout<<"电话号码不存在或不存在该号码所对应的包裹\n";

break;

}

}

}

void pickByCode(){

string codeOfUser;

string phoneOfUser;

cout<<"请取件码取件\n";

while (1){

cin >> codeOfUser;

if (Max.searchCode(codeOfUser)||Mid.searchCode(codeOfUser)||Min.searchCode(codeOfUser)){

if (Max.searchCode(codeOfUser)){

phoneOfUser = Max.findPhoneByCode(codeOfUser);

while (Max.searchPhone(phoneOfUser)){

Max.deleteNodeByPhone(phoneOfUser);

}

}

if (Mid.searchCode(codeOfUser)){

phoneOfUser = Mid.findPhoneByCode(codeOfUser);

while (Mid.searchPhone(phoneOfUser)){

Mid.deleteNodeByPhone(phoneOfUser);

}

}

if (Min.searchCode(codeOfUser)){

phoneOfUser = Min.findPhoneByCode(codeOfUser);

while (Min.searchPhone(phoneOfUser)){

Min.deleteNodeByPhone(phoneOfUser);

}

}

cout<<"电话号为"<<phoneOfUser<<"的相关的所有包裹已经取出\n";

break;

}

else{

cout<<"取件码不存在或不存在该取件码所对应的包裹\n";

break;

}

}

}

void users(){

while(1){

char command;

cout<<"输入相关数字进行相关操作\n";

cout<<"\*\*\*\*\*\*\*1.电话号取包裹\*\*\*\n";

cout<<"\*\*\*\*\*\*\*2.取件码取包裹\*\*\*\n";

cout<<"\*\*\*\*\*\*\*3.退出用户系统\*\*\*\n";

cin>> command;

if (command == '1') {

pickByPhone();

}

else if (command =='2') {

pickByCode();

}

else if (command =='3') break;

else cout<<"输入指令无效，请重新输入\n";

}

}

bool menu(){

while(1){

char command;

cout<<"输入相关数字表示您的身份\n";

cout<<"\*\*\*\*\*\*\*1.管理员\*\*\*\*\*\*\*\*\*\n";

cout<<"\*\*\*\*\*\*\*2.用户 \*\*\*\*\*\*\*\*\*\n";

cout<<"\*\*\*\*\*\*\*3.结束今天\*\*\*\*\*\*\*\n";

cout<<"\*\*\*\*\*\*\*4.退出系统\*\*\*\*\*\*\*\n";

cin>> command;

if (command == '1') {

admin();

}

else if (command =='2') {

users();

}

else if (command =='3') break;

else if (command =='4') return false;

else cout<<"输入指令无效，请重新输入\n";

}

return true;

}

void loadPack(){

ifstream inputPack;

inputPack.open("data.txt");

while (!inputPack.eof()){

pak packTemp;

string codePart;

inputPack >> packTemp.size;

//cout<<"请输入包裹主人电话号码\n";

inputPack >> packTemp.phone;

//cout<<"请输入包裹主人姓名\n";

inputPack >> packTemp.name;

packTemp.date = Date;

inputPack >> packTemp.packCode;

if (packTemp.size=="1"){

Max.insertAtTail(packTemp);

}

else if (packTemp.size=="2"){

Mid.insertAtTail(packTemp);

}

else if (packTemp.size=="3"){

Min.insertAtTail(packTemp);

}

}

cout<<"加载完成\n";

inputPack.close();

}

void updatePack(){

Max.deleteOutdate(Date);

Min.deleteOutdate(Date);

Mid.deleteOutdate(Date);

}

void savePack(){

ofstream savePack;

savePack.open("data.txt");

while (Max.head){

savePack << Max.head->data.size <<" ";

//cout << Max.head->data.size <<" ";

//cout<<"请输入包裹主人电话号码\n";

savePack << Max.head->data.phone<<" ";

//cout<<"请输入包裹主人姓名\n";

savePack << Max.head->data.name<<" ";

//packTemp.date = Date;

savePack << Max.head->data.packCode<<endl;

Max.deleteAtHead();

}

while (Mid.head){

savePack << Mid.head->data.size <<" ";

//cout<<"请输入包裹主人电话号码\n";

savePack << Mid.head->data.phone<<" ";

//cout<<"请输入包裹主人姓名\n";

savePack << Mid.head->data.name<<" ";

//packTemp.date = Date;

savePack << Mid.head->data.packCode<<endl;

Mid.deleteAtHead();

}

while (Min.head){

savePack << Min.head->data.size <<" ";

//cout<<"请输入包裹主人电话号码\n";

savePack << Min.head->data.phone<<" ";

//cout<<"请输入包裹主人姓名\n";

savePack << Min.head->data.name<<" ";

//packTemp.date = Date;

savePack << Min.head->data.packCode<<endl;

Min.deleteAtHead();

}

cout<<"保存完成\n";

savePack.close();

}

int main() {

bool needToExit;

loadPack();

while(1){

cout<<"现在是第"<<Date<<"天"<<endl;

cout<<"已清除过期包裹\n";

needToExit = menu();

if (!needToExit) break;

Date++;

updatePack();

cout<<"今天结束\n";

}

savePack();

return 0;

}

### 1.1.3 测试数据和结果：

#### 1.1.3.1 测试数据

##### 1.1.3.1.1 data.txt

每行从左到右依次是货架编码、电话号码，姓名和取件码

1 18907765205 lxy 11

2 15278666682 ww 21

3 18077691108 wzc 31

3 15949390045 yzw 32

##### 1.1.3.2 Input.txt

每行从左至右依次是货架编码电话号码和姓名

1 18077691108 wzc

2 17058582345 wzx

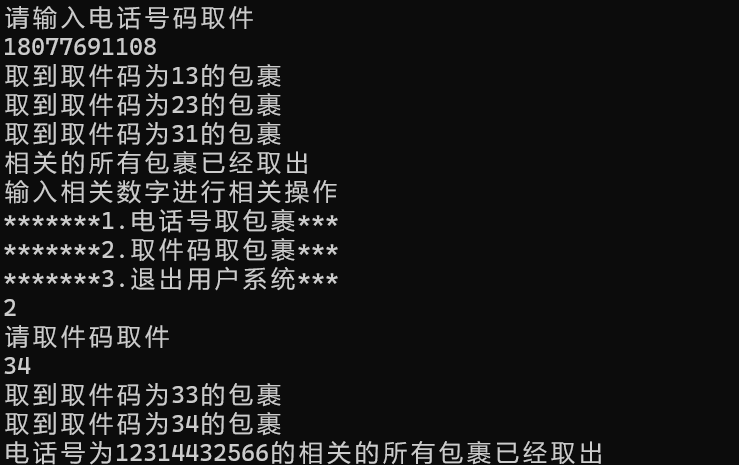
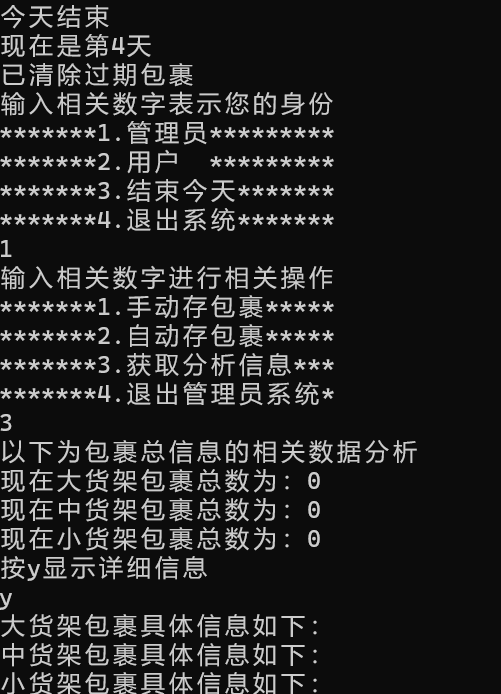
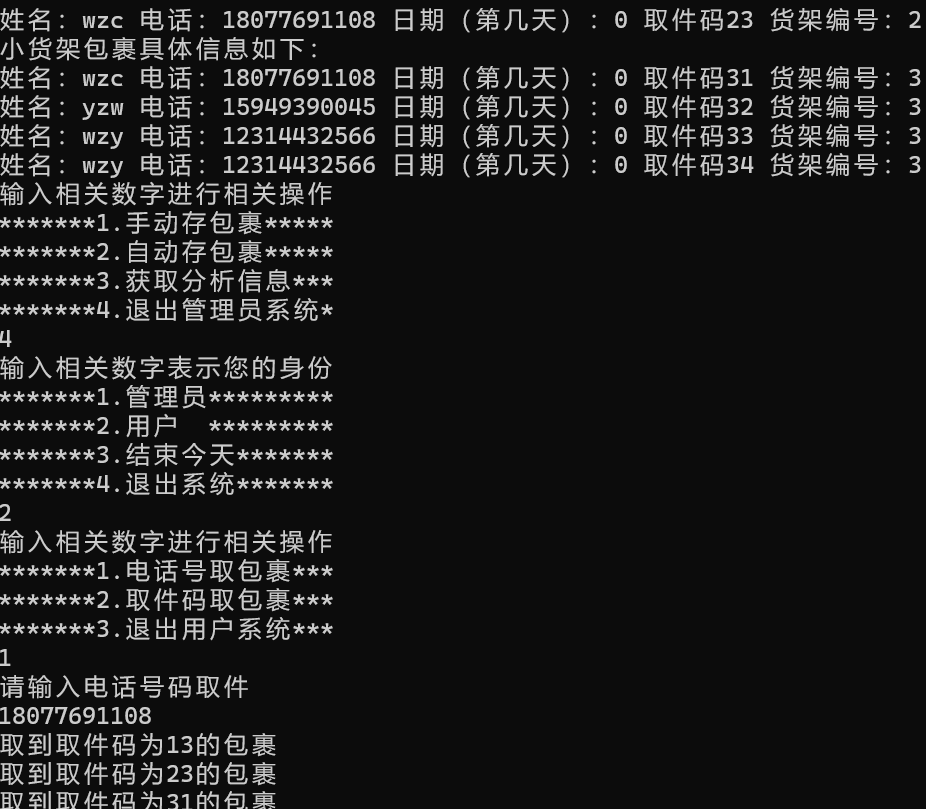
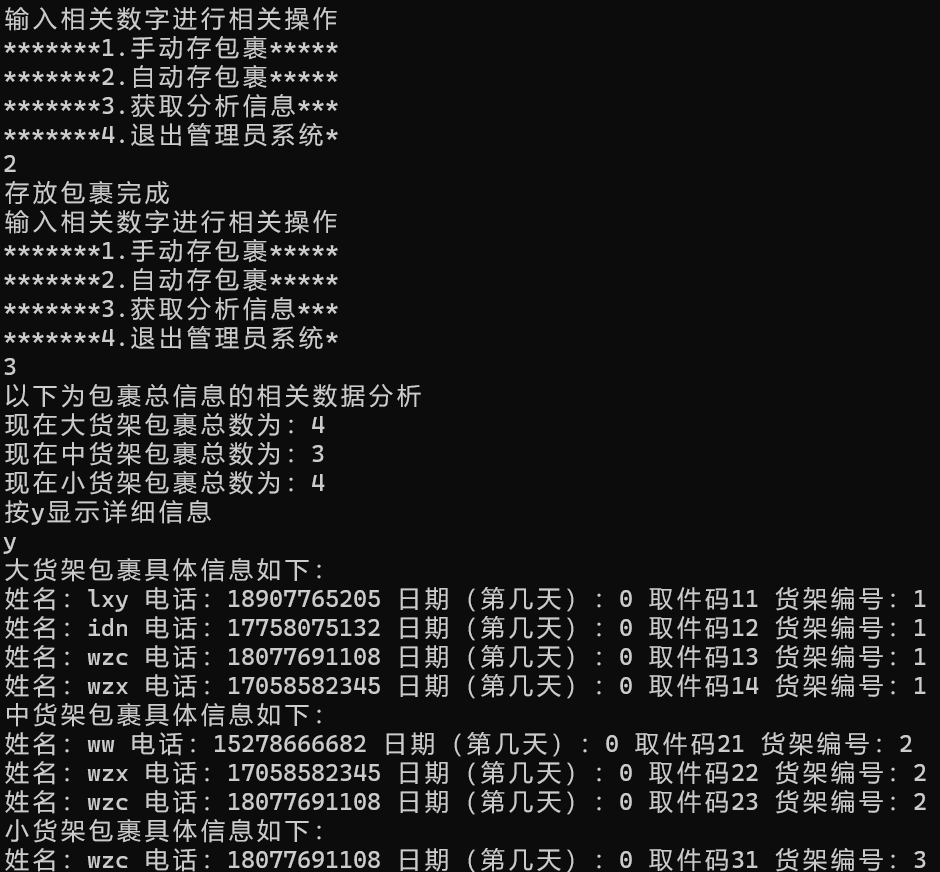
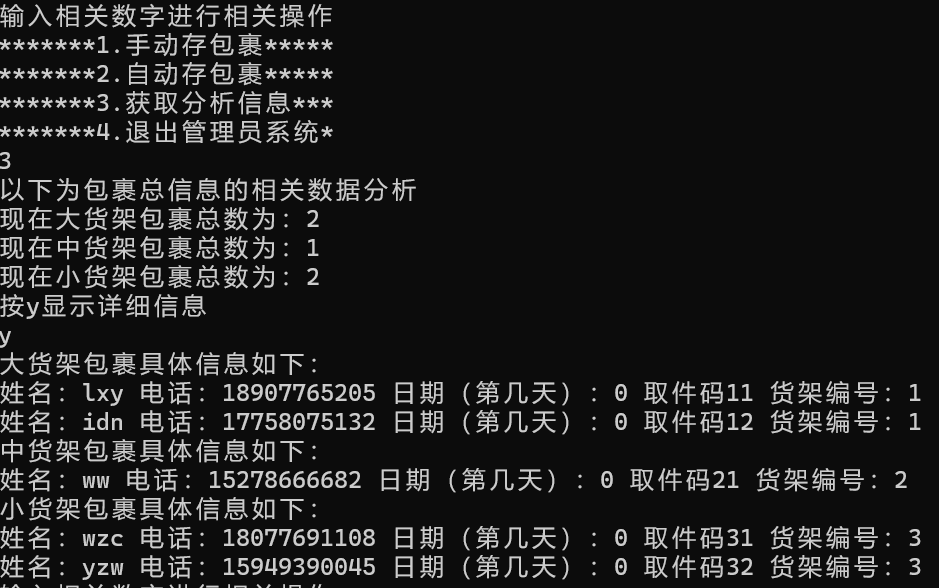
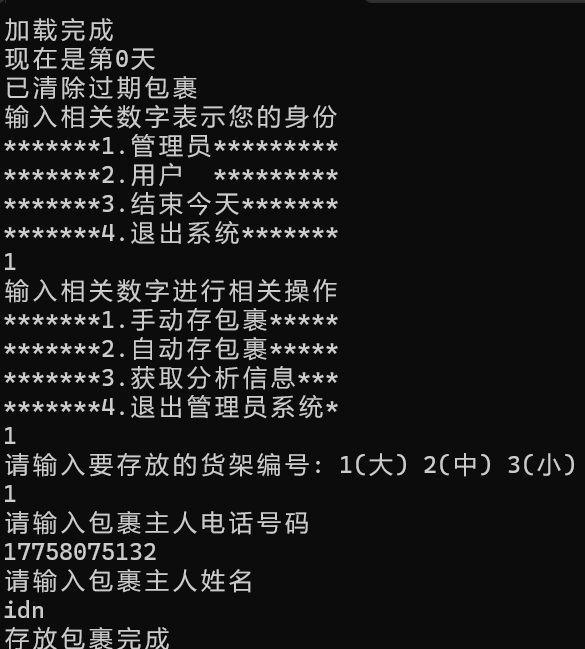
3 12314432566 wzy

1 17058582345 wzx

2 18077691108 wzc

3 12314432566 wzy

#### 1.1.3.2 测试结果



### 1.1.4 时间复杂度：

存包裹O(1)

电话号码取包裹O（N）（N最大为所有包裹数）

取件码取包裹O（2N）

清除过期包裹O (N)

### 1.1.5 反思：

部分操作可以抽象为一个函数被反复调用，减少代码量。深刻理解了线性表中的双向链表的相关操作。

### 1.1.6 改进：

### 1.1.7 该题代码行：

505行

## 1.2 算术表达式求值

### 1.2.1 数据结构：

采用了栈（stack）来实现

### 1.2.2 算法设计思想：

在依次读取字符串进行运算符优先发求算术表达式的值并同时检测错误较难实现。为了方便。

先使用iscorrect（）函数读取一遍输入进来的字符串进行检测错误，若检测出错误，提示错误信息，并使用户重新输入字符串直至字符串正确。

然后利用两个栈：存放数字的数栈和操作符的操作符栈，用运算符优先法进行中缀表达式求值计算。求值调用calculator（）函数，其中获取运算符优先级用getPriority（）函数，具体的四则运算用Calculate（）函数

信息提示和错误提示分别用INFO（）函数和ERROROUT（）函数。

为了实现用栈计算算数表达式的值，需设置两个工作栈：用于存储运算符的栈opter，以及用于存储操作数及中间结果的栈opnd。

这两个栈发生变动时将实时提示相关信息。

算法基本思想如下：

（1）首先将数栈设为空栈。

（2）依次读入表达式的每个字，读入字符若是数则入数栈，读入字符若是运算符，则将此运算符c与操作符栈的栈顶元素top比较优先级后执行相应的操作，具体操作如下：

(i)若top的优先级小于c，即top<c，则将c直接入栈，并读入下一字符赋值给c；

(ii)若top的优先级等于c，即top=c，则弹出opter的栈顶元素，并读入下一字符赋值给c，这一步目的是进行括号操作;

（iii）若top优先级高于c，即top>c，则表明可以计算，此时弹出数栈顶两个元素，并且弹出操作符栈顶的的运算符，计算后将结果放入数栈中。直至当前读入的字符为‘#’，此时求值结束。输出结果。

### 1.2.3 源程序：

#include <iostream>

#include <Windows.h>

#include <stack>

#include <queue>

#include <string>

#include <iomanip>

#include <sstream>

using namespace std;

#define red SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), FOREGROUND\_INTENSITY | FOREGROUND\_RED) //红色

#define yellow SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), FOREGROUND\_INTENSITY | FOREGROUND\_RED | FOREGROUND\_GREEN) //黄色

#define green SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), FOREGROUND\_INTENSITY | FOREGROUND\_GREEN) //绿色

#define white SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), FOREGROUND\_INTENSITY | FOREGROUND\_RED | FOREGROUND\_GREEN | FOREGROUND\_BLUE) //白色

#define ching SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), FOREGROUND\_INTENSITY | FOREGROUND\_GREEN |FOREGROUND\_BLUE) //青色

void INFO(string info) {

cout << "["; green; cout << "INFO"; white; cout << "]:" << info ;

}

void ERROROUT(string error) {

cout << "["; red; cout << "ERROR"; white; cout << "]:" << error << endl;

}

// 由于使用运算符优先法，故要获取优先级

int getPriority(char ch) {

int level = 0; // 优先级

switch(ch) {

case '(':

level = 1;

break;

case '+':

case '-':

level = 2;

break;

case '\*':

case '/':

level = 3;

break;

default:

break;

}

return level;

}

// 进行相应的计算

double Calculate(char ch, int a, int b) {

double result = 0;

switch(ch) {

case '+':

result = b + a;

break;

case '-':

result = b - a;

break;

case '\*':

result = b \* a;

break;

case '/':

if(a == 0) {

ERROROUT( "除数不能为0。\n");

exit(1);//结束程序

}

else

result = b / a;

break;

default:

break;

}

return result; // 返回计算得到的结果

}

void calculator(const string &s){

std::stack<double> m\_integer; // 数栈

std::stack<char> op; // 操作符栈

char c; // c存储操作符栈顶的元素

int num, a, b; // result存储计算的结果,a,b存储从数栈中取出的两个数

double result;

for (int i = 0; i<s.size(); i++){

ching;cout<<"当前输入的字符为：";white;cout<<s[i]<<endl;

if (s[i] == '#') continue;

if(isdigit(s[i])) { // 如果是数字

num = 0;

do {

num = num \* 10 + (s[i] - '0'); // ch - '0'根据ASCAII码，字符与数字之间的转换关系

i++;

ching;cout<<"当前输入的字符为：";white;cout<<s[i]<<endl;

}while(isdigit(s[i]));

m\_integer.push(num); // 存到数栈中

INFO("当前数已存至数栈中,该数为：");cout<<num<<endl;

}

if(s[i] == '(') { // (:左括号

op.push(s[i]);

INFO("当前左括号已存至操作符栈中\n");

}

if(s[i] == '+' || s[i] == '-' || s[i] == '\*' || s[i] == '/') { // 操作符

if(op.empty()) { // 如果栈空，直接压入栈

op.push(s[i]);

INFO("当前操作符已存至数栈中,该操作符为：");cout<<s[i]<<endl;

}

else {

// 比较栈op顶的操作符与当前操作符的优先级

// 如果当前优先级高，入栈，否则，弹出栈中的操作符，除非1.弹出操作符小于当前优先级 2.或出来( 3.或栈空

while(!op.empty()) {

c = op.top();

if(getPriority(s[i]) <= getPriority(c)) {

// 优先级低或等于

// 取出栈中操作符和数栈中两个数进行运算，再将结果放回数栈

result = 0;

a = m\_integer.top(); // 第二个操作数,因为栈是后进先出

m\_integer.pop();

INFO("有数出数栈，该数为：");cout<< a <<endl;

b = m\_integer.top(); // 第一个操作数

INFO("有数出数栈，该数为：");cout<< b <<endl;

m\_integer.pop();

result = Calculate(c, a, b); // 计算

//输出结果为两位小数

INFO("有数入数栈，该数为：");cout<<setiosflags(ios::fixed)<<setprecision(2)<<result<<endl;

m\_integer.push(result); // 把计算结果压入栈中

op.pop(); // 操作符出栈

INFO("有操作符出操作符栈，该操作符为");cout<< c <<endl;

} else // 当前操作符优先级高于栈中操作符

break;

} // while结束

op.push(s[i]); // 防止不断的推出操作符，最后空栈了;或者当前操作符优先级高了

INFO("当前操作符已存至数栈中,该操作符为：");cout<<s[i]<<endl;

