**上海应用技术大学课程设计报告**

课程名称 《数据结构课程设计》

设计题目 各种排序&哈夫曼树的建立&订票系统

院系 计算机科学与信息工程学院 专业 软件工程 班级19104221

姓名 倪路 学号 1910400731 指导教师 柏海芸 日期 2021-1-15

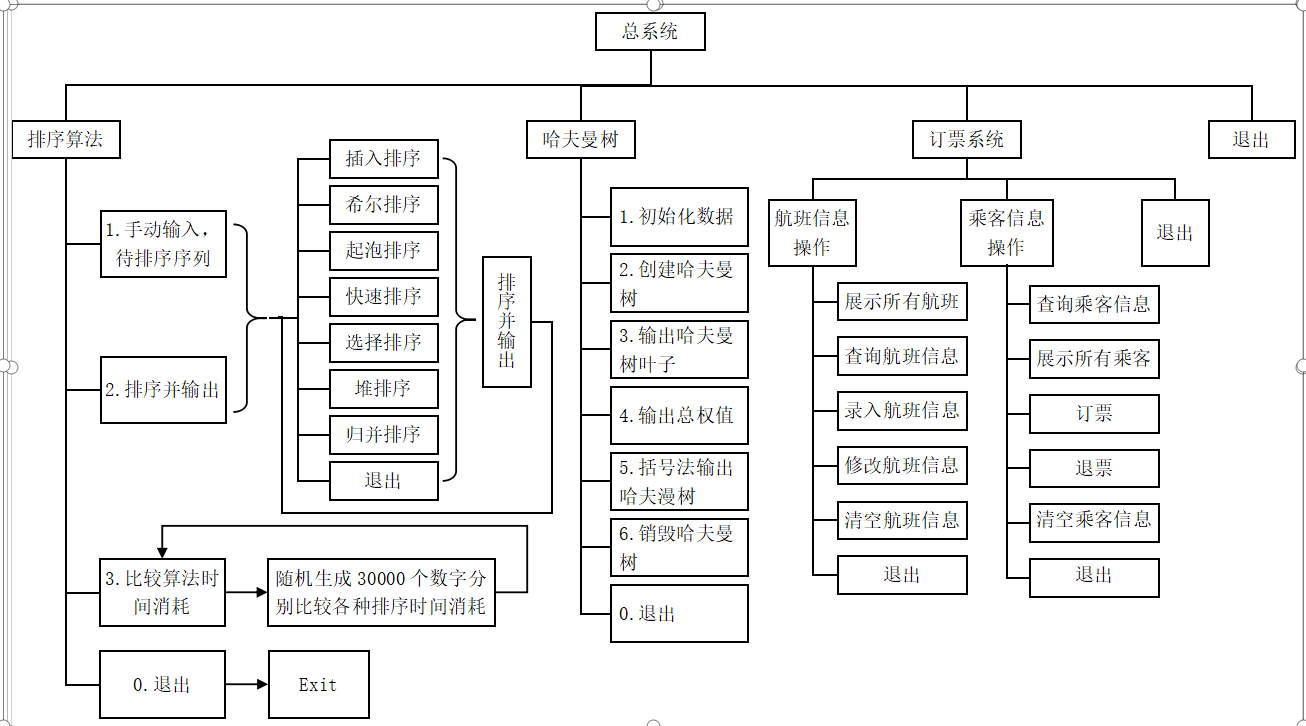
1. **目的与要求**
   1. 巩固和加深对常见数据结构的理解和掌握
   2. 掌握基于数据结构进行算法设计的基本方法
   3. 掌握用高级语言实现算法的基本技能
   4. 掌握书写程序设计说明文档的能力
   5. 提高运用数据结构知识及高级语言解决非数值实际问题的能力
2. **课程设计内容说明**

1、主菜单设计及运行说明

**(1)总体说明**

**本次课程设计我一共编写了三个项目，分别是各种排序算法、哈夫曼树的创建以及订票系统。**

**·系统流程图**

****

1. **主菜单设计**

**主菜单一共设计了三个小模块，分别对应了三个系统的界面**

1. **运行说明**

**用户可以在主菜单根据自己的选择进入不同的系统，以及可以执行退出系统操作**

**2、项目一：各种排序算法**

* + 1. 对设计任务内容的概述

**·问题描述：对30000个随机整数，利用插入排序、希尔排序、起泡排序、快速排序、选择排序、堆排序、归并排序等排序方法进行排序，并统计每一种排序上机所花费的时间。**

**·要求：输入的数据形式为任何一个正整数，大小不限。 输出的形式：数字大小逐个递增的数列**

* + 1. 需求分析或模块功能描述

**·需求分析：对题目进行分析后，本系统需要实现对30000个随机整数的排序并且输出各种不同排序算法的时间消耗；除此之外还应满足用户可手动输入待排序序列进行排序并输出。**

**·模块功能描述：根据题目需求，我一共设计了三个模块，模块一负责用户手动输入待排序序列，模块二负责比较各种排序算法对30000个随机整数的排序时间消耗，模块三负责对用户输入的待排序序列进行排序并输出。**

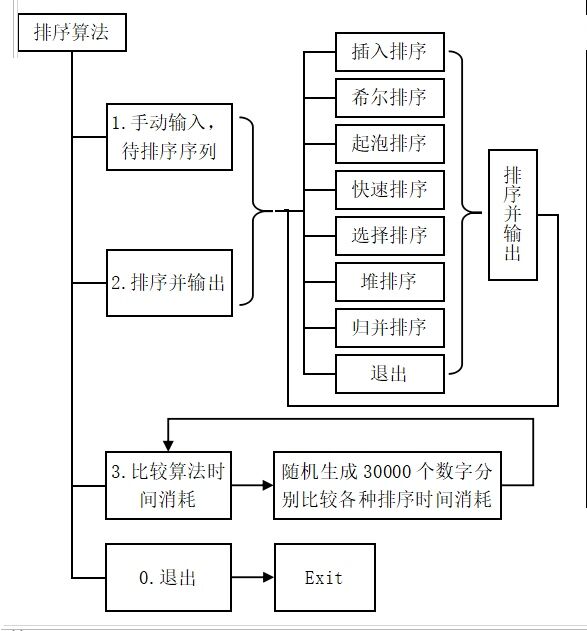
* + 1. 数据的逻辑结构、存储结构设计

**·在本系统中数据的逻辑结构为线性结构，因为本次参与排序的数据均为整型，所以线性结构比较符合本系统要求**

**·存储结构选择的是顺序结构，具体存储数据我使用数组的形式，原因是，题目明确要求了比较时间算法的数据个数为30000个，所以使用数组开辟一个30000个整型大小空间来存储这30000个随机整数是比较合理的**

* + 1. 主要模块流程图或部分源代码说明

**·主要模块流程图**

****

**·详细说明：**

**当用户进入本系统后，用户可以选择手动输入一个待排序序列，输入完成后会将用户数据暂存在一个数组中，之后需用户选择排序并输出并指定排序方式，程序就会将用户所输入待排序序列进行排序并输出至屏幕。用户选择比较时间消耗功能后，程序会随机生成30000个待排序整数分别使用各种排序算法对数据进行排序然后将各个算法所消耗时间输出至屏幕上。**

**·部分源代码说明：**

**在本程序中，进行各个排序算法时间消耗统计的源码如下：**

**clock\_t start\_time = clock();**

**sort(); //排序算法**

**clock\_t end\_time = clock();**

**cout << "排序耗时 : " << (double)(end\_time - start\_time) / CLOCKS\_PER\_SEC**

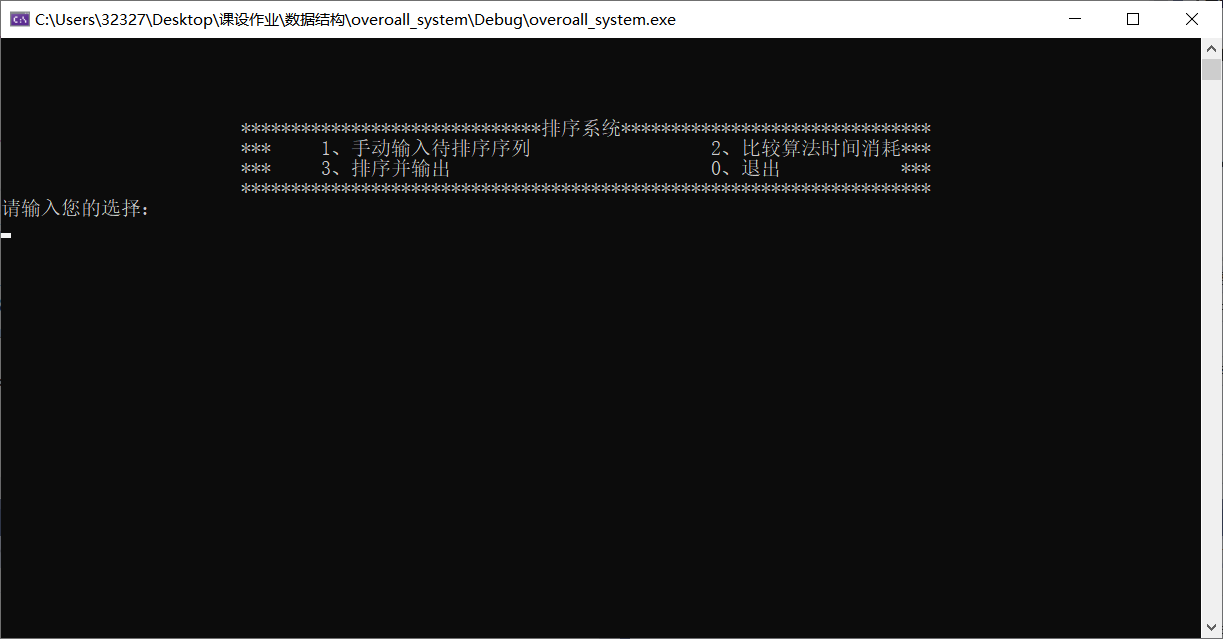
**流程图：**

****

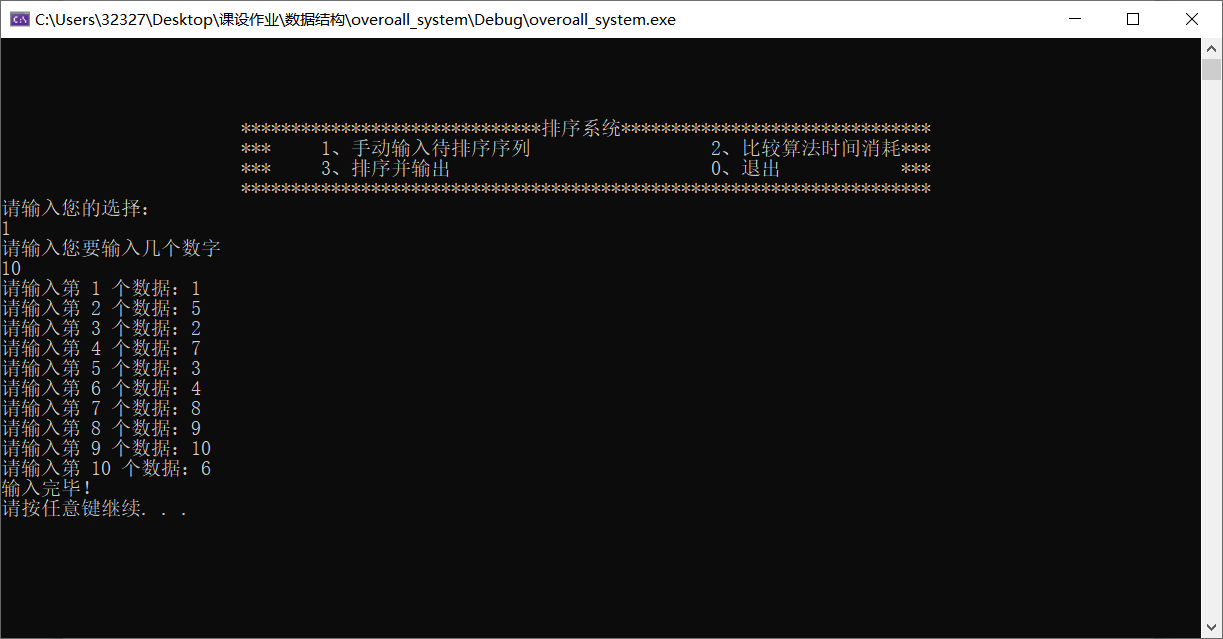
**要执行本代码需要导入ctime 库，通过该程序段，可以实现对各种算法耗时的精确统计**

