《算法与数据结构》
课程设计报告

学院（系） 电子电气工程学院（计算机系）

班级学号

学生姓名

学生角色 ☑ 组长 ☐ 组员

指导老师 史志才

学　　期 2022 – 2023 - 2

日　　期： 2023 年 6 月 19 日 - 6 月 27 日

目录

一、约瑟夫生死游戏

1.1 题目

有 N 个旅客同乘一条船，因为严重超载，加上风高浪大，危险万分；因此船长告诉乘客，只有将全船一半的旅客投入海中，其余人才能幸免于难；无奈，大家只得同意这种办法，并议定 N 个人围成一圈，由第一个人开始，依次报数，数到第 9 人，便把他投入大海中，然后再从他的下一个人开始，数到第 9 人，再把他投入大海中，如此循环地进行，直到剩下 N/2 个乘客为止。问哪些乘客是将被投入大海的？输出这些乘客的姓名和位置。

1.2 概要设计

输入模块：输入总人数 N 及报数上限 k。

计算模块：以 k 为计数上限开始循环计数，每经过 k 个结点，就删除第 k 个结点；然后从下一个开始，重新循环，删除之后第 k 个节点；最终总结点≤总人数 N/2 时，循环结束。

输出模块：按删除次序，依次输出每个被删除的旅客结点的编号以及位置；然后从小到大遍历输出幸存者的编号。

1.3 详细设计和编码

1.3.1 存储结构

```
typedef struct Node {
    int data;
    int pos;
    struct Node* next;
} Node;
```

data 为每个节点的数据，在本程序中，为了节约时间，用循环自动按照从小到大遍历的数据命名；pos 为每个节点在输入时的位置信息。

1.3.2 重点函数

```
Node* createNode(int data, int pos)
```

根据主函数中乘客数 N 作为循环上限进行循环，data 为乘客序号，可以由用

户手动输入；pos 记录遍历次数，也就是乘客录入的顺序。

```
void insertAtEnd(Node** head, int data, int pos)
```

用于题目需要一个循环单链表，尾结点序号连接到链表的头结点，实现循环链表。在结点输入时，需要将每一个节点插入链表的尾结点之后，同时连接到链表的头结点。

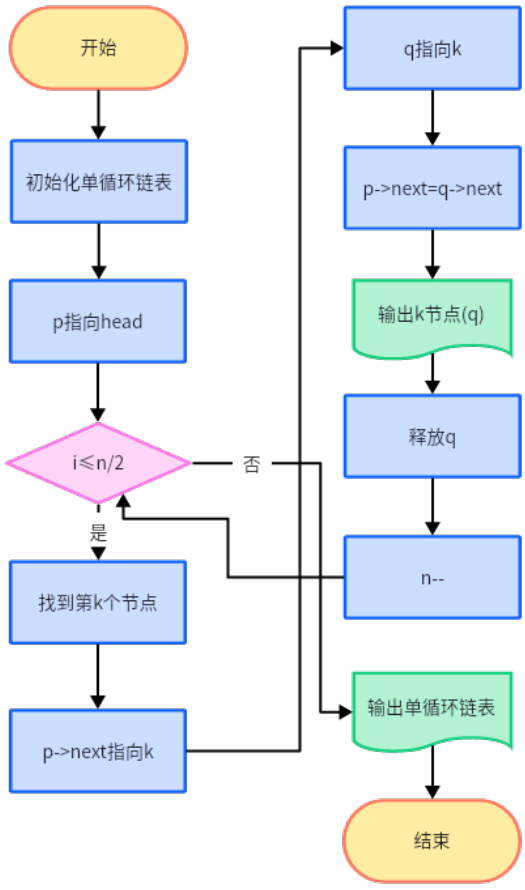
```
void printList(Node* head)
```

从头结点开始打印链表，但只打印结构体中的 data 成员，不需要输出 pos 成员，打印到尾结点后，停止打印。

```
void deleteKthNode(Node** head, int k)
```

删除第 k 个节点，head 为开始计数 k 的起始结点，并非链表的头结点，找到第 k 个元素后，把本函数下次输入的头结点更新为要删除的元素，并把元素的数据进行释放，把下次输入的头结点更新为下一个元素，即为新一轮计数 k 的起始起点。

1.3.3 流程图



1.4 调试分析及运行结果

```
乘客数(N)：35
计数上限(k)：18
死亡名单：
死者：18 位置：18
死者：1 位置：1
死者：20 位置：20
死者：4 位置：4
死者：24 位置：24
死者：9 位置：9
死者：30 位置：30
死者：16 位置：16
死者：5 位置：5
死者：28 位置：28
死者：17 位置：17
死者：8 位置：8
死者：34 位置：34
死者：27 位置：27
死者：22 位置：22
死者：15 位置：15
死者：13 位置：13
死者：12 位置：12
幸存者名单：
14 19 21 23 25 26 29 31 32 33 35 2 3 6 7 10 11

进程已结束,退出代码0
```

二、八皇后问题

2.1 题目

在 8×8 的国际象棋棋盘上，安放 8 个皇后，要求没有一个皇后能够“吃掉”任何其他一个皇后，即没有两个或两个以上的皇后占据棋盘上的同一行、同一列或同一条对角线。输出结果以 8×8 的棋盘形式显示。

2.2 概要设计

编写一个打印棋盘的函数按顺序打印所有皇后摆放后棋盘的形式，存在皇后的位置输出 Q，不存在皇后的位置输出-。

编写一个皇后判定函数，判断新皇后可以加入的位置情况：行、列及对角线上是否存在其他皇后，若存在则往回递归，将上一个皇后更换为另一个可以摆放的位置后，重新递归寻找新皇后的位置，直至第八个皇后可以找到摆放的位置；最后调用打印函数，将棋盘打印出来

2.3 详细设计和编码

2.3.1 存储结构

```
int board[N][N]
```

通过一个二维数组表示整个棋盘，元素为 1 即为该格为皇后所在位置，元素为 1 即该格为空。

2.3.2 重点函数

```
void printBoard(int board[N][N])
```

二维数组通过两次循环嵌套，逐行打印棋盘数据，元素为 1 打印 Q，元素为 0 打印空。

```
bool isSafe(int board[N][N], int row, int col)
```

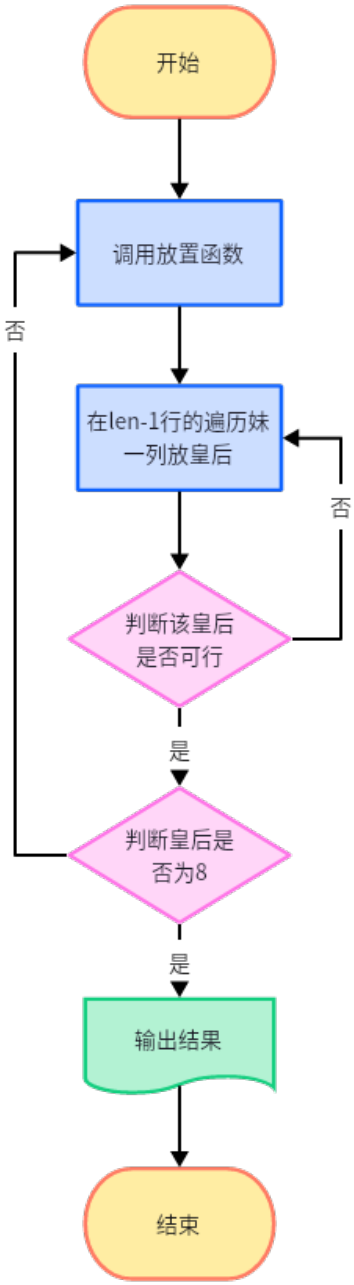
根据新录入的行列坐标，判断这一个皇后是否会“吃掉”棋盘上已有的其他皇后，如果会则返回 FALSE，往回递归寻找另一个合适的皇后位置；如果不会则返回 TRUE，继续运行程序。

```
bool solveNQueens(int board[N][N], int row)
```

先尝试在第 row 行的第 i 列进行摆放，然后开始判定是否存在“吃掉”情况，

若出现“吃掉”，则进行递归，尝试在第 row 行的第 i+1 列进行摆放，然后继续判定，直到 row 等于 N 停止递归。

2.3.3 流程图



2.4 调试分析及运行结果

```
请在第一行选择一列作为皇后的初始坐标，输入纵坐标：
6
-  -  -  -  -  -  Q  -
Q  -  -  -  -  -  -  -
-  -  Q  -  -  -  -  -
-  -  -  -  -  -  -  Q
-  -  -  -  -  Q  -  -
-  -  -  Q  -  -  -  -
-  Q  -  -  -  -  -  -
-  -  -  -  Q  -  -  -
```

三、航班信息的查询与检索

3.1 题目

每个航班记录包括七项，分别是航班号、起点站、终点站、起飞时间、到达时间、飞机型号和票价；对飞机航班信息按航班号进行排序；按航班的航班号、起点站、到达站、起飞时间以及到达时间等信息进行查询。要求按航班号的查找采用折半查找方法，其他采用顺序查找方法。

3.2 概要设计

通过主页菜单设计，程序整体分为录入航班信息、按照航班号查询航班信息、按照其他信息查询航班信息三个功能，通过 1、2、3 三个数字输入进行选择。

录入航班信息：在加载 flight.txt 查看已有信息的基础上，在后面新增一行航班信息。

按照航班号查询航班信息：针对航班号字符串的大小进行冒泡排序，然后使用二分法进行查找，找到对应的航班号后，确定结构体数组序号，根据序号输出该序号下的完整结构体信息，即该航班号的全部信息。

按照其他信息查询航班信息：用户可以根据 1~6 进行选择按照哪个参数进行查询，然后对整个结构体进行遍历，找到符合要求的航班就直接输出，然后继续查找，直到遍历完整个结构体为止。

3.3 详细设计和编码

3.3.1 存储结构

```
typedef struct {
    char flightNo[10];
    char departure[20];
    char destination[20];
    char departureTime[10];
    char arrivalTime[10];
    char aircraftModel[20];
```

```
    float ticketPrice;
} FlightRecord;
```

在结构体中定义航班信息的全部成员：航班号、起点站、终点站、起飞时间、到达时间、飞机型号和票价。

3.3.2 重点函数

```
void sortByFlightNo(FlightRecord records[], int numRecords)
```

通过 strcmp 函数进行字符的大小比较，以此实现字符串的冒泡排序。通过 FlightRecord tmp 实现结构体交换。

```
int binarySearch(FlightRecord records[], int numRecords, char searchFlightNo[])
```

numRecords 为航班数据的总条数，以该数量的一半作为折半的中点进行查找，若航班号的字符串信息大于中点的数据，则往右找，反之同理，以此循环，直到折半的中点等于航班的信息，查找结束。

```
void displayFlightRecord(FlightRecord record)
```

根据结构体数组，打印该数组对应的全部结构体成员。

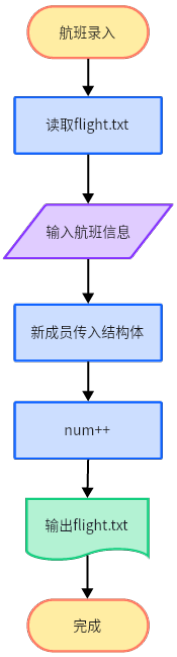
```
void save(int numRecords, FlightRecord fight[])
```