} // else

}

if(s[i] == ')') { // 如果是右括号，一直推出栈中操作符，直到遇到左括号(

while(op.top() != '(') {

c = op.top(); // 取出栈顶操作符

// 取出栈中操作符和数栈中两个数进行运算，再将结果放回数栈

result = 0;

a = m\_integer.top(); // 第二个操作数,因为栈是后进先出

m\_integer.pop();

INFO("有数出数栈，该数为：");cout<< a <<endl;

b = m\_integer.top(); // 第一个操作数

INFO("有数出数栈，该数为：");cout<< b <<endl;

m\_integer.pop();

result = Calculate(c, a, b); // 计算

//输出结果为两位小数

INFO("有数入数栈，该数为：");cout<<setiosflags(ios::fixed)<<setprecision(2)<<result<<endl;

m\_integer.push(result); // 把计算结果压入栈中

INFO("有操作符出操作符栈，该操作符为");cout<< c <<endl;

op.pop(); // 把操作符推出栈

}

op.pop(); // 把左括号(推出栈

INFO("当前左括号已弹出操作符栈\n");

}

}

while(!op.empty()) { // 当栈不空，继续取出操作符进行计算

c = op.top(); // 取出栈顶操作符

// 取出栈中操作符和数栈中两个数进行运算，再将结果放回数栈

result = 0;

a = m\_integer.top(); // 第二个操作数,因为栈是后进先出

m\_integer.pop();

INFO("有数出数栈，该数为：");cout<< a <<endl;

b = m\_integer.top(); // 第一个操作数

INFO("有数出数栈，该数为：");cout<< b <<endl;

m\_integer.pop();

result = Calculate(c, a, b); // 计算

//输出结果为两位小数

INFO("有数入数栈，该数为：");cout<<setiosflags(ios::fixed)<<setprecision(2)<<result<<endl;

m\_integer.push(result); // 把计算结果压入栈中

INFO("有操作符出操作符栈，该操作符为");cout<< c <<endl;

op.pop(); // 把操作符推出栈

}

INFO("有数出数栈，该数为：");cout<<setiosflags(ios::fixed)<<setprecision(2)<<m\_integer.top()<<endl;

INFO("结果为");

cout<<setiosflags(ios::fixed)<<setprecision(2)<<m\_integer.top()<<endl;

m\_integer.pop();

}

bool iscorrect(const string &s)

{

int zuokuohao = 0;

char ch;

int onlyOPND = 0;

int isColiedOPND = 0;

stack<char> brackets;

for (int i = 0; i<s.size(); i++)

{

if (i!=0 && i != s.size()-1 && s[i] == '#'){

ERROROUT( "表达式中间含有#，请重新输入" );cout << endl;

return false;

}

if (s[i] == '(') {

//if (s[i])

brackets.push(s[i]);

isColiedOPND == 0;

}

if (isdigit(s[i])) isColiedOPND = 1;

if (s[i] == '+' || s[i] == '-' || s[i] == '\*' || s[i] == '/'){

if (isColiedOPND == 0) {

ERROROUT("运算符位置错误使得为一目计算符或者没有任何与该运算符匹配的操作数");cout<< endl;

return false;

}

else isColiedOPND = 0;

}

if (!(isdigit(s[i]) || s[i] == '+' || s[i] == '-' || s[i] == '\*' || s[i] == '/' || s[i] == '#' || s[i] == '(' || s[i] == ')')){

ERROROUT( "含有非法字符，请重新输入" );cout << endl;

return false;

}

else if (s[i] == ')'){

if (brackets.empty())

{

ERROROUT("括号没有成对，请重新输入");cout<< endl;

return false;

}

if (isColiedOPND == 0){

ERROROUT( "右括号左边无数字" );cout << endl;

Return false;

}

else brackets.pop();

}

}

if (s == "" || s == " " || s == "\n"||s == "#"||s=="##"){

ERROROUT("无输入，请重新输入" );cout << endl;

return false;

}

if (s[0] != '#'){

ERROROUT( "没有以#开头，请重新输入"); cout<< endl;

return false;

}

if (s[s.size() - 1] != '#'){

ERROROUT( "没有以#结尾，请重新输入"); cout<< endl;

return false;

}

if (brackets.empty()) return true;

else {

ERROROUT("括号没有成对，请重新输入");cout<< endl;

return false;

}

}

int main()

{

int flag = 1;

INFO("请输入算式表达式,以#开头，并以#结尾：\n");

string s;

while (flag)

{

cin >> s;

if (iscorrect(s)) flag = 0;

}

INFO("输出序列如下：");

cout << s <<endl;

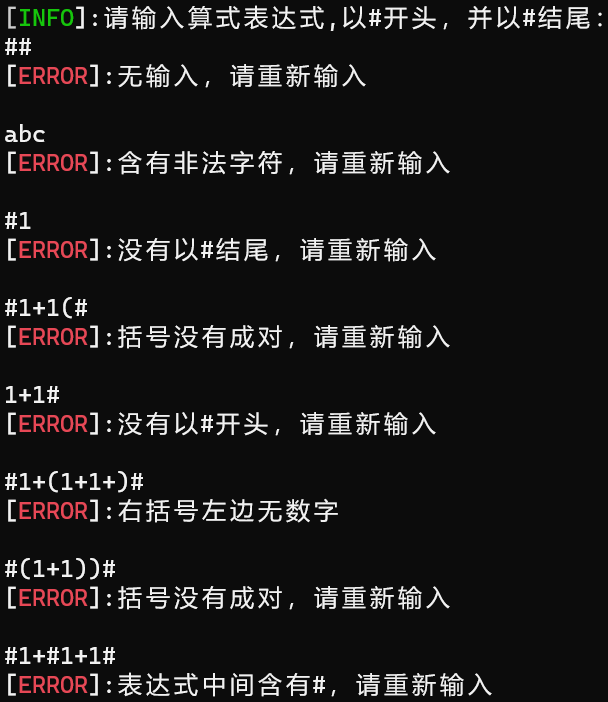
calculator(s);

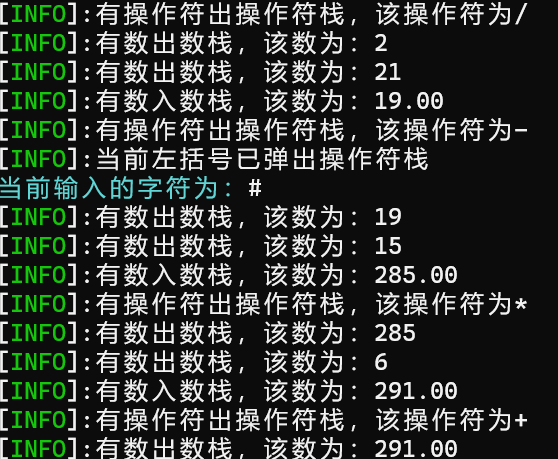
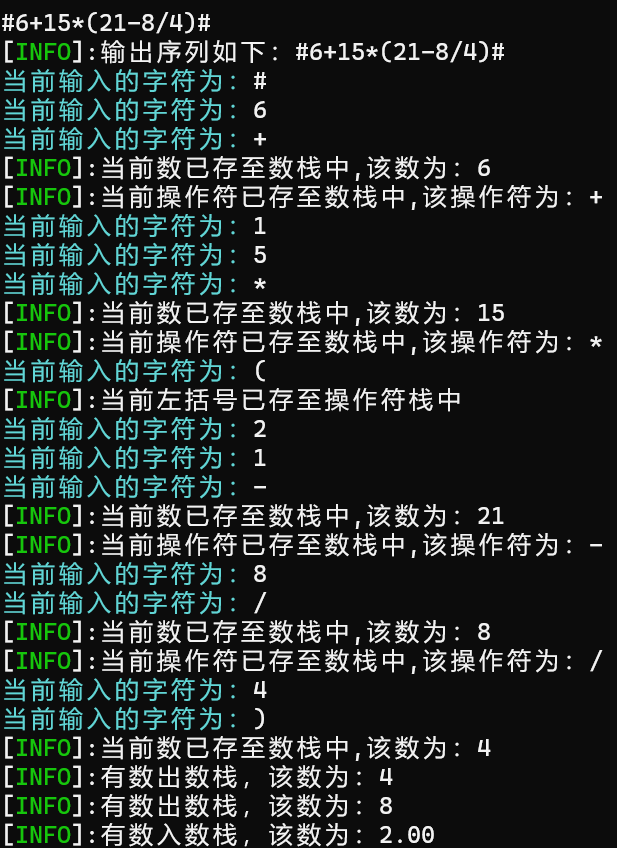
system("pause");

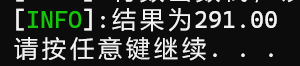
return 0;

}

### 1.2.4测试数据和结果







### 1.2.5 时间复杂度：

O（N+KM+N+M），N为数字个数，M为符号个数，K为一常数，遍历N次即可获得数字，根据符号个数进行出入栈，理论上为一个线性变化的函数，故为KM，检测错误需要（N+M）。

### 1.2.6 反思：

一些错误情况未细分提示。

未对小数的情况进行操作。

### 1.2.7 改进：

### 1.2.8 该题代码行：

252行

## 1.3 特殊路径统计

### 1.3.1 数据结构：

采用了队列（queue）来实现BFS

采用邻接表来实现图

非表头结点机构体（ArcNode）

typedef struct ArcNode {

int adjvex;

struct ArcNode\* nextarc;

}ArcNode;

表头结点 结构体(VexNode)

typedef struct VexNode {

int maxdata; //所在路径上的最大值，判断是否为最大路径

ArcNode\* firstarc;

}VexNode;

邻接表结构体（Graph）

typedef struct Graph { //邻接表结构类型

VexNode\* VNode; //定义邻接表

int vexnum; //顶点个数

}Graph;

### 1.3.2 算法设计思想：

首先调用CreatGraph()函数获取创建邻接表。然后调用CountRoad()函数获取特殊路径数，该函数中又对一个个点调用BFS（）函数中找到并输出特殊路径。

在BFS函数中，先保存当前路径的最大值，然后在拓展时判断当前结点是否比路径起点小，若小，则不让该点加入BFS辅助拓展队列中；若大，则判断是否比当前路径最大值小。

若比当前路径最大值小，则加入拓展队列中，若大，则为一条新的特殊路径，输出该路径并更新最大路径最大值。

### 1.3.3 源程序：

#include <iostream>

#include <queue>

#include <stdlib.h>

#include <Windows.h>

using namespace std;

#define red SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), FOREGROUND\_INTENSITY | FOREGROUND\_RED) //红色

#define yellow SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), FOREGROUND\_INTENSITY | FOREGROUND\_RED | FOREGROUND\_GREEN) //黄色

#define green SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), FOREGROUND\_INTENSITY | FOREGROUND\_GREEN) //绿色

#define white SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), FOREGROUND\_INTENSITY | FOREGROUND\_RED | FOREGROUND\_GREEN | FOREGROUND\_BLUE) //白色

#define ching SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), FOREGROUND\_INTENSITY | FOREGROUND\_GREEN |FOREGROUND\_BLUE) //青色

void INFO(string info) {

cout << "["; green; cout << "INFO"; white; cout << "]:" << info ;

}

void ERROROUT(string error) {

cout << "["; red; cout << "ERROR"; white; cout << "]:" << error << endl;

}

//以邻接表为数据结构

//非表头结点

typedef struct ArcNode {

int adjvex;

struct ArcNode\* nextarc;

}ArcNode;

//表头结点

typedef struct VexNode {

//int data;

int maxdata; //所在路径上的最大值，判断是否为最大路径

ArcNode\* firstarc;

}VexNode;

typedef struct Graph { //邻接表结构类型

VexNode\* VNode; //定义邻接表

int vexnum; //顶点个数

}Graph;

int \*visit; //标记点是否被访问过，动态申请

int count = 0; //特殊路径总数

int num; //顶点个数

int lastSecondeRoad = 0; //第二大的点的特殊路径数

int isConnectedlast = 0; //第一大的点与第二大的点是否联通

void CreatGraph(Graph &G){ //动态申请空间建图

while (1){

INFO("请输入顶点个数\n");

cin >> num;

//顶点个数小于2错误

if (num <= 1)

ERROROUT("顶点个数过少，不能构成路径,请重新输入顶点个数\n");

else

break;

}

if (num == 2){ //顶点个数为2直接得路径数不必建图

int fake,fakes;

INFO("请输入相关信息：\n");

cin >> fake; cin >> fakes;

return ;

}

else {

G.vexnum = num;

G.VNode = (VexNode\*)malloc ((num+1)\* sizeof(VexNode));

}

//为访问数组动态分配空间

visit = (int\*)malloc((num+1) \* sizeof(int));

//邻接表初始化，所有单向链表均为空表

for (int i = 1; i <= G.vexnum; i++) {

//G.VNode[i].data = i+1;

G.VNode[i].firstarc = NULL;

}

int a;

ArcNode \*p,\*q;

INFO("请输入相关信息：\n");

for (int i = 1; i <= G.vexnum; i++){

cin >> a;

//判断第一大的点和第二点是否联通

if ((i == G.vexnum-1 && a == G.vexnum)||(a == G.vexnum-1 && i == G.vexnum))

isConnectedlast = 1;

if (a > 0) {

p = (ArcNode\*)malloc(sizeof(ArcNode));

p->nextarc = NULL;

p->adjvex = i;

q = G.VNode[a].firstarc;

if (q == NULL) {

G.VNode[a].firstarc = p;

}

else {

while (q->nextarc != NULL) {

q = q->nextarc;

}

q->nextarc = p;

}

//无向图创建对称边

p = (ArcNode\*)malloc(sizeof(ArcNode));

p->nextarc = NULL;

p->adjvex = a;

q = G.VNode[i].firstarc;

if (q == NULL) {

G.VNode[i].firstarc = p;

}

else {

while (q->nextarc != NULL) {

q = q->nextarc;

}

q->nextarc = p;

}

}

}

}

void BFS(Graph G,int i){ //BFS获取路径

queue<int> q;

visit[i] = 1;

q.push(i);

int shouldNewline = 0;

int maxInRoad;

while (!q.empty()){

maxInRoad = q.front();

q.pop();

ArcNode \*p = G.VNode[maxInRoad].firstarc;

while (p){

if (!visit[p->adjvex] && p->adjvex >= i) {

if (p->adjvex >= G.VNode[maxInRoad].maxdata){

//若当前结点大于当前路径的最大值，更新

count++;

cout << i << "->" << p->adjvex << " ";

shouldNewline = 1;

}

else{

G.VNode[p->adjvex].maxdata = G.VNode[maxInRoad].maxdata;

}

visit[p->adjvex] = 1;

q.push(p->adjvex);

}

p = p->nextarc;

}

}

if (shouldNewline){

cout << endl;

}

}

int CountRoad (Graph &G){ //计算特殊路径

INFO("特殊路径为：\n") ;

if (num == 2){

cout <<"1->2"<<endl;

count = 2; //顶点个数为2直接得出结果

}

else{

for (int i = 1; i < G.vexnum-1; i++){

for (int i = 1; i <= G.vexnum; i++){

visit[i] = 0; //辅助数组Visit初始化

G.VNode[i].maxdata = i; //当前结点所在路径上的最大值初始化

}

BFS(G, i);

}

if (isConnectedlast){

count++;

cout <<G.vexnum-1<<"->"<<G.vexnum<<endl;

}

}

INFO("特殊路径数为：");cout<<count<<endl;

}

int main(){

Graph G;

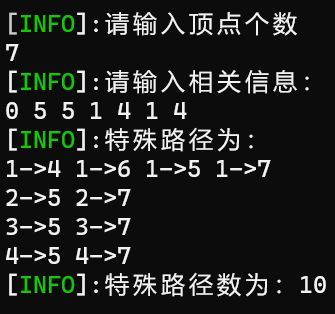
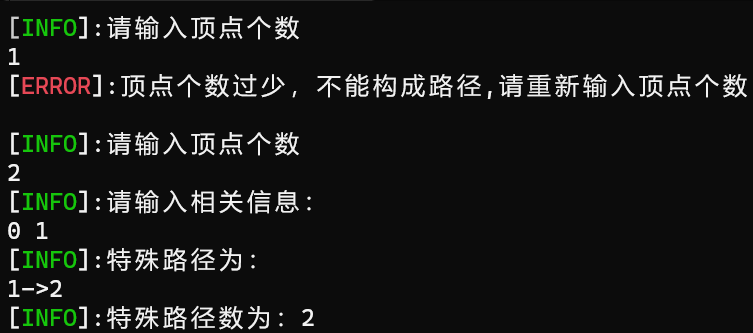
CreatGraph(G);

CountRoad(G);

return 0;

}

### 1.3.4 测试数据和结果



### 1.3.5 时间复杂度：

BFS的时间复杂度为O（n^2）

如果顶点个数为2，则为O（1）

### 1.3.6 改进：

首先，若顶点个数小于2，则不可能有特殊路径。

其次，若顶点个数为2，特殊路径肯定只有一个，即1->2。那就不必创建图和进行BFS遍历了，若出现这种情况的时间复杂度为O（1）

然后，肯定不会出现路径起点为**最大的数（biggest，以下简称B）**，故不对最大的数进行BFS遍历，减少了0-3次的拓展时间。

并且，路径起点为**第二大的数（second biggest，以下简称SB）**.

路径数要么为0，要么为 1: SB-> B. 那么也不需要对第二大的店进行BFS拓展，只需要判断它与最大的数是否联通即可，这个可以在输入父结点信息时就可以判断。减少了最少1次，最多3次的BFS拓展。

### 1.3.7 反思：

其实，若起点为**第三大的数（简称TB）**，还可以继续改进：根据之前获取的B与SB是否相连，若B与SB相连，则TB的可成路径邻接点肯定只能要么B，要么SB，是B特殊路径有1条，是SB特殊路径有两条。若B与SB，判断TB的邻接点里有B和SB这两个的几个，有几个就有几条特殊路径。

如果是对TB常规BFS扩展，当TB的邻接点里有SB时，还需要SB进行扩展，增加了额外的不必要的拓展。

此外，该程序的CreateGraph函数和 CountRoad函数，共同使用的变量较多，在程序里我将这些设为全局变量来减少函数传参个数，恰当的办法是将它们（共同使用的变量和函数）封装为一个类。

这两个函数的缺点还有一个就是耦合度较高，以后编程时我要注意到这个问题。

### 1.3.8 该题代码行：

166行

## 1.4 公交路线提示

### 1.4.1 数据结构：

采用了邻接表存储公交路线图的数据，邻接表的相关类或结构体如下：

class ArcNode //邻接表结点类

{

public:

int id; //边的下一个id

ArcNode \*next;

ArcNode(int id) : id(id), next(nullptr) {}

};

class VertexHead //邻接表表头类

{

public:

int id; //表头id

ArcNode \*first;

VertexHead() : first(nullptr) {}

};

//存储公交路线图中的路线类

class Line

{

public:

int no; //路线编号

};

//存储公交路线图中的站点类

class Station

{

public:

int id; //站点id

string name; //站点名称

set<int> buss; //经过站点的公交车的路线，set便于去重以及集合运算

Station() {}

Station(string name, int busNo) : name(name) { buss.insert(busNo); }

};

//将路线+站点封装为一个类

class LineBus

{

public:

Line line; //路线

vector<int> stations; //每一条路线的所有站点

};

//邻接表类记录相关数据

class Graph

{

public:

VertexHead heads[MAX]; //所有的表头

vector<LineBus> lines; //文件中的所有路线

vector<Station> stations; //存储所有的站点

vector<int> lines\_no; //所有路线编号

int line\_num; //路线数

int station\_num; //站点数

Graph()

{

for (int i = 0; i < MAX; ++i)

{

heads[i].id = i;

}

}

void insertArc(int &s1, int &s2)

{

ArcNode \*p = heads[s1].first;

while (p != nullptr)

{

if (p->id == s2)

{

return;

}

p = p->next;

}

ArcNode \*arc = new ArcNode(s2);

arc->next = heads[s1].first;

heads[s1].first = arc;

ArcNode \*arc2 = new ArcNode(s1);

arc2->next = heads[s2].first;

heads[s2].first = arc2;

}

};

### 1.4.2 算法设计思想：

#### 1.4.2.1 核心算法

获取经过站点最少路线时，使用迪杰斯特拉算法；获取最少换乘路径时，使用BFS算法。

#### 1.4.2.2 程序其他部分说明以及为何要大量使用STL库中的容器

将核心的显示功能（menu）封装为函数和根据选择封装为一个函数，进行数据处理（enquiry）封装为一个函数。将这两个函数需要的初始化和处理细节分别封装为相关函数。

本程序的核心在于如何承载数据，以及如何方便地使用相关数据。

由于需要进行类似于站点与路线的一对多或多对一映射，以及需要进行类似于集合的运算，还需要快速查找并获取到相关数据。同时，这些数据之间的对应个数并不确定。因而本程序需要大量使用STL库中的相关容器。

#### 1.4.2.3 相关容器

将站点转换为数字可以便捷地进行相关操作，而要在之后的操作中快速查找到相关数字所对应的站点或者相关站点所对应的数字。因此使用两个map容器进行相互映射。

BFS算法需要使用队列，故使用queue容器，输出时则需要使用stack容器

为了快速找到两个站点间有哪些路线可以相通，最好是将将每个站点所“拥有”的路线作为一个集合，两个集合进行交集运算即可

要转车的点必要有多个，同时每个站点又大概率又多个路线，且个数不是固定的。故转车的点的路线用vector存储，然后将点和路线有用pair存储（pair<int, vector<int>>），多个点又用vector存储。

将转车的点（pair<int, vector<int>>）以及我们要前往的点所映射的数字又可以合并为一个pair，再用vector存储，可以获得存储换乘路线的一个个结点，连起来便于输出路线。

#### 1.4.2.4 函数说明

Menu函数，输出菜单

Enquiry函数，根据指令调用相关函数

Check函数，输入起点和终点时在map容器里查找是否有对应的站点，方便提示用户

readFileRoutine函数，从文件中读取公交路线，获取每条路线号并存放进相关vector容器里，读取该路线的所有站点存放进邻接表里，如果读取到的站点之前就出现过，那么它会是某一条路线的始发站，在对应站点的表头插入下一站点。

Dijkstra函数，找到最少站点路径，使用Dijkstra算法。

findLeastTransfer函数，找到最少换乘路线，使用BFS算法，进行拓展时用到intersection函数

intersection函数，获取两个站点之间路线的交集

show函数，输出路线，根据参数传进来的栈，进行栈pop操作然后将找到的相映射的所要输出信息存进buss，再将buss处理输出最终路线即可

### 1.4.3 源程序：

#### 1.4.3.1 Buses.h

#ifndef BUSES\_H

#define BUSES\_H

#include <bits/stdc++.h>

using namespace std;

const int MAX = 12000;

const int line\_max = 2000; //最多2000条公交路线

class ArcNode

{

public:

int id; //边的下一个id

ArcNode \*next;