* + 1. 调试分析或程序测试

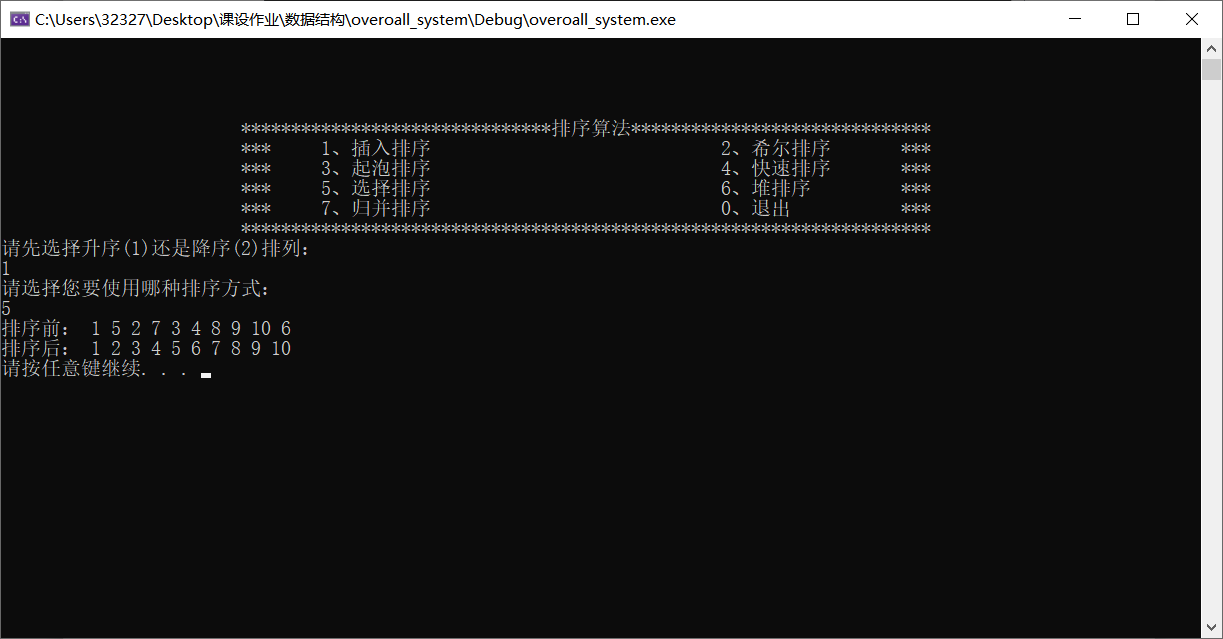
**程序调试：**



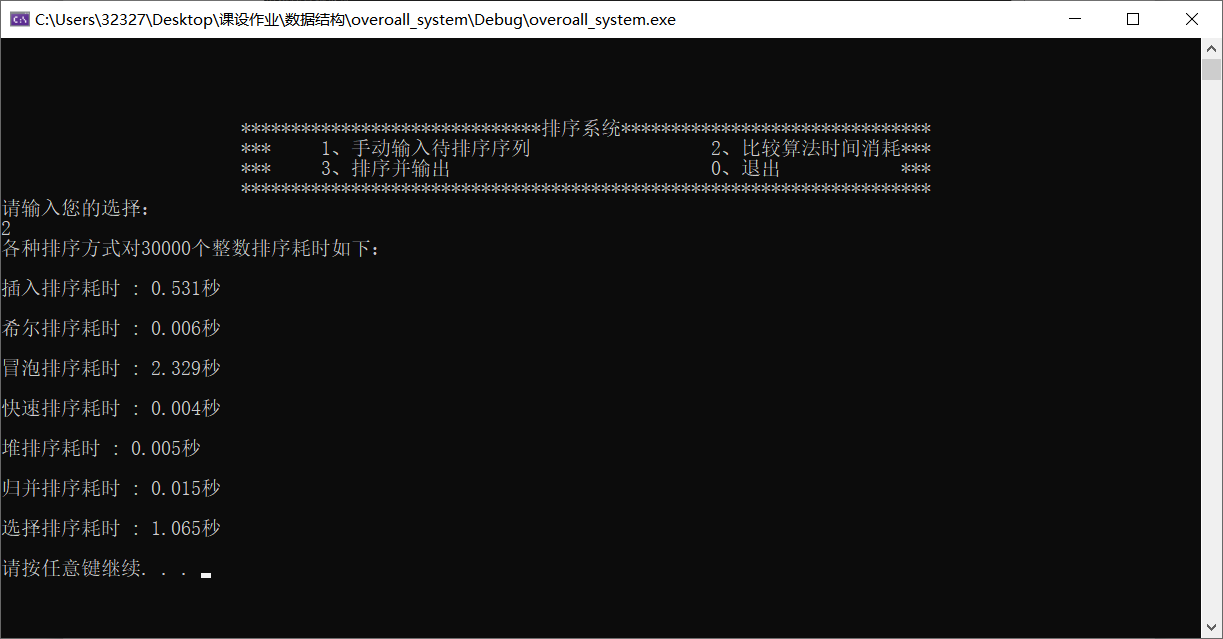
主界面



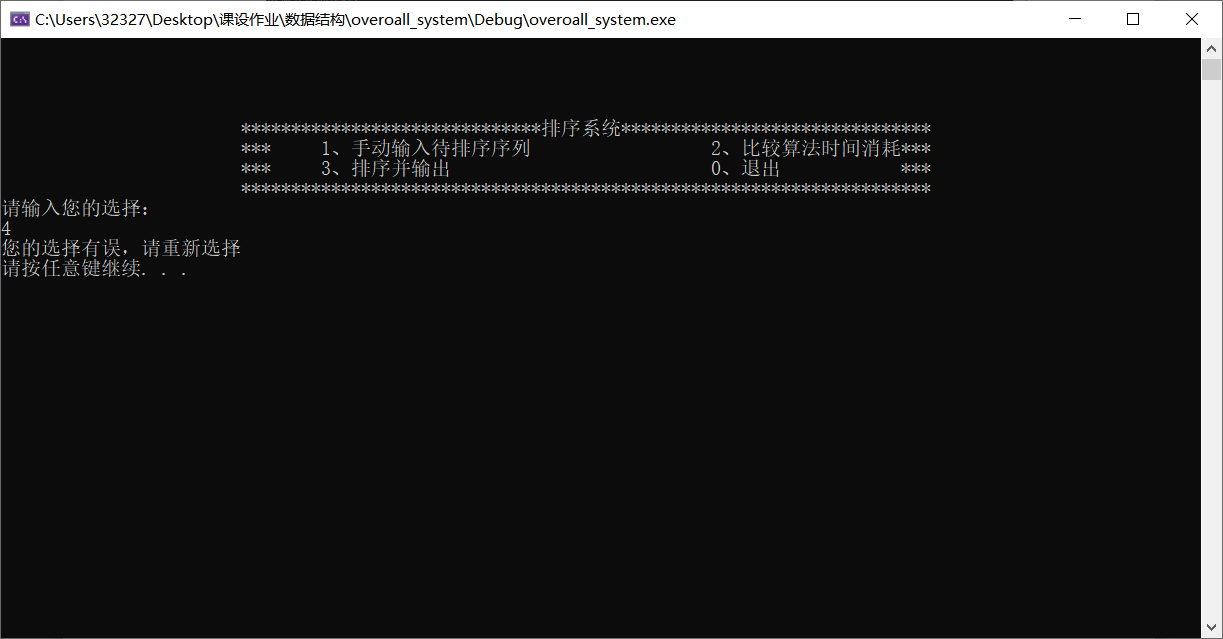
输入待排序序列



排序输出



比较算法时间消耗



输入有误

**根据程序调试结果，可以很清晰的得出各种排序算法的时间效率高低，快速排序和堆排序、希尔排序三种排序算法时间效率相较于其他算法比较高效，而冒泡排序时间效率则显得很低下。**

* + 1. 尚未解决的问题或改进方向

**在本程序中对各种排序算法比较时间消耗时只是简单的对各种算法进行了时间消耗的计算，没有按照各种算法时间效率高低进行排序，程序在这一点上还就可以进行稍微的改进，将各种排序算法按照效率由高到低进行排序，能更加直观的让用户知道各种排序算法的效率高低。**

**3、项目二：哈夫曼树的创建**

(1)对设计任务内容的概述

**（1）问题描述：建立建立最优二叉树函数**

**（2）要求：可以建立函数输入二叉树，并输出其赫夫曼树**

(2)需求分析或模块功能描述

**·需求分析：对题目进行分析后了解到，本系统需要实现用户输入一组数据用于创建哈夫曼树，还需要实现一个函数来构建该哈夫曼树，一个函数用于计算该哈夫曼的权值以验证该哈夫曼树是否为最优二叉树，一个函数用于输出该哈夫曼树。**

**·模块功能描述：根据题目需求，我一共设计了五大模块，模块一负责用户手动输入一组初始数据以用于创建哈夫曼树，模块二则负责将用户输入的数据按照哈夫曼树的规则组合成为一棵哈夫曼树，模块三负责将创建好的哈夫曼树进行输出，这里可以根据用户的选择可以输出哈夫曼树的叶子节点，以及可以用括号法输出哈夫曼树；模块四负责计算哈夫曼树的带权路径的值，用户可以通过此数据来验证该二叉树是否为最优二叉树；模块五则用于销毁二叉树。**

(3)数据的逻辑结构、存储结构设计

**·由于哈夫曼树其本质为一颗二叉树，所以在本系统中数据的逻辑结构为树形结构。**

**·存储本系统中所需数据我选择了使用链式存储结构，原因是，哈夫曼树的每一个节点之间链接关系需要即时的进行更新改变，每一个节点都有一个指针域以及一个数据域，所以一般的线性存储结构无法满足要求，而且使用链式结构可以更加方便二叉树的遍历等操作，所以链式存储结构为本程序的最佳存储结构。**

(4)主要模块流程图或部分源代码说明

**·主要模块流程图**

****

**·详细说明：**

**当用户进入本系统后，需首先进行第一模块的数据初始化操作，输入一组数据来创建一颗哈夫曼树，输入完成后程序首先会对数据进行一个升序排序，之后便可以进行第二模块的哈夫曼树的创建操作。哈夫曼树创建完成后用户即可进行模块三对哈夫曼树进行输出；模块四则可以计算用户输入的数据所创建的哈夫曼树的带权路径的值并输出；当用户操作完毕后，可以执行模块五来销毁该哈夫曼树释放内存。**

**·部分源代码说明：**

**在本程序中，将用户所输入数据创建成一棵哈夫曼树的源码如下：**

**void creat\_HTree(BTNode\*\* B, int a[], int n)**

**{**

**int min1, min2;**

**for (int i = 0; i < n - 1; i++)**

**{**

**int k = i;**

**while (flag[k] != false) k++;**

**int j = i + 1;**

**while (flag[j] != false) j++;**

**min1 = B[k]->data;**

**min2 = B[j]->data;**

**while (min1 > min2)**

**{**

**min1 = min2;**

**if ((j + 1) < n && min2 > B[(j + 1)]->data) min2 = B[++j]->data;**

**else min2 = B[k]->data;**

**}**

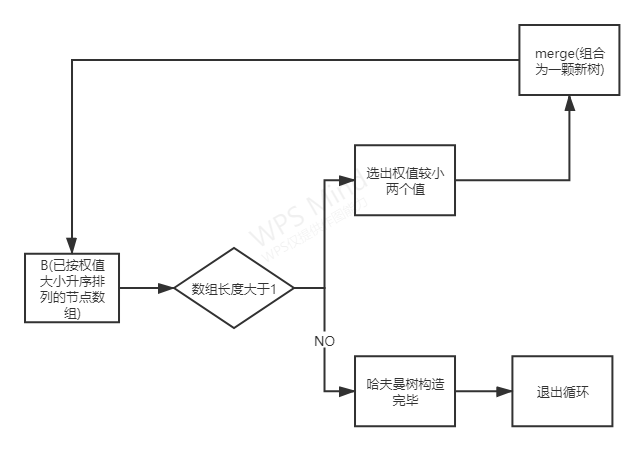
**flag[k] = true;**

**merge(B, \*B[k], \*B[j], j);**

**}**

**}**

**流程图：**

****

**在此段程序中，首先从用户所输入数据进行升序排序后的第一个元素开始循环，每次通过迭代选择出所剩数据的两个最小值，通过merge函数将两个最小的节点组合成为一颗新的二叉树，每两个节点组合成为一颗新二叉树后，这两个节点的父节点会添加至当前剩余数据的首个位置，以参与接下来的迭代，如此迭代直到用户的数据都加入到哈夫曼树中为止。其中flag标识用于表示某个节点是否已经添加入哈夫曼树中。**

**根据本程序段的分析，容易得到创建该哈夫曼树的时间复杂度为O(nlogn)**

**计算哈夫曼树带权路径总和的算法代码段如下：**

**int get\_weight(BTNode\* B, int layer)**

**{**

**if (B == NULL) return 1;**

**if (B->Lchild == NULL && B->Rchild == NULL) return B->data \* layer;**

**if (B->Lchild != NULL && B->Rchild != NULL)**

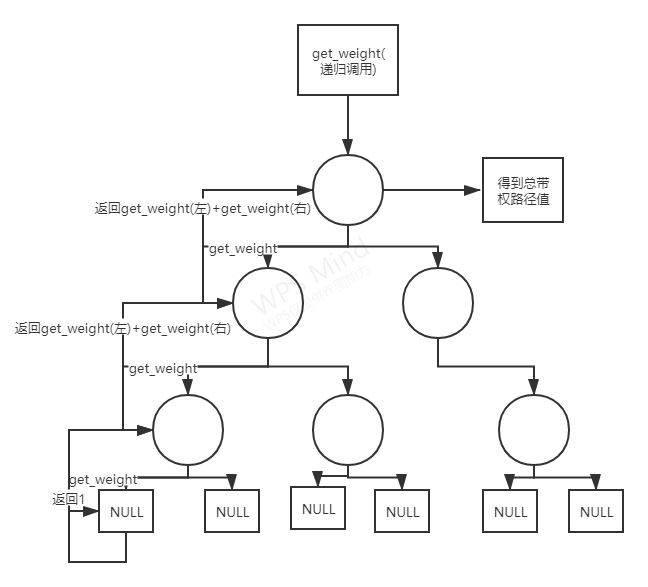
**return get\_weight(B->Lchild, layer + 1) + get\_weight(B->Rchild, layer + 1);**