输出航班数据的总条数，同时也是存储循环的上限，然后将结构体数组的每一个元素存进 flight.txt。

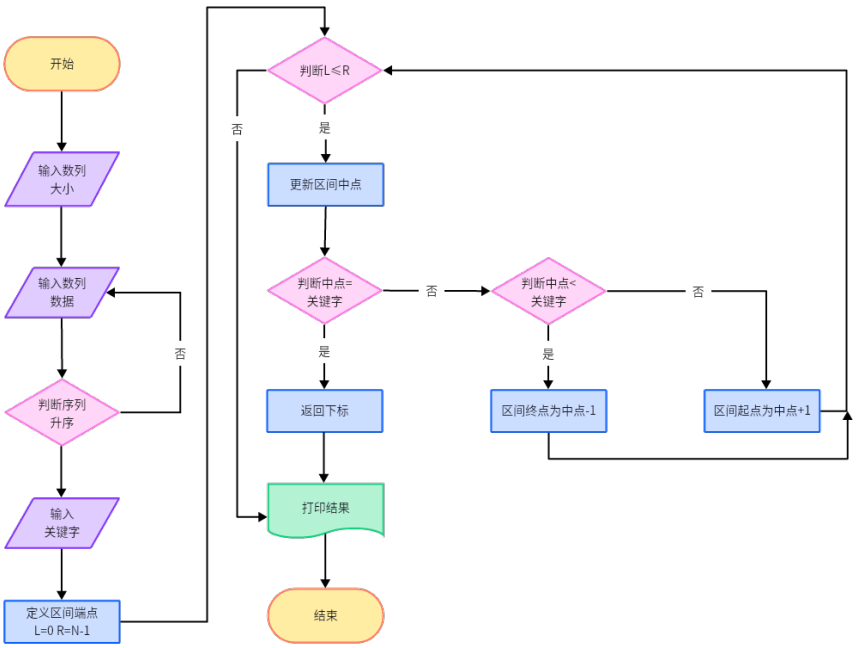
```
void load(int* numRecords, FlightRecord* fight)
```

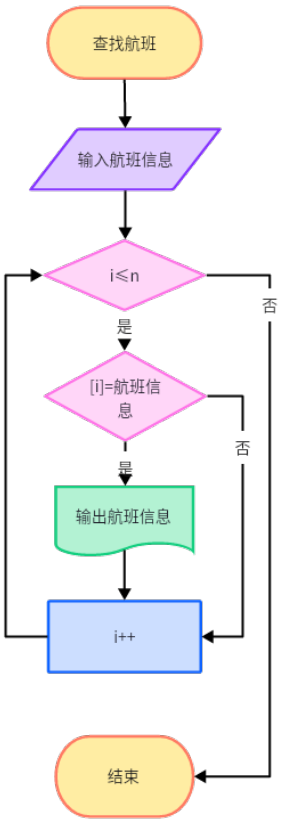
首先读取 fligt.txt 的数据条数，同时也是读取循环的上限，然后将 flight.txt 的每一行数据读取到结构体数组中。。

3.3.3 流程图


航班录入
读取flight.txt
输入航班信息
新成员传入结构体
num++
输出flight.txt
完成

折半查找：


开始
输入数列
大小
输入数列
数据
否
判断序列
升序
输入
关键字
定义区间端点
L=0 R=N-1
判断L≤R
否
是
更新区间中点
判断中点=
关键字
否
判断中点<
关键字
否
是
是
返回下标
区间终点为中点-1
区间起点为中点+1
打印结果
结束



3.4 调试分析及运行结果

xt × flight.txt × ex8-Huffman.cpp × main.cpp ×

```
4
MU0123  北京 上海 08:00   10:00   Boeing737   800
CZ0456  上海 北京 12:00   14:00   AirbusA320  1000
9C0789  广州 深圳 15:00   16:00   Boeing747   500
3u8633  sha pvg 1200    1230    c919    2000
```

四、内部排序

4.1 题目

设计一个测试程序比较几种内部排序算法的关键字比较次数和移动次数以取得直观感受。要求：

1.对起泡排序、直接排序、简单选择排序、快速排序、希尔排序、堆排序、归并排序算法进行比较；

2.待排序记录的关键字为整数。其中的数据要用伪随机产生程序产生(如10000 个)，至少用 5 组不同的输入数据做比较，再使用各种算法对其进行排序，记录其排序时间，再汇总比较。

3.每次测试完毕显示各种比较指标值的列表，以便比较各种排序的优劣。

4.2 概要设计

将伪随机生成的 10000 个元素存入数组中，然后先通过冒泡排序记录关键字比较次数、交换次数、算法时间，然后将这个过程循环 5 次后得到平均指标。将这个过程更换为直接排序、简单选择排序、快速排序、希尔排序、堆排序、归并排序算法，最后将所有指标进行输出，直观地比较这些排序的区别。

4.3 详细设计和编码

4.3.1 存储结构

```
int* arr = (int*)malloc(size * sizeof(int));
```

根据 size 大小，分配数组的空间大小。

```
struct cmp{
    double time;
    int cmpcnt;
    int movcnt;
};
```

定义需要比较的指标：算法时间、关键字比较次数、交换次数。

4.3.2 重点函数

```
int* generateRandomArray(int size)
```

根据 size 大小，以及 random 函数，随机定义 size 个不同的数据，进行排序。

```
void swap(int* a, int* b)
```

实现 a 数据与 b 数据的交换。

```
void bubbleSort(int arr[], int size, int* comparisons, int* moves)
```

通过每两个数据的比较，将最大的数据交换至最后一位，以此类推，实现冒泡排序。

```
void selectionSort(int arr[], int size, int* comparisons, int* moves)
```

找到数组中最小的元素，把最小的元素交换至第一位，以此类推，实现简单选择排序。

```
void insertionSort(int arr[], int size, int* comparisons, int* moves)
```

先确定数组中的已排序部分，然后把关键字后的未排序部分插入已排序部分中的合适位置，以此类推，实现插入排序。

```
void quickSort(int arr[], int low, int high, int* comparisons, int*
moves)
```

将数组分成两部分，一部分是小于基准值的元素，另一部分是大于基准值的元素。可以使用两个指针，一个从左往右找大于基准值的元素，另一个从右往左找小于基准值的元素，然后交换它们的位置，直到两个指针相遇，最终实现快速排序。

```
void shellSort(int arr[], int size, int* comparisons, int* moves)
```

选择一个增量序列，按照增量对数组进行分组，每个分组包含相距为增量的元素。对每个分组使用插入排序算法进行排序。

```
void heapSort(int arr[], int n, int* comparisons, int* moves)
```

将数组看作是一个完全二叉树，从最后一个非叶子节点开始，依次向上调整各个节点的位置，使得每个父节点的值都大于等于其子节点的值。由于最大堆的根节点是序列中的最大值，将其与序列的最后一个元素交换，并将序列长度-1。然后重新调整堆，使得剩余的元素仍构成一个最大堆。以此类推，实现堆排序。

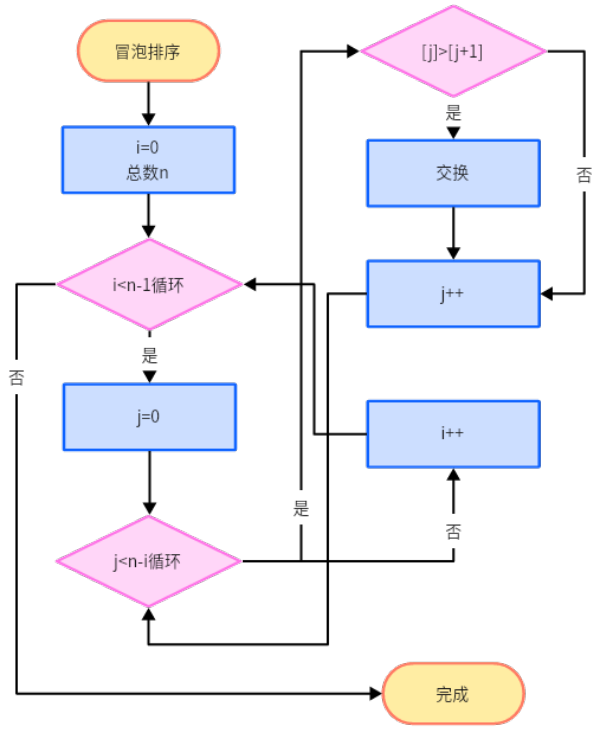
```
void mergeSort(int* arr, int left, int right, int* compareCount, int*
moveCount)
```

将数组不断地对半分割，直到每个子列表只包含一个元素。将相邻的列表两

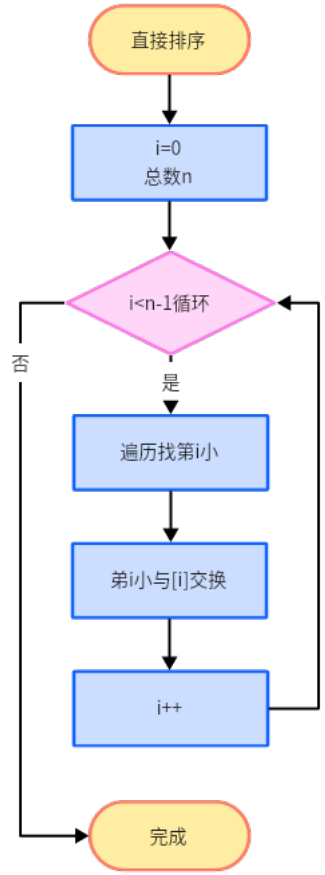
两合并，并按照顺序进行排序。比较两个子列表中的元素，将较小的元素放入结果列表中。以此类推，实现归并排序。

4.3.3 流程图

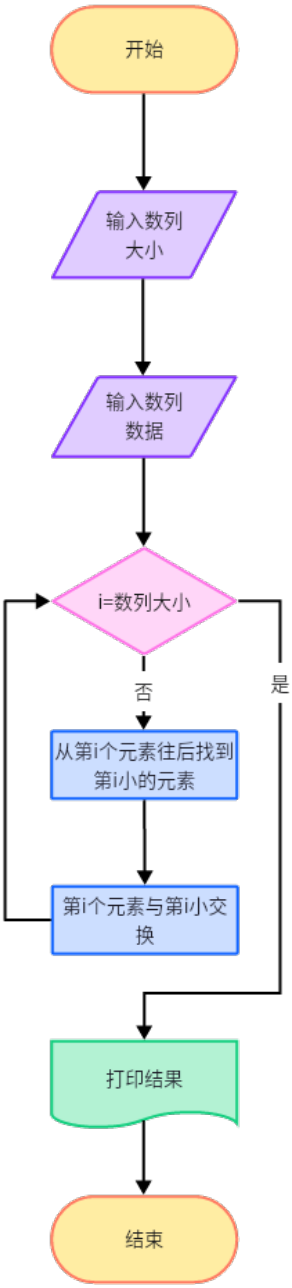
起泡排序：


冒泡排序
i=0
总数n
i<n-1循环
是
否
j=0
j<n-i循环
是
否
[j]>[j+1]
是
交换
否
j++
i++
完成

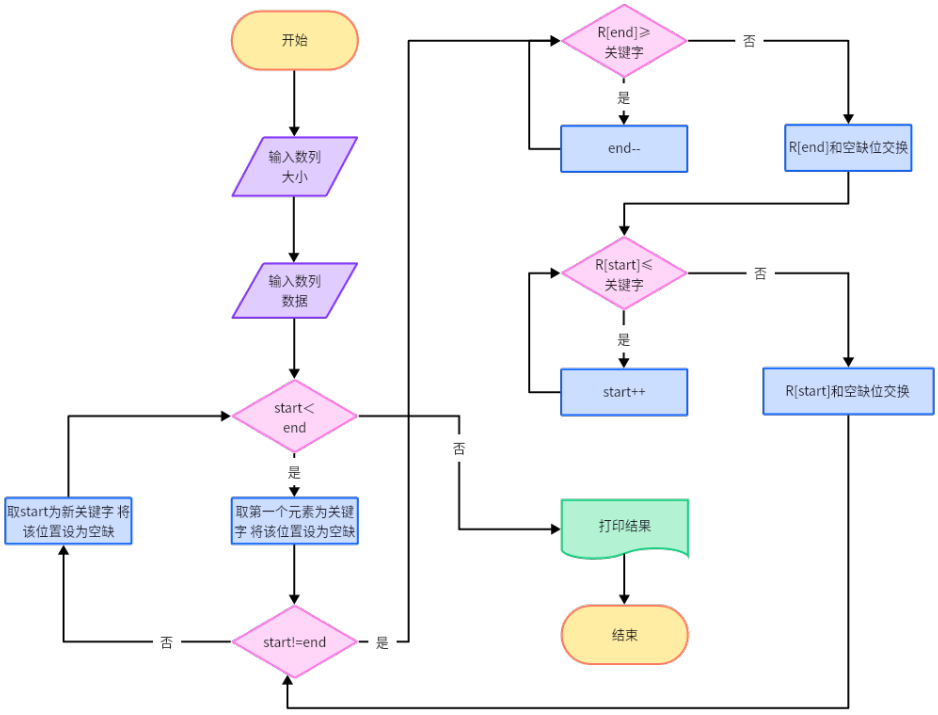
直接排序：


直接排序
i=0
总数n
i<n-1循环
否
是
遍历找第i小
弟i小与[i]交换
i++
完成

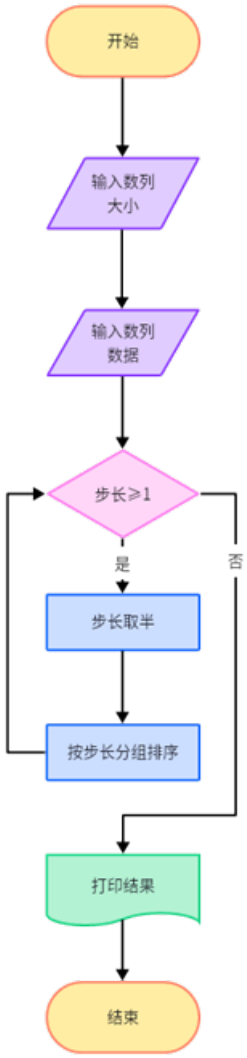
简单选择排序：


开始
输入数列
大小
输入数列
数据
i=数列大小
否
是
从第i个元素往后找到
第i小的元素
第i个元素与第i小交
换
打印结果
结束

快速排序：


开始
输入数列大小
输入数列数据
start< end
是
否
取第一个元素为关键字 将该位置设为空缺
取start为新关键字 将该位置设为空缺
start!=end
否
是
R[end]≥关键字
是
否
end--
R[end]和空缺位交换
R[start]≤关键字
是
否
start++
R[start]和空缺位交换
打印结果
结束

