**《数据结构》课程设计报告**

**学生姓名 杨 帆**

**班 级 计算机2003**

**学 号 201205130**

**指导老师 陈 宏 建**

**成 绩**

**时 间 2021.12.22～2021.12.30**

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 项目名称 | 考核比重 | 得分 |
| 1 | 方案设计 | 10% |  |
| 2 | 系统设计 | 50% |  |
| 3 | 设计验收 | 20% |  |
| 4 | 设计报告 | 20% |  |

**2022年 1月 3 日**

**《数据结构课程设计》任务书**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 课程编号 | 10130090 | 学分 | 1 | 周数 | 1周 |
| 学号 | 201205130 | 专业 | 计算机 | | |
| 姓名 | 杨帆 | 班级 | 计算机2003 | | |
| 目的与要求：  培养学生综合程序设计的能力，训练学生灵活应用所学数据结构知识，独立完成问题分析、总体设计、详细设计和编程实现等软件开发全过程的综合实践能力。巩固、深化学生的理论知识，提高编程水平，并在此过程中培养他们严谨的科学态度和良好的学习作风。为今后学习其他计算机课程打下基础。  能够根据计算机及应用领域复杂工程问题，选择适当的数据结构及其存储结构分析确定解决问题的算法，并能对其进行有效性和时空性能分析。  通过对计算机及应用领域的复杂工程的综合分析，利用数据结构的有关算法设计出复杂问题系统的解决方案并能对方案进行合理有效分析并得出结论。 | | | | | |
| 主要任务及具体要求：  任务：  设计并实现一个复杂问题的系统（根据学号顺序选两题完成，题目见报告），开发工具自选，并根据要求写出设计报告。  要求：  课程设计为学生提供了一个既动手又动脑，独立实践的机会，将书本上的理论知识和工作、生产实际有机地结合起来，从而锻炼学生分析问题、解决实际问题的能力，提高学生的编程序能力和创新意识。在处理每个题目时，要求从分析题目的需求入手，按设计抽象数据类型、构思算法、通过算法的设计实现抽象数据类型、编制上机程序和上机调试等若干步骤完成题目，最终写出完整的课程设计与程序分析报告。前期准备工作完备与否直接影响到后序上机调试工作的效率。  进程安排：   |  |  |  |  | | --- | --- | --- | --- | | 时间 | 内容 | 时间（天） | 授课形式 | | 1 | 课题讲解，方案设计 | 0.5 | 讲练结合 | | 2 | 系统设计 | 4 | 讲练结合 | | 3 | 设计验收 | 2 | 讲练结合 | | 4 | 设计报告 | 0.5 | 讲练结合 |   2021年12月22日 | | | | | |

**《数据结构课程设计》答辩记录表**

|  |  |  |  |
| --- | --- | --- | --- |
| **学号** |  | **专业** |  |
| **姓名** |  | **班级** |  |
| **答辩记录：**  **答辩人： 日期：** | | | |

**目录**

[一、课程设计题目描述 5](#_Toc91928671)

[1.1 问题一 5](#_Toc91928672)

[1.2 问题二 5](#_Toc91928673)

[二、课题数据结构与算法 6](#_Toc91928674)

[2.1 课题一数据结构与算法 6](#_Toc91928675)

[2.2 课题二数据结构与算法 7](#_Toc91928676)

[三、测试数据及运行结果 8](#_Toc91928677)

[3.1 课题一测试数据及运行结果 8](#_Toc91928678)

[3.2 课题二测试数据和运行结果 18](#_Toc91928679)

[四、体会与总结 23](#_Toc91928680)

[4.1 课题一的体会与总结 23](#_Toc91928681)

[4.2 课题二的体会与总结 23](#_Toc91928682)

[五、源程序 24](#_Toc91928683)

[5.1 课题一 24](#_Toc91928684)

[5.2 课题二 29](#_Toc91928685)