**else if (B->Lchild != NULL) return get\_weight(B->Lchild, layer + 1);**

**else if (B->Rchild != NULL) return layer \* get\_weight(B->Rchild, layer + 1);**

**}**

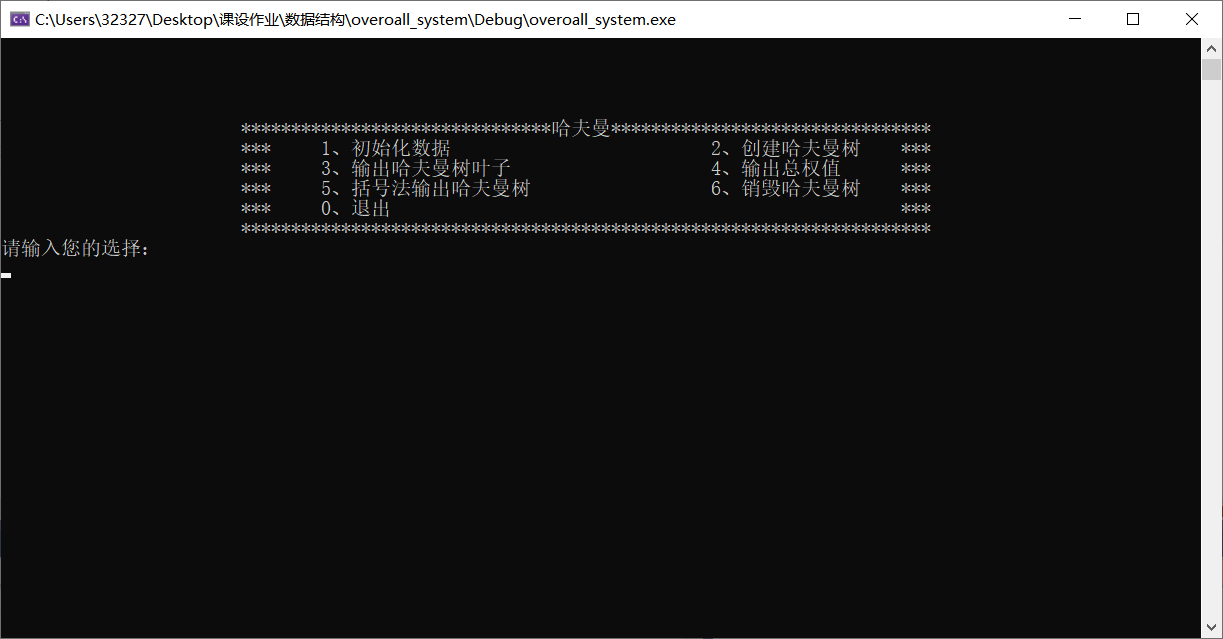
**流程图：**

****

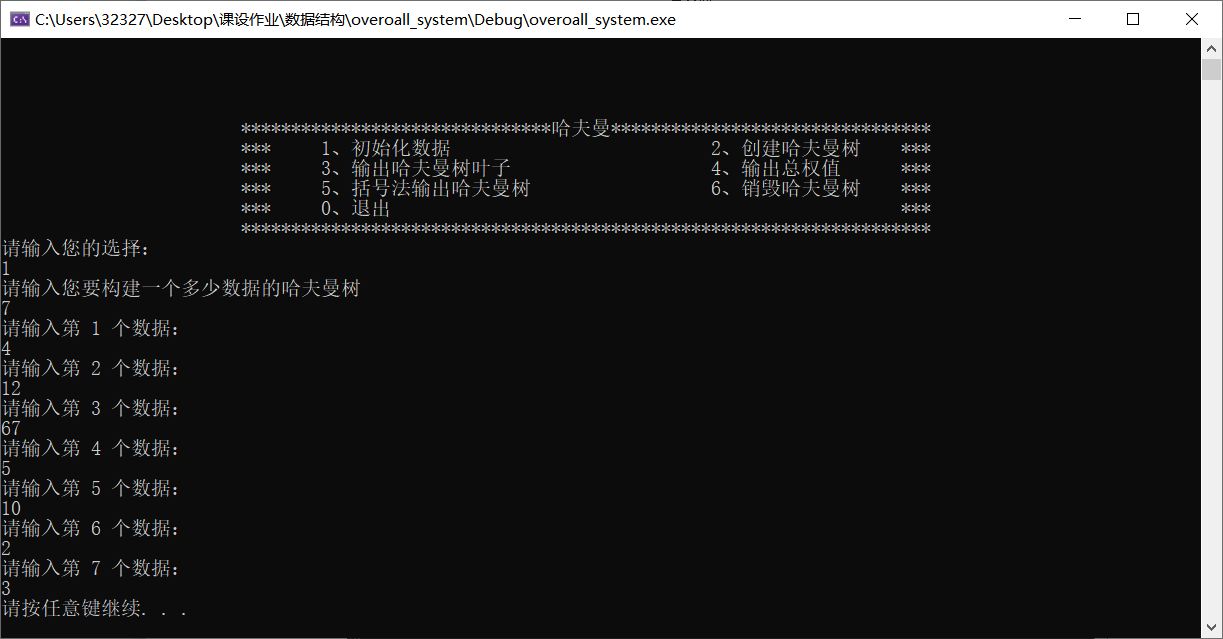
**该函数使用了递归的思路，首先函数参数为当前节点以及当前节点的层数，如果当前节点为叶子节点，则返回树高为1，然后乘上该节点的权重作为该节点的带权路径值，然后回调，直到回到根节点，最终得到树的总带权路径值。**