希尔排序：



4.4 调试分析及运行结果

	起泡排序	直接排序	选择排序	快速排序	希尔排序	堆排序	归并排序
排序时间：	0.2528	0.0892	0.0568	0.0008	0.0012	0.0018	0.0014
比较次数：	49995000	49995000	25258274	183779	249642	166031	120449
移动次数：	24801237	9984	25248276	73503	129637	372030	267232

五、校园导游咨询

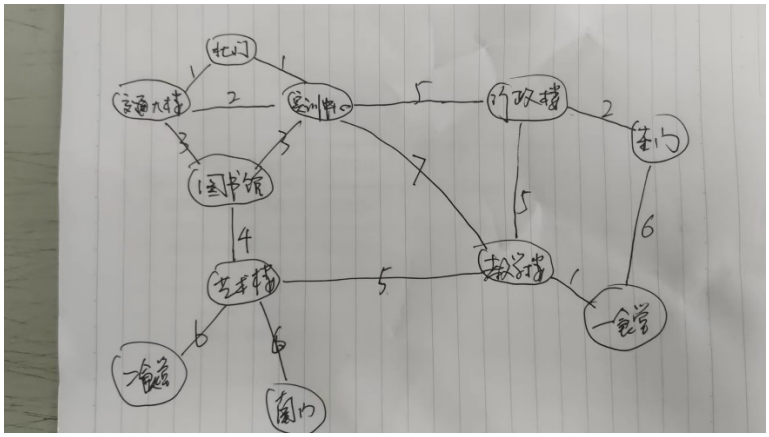
5.1 题目

设计一个校园导游程序，为来访的客人提供各种信息查询服务。要求：

1.设计上海工程技术大学的校园平面图，所含景点不少于 10 个。各景点信息包括代号、名称、简介等信息。

2.为来访客人提供图中任意景点相关信息的查询（利用不同的遍历方法）。

3.为来访客人提供图中任意景点的问路查询，即查询任意两个景点之间的一条最短的简单路径。



5.2 概要设计

通过主页菜单设计，程序整体分为节点查询和节点导航两个功能，通过 1、2 数字输入进行选择。

节点查询：校园的道路是双向通行的，可设校园平面图是无向图。运用无向图表示学校内的景点情况，对于每个节点储存它的名称和介绍，利用深度优先和广度优先和查找景点，确定节点序号，然后输出改节点的结构体信息，即景点介绍。

节点导航：在邻接矩阵中，存储每两个节点之间的距离，若无法直接联通即为无穷大，用 dijkstra 算法计算起点到别的点的最短距离，当算法遍历到了终点时，返回运算结构，输出最短路径和最短距离，实现导航功能。

5.3 详细设计和编码

5.3.1 存储结构

```
typedef struct {
    char name[50];
    std::string description;
} Node;
```

学校景点名称、景点介绍

```
typedef struct {
    int queue[MAX_NODES];
    int front, rear;
} Queue;
```

包括学校所有节点的队列，用于实现遍历查找。

5.3.2 重点函数

```
int breadthFirstSearch(int eNode, int numNodes)
```

BFS 广度搜索：将起始顶点放入队列中；从队列中取出一个顶点作为当前顶点，并将其标记为已访问；访问当前顶点，将该访问节点输出，同时判断该节点是否为重点，若是则中断循环，返回节点数据；将当前顶点的未访问过的相邻顶点放入队列中。以此类推，直至遍历实现全部搜索。

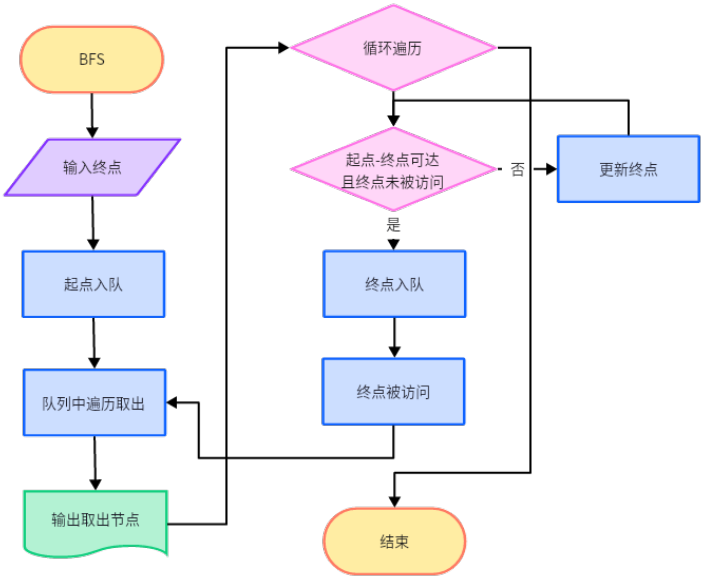
```
void depthFirstSearch(int startNode, int eNode, int numNodes)
```

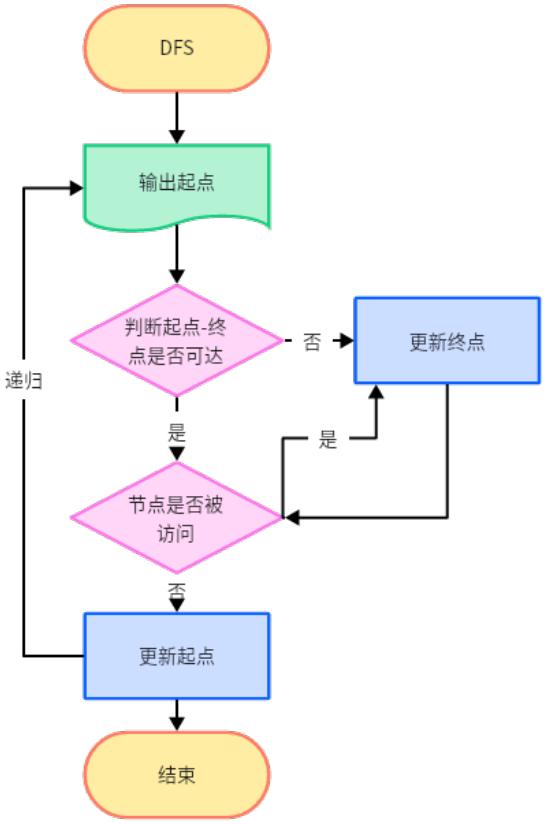
从起始节点开始，访问该节点并标记为已访问，然后递归地访问其未被访问的邻居节点。

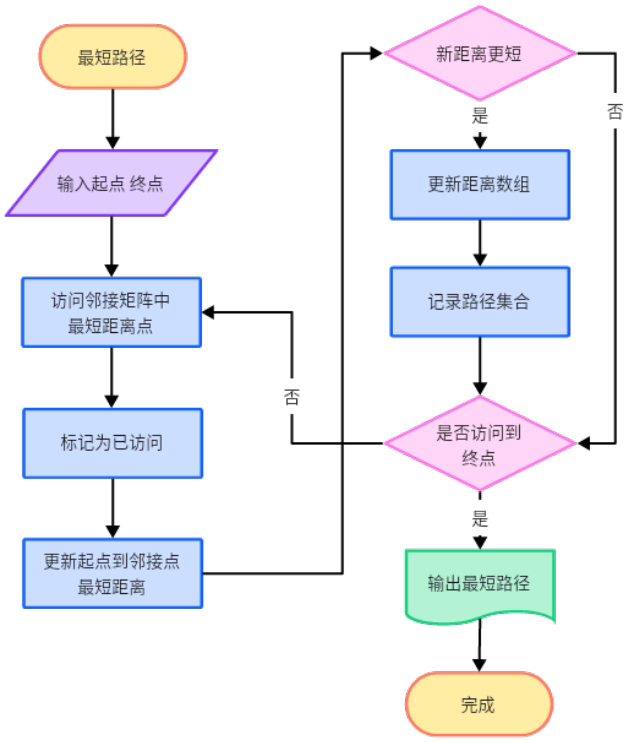
```
void dijkstra(int start, int end)
```

在邻接矩阵数组中选择未访问节点中，距离最小的节点，并将其标记为已访问。更新与该节点相邻节点的最短距离。如果经过该节点到达相邻节点的距离更短，则更新距离数组中的值。直至访问到 end 节点为 start 出发的最短距离节点时，返回主函数。

5.3.3 流程图


BFS
输入终点
起点入队
队列中遍历取出
输出取出节点
循环遍历
起点-终点可达
且终点未被访问
否
更新终点
是
终点入队
终点被访问
结束


DFS
输出起点
判断起点-终
点是否可达
否
更新终点
递归
是
是
节点是否被
访问
否
更新起点
结束



5.4 调试分析及运行结果

六、哈希表的应用

6.1 题目

每个通讯者的信息包括手机号码、姓名、住址，设计哈希表实现通讯录查找系统。要求以手机号码为关键字，采用线性探测再散列法解决冲突；可按手机号码查找通讯者的信息。

6.2 概要设计

通过主页菜单设计，程序整体分为好友录入和好友查询两个功能，通过 1、2 数字输入进行选择。

好友录入：通过录入的手机号信息，进行哈希运算，将函数结果存放进 contacts.txt 中，如出现冲突，则采用线性探测。

好友查询：将输入的手机号进行哈希运算，然后在 contacts.txt 中进行哈希匹配，若匹配成功，再进行一次比较查询，确定没有因为冲突而影响结构，比较成功后输出节点信息，即好友信息。

6.3 详细设计和编码

6.3.1 存储结构

```
typedef struct {
    char name[50];
    char address[100];
} Contact;
```

联系人姓名、地址。

```
typedef struct {
    char phoneNum[20];
    Contact contact;
} HashNode;
```

哈希表节点：手机号、联系人结构体。

6.3.2 重点函数

```
int hashFunction(char phoneNum[])
```

用哈希函数计算手机号对应的哈希值。

```
void insertContact(HashNode hashTable[], char phoneNum[], char name[], char address[])
```

将手机号对应的哈希值插入到哈希表中，如果冲突则采用线性探测进行插入。确定好插入位置后，将手机号对应的节点全部插入进哈希表中。

```
void save(HashNode hashTable[])
```