ArcNode(int id) : id(id), next(nullptr) {}

};

class VertexHead

{

public:

int id; //表头id

ArcNode \*first;

VertexHead() : first(nullptr) {}

};

//路线类

class Line

{

public:

int no; //路线编号

};

//站点类

class Station

{

public:

int id; //站点id

string name; //站点名称

set<int> buss; //经过站点的公交车的路线

Station() {}

Station(string name, int busNo) : name(name) { buss.insert(busNo); }

};

//路线+站点

class LineBus

{

public:

Line line; //路线

vector<int> stations; //每一条路线的所有站点

};

class Graph

{

public:

VertexHead heads[MAX]; //所有的表头

vector<LineBus> lines; //文件中的所有路线

vector<Station> stations; //存储所有的站点

vector<int> lines\_no; //所有路线编号

int line\_num; //路线数

int station\_num; //站点数

Graph()

{

for (int i = 0; i < MAX; ++i)

{

heads[i].id = i;

}

}

void insertArc(int &s1, int &s2)

{

ArcNode \*p = heads[s1].first;

while (p != nullptr)

{

if (p->id == s2)

{

return;

}

p = p->next;

}

ArcNode \*arc = new ArcNode(s2);

arc->next = heads[s1].first;

heads[s1].first = arc;

ArcNode \*arc2 = new ArcNode(s1);

arc2->next = heads[s2].first;

heads[s2].first = arc2;

}

};

#endif

#### 1.4.3.2 Main.cpp

#include "buses.h"

#include <bits/stdc++.h>

using namespace std;

void menu();

map<string, int> maps;

map<int, string> remaps;

vector<pair<int, vector<int>>> path; //存放转车点

vector<pair<pair<int, vector<int>>, int>> path2; //存放转车点

Graph \*graph = new Graph();

/\* 从文件中读取公交线路 \*/

void readFileRoutine()

{

ifstream in("routine.txt");

if (in.fail())

{

cout << "公交线路打开失败！" << endl;

exit(0);

}

string bus;

string stations;

string station;

stringstream ss;

int count = 0;

while (in.peek() != EOF)

{

LineBus cur\_line;

//获取每一条线路号

getline(in, bus, ' ');

int busNo = atoi(bus.substr(0, bus.size() - 1).c\_str());

cur\_line.line.no = busNo;

graph->lines\_no.push\_back(busNo);

//获取该路线所有的站点

getline(in, stations, '\n');

while (stations[0] == ' ')

{

stations = stations.substr(1, stations.size() - 1);

}

ss.str(stations);

while (getline(ss, station, ','))

{

// 1. 判断station是否已经出现在了图中

bool flag = false;

int index = -1;

for (int i = 0; i < graph->stations.size(); ++i)

{

if (graph->stations[i].name == station)

{

flag = true;

index = i;

break;

}

}

if (flag)

{

//如果当前站点之前就出现过

graph->stations[index].buss.insert(busNo);

cur\_line.stations.push\_back(graph->stations[index].id);

if (cur\_line.stations.size() != 1)

{

//说明是始发站

graph->insertArc(cur\_line.stations[cur\_line.stations.size() - 1],

cur\_line.stations[cur\_line.stations.size() - 2]);

}

}

else

{

Station s;

count++;

s.id = count;

s.name = station;

s.buss.insert(busNo);

graph->stations.push\_back(s);

maps.insert(make\_pair(station, count));

remaps.insert(make\_pair(count, station));

cur\_line.stations.push\_back(count);

if (cur\_line.stations.size() != 1)

{

//说明是始发站

graph->insertArc(cur\_line.stations[cur\_line.stations.size() - 1],

cur\_line.stations[cur\_line.stations.size() - 2]);

}

}

}

graph->lines.push\_back(cur\_line);

bus = "";

ss.clear();

ss.str("");

stations = "";

station = "";

}

in.close();

graph->line\_num = graph->lines\_no.size();

graph->station\_num = graph->stations.size();

}

/\* 获取两个站点之间路线的交集 \*/

bool intersection(int start\_id, int end\_id)

{

vector<int> inter(line\_max);

int s\_index = -1;

int e\_index = -1;

for (size\_t i = 0; i < graph->stations.size(); i++)

{

if (s\_index != -1 && e\_index != -1)

{

break;

}

else

{

if (graph->stations[i].id == start\_id)

{

s\_index = i;

}

else if (graph->stations[i].id == end\_id)

{

e\_index = i;

}

}

}

auto it =

set\_intersection(graph->stations[s\_index].buss.begin(),

graph->stations[s\_index].buss.end(),

graph->stations[e\_index].buss.begin(),

graph->stations[e\_index].buss.end(), inter.begin());

inter.resize(it - inter.begin());

if (inter.empty())

{

return false;

}

else

{

path.push\_back(make\_pair(start\_id, inter));

path2.push\_back(make\_pair(make\_pair(start\_id, inter), end\_id));

return true;

}

}

int min(int dist[], bool flag[], int size)

{

int min = MAX;

int min\_index = -1;

for (int i = 1; i <= size; ++i)

{

if (!flag[i] && dist[i] < min)

{

min = dist[i];

min\_index = i;

}

}

return min\_index;

}

void show(stack<int> line, string start, string end)

{

vector<vector<int>> buss;

int \*stations = new int[line.size()];

int count = 0;

while (!line.empty())

{

int top = line.top();

line.pop();

stations[count] = top;

if (line.empty())

{

continue;

}

int next = line.top();

vector<int> tmp;

for (set<int>::iterator it1 = graph->stations[top].buss.begin();

it1 != graph->stations[top].buss.end(); ++it1)

{

for (set<int>::iterator it2 = graph->stations[next].buss.begin();

it2 != graph->stations[next].buss.end(); ++it2)

{

if (\*it1 == \*it2)

{

tmp.push\_back(\*it1);

}

}

}

buss.push\_back(tmp);

++count;

}

/\* 输出最终路线 \*/

int cur\_line = buss[0][0];

cout << "从 " << start << " 到 " << end << " 的路线如下： \n";

int pre\_line = cur\_line;

for (int i = 0; i < count; ++i)

{

bool flag = false;

for (int j = 0; j < buss[i].size(); ++j)

{

if (buss[i][j] == cur\_line)

{

flag = true;

break;

}

}

if (!flag)

{

if (buss[i].size() > 0)

{

cur\_line = buss[i][0];

cout << "\t乘坐： " << pre\_line << " 号公交到 "

<< graph->stations[i].name << " \n";

pre\_line = cur\_line;

}

}

}

cout << "\t乘坐： " << pre\_line << " 号公交到 " << end << " \n";

}

void Dijkstra(string start, string end)

{

int start\_id = maps[start];

int end\_id = maps[end];

int size = graph->stations.size();

int \*dist = new int[size + 1];

int \*p = new int[size + 1];

bool \*flag = new bool[size + 1];

for (int i = 0; i <= size; ++i)

{ /\*1. 初始化\*/

flag[i] = false;

dist[i] = MAX;

p[i] = 0;

}

ArcNode \*ptr = graph->heads[start\_id].first;

while (ptr != NULL)

{

dist[ptr->id] = 1;

p[ptr->id] = start\_id;

ptr = ptr->next;

}

flag[start\_id] = true;

for (size\_t i = 1; i < size; i++)

{

if (flag[end\_id])

{

break;

}

int min\_index = min(dist, flag, size);

flag[min\_index] = true;

int min\_dist = dist[min\_index];

/\* 更新 \*/

if (min\_dist < MAX)

{

for (size\_t j = 1; j <= size; j++)

{

if (!flag[j])

{

int distance = MAX;

ArcNode \*ptr = graph->heads[min\_index].first;

while (ptr != nullptr)

{

if (ptr->id == j)

{

distance = 1;

break;

}

ptr = ptr->next;

}

if (!flag[j] && distance != MAX && min\_dist + distance < dist[j])

{

dist[j] = min\_dist + distance;

p[j] = min\_index;

}

}

}

}

}

int cur\_index = end\_id;

int pre\_index = p[cur\_index];

stack<int> line;

while (pre\_index != cur\_index)

{

line.push(cur\_index);

cur\_index = pre\_index;

pre\_index = p[pre\_index];

}

line.push(start\_id);

show(line, remaps[start\_id], remaps[end\_id]);

delete[] dist;

delete[] p;

delete[] flag;

}

void findLeastTransfer(string start, string end)

{

//找到起始点和终点

int start\_id = maps[start];

int end\_id = maps[end];

//设定转车最大次数

int N = 4;

int n = 0;

queue<pair<pair<int, int>, int>> que;

que.push(make\_pair(make\_pair(start\_id, 0), 0));

int pre\_level = 0;

int preParent = 0;

while (n < N)

{

int parent = que.front().second;

start\_id = que.front().first.first;

int level = que.front().first.second;

if (level != pre\_level)

{

n++;

pre\_level = level;

}

que.pop();

if (parent != 0)

{

intersection(parent, start\_id);

}

bool flag = intersection(start\_id, end\_id);

if (flag)

{ //找到通路

break;

}

else

{ //没有找到

if (!path.empty())

{

path.pop\_back();

}

int s\_index = -1;

for (size\_t i = 0; i < graph->stations.size(); i++)

{

if (graph->stations[i].id == start\_id)

{

s\_index = i;

break;

}

}

//把所有跟跟站点相关的站点都放入队列 d

for (auto it = graph->stations[s\_index].buss.begin(); it != graph->stations[s\_index].buss.end(); ++it)

{

for (size\_t i = 0; i < graph->lines.size(); i++)

{

if (graph->lines[i].line.no == \*it)

{

for (size\_t j = 0; j < graph->lines[i].stations.size(); j++)

{

if (graph->lines[i].stations[j] != start\_id)

{

que.push(make\_pair(make\_pair(graph->lines[i].stations[j], level + 1), start\_id));

}

}

}

}

}

}

}

if (n >= N)

{

cout << "从 " << start << " 到 " << end

<< " 乘坐公交车需要大于10次转车，建议选择其它交通工具！";

return;

}

if (path.size() == 0)

{

cout << "查询失败！" << endl;

exit(0);

}

stack<pair<int, vector<int>>> final\_res; //最终公交车的结果

auto tmp = path2[path2.size() - 1];

final\_res.push(tmp.first);

int index = path2.size() - 2;

if (index >= 0)

{

int bef\_end = tmp.first.first;

for (int i = index; i >= 0; --i)

{

int i\_index = -1;

for (int j = i; j >= 0; --j)

{

if (path2[j].second == bef\_end)

{

i\_index = j;

}

}

if (i\_index != -1)

{

final\_res.push(path2[i\_index].first);

bef\_end = path2[i\_index].first.first;

}

i = i\_index;

--i;

}

}

cout << "\n从 " << start << " 到 " << end << " 的乘车路线如下： \n\t";

auto t\_start = final\_res.top();

final\_res.pop();

cout << "从 " << remaps[t\_start.first] << " 乘坐： ";

for (size\_t j = 0; j < t\_start.second.size(); j++)

{

cout << t\_start.second[j] << "线";

if (j != t\_start.second.size() - 1)

{

cout << " 或 ";

}

}

while (!final\_res.empty())

{

auto t = final\_res.top();

cout << " 到 " << remaps[t.first] << ",然后乘坐: ";

for (size\_t j = 0; j < t.second.size(); j++)

{

cout << t.second[j] << "线";

if (j != t.second.size() - 1)

{

cout << " 或 ";

}

}

final\_res.pop();

}

cout << " 到终点站： " << end;

}

bool check(string start, string end)

{

return maps.find(start) != maps.end() && maps.find(end) != maps.end();

}

void enquiry(int num)

{

string start;

string end;

cout << "\n请输入起点： ";

cin >> start;

cout << "请输入终点： ";

cin >> end;

if (check(start, end))

{

if (start == end)

{

cout << "输入的起始站点与终点一致！无需换乘！" << endl;

return;

}

if (num == 1)

{

findLeastTransfer(start, end);

path.clear();

}

else if (num == 2)

{

Dijkstra(start, end);

path.clear();

}

}

else

{

cout << "输入有误，请重新输入！" << endl;

}

cout << "\n\n";

}

int main()

{

readFileRoutine();

int num = 0;

while (1)

{

system("cls");

menu();

cout << "\n\t请输入你要查询的方式： ";

cin >> num;

switch (num)

{

case 1:

{

enquiry(1);

system("pause");

break;

}

case 2:

{

enquiry(2);

system("pause");

break;

}

case 3:

{

cout << "\n\t谢谢使用！" << endl;

exit(0);

}

default:

break;

}

}

return 0;

}

void menu()

{

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

cout << "\*\* 欢迎使用公交查询系统 \*\*" << endl;

cout << "\*\* [1] 查询最少换乘路线 \*\*" << endl;

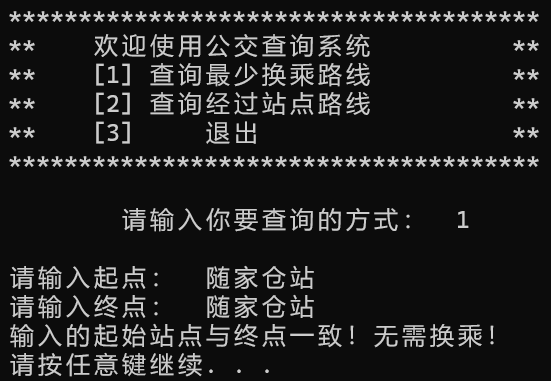
cout << "\*\* [2] 查询经过站点路线 \*\*" << endl;

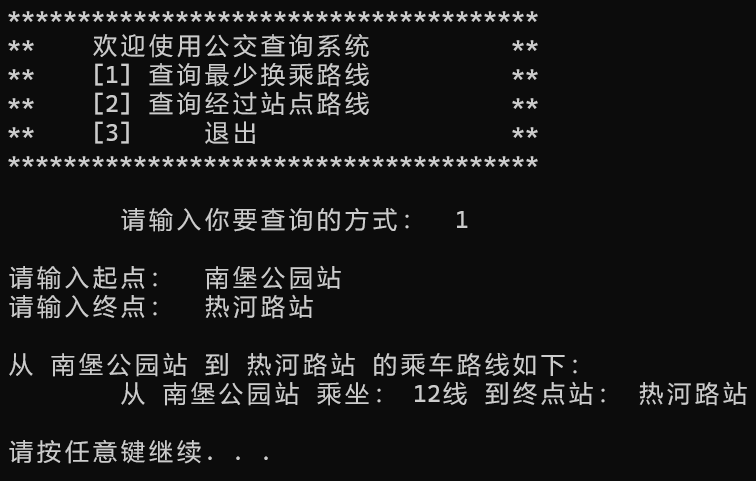
cout << "\*\* [3] 退出 \*\*" << endl;

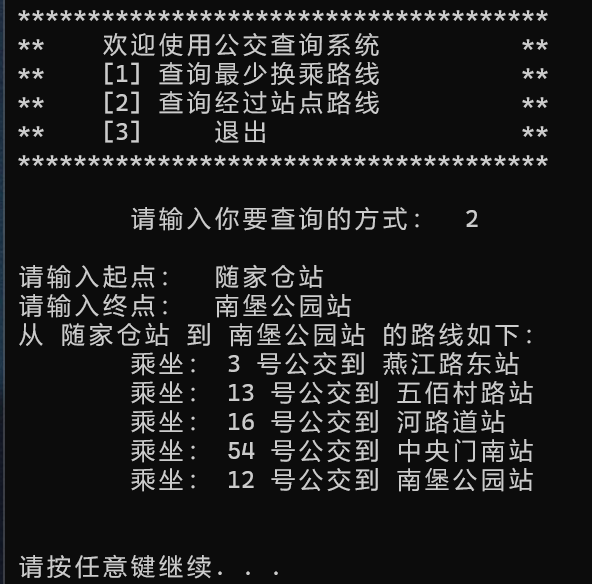
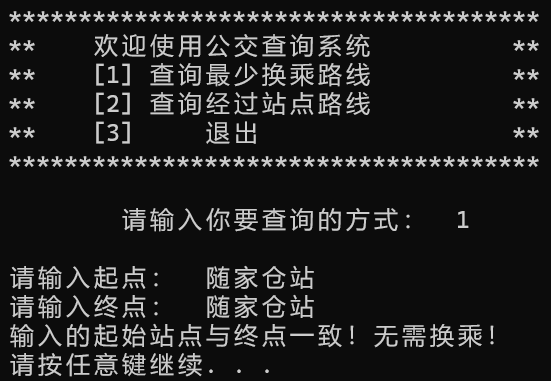
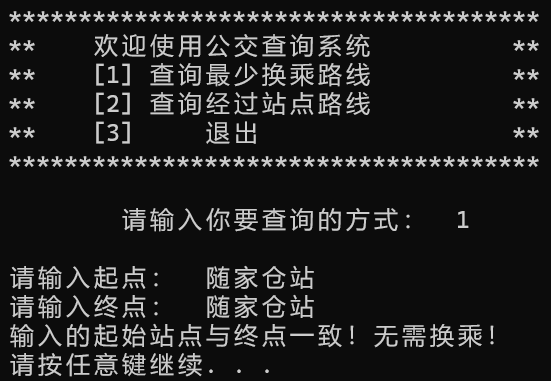
cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

}

### 1.4.4 测试数据和结果





### 1.4.5 时间复杂度：

迪杰斯特拉算法O(n^2)

BFS算法O（N）

### 1.4.6 反思：

BFS算法理论上来说，由于在求路径时我们已经知道起点和终点，实际上我可以考虑使用双向BFS进行优化，即从起点和终点都进行BFS，减少不必要的搜索。（可以在之后选做题中的词梯和迷宫问题中使用）

### 1.4.7 改进：

1. 增加控制最大转车次数，如果要换乘的次数超过10次会进行提示，建议改乘坐其他交通工具。
2. 如果起点和终点一致直接输出不必调用求路径的函数

### 1.4.8 该题代码行：

604行

## 1.5 霍夫曼编码

### 1.5.1 数据结构：

采用一维数组存储霍夫曼树节点，霍夫曼树的节点结构体除了包含左右子树即父结点位置下标还有是否被访问的flag及对应的字符在文章中的个数和编码串。

结构体Node如下：

typedef struct Node

{

char alpha='\0';

int num=0; //所在文章的该字符的个数

bool flag=true;//是否被访问过

string code="";

int Father=0,LChild=0,RChild=0;

}Node;

### 1.5.2 算法设计思想：

先对可能存在的字符的节点初始化，然后使用ReadFromFile函数读取源文件，然后在主函数进行霍夫曼建树编码过程，期间使用SearchMin函数。再将对应编码写入对应文件，使用Encode和EncodeOutput函数，最后再解码写入对应文件。使用RecodeOutput函数。

### 1.5.3 源程序：

#include <iostream>

#include <stdio.h>

#include <stdlib.h>

#include <fstream>

#include <string>

#include <algorithm>

using namespace std;

#define chNum 255

#define INF 10000

//设置为asiica码表字符个数

typedef struct Node

{

char alpha='\0';

int num=0; //所在文章的该字符的个数

bool flag=true;//是否被访问过

string code="";

int Father=0,LChild=0,RChild=0;

}Node;

Node nodes[2\*chNum];

typedef struct BiTNode

{

Node data;

BiTNode \*LChild,\*RChild;

}BiTNode,\*BiTree;

bool ReadFromFile(Node a[])

{

char ch;

int loc;

FILE \*fp;

if((fp=fopen("source.txt","r"))==NULL){

cout<<"文件打开失败！\n";

return false;

}

else

{

while(!feof(fp))

{

ch=fgetc(fp);

loc=int(ch);

a[loc].num++;

}

}

fclose(fp);

return true;

}

int SearchMin(Node a[],int n)

{

int min=INF,k;

for(int i=0;i<n;i++){

if(a[i].flag && (a[i].num<min)){

min=a[i].num;

k=i;

}

}

return k;

}

char h[20];

void Encode(Node a[])

{

string s;

for(int i=0;i<chNum+1;i++)

{

s="";

int k=0,f=-1,c=i;

while(f!=0)

{

f=a[c].Father;

if(a[f].LChild==c)

{

s+='0';

}

if(a[f].RChild==c)

{

s+='1';

}

c=f;

}

reverse(s.begin(),s.end());

a[i].code=s;

}

}

void EncodeOutput(Node a[])

{

char ch;

int x;

ifstream infile;

infile.open("source.txt");

ofstream outfile;

outfile.open("code.txt");

if(outfile&&infile)

{

while(!infile.eof()){

infile>>std::noskipws;

//使之可以读取到空格

infile>>ch;

x=int(ch);

outfile<<a[x].code;

}

}

else

{

cout<<"打开失败！\n";

}

outfile.close();

infile.close();

}

void RecodeOutput(Node a[])

{

char ch;

int x;

ifstream infile;

infile.open("code.txt");

ofstream outfile;

outfile.open("recode.txt");

if(outfile&&infile)

{

infile>>noskipws;

while(!infile.eof())

{

infile>>ch;

if((ch!='0')&&(ch!='1'))

{

outfile<<ch;

}

else if(ch==' ')

{

cout<<" ";

}

else

{

int k=2\*chNum;

while(k>chNum)

{

if(ch=='0')

{

k=a[k].LChild;

}

if(ch=='1')

{

k=a[k].RChild;

}

if(k>chNum)

{

infile>>ch;

}

}

outfile<<a[k].alpha;

}

}

}

else

{

cout<<"打开失败！\n";

}

outfile.close();

infile.close();

}

int main()

{

//初始化

for(int i=0;i<chNum+1;i++)

{

nodes[i].alpha= i;

}

if( !ReadFromFile(nodes) )return 0;

int n=chNum+1;

for(int i=0;i<chNum;i++)

{

int a1,a2;

a1=SearchMin(nodes,n);

nodes[a1].flag=false;

a2=SearchMin(nodes,n);

nodes[a2].flag=false;

nodes[n].num=nodes[a1].num+nodes[a2].num;

nodes[n].flag=true;

nodes[n].LChild=a1;

nodes[n].RChild=a2;

nodes[a1].Father=n;

nodes[a2].Father=n;

n++;

}

Encode(nodes);

EncodeOutput(nodes);

cout<<"文件编码成功！\n";

RecodeOutput(nodes);

cout<<"文件译码成功！\n";

return 0;