(5)调试分析或程序测试

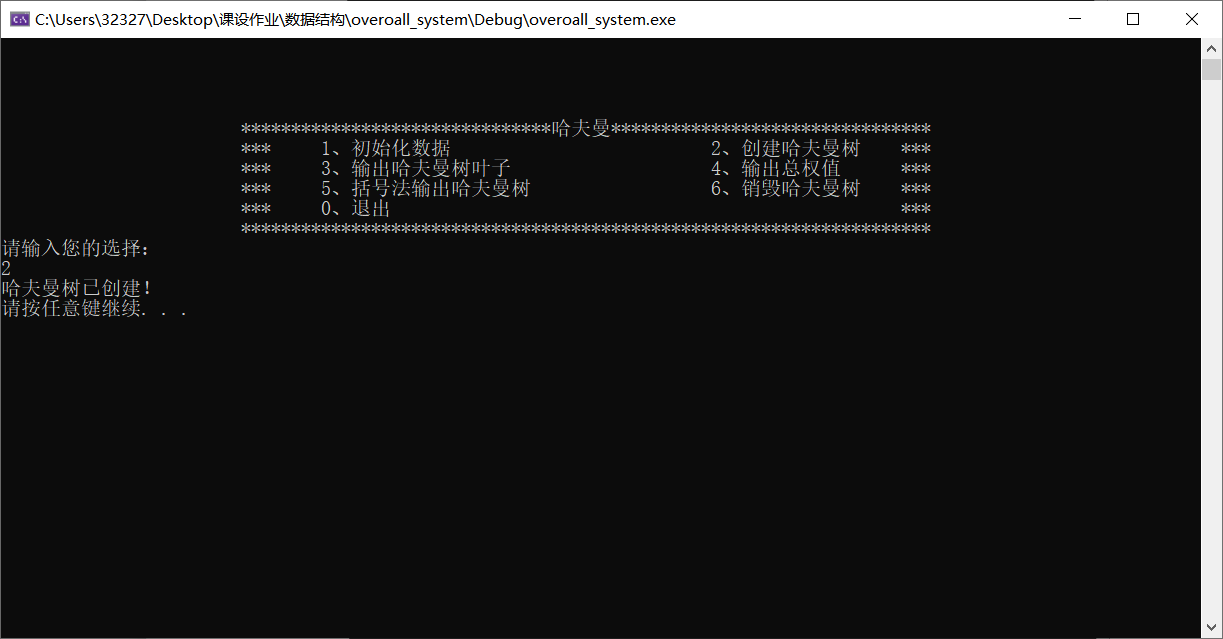
**程序调试：**



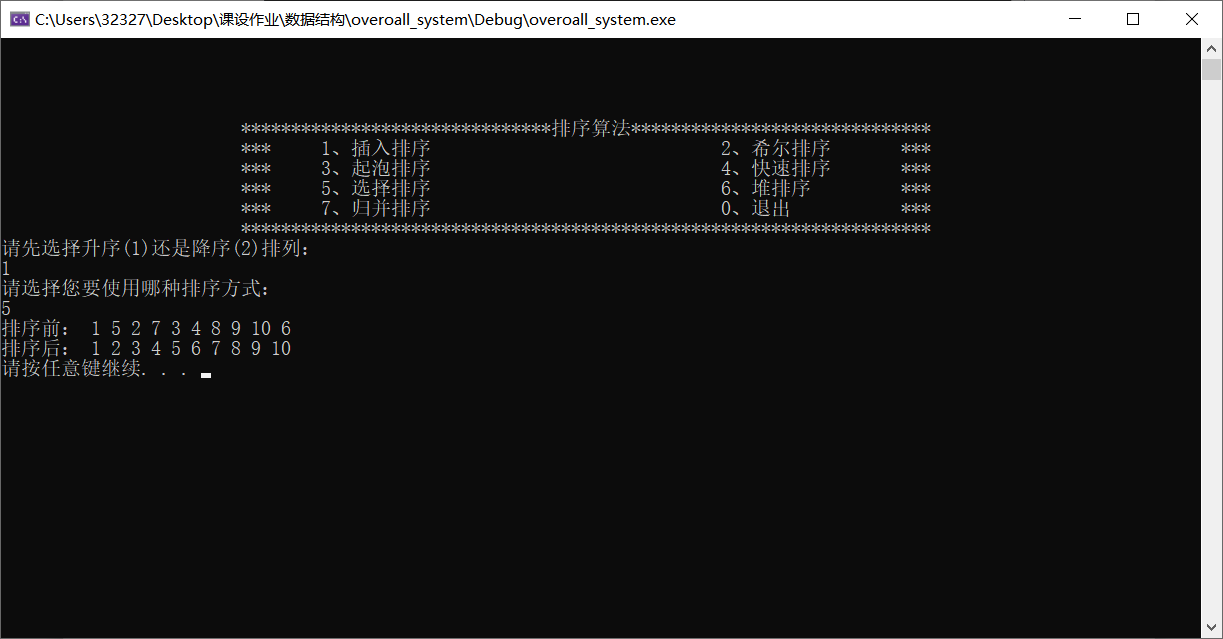
主界面



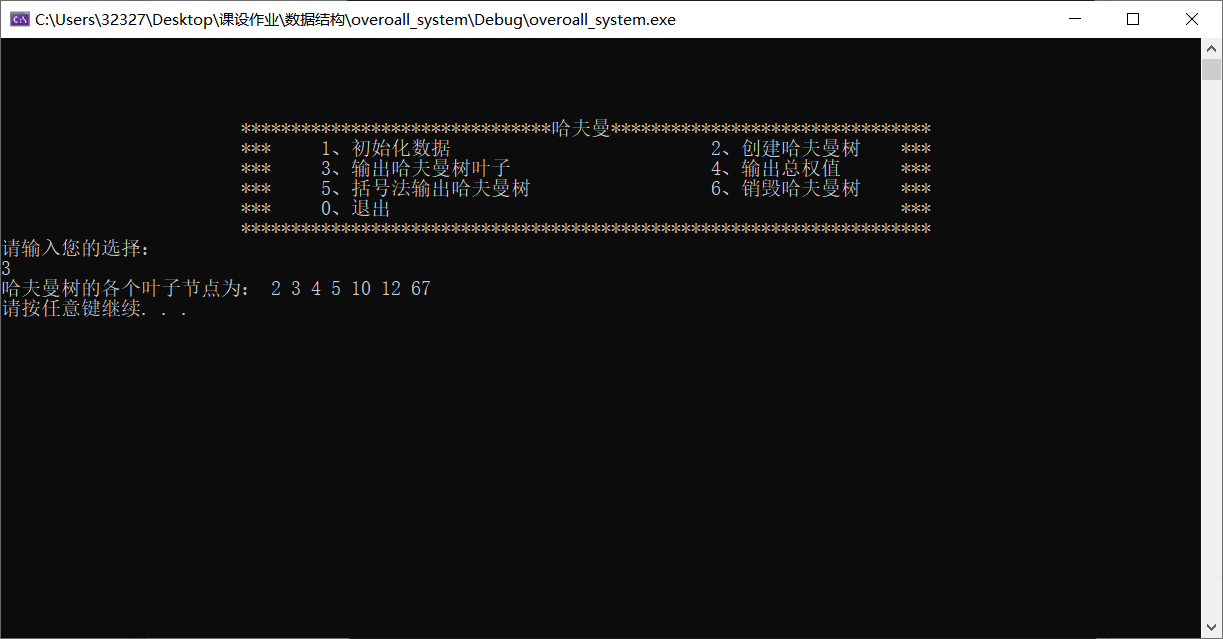
输入数据



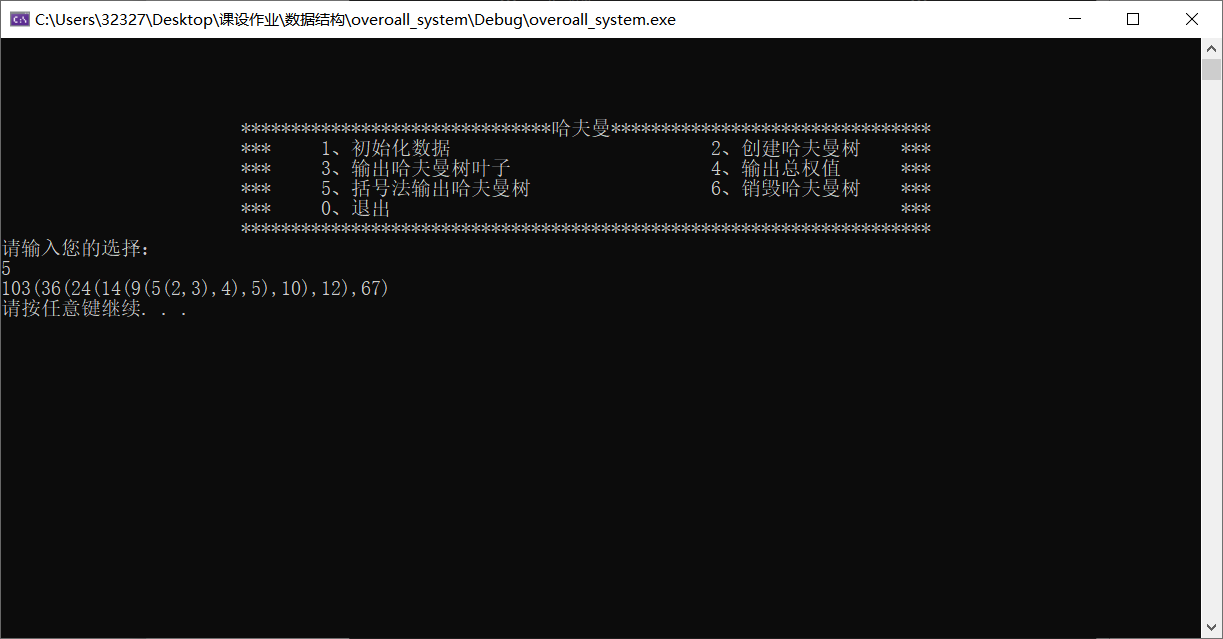
创建哈夫曼树



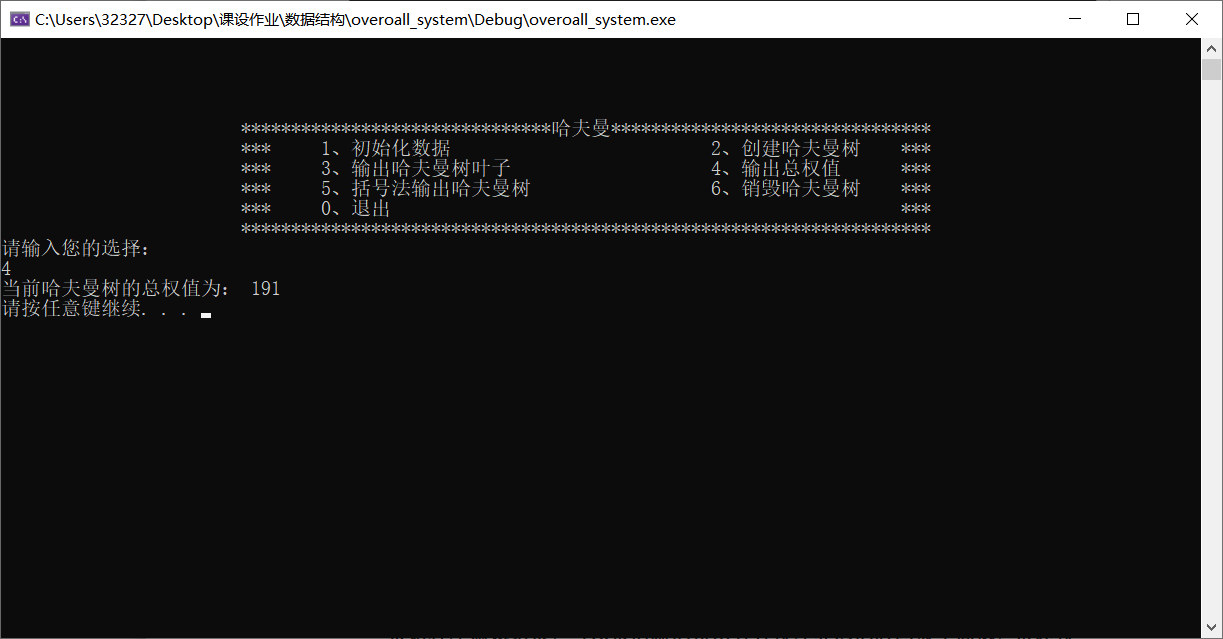
排序输出



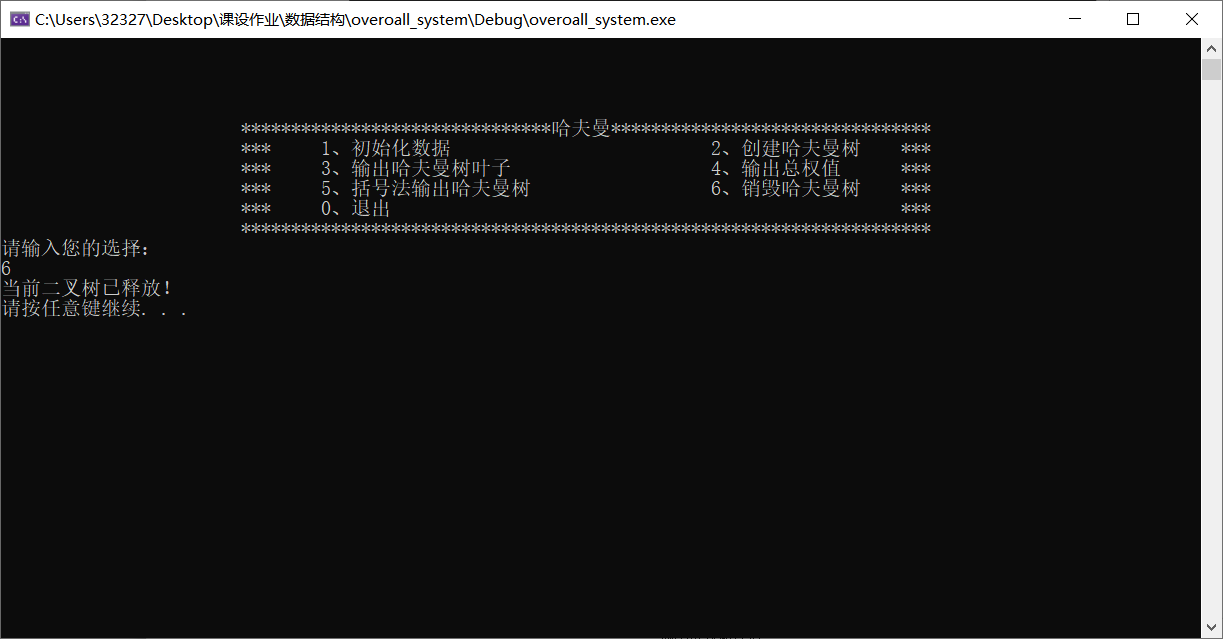
输出哈夫曼树叶子



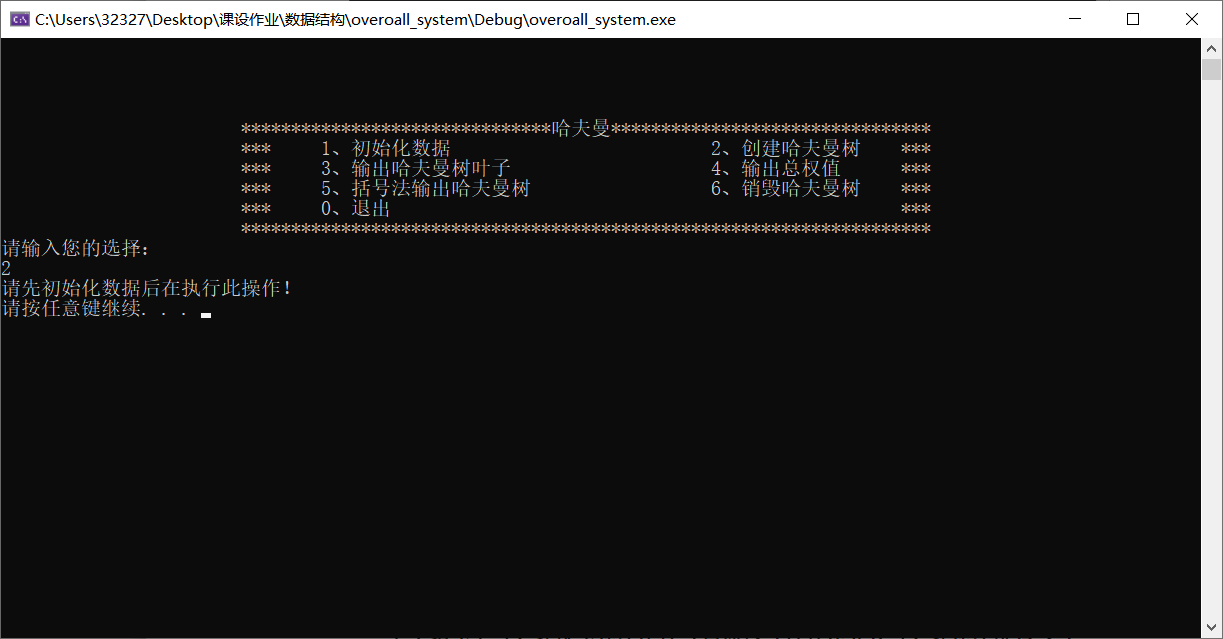
括号法输出哈夫曼树



输出带权路径值



销毁二叉树



输入有误

**根据程序调试结果，可以很清楚的得出本程序能正确的将用户所输入数据创建为一颗最优二叉树即哈夫曼树。本程序还可以通过用户计算带权路径的值来验证该二叉树是否为最佳，使得程序更加科学直观，对于用户的错误输入也有很好的提示功能。**

(6)尚未解决的问题或改进方向

**本程序所实现的创建哈夫曼树的算法时间按复杂度还没有达到最佳，可以通过使用两个队列来存储用户输入数据来实现减少时间复杂度。具体做法为：使用两个队列，从小到大将用户所输入数据升序排序后数组的元素加入队列1，队列2为空，从1，2队列中选出最小的2个数，将选出的2个数的和加到队列2的尾部，如果剩余个数大于等于1，重复执行该步骤；否则，结束循环，哈夫曼树构造完成；**

**在计算哈夫曼树带权路径总值时，程序使用的是递归算法，对时间空间消耗较大，可以改进为迭代算法以减少时间空间消耗。**

**4、订票系统**

(1)对设计任务内容的概述

**（1）问题描述**

**通过此系统可以实现如下功能： 1）录入：可以录入航班情况（数据可以存储在一个数据文件中，数据结构、具体数据自定） 2）查询：可以查询某个航线的情况（如，输入航班号，查询起降时间，起飞抵达城市，航班票价，票价折扣，确定航班是否满仓）； 可以输入起飞抵达城市，查询飞机航班情况； 3）订票：（订票情况可以存在一个数据文件中，结构自己设定）可以订票，如果该航班已经无票，可以提供相关可选择航班； 4）退票： 可退票，退票后修改相关数据文件；客户资料有姓名，证件号，订票数量及航班情况，订单要有编号。 5）修改航班信息：当航班信息改变可以修改航班数据文件**

**（2）要求**

**根据以上功能说明，设计航班信息，订票信息的存储结构，设计程序完成功能；**

(2)需求分析或模块功能描述

**·需求分析：对题目进行分析后了解到，本系统需要实现对航班信息的输入、存储、修改、删除、查询等等功能，以及还应实现乘客信息的相关操作，对于乘客购买机票后，应随机生成一个该机票的订单编号；在用户通过起降地点购买机票时，若当前没有直达的航班，则系统应自动提供其他航班线路。**

**·模块功能描述：根据题目需求，我一共设计了两大模块，模块一负责对航班信息的相关操作，模块二负责对用户信息的相关操作。在模块一中，我实现了对航班信息的展示，查询，录入，修改，清空等操作；在模块二中，我实现了对乘客信息的查询、修改、展示、清空以及订票和退票等操作，将模块一和二分离，可以让用户对本程序的操作更加有条理。**

(3)数据的逻辑结构、存储结构设计

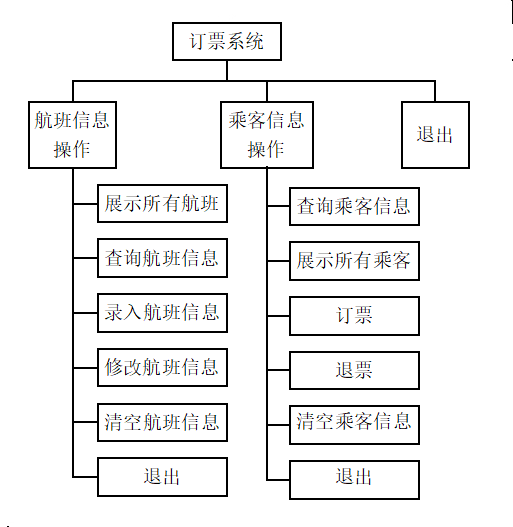
**·本程序的数据的逻辑结构，我选择的是线性结构，这可以让程序对航班以及用户信息的增删改等操作更加简单。**

**·存储本系统中各个航班以及用户信息我选择了使用链式存储结构，原因是，在本程序中，需要实现大量的对各种航班，用户的信息的增删改查询等操作，若用顺序存储结构则会大大增加程序的负担，链式存储则可以很好的避免这一问题。**

**·存储每一个航班以及用户信息我分别定义了一个航班类和一个用户类用于存储相关信息，在此基础上，程序又定义了两个结构体，以用于实现对所有航班信息和用户信息的链式存储。**

(4)主要模块流程图或部分源代码说明

**·主要模块流程图**

****

**·详细说明：**

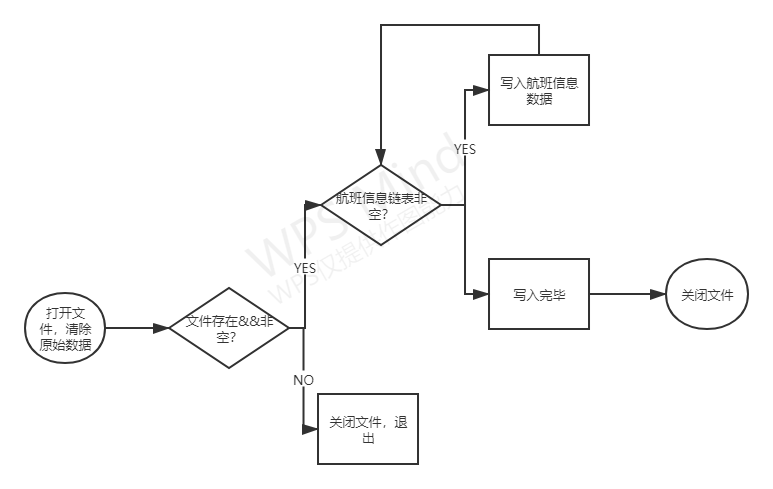
**当用户进入本系统后，需手动选择进入航班信息操作系统或者乘客信息操作系统，若用户选择航班信息操作系统后，系统进入相应界面，在这个界面下，可供用户选择的功能有：展示所有航班，查询航班信息，录入航班信息，修改航班信息，清空航班信息以及退出系统功能；若用户进入用户信息操作系统，系统则会进入对应界面，在这个界面下，用户除了可以选择与航班信息操作类似的功能外，还可以在这个界面下购票和退票。**

**·部分源代码说明：**

**①**

1. **void** savefile\_flight(Flight\_list\_head\* f\_head)
2. {
3. ofstream ofs("flight\_info.txt", ios::out | ios::trunc);
4. **if** (f\_head->head == NULL)
5. {
6. ofs.close();
7. **return**;
8. }
9. Flight\_list\_node\* p = f\_head->head;
10. **while** (p != NULL)
11. {
12. Flight f = p->flight;
13. ofs <<f;
14. p = p->next;
15. }
16. ofs.close();
17. }