把哈希表的全部节点的成员逐行保存到 contacts.txt。

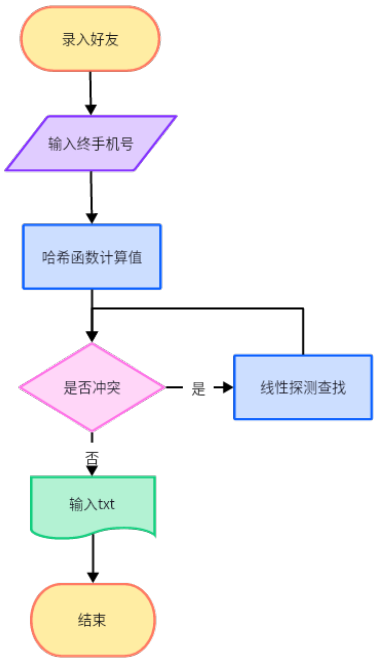
```
void load(HashNode* hashTable)
```

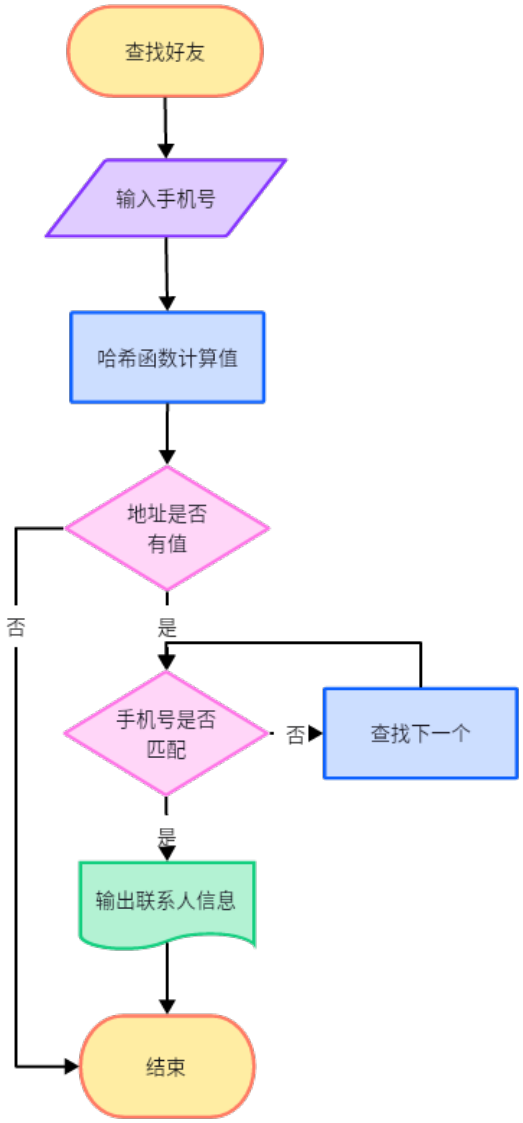
把 contacts.txt 中的全部数据逐行读取到哈希表中。

```
void findContact(HashNode hashTable[], char phoneNum[])
```

根据手机号码查找通讯者信息，如遇冲突的情况，则使用线性探测再散列法查找。

6.3.3 流程图





6.4 调试分析及运行结果

七、哈夫曼编码/译码器

7.1 题目

设计一个哈夫曼编码/译码系统，对一个文本文件中的字符进行哈夫曼编码，生成编码文件；反过来，可将一个编码文件译码还原为一个文本文件(.txt)。要求：

1.输入一个待压缩的文本文件名，统计文本文件中各字符的个数作为权值，生成哈夫曼树；

2.将文本文件利用哈夫曼树进行编码，生成压缩文件；

3.输入一个待解压的压缩文件名称，并利用相应的哈夫曼树将编码序列译码；

4.可显示指定的压缩文件和文本文件；

7.2 概要设计

通过主页菜单设计，程序整体分为对文件编码、对输入进行编码、对编码数据文件进行译码三个功能，通过数字 1~3 输入进行选择。

对文件编码：根据输入的数据，经过遍历，运算得出每个字符出现的次数，然后根据出现次数进行排序，并将字符和出现次数输出到 data.txt 中；然后生成哈夫曼树，按照左 0 右 1 的原则进行哈夫曼编码，每一个字符拥有自己唯一的编码。按照哈夫曼编码对数据文件进行编写，实现文件的压缩，输出到CodeFile.txt。

对输入进行编码：根据输入的字符和对应的权重，进行排序，生成哈夫曼树，按照左 0 右 1 的原则进行哈夫曼编码，每一个字符拥有自己唯一的编码。

压缩后的编码译码：根据 data.txt 中的数据重新生产哈夫曼树，然后读取 CodeFile.txt 中的编码，按照左 0 右 1 的原则，找到叶子节点，确认编码对应的字符，实现解码，最终将解码内容输出到 DecodeFile.txt。

7.3 详细设计和编码

7.3.1 存储结构

```
struct HuffmanTreeNode{
    char data;
    double weight;
```

```
    int parent,lchild,rchild;
};
```

哈夫曼树的节点：数据、权重、父亲数据、儿子数据。

```
struct HuffmanTree{
    HuffmanTreeNode *node;
    int n;
};
```

哈夫曼树：节点、节点总数。

7.3.2 重点函数

```
void SortHufmtree(HuffmanTree *tree)
```

通过循环遍历，计算每一个字符的出现次数，然后将出现次数转换为权重，从而进行节点的排序。

```
HuffmanTree* BuildHuffmanTree(HuffmanTree *tree)
```

创建一个最小堆，用于选择最小频率的节点；创建所有叶子节点，每个节点包含一个字符和频率；持续循环直到只剩一个节点；从二叉树中选择两个最小频率的节点；创建一个新的父节点，并将这两个节点设为其子节点；继续寻找新的最小频率的节点。

```
Codetype* HuffmanCode(HuffmanTree *tree)
```

如果是叶子节点，则打印字符和对应的哈夫曼编码，否则递归调用左右子树，并在每次递归时更新哈夫曼编码字符串。

```
HuffmanTree* CreateHuffmanTreeFromSourceFile()
```

从文件中读取文本，将字符和权重的对应信息输出到 data.txt，完成构建哈夫曼树，得出每个字符的哈夫曼编码，并且输出对文本压缩后的哈夫曼编码到 CodeFile.txt。

```
HuffmanTree* CreateHuffmanTreeByInput()
```

从文件中读取文本，构建哈夫曼树，得出每个字符的哈夫曼编码。

```
HuffmanTree* CreateHuffmanTreeFromDataFile()
```

在译码过程中，需要重新生成哈夫曼树，哈夫曼树的构建来自 data.txt。在 data 文本中，根据频率可以重新生成一个哈夫曼树，以及每个字符的哈夫曼编码，来

帮助译码的实现。

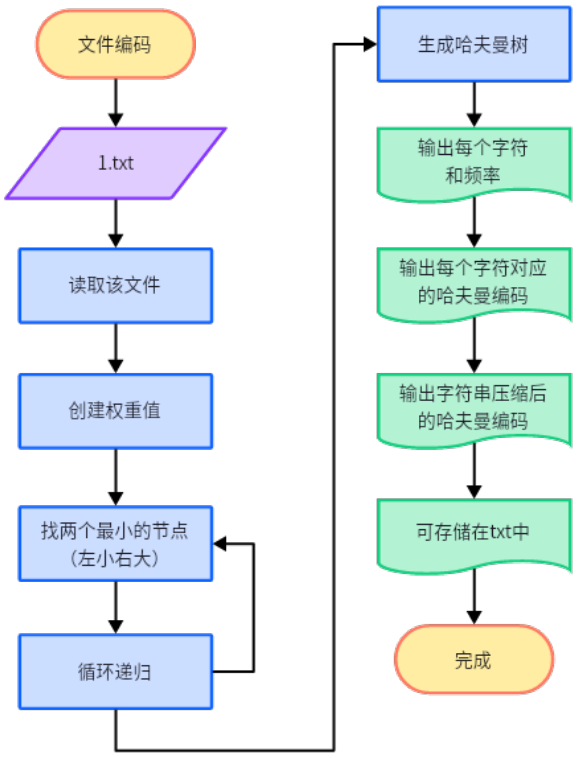
```
HuffmanTree* Encoding(HuffmanTree *tree)
```

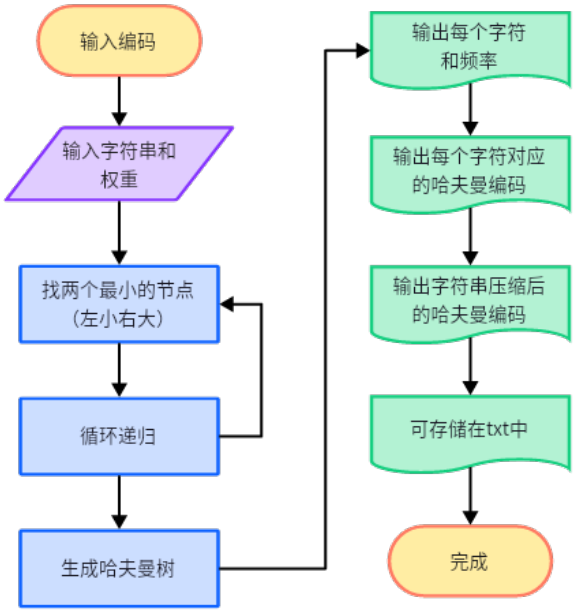
根据哈夫曼树，可以得出每个字符的哈夫曼编码，以此输出对文本压缩后的哈夫曼编码到 CodeFile.txt。

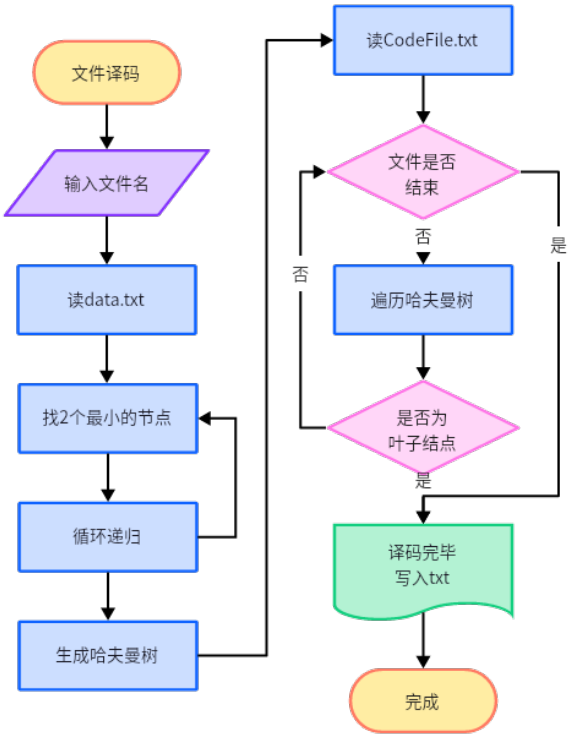
```
void Decoding(HuffmanTree *tree)
```

根据 CodeFile.txt 中被压缩的数据、每个字符的哈夫曼编码进行译码，以此输出原文本到 DeodeFile.txt。

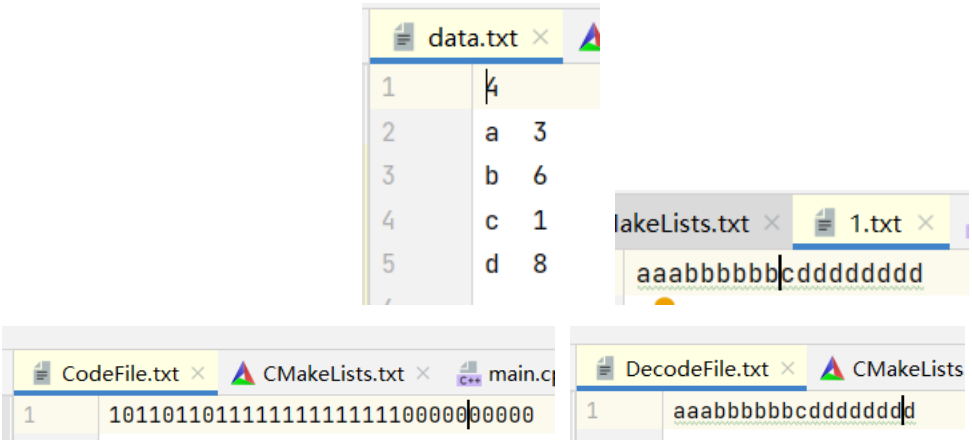
7.3.3 流程图




输入编码
输入字符串和
权重
找两个最小的节点
（左小右大）
循环递归
生成哈夫曼树
输出每个字符
和频率
输出每个字符对应
的哈夫曼编码
输出字符串压缩后
的哈夫曼编码
可存储在txt中
完成