}

### 1.5.4 测试数据和结果

#### 1.5.4.1 source.txt

When you are old and grey and full of sleep

And nodding by the fire take down this book

And slowly read and think of the soft look

Your eyes had once and of their shadows deep;

How many loved your moments of glad grace

And loved your beauty with love false or true

But one man loved the pilgrim Soul in you

And loved the sorrows of your changing face;

And bending down beside the glowing bars

Murmur a little sadly how Love fled

And paced upon the mountains overhead

And hid his face amid a crowd of stars

William Butler Yeats

To see a World in a Grain of Sand And a Heaven in a Wild Flower

Hold Infinity in the palm of your hand And Eternity in an hou

William Blake

National debate in the United States on the role played by race in college admissions culminated with the Supreme Court hearing challenges against two universities

To create a diverse student body many colleges and universities in the US consider a student s race as a factor in their admissions process Such race-conscious policies known as affirmative action have been repeatedly upheld by the nation s top court in past decades

Educators fear a ripple effect if the conservative court decides that the affirmative action policy is illegal

The baseline for permissible affirmative action programs in US higher education was established in X X X X

Citing Harvard University as the model Justice Lewis Powell said that in evaluating applicants race could not be the determinative factor but the university could use race as one of the many factors just as it uses other traits such as a special talent for music science or athletics and even the fact that an applicant s parents attended the university

In X X X the Supreme Court upheld an admissions program at the University of Texas at Austin ruling that the university could continue to consider race as a factor

This time the challenges were brought against Harvard and the University of North Carolina or UNC with the court hearing the arguments in late October

The new Supreme Court is the most conservative for X years The six justices appointed by Republican presidents and the three appointed by Democrats appeared divided along ideological lines

The court is likely to overturn some or all of such case precedents based on sharp and skeptical questioning from the conservative justices

During court argument Justice Clarence Thomas asked lawyers for the universities to define diversity He said It seems to mean everything for everyone Justice Samuel A Alito Jr asked what underrepresented minority meant

The justices are not expected to finalize their opinions until late June or early July If they rule that affirmative action is unconstitutional the number of black and Latino students would be reduced in colleges and universities nationwide particularly at elite institutions

Affirmative action is a government policy designed to help minorities and disadvantaged groups find employment gain admission to universities and obtain housing

Race-conscious policies aim to address discrimination that denies underrepresented students access to higher education

Until the X X X s and X X X s Harvard and UNC refused to admit large numbers of black students and other students of color Both schools said affirmative action allows them to select a diverse student body to create an inclusive educational environment that benefits all students

However opponents of affirmative action targeted the universities arguing that their programs violate equal protection principles and discriminate against Asian American students

Students for Fair Admissions or SFFA a conservative group that brought both challenges to the Supreme Court sued Harvard and UNC in X X X

The group alleged that Harvard intentionally discriminated against Asian American applicants by holding them to a higher standard in undergraduate admissions and specifically limiting the number of Asian Americans it admits each year

While Harvard is a private university the plaintiff said the institution was violating the X X X X Civil Rights Act that prohibits schools receiving federal funds from discriminating based on race

In the UNC case the group said the school policy is subject to the same law as well as the X Xth Amendment s guarantee of equal protection which covers state universities It said the school discriminated against white and Asian applicants by giving preference to black Hispanic and Native American students

SSFA lawyer Patrick Strawbridge accused UNC of using race behind opaque procedures in awarding mammoth racial preferences to African Americans and Hispanics

A white out-of-state male who had only a X percent chance of admission would have a X X percent chance if UNC treated him as an African American and a X X percent chance if it treated him as a Hispanic he said

In contrast to Strawbridge s suggestion US District Judge Loretta C Biggs found that the university continues to face challenges admitting and enrolling underrepresented minorities particularly African American males Hispanics and Native Americans

In October last year she ruled in favor of UNC saying it had not shown illegal bias against white and Asian American students

The university has been struggling to build a diverse student population In a state that is X X percent black just X percent of the undergraduate student population is African American

Biggs wrote in her ruling Ensuring that our public institutions of higher learning are open and available to all segments of our citizenry (is) an institutional obligation

SSFA filed an appeal at an appeals court in Richmond Virginia and at the Supreme Court In January the Supreme Court decided to hear the challenge even though the appeals court has not yet ruled

The lawsuit brought against Harvard by SSFA centers on the treatment of Asian American students who have on average better standardized test scores and grades than any other ethnic group including whites

Harvard admissions consider a student s academic extracurricular athletic and personal ratings The latter category attempts to assess how an applicant impacts people around him or her and the contributions the student might make

SFFA accused Harvard of discriminating against Asian American students by using a subjective standard to gauge traits such as likability courage and kindness

In X X X the district court ruled in favor of Harvard finding that it did not discriminate against Asian Americans In November X X an appellate court affirmed the district court decision ruling that it did not clearly err in finding that Harvard did not intentionally discriminate against Asian Americans

Harvard denied the accusation saying that Asian American enrollments have consistently risen The university s lead lawyer Seth Waxman said during the Supreme Court argument that if the school abandoned consideration of race as a factor representation of African American and Hispanic students in admissions - not white students - would decline

The X X X Asian American Voter Survey found that X X percent of Asian American voters favor affirmative action programs designed to help black people other minorities and women gain better access to higher education

Among Chinese Americans support for affirmative action stands at X X percent the lowest within the Asian American community

A Chinese American lawyer in Silicon Valley California said he opposes affirmative action because it is outdated

A long time has passed since affirmative action was created We don t need it anymore said the father of two who requested anonymity He said he fears that his children now in middle school will be disadvantaged when applying for college

Chinese for Affirmative Action an organization based in San Francisco has been trying to persuade the Chinese community that the discrimination it faces is not the result of affirmative action

Instead of focusing on affirmative action the activists called for opponents to question other areas of the admissions process such as legacy admissions and athletic preferences

According to the National Bureau of Economic Research X X percent of white students admitted to Harvard fall under the categories of recruited athletes legacy students and children of faculty and staff members This percentage also includes the dean s interest list which consists of applicants whose parents or relatives have made donations to the university

#### 1.5.4.2 code.txt

011011010001110000100011111011100010001111111100010000011100100110011010111110010001101011101111101000001101110111110010001101011110011000111011000110011100101001101110101011000000000111101111101111100111010001101011110000010110101101010101000011111111011011111011101111011011100001111001101010010000011110111100100111101000111110100010011010110001111011011101010010111101101110010001010011110111111011111001110100011010111010101100001001101010110011011101110100000110011010111110010001101011110110111010101000100111101111001010011011110110111000011101010010100110101111101100001000101001111011101111011011011101001000111010011100011011100000101111011101100110101110010100010010000111110010001101011100101001101111011011100001010010011101010111011001101000100110101010111111010000000011110011011011010110111100110010001001101011111101101100100011011101110110000100011010001101011111011100010001110100111110110001011011000010001011010111100101001101110111110110011001101011101111101001100100100001111101111100111010001101011101100001000110100011010111110111000100011101001110110111000110000111101111011101110110101101010110111011101100001000110100011110011011000110001010001110010010011110110100001110001111101111011010001100111101111100101000000111110110110010001110110000100011010001101011110110111000011101111010100110001111101001010110110111011010010010001110110011110101000111110111000100011111111011111001110100011010111011000010001101000110101111011011100001110101001001000100001001101010101111001010011011111011100010001110100111100100111011001000011111101010000111111111001101100100100000110110110101101111100111010001101011101101110001000110101010100001111111111010001001101011000111011011100001011010110100001111011011100001110111110110000100110101101010000111111110110111110001000101111110111101101101111100011101001101100011101001111100111011001010101110110110000011101011100110100110011011101110111000100110101111011011010110010001101000111100110011000001101011011111001110100011010111011110110010010000110101110011101111000101000111101101110000111110110001000111100010111100101010000101111001000110100001000111000011001101011011111001110100011010111011101010110101110111010100101111100110110010010000111110011011010101101011111001111001001000010011010111010111001010011011101011011110001000101111110111101101101001010011000110010101100110110111011010001100111101101100000010011101101101110100011001011010111011111001111000010111010100000011111001110110110100001001000110011010111101010001111100111011011011110101001100101010001110010100110111011010011100100011010111100111010001101011111001110011001000011000011010001000111101010001111100111011011010010100110011010111100111110001100001001101010000100110111100110010001001100110101110011001111000100110101010001010101111011101111010100011110110111000011101111011000110011011011100101001101111101110001000111010011101110110010001101011110011101000110101110110110101010110000100100010101011110111011110101000111110010001110111000100011111011110110110100101001100011001010110011011011101101000110110011001001111010001101111110111101101000011001011101000101000110001100111110100000110111110010110001111010100011110110111000011101101100010001010101100011010111011010011011110010110000101111001010001111011011100001110100001001100000111011110011001100110111000011010111011011111011101110100110010010000111101010001111001000100110001100000011111000111110011010110110101001010101101000101000010111110010001110110011011010101000110010110001101011101101011010101101110111101101110000111011010010011101111001000001101100001111001111110010001110100101111101110000110001001010100001111111110010011101100011000110000010000111110000101111110001111111001010100001011011111101101101010010111001111000101000110100001000101101010111010000010111111011111101111100111100001011110010010000011001011000111110011111010101000110100001000101000111010110110011111010000100010111110110111001011010110111011111011011001000110111011110010001001100011000000111110000101111110010001101011100111100010100011010000100010110101011101000001011111010100011110110111000011101101100001101001111100100010100001011010110100000100111110011101011011001111101000010001011111010111101001100100100001111100010111111001111001101100100101011001001001111010100011110110111000010100100111110011010110110101001010101101000101000010111101111001000010100100000101010111111101101001001111001001110111010011001001000000110011001001000101000010110010101000100011101011110111100010011001010100101010000010111110011110110000010011010110001111100010111111001001101001101010010011011011001011101000110100011111001001010111010001010001110111011000011010001110110111000000100011101000000111100001100101100011010011001101110111001110111100111000001100110101110110111110111011110110111000011110001100101110100010100011101011111011001001111011110010001000111010010111111010100011101111011000101101111111010000100101100110100000101111110111111011110110110101011010001111001011001011001001000101111100110000110001001111100111010010100111100111100110000011100010011010011000010010101111110101001101111011011100001111001000101000010100001000011011100101110100011010001111001000100011101001011111110100001001010101101000001011111011011101100101111110110111000011111001001101001101010010011011011001011101000110100011111001001010111010001010001110111100010011001010100101101110111101001011111010011000110000001111111000110011111011111101111100111100011100001110110111110001010000110010101000000111100110001001001110111100000100110110101001010101101001101110110000011111001001101001101010010011011011001011101000110100011111001001010111010001010001110111100100001001111101001100110110010111110101000111011011000011010011110111010100111110111000001001110001101000111100101100101110100010100011101101011100010111100001011011110001101110110010100101011100001101011110101000111111001100011100110001110011000111001100011111011111101111100111111101010111010100001111111100110010110001000011011100010011010111011011000100010100011010000100010110101011110111011111000101111101101110000111110110001011010000011001110110110011001110101101110101001000011101101101011000011010110100101111011011011100001001101010000110001100111010111001010110101111011011101100101111110101000111000001101110001100001111100101110101000011111111110001111001111001100101010010110010001011010111101001100100100001111001000100011101100110101111000001010111110110111000111101101110000111110100001011000010011011010101000110010111010001101000111100110110010010101100100100111011011100111101111110110111000011100111100010100011010000100010110101011110111011110010001000111011001101011100111010100011101001100100100001111100010111100101000000111001010011011110110111000011111011011001000110111011110011011001001010110010010001011110110110010001110101101111111000101111101010111110011101010000101111001010110111000001001111011010011001010101101011110101001111001001110111110001011111100111010101111000010010101011000110011110111100011000001000101111110011000100100111110110001110101101010010111010110010101000010001001000011100100100111110010110111001100000101110101001001011111100100011010111000001101000100011110110111000011110011011001001010111111011011101100101111111001000111110001111001111001100101010010110010001011111010111101111011000100000100010110101111110010111011000100011010000110101111011011100001110011110001010001101000010001011010101111011101111101111110111100110011110001111110011000111111001100011100110001111011011100001110110100100111011110010000011011000011110011111100100011101001011111001110111100111000001100110101111100100011111001101011011010100101010110100010100001011110111100100001001111101001100110110111110010111111011011100001110110110001000101000110100001000101101010111101110111001010011011110011110000010011111011110001011111100101111110011100011101011011101010001110100001110110010101000011111111101101110110010111111011011100001110011110001010001101000010001011010101111011101111001000100011101100110101111001000101000101110101000001110001111011001011110010001010000101101011010000010011101001100100100001111100010111111001111001101100100101011001001001111101111110111110011110001110101001011111011101011011000011110110111000011110010011101100011000110000010000111110000101111011010100001000001110110111010000100011101111101110101111111000111111100101010000101101111100110010110001000011011100010011010111110010001101011110110111000011101101100010001010001101000010001011010101111011101110010100110111011010000001001001011011101111001111111100010000100110010101000110011100100100111011011000011010000100111111111011010110101011011101111011011100001111001000100011101001011111011100001100010010101000011111111101101110000111110001000111110011111011000010001011010111110101000111011001100101100011101101101101110010101100100110111000010011111011111101111100111100011100001111000000011010111101101001001110111100100000110110000111100111111001000111010010111111010010111110110111000011111011000100101101111110010001010000101000010000110111001011101000110100011110011000100100111111001100011111111011100001100010001011111111001111000111000011101011010100111110111110110110010001110101101110101001000001011111100011110011110001010101000101100011010111011011111011101111001111101000001111000111011011101100101010010110010001110111100100000010110101101000010001011010111111001000110101111011011100001111011011100100000000111110001111001111000101010100010110001101011101101111101110111011010001000001101100010100100100110010110101111110001111001111000011000100000110101111101010100011011010110100001101011111000110000101000011111111101011010000001001100001001111110101001011000110011101100101010000000101111110111111011111001111000111000011110010001000111010010111111010010111101100101010011110100001100110111011110110010111001000110100001001011001110100100011101010010110110000111001001001111100011000110011100101001101110101001111001001110111100101100010100011101111001000001001000011010000100010110101111011011111000101000110101110010100011101010111011000100011110111110010001101011101011001111010000111101011101010010110001100111001100110100111000010110111010001010001010100001111111110011001000010110110111101101110000111100100010100001010000100001101110010111010001101000111011011001000111010110111010100100000101111110111111011110110100010000111010010101000011111111100100010001110100101111111000100011111001111101100001000101111101101100110011101011011101010010000111100111111011001100010000010001001000011110011110001110001011011011000101111110001011001111010001101011101100110001101011101110000010001011111001100010010011110110111000011100111100010100011010000100010110101011101000001011111011001011111010000100110101010000001111101010100011010000100010110101011110111011111111100110010000111010111001010110101110011001111011111010100000011011001011111011001011111011000011001000111000001101000010011011101011011101010100001111111110011000100100111000001101000010011011100010100000011111101101100110011101011011101010010000111011010011100110110001110000110011110011101111001110011001010101100101110110110011010011111111000101100111101000110101110110101011101100101111100111100011010000010001000000111100100000010100010001011000110101111101101010100000100100101010111101110111110110000110010001011111110111111011111001111000111000011101101100100011101011011101010010000010111111000100000111100000101011111000100111110110111100001001010110001101011110110010111100110101010001100011001010011011011000001111011011100001010010011100100111101010100010100010100001011110011110001011101001100111011001100101100011101101100110011110000001110010010011100011000100011001101110111011011001100111011001101110111111001100111100110111101101110000110111011101000011101100000111101101110110010111111100100110100110101001001101101100101110100011010001111100100101011101000101000111101001011110011110001001000101000010110111010101100111101110100010100011000110011110110111000011110000011111011001101110000100111001010011011101101110110011001001010011110111111001000110101110110110101111001011101010000010111010110110011111010000100010110101111011010100100011101100110101110110111000111010000011010001111001000011010111101010001111001000100110001100000011111000010111111001000110101110011110001010001101000010001011010101110100000101111100011001011101000101000011010110101101000011101111011000100101110101001000111011001100010001100110111011111001011111000011001010101100011110101000010110111010101100111101110100010100001011111101111100111010011010011010100100110110110010111010001101000111110010010101110100010100011110100101111110011101111100100011010000100100011011000010001011111011110001001100101010010110111011111010000010110100111111000000110101111011001011101110000011000111101111101101010100000100100101010111010000010111111001000110101111101010100101110011010001101110010001011110001111100011010111011111010000100011101111001011111001101010100011010111000110110011110011000010110111011011000010001011111011111110010101000111110011010110110101001010101101000101000111101100101110011110001010001101000010001011010101110100000101111110010001101011100100110111101111001010100011101110001000111010110101000011111111110111111011111001111101011001001000000110011001001000101000010110010101000100011101011110111100010011001010100101010000010111111001010110110111101100101111100110101101001000000101010111111010101001011001001001010110110101010001100101110100010100011110110111011001011111110100001000101000001011110011110001101000001000100000011110010000001010001000101100011010111010110110011111010000100010110101111110010010100100000101010111110110010111011101010011111011100000100111000110100011110010110010111010001010001111101111110111101101100010001011101001100111101101110000111111001100011100110001110011000111010111111001000110101111110011000111001100011100110001110101111001100101100010000110111000100110101111100100011010111011011000011010000100111111111010000010011000111010100011010111101100101111100110101101101010101111101100110001000111110001111000001111101100110111000010001011110010100110111011011101100110010010100111101111010110110011111010000100010110101111110010001101011100101011011100000100111010110110011111010000100010110101111001010011011110010001001100001001001111110110100011001010110111011101011001001110001000100110001011110101110010101101011111001001101001101010010011011011001011101000110100011111001001010111010001010001111100011000110000100110101010111110110111000011011011110110010111010100001100000100101011111110011111010101000110100001000101000111010110110011111010000100010111110110111001011010110111011110110010111100100100000110010110001111100100011110101000100100110000111010110100011010001110001101000111100101100101110100010100011000110011100010000011011010010000101000110110000100010111111011011101100101111101101110001000000100110101010110101111110001100011001110101101100111110100001000101101011111101111110111100110010001001101010000011010000100111001001111001111000101000000100010110101111001010011011111001001101001101010010011011011001011101000110100011111001001010111010001010001111011110001000111110001011000110101111011011100001110011110001010001101000010001011010101110100000101111110001000111110011110101000011111111101101110110010111111011011100001010010011101111001000010011111010011001101100101111001101101000100110011001011000111000001100110100111110001100111011110010000101011000100101011101000101000111011110010010101000100101010011110011000000101111110010001101011111010101001011001001001010110110101010001100101100011111000111111100101010000101101111110011100101101011001000111100111011011000001001010100101100100011101011011001111101000010001011010111111011111101111011010011011001111101000010001011010111110011000100100111100111110011001010010011110011101101011011010100101010110100010100001011110010010011101101001100111110010011111001001110111110011110010001010000101000010000110111001011101000110100011101111101000010001110111101111011011101100101111101101110100001000111011111011101011111011011100101011011101111001001110110001100011000001000011111000010111110110010111101101110000111011010010011101111001000001101100001111001111110010001110100101111101010011100011010111001100101100010000110111000100110101111100100011010111011011000011010000100111111111101010001111110011000111111001100011100110001111101111110111110011110001110000111011111010000100011101111011111000110001100000011111000110101111011011101100101111100110010110001000011011100010011010111101010001011000100010111010001010001100011000110011011101111101010100101100100100101011011010101000110010110001101011111000111111100101010000101101111110011100101101011001000111100111011011000001001010100101100100011111000111100111100110010101001011001000101101011110110111110111011101110001001100110101010100001111111110110111000011011011110110010111110011101110101001111101110000010011101011011110010001101011000100110101111010100011100111100011010000010001111101001100110100011111001011000111110011010110110101001010101101000101000010111111001000110101110101011110000100101010100110101010010110001100011001101110111011001010110110101010111010100001111111110110111000011110000011111011001101110000100111001010011011110011100101101011001000111100111011011000001001010100101100100001011111010101111111001101011011010101011010111100011001001001110111110111000011000100111110111111011110110110100011101010011000001110011001011000100001101110001001101011110100101111110011101111001001010001101110010110001110011110001010001101000010001011010101111011101111011011100001110111100110011001010100010111010100110100110111010111001010110101111011011100001111010100001011011101010110011110111010001010001110110101110001011110011011010001001100110010111010100001111111110110111000011111100110001110011000111001100011100110001111001111111010001101101001100111100111110101010011111011101011010111110011101001010111111011011101100101111101111001000010011101010011011110101011010111101011001001110001000100110001011110100000100100001010001101101010000111111111001100001101000001001100011001111001100011110001101001011111001100100001011011011111010101001011001001001010110110101010001100101110101000011111111011011111000101000110101110010100011101001100100100001111101111110111100110011110001111011011100001110110110000110100001001111111111001011000101000111101101110000111011111010000100011101111011101011100101011010111101101110000111010110010011100010001001100111011110001001100101010010110111011110100101111010100111011011101101100100001001010111111011001011110110111000011101011100110110000111011001100011010111111000101111011010100001100011001111100010111110110111000011111100110001110011000101101110111100111011011000010001101011011000010001011111010111101111100111110001001100100010110000001110010100110111000001100110100111110001100111011110010000101011000100101011101000101000111011010101110101010010011101111001000100011010000100010111101011011110010110001110011110001010001101000010001011010101110100000101111111001100111101111101011100101011010111101101110000111010110010011100010001001100111110101010010110010010010101101101010100011001011000110101111100011111110010101000010110111110110101011101010101100011111001000110101111001110010110101100100011111000111100111100110010101001011001000101101011110110111110111011101111110100011011010100001111111101111001000001001100000100000100010010000111101100101110110111011001100100101001111011110011001010100101011110110010001010100101111100100011010111011010000110010111010001101000111100111011011000001001010100101100100011101011011001111101000010001011010111111011111101111011010010110100110011111001001110111011001100011010111011100000100111011011011100110010110100101010010100111101111011010011011010011000110101011011101001010110100111110001111100100101001000111010100011010111011011000011010000100111111111001010011011100111010110101000011111111010011001001000011101101110000111010101000110101110010011110110000110011010011100011101111001000010100100001101000111010000001011111010100011111000110101110001001101010101000011111111110110110011011011011000101011011101110100110010010101011000110011101111001000001001100000100000100010010000010111110110010111100111010011001001010100101100100011110011101101100000100101010010110010000101111110010001101011100110010101001010111101100100010101001001011111101111110111111110011101110110101011101010101100011100100011110110011001100001010011000110011000101101111001011000111110110110001100000111011010101110001011101110110011010111001010000110011011101111100111111001100011111101111000001001001000010001011111100100111011001000100100001110010100110111110011010110110101001010101101000101000111011010100100011101100110101110111011000011010001111100111111001100011100110001110111100000100100100001000101111110010011101100100010010000111101010011011101101100001101000010011111111110110100000110010110001101011101110101011011011111000101111110010001111001110100110010010101001011001000111100111011011000001001010100101100100011111001000110101111100111111001100011100110001110111100000100100100001000101111110010011101100100010010000111101010011011110101011111101101000001100101100011010111011101010110110111110001011111100111001100101010010101111011001000101010010111011100001110101110010101101011111011111101111001100111100011110010001010001011010011000101101111110110010111011010011011010011000110101011011101001010110100111110001110101111010100111011111011111000010110111010001010001110110110000110100111101101000100101001011011010010101001010111110110110011001111101001111100011101101101011001001000001011101111001111001111111111110110100011101001111101111101011111001100010001111000110101111011011101100101111110110111000011100111100010100011010000100010110101011110111011110010001010001011101010000011100001011111011001011110011011001001000011110010011101100011000110000010000111110000101111110011010110110101010111011101010000111111111100100011010111000100001000010011000110010101000011111111001111000110100000100010000001111001000000101000100010110001101011111011010101000001001001010101110100000101111011110110001001011101010010001110110011000100011001101110111100111010011001001010100101100100011110011101101100000100101010010110010001111101101100011000000101111001100101010010101111011001000101010010010111111001000110101110110100001100101110100011010001111001110110110000010010101001011001000010111111111011111101111001100111100011101101101101110010101100100110111000010011101100110001011011111110111000011000100111010101110000111010000111011000001101011110101000111100110110000110100100100111001010011011101101100001101000010011111111101011100110111010101000011111111101010111110111011001101011110000010101111101010111000100110101100011110100110001100000011111110001100111011011110101100010111111000111111100101010000101101111101101010111010101011000111110010001101011110011100101101011001000111100111011011000001001010100101100100011101011011001111101000010001011010111111011111101111100111100011100001110011110001010001101000010001011010101111011101110111011000101111011011100000010001110101101101000011101111101111101100101010000111111111011001011101101110011110100110011010111110011111010101000110100001000101000111010110110011111010000100010111110111100010011110001110110011001011101000101000111111001100111100011111001110101101111001011000111101101110110010111111010010111111100110001110011000111011110000010010010000100010111110110111011001100100101001111011110110110010001110101101111111100110001110111100000100100100001000101111100101001101111011011100001110011110001101000001000111110100110011010001111100101100011101011011001111101000010001011111011110001001111000111011001100101110100010100011110100101111100111010011001001010100101100100011110011101101100000100101010010110010001111101111110111101101000111010011111011111010111101101010100001010110001111010100011101110000010011101000011101100101010000111111110110110101010000101001110100101010000111111111011011101100101111100100011101001110111100011101101110110010101001011110101000010110111010101100111101110100010100001011110010100110111011101010011111011100000100111011000001100010010001010100001111111111000100000111001001111000010001111100100011010111110000110111001010011001100011011101100000111101100101111100011000110011101010000111111101100001000101101011110010100110111001000111010011110010101010111010011011011000001000010011011101110110110111111110100101011011011110011111001000111101010000101101110101011001111011101000101000110001100111001001101110110010100111111100101110100010100011111111011111101111011010010110100110011111001001110111100110101001100000110101111100100011111000111100111100001100011001111100101111111001000111110001111001111000011000110001011111001000100011101001011111101010001111001111101010101001001110110110001010001101011101101000101101001000111111010100010101100111110010001101011111001011111101101110000111011010010011101111001000001101100001111001111110010001110100101111111100110011110001110110110011110010000011111000100110111011110110111000011101101001001110111100100000110110000111100111111001000111010010111111101000010010101011010000110101111011001011101110000110001001111011011100001111001001110110001100011000001000011111000111000001101000100011110110111000100011101111101110111101101110000111110001111001111000011000110001011111001000100011101001011111011101100010111110000010101111111011100001011111010000111011000001101011111011111101111100111100011100001110110011000110101010100111101010111110110111010000100011101111101110101111111000111111100101010000101101111100110010110001000011011100010011010111011011111011101110110100101101001100111110010011101111001000010001011000010001011110010100011110110111000011110110100000110010111101100001000101111100101001101111001110010110101100100011110011101101100000100101010010110010001110101101100111110100001000101101011110110101011100010111011101100001101000111001010001111100001101000010011000111110001110110111000101110110000100111010110111100100011010110001001101010100110110110000011010111101100001011011111010110010001001000000101111110010001101011101111101001100110100000101111101101110110010001111100100011011101110010101101110000010011100010110111010001010100101110111110100001000111011110111101010001001001100001111101010101000011111111011010101110101010110000101111110111111011110011001011000100001101110001001101011111001101011011010100101010110100010100001011111001000101000010110101101000001001111100111010110110011111010000100010111110101111110010010110011010000110110101010010111000100111110111011010011001001000111010001001010100100011101100110001001111100101101110011000001011101010010111110010001101011101111000001000101001010001100011001110100110010111010100001111101011111111001111000111000011101100110010111011000010011110010110010110000111110010010011011101111100101110110001101100111101011010111110110010111110001010101000010101011110111000100110101111110010001111100011110011110011001010100101100100010111111010110110011110110010010101101011110111100000010011110011000001111100010000100011110001101011101110101011011011100100100111011100000100111110010001101011110110111000011110010001010001011010010100110111001111011101000101000010111110110111000011101011011001111101000010001011111110110101001111101110101111111011011001001111010001111101111110111101101001100111110010011111001001110111110010010100100011101010001101011100110010110001000011011100010011010111001010011011111010101001011001001001010110110101010001100101110101000011111111110001111111001010100001011011111100111001011010110010001111001110110110000010010101001011001000111010110110011111010000100010110101111011011111011101110011101011010100001111111111001110101001110110111011011001000010010101110100011010001110101101111001000110101100010011010111101100101110111111100001110111110001111011010011001010101101011110101001111001001110111110001011110110010101001111011100011011110100110010101011110111011110010001000111010011000111110001111100100011010111100111101101010001101010000000101010111111011111101111001100111100011111100110001111110011000111001100011110110111000011111010101001011011010010101001010111111001000100011101001011111010000111011000001101011110101000111100110110000110100100100111001010011011100110010110001000011011100010011010111100110101010001101010101000011111111101101110110010111111010101111111010101011010111100000101011111110101010010110010010010101101101010100011001011000111110001111111001010100001011011111100111001011010110010001111001110110110000010010101001011001000010111111100110011110001110110100000010001101000110110011011100001001111110011000111111001100011111111001000111110001111001111000001100011001100101100011110010001000111010010111111100100110100110101001001101100001101011110110111000011111010101001011011010010101001010111111001000100011101001011111110100001001010100101101000101000111010000111011001010100001111111110110111011001011111101010111111101010101101011110000010101111110010011000001100010001100110111011100001000100111101010001111001101010100011010101010000111111111011011101100101111100110010110001000011011100010011010111110101010110101111000001010111111010100010110001000101110100010100011000110001100110111011111010101001011001001001010110110101010001100101100011111000111111100101010000101101111110011100101101011001000111100111011011000001001010100101100100001011111111101111110111100110010110001000011011100010011010111110100001000101000011010111101101110000111110010010100100011101011100101110100010100011101011100110111010101000011111111101101110110010111111001110010110101100100011110011101101100000100101010010110010001110001000010000100110001100110110000100010110101111011101100001101000111100100010100001011010010110110001000101101100110111011101001010010100010001111111001111000111000011100111100010100011010000100010110101011110111011101011110110000011001101011101100110001101011101110000010011101101001000101101110111011011010011001001111101111011011001000111010111001010110101111101000111010010101000011111111101101110000111011010010011101111001000001101100001111001111110010001110100101111111000100011111001111101100001000101111110110111011001011111101010011011110110111000011101011001001110001000100110011111000110111110010001101000101000000110101111001000101000010110101101000001001100101110100010100011100101001101110100110010010000111110001011111100111100110110010010101100100100111010000001111001000000101000100010111100101110100010100011100101001101111001110100110010010101001011001000111100111011011000001001010100101100100011111001000110101110011001010100101011110110010001010100101110101101100111110100001000101101011111010100011111001101011011010100101010110100010100001011110011001100111100000101011111011010101110101010110001110101101100111110100001000101101011110011001100111011010100100011101100110101111101000010010011001010100000011111011111101111100111100011100001111110011000111111001100011100110001111001110010110101100100011110011101101100000100101010010110010001110110100010100101011000010011101101001001110100001101000110111011110011000100011110001101011110110111011001011111111001100011100110001110111100000100100100001000101111100101001101111001110010110101100100011110011101101100000100101010010110010001110011010010101100001000101111100110110000110100100100111110010011010011010100100110110110010111010001101000111110010010101110100010100011101111001000010011111010011001101100101111110100000101101001111110000001101011110110010111011100000110001111011101101110110011001001010011110111101111000000100111100110000011100101011011100000100111110110101010000010010010101011101000001011111100100011010111011010100101101100001000111011111110010101000111011011100010111011000010011111001001010010000010101011111011001011101110101001111101110000010011100011010001111001011001011101000101000111110111111011111001110110110001010000111111111001111110111010101000000010100011110011101101100000100101010010110010000101111010100111011110011110001001001011111100110001001001111100100110100110101001001101101100101110100011010001111100100101011101000101000111010110111100100011010010111111001011111111001100011100110001110111100000100100100001000101111110110111000011101100001001101010000101101111101101011010101101110101010001111011011100001111001110010110101100100011110011101101100000100101010010110010001111001000101101101101100011110001010101111011101111101111110111110011101111001111110111010101000000010100011110011101101100000100101010010110010001110110011000110101110111000001001111010100011101101001101001100101010010001010001110110100010111000110001100000110111011110011111111000110010101001100010010010001010110011101011100101011010111011100001110010011110011110001001010000101111110010011010011010100100110110110010111010001101000111110010010101110100010100011101101110001001011000011101010001111010101111110100101111001000111101111010110010110001101011111011111101111111100111011101100001010000111111111011101011011000011101110110001011110111101100010101010001101011101011010100010010000111110010011010011010100100110110110010111010001101000111110010010101110100010100011101101011100010111110010010000011001011000110101111110110110100000111110100010100011110111111000000000110101111010101111111001000110111011011000100100000111010111001010110101111011011100001111001101100101101110000010011100101001101111011011010100101110110101011100010111010000000110011010011100001011011000110101111100100000101000110111011011010101011110111011111100110010000111010111001010110101110111000011110011000011000100010111110110111011001011111011101010010111110010011101010011001101001000001000111100000100110101111101010001111101101010110101101001100000111010110010011100010001001100111011010110100110001100111011011100011111010101001011100110100011011100100010111100011111000110101110110101011100001000111110001111001111001100110111010101000011111111100110001001001111001000100110001100000011111000111110111111011111001111110111010101000000010100011110011000100100111100111010011010011010100100110110110010111010001101000111100111010010101110100010100011111001000111001001000111111100100010100110110110011001011101000101000111011011111000101000110101111010100011101101001110010001111001111100010011001000100101010010110010001011101110110001011110110111000000100011110110100110111010101000011111111101100101110111100000100010100111110011010000111101101110000111100111111011101010100000001010001111001000101101101101100011110001010101111011101111011011101100101111110110111000011111010101001011001001001010110110101010001100101110100010100011110101011111100110110010010000010111110100101111100000101011111101101110000111010000001010011101100101111100101001101111100100110100110101001001101101100101110100011010001111100100101011101000101000111110111111011110011001111000010110110001100110101110010100110111100110001010010001110101101010000111111110010100011111001001101001101010010011011011001011101000110100011111001001010111010001010001111011011100001111100100101011101000110110100101101101011111001011000110001100000110101111001100010010011100100111100111100010100000010001011010111110110010111001100110100111000010110111010001010001110010101101110000010011111000100000110001011110010100110111101101110000111110011010110110101001010101101000101000010111101111001000010100100000101010111101010011110010011101111100010111101100000011111110010010110111011111001101011011010100101010110100010100001011111100100011010111110010110111001100000101110101001011101111001000001001100000100000100010010000010111111011111101111100111010010100100010010011010101010000111111111011001011110110111000011101101000011001011101000101000110001100111011010001100111010000011000011111100101001101110110110101010010001010000010110110101010010111100111110100000101000110001001001001110111111001100011100110001110111100000100100100001000101111100101001101110110101011101010101100011101011011001111101000010001011010111111001101011011010101011101100011010111101100101110011001011000100001101110001001101011110011011000110001100111001111000110100000100111101101110000111100101100101100001111100100100101000001011110010100110111010000010010010000111101010110001101011111001011011100110000010110000101111011000000111111100100101101110111010110110011111010000100010110101111110010001101011110010011101010011001101001000001000111001010011011110011011001001000111011001011110111011111001000110101110101101111001001101001101111101100001101100110111000010001011111111001111000111010100101111011110000010010010000100010111100011111000111110001100010100101111010100010010011000011111010000010111110110111000011111010000110010001110101111101010001011000010000001011011111011001010010110111110110101011101010100100111011110010001010000101101001011011010111100101001101111100011110011110011001010100101100100010110101111011010101110001001010001110111101100010000010001011010111100100100111010000001100110010111010001101000010111101110110000110100011111011011001101000011111010001010001100101110100010100001011111011001011110110111000011100111100010100011010000100010110101011110111011111011111101111