**流程图：**

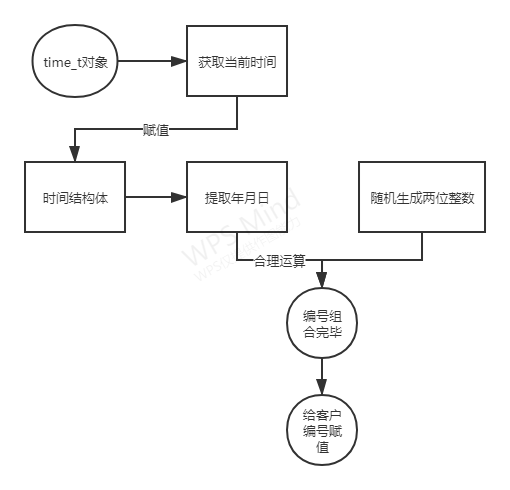
****

**本程序段实现的是对航班信息文件的保存操作，程序所使用的打开文件的方式为ios::trunc，即在每次保存时首先打开文件并将文件原有数据抹去，这样操作的目的是为了方便对文件信息修改后的更新操作。**

**②**

1. **time\_t** nowtime;  //创建一个time\_t对象（一个64位时间变量）
2. **struct** **tm** p; //系统定义的一个时间结构体，里面有年、月、日等属性
3. time(&nowtime);  //获取目前时间
4. localtime\_s(&p, &nowtime);  //将日历时间转成本地时间存放到p中
5. #pragma warning(suppress : 4996)//关闭警告
7. srand((**int**)time(0));
8. **int** time = (p.tm\_year - 100) \* 1000000 + (p.tm\_mon + 1) \* 10000 + (p.tm\_mday) \* 100 + ((random(10000) + random(1000)) % 99+1);
9. c\_no = to\_string(time);

**流程图：**

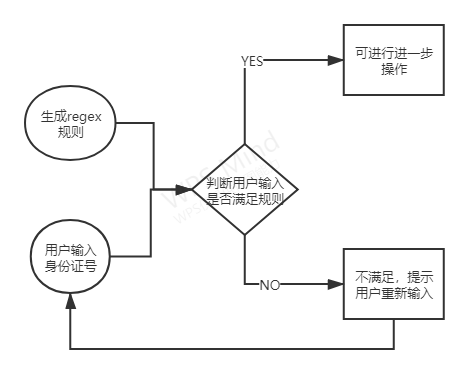
****

**本代码段，实现的是为客户随机生成一个订单编号，在代码中使用到了系统定义的时间结构体，并使用了当前的本地时间给该结构体赋值，然后将当前本地时间的年月日作为订单编号的前六位，最后用到了随机函数随机生成订单编号后两位以达到订单编号的差异性，保证用户的订单编号尽量唯一**

**③**

1. cout << "请输入客户的身份证号：" << endl;
2. cin >> c\_idcard;
3. regex pattern("\\d{17}(x|\\d)");
4. **if** (regex\_match(c\_idcard, pattern))
5. {
6. C.set\_idcard(c\_idcard);
7. }
8. **else**
9. {
10. cout << "您的身份证号输入有误，请重新输入！" << endl;
11. }

**流程图：**

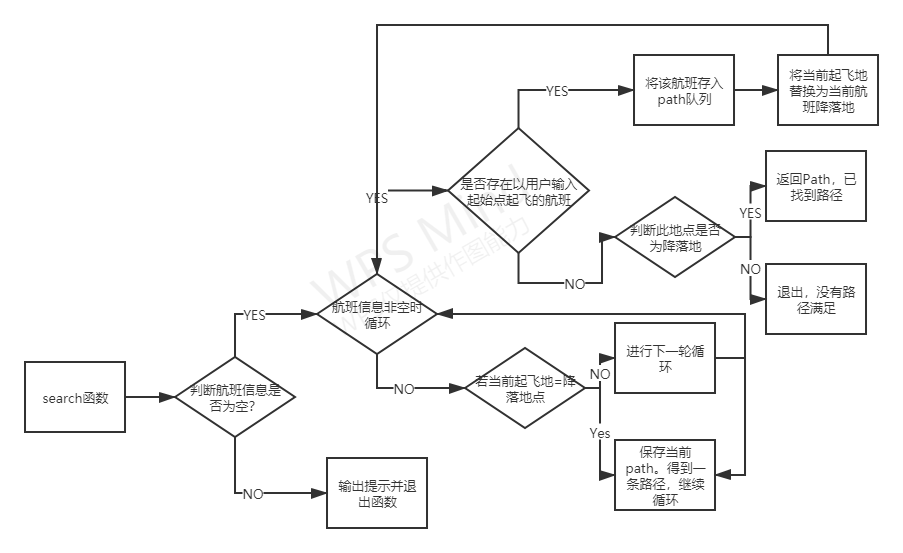
****

**本代码段用于检查用户输入是否合法，使用到了正则表达式，可以简化对用户输入的合法性判断操作，在程序的其他需要输入检验的地方程序也是使用的这种方法进行检验**

**④**

1. **void** search\_path(Flight\_list\_head\* f\_head, string loc\_begin, string loc\_end, queue<queue<Flight>>& Path)
3. **bool** flag = **true**;
4. Flight\_list\_node\* p = f\_head->head;
5. queue<Flight> path;
6. queue<Flight\_list\_node\*> start;
7. **while** (p != NULL)
8. {
9. **if** (p->flight.get\_begin\_loc() == loc\_begin)
10. {
11. start.push(p);
12. }
13. p = p->next;
14. }
15. **while** (!start.empty())
16. {
17. flag = **true**;
18. p = f\_head->head;
19. Flight\_list\_node\* temp = start.front();
20. start.pop();
21. loc\_begin = temp->flight.get\_end\_loc();
22. path.push(temp->flight);
23. **while** (flag)
24. {
25. **while** (p != NULL && p->flight.get\_begin\_loc() != loc\_begin)
26. {
27. p = p->next;
28. }
29. **if** (p == NULL)
30. {
31. **while** (!path.empty())
32. {
33. path.pop();
34. }
35. **break**;
36. flag = **false**;
37. }
38. **else**
39. {
40. path.push(p->flight);
41. loc\_begin = p->flight.get\_end\_loc();
42. p = f\_head->head;
43. }
44. **if** (loc\_begin == loc\_end)
45. {
46. queue<Flight> p;
47. **while** (!path.empty())
48. {
49. p.push(path.front());
50. path.pop();
51. }
52. Path.push(p);
53. flag = **false**;
54. **break**;
55. }
56. }
57. }

**流程图：**

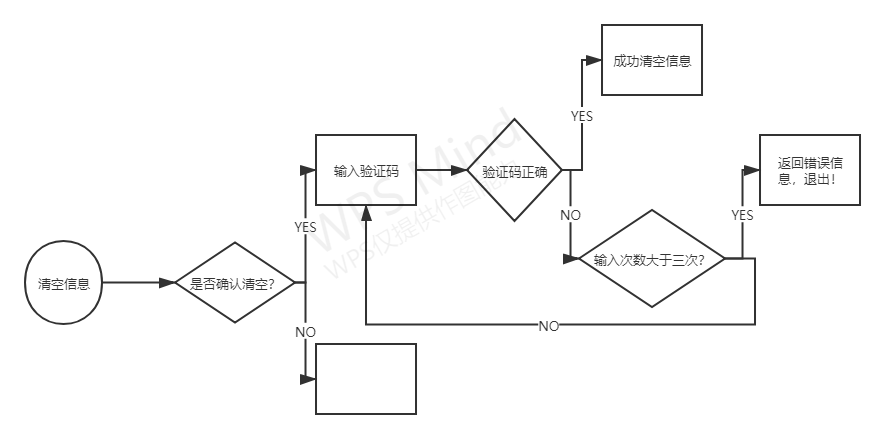
****

**本代码段用于当乘客所购航班无法达到要求时系统自动筛选符合要求的航班路径。代码中使用到了queue队列容器，用于存储符合要求的航班路径，在函数执行过程中，程序会不断迭代航班的起始地点，找到所有满足从乘客起始地出发并能最终到达乘客所要去的终点的所有航班，并保存至函数所传入的path队列中，最终得到一个包含了所有可选路径的queue队列供乘客选择。**

**⑤**

1. **void** clear\_client\_info()
2. {
3. cout << "\*\*\*\*\*\*\*\*\*\*\*敏感操作\*\*\*\*\*\*\*\*\*\*\*" << endl;
4. cout << "您确定要清空乘客信息吗？（Y/N）" << endl;
5. **char** choice;
6. B:  cin >> choice;
7. **int** count = 1;
8. A:  **if** (choice == 'Y' || choice == 'y')
9. {
10. **int** randnum;
11. srand((**int**)time(NULL));
12. randnum = (random(1000000)+100000)\*(random(100)+10)%1000000;
13. cout << "请输入验证码再次确认  " << randnum << endl;
14. **int** input;
15. cin >> input;
16. **if** (input == randnum)
17. {
18. cout << "当前乘客信息已清空！" << endl;
19. ofstream ofs("client\_info.txt", ios::out | ios::trunc);
20. ofs.close();
21. }
22. **else**
23. {
24. cout << "验证码错误，请再次输入（还剩余 " << (3 - count) << " 次输入机会！）";
25. count++;
26. **if** (count == 4)
27. {
28. cout << "\n您已多次输错验证码，已退出！" << endl;
29. **return**;
30. }
31. **else**
32. {
33. **goto** A;
34. }
35. }
36. }
37. **else** **if** (choice == 'N' || choice == 'n')
38. {
39. cout << "已撤销清空操作！" << endl;
40. **return**;
41. }
42. **else**
43. {
44. cout << "您的输入有误，请重新输入！" << endl;
45. **goto** B;
46. }
47. }

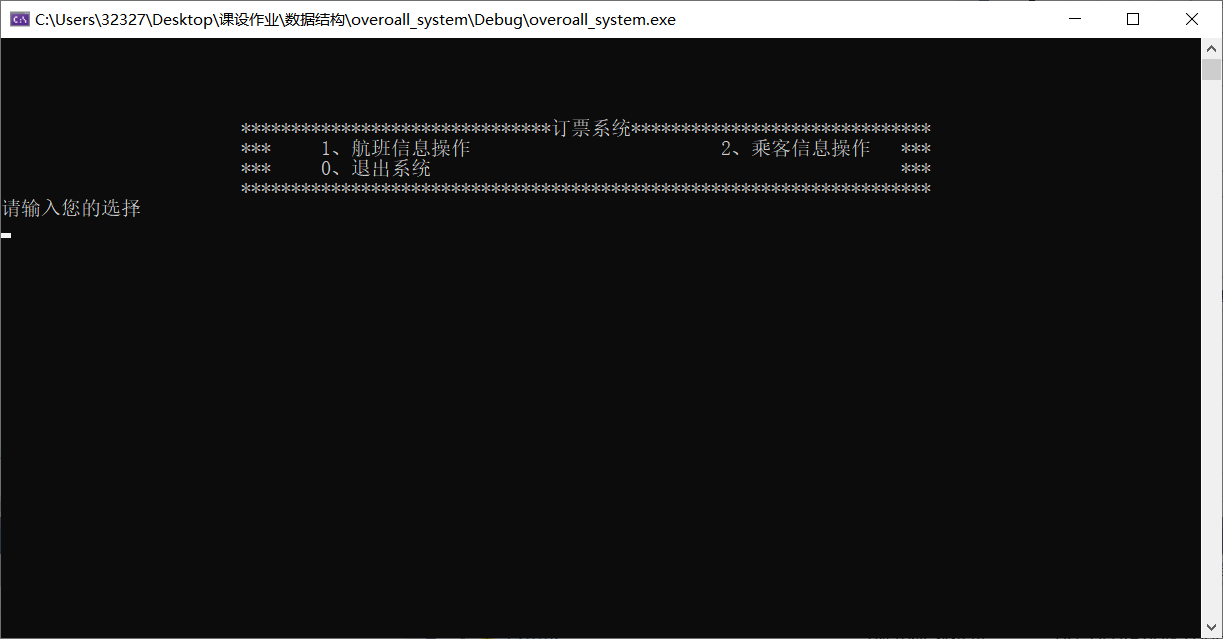
**流程图：**

****

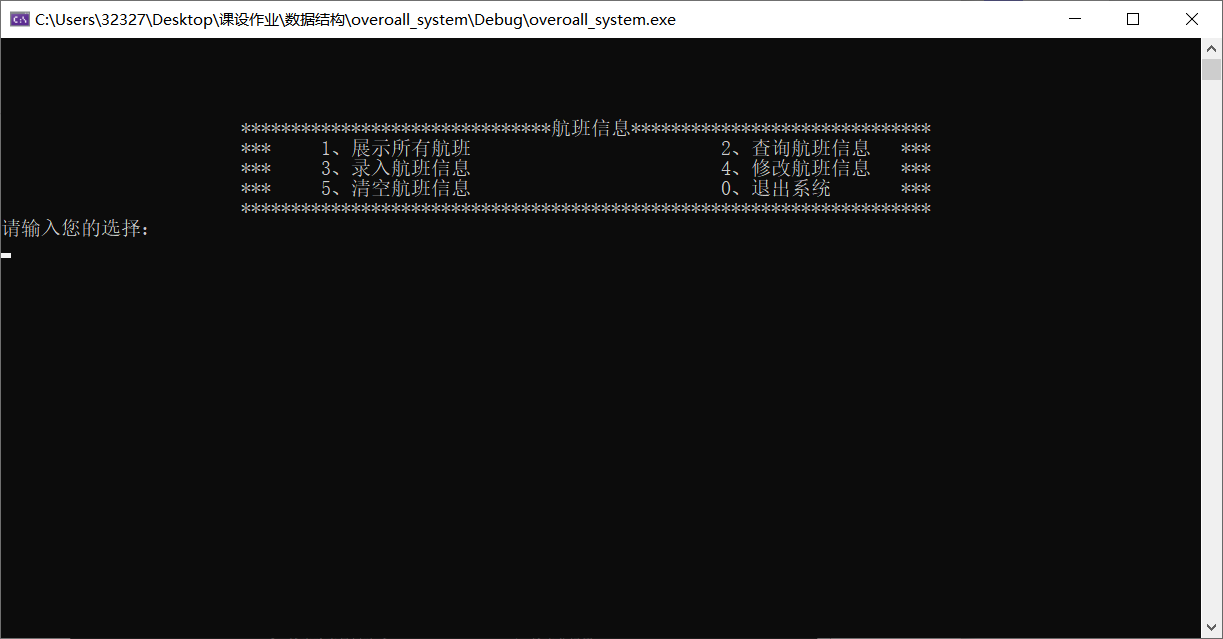
**本程序段用于在用户选择清空信息选项时来保护文件信息不会被用户误删，增加了两个验证环节。首先用户需选择确认清空，之后还需正确输入验证码才可以成功将文件信息清空，大大增加了程序的安全性。**