文件译码
输入文件名
读data.txt
找2个最小的节点
循环递归
生成哈夫曼树
读CodeFile.txt
文件是否
结束
否
是
否
遍历哈夫曼树
是否为
叶子结点
是
译码完毕
写入txt
完成

7.4 调试分析及运行结果



八、课程设计总结

经过本次课程设计，我通过编写一些有趣的程序代码来使算法与数据结构课上的理论知识化归实践，同时，我也更加地理解了数据结构的有关知识。我熟悉了队列、堆栈等线性存储结构和树、图等非线性存储结构在实际编程中的使用，并且能按照不同的需求选择较为恰当的存储结构实现对应功能并完成简单的程序功能设计。

题目一的重点在于循环单链表的设置和删除结点的实现，需要循环链表来保证能够一次次不停的遍历链表直至输出到一半人数的时候才会停止，这里其中涉及了链表的抽象数据类型定义，单链表的创建，以及链表中节点的删除。同时，通过借助网上的教学资源，我对循环链表和单链表之间的区别更加清晰。

题目三是八皇后问题，主要在于对这个问题的求解方法。我认为比较难的地方在于有时前几行的放置会导致下一行无论哪一列都不能放置皇后，也算是算法设计的一个经典题型了。这个实验加强了我对于递归的理解。

题目四中由于需要用到折半查找，一开始我苦恼于字符串的比较，后来使用了 strcmp 解决了问题。在这一题中复习了 fstream.h 头文件，将结果导出到了 txt 文件中，同时也从 txt 文件中读取历史数据，减少多次调试的输入麻烦。但是这一题不足的地方主要在于每一次都要对整个数据进行排序，而不是针对新输入的数据进行插入排序，增强了程序的时间复杂度。

题目五是内部排序的效率比较，在本题中学习了 time.h 的使用，通过获取当前时间，然后作差得出某一段算法的具体运行时间。从结果上看，快速排序、希尔排序、堆排序、归并排序都是很快的

题目六校园景点，BFS 和 DFS 算法较为容易，但是在邻接矩阵中一开始可达判定标准为 0，后来意识到问题后才改为无穷大。在后续的 dijkstra 算法中，我顺利计算出了最短距离，但是最短路径发现了很多的困难。最后采取了记录遍历过程中提取出来的每个点的上一个点，然后从最后一个点往回递归，输出最短路径。

题目七加深了我对哈希表的认识，也学会了怎样用代码实现哈希表的建立以及用线性探测法解决地址冲突。好的散列函数应该让散列地址分布均匀，减少空间浪费。

实验八哈夫曼编码，我学会了如何使用 fopen 函数，对于输入内容比较多的情况，用从文本读入的方法可以节省很多时间也更加方便。哈夫曼编码和译码其实是互逆的过程，是一种无损压缩，关键在于构造哈夫曼树，构造好哈夫曼树以后求出字符对应的哈夫曼编码表，可以进行编码和译码。在大量代码的调试过程中，需要经常设置断点查看不同监视，才能更好更快地发现自己的 bug，才能理解别人的变量代表的是什么意思，以及函数的功能与原理是什么。

在整个算法数据结构设计课中我收获的不单单只是这么一些理解，对我来说，从写代码的过程中我逐渐一步步了解到了机器思维是怎样的一个过程，在校对代码的过程中，一定要带入机器思维，这是我体悟最深也是了解最明确的一点。因为目前我的能力还不是非常成熟，无法做到一次就让自己的代码运行成功，那么每一次的 debug 都是我一次次不断的在失败中校正自己思路的过程，也是将自己的思维一次次多元化，多向化的过程。

在本次实验中，我也大量使用了 CLion 作为我的开发环境，在 CLion 内部的控制台可以更直观地看到输入和输出。同时在调试过程中，控制台和调试接口在相同界面的使用体验会更好。但有部分情况下，例如涉及到 system 语句清空命令台的时候会出现些许 bug，需要切换到外部控制台继续调试。

除此以外，在这两周不到的学习时间中，小组合作也是一件非常重要的事情。一个人很容易陷入自己的思维空间里，只有跟别人交流讨论的时候，才有可能从另一个角度思考问题解决问题。

参考文献

[1].赵敏媛，施一萍，张辉.数据结构[M].北京:中国铁道出版社，2011.

附录：程序清单

1.约瑟夫生死游戏

```
#include <cstdio>
#include <cstdlib>

typedef struct Node {
    int data;
    int pos;
    struct Node* next;
} Node;

Node* createNode(int data, int pos) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->pos = pos;
    newNode->next = NULL;
    return newNode;
}

void insertAtEnd(Node** head, int data, int pos) {
    Node* newNode = createNode(data, pos);
    if (*head == NULL) {
        *head = newNode;
        newNode->next = *head;
    } else {
        Node* temp = *head;
        while (temp->next != *head) {
            temp = temp->next;
        }
        temp->next = newNode;
        newNode->next = *head;
    }
}

void printList(Node* head) {
    Node* temp = head;
    do {
        printf("%d ", temp->data);
        temp = temp->next;
    } while (temp != head);
    printf("\n");
}

void deleteKthNode(Node** head, int k) {
    if (k == 1) {
        Node* temp = *head;
        *head = (*head)->next;
        free(temp);
        return;
    }
    Node* curr = (*head)->next;
    int count = 2;
    while (curr != NULL) {
        if (count == k) {
            (*head)->next = curr->next;
            printf("死者: %d ", curr->data);
            printf("位置: %d\n", curr->pos);
            free(curr);
            return;
        } else {
            *head = curr;
            curr = curr->next;
            count++;
        }
    }
```

```
}

int main(){
    int N, k, pos;
    printf("乘客数(N)：");
    scanf("%d", &N);
    printf("计数上限(k)：");
    scanf("%d", &k);
    Node* head = NULL;
    for (int i = 1; i <= N; i++) {
        pos = i;
        insertAtEnd(&head, i, pos);
    }
    printf("死亡名单：\n");
    struct Node* current = head;
    for(int i = N; i > N/2;current = current->next, i--){
        deleteKthNode(&current, k);
    }
    printf("幸存者名单：\n");
    printList(current);
    return 0;
}
```

2.八皇后问题

```
#include <cstdio>
#define N 8

void printBoard(int board[N][N]) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            printf("%c  ", board[i][j] ? 'Q' : '-');
        }
        printf("\n");
    }
}

bool isSafe(int board[N][N], int row, int col) {
    int i, j;
    // 检查列
    for (i = 0; i < row; i++)
        if (board[i][col])
            return false;
    // 左上对角线
    for (i = row, j = col; i >= 0 && j >= 0; i--, j--)
        if (board[i][j])
            return false;
    // 右上对角线
    for (i = row, j = col; i >= 0 && j < N; i--, j++)
        if (board[i][j])
            return false;
    return true;
}

bool solveNQueens(int board[N][N], int row) {
    if (row == N)
        return true;
    for (int col = 0; col < N; col++) {
        if (isSafe(board, row, col)) {
            board[row][col] = 1;
            if (solveNQueens(board, row + 1))
                return true;
            board[row][col] = 0;
        }
    }
    return false;
}

int main() {
    int board[N][N] = {0};
    printf("请在第一行选择一列作为皇后的初始坐标，输入纵坐标：\n");
    int x;
    scanf("%d", &x);
    board[0][x] = 1;
    if (solveNQueens(board, 1)) {
        printBoard(board);
    }
    return 0;
}
```

3.航班信息的查询与检索

```
#include <cstdio>
#include <cstring>
#include <iostream>
#include <fstream>
#define MAX_FLIGHTS 100

typedef struct {
    char flightNo[10] = "-1";
    char departure[20] = "-1";
    char destination[20] = "-1";
    char departureTime[10] = "-1";
    char arrivalTime[10] = "-1";
    char aircraftModel[20] = "-1";
    float ticketPrice = 0;
} FlightRecord;

int binarySearch(FlightRecord records[], int numRecords, char searchFlightNo[]) {
    int low = 0, high = numRecords - 1, mid;

    while (low <= high) {
        mid = (low + high) / 2;
        if (strcmp(records[mid].flightNo, searchFlightNo) == 0) {
            return mid;
        }
        if (strcmp(records[mid].flightNo, searchFlightNo) < 0) {
            low = mid + 1;
        } else {
            high = mid - 1;
        }
    }
    return -1;
}

void sortByFlightNo(FlightRecord records[], int numRecords) {
    int i, j;
    FlightRecord temp;
    for (i = 0; i < numRecords - 1; i++) {
        for (j = 0; j < numRecords - i - 1; j++) {
            if (strcmp(records[j].flightNo, records[j + 1].flightNo) > 0) {
                temp = records[j];
                records[j] = records[j + 1];
                records[j + 1] = temp;
            }
        }
    }
}

void displayFlightRecord(FlightRecord record) {
    printf("航班号：%s\n", record.flightNo);
    printf("起点站：%s\n", record.departure);
    printf("终点站：%s\n", record.destination);
    printf("起飞时间：%s\n", record.departureTime);
    printf("到达时间：%s\n", record.arrivalTime);
    printf("飞机型号：%s\n", record.aircraftModel);
    printf("票价：%.2f\n", record.ticketPrice);
    printf("------------------------------\n");
}

void save(int numRecords, FlightRecord fight[]){
    std::ofstream ofs;
    ofs.open("flight.txt", std::ios::out);
    ofs<<numRecords<<std::endl;
    for (int i = 0; i < numRecords; i++)
    {
        ofs<<fight[i].flightNo<<"\t"
            <<fight[i].departure<<"\t"
```

```
            <<fight[i].destination<<"\t"
            <<fight[i].departureTime<<"\t"
            <<fight[i].arrivalTime<<"\t"
            <<fight[i].aircraftModel<<"\t"
            <<fight[i].ticketPrice<<std::endl;
    }
    ofs.close();
}

void load(int* numRecords, FlightRecord* fight) {
    std::ifstream ifs;
    ifs.open("flight.txt", std::ios::in);
    if (ifs.is_open()) {
        printf("成功读取通讯录，请您选择：\n");
        ifs>>*numRecords;
        for (int i = 0; i < *numRecords; i++) {
            ifs>>fight[i].flightNo
                >>fight[i].departure
                >>fight[i].destination
                >>fight[i].departureTime
                >>fight[i].arrivalTime
                >>fight[i].aircraftModel
                >>fight[i].ticketPrice;
        }
    } else {
        printf("flight.txt");
    }
    ifs.close();
}

int main() {
    FlightRecord flightRecords[MAX_FLIGHTS];
    int numRecords;
    while (true) {
        system("cls");
        load(&numRecords, flightRecords);
        char fno[10];
        char depart[20];
        char des[20];
        char departTime[10];
        char arriTime[10];
        char airModel[20];
        float tkPrice;

        int choice;
        printf("------[1]航班录入--------\n");
        printf("------[2]航班号查询------\n");
        printf("------[3]其他查询--------\n");
        printf("------[4]退出系统--------\n");
        scanf("%d", &choice);
        switch (choice) {
            case 1: {
                printf("请输入航班号：");
                scanf(" %s", &fno);
                printf("请输入起点：");
                scanf(" %s", &depart);
                printf("请输入终点：");
                scanf(" %s", &des);
                printf("请输入起飞时间：");
                scanf(" %s", &departTime);
                printf("请输入到达时间：");
                scanf(" %s", &arriTime);
                printf("请输入飞机型号：");
                scanf(" %s", &airModel);
                printf("请输入票价：");
                scanf("%f", &tkPrice);

                strcpy(flightRecords[numRecords].flightNo,fno);
```

```
            strcpy(flightRecords[numRecords].departure,depart);
            strcpy(flightRecords[numRecords].destination,des);
            strcpy(flightRecords[numRecords].departureTime,departTime);
            strcpy(flightRecords[numRecords].aircraftModel,airModel);
            strcpy(flightRecords[numRecords].arrivalTime,arriTime);
            flightRecords[numRecords].ticketPrice=tkPrice;
            numRecords++;

            save(numRecords, flightRecords);
            system("pause");
            break;
        }
        case 2: {
            printf("请输入航班号：");
            scanf(" %s", &fno);
            sortByFlightNo(flightRecords,numRecords);
            int i = binarySearch(flightRecords,numRecords,fno);
            if(i==-1){
                printf("无该航班");
            }else{
                displayFlightRecord(flightRecords[i]);
            }
            system("pause");
            break;
        }
        case 3:{
            int sort;
            system("cls");
            printf("请选择查询内容：\n");
            printf("------[1]起点--------\n");
            printf("------[2]终点--------\n");
            printf("------[3]起飞--------\n");
            printf("------[4]到达--------\n");
            printf("------[5]型号--------\n");
            printf("------[6]票价--------\n");
            printf("------[7]退出--------\n");
            scanf("%d", &sort);
            switch (sort) {
                case 1:{
                    printf("请输入起点：");
                    scanf(" %s", &depart);
                    for (int i = 0; i < numRecords; ++i) {
                        if(strcmp(flightRecords[i].departure, depart) == 0){
                            displayFlightRecord(flightRecords[i]);
                        }
                    }
                    system("pause");
                    break;
                }
                case 2:{
                    printf("请输入终点：");
                    scanf(" %s", &des);
                    for (int i = 0; i < numRecords; ++i) {
                        if(strcmp(flightRecords[i].destination, des) == 0){
                            displayFlightRecord(flightRecords[i]);
                        }
                    }
                    system("pause");
                    break;
                }
                case 3:{
                    printf("请输入起飞时间：");
                    scanf(" %s", &departTime);
                    for (int i = 0; i < numRecords; ++i) {
                        if(strcmp(flightRecords[i].departureTime, departTime)
== 0){
                            displayFlightRecord(flightRecords[i]);
                        }
```

```
                    }
                    system("pause");
                    break;
                }
                case 4:{
                    printf("请输入到达时间：");
                    scanf(" %s", &arriTime);
                    for (int i = 0; i < numRecords; ++i) {
                        if(strcmp(flightRecords[i].arrivalTime, arriTime) ==
0){
                            displayFlightRecord(flightRecords[i]);
                        }
                    }
                    system("pause");
                    break;
                }
                case 5:{
                    printf("请输入飞机型号：");
                    scanf(" %s", &airModel);
                    for (int i = 0; i < numRecords; ++i) {
                        if(strcmp(flightRecords[i].aircraftModel, airModel) ==
0){
                            displayFlightRecord(flightRecords[i]);
                        }
                    }
                    system("pause");
                    break;
                }
                case 6:{
                    printf("请输入票价：");
                    scanf("%f", &tkPrice);
                    for (int i = 0; i < numRecords; ++i) {
                        if(flightRecords[i].ticketPrice==tkPrice){
                            displayFlightRecord(flightRecords[i]);
                        }
                    }
                    system("pause");
                    break;
                }
                default:{
                    system("pause");
                    break;
                }
            }
        }
        default: {
            return 0;
        }
    }
}
```