#### 1.5.4.3 recode.txt

When you are old and grey and full of sleep

And nodding by the fire take down this book

And slowly read and think of the soft look

Your eyes had once and of their shadows deep;

How many loved your moments of glad grace

And loved your beauty with love false or true

But one man loved the pilgrim Soul in you

And loved the sorrows of your changing face;

And bending down beside the glowing bars

Murmur a little sadly how Love fled

And paced upon the mountains overhead

And hid his face amid a crowd of stars

William Butler Yeats

To see a World in a Grain of Sand And a Heaven in a Wild Flower

Hold Infinity in the palm of your hand And Eternity in an hou

William Blake

National debate in the United States on the role played by race in college admissions culminated with the Supreme Court hearing challenges against two universities

To create a diverse student body many colleges and universities in the US consider a student s race as a factor in their admissions process Such race-conscious policies known as affirmative action have been repeatedly upheld by the nation s top court in past decades

Educators fear a ripple effect if the conservative court decides that the affirmative action policy is illegal

The baseline for permissible affirmative action programs in US higher education was established in X X X X

Citing Harvard University as the model Justice Lewis Powell said that in evaluating applicants race could not be the determinative factor but the university could use race as one of the many factors just as it uses other traits such as a special talent for music science or athletics and even the fact that an applicant s parents attended the university

In X X X the Supreme Court upheld an admissions program at the University of Texas at Austin ruling that the university could continue to consider race as a factor

This time the challenges were brought against Harvard and the University of North Carolina or UNC with the court hearing the arguments in late October

The new Supreme Court is the most conservative for X years The six justices appointed by Republican presidents and the three appointed by Democrats appeared divided along ideological lines

The court is likely to overturn some or all of such case precedents based on sharp and skeptical questioning from the conservative justices

During court argument Justice Clarence Thomas asked lawyers for the universities to define diversity He said It seems to mean everything for everyone Justice Samuel A Alito Jr asked what underrepresented minority meant

The justices are not expected to finalize their opinions until late June or early July If they rule that affirmative action is unconstitutional the number of black and Latino students would be reduced in colleges and universities nationwide particularly at elite institutions

Affirmative action is a government policy designed to help minorities and disadvantaged groups find employment gain admission to universities and obtain housing

Race-conscious policies aim to address discrimination that denies underrepresented students access to higher education

Until the X X X s and X X X s Harvard and UNC refused to admit large numbers of black students and other students of color Both schools said affirmative action allows them to select a diverse student body to create an inclusive educational environment that benefits all students

However opponents of affirmative action targeted the universities arguing that their programs violate equal protection principles and discriminate against Asian American students

Students for Fair Admissions or SFFA a conservative group that brought both challenges to the Supreme Court sued Harvard and UNC in X X X

The group alleged that Harvard intentionally discriminated against Asian American applicants by holding them to a higher standard in undergraduate admissions and specifically limiting the number of Asian Americans it admits each year

While Harvard is a private university the plaintiff said the institution was violating the X X X X Civil Rights Act that prohibits schools receiving federal funds from discriminating based on race

In the UNC case the group said the school policy is subject to the same law as well as the X Xth Amendment s guarantee of equal protection which covers state universities It said the school discriminated against white and Asian applicants by giving preference to black Hispanic and Native American students

SSFA lawyer Patrick Strawbridge accused UNC of using race behind opaque procedures in awarding mammoth racial preferences to African Americans and Hispanics

A white out-of-state male who had only a X percent chance of admission would have a X X percent chance if UNC treated him as an African American and a X X percent chance if it treated him as a Hispanic he said

In contrast to Strawbridge s suggestion US District Judge Loretta C Biggs found that the university continues to face challenges admitting and enrolling underrepresented minorities particularly African American males Hispanics and Native Americans

In October last year she ruled in favor of UNC saying it had not shown illegal bias against white and Asian American students

The university has been struggling to build a diverse student population In a state that is X X percent black just X percent of the undergraduate student population is African American

Biggs wrote in her ruling Ensuring that our public institutions of higher learning are open and available to all segments of our citizenry (is) an institutional obligation

SSFA filed an appeal at an appeals court in Richmond Virginia and at the Supreme Court In January the Supreme Court decided to hear the challenge even though the appeals court has not yet ruled

The lawsuit brought against Harvard by SSFA centers on the treatment of Asian American students who have on average better standardized test scores and grades than any other ethnic group including whites

Harvard admissions consider a student s academic extracurricular athletic and personal ratings The latter category attempts to assess how an applicant impacts people around him or her and the contributions the student might make

SFFA accused Harvard of discriminating against Asian American students by using a subjective standard to gauge traits such as likability courage and kindness

In X X X the district court ruled in favor of Harvard finding that it did not discriminate against Asian Americans In November X X an appellate court affirmed the district court decision ruling that it did not clearly err in finding that Harvard did not intentionally discriminate against Asian Americans

Harvard denied the accusation saying that Asian American enrollments have consistently risen The university s lead lawyer Seth Waxman said during the Supreme Court argument that if the school abandoned consideration of race as a factor representation of African American and Hispanic students in admissions - not white students - would decline

The X X X Asian American Voter Survey found that X X percent of Asian American voters favor affirmative action programs designed to help black people other minorities and women gain better access to higher education

Among Chinese Americans support for affirmative action stands at X X percent the lowest within the Asian American community

A Chinese American lawyer in Silicon Valley California said he opposes affirmative action because it is outdated

A long time has passed since affirmative action was created We don t need it anymore said the father of two who requested anonymity He said he fears that his children now in middle school will be disadvantaged when applying for college

Chinese for Affirmative Action an organization based in San Francisco has been trying to persuade the Chinese community that the discrimination it faces is not the result of affirmative action

Instead of focusing on affirmative action the activists called for opponents to question other areas of the admissions process such as legacy admissions and athletic preferences

According to the National Bureau of Economic Research X X percent of white students admitted to Harvard fall under the categories of recruited athletes legacy students and children of faculty and staff members This percentage also includes the dean s interest list which consists of applicants whose parents or relatives have made donations to the university

### 1.5.5 时间复杂度：

霍夫曼算法时间复杂度O(n^3)

### 1.5.6 反思：

可以将建树过程编为一个函数然后在主函数调用。减少程序耦合度。

可以将获取权值最小的两个节点使用其他排序方法减少时间复杂度

### 1.5.7 该题代码行：

195行

## 1.6 排序算法比较

### 1.6.1 数据结构：

快速排序中使用的栈

### 1.6.2 算法设计思想：

由于需要使用八种排序对同一个样本进行比较，故用一个类将输出函数，八种排序以及相关函数进行封装。

生成样本函数分为三种，一种使用rand函数随机生成样本，一种生成升序样本，一种生成降序样本

在快速排序函数中，由于样本数较大，使用递归方法使会使系统给予的堆栈溢出，故使用非递归方法，相比于递归方法，使用栈来记录每次快速排序的起始点，从而使用循环模拟快速排序的递归方法，在循环中需要判断保证基准的左右分区存在

以下为八种排序的思想：

1. 冒泡排序（Bubble Sort）：是一种简单直观的排序算法。它重复地走访过要排序的数列，一次比较两个元素，如果他们的顺序错误就把他们交换过来。走访数列的工作是重复地进行直到没有再需要交换，也就是说该数列已经排序完成。这个算法的名字由来是因为越小的元素会经由交换慢慢"浮"到数列的顶端。
2. 选择排序（Selection Sort）: 首先在未排序序列中找到最小（大）元素，存放到排序序列的起始位置。第二步再从剩余未排序元素中继续寻找最小（大）元素，然后放到已排序序列的末尾。重复第二步，直到所有元素均排序完毕。
3. 插入排序：插入排序是一种最简单直观的排序算法，它的工作原理是通过构建有序序列，对于未排序数据，在已排序序列中从后向前扫描，找到相应位置并插入
4. 希尔排序: 先将整个待排序的记录序列分割成为若干子序列分别进行直接插入排序，待整个序列中的记录"基本有序"时，再对全体记录进行依次直接插入排序。
5. 归并算法：将已有序的子序列合并，得到完全有序的序列；即先使每个子序列有序，再使子序列段间有序。将两个有序表合并成一个有序表。
6. 快速排序：快速排序是在冒泡排序基础上的递归分治法。
7. 堆排序：堆排序是一种利用堆的概念来排序的选择排序。堆近似完全二叉树，它满足子结点的键值或索引总是小于（或者大于）它的父节点。
8. 基数排序：将整数按位数切割成不同的数字，然后按每个位数分别比较

### 1.6.3源程序：

#### 1.6.3.1 Sort.h

#ifndef SORT\_H

#define SORT\_H

#include <iostream>

#include <time.h>

using namespace std;

class Sort{

public:

int startTime;

int endTime;

int num;

int \*a;

int \*origin;

int len;

public:

Sort (int \*array,int numArr): origin(array),num(numArr),len(numArr){

}

void ShowSort(int info = -1); //info表示为第几个样本

void copyArr(); //每种排序前的初始化

void StraightInsertionSort();

void ShellSort();

void SimpleSelectionSort();

void HeapSort();

void HeapMax(int start,int end);//大根堆，维护堆的函数

void BubbleSort();

int NumberOfThree(int low,int high);//快排辅助函数

void QuickSort(int left,int right);

int QuickSortNonRecursive(int low,int high); //快排的非递归辅助函数

void MergeSort(); //2路归并非递归

void Merge(int low,int mid,int high);//归并过程

void MSort(int T\_len); //2路归并的非递归调用

void RadixSort();

int maxbit (); //基数排序的辅助函数，用于求数据的最大位数

};

#endif

#### 1.6.3.2 Sort.cpp

#include<iostream>

#include"sort.h"

#include<time.h>

#include<stack>

using namespace std;

//冒泡

void Sort::BubbleSort(){

for (int i = 0;i < len - 1;i++){

for (int j = 0;j < len - i -1;j++){

if (a[j] > a[j+1]){

int temp = a[j+1];

a[j] = a[j+1];

a[j+1] = temp;

}

}

}

}

//选择

void Sort::SimpleSelectionSort(){

for (int i = 0;i < len - 1;i++){

int min = i;

for (int j = i+1;j < len;j++){

if (a[j] < a[min])

min = j;

}

int temp = a[i];

a[i] = a[min];

a[min] = temp;

}

}

//插入

void Sort::StraightInsertionSort(){

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

int key = a[i];

int j = i-1;

while((j>=0) && (key<a[j])){

a[j+1] = a[j];

j--;

}

a[j+1] = key;

}

}

//希尔

void Sort::ShellSort(){

int h = 1;

while (h < len / 3) {

h = 3 \* h + 1;

}

while (h >= 1) {

for (int i = h; i < len; i++) {

for (int j = i; j >= h && a[j] < a[j - h]; j -= h) {

int temp = a[j];

a[j] = a[j-h];

a[j-h] = temp;

}

}

h = h / 3;

}

}

//归并

void Sort::Merge(int low,int mid,int high){

int \*T = new int[high - low + 1];

int i = low,j = mid + 1,k = 0;

while (i <= mid && j <= high){

if ( a[i] <= a[j] ){

T[k] = a[i]; i++; k++;

}

else {

T[k] = a[j]; j++; k++;

}

}

while ( i <= mid ){

T[k] = a[j]; i++; k++;

}

while ( j <= high ){

T[k] = a[j]; j++; k++;

}

for (k = 0,i = low; i<=high; k++,i++)

a[i] = T[k];

}

void Sort::MSort(int T\_len){

int i = 1;

while (i+2\*T\_len <= len){

//归并两个子序列

Merge(i,i+T\_len-1,i+2\*T\_len-1);

i = i+2\*T\_len;

}

if (i + T\_len <= len){

Merge(i,i+T\_len-1,len); //归并最后两个子序列

}

}

void Sort::MergeSort(){

for (int T\_len = 1; T\_len <= len; T\_len = 2\* T\_len){

MSort(T\_len);

}

}

//快排

int Sort::NumberOfThree(int low,int high){

int mid = low + ((high - low) >> 1);//右移相当于除以2

if (a[mid] > a[high])

{

std::swap(a[mid],a[high]);

}

if (a[low] > a[high])

{

std::swap(a[low],a[high]);

}

if (a[mid] > a[low])

{

std::swap(a[mid],a[low]);

}

//此时，arr[mid] <= arr[low] <= arr[high]

return a[low];

}

int Sort::QuickSortNonRecursive(int low,int high){

int pivot = NumberOfThree(low,high];

while (low < high) {

while (low < high && a[high] >= pivot) {

--high;

}

a[low] = a[high];

while (low < high && a[low] <= pivot) {

++low;

}

a[high] = a[low];

}

a[low] = pivot;

return low;

}

void Sort::QuickSort(int left,int right){

//手动利用栈来存储每次快排的起始点

//栈非空时循环获取中轴入栈

stack<int> s;

if( left<right )

{

int boundary = QuickSortNonRecursive(left,right);

if( boundary-1>left ) //确保左分区存在

{

//将左分区端点入栈

s.push(left);

s.push(boundary-1);

}

if( boundary+1<right ) //确保右分区存在

{

s.push(boundary+1);

s.push(right);

}

while( !s.empty() )

{

//得到某分区的左右边界

int r = s.top();

s.pop();

int l = s.top();

s.pop();

boundary = QuickSortNonRecursive(l,r);

if( boundary-1>l ) //确保左分区存在

{

//将左分区端点入栈

s.push(l);

s.push(boundary-1);

}

if( boundary+1<r ) //确保右分区存在

{

s.push(boundary+1);

s.push(r);

}

}

}

}

//堆排序

void Sort::HeapMax(int start,int end){

int parent = start;

int child = parent \* 2 + 1;

//建立堆

while (child <= end){

if (child + 1 <= end && a[child] < a[child + 1])

child++;

if (a[parent] > a[child])

return; //维护完成

else{

std::swap(a[parent],a[child]);

parent = child;

child = parent \* 2 + 1;

}

}

}

void Sort::HeapSort(){

int i;

//初始化

for (i = len/2 - 1;i >= 0;i--)

Sort::HeapMax(i,len-1);

for (i = len -1;i > 0;i--){

std::swap(a[0],a[i]);

Sort::HeapMax(0,i-1);

}

}

//基数排序

int Sort::maxbit(){

int maxData = a[0];

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

if (maxData < a[i])

maxData = a[i];

}

int d = 1;

int p = 10;

while (maxData >= p){

maxData /= 10;

++d;

}

return d;

}

void Sort::RadixSort(){

int d = maxbit();

int \*tmp = new int[len];

int \*count = new int[10];//末位计数器

int i,j,k;//循环变量

int radix = 1;

for (i = 1;i <= d;i++){

for(j = 0; j < 10; j++)

count[j] = 0; //每次分配前清空计数器

for(j = 0; j < len; j++){

k = (a[j] / radix) % 10; //统计每个桶中的记录数

count[k]++;

}

for(j = 1; j < 10; j++)

count[j] = count[j - 1] + count[j]; //将tmp中的位置依次分配给每个桶

for(j = len - 1; j >= 0; j--){ //将所有桶中记录依次收集到tmp中

k = (a[j] / radix) % 10;

tmp[count[k] - 1] = a[j];

count[k]--;

}

for(j = 0; j < len; j++) //将临时数组的内容复制到data中

a[j] = tmp[j];

radix = radix \* 10;

}

delete []tmp;

delete []count;

}

void Sort::copyArr(){

a = (int\*)malloc(sizeof(int)\*num);

for (int i = 0;i<len;i++){

a[i] = origin[i];

}

}

void Sort::ShowSort(int info){

cout<<"第"<<info<<"个样本的排序用时如下:"<<endl;

copyArr();

cout<<"排序前数组初始化完成"<<endl;

startTime = clock();

BubbleSort();

endTime = clock();

free(a);

cout<<"该样本冒泡排序用时："<<endTime - startTime<<" ms"<<endl;

copyArr();

startTime = clock();

SimpleSelectionSort();

endTime = clock();

cout<<"该样本选择排序用时："<<endTime - startTime<<" ms"<<endl;

copyArr();

startTime = clock();

StraightInsertionSort();

endTime = clock();

cout<<"该样本插入排序用时："<<endTime - startTime<<" ms"<<endl;

copyArr();

startTime = clock();

ShellSort();

endTime = clock();

cout<<"该样本希尔排序用时："<<endTime - startTime<<" ms"<<endl;

copyArr();

startTime = clock();

MergeSort();

endTime = clock();

cout<<"该样本归并排序用时："<<endTime - startTime<<" ms"<<endl;

copyArr();

startTime = clock();

HeapSort();

endTime = clock();

cout<<"该样本堆排序用时："<<endTime - startTime<<" ms"<<endl;

copyArr();

startTime = clock();

QuickSort(0,49999);

endTime = clock();

cout<<"该样本快速排序用时："<<endTime - startTime<<" ms"<<endl;

copyArr();

startTime = clock();

RadixSort();

endTime = clock();

cout<<"该样本基数排序用时："<<endTime - startTime<<" ms"<<endl;

}

#### 1.6.3.3 Main.cpp

#include <iostream>

#include"sort.h"

#include <time.h>

#include<fstream>

using namespace std;

#define arrNumber 50000

/\* run this program using the console pauser or add your own getch, system("pause") or input loop \*/

//随机创建样本

void CreateRandomTxt()

{

srand((int)time(0));

int i = 1;

ofstream out;

out.open("input.txt");

if (!out)

{

cout << "打开文件失败!";

exit(1);

}

for (i; i < 50001; i++)

{

out << rand() % 50000 + 1<<' ';

if (i%20==0)

{

out << endl;

}

}

out.close();

}

//创建升序样本

void CreateOrderTxtp()

{

int i = 1;

ofstream out;

out.open("input.txt");

if (!out)

{

cout << "打开文件失败!";

exit(1);

}

for (i; i < 50001; i++)

{

out << i<<' ';

if (i % 20 == 0)

{

out << endl;

}

}

cout<<"创建升序样本完成"<<endl;

out.close();

}

//创建降序样本

void CreateOrderTxtn()

{

int i = 50000;

ofstream out;

out.open("input.txt");

if (!out)

{

cout << "打开文件失败!";

exit(1);

}

for (i; i >0; i--)

{

out << i<<' ';

if (i % 20 == 0)

{

out << endl;

}

}

cout<<"创建降序样本完成"<<endl;

out.close();

}

//将样本输入数组

void Inarr(int a[])

{

int i = 1;

ifstream in;

in.open("input.txt");

if (!in)

{

cout << "打开样本文件失败!";

exit(0);

}

for (i; i <= 50000; i++)

{

in >> a[i];

}

in.close();

}

int main( ) {

int a[arrNumber];

cout <<"第一个样本（升序）排序如下"<<endl;

CreateOrderTxtp();

Inarr(a);

Sort temp(a,arrNumber);

temp.ShowSort(1);

cout <<"第二个样本（降序）排序如下"<<endl;

CreateOrderTxtn();

Inarr(a);

Sort temp1(a,arrNumber);

temp1.ShowSort(2);

for (int i = 3;i <= 10;i++){

CreateRandomTxt();

Inarr(a);

Sort temp3(a,arrNumber);

temp3.ShowSort(i);

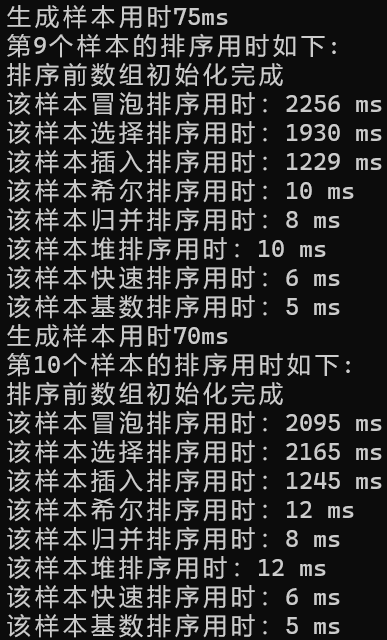
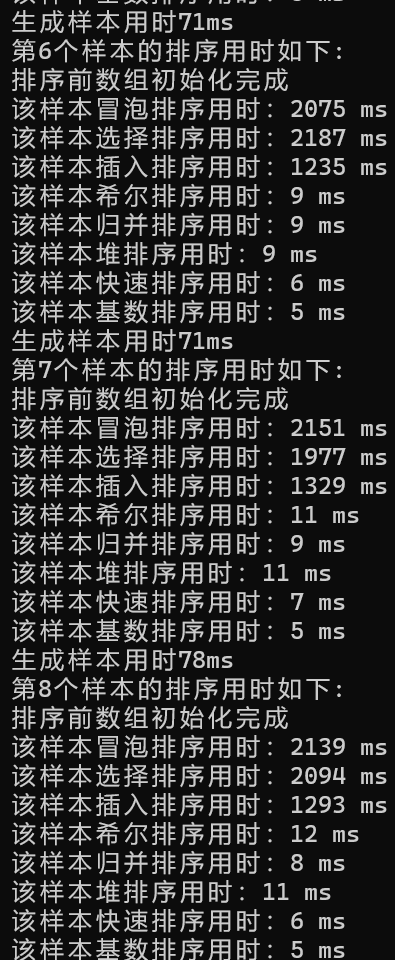
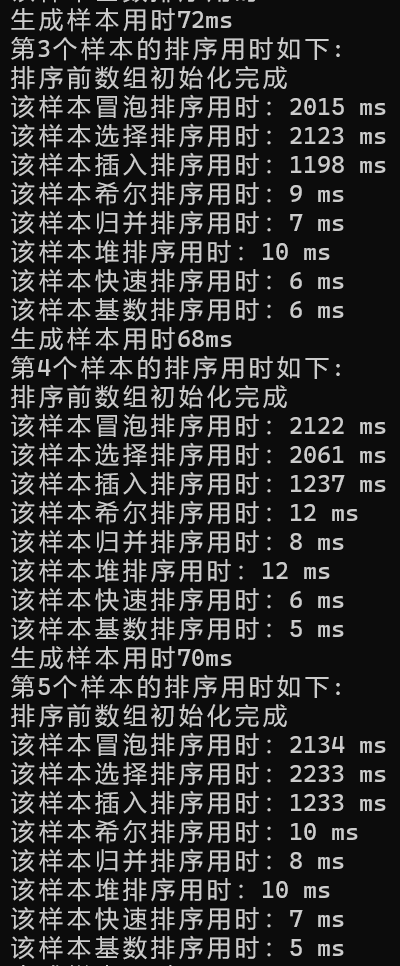
}

return 0;

}

### 1.6.4 测试数据和结果：

（改进前）



**(改进后的部分结果)**

**快速排序运行时间在升序样本中由1318ms变为3ms;降序样本由1463ms变为167ms**

### 1.6.5 时间复杂度：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 排序法 | 平均时间 | 最差情形 | 稳定度 | 额外空间 | 备注 |
| 冒泡 | O(n2) | O(n2) | 稳定 | O(1) | n小时较好 |
| 选择 | O(n2) | O(n2) | 不稳定 | O(1) | n小时较好 |
| 插入 | O(n2) | O(n2) | 稳定 | O(1) | 大部分已排序时较好 |
| 基数 | O(logRB) | O(logRB) | 稳定 | O(n) | B是真数(0-9)，  R是基数(个十百) |
| Shell | O(nlogn) | O(ns) 1<s<2 | 不稳定 | O(1) | s是所选分组 |
| 快速 | O(nlogn) | O(n2) | 不稳定 | O(nlogn) | n大时较好 |
| 归并 | O(nlogn) | O(nlogn) | 稳定 | O(1) | n大时较好 |
| 堆 | O(nlogn) | O(nlogn) | 不稳定 | O(1) | n大时较好 |

### 1.6.6 反思：

由于每次运行都需要随机生成样本到文件中，可增加一个函数来控制是否需要新生成新的样本

可在类中有一个枚举记录排序名称，再将输出八种排序算法时间的函数分解为一个根据参数输出其中一种排序函数时间的函数，再在类中构建另外一个函数来循环输出所有排序算法时间的函数，提高健壮性，同时也便于控制输出排序算法时间的种类个数。

### 1.6.7 改进

结果显示，在面临升降序样本和随机样本时，快速排序运行时间差距较大，这是由于在待排数组基本有序的情况下，选择使用固定基准效率会大大降低，若数组元素是随机的，使用固定基准常常优于随机基准。可改为三数取中的办法选择基准。在程序中改进后快速排序运行时间在升序样本中由1318ms变为3ms;降序样本由1463ms变为167ms

### 1.6.8 该题代码行：

430行

# 第二部分 选做题

## 2.1 -8：棋局评估【2】

### 2.1.1数据结构：

二维数组

### 2.1.2 算法设计思想：

Alice（以下简称A）与Bob（以下简称B）都以最优策略下棋，用二维数组记录两人下棋的情况。先统计当前空格数（两人还没下的格子），然后进行深度优先搜索进行递归，搜索所有下棋的策略，当空格数为0时递归结束，并开始回溯，计算A得分情况并返回A得分情况，并把每一条策略进行比较，得出Alice的最优得分情况

每步递归根据当前递归下棋的对象改变下一步递归中下棋的对象。判断找寻当前对象的最优策略时需要比较函数回溯得来的A得分情况（B最优策略是使A得分情况最少，A最优策略是使A得分情况最多）

### 2.1.3 源程序：

#include<iostream>

using namespace std;

#define MAX -99

#define MIN 99

//采用深度优先算法

// 0:null

// 1:Alice: X(first)

// 2:Bob: O

int board[3][3]; //数组形式记录棋盘

bool Judge (int side){

//判断谁胜利

for (int i = 0;i<3;i++){

//行连成一条线

if (board[i][0] == board[i][1] && board[i][1] == board[i][2] && board[i][2] == side )

return true;

//列连成一条线

if (board[0][i] == board[1][i] && board[1][i] == board[2][i] && board[2][i] == side )

return true;

}

//判断对角线

if (board[0][0] == board[1][1] && board[1][1] == board[2][2] && board[2][2] == side)

return true;

if(board[0][2] == board[1][1] && board[1][1] == board[2][0] && board[2][0] == side)

return true;

return false;

}

int DFS(int side){

int nullNum = 0;

int maxFinal = MAX;

int minFinal = MIN;

for (int i = 0; i < 3; i++){

for (int j = 0; j < 3; j++){

if (board[i][j] == 0)

nullNum ++;

}

}

//返回得分情况

if (side == 1 && Judge(2))

return -nullNum-1;

else if (side == 2 &&Judge(1))

return nullNum+1;

else if (nullNum == 0)

return 0;

//进行下棋

for (int i = 0; i < 3; i++){

for (int j = 0; j < 3; j++){

if (board[i][j] == 0){

board[i][j] = side;

if (side == 1) //当前为Alice下棋，计算 下一步Bob下棋Bob最优策略得分

maxFinal = max(maxFinal,DFS(2));

if (side == 2) //当前为Bob下棋，计算 下一步Alice下棋Alice最优策略得分

minFinal = min(minFinal,DFS(1));

board[i][j] = 0;//回溯时重新初始化

}

}

}

if (side == 1) return maxFinal;

if (side == 2) return minFinal;

}

int main() {

int group;

cin >> group;

while (group--){

for (int i = 0; i < 3; i++){

for (int j = 0; j < 3; j++){

cin >> board[i][j];

}

}

cout << DFS(1) << endl;

}

return 0;

}

//代码总行数：73行

### 2.1.4 测试数据和结果：

输入：

3

1 2 1

2 1 2

0 0 0

2 1 1

0 2 1

0 0 2

0 0 0

0 0 0

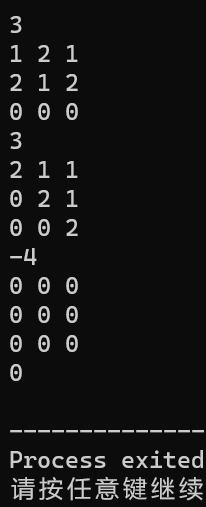
0 0 0

输出：

3

-4

0



### 2.1.5 时间复杂度：

DFS时间复杂度O(n^2），判断输赢的函数时间复杂度O（n^2）

### 2.1.6 该题代码行：

73行

## 2.2 -10：URL映射【2】

### 2.2.1 数据结构：

字符串类型string，将字符串与流关联的对象stringstream，STL库中的vector类型

### 2.2.2 算法设计思想：

每条规则分为两部分：规则本体和规则名字，故可封装在一起作为一个类Rule。获取规则和地址时用字符串存储，然后进行分割成多个可匹配项，由于需要便捷按“/”分割，故使用string和stringstream，由于有多个规则和地址且不定长，故使用vector来做为二维字符串类型数组，vector<string>存储地址，vector<Rule>存储规则。

思想很简单。获取当前地址时逐个与每条规则的每个可匹配项尝试匹配，匹配到其中一条规则根据该规则输出相应规则名字。

值得注意的是地址末尾若有分隔符，该分隔符不能当作分隔符，而是当作地址最后一个可匹配项的一部分。在之后的分割过程和比较判断匹配过程需特别注意到这种情况。

### 2.2.3 源程序：

#include<cstdio>

#include<algorithm>

#include<iostream>

#include<iomanip>

#include<vector>

#include<string>

#include<string.h>

#include<sstream>

using namespace std;

//规则

class Rule{

public:

string name;

vector<string> content;

Rule(string in\_name,vector<string> in\_content) :name(in\_name),content(in\_content){}

};

//存储数条规则的变量

vector<Rule> rules;

bool isMatch;

//string与int相互转换的函数

//stringstream方便类型转换及字符串分割

int StringToInt(string s){

int answer;

stringstream ss(s);

ss >> answer;

return answer;

}

string IntToString(int answer){

stringstream ss;

ss << answer;

return ss.str();

}

bool JudgeIsNum(string s){

for (int i = 0; i < s.length(); i++)

{

if (s[i] > '9' || s[i] < '0') return false;

}

return true;

}

void HandleURLRule (string s,string name){

int spiltLoc;

string str;

vector<string> content;

bool isSpilt = true;

//URL最后一个字符为 / 是不算做分隔符的,之后也要加入进字符串组里

if (s[s.length()-1] != '/'){

isSpilt = false;

s = s + "/"; //该数据类型同样方便字符拼接

}

spiltLoc = s.find("/");

while (spiltLoc != -1){

str = s.substr(0,spiltLoc); //获取分割后的其中一个字符串

s = s.substr(spiltLoc + 1,s.length());

content.push\_back(str);

spiltLoc = s.find("/");

}

if (isSpilt){

content.push\_back("/");

}

rules.push\_back(Rule(name,content));

}

//判断符合规则函数，就一个一个逐个if-else地去判断

vector<string> Match (int k, string url){

int spiltLoc;

int i;

string str;

bool isSpilt = true;

vector<string> result;

vector<string> urlCur;

if (url[url.length() - 1] != '/')

{

isSpilt = false;

url = url + "/";

}

int urlIndex = 0;

spiltLoc = url.find("/");

while (spiltLoc != -1)

{

str = url.substr(0,spiltLoc);

url = url.substr(spiltLoc + 1,url.length());

urlCur.push\_back(str);

spiltLoc = url.find("/");

}

if (isSpilt)

urlCur.push\_back("/");

for (i = 0; i < rules[k].content.size() && urlIndex < urlCur.size(); i++){

if (rules[k].content[i] == "<path>" && urlCur[urlIndex].length() != 0){

str = urlCur[urlIndex];

for (int j = urlIndex + 1;j < urlCur.size();j++){

if (urlCur[j] == "/"){

str = str + "/";