(5)调试分析或程序测试

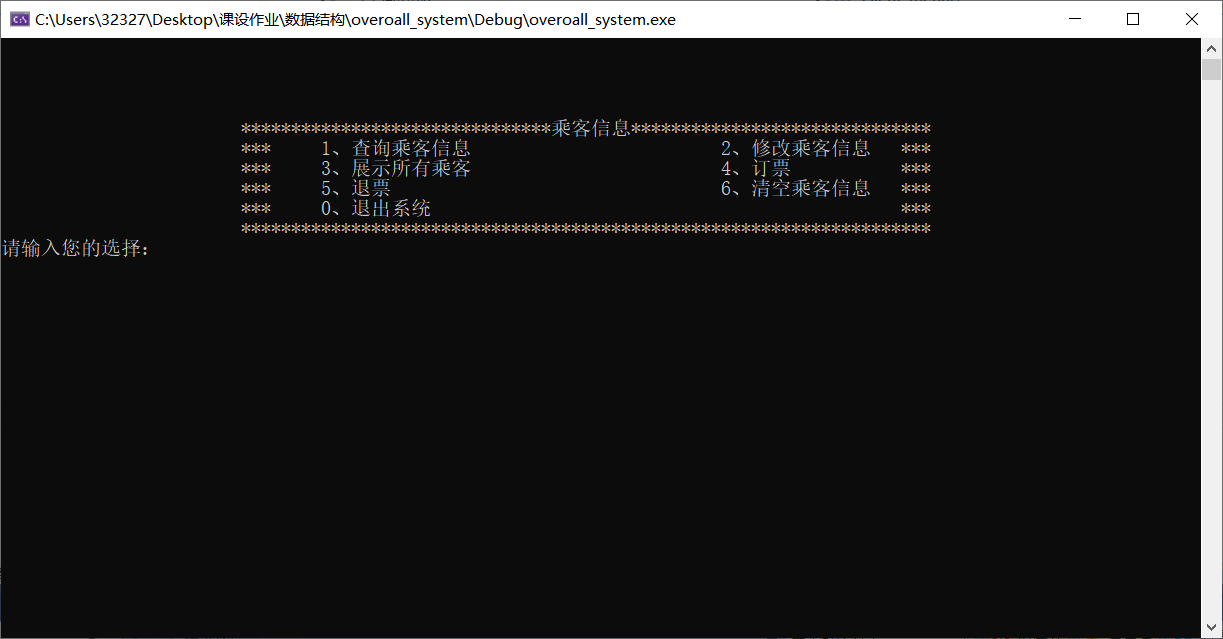
**程序调试：**



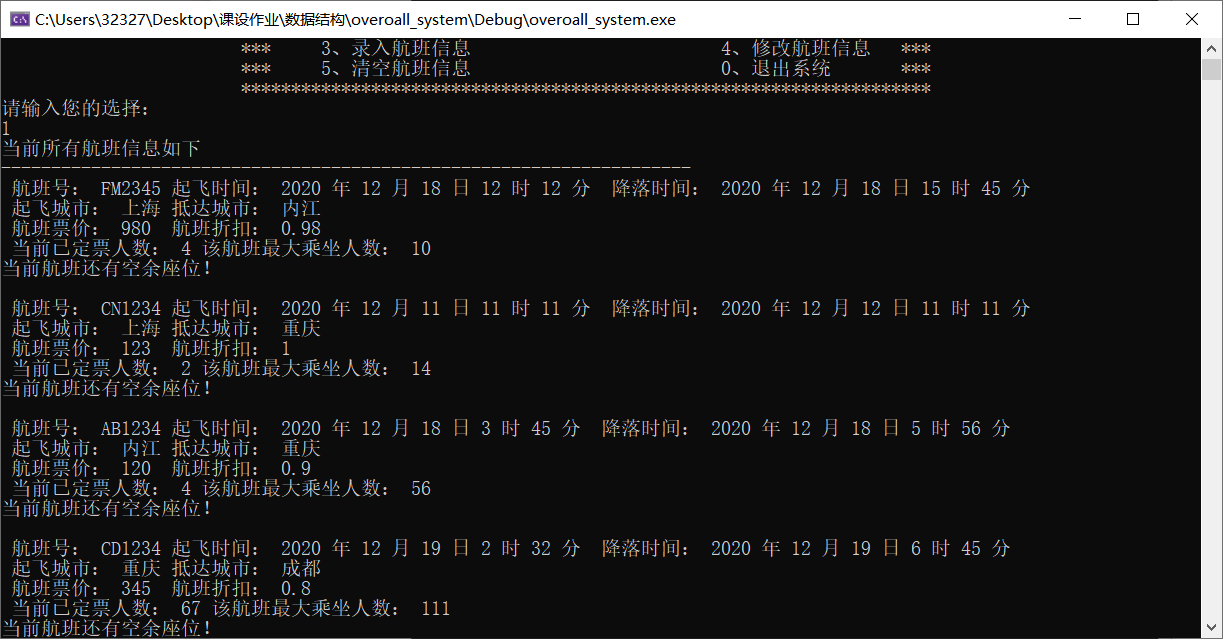
主界面



航班信息操作系统界面



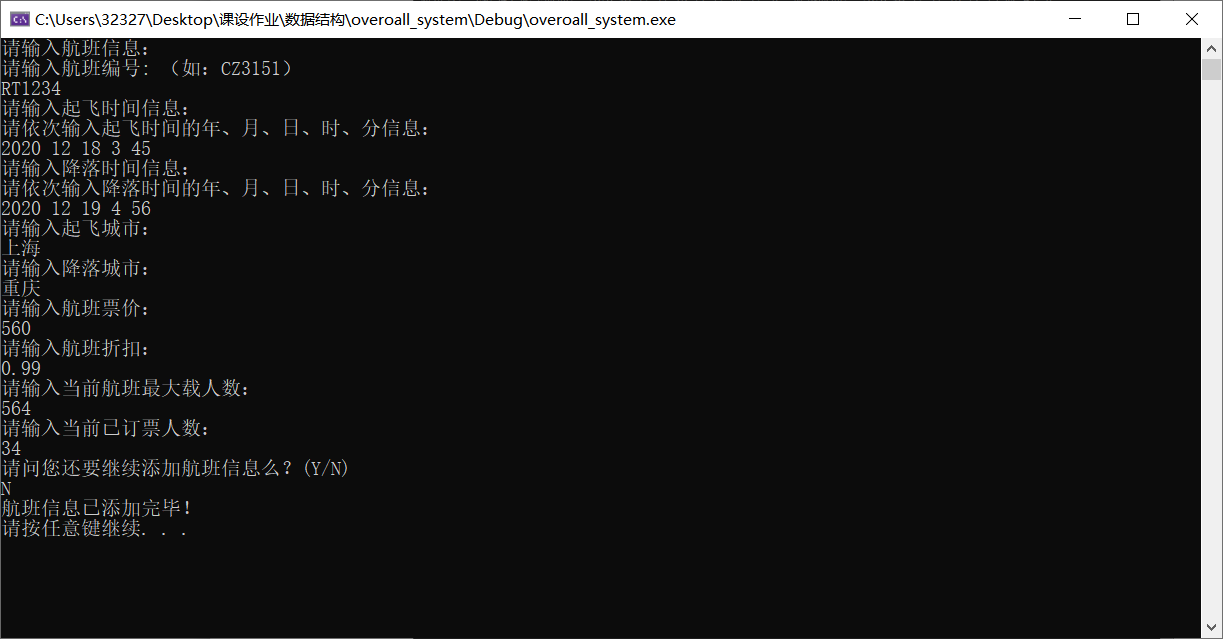
乘客信息操作系统界面



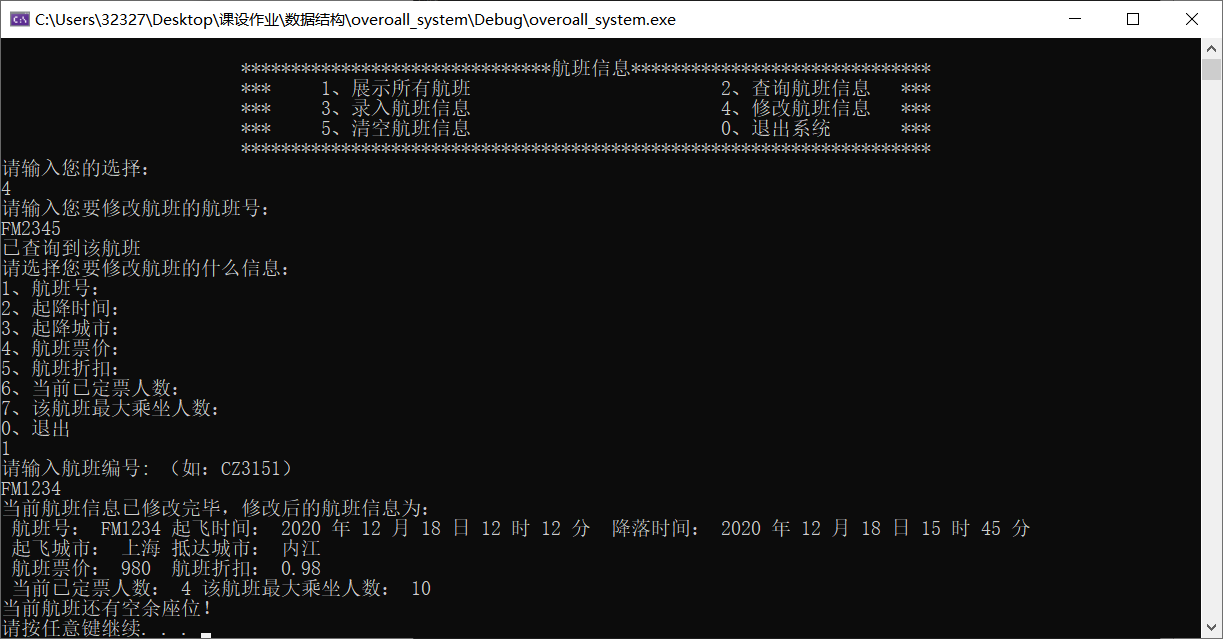
展示所有航班信息



查询航班信息



录入航班信息



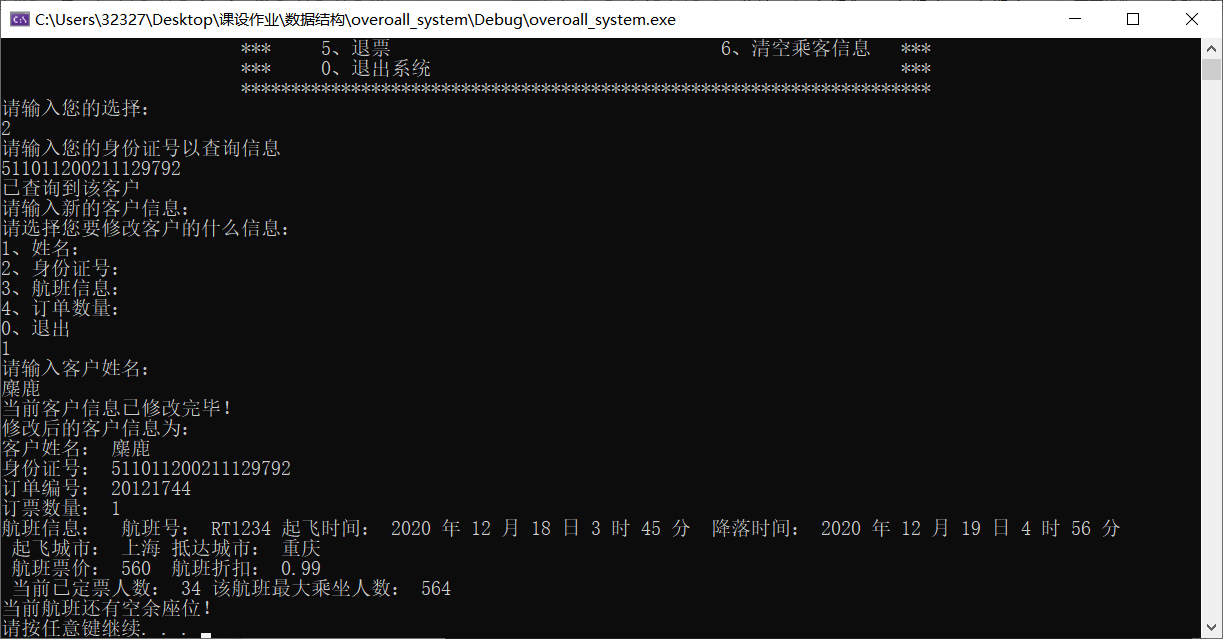
修改航班信息



清空航班信息



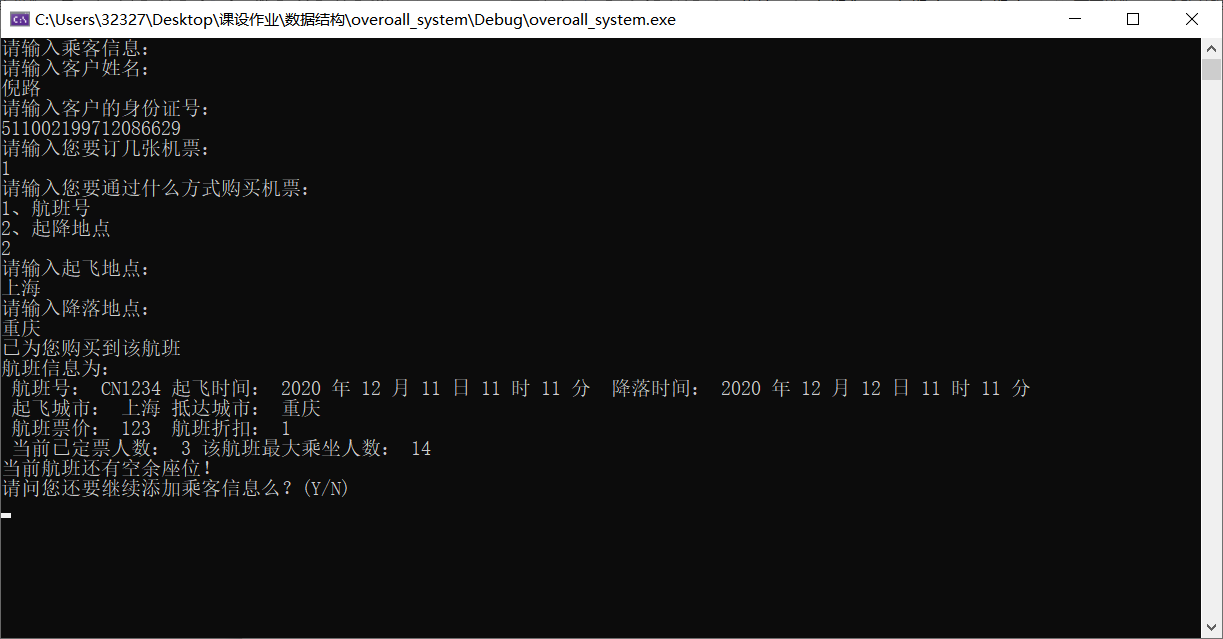
查询乘客信息



修改用户信息



展示用户信息



订票

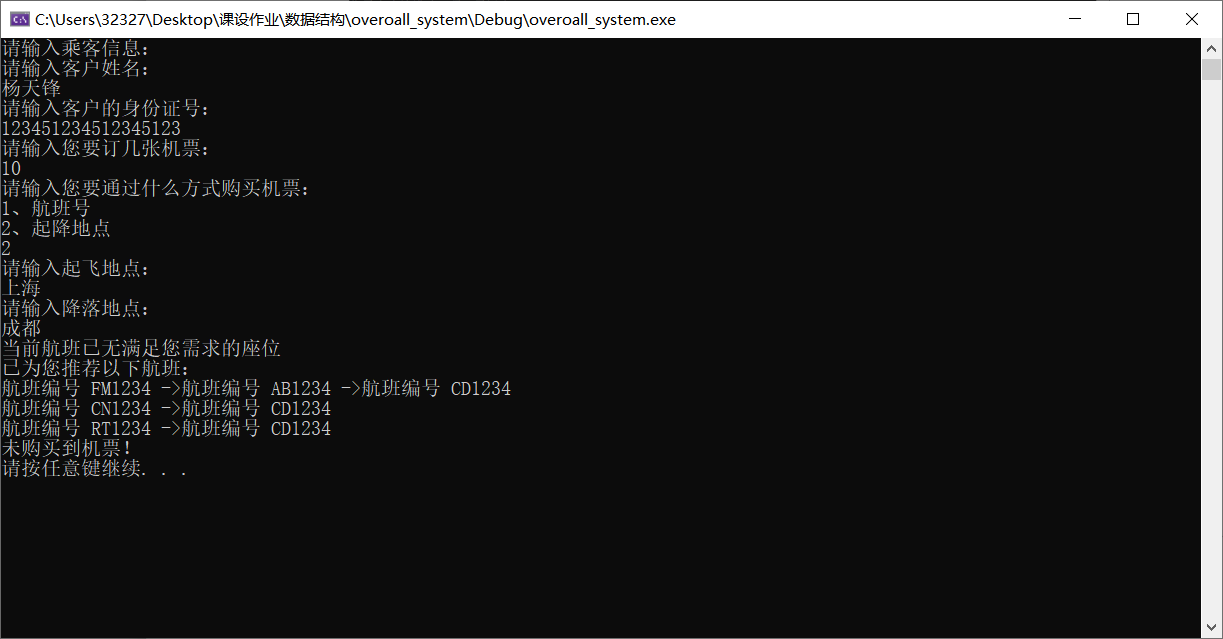


退票



清空用户信息

**其他部分功能展示：**



推荐航班



输入合法性判断

**根据程序调试结果，程序可以很好的完成题目所要求的功能，在界面的排版上也对功能做出了两大分类，方便用户的操作，并且对于用户输入的数据也有很友好的合法性检测功能，操作难度上降低了很多。**

(6)尚未解决的问题或改进方向

**本程序的航班信息虽然已经比较全面，但是和真实的航班信息相比还是有所差异，比如程序为了方便操作将，每个航班的起降时间定在某一天的某个时分，因此当乘客购买车票时无法即时的对航班信息的时间更改，为此，改进方案如下可以将航班信息的日期将具体的某一天的某个时分仅仅更改为某一个时分，具体哪个日期应由乘客来手动输入；在乘客所买航班不符合乘客要求后程序提供航班方案时，程序只是简单的给出了可选航班的航班号，不方便乘客挑选路径，所以可以改进为将航班的航班号，起降地点以及时间都输出出来方便乘客挑选合适路径；在展示乘客信息时没有做安全检测，这可能会导致乘客隐私泄露，所以可以改进为，设置一个管理员密码，只有管理员密码输入正确后才可成功浏览到乘客信息。**

1. **结论及体会**

**通过此次课程设计，使我更加扎实的掌握了有关链表和结构体以及类方面的知识，在设计过程中，中虽然遇到了一些问题，但经过一次又一次的思考，一遍又一遍的检查终于找出了原因。所在，也暴露出了前期我在这方面的知识欠缺和经验不足。实践出真知，通过亲自动手制作，使我们掌握的知识不再是纸上谈兵。这次课程设计终于顺利完成了，在今后社会的发展和学习实践过程中，一定要不懈努力，不能遇到问题就想到要退缩，一定要不厌其烦的发现问题所在，然后一一进行解决，只有这样，才能成功的做成想做的事，才能在今后的道路上劈荆斩棘，而不是知难而退，那样永远不可能收课程设计诚然是一门专业课，给我很多专业知识以及专业技能上的提升，同时又是一门讲道课，一门辩思课，给了我许多道，给了我很多思，给了我莫大的空间。同时，设计让我感触很深。使我对抽象的理论有了具体的认识。通过这次课程设计，我对建立一个工程有了一个新的认识，首先是想清楚这个系统的流程，将框架搭建起来后，再起写一些细节方面的东西，千万不能稀里糊涂的写，到最后没有用处，白白浪费时间。**

**我认为此次课设不仅培养了独立思考、动手操作的能力，在各种其它能力上也都有了提高。同时也希望今后能够多花时间去思考，去动手写一些代码。**

**附录1：参考文献**

[1] 《数据结构教程（第5版）》，李春葆，清华大学出版社，2017

[2]《数据结构》，杨剑，清华大学出版社，2011

[3]《数据结构(C语言版)》，严蔚敏 吴伟民，清华大学出版社，1997

[4]《Data Structures Using C数据结构（C语言版）》，R Krishnamoorthy、G Indirani Kumaravel，清华大学出版社，2009-9

[5]《C++数据结构与程序设计 （美）Robert L.Kruse/Alexander J.Ryba著/钱丽萍译》，

清华大学出版社，2004

[6]《计算机算法设计与分析（第2版）》，王晓东， 电子工业出版社， 2004

**附录2：部分源代码清单**

**一：排序算法源码：**

**·插入排序：**

1. **void** Insert\_sort(**int** a[], **int** len)
2. {
3. **int** j = 0;
4. **int** temp = 0;
5. **for** (**int** i = 2; i <= len; i++)
6. {
7. **if** (a[i] < a[i - 1])
8. {
9. temp = a[i];
10. **int** j = i - 1;
11. **while** (j >= 1 && temp < a[j])
12. {
13. a[j + 1] = a[j];
14. j--;
15. }
16. a[j + 1] = temp;
17. }
18. }
19. }

**·选择排序**

1. **void** select\_sort(**int**\* a, **int** n)
3. **for** (**int** i = 1; i < n; i++)
4. {
5. **int** k = i;
6. **for** (**int** j = i + 1; j <= n; j++)
7. {
8. **if** (a[k] > a[j])
9. {
10. k = j;
11. }
12. }
13. **if** (k != i)
14. {
15. **int** temp = a[k];
16. a[k] = a[i];
17. a[i] = temp;
18. }
19. }

**·希尔排序**

1. **void** select\_sort(**int**\* a, **int** n)
3. **for** (**int** i = 1; i < n; i++)
4. {
5. **int** k = i;
6. **for** (**int** j = i + 1; j <= n; j++)
7. {
8. **if** (a[k] > a[j])
9. {
10. k = j;
11. }
12. }
13. **if** (k != i)
14. {
15. **int** temp = a[k];
16. a[k] = a[i];
17. a[i] = temp;
18. }
19. }

**·快速排序**

1. **void** quick\_sort(**int**\* a, **int** low, **int** high)
3. **if** (low >= high)
4. **return**;