# 一、课程设计题目描述

## 1.1 问题一

**自来水管道架设问题**

【问题描述】

若要在扬州大学的八个居民区（A区、B区、C区、D区、E区、F区、G区、H区）之间架设自来水管道，如何以最低的经济代价架设这个自来水管道。

【基本要求】

（1）利用两种方法Prim算法和Kruskual算法生成自来水管道的架设方案。

（2）求出A居民区到其余各居民区的距离。

（3）将八个居民区设计为一个DAG图，求出一组拓扑排序序列和关键路径。

【测试数据】

分别对每种方法选定三组测试数据进行测试，验证程序的正确性。

## 1.2 问题二

**哈夫曼树与哈夫曼编码**

【问题描述】

创建一棵哈夫曼树并实现哈夫曼编码的输出。

【基本要求】

（1）输入一个文本，统计各字符出现的频度，输出结果 。

（2）使用字符出现的频度构造哈夫曼树 。

（3）输出各字符按上述频度所构造的哈夫曼编码 。

（4）输入一个由0和1组成的代码序列，翻译并输出与之对应的文本，若最后的代码子序列不能译为文本，则输出相关出错信息。

【测试数据】

分别对选定两组测试数据进行测试，验证程序的正确性。

# 二、课题数据结构与算法

## 2.1 课题一数据结构与算法

课题一的基本目标是在扬州大学的8个居民区（A~H）之间用最低的经济代价架设自来水管道。为了给出自来水管道的架设方案，我们得先对问题进行分析，选择合适的数据结构来表示居民区以反映其逻辑关系。

显然8个居民区是多对多的逻辑关系，我们可以用图这种数据结构来解决这个问题。由于自来水管道的两个端点是互通的，并无方向性的问题，并且居民区之间是有距离的，要考虑权值，所以针对自来水管道的架设问题，选择无向网这种特定的数据结构来表示逻辑关系，存储结构选择用邻接矩阵。针对基本要求的第三问，因为要设计成DGA图，所以选择用DGA网来设计居民区，存储结构选择用邻接表。

完成对数据结构的设计后，就要针对不同的问题给出不同的算法方案。

首先是自来水管道的架设方案。既然是自来水管道，那么就不能有居民区没有架设到自来水管道，也就是一个连通图。题目要求经济代价要最低，也就是铺设的自来水管道的距离要最小，转换到我们的数据结构中，就是边的权值之和要最小，我们可以用最小生成树的相关算法来解决这个问题。所以，我们可以用Prim算法和Kruskual算法给出最经济的自来水管道的架设方案。

问题2要求我们求出居民区A到其它居民区的距离，也就是顶点A和其它顶点之间都要找到一条最短的路径，使得这条路径的权值之和最小。这个问题我们可以使用图的最短路径相关算法，在本次课设中，我选用了Dijkstra算法来求得居民区A到其它各居民区的最短距离。

问题3要求我们将居民区设计成一个DAG图，求出一组拓扑排序和关键路径。这个问题的用意已经很明确，我们只要实现有向无环图的拓扑排序算法和利用拓扑排序算法的关键路径算法，求出相关结果即可。

以上就是对课题一的问题分析，以及根据分析选择的数据结构和相关算法。

## 2.2 课题二数据结构与算法

课题二的基本目标是根据用户输入的文本创建一个哈夫曼树,然后利用这个哈夫曼树对这个文本进行哈夫曼编码，以及将一串哈夫曼编码解码为文本。

用户输入的文本毫无疑问会使用字符串来存储，字符串读取后就要计算各个字符的频度，为了之后编解码方便，我选择用一个结构体数组来存储字符及其频度和哈夫曼编码的信息。

然后利用存储每个字符信息的结构体数组中的频度信息来构建一个哈夫曼树，这个哈夫曼树用三叉链表来存储，包括左孩子、右孩子和双亲三个指针域，除了指针域外，我定义的哈夫曼树结点还包括一个Int型的权重和一个标志位，标志位主要用于解决获取字符对应编码时权值重复的问题。在构建哈夫曼树的过程中，需要一个存放森林的HC List，我这里选择用一个元素是哈夫曼树结点的单链表来实现，之所以不选择数组来实现这个表是因为需要频繁的插入和删除。创建哈夫曼树的算法主要过程就是在HC List中选取权值最小的两颗森林，最为新结点的左右子树（左子树权值<右子树权值），然后将这个新结点插入HCList中，当HC List中还剩两个结点时，这两个结点的双亲就是哈夫曼树的根，至此创建哈夫曼树完毕。