}

else

str = str + "/" + urlCur[j];

}

result.push\_back(str);

urlIndex = urlCur.size();

i = rules[k].content.size();

}

else if (rules[k].content[i] == "<int>" && urlCur[urlIndex].length() != 0){

if (JudgeIsNum(urlCur[urlIndex])){

result.push\_back(IntToString(StringToInt(urlCur[urlIndex])));

urlIndex++;

}

else {

isMatch = false;

break;

}

}

else if (rules[k].content[i] == "<str>" && urlCur[urlIndex].length() != 0){

result.push\_back(urlCur[urlIndex]);

urlIndex++;

}

else {

if (rules[k].content[i] == urlCur[urlIndex])

urlIndex++;

else{

isMatch = false;

break;

}

}

}

//未匹配完整

if (urlIndex != urlCur.size() || i < rules[k].content.size()){

isMatch = false;

}

return result;

}

int main(){

string format,name,url;

int ruleNum,addressNum;

int error;

int j;

cin >> ruleNum >> addressNum;

for (int i = 0; i < ruleNum; i++)

{

cin >> format >> name;

format = format.substr(1, format.length());

HandleURLRule(format,name);

/\* code \*/

}

for (int i = 0; i < addressNum; i++)

{

cin >> url;

url = url.substr(1,url.length());

//开始对当前地址进行规则匹配

for ( j = 0; j < ruleNum; j++)

{

isMatch = true;

vector<string> vec = Match(j,url);

if (isMatch)

{

cout << rules[j].name;

for (int h = 0; h < vec.size(); h++)

{

cout << " "<< vec[h];

/\* code \*/

}

cout << endl;

break;

/\* code \*/

}

/\* code \*/

}

if (j == ruleNum) cout<<"404"<<endl;

/\* code \*/

}

return 0;

}

### 2.2.4 测试数据和结果：

输入：

5 4

/articles/2003/ special\_case\_2003

/articles/<int>/ year\_archive

/articles/<int>/<int>/ month\_archive

/articles/<int>/<int>/<str>/ article\_detail

/static/<path> static\_serve

/articles/2004/

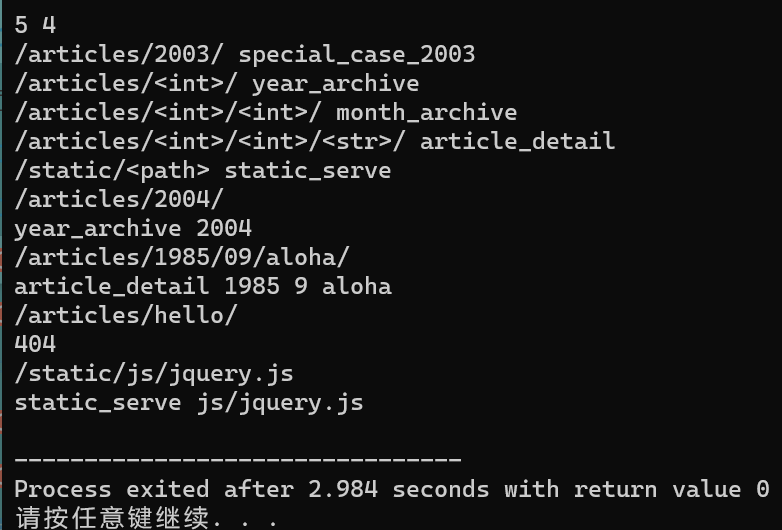
/articles/1985/09/aloha/

/articles/hello/

/static/js/jquery.js

输出：

year\_archive 2004  
article\_detail 1985 9 aloha  
404  
static\_serve js/jquery.js



### 2.2.5 时间复杂度：

时间复杂度O(n\*m+(n\*m)\*j）

(n为规则数，m为地址数，j为分割后的字符串组中字符串的个数)

### 2.2.6 该题代码行：

172行

## 2.3 -11：词梯【2】

### 2.3.1 数据结构：

存储单词表为了可以快速通过单词字符串查找到是否存在相对应的单词，故使用STL的map容器，即map<string, string>。

为了存储词梯路径边两点的信息，用STL的pair<string,string>

同时，为了存储从起点或终点开始搜索的边，同样使用STL的map<string, string>

两个方向的访问辅助哈希表同样使用STL的map<string, string>

在双向BFS时要使用队列，即STL的queue<string>

由于词梯路径数量不定，故用STL的vector<string>存储可以用来输出路径的中间单词

### 2.3.2 算法设计思想：

#### 2.3.2.1 算法核心：

双向BFS

#### 2.3.2.2 算法说明：

先读取文件用CreatDict()函数创建字典，在获取起点词和终点词时同时判断输入是否合法。

然后用WorldLadder函数找到并输出词梯路径。

可以对起点词根据改变规则得到单词进行BFS搜索，搜索到终点词即可获得最短路径。单词的获得可以通过对要变换的词每个位置逐个变换一个字母，然后判断是否在单词表里获得。

由于随着终点词所在的搜索层次的深度增加，所对应的宽度也可能增加，且该层次除了终点词，其他词的被搜索都是没有必要且浪费时间的。同时也有可能面临着搜索空间的爆炸问题。

因为我们同时知道起点词和终点词，可以对BFS进行优化为**双向BFS**。同时从起点和终点两个方向开始搜索，一旦搜索到相同的值，意味着找到了一条联通起点和终点的最短路径。

这样可以起到剪枝和提高效率的作用。

双向 BFS的基本实现思路如下：

1. 创建「两个队列」分别用于两个方向的搜索；
2. 创建「两个哈希表（map<string, string>）」用于「解决相同节点重复搜索」和「记录转换次数」；
3. 为了尽可能让两个搜索方向“平均”，每次从队列中取值进行扩展时，先判断哪个队列容量较少；
4. 如果在搜索过程中「搜索到对方搜索过的节点」，说明找到了最短路径。

输出路径则借助「搜索到对方搜索过的节点」（简称mid），找到mid到start的路径并反转，与mid到end的路径拼接即为整条路径。

### 2.3.4 源程序：

#include <bits/stdc++.h>

using namespace std;

map<string, string> dict,startSet,endSet; //单词表，起点出发搜索的边集合 终点出发搜索的边集合

vector<string> midPath; //存储中间点以便找到路径

int len; //字符串 长度

typedef pair<string,string> line;

ifstream in;

void CreatDict (){

in.open("words.txt");

if (!in){

cout << "打开文件 words.txt 失败";

exit(1);

}

string temp;

while ( getline(in,temp) ){

dict.insert(make\_pair(temp,temp));

}

in.close();

cout <<"已读取建立单词表"<<endl;

}

bool Check(string isExit){

if (dict.count(isExit)) return true;

else return false;

}

//flag : 0为从起点开始搜索，1为从终点开始搜索

int update(queue <string> &q,map<string, string> &visit,map<string, string> &other,int count,int flag){

int isFind = 0;

for(int i = 0;i < count;i++){

string st = q.front();

q.pop();

for (int j = 0;j < len;j++){

string temp = st;

for (char ch = 'a';ch <='z';ch++){

temp[j] = ch;

if (dict.count(temp) && (!visit.count(temp))){

//若单词存在且没在当前方向被访问过

q.push(temp);

//存出边以便找寻路径

if(flag == 0) startSet[temp] = st;

else{

endSet[temp] = st;

}

visit[temp] = temp;

//若单词被另一方向搜索过，则找到路径，返回值为-1

if (other.count(temp)) {

midPath.push\_back(temp);

isFind = -1;

}

}

}

}

}

if (isFind == -1) return isFind;

else return q.size();

}

//双向BFS

void WorldLadder(string start,string end){

//初始化

queue <string> fromStart,fromEnd;

map<string, string> visitS,visitE;

//从起点或终点开始的搜索队列访问数组

visitS[start] = start;

visitE[end] = end;

//从起点或终点开始的搜索队列

fromStart.push(start);

fromEnd.push(end);

int countS = 1,countE = 1;//当前搜索层的元素个数

int isExitRoad = 0;

//双向BFS主要过程

while (!fromStart.empty() && !fromEnd.empty()){

if (fromStart.size() <= fromEnd.size()){

countS = update(fromStart,visitS,visitE,countS,0);

}

else{

countE = update(fromEnd,visitE,visitS,countE,1);

}

//返回值为-1则说明词梯找到，终止搜索

if (countS == -1 || countE == -1) {

cout << "词梯已找寻到\n";

isExitRoad = 1;

break;

}

}

if (!isExitRoad) cout <<"词梯不存在"<<endl;

//输出路径

cout <<"开始输出词梯\n"<<endl;

for (auto it = midPath.begin(); it != midPath.end(); it++)

{

vector<string> path;

string tmp = \*it;

string tmp1 = \*it;

path.push\_back(tmp);

//根据中间词找到该词到起点的路径并转换

while (tmp != start){

tmp = startSet[tmp];

path.push\_back(tmp);

};

reverse(path.begin(),path.end());

//根据中间词找到该词到终点的路径

while (tmp1 != end){

tmp1 = endSet[tmp1];

path.push\_back(tmp1);

}

for (auto it = path.begin(); it != path.end(); it++)

{

cout <<(\*it);

if (it != path.end()-1){

cout<<"->";

}

}

cout <<endl;

}

}

int main(){

CreatDict();

string start,end;

//输入

while (1){

cout <<"请输入起点词\n";

cin >>start;

if (Check(start)) break;

else {

cout <<"该词不存在，请重新输入\n";

continue;

}

}

while (1){

cout <<"请输入终点词\n";

cin >>end;

if (start.length() != end.length()){

cout <<"两词长度不等，请重新输入终点词\n";

continue;

}

if (Check(end)) break;

else {

cout <<"该词不存在，请重新输入\n";

continue;

}

}

//处理输出

if (start == end){

cout << "两词相同，无需转换，词梯长度为0\n";

}

else {

len = start.length();

WorldLadder(start,end);

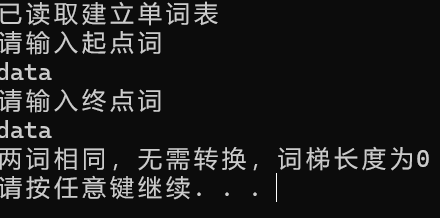
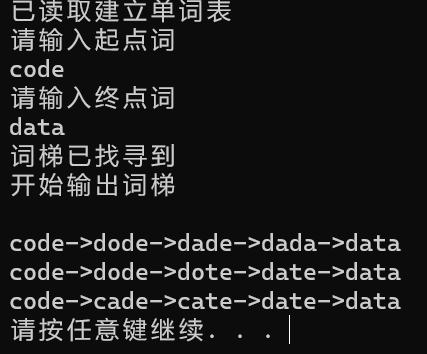
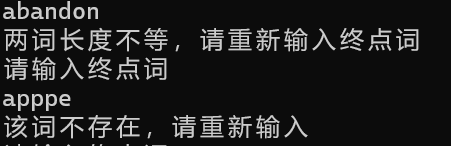
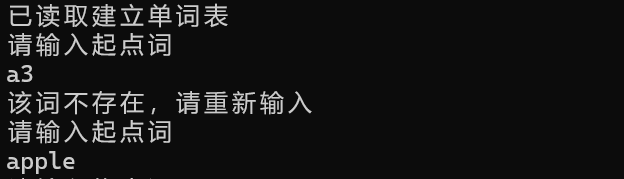
}

system("pause");

return 0;

}

### 2.3.5 测试数据和结果





### 2.3.6 时间复杂度：

朴素BFS的时间复杂度大概为O(n^2),双向BFS算法时间复杂度大概为O (（n^2）/2)。

### 2.3.7 反思：

函数间的耦合度较高，且每个函数所拥有的功能超过两个，不便于后续的维护和复用。

判断是否被访问过的哈希表和存储单词之间的边的信息的map容器实际上可以合并在一起，减少空间的使用。

函数里的嵌套较多，可以尝试将其中的一些嵌套封装为函数来被调用。

### 2.3.8 改进：

### 2.3.9 该题代码行：

152行

## 2.4-15：迷宫问题【2】

### 2.4.1 数据结构：

用于存储坐标的结构体point

struct point{

int x;

int y;

};

由于存储迷宫地图的结构体 maze

typedef struct maze{

char mazeMap[MAZEMAXSIZE][MAZEMAXSIZE];

}maze,\*mazepoint;

### 2.4.2 算法设计思想：

由于题目要求中使用的是栈的操作。故本题采用DFS回溯算法。使用字符数组记录迷宫地图。

使用函数CreateMaze（）从文件中读取迷宫，文件中第一行的两个数字分别是迷宫的长度和宽度，然后后面为迷宫，其中0表示通路可走，1表示墙壁不可走。然后获取迷宫入口和出口位置，判断是否输入错误。输入错误情况如下：1，输入的坐标超过迷宫大小； 2.输入的坐标为墙壁。鉴于入口和出口位置在迷宫里面具有一定的合理性，故不判断为错误。

使用函数Maze()进行DFS算法和回溯算法找出路径。并输出路径图和以坐标和箭头表示的路径。

### 2.4.3 源程序：

#include<iostream>

#include<fstream>

#include<iomanip>

#include<stack>

#include<Windows.h>

using namespace std;

#define red SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), FOREGROUND\_INTENSITY | FOREGROUND\_RED) //红色

#define blue SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), FOREGROUND\_BLUE) //蓝色

#define yellow SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), FOREGROUND\_INTENSITY | FOREGROUND\_RED | FOREGROUND\_GREEN) //黄色

#define green SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), FOREGROUND\_INTENSITY | FOREGROUND\_GREEN) //绿色

#define white SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), FOREGROUND\_INTENSITY | FOREGROUND\_RED | FOREGROUND\_GREEN | FOREGROUND\_BLUE) //白色

#define ching SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), FOREGROUND\_INTENSITY | FOREGROUND\_GREEN |FOREGROUND\_BLUE) //青色

#define MAZEMAXSIZE 200

//1为墙壁，0为通路

//坐标

struct point{

int x;

int y;

};

//迷宫

typedef struct maze{

char mazeMap[MAZEMAXSIZE][MAZEMAXSIZE];

}maze,\*mazepoint;

bool check\_board(int x,int mx){

if (x < 0) {

cout << "您的坐标超出边界"<<endl;

return false;

}

else if (x >= mx) {

cout << "您的坐标超出边界"<<endl;

return false;

}

else return true;

}

//创建迷宫

void CreateMaze(mazepoint p,point &start,point &end ,int &x,int &y)

{

//存储迷宫长和宽

//int x, y;

ifstream in;

in.open("maze.txt");

if (!in)

{

cout << "打开 maze.txt 失败" << endl;

exit(1);

}

//输入迷宫

in >> x >> y;

for (int i = 0; i < x; i++)

{

for (int j = 0; j < y; j++)

{

in >>p->mazeMap[i][j];

}

}

in.close();

//输出迷宫

cout << "原迷宫为:\n(用1表示墙壁)\n(用0表示通路)" << endl;

cout<<" ";

for (int j = 0; j < y; j++)

cout<< setw(3)<<setiosflags(ios::left)<<j;

cout << endl;

for (int i = 0; i < x; i++)

{

cout << setw(3)<<setiosflags(ios::left) << i;

for (int j = 0; j < y; j++)

{

if ( p->mazeMap[i][j] == '1'){

//墙壁

red;

cout << p->mazeMap[i][j]<<" ";

white;

}

else if (p->mazeMap[i][j] == '0'){

//通路

yellow;

cout << p->mazeMap[i][j]<<" ";

white;

}

if (j == y - 1)

cout << endl;

}

}

cout << "你可输入的横坐标范围为 0至"<<x-1<<endl;

cout << "你可输入的纵坐标范围为 0至"<<y-1<<endl;

while (1){

cout << "请输入迷宫的入口坐标:";

cin >> start.x >> start.y;

if (check\_board(start.x,x) && check\_board(start.y,y)) {

if (p->mazeMap[start.x][start.y] =='1'){

cout << "输入的入口坐标为墙壁，请重新输入\n"<<endl;

continue;

}

else break;

}

else {

cout <<"输入入口坐标有误，请重新输入\n"<<endl;

continue;

}

}

while (1){

cout << "请输入迷宫的出口坐标:";

cin >> end.x >> end.y;

if (check\_board (end.x,x)&& check\_board(end.y,y)){

if (p->mazeMap[start.x][start.y] =='1'){

cout << "输入的出口坐标为墙壁，请重新输入\n"<<endl;

continue;

}

else break;

}

else {

cout <<"输入出口坐标有误，请重新输入\n"<<endl;

continue;

}

}

}

//走迷宫

void Mazes(mazepoint p,point &start,point &end,int &boardX,int &boardY ){

stack<point> Stack;

Stack.push(start);

//cur表示当前位置

point cur = start,q;

int cur\_x = start.x;

int cur\_y = start.y;

//是否走过标志

bool flag = false;

//依次下左上右表示四个方向的移动，i表示几个方向被尝试过；

int moveing[4][2] = {{0,1},{1,0},{0,-1},{-1,0}};

int i = 0;

while (!Stack.empty() && ((cur.x != end.x)||(cur.y != end.y)) ){

cur = Stack.top();

flag = 0;

i = 0;

//从当前点开始尝试四个方向

while (flag == 0 && i < 4) {

cur\_x = cur.x + moveing[i][0];

cur\_y = cur.y + moveing[i][1];

if (p->mazeMap[cur\_x][cur\_y] == '0'){

cur.x = cur\_x;

cur.y = cur\_y;

Stack.push(cur);

// \*表示通路

p->mazeMap[cur\_x][cur\_y] = '\*';

flag = 1; //该点已经走过

}

i++; //换方向

}

if (i == 4 && flag == 0)

{

p->mazeMap[cur.x][cur.y] = '#'; //那么将当前点置为墙壁,便于处理

Stack.pop(); //将该点出栈回到上一个点

}

}

p -> mazeMap[start.x][start.y] ='\*';

p -> mazeMap[end.x][end.y] ='\*';

if (Stack.empty()){

cout << "没有路径"<<endl;

}

else{

//输出迷宫路径图

cout << "解后迷宫路径如下\n(用1表示墙壁)\n(用0表示通路)\n(用\*表示路径)\n(用#表示用算法回溯时探索过的路径，但不是答案路径)"<<endl;

cout<<" ";

for (int j = 0; j < boardY; j++)

cout<< setw(3)<<setiosflags(ios::left)<<j;

cout << endl;

for (int i = 0; i < boardX; i++)

{

cout << setw(3)<<setiosflags(ios::left) << i;

for (int j = 0; j < boardY; j++)

{

if ( p->mazeMap[i][j] == '1'){

//墙壁

red;

cout << p->mazeMap[i][j]<<" ";

white;

}

else if (p->mazeMap[i][j] == '0'){

//通路

yellow;

cout << p->mazeMap[i][j]<<" ";

white;

}

else if (p->mazeMap[i][j] == '\*'){

green;

cout << p->mazeMap[i][j]<<" ";

white;

}

else if (p->mazeMap[i][j] == '#'){

ching;

cout << p->mazeMap[i][j]<<" ";

white;

}

if (j == boardY - 1)

cout << endl;

}

}

cout << endl;

//输出迷宫路径表

//栈反转

stack<point> temp;

while (!Stack.empty()){

q = Stack.top();

Stack.pop();

temp.push(q);

}

cout << "以坐标显示的迷宫路径如下："<<endl;

int huanhang = 0;

while (!temp.empty()){

q = temp.top();

temp.pop();

huanhang++;

cout << "("<< q.x <<","<<q.y<<")";

if (!temp.empty()) cout << "->";

if (huanhang % 8 == 0) cout<<endl;

}

//StackTraverse(S);

}

}

int main(){

maze map;

mazepoint p = &map;

point start,end;

int x,y;