4. 内部排序

```
#include <cstdio>
#include <cstdlib>
#include <ctime>
#include <iomanip>
#include <iostream>

int* generateRandomArray(int size) {
    int* arr = (int*)malloc(size * sizeof(int));
    srand(time(NULL));
    for (int i = 0; i < size; i++) {
        arr[i] = rand() % 1000;
    }
    return arr;
}

void swap(int* a, int* b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

void bubbleSort(int arr[], int size, int* comparisons, int* moves) {
    for (int i = 0; i < size - 1; i++) {
        for (int j = 0; j < size - i - 1; j++) {
            (*comparisons)++;
            if (arr[j] > arr[j+1]) {
                (*moves)++;
                swap(&arr[j], &arr[j+1]);
            }
        }
    }
}

void selectionSort(int arr[], int size, int* comparisons, int* moves) {
    for (int i = 0; i < size - 1; i++) {
        int minIndex = i;
        for (int j = i + 1; j < size; j++) {
            (*comparisons)++;
            if (arr[j] < arr[minIndex]) {
                minIndex = j;
            }
        }
        if (minIndex != i) {
            (*moves)++;
            swap(&arr[i], &arr[minIndex]);
        }
    }
}

void insertionSort(int arr[], int size, int* comparisons, int* moves) {
    for (int i = 1; i < size; i++) {
        int key = arr[i];
        int j = i - 1;
        while (j >= 0 && arr[j] > key) {
            (*comparisons)++;
            (*moves)++;
            arr[j+1] = arr[j];
            j--;
        }
        (*comparisons)++;
        arr[j+1] = key;
    }
}

int partition(int arr[], int low, int high, int* comparisons, int* moves) {
    int pivot = arr[high];
```

```
    int i = low - 1;
    for (int j = low; j <= high - 1; j++) {
        (*comparisons)++;
        if (arr[j] < pivot) {
            i++;
            (*moves)++;
            swap(&arr[i], &arr[j]);
        }
    }
    (*moves)++;
    swap(&arr[i+1], &arr[high]);
    return (i + 1);
}

void quickSort(int arr[], int low, int high, int* comparisons, int* moves) {
    if (low < high) {
        int pi = partition(arr, low, high, comparisons, moves);
        quickSort(arr, low, pi - 1, comparisons, moves);
        quickSort(arr, pi + 1, high, comparisons, moves);
    }
}

void shellSort(int arr[], int size, int* comparisons, int* moves) {
    for (int gap = size / 2; gap > 0; gap /= 2) {
        for (int i = gap; i < size; i++) {
            int key = arr[i];
            int j = i;
            while (j >= gap && arr[j - gap] > key) {
                (*comparisons)++;
                (*moves)++;
                arr[j] = arr[j - gap];
                j -= gap;
            }
            (*comparisons)++;
            arr[j] = key;
        }
    }
}

void heapify(int arr[], int n, int i, int* comparisons, int* moves) {
    int largest = i;
    int left = 2 * i + 1;
    int right = 2 * i + 2;
    if (left < n && arr[left] > arr[largest]) {
        *comparisons += 1;
        largest = left;
    }
    if (right < n && arr[right] > arr[largest]) {
        *comparisons += 1;
        largest = right;
    }
    if (largest != i) {
        *moves += 3;
        swap(&arr[i], &arr[largest]);
        heapify(arr, n, largest, comparisons, moves);
    }
}

void heapSort(int arr[], int n, int* comparisons, int* moves) {
    for (int i = n / 2 - 1; i >= 0; i--) {
        heapify(arr, n, i, comparisons, moves);
    }
    for (int i = n - 1; i >= 0; i--) {
        *moves += 3;
        swap(&arr[0], &arr[i]);
        heapify(arr, i, 0, comparisons, moves);
    }
}
```

```
void merge(int* arr, int left, int mid, int right, int* compareCount, int*
moveCount) {
    int i = left;
    int j = mid + 1;
    int k = 0;
    int* temp = (int*)malloc((right - left + 1) * sizeof(int));
    while (i <= mid && j <= right) {
        *compareCount += 1;
        if (arr[i] <= arr[j]) {
            temp[k++] = arr[i++];
        } else {
            temp[k++] = arr[j++];
        }
        *moveCount += 1;
    }
    while (i <= mid) {
        temp[k++] = arr[i++];
        *moveCount += 1;
    }
    while (j <= right) {
        temp[k++] = arr[j++];
        *moveCount += 1;
    }
    for (int m = 0; m < k; m++) {
        arr[left + m] = temp[m];
        *moveCount += 1;
    }
    free(temp);
}

void mergeSort(int* arr, int left, int right, int* compareCount, int* moveCount) {
    if (left < right) {
        int mid = (left + right) / 2;
        mergeSort(arr, left, mid, compareCount, moveCount);
        mergeSort(arr, mid + 1, right, compareCount, moveCount);
        merge(arr, left, mid, right, compareCount, moveCount);
    }
}

struct cmp{
    double time=0;
    int cmpcnt=0;
    int movcnt=0;
};

int main(){
    int dataSize = 10000;
    int* arr;
    cmp cmpSort[7];
    //冒泡排序
    for (int i = 0; i < 5; i++) {
        arr = generateRandomArray(dataSize);
        int compareCount = 0;
        int moveCount = 0;
        clock_t start = clock();
        bubbleSort(arr, dataSize, &compareCount, &moveCount);
        clock_t end = clock();
        cmpSort[0].time=cmpSort[0].time+(double)(end - start) / CLOCKS_PER_SEC;
        cmpSort[0].cmpcnt=cmpSort[0].cmpcnt+compareCount;
        cmpSort[0].movcnt=cmpSort[0].movcnt+moveCount;
        free(arr);
    }
    cmpSort[0].time=cmpSort[0].time/5;
    cmpSort[0].cmpcnt=cmpSort[0].cmpcnt/5;
    cmpSort[0].movcnt=cmpSort[0].movcnt/5;

    //选择排序
```

```
for (int i = 0; i < 5; i++) {
    arr = generateRandomArray(dataSize);
    int compareCount = 0;
    int moveCount = 0;
    clock_t start = clock();
    selectionSort(arr, dataSize, &compareCount, &moveCount);
    clock_t end = clock();
    cmpSort[1].time=cmpSort[1].time+(double)(end - start) / CLOCKS_PER_SEC;
    cmpSort[1].cmpcnt=cmpSort[1].cmpcnt+compareCount;
    cmpSort[1].movcnt=cmpSort[1].movcnt+moveCount;
    free(arr);
}
cmpSort[1].time=cmpSort[1].time/5;
cmpSort[1].cmpcnt=cmpSort[1].cmpcnt/5;
cmpSort[1].movcnt=cmpSort[1].movcnt/5;

//直接选择排序
for (int i = 0; i < 5; i++) {
    arr = generateRandomArray(dataSize);
    int compareCount = 0;
    int moveCount = 0;
    clock_t start = clock();
    insertionSort(arr, dataSize - 1, &compareCount, &moveCount);
    clock_t end = clock();
    cmpSort[2].time=cmpSort[2].time+(double)(end - start) / CLOCKS_PER_SEC;
    cmpSort[2].cmpcnt=cmpSort[2].cmpcnt+compareCount;
    cmpSort[2].movcnt=cmpSort[2].movcnt+moveCount;
    free(arr);
}
cmpSort[2].time=cmpSort[2].time/5;
cmpSort[2].cmpcnt=cmpSort[2].cmpcnt/5;
cmpSort[2].movcnt=cmpSort[2].movcnt/5;

//快速排序
for (int i = 0; i < 5; i++) {
    arr = generateRandomArray(dataSize);
    int compareCount = 0;
    int moveCount = 0;
    clock_t start = clock();
    quickSort(arr,0, dataSize - 1, &compareCount, &moveCount);
    clock_t end = clock();
    cmpSort[3].time=cmpSort[3].time+(double)(end - start) / CLOCKS_PER_SEC;
    cmpSort[3].cmpcnt=cmpSort[3].cmpcnt+compareCount;
    cmpSort[3].movcnt=cmpSort[3].movcnt+moveCount;
    free(arr);
}
cmpSort[3].time=cmpSort[3].time/5;
cmpSort[3].cmpcnt=cmpSort[3].cmpcnt/5;
cmpSort[3].movcnt=cmpSort[3].movcnt/5;

//希尔排序
for (int i = 0; i < 5; i++) {
    arr = generateRandomArray(dataSize);
    int compareCount = 0;
    int moveCount = 0;
    clock_t start = clock();
    shellSort(arr, dataSize, &compareCount, &moveCount);
    clock_t end = clock();
    cmpSort[4].time=cmpSort[4].time+(double)(end - start) / CLOCKS_PER_SEC;
    cmpSort[4].cmpcnt=cmpSort[4].cmpcnt+compareCount;
    cmpSort[4].movcnt=cmpSort[4].movcnt+moveCount;
    free(arr);
}
cmpSort[4].time=cmpSort[4].time/5;
cmpSort[4].cmpcnt=cmpSort[4].cmpcnt/5;
cmpSort[4].movcnt=cmpSort[4].movcnt/5;

//堆排序
```

```
    for (int i = 0; i < 5; i++) {
        arr = generateRandomArray(dataSize);
        int compareCount = 0;
        int moveCount = 0;
        clock_t start = clock();
        heapSort(arr, dataSize, &compareCount, &moveCount);
        clock_t end = clock();
        cmpSort[5].time=cmpSort[5].time+(double)(end - start) / CLOCKS_PER_SEC;
        cmpSort[5].cmpcnt=cmpSort[5].cmpcnt+compareCount;
        cmpSort[5].movcnt=cmpSort[5].movcnt+moveCount;
        free(arr);
    }
    cmpSort[5].time=cmpSort[5].time/5;
    cmpSort[5].cmpcnt=cmpSort[5].cmpcnt/5;
    cmpSort[5].movcnt=cmpSort[5].movcnt/5;

    // 归并排序
    for (int i = 0; i < 5; i++) {
        arr = generateRandomArray(dataSize);
        int compareCount = 0;
        int moveCount = 0;
        clock_t start = clock();
        mergeSort(arr, 0, dataSize - 1, &compareCount, &moveCount);
        clock_t end = clock();
        cmpSort[6].time=cmpSort[6].time+(double)(end - start) / CLOCKS_PER_SEC;
        cmpSort[6].cmpcnt=cmpSort[6].cmpcnt+compareCount;
        cmpSort[6].movcnt=cmpSort[6].movcnt+moveCount;
        free(arr);
    }
    cmpSort[6].time=cmpSort[6].time/5;
    cmpSort[6].cmpcnt=cmpSort[6].cmpcnt/5;
    cmpSort[6].movcnt=cmpSort[6].movcnt/5;
    std::cout<<std::setw(14)<<" ";
    std::cout<<"起泡排序"<<std::setw(13)<<"直接排序"
    <<std::setw(14)<<"选择排序"<<std::setw(14)<<"快速排序"
    <<std::setw(13)<<"希尔排序"<<std::setw(13)<<"堆排序"<<std::setw(14)<<"归并排序
";
    printf("\n 排序时间：");
    for (int i = 0; i < 7; ++i) {
        std::cout<<std::setw(12)<<cmpSort[i].time;
    }
    printf("\n 比较次数：");
    for (int i = 0; i < 7; ++i) {
        std::cout<<std::setw(12)<<cmpSort[i].cmpcnt;
    }
    printf("\n 移动次数：");
    for (int i = 0; i < 7; ++i) {
        std::cout<<std::setw(12)<<cmpSort[i].movcnt;
    }
}
```