5. **int** left = low, right = high;
6. **int** p = a[high];
7. **while** (left < right)
8. {
9. **while** (left < right && a[left] <= p)
10. {
11. left++;
12. }
13. a[right] = a[left];
14. **while** (left < right && a[right] >= p)
15. {
16. right--;
17. }
18. a[left] = a[right];
19. }
20. **if** (left >= right)
21. {
22. a[left] = p;
23. }
24. quick\_sort(a, low, right - 1);
25. quick\_sort(a, right + 1, high);

**·冒泡排序**

1. **void** pop\_sort(**int**\* a, **int** n)
2. {
3. **for** (**int** i = 1; i <= n; i++)
4. {
5. **for** (**int** j = n; j > i; j--)
6. {
7. **if** (a[j] < a[j - 1])
8. {
9. **int** temp = a[j];
10. a[j] = a[j - 1];
11. a[j - 1] = temp;
12. }
13. }
14. }
15. }

**·归并排序**

1. **void** merge(**int** a[], **int** low, **int** mid, **int** high)
2. {
3. **int** N = high - low + 1;
4. **int**\* b = **new** **int**[N];
5. **int** left = low;
6. **int** right = mid + 1;
7. **int** i = 0;
8. **while** (left <= mid && right <= high)
9. {
10. b[i++] = (a[left] < a[right]) ? a[right++] : a[left++];
11. }
12. **while** (left <= mid)
13. {
14. b[i++] = a[left++];
15. }
16. **while** (right <= high)
17. {
18. b[i++] = a[right++];
19. }
20. **for** (**int** i = 0; i < N; i++)
21. {
22. a[low + i] = b[i];
23. }
24. **delete**[] b;
25. }
26. **void** mergesort(**int** a[], **int** low, **int** high)
27. {
28. **if** (low < high)
29. {
30. **int** mid = (low + high) / 2;
31. mergesort(a, low, mid);
32. mergesort(a, mid + 1, high);
33. merge\_reverse(a, low, mid, high);
34. }
36. }

**·堆排序**

1. **void** shift(**int**\* a, **int** low, **int** n)
2. {
3. **int** i = low, j = 2 \* i;
4. **int** temp = a[low];
5. **while** (j <= n)
6. {
7. **if** (j < n && a[j] < a[j + 1])
8. {
9. j++;
10. }
11. **if** (temp < a[j])
12. {
13. a[i] = a[j];
14. i = j;
15. j \*= 2;
16. }
17. **else**
18. {
19. **break**;
20. }
21. }
22. a[i] = temp;
23. }
24. **void** HeapSort(**int**\* a, **int** n)
25. {
26. **for** (**int** i = n / 2; i >= 1; i--)
27. {
28. shift(a, i, n);
29. }
30. **for** (**int** i = n; i > 1; i--)
31. {
32. swap(a[i], a[1]);
33. shift(a, 1, i - 1);
34. }
36. }

**二、哈夫曼树源码：**

**·Huffman.h**

1. #include<iostream>
2. #include<algorithm>
3. **using** **namespace** std;
4. #define MAXSIZE 100
5. **struct** BTNode {
6. **int** data;
7. BTNode\* Lchild, \* Rchild;
8. };
9. **void** sort(**int**\* a, **int** n);
10. **void** Init(**int**\* a, BTNode\*\* B, **int**& n);
11. **void** merge(BTNode\*\* B, BTNode& min1, BTNode& min2, **int** n);
12. **void** creat\_HTree(BTNode\*\* B, **int** a[], **int** n);
13. **void** dis\_Tree(BTNode\* B);
14. **void** Dis\_Tree(BTNode\*& B);
15. **int** get\_weight(BTNode\* B, **int** layer);
16. **void** Get\_weight(BTNode\* B);
17. **void** free\_BTree(BTNode\*& B);
18. **void** free\_BTREE(BTNode\*& B);
19. **void** dis\_BT(BTNode\* B);
20. **void** Dis\_HTree(BTNode\* B);
21. **void** dis\_Huff\_menu();
22. **void** Huff\_main\_func();