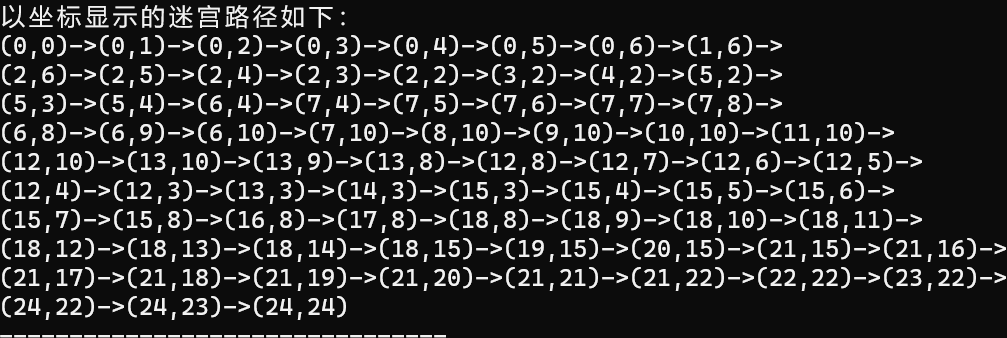
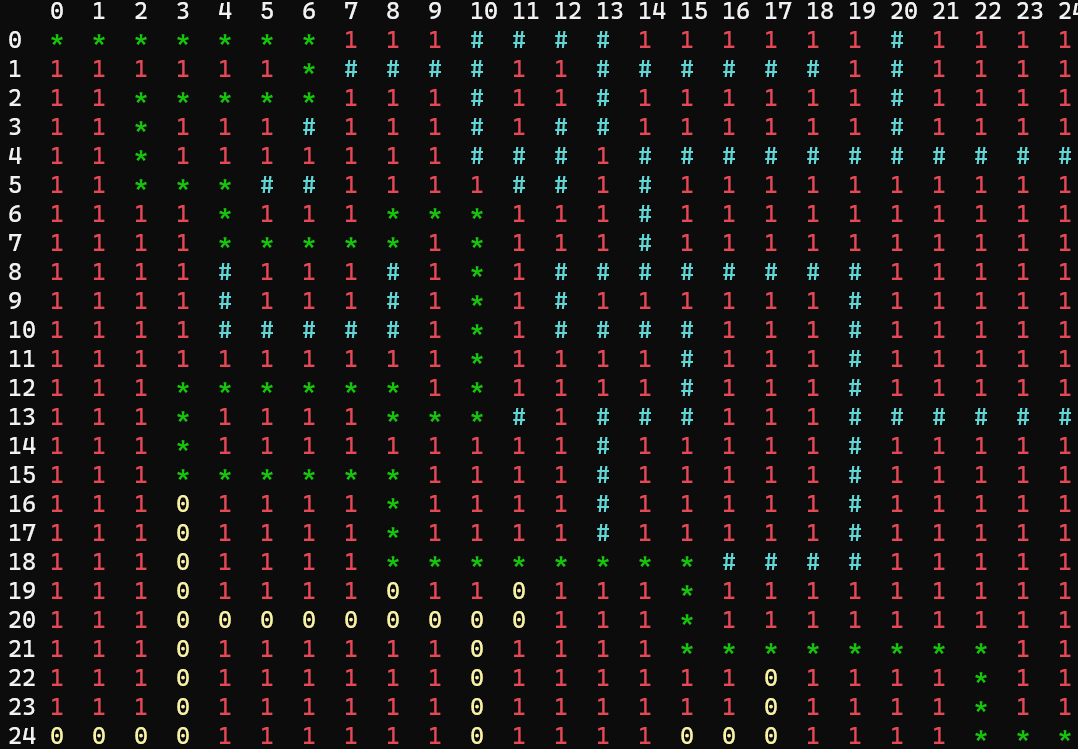
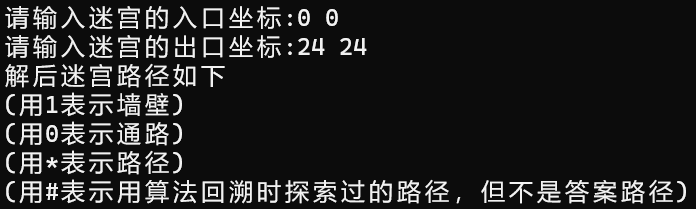
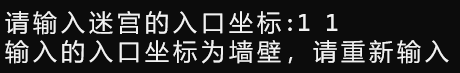
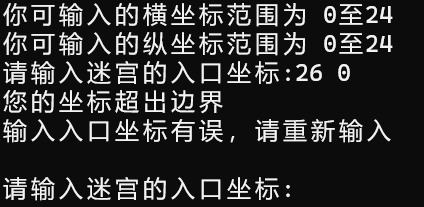
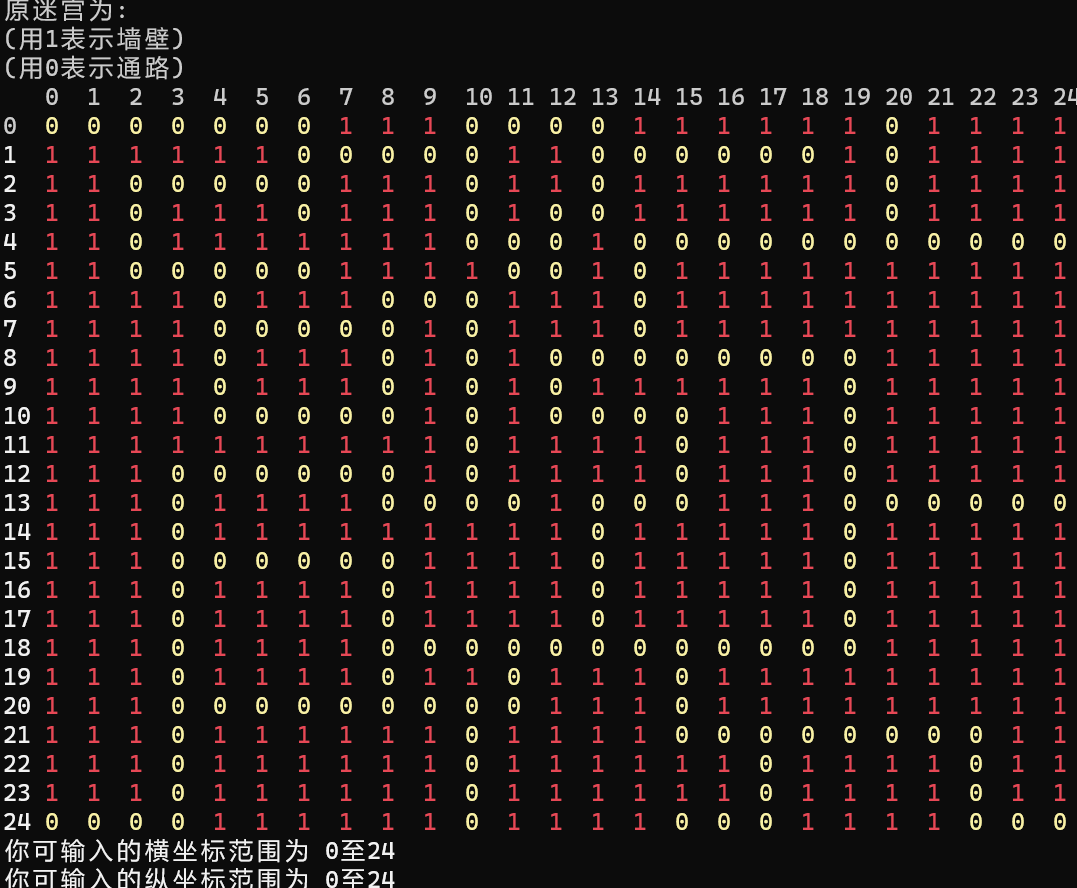
CreateMaze(p,start,end,x,y);

Mazes(p,start,end,x,y);

return 0;

}

### 2.4.4 测试数据和结果



### 2.4.5 时间复杂度：

DFS算法O（N）

### 2.4.6 反思：

还可以将输出功能封装为一个函数，还可以再做出一个功能，利用清屏函数和Sleep函数将程序找路径的过程演示为一个“动画”。

### 2.4.7 改进：

### 2.4.8 该题代码行：

234行

## 2.5-20：平衡二叉树操作的演示【4】

### 2.5.1 数据结构：

采用二叉树的链式存储结构。下面是二叉树结点（node）的结构体

struct Node {

int key;

int value;

int height;

Node \*left;

Node \*right;

// 构造函数

Node(int k, int v) : key(k), value(v), height(1), left(nullptr), right(nullptr) {}

};

输出数据时采用层序遍历，使用到了STL中的队列（queue）

源文件里有n行，每行有两个整数，用一个空格间隔，第一个数字为关键字值，第二个为该关键字所对应的值value

### 2.5.2 算法设计思想：

在主函数里获取源文件里的键值对并建立AVL树，打印输出菜单，并根据用户输入的指令进行相应操作,即查找，插入，删除，输出整个AVL树，退出程序以及提示输入指令错误。并调用相关函数，

每个操作函数又由几个基础的对树操作的基础函数组成;

1. 获取节点的高度函数 int getHeight(Node \*node)
2. 取节点的平衡因子函数 int getBalanceFactor(Node \*node)
3. 更新节点的高度函数 void updateHeight(Node \*node)
4. 左旋转函数 Node\* leftRotate(Node \*node)
5. 右旋转函数 Node\* rightRotate(Node \*node)
6. 寻找右子树最小结点函数 Node\* findMin(Node \*node)

对于AVL树的三个基础操作函数查找，插入及删除函数如下

1. 查找函数Node\* find(Node \*node, int key)：

采用递归的方法，若要查找的关键字比当前结点的关键字大，则找寻当前结点的右子树；若小，则找寻左子树。直至找到关键字值相同的结点或者递归至叶子结点仍未找到关键字值相同的结点

1. 插入函数Node\* insert(Node \*node, int key, int value)

同样采用递归的方法，按照插入二叉查找树结点的方法插入结点，如果key已经存在，则更新节点的值。然后更新节点高度并计算平衡因子，调用相关函数。

如果节点的平衡因子大于1，则需要进行旋转操作。其中如果新插入的节点在节点的左子树的左侧，则进行右旋；如果新插入的节点在节点的左子树的右侧，则先进行左旋再进行右旋。

如果节点的平衡因子小于-1，则需要进行旋转操作。其中如果新插入的节点在节点的右子树的右侧，则进行左旋；如果新插入的节点在节点的右子树的左侧，则先进行右旋再进行左旋。

1. 删除函数Node\* remove(Node \*node, int key)

查找到对应节点，如果节点无子树，删除该结点即可。如果节点仅有一个子树，对应节点的位置由它的孩子结点代替。如果节点既有左子树又有右子树，则找到右子树中的最小节点来替换对应节点。然后更新节点的值。然后更新节点高度并计算平衡因子，调用相关函数。

1. 建立AVL树，即读取文件使用插入函数直至读取文件完毕。
2. 输出AVL树函数，void printTree(Node \*node)：

使用层序遍历输出对应结点的key值和左右孩子结点key值。

1. 将修改后的树覆盖写入源文件函数void printTreeFile(Node\* root,ofstream &fout)

用户决定执行退出操作时调用。

### 2.5.3 源程序：

#include <iostream>

#include <fstream>

#include <stdlib.h>

#include <iomanip>

#include <queue>

using namespace std;

// 定义AVL树节点

struct Node {

int key;

int value;

int height;

Node \*left;

Node \*right;

// 构造函数

Node(int k, int v) : key(k), value(v), height(1), left(nullptr), right(nullptr) {}

};

// 获取节点的高度

int getHeight(Node \*node) {

if (node == nullptr) return 0;

return node->height;

}

// 获取节点的平衡因子

int getBalanceFactor(Node \*node) {

if (node == nullptr) return 0;

return getHeight(node->left) - getHeight(node->right);

}

// 更新节点的高度

void updateHeight(Node \*node) {

node->height = max(getHeight(node->left), getHeight(node->right)) + 1;

}

// 左旋

Node\* leftRotate(Node \*node) {

Node \*temp = node->right;

node->right = temp->left;

temp->left = node;

updateHeight(node);

updateHeight(temp);

return temp;

}

// 右旋

Node\* rightRotate(Node \*node) {

Node \*temp = node->left;

node->left = temp->right;

temp->right = node;

updateHeight(node);

updateHeight(temp);

return temp;

}

Node\* findMin(Node \*node) {

if (node == nullptr || node->left == nullptr) return node;

return findMin(node->left);

}

// 在AVL树中插入新节点

Node\* insert(Node \*node, int key, int value) {

if (node == nullptr) return new Node(key, value);

if (key < node->key) {

node->left = insert(node->left, key, value);

} else if (key > node->key) {

node->right = insert(node->right, key, value);

} else {

// 如果key已经存在，则更新节点的值

node->value = value;

return node;

}

// 更新节点的高度

updateHeight(node);

// 计算节点的平衡因子

int balanceFactor = getBalanceFactor(node);

// 如果节点的平衡因子大于1，则需要进行旋转操作

if (balanceFactor > 1) {

// 如果新插入的节点在节点的左子树的左侧，则进行右旋

if (key < node->left->key) {

return rightRotate(node);

}

// 如果新插入的节点在节点的左子树的右侧，则先进行左旋再进行右旋

if (key > node->left->key) {

node->left = leftRotate(node->left);

return rightRotate(node);

}

}

// 如果节点的平衡因子小于-1，则需要进行旋转操作

if (balanceFactor < -1) {

// 如果新插入的节点在节点的右子树的右侧，则进行左旋

if (key > node->right->key) {

return leftRotate(node);

}

// 如果新插入的节点在节点的右子树的左侧，则先进行右旋再进行左旋

if (key < node->right->key) {

node->right = rightRotate(node->right);

return leftRotate(node);

}

}

return node;

}

// 在AVL树中查找节点

Node\* find(Node \*node, int key) {

if (node == nullptr) return nullptr;

if (key < node->key) {

return find(node->left, key);

} else if (key > node->key) {

return find(node->right, key);

} else {

return node;

}

}

// 在AVL树中删除节点

Node\* remove(Node \*node, int key) {

if (node == nullptr) return nullptr;

if (key < node->key) {

node->left = remove(node->left, key);

} else if (key > node->key) {

node->right = remove(node->right, key);

} else {

if (node->left == nullptr && node->right == nullptr) {

delete node;

return nullptr;

} else if (node->left == nullptr) {

Node \*temp = node->right;

delete node;

return temp;

}

else if (node->right == nullptr) {

Node \*temp = node->left;

delete node;

return temp;

} else {

// 如果节点既有左子树又有右子树，则找到右子树中的最小节点来替换当前节点

Node \*successor = findMin(node->right);

node->key = successor->key;

node->value = successor->value;

node->right = remove(node->right, successor->key);

}

}

// 更新节点的高度

updateHeight(node);

// 计算节点的平衡因子

int balanceFactor = getBalanceFactor(node);

// 如果节点的平衡因子大于1，则需要进行旋转操作

if (balanceFactor > 1) {

// 如果节点的左子树的平衡因子大于0，则进行右旋

if (getBalanceFactor(node->left) > 0) {

return rightRotate(node);

}

// 如果节点的左子树的平衡因子小于0，则先进行左旋再进行右旋

if (getBalanceFactor(node->left) < 0) {

node->left = leftRotate(node->left);

return rightRotate(node);

}

}

// 如果节点的平衡因子小于-1，则需要进行旋转操作

if (balanceFactor < -1) {

// 如果节点的右子树的平衡因子小于0，则进行左旋

if (getBalanceFactor(node->right) < 0) {

return leftRotate(node);

}

// 如果节点的右子树的平衡因子大于0，则先进行右旋再进行左旋

if (getBalanceFactor(node->right) > 0) {

node->right = rightRotate(node->right);

return leftRotate(node);

}

}

return node;

}

//打印二叉树

void printTree(Node \*node) {

if (node == nullptr) return;

queue<Node \*> q;

q.push(node);

Node \*temp;

cout<<"key lchild rchild"<<endl;

while (!q.empty()){

temp = q.front();

q.pop();

cout << std::left << setw(4)<<temp->key;

if (temp->left) {

q.push(temp->left);

cout<< std::left << setw(7)<<temp->left->key;

}else {

cout<< std::left << setw(7)<<" ";

}

if (temp->right) {

q.push(temp->right);

cout<< std::left << setw(7)<<temp->right->key<<endl;;

}else {

cout<<endl;

}

}

cout << endl;

}

//存文件

void printTreeFile(Node\* root,ofstream &fout) {

if (root == nullptr) {

return;

}

fout << root->key<<" "<<root->value << endl;

printTreeFile(root->left,fout);

printTreeFile(root->right,fout);

}

int main() {

// 从文件中读取数据

ifstream fin("data.txt");

if (!fin) {

cerr << "打开文件data.txt失败." << endl;

return 1;

}

// 建立AVL树

Node \*root = nullptr;

int key, value;

while (fin >> key >> value) {

root = insert(root, key, value);

}

while (1){

// 菜单

int command;

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

cout<<"\*\* 1.查找节点 \*\*"<<endl;

cout<<"\*\* 2.插入节点 \*\*"<<endl;

cout<<"\*\* 3.删除节点 \*\*"<<endl;

cout<<"\*\* 4.输出AVL树\*\*"<<endl;

cout<<"\*\* 5. 退出 \*\*"<<endl;

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

cin >> command;

if (command == 1){

// 查找节点

int searchKey;

cout << "请输入一个整数关键字Key进行查找节点: ";

cin >> searchKey;

Node \*node = find(root, searchKey);

if (node != nullptr) {

cout << "Key: " << node->key << ", Value: " << node->value << endl;

} else {

cout << "关键字未找到." << endl;

}

system("pause");

} else if (command == 2){

// 插入节点

int insertKey, insertValue;

cout << "请输入一个整数关键字Key和一个整数Value进行插入节点: ";

cin >> insertKey >> insertValue;

root = insert(root, insertKey, insertValue);

cout << "插入成功."<<endl;

system("pause");

} else if (command == 3){

// 删除节点

int removeKey;

cout << "请输入一个整数关键字Key进行删除节点: ";

cin >> removeKey;

root = remove(root, removeKey);

cout << "删除成功，接下来将输出整棵树观察结果"<<endl;

// 以可验证的方式输出结果

cout << "AVL Tree: " << endl;

printTree(root);

system("pause");

} else if (command == 4){

// 以可验证的方式输出结果

cout << "AVL Tree: " << endl;

printTree(root);

system("pause");

} else if (command == 5){

// 修改后结果存放至data.txt，以覆盖的形式

ofstream fout("data.txt");

if (!fout) {

cerr << "打开文件data.txt失败." << endl;

return 1;

}

printTreeFile(root,fout);

break;

} else {

cout << "输入指令无效，请重新输入"<<endl;

system("pause");

}

//清屏

system("cls");

}

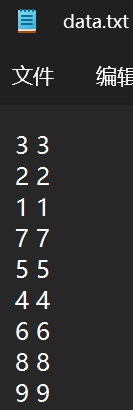
system("pause");

return 0;

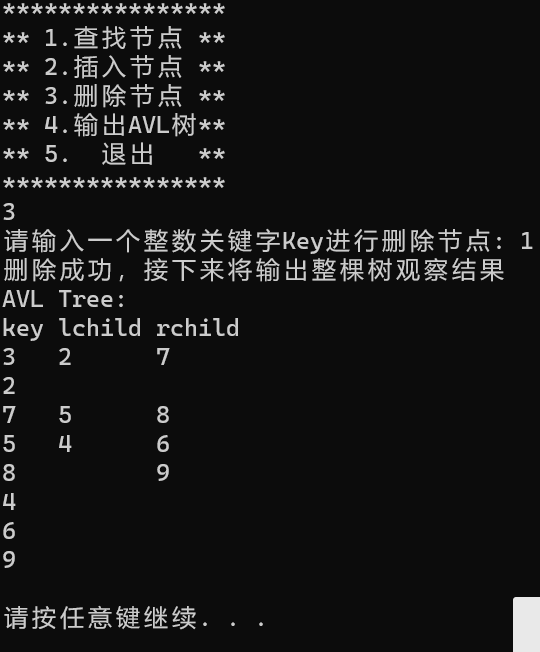
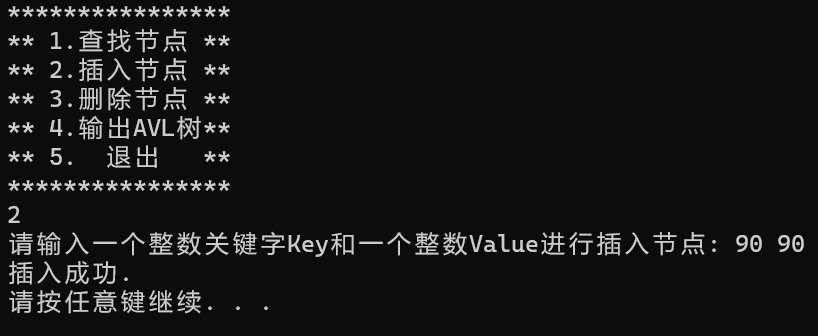
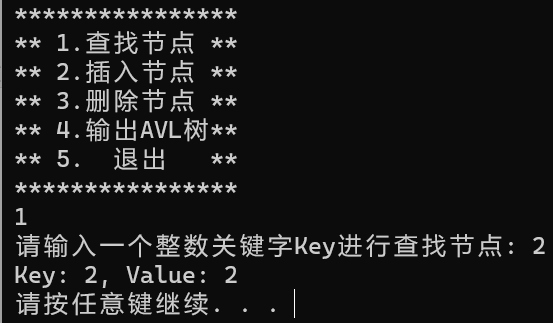
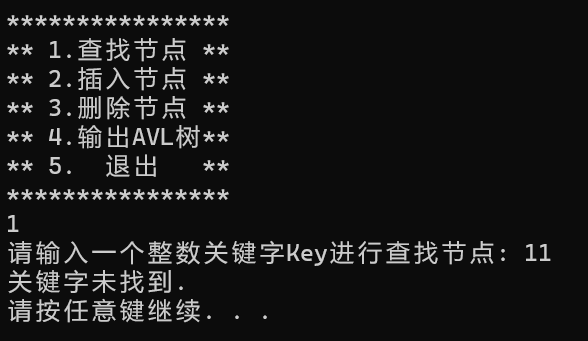
}

### 2.5.4 测试数据和结果

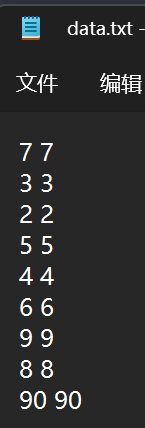
#### 测试数据



#### 2.5.4.2 结果



#### 2.5.4.3 测试结果中的源文件



### 2.5.5 时间复杂度：

* insert(Node \*node, int key, int value)：O(h)
* remove(Node \*node, int key)：O(h)
* find(Node \*node, int key)：O(h)
* 其中，h为AVL树的高度。由于AVL树是一种平衡二叉树，所以它的高度一般不会超过O(logn)，因此这段代码中的各个函数的时间复杂度也就不会超过O(logn)

find(Node \*node, int key)：该函数的时间复杂度为O(h)，其中h为AVL树的高度。在最坏的情况下，每次查找操作都会沿着AVL树的最底层进行搜索，这样就会导致时间复杂度升高。但是由于AVL树是一种平衡二叉树，所以它的高度一般不会超过O(logn)，因此查找操作的时间复杂度也就不会超过O(logn)。

remove(Node \*node, int key)：该函数的时间复杂度为O(h)，其中h为AVL树的高度。在最坏的情况下，每次删除操作都会沿着AVL树的最底层进行搜索，这样就会导致时间复杂度升高。但是由于AVL树是一种平衡二叉树，所以它的高度一般不会超过O(logn)，因此删除操作的时间复杂度也就不会超过O(logn)。

insert(Node \*node, int key, int value)：该函数的时间复杂度为O(h)，其中h为AVL树的高度。在最坏的情况下，每次插入操作都会沿着AVL树的最左侧或最右侧进行搜索，这样就会导致时间复杂度升高。但是由于AVL树是一种平衡二叉树，所以它的高度一般不会超过O(logn)，因此插入操作的时间复杂度也就不会超过O(logn)。

### 2.5.6 反思：

菜单功能呢写在主函数不方便复用。输出树的信息不够直观可视化。

### 2.5.7 改进：

仅在退出时才把现在在程序上的树保存至源文件，不执行其他操作就保存，减少程序不必要运行，

### 2.5.8 该题代码行：

300行

第三部分 心得体会

#### 3.1 程序总行数：

共3083行

#### 3.2 心得：

这学期老师生动形象地为我们介绍了各种各样的数据结构，我真的学到了很多，由于这些数据结构知识地帮助，我将会在今后的的学习和工作中得到了优厚的基础

编写程序的过程中，我学会了如何查阅获取到相关的知识，同时还拓宽了我的知识面，让我了解了很多课外的知识。

同时我还收获了许多在网上查找问题，错误和代码的途径。

学习了数据结构，我对计算机程序的运行和优化有了更深刻的认识。

写完这些程序真的非常有成就感，精心制作了这份报告以供查验，感谢您的审阅！