5.校园导游咨询

```
#include <cstdio>
#include <string>
#define MAX_NODES 11

typedef struct {
    char name[50];
    std::string description;
} Node;

int adjacencyMatrix[MAX_NODES][MAX_NODES]={
        {0,1,2,3,0x3f,0x3f,0x3f,0x3f,0x3f,0x3f,0x3f},
        {1,0,1,0x3f,0x3f,0x3f,0x3f,0x3f,0x3f,0x3f,0x3f},
        {2,1,0,3,0x3f,0x3f,0x3f,5,0x3f,7,0x3f},
        {3,0x3f,3,0,4,0x3f,0x3f,0x3f,0x3f,0x3f,0x3f},
        {0x3f,0x3f,0x3f,4,0,6,6,0x3f,0x3f,5,0x3f},
        {0x3f,0x3f,0x3f,0x3f,6,0,0x3f,0x3f,0x3f,0x3f,0x3f},
        {0x3f,0x3f,0x3f,0x3f,6,0x3f,0,0x3f,0x3f,0x3f,0x3f},
        {0x3f,0x3f,5,0x3f,0x3f,0x3f,0x3f,0,2,5,0x3f},
        {0x3f,0x3f,0x3f,0x3f,0x3f,0x3f,0x3f,2,0,0x3f,6},
        {0x3f,0x3f,7,0x3f,5,0x3f,0x3f,5,0x3f,0,0x3f},
        {0x3f,0x3f,0x3f,0x3f,0x3f,0x3f,0x3f,0x3f,6,2,0}
};

Node nodes[MAX_NODES]={
        {"交通大楼","全称为现代交通中心，其中主要为机汽、轨道、电气三个学院的实验
室，以及行政办公场所。\n"},
        {"北门","上海市松江区广富林路 2793 号。\n"},
        {"实训中心","工程实训中心所属楼，满足学生日常使用机房、实践操作类课程的功
能。\n"},
        {"图书馆","其建筑面积为 27000 余平方，实行全开架借阅和计算机管理，共有 7 个借
阅合一的专业阅览室及 1 个地下书库。\n"},
        {"艺术楼","纺织服装学院、艺术设计学院所属楼。\n"},
        {"二食堂","离宿舍区最近最大的食堂，包括清真食堂、飞行食堂。\n"},
        {"南门","上海市松江区文汇路 800 弄。\n"},
        {"行政楼","俗称麻将楼，有一个 19 层楼高的主楼和 3 栋裙楼，学校大部分的办公部
门都在此楼。\n"},
        {"东门","上海市松江区龙腾路 333 号，为学校正门。\n"},
        {"教学楼","共有 A~F6 栋楼，承担大部分日常教学工作。\n"},
        {"一食堂","距离教学楼最近的食堂，包括了教工食堂。\n"}
};

bool visited[MAX_NODES];
int dist[MAX_NODES];
int last[MAX_NODES];

typedef struct {
    int queue[MAX_NODES];
    int front, rear;
} Queue;

void initQueue(Queue* q) {
    q->front = 0;
    q->rear = -1;
}

bool isQueueEmpty(Queue* q) {
    return (q->front > q->rear);
}

void enqueue(Queue* q, int node) {
    q->queue[++(q->rear)] = node;
}

int dequeue(Queue* q) {
    return q->queue[(q->front)++];
}
```

```
int breadthFirstSearch(int eNode, int numNodes) {
    Queue q;
    initQueue(&q);
    printf("BFS: ");
    visited[0] = true;
    enqueue(&q, 0);
    while (!isQueueEmpty(&q)) {
        int currentNode = dequeue(&q);
        printf("%d ", currentNode);
        if(currentNode==eNode) return currentNode;
        for (int i = 0; i < numNodes; i++) {
            if (adjacencyMatrix[currentNode][i]!=0x3f && !visited[i]) {
                enqueue(&q, i);
                visited[i] = true;
            }
        }
    }
    return -1;
}

void depthFirstSearch(int startNode, int eNode, int numNodes) {
    visited[startNode] = true;
    printf("%d ", startNode);
    for (int i = 0; i < numNodes; i++) {
        if (adjacencyMatrix[startNode][i]!=0x3f && !visited[i]) {
            depthFirstSearch(i, eNode, numNodes);
        }
    }

}

void dijkstra(int start, int end) {
    for (int i = 0; i < MAX_NODES; i++) {
        dist[i] = 0x3f;
        visited[i] = false;
    }

    dist[start] = 0;

    for (int count = 0; count < MAX_NODES - 1; count++) {
        int minDist = 0x3f;
        int minIndex = -1;

        for (int i = 0; i < MAX_NODES; i++) {
            if (!visited[i] && dist[i] < minDist) {
                minDist = dist[i];
                minIndex = i;
            }
        }

        if (minIndex == -1) {
            break;
        }

        visited[minIndex] = true;

        // 更新距离数组
        for (int i = 0; i < MAX_NODES; i++) {
            if (!visited[i] && adjacencyMatrix[minIndex][i] != 0x3f
                && dist[minIndex] + adjacencyMatrix[minIndex][i] < dist[i]) {
                dist[i] = dist[minIndex] + adjacencyMatrix[minIndex][i];
                last[i] = minIndex;
            }
        }
    }
    printf("从 %s 到 %s 的最短距离: %d\n", nodes[end].name, nodes[start].name,
dist[end]);
```

```
}

int way(int start, int end){
    if(last[end] == start){
        printf("%s ", nodes[start].name);
        return start;
    }else{
        printf("%s ", nodes[last[end]].name);
        return way(start, last[end]);
    }
}

void mapshow(){
    printf("   1.北门\n\n");
    printf("0.交通楼 2.实训楼 8.行政楼 9.东门\n\n");
    printf("   3.图书馆\n\n");
    printf("   4.艺术楼    9.教学楼  10.一食堂\n\n");
    printf("5.二食堂\n\n");
    printf("        6.南门\n\n");
}

int main() {
    while (true) {
        system("cls");
        mapshow();
        int choice;
        printf("-------[1]节点查询-------\n");
        printf("-------[2]节点导航-------\n");
        printf("-------[3]退出地图-------\n");
        scanf("%d", &choice);
        switch (choice) {
            case 1: {
                printf("输入要查询的节点序号：");
                int eNode;
                scanf("%d", &eNode);

                for (int i = 0; i < MAX_NODES; i++) {
                    visited[i] = false;
                }
                int bfs = breadthFirstSearch(eNode, MAX_NODES);

                for (int i = 0; i < MAX_NODES; i++) {
                    visited[i] = false;
                }

                printf("\nDFS: ");
                depthFirstSearch(0, eNode, MAX_NODES);

                if (bfs > 10 || bfs < 0) {
                    printf("无该节点");
                }
                else {
                    printf("\n%d 号位置：", bfs);
                    printf("%s ", nodes[bfs].name);
                    printf("%s", nodes[bfs].description.c_str());
                }
                system("pause");
                break;
            }
            case 2: {
                printf("\n 输入导航起点 终点序号：");
                int startNode, endNode;
                scanf("%d %d", &startNode, &endNode);
                dijkstra(endNode, startNode);
                printf("从 %s 到 %s 的最短路径: ", nodes[startNode].name,
nodes[endNode].name);
                printf("%s ", nodes[startNode].name);
                way(endNode, startNode);
```

```
                printf("\n");
                system("pause");
                break;
            }
            default: {
                system("cls");
                return 0;
            }
        }
    }
}
```

6.哈希表的应用

```
#include <cstdio>
#include <cstring>
#include <fstream>
#define TABLE_SIZE 10

typedef struct {
    char name[50] = "-1";
    char address[100] = "-1";
} Contact;

typedef struct {
    char phoneNum[20] = "-1";
    Contact contact;
} HashNode;

void initHash(HashNode hashTable[]) {
    for (int i = 0; i < TABLE_SIZE; i++) {
        strcpy(hashTable[i].phoneNum, "-1");
    }
}

int hashFunction(char phoneNum[]) {
    int sum = 0;
    for (int i = 0; i < strlen(phoneNum); i++) {
        sum += phoneNum[i];
    }
    return sum % TABLE_SIZE;
}

void insertContact(HashNode hashTable[], char phoneNum[], char name[], char
address[]) {
    int index = hashFunction(phoneNum);
    while (strcmp(hashTable[index].phoneNum, "-1") != 0) {
        index = (index + 1) % TABLE_SIZE;
    }

    strcpy(hashTable[index].phoneNum, phoneNum);
    strcpy(hashTable[index].contact.name, name);
    strcpy(hashTable[index].contact.address, address);
}

void save(HashNode hashTable[]){
    std::ofstream ofs;
    ofs.open("contacts.txt", std::ios::out);
    for (int i = 0; i < TABLE_SIZE; i++)
    {
        ofs<<hashTable[i].phoneNum<<"\t" <<
        hashTable[i].contact.name<<"\t" <<
        hashTable[i].contact.address<<std::endl;

    }
    ofs.close();
}

void load(HashNode* hashTable){
    std::ifstream ifs;
    ifs.open("contacts.txt", std::ios::in);
    if (ifs.is_open())
    {
        printf("成功读取通讯录，请您选择：\n");
        for (int i = 0; i < TABLE_SIZE; i++)
        {
            ifs>>hashTable[i].phoneNum
            >>hashTable[i].contact.name
            >>hashTable[i].contact.address;
        }
```

```
    }else{
        printf("未找到 contacts.txt");
    }
    ifs.close();

}

void findContact(HashNode hashTable[], char phoneNum[]) {
    int index = hashFunction(phoneNum);
    while (strcmp(hashTable[index].phoneNum, phoneNum) != 0) {
        index = (index + 1) % TABLE_SIZE;
        if (strcmp(hashTable[index].phoneNum, "") == 0 || index ==
hashFunction(phoneNum)) {
            printf("未找到该手机号码对应的通讯者信息。\n");
            return;
        }
    }

    printf("姓名: %s\n", hashTable[index].contact.name);
    printf("地址: %s\n", hashTable[index].contact.address);
}

int main() {
    HashNode hashTable[TABLE_SIZE];
    initHash(hashTable);

    while (true) {
        system("cls");
        load(hashTable);
        char name[50];
        char address[100];
        char phoneNum[20];
        int choice;
        printf("-------[1]录入好友-------\n");
        printf("-------[2]查询好友-------\n");
        printf("-------[3]退出系统-------\n");
        scanf("%d", &choice);
        switch (choice) {
            case 1: {
                printf("请输入手机号: ");
                scanf(" %s", &phoneNum);
                printf("请输入姓名: ");
                scanf(" %s", &name);
                printf("请输入地址: ");
                scanf(" %s", &address);
                insertContact(hashTable, phoneNum, name, address);

                save(hashTable);
                system("pause");
                break;
            }
            case 2: {
                printf("请输入手机号: ");
                scanf(" %s", &phoneNum);
                findContact(hashTable, phoneNum);
                printf("\n");
                system("pause");
                break;
            }
            default: {
                return 0;
            }
        }
    }
}
```