**·Huffman.cpp**

1. #include"Huffman.h"
2. **int**\* A = **new** **int**[MAXSIZE];
3. **bool**\* flag = **new** **bool**[MAXSIZE];
4. **void** sort(**int**\* a, **int** n)
5. {
6. **for** (**int** i = 0; i < n; i++)
7. {
8. **for** (**int** j = n - 1; j > i; j--)
9. {
10. **if** (a[j] < a[j - 1])
11. {
12. **int** temp = a[j];
13. a[j] = a[j - 1];
14. a[j - 1] = temp;
15. }
16. }
17. }
18. }
19. **void** Init(**int**\* a, BTNode\*\* B, **int**& n)
20. {
21. cout << "请输入您要构建一个多少数据的哈夫曼树" << endl;
22. cin >> n;
23. **if** (n <= 0 || n > MAXSIZE)
24. {
25. cout << "您输入的长度有误！" << endl;
26. **return**;
27. }
28. **for** (**int** i = 0; i < n; i++)
29. {
30. cout << "请输入第 " << (i + 1) << " 个数据：" << endl;
31. cin >> a[i];
32. flag[i] = **false**;
33. }
34. sort(a, n);
35. **for** (**int** i = 0; i < n; i++)
36. {
37. B[i] = **new** BTNode;
38. B[i]->data = a[i];
39. B[i]->Lchild = B[i]->Rchild = NULL;
40. }
41. }
42. **void** merge(BTNode\*\* B, BTNode& min1, BTNode& min2, **int** n)
43. {
44. BTNode\* temp = **new** BTNode;
45. temp->data = min1.data + min2.data;
46. temp->Lchild = &min1;
47. temp->Rchild = &min2;
48. B[n] = temp;
49. }
50. **void** creat\_HTree(BTNode\*\* B, **int** a[], **int** n)
51. {
52. **int** min1, min2;
53. **for** (**int** i = 0; i < n - 1; i++)
54. {
55. **int** k = i;
56. **while** (flag[k] != **false**)
57. {
58. k++;
59. }
60. **int** j = i + 1;
61. **while** (flag[j] != **false**)
62. {
63. j++;
64. }
65. min1 = B[k]->data;
66. min2 = B[j]->data;
67. **while** (min1 > min2)
68. {
69. min1 = min2;
70. **if** ((j + 1) < n && min2 > B[(j + 1)]->data)
71. {
72. min2 = B[++j]->data;
73. }
74. **else**
75. {
76. min2 = B[k]->data;
77. }
78. }
79. flag[k] = **true**;
81. merge(B, \*B[k], \*B[j], j);
82. }
83. }
85. **void** dis\_Tree(BTNode\* B)
86. {
87. **if** (B == NULL)
88. {
89. **return**;
90. }
91. **if** (B->Lchild == NULL && B->Rchild == NULL)
92. {
93. cout << B->data << " ";
94. }
95. **if** (B->Lchild != NULL)
96. {
97. dis\_Tree(B->Lchild);
98. }
99. **if** (B->Rchild != NULL)
100. {
101. dis\_Tree(B->Rchild);
102. }
103. }
104. **void** Dis\_Tree(BTNode\*& B)
105. {
106. **if** (B == NULL)
107. {
108. cout << "当前哈夫曼树未初始化，请先执行1、2操作！" << endl;
109. **return**;
110. }
111. **if** (B->Lchild == NULL && B->Rchild == NULL)
112. {
113. cout << "哈夫曼树未创建，请先创建哈夫曼树(操作2)！" << endl;
114. **return**;
115. }
116. cout << "哈夫曼树的各个叶子节点为： ";
117. dis\_Tree(B);
118. cout << endl;
120. }
121. **int** get\_weight(BTNode\* B, **int** layer)
122. {
123. **if** (B == NULL)
124. {
125. **return** 1;
126. }
127. **if** (B->Lchild == NULL && B->Rchild == NULL)
128. {
129. **return** B->data \* layer;
130. }
131. **if** (B->Lchild != NULL && B->Rchild != NULL)
132. {
133. **return** get\_weight(B->Lchild, layer + 1) + get\_weight(B->Rchild, layer + 1);
134. }
135. **else** **if** (B->Lchild != NULL)
136. {
137. **return**  get\_weight(B->Lchild, layer + 1);
138. }
139. **else** **if** (B->Rchild != NULL)
140. {
141. **return** layer \* get\_weight(B->Rchild, layer + 1);
142. }
143. }
144. **void** Get\_weight(BTNode\* B)
145. {
146. **if** (B == NULL)
147. {
148. cout << "当前哈夫曼树未初始化，请先执行1、2操作！" << endl;
149. **return**;
150. }
151. **if** (B->Lchild == NULL && B->Rchild == NULL)
152. {
153. cout << "哈夫曼树未创建，请先创建哈夫曼树(操作2)！" << endl;
154. **return**;
155. }
156. **int** layer = 0;
157. **int** weight = get\_weight(B, layer);
158. cout << "当前哈夫曼树的总权值为： " << weight << endl;
159. }
161. **void** free\_BTree(BTNode\*& B)
162. {
163. **if** (B != NULL)
164. {
165. free\_BTree(B->Lchild);
166. free\_BTree(B->Rchild);
167. **delete** B;
168. B = NULL;
169. }
170. }
171. **void** free\_BTREE(BTNode\*& B)
172. {
173. **if** (B == NULL)
174. {
175. cout << "当前哈夫曼树未初始化，无需销毁！" << endl;
176. **return**;
177. }
178. **if** (B->Lchild == NULL && B->Rchild == NULL)
179. {
180. cout << "哈夫曼树未创建，无需销毁！" << endl;
181. **return**;
182. }
183. free\_BTree(B);
184. cout << "当前二叉树已释放！" << endl;
186. }
187. **void** dis\_BT(BTNode\* B)
188. {
190. **if** (B != NULL)
191. {
192. cout << B->data;
193. **if** (B->Lchild != NULL || B->Rchild != NULL)
194. {
195. cout << "(";
196. dis\_BT(B->Lchild);
197. **if** (B->Rchild != NULL)
198. {
199. cout << ",";
200. }
201. dis\_BT(B->Rchild);
202. cout << ")";
203. }
204. }
205. }
206. **void** Dis\_HTree(BTNode\* B)
207. {
208. **if** (B == NULL)
209. {
210. cout << "当前哈夫曼树未初始化，请先执行1、2操作！" << endl;
211. **return**;
212. }
213. **if** (B->Lchild == NULL && B->Rchild == NULL)
214. {
215. cout << "哈夫曼树未创建，请先创建哈夫曼树(操作2)！" << endl;
216. **return**;
217. }
218. dis\_BT(B);
219. cout << endl;
220. }
221. **void** dis\_Huff\_menu()
222. {
223. cout << endl << endl << endl << endl;
224. cout << "\t\t\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*哈夫曼\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;
225. cout << "\t\t\t\*\*\*\t1、初始化数据\t\t\t       2、创建哈夫曼树\t  \*\*\*" << endl;
226. cout << "\t\t\t\*\*\*\t3、输出哈夫曼树叶子\t\t       4、输出总权值\t  \*\*\*" << endl;
227. cout << "\t\t\t\*\*\*\t5、括号法输出哈夫曼树\t\t       6、销毁哈夫曼树\t  \*\*\*" << endl;
228. cout << "\t\t\t\*\*\*\t0、退出\t\t\t      \t\t\t\t  \*\*\*" << endl;
229. cout << "\t\t\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;
230. }
232. **void** Huff\_main\_func()
233. {
234. BTNode\* B[MAXSIZE];
235. **for** (**int** i = 0; i < MAXSIZE; i++)
236. {
237. B[i] = NULL;
238. }
239. **int** n = MAXSIZE;
240. **int** choice;
241. **bool** flag = **false**;
242. **while** (1)
243. {
244. dis\_Huff\_menu();
245. cout << "请输入您的选择：" << endl;
246. cin >> choice;
247. **switch** (choice)
248. {
249. **case** 1:
250. Init(A, B, n);
251. flag = **true**;
252. system("pause");
253. system("cls");
254. **break**;
255. **case** 2:
256. **if** (flag)
257. {
258. creat\_HTree(B, A, n);
259. cout << "哈夫曼树已创建！" << endl;
260. }
261. **else**
262. {
263. cout << "请先初始化数据后在执行此操作！" << endl;
264. }
265. system("pause");
266. system("cls");
267. **break**;
268. **case** 3:
269. Dis\_Tree(B[n - 1]);
270. system("pause");
271. system("cls");
272. **break**;
273. **case** 4:
274. Get\_weight(B[n - 1]);
275. system("pause");
276. system("cls");
277. **break**;
278. **case** 5:
279. Dis\_HTree(B[n - 1]);
280. system("pause");
281. system("cls");
282. **break**;
283. **case** 6:
284. free\_BTREE(B[n - 1]);
285. flag = **false**;
286. system("pause");
287. system("cls");
288. **break**;
289. **case** 0:
290. **return**;
291. **default**:
292. cout << "您的输入有误，请重新输入" << endl;
293. system("pause");
294. system("cls");
295. **break**;
296. }
297. }
298. **订票系统部分源码：**

**·define.h**

1. #include<string>
2. #include<iostream>
3. #include<ctime>
4. #include<regex>
5. #include<queue>
6. #include<fstream>
7. #include<cstdio>
8. #define random(x) rand()%x
9. **using** **namespace** std;
10. **class** FTime {
11. **public**:
12. FTime();
13. FTime(**int** year, **int** month, **int** day, **int** hours, **int** minte);
14. **void** set\_time(**int** year, **int** month, **int** day, **int** hours, **int** minte);
15. **bool** mod\_Time();
16. **void** dis\_time();
17. **bool** is\_legal(**int** year, **int** month, **int** day, **int** hours, **int** minte);
18. **bool** is\_day\_legal(**int** year, **int** month, **int** day);
19. **bool** is\_leap(**int** year);
20. **int** get\_year();
21. **int** get\_month();
22. **int** get\_day();
23. **int** get\_hour();
24. **int** get\_minite();
25. **private**:
26. **int** m\_year;
27. **int** m\_month;
28. **int** m\_day;
29. **int** m\_hours;
30. **int** m\_minite;
31. };
32. **class** Flight {
33. **public**:
34. Flight();
35. Flight(string no, FTime t\_begin, FTime t\_end, string l\_begin, string l\_end, **float** price, **float** disc, **int** num, **int** maxsize);
36. **void** set\_flight(string no, FTime t\_begin, FTime t\_end, string l\_begin, string l\_end, **float** price, **float** disc, **int** num, **int** maxsize);
37. **void** dis\_flight();
38. **void** mod\_flight();
39. **void** set\_no(string no);
40. **void** set\_time(FTime begin, FTime end);
41. **void** set\_price(**float** price);
42. **void** set\_loc(string begin, string end);
43. **void** set\_disc(**float** disc);
44. **void** set\_num(**int** num);
45. **void** set\_maxsize(**int** maxsize);
46. string get\_no();
47. FTime get\_begin\_time();
48. FTime get\_end\_time();
49. **float** get\_price();
50. string get\_begin\_loc();
51. string get\_end\_loc();
52. **float** get\_disc();
53. **int** get\_num();
54. **int** get\_maxsize();
55. **private**:
56. string f\_no;  //航班编号
57. FTime time\_begin, time\_end;  //起降时间
58. **float** f\_price;  //票价
59. string loc\_begin, loc\_end;  //起降地点
60. **float** f\_disc;  //折扣
61. **int** f\_num;  //当前已定人数
62. **int** f\_maxsize;  //最大乘坐人数
63. };
64. **class** Client {
65. **public**:
66. Client();
67. Client(string name, string id, string no, **int** num, Flight flight);
68. **void** set\_client(string name, string id, string no, **int** num, Flight flight);
69. **void** dis\_client();
70. **void** mod\_client();
71. **void** set\_name(string name);
72. **void** set\_idcard(string id);
73. **void** set\_no(string no);
74. **void** set\_num(**int** num);
75. **void** set\_flight(Flight flight);
76. string get\_name();
77. string get\_idcard();
78. string get\_no();
79. **int** get\_num();
80. Flight get\_flight();
82. **private**:
83. string c\_name;  //客户姓名
84. string c\_idcard;  //身份证号
85. string c\_no;  //订单编号
86. **int** c\_num;  //订票数量
87. Flight c\_flight;  //所定航班
88. };
89. **struct** Flight\_list\_node {   //航班数据链表节点
90. Flight flight;  //航班信息
91. Flight\_list\_node\* next;  //下一个航班节点
92. };
93. **struct** Flight\_list\_head { //航班数据链表头
94. Flight\_list\_node\* head;  //头指针
95. **int** num;   //航班总数
96. };
97. **struct** Client\_list\_Node { //客户数据链表节点
98. Client client;  //客户信息
99. Client\_list\_Node\* next;   //下一个客户
100. };
101. **struct** Client\_list\_head {
102. Client\_list\_Node\* head;  //头
103. **int** num;  //客户总数
104. };

**·File\_operation\_flight.h**

1. #include"define\_flight.h"
2. **void** input\_flight(Flight& F);  //输入航班信息
3. **void** buy\_ticket\_no(Flight\_list\_head\* f\_head, Flight& F, string no, **int** c\_num);  //通过航班编号买票
4. **void** search\_path(Flight\_list\_head\* f\_head, string loc\_begin, string loc\_end, queue<queue<Flight>>&);  //求出其他可达路径
5. **void** buy\_ticket\_loc(Flight\_list\_head\* f\_head, Flight& F, string loc\_begin, string loc\_end, **int** c\_num);  //通过起始地点买票
6. **bool** input\_client(Client& C, Flight\_list\_head\* f\_head);  //输入乘客信息
7. **void** pushback\_client(Client\_list\_head\* c\_head, Flight\_list\_head\* f\_head); //添加乘客信息
8. **void** pushback\_flight(Flight\_list\_head\* f\_head);  //添加航班信息
9. **void** savefile\_flight(Flight\_list\_head\* f\_head);  //保存航班信息
10. **bool** readfile\_flight(Flight\_list\_head\* f\_head);  //读入航班信息
11. **void** savefile\_client(Client\_list\_head\* c\_head);  //保存乘客信息
12. **bool** readfile\_client(Client\_list\_head\* c\_head);   //读入乘客信息
13. **void** search\_flight\_loc(Flight\_list\_node\* p, string loc\_begin, string loc\_end);  //通过起始地点查找航班
14. **void** search\_flight\_no(Flight\_list\_node\* p, string no);  //通过航班编号查找航班
15. **void** search\_flight(Flight\_list\_head\* f\_head);  //查询航班
16. **void** init\_list(Client\_list\_head\*& c\_head, Flight\_list\_head\*& f\_head);  //初始化航班头和乘客头
17. **void** refund\_ticket(Client\_list\_head\*& c\_head, Flight\_list\_head\* f\_head);  //退票
18. **void** delete\_flight(Flight\_list\_head\* f\_head, Flight flight,**int** num);  //删除航班
19. **void** init\_flight(Flight\_list\_head\*& f\_head);  //重置机头
20. **void** init\_client(Client\_list\_head\*& c\_head);  //重置客户头
21. **void** dis\_flight\_info(Flight\_list\_head\*& f\_head); //展示所有航班
22. **void** search\_client(Client\_list\_head\* c\_head); //查询乘客信息
23. **void** modi\_flight(Flight\_list\_head\*& f\_head);  //修改航班信息
24. **void** modi\_client(Client\_list\_head\*& c\_head);  //修改乘客信息
25. **void** dis\_client\_info(Client\_list\_head\*& c\_head);  //展示所有乘客信息
26. **void** clear\_flight\_info();  //清空航班信息
27. **void** clear\_client\_info();  //清空乘客信息