7.哈夫曼编码/译码器

```
#include <cstdio>
#include <conio.h>
#include <cstdlib>
#include <iostream>

#define MAXVAL 10000.0
struct HuffmanTreeNode{
    char data;
    double weight;
    int parent,lchild,rchild;
};
struct HuffmanTree{
    HuffmanTreeNode *node;
    int n;
};
struct Codetype{
    char *bits;
    int start;
};
void SortHufmtree(HuffmanTree *tree){
    int i,j,k;
    HuffmanTreeNode temp;
    if (tree==NULL)
        return;
    for (i=0;i<tree->n;i++)
    {
        k=i;
        for(j=i+1;j<tree->n;j++)
            if (tree->node[j].weight>tree->node[k].weight)
                k=j;
        if (k!=i)
        {
            temp=tree->node[i];
            tree->node[i]=tree->node[k];
            tree->node[k]=temp;
        }
    }
}
Codetype* HuffmanCode(HuffmanTree *tree){
    int i,j,p,k;
    Codetype *code;
    if (tree==NULL)
        return NULL;
    code=(Codetype*)malloc(tree->n*sizeof(Codetype));
    for (i=0;i<tree->n;i++)
    {
        printf("%c ",tree->node[i].data);
        code[i].bits=(char*)malloc(tree->n*sizeof(char));
        code[i].start=tree->n-1;
        j=i;
        p=tree->node[i].parent;
        while(p!=-1){
            if (tree->node[p].lchild==j)
                code[i].bits[code[i].start]='0';
            else
                code[i].bits[code[i].start]='1';

            code[i].start--;
            j=p;
            p=tree->node[p].parent;
        }
        for (k=code[i].start+1;k<tree->n;k++)
            printf("%c",code[i].bits[k]);
        printf("\n");
    }
    return code;
```

```
}
HuffmanTree* BuildHuffmanTree(HuffmanTree *tree){
    int i,j,p1,p2,m;
    float small1,small2;
    m=2*(tree->n)-1;
    for (i=tree->n;i<m;i++)
    {
        tree->node[i].parent=-1;
        tree->node[i].lchild=-1;
        tree->node[i].rchild=-1;
        tree->node[i].weight=0.0;
    }
    for (i=tree->n;i<m;i++)
    {
        small1=small2=MAXVAL;
        for (j=0;j<=i-1;j++)
        {
            if (tree->node[j].parent==-1 && tree->node[j].weight<=small1)
            {
                small1=tree->node[j].weight;
                p1=j;
            }
        }
        for (j=0;j<=i-1;j++)
        {
            if (tree->node[j].parent==-1 && tree->node[j].weight<=small2 &&
(j!=p1))
            {
                small2=tree->node[j].weight;
                p2=j;
            }
        }
        tree->node[p1].parent=tree->node[p2].parent=i;
        tree->node[i].weight=tree->node[p1].weight+tree->node[p2].weight;

        tree->node[i].lchild=p1;
        tree->node[i].rchild=p2;
    }
    return tree;
}
HuffmanTree* CreateHuffmanTreeFromSourceFile(){
    FILE *textfile,*datafile;
    char ch;
    int i,j=0,n=0;
    float count[128];
    HuffmanTree *tree;

    if ((textfile=fopen("1.txt","r"))==NULL){
        printf("\n找不到1.txt");
        return NULL;
    }
    for(i=0;i<128;i++)
        count[i]=0;

    ch=fgetc(textfile);
    while(!feof(textfile)){
        for(i=0;i<128;i++)
            if (ch==char(i)){
                count[i]++;
                break;
            }
        ch=fgetc(textfile);
    }

    datafile=fopen("data.txt","w");
    for (i=0;i<128;i++)
        if (count[i]!=0)
            n++;
```

```
    fprintf(datafile,"%d\n",n);
    tree=(HuffmanTree*)malloc(sizeof(HuffmanTree));
    tree->node=(HuffmanTreeNode*)malloc((2*n-1)*sizeof(HuffmanTreeNode));

    tree->n=n;

    printf("\n 源文件的字符集及其权值信息如下：\n");

    for(i=0;i<128;i++)
        if (count[i]!=0)
        {
            fprintf(datafile,"%c  %.0f\n",char(i),count[i]);
            printf("%c %.0f\n",char(i),count[i]);
            tree->node[j].data=char(i);
            tree->node[j].weight=count[i];
            tree->node[j].parent=-1;
            tree->node[j].lchild=-1;
            tree->node[j].rchild=-1;
            j++;
        }
    SortHufmtree(tree);
    BuildHuffmanTree(tree);
    fclose(textfile);
    fclose(datafile);
    printf("\n 哈夫曼树建立完毕，已将权值信息保存至 data.txt\n");
    return tree;
}

HuffmanTree* CreateHuffmanTreeByInput(){
    int n;
    HuffmanTree *tree;
    int i;
    FILE *datafile;
    tree=(HuffmanTree*)malloc(sizeof(HuffmanTree));
    datafile=fopen("data.txt","w");

    printf("请输入字符数：");
    scanf("%d",&n);
    if (n<=0)
    {
        printf("\n 您的输入有误。\n");
        return NULL;
    }
    tree->node=(HuffmanTreeNode*)malloc((2*n-1)*sizeof(HuffmanTreeNode));
    tree->n=n;

    for (i=0;i<n;i++)
    {
        tree->node[i].parent=-1;
        tree->node[i].lchild=-1;
        tree->node[i].rchild=-1;
        tree->node[i].weight=0.0;
    }
    fprintf(datafile,"%d\n",n);
    for (i=0;i<n;i++)
    {
        printf("\n 输入第%d 个字符及其权值：",i+1);
        std::cin>>tree->node[i].data;
        std::cin>>tree->node[i].weight;
        fprintf(datafile,"%c  %.3f\n",tree->node[i].data,tree->node[i].weight);
    }
    fclose(datafile);

    SortHufmtree(tree);
    BuildHuffmanTree(tree);
    printf("\n 哈夫曼树建立完毕，已将权值信息保存至 data.txt\n");
    return tree;
}
```

```
HuffmanTree* CreateHuffmanTreeFromDataFile(){
    FILE *datafile;
    int i,n;
    HuffmanTree *tree;
    if ((datafile=fopen("data.txt","r"))==NULL){
        printf("\n 哈夫曼树尚未建立\n");
        return NULL;
    }

    fscanf(datafile,"%d",&n);
    fgetc(datafile);
    tree=(HuffmanTree*)malloc(sizeof(HuffmanTree));
    tree->node=(HuffmanTreeNode*)malloc((2*n-1)*sizeof(HuffmanTreeNode));
    tree->n=n;
    for (i=0;i<n;i++){
        tree->node[i].data=fgetc(datafile);
        fscanf(datafile,"%f\n",&tree->node[i].weight);
        tree->node[i].parent=-1;
        tree->node[i].lchild=-1;
        tree->node[i].rchild=-1;
    }
    fclose(datafile);

    SortHufmtree(tree);
    BuildHuffmanTree(tree);
    return tree;
}


HuffmanTree* Encoding(HuffmanTree *tree){
    FILE *textfile,*codefile;
    char ch;
    int i,j;
    Codetype *code;

    if (tree==NULL)
    {
        tree=CreateHuffmanTreeFromDataFile();
        if (tree==NULL)
            return NULL;
    }

    if ((textfile=fopen("1.txt","r"))==NULL){
        printf("\n 找不到文件 1.txt\n");
        return NULL;
    }

    printf("\n 源文件的原文如下：\n");
    ch=fgetc(textfile);
    while(!feof(textfile)){
        printf("%c",ch);
        ch=fgetc(textfile);
    }

    printf("\n 字符集的哈夫曼编码如下：\n");
    code=HuffmanCode(tree);
    codefile=fopen("CodeFile.txt","w");
    fseek(textfile,0,SEEK_SET);
    ch=fgetc(textfile);
    while (!feof(textfile))
    {
        for(i=0;i<tree->n;i++)
            if (ch==tree->node[i].data){
                for(j=code[i].start+1;j<tree->n;j++)
                    fputc(code[i].bits[j],codefile);
                break;
            }
        if (i==tree->n)
```

```
        {
            fclose(codefile);
            return NULL;
        }
        ch=fgetc(textfile);
    }
    fclose(codefile);

    printf("\n 编码成功,已将代码写入 CodeFile.txt,代码如下: \n");
    codefile=fopen("CodeFile.txt","r");
    ch=fgetc(codefile);
    while(!feof(codefile)){
        printf("%c",ch);
        ch=fgetc(codefile);
    }
    printf("\n");
    fclose(textfile);
    fclose(codefile);
    return tree;
}

void Decoding(HuffmanTree *tree){
    FILE *codefile,*decodefile;
    char ch;
    int i;

    if (tree==NULL){
        tree=CreateHuffmanTreeFromDataFile();
        if (tree==NULL)
            return ;
    }

    if ((codefile=fopen("CodeFile.txt","r"))==NULL){
        printf("\n 找不到文件 CodeFile.txt\n");
        return;
    }

    printf("\n 代码文件原文如下: \n");
    ch=fgetc(codefile);
    while(!feof(codefile)){
        printf("%c",ch);
        ch=fgetc(codefile);
    }

    decodefile=fopen("DecodeFile.txt","w");
    fseek(codefile,0,SEEK_SET);
    ch=fgetc(codefile);
    while(!feof(codefile))
    {
        for(i=2*tree->n-2;tree->node[i].lchild>=0 && tree->node[i].rchild>=0 &&
ch!=EOF;)
        {
            if(ch=='0')
                i = tree->node[i].lchild;
            else if(ch=='1')
                i = tree->node[i].rchild;
            else{
                printf("\n 存在异常字符%c，不能正确译码。\n",ch);
                return ;
            }
            if (i==-1){
                printf("\n 编码与当前哈夫曼树结构不相符，不能正确译码。\n",ch);
                fclose(decodefile);
                return;
            }
            ch=fgetc(codefile);
        }
        if (tree->node[i].lchild>=0 && tree->node[i].rchild>=0){
```

```
            printf("\n 代码文件编码内容不完整，不能完整译码。\n",ch);
            fclose(decodefile);
            return;
        }

        fputc(tree->node[i].data,decodefile);
    }
    fclose(decodefile);

    printf("\n\n 译码成功，译文已保存至 DecodeFile.txt\n");
    printf("译文如下：\n");
    decodefile=fopen("DecodeFile.txt","r");
    ch=fgetc(decodefile);
    while(!feof(decodefile)){
        printf("%c",ch);
        ch=fgetc(decodefile);
    }
    printf("\n");
    fclose(codefile);
    fclose(decodefile);
}

int main(){
    HuffmanTree *tree=NULL;
    int choice;
    while(true) {
        system("cls");
        printf("------[1]对文件进行编码--------\n");
        printf("------[2]对输入进行编码--------\n");
        printf("------[3]对数据进行译码--------\n");
        printf("------[4]退出哈夫曼系统--------\n");
        scanf("%d", &choice);
        printf("\n");
        switch (choice)
        {
            case 1:{
                tree=CreateHuffmanTreeFromSourceFile();
                tree=Encoding(tree);
                system("pause");
                break;
            }
            case 2:{
                tree=CreateHuffmanTreeByInput();
                tree=Encoding(tree);
                system("pause");
                break;
            }
            case 3:{
                Decoding(tree);
                system("pause");
                break;
            }
            case 4:{
                return 0;
            }
            default:{
                printf("无效的选项。\n");
                system("pause");
                break;
            }
        }
    }
}
```