在生成哈夫曼编码的时候，需要使用栈来保存结点来顺利地进行遍历，为了标记结点被访问的次数，栈中每个元素还设置了一个标志位。生成哈夫曼编码的算法类似于树的遍历，只不过每访问一个结点就要进行判断是否是叶子结点，如果是对应权值的叶子结点，就返回之前记录的由0，1组成的路径。为了解决字符频度相同造成哈夫曼一些权值相同的问题，我特地在每个结点设置了一个标志位来标识已经被访问过的叶子结点，并且设置了一个数组来记录已经被访问过的权值，以此来解决权值重复造成的哈夫曼编码相同的问题。

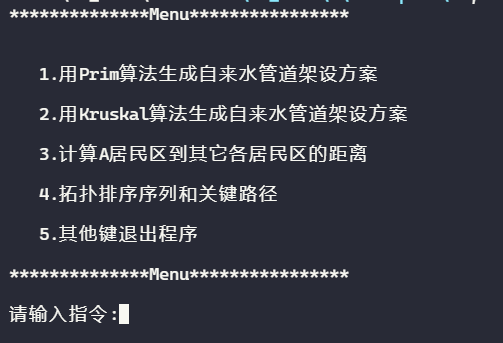
哈夫曼解码问题相对比较简单，只要把0，1组成的编码串输入哈夫曼树中遍历即可，如果没有多出的字符，则说明解码全部匹配；否则说明解码不成功，只有部分匹配。

以上就是对课题二的问题分析，以及根据分析选择的数据结构和相关算法。

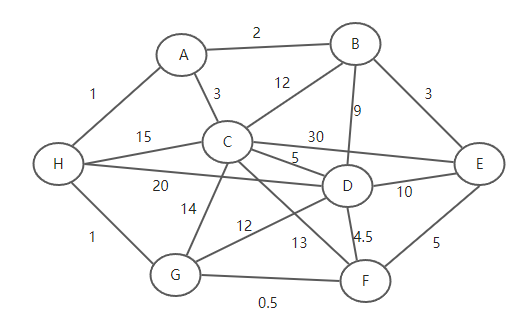
# 三、测试数据及运行结果

## 3.1 课题一测试数据及运行结果

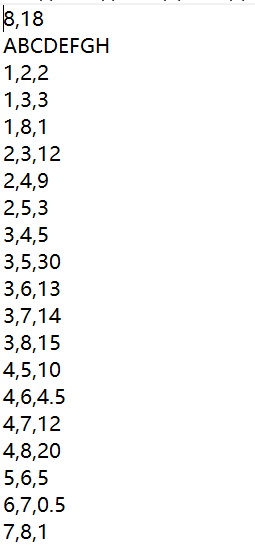
运行程序后，程序会显示出菜单：



我们首先输入指令1，测试第一个问题，第一组测试数据为：



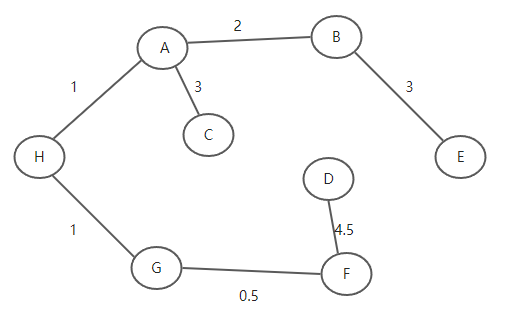
为了测试方便，我将数据写到了txt文件中，第一组测试数据的文本文件内容为：



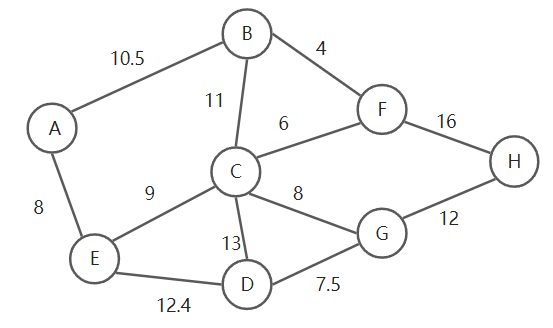
其中第一行是顶点数和弧数，第二行是顶点的编号，之后每行就是每条弧的起点，终点，权值。现在我们查看一下程序的运行结果：



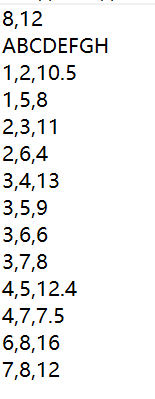
我们将此架设方案可视化，可得最小生成树：



可以看到此架设方案满足我们的要求，因此是正确的。接下来测试第二组数据，第二组数据的图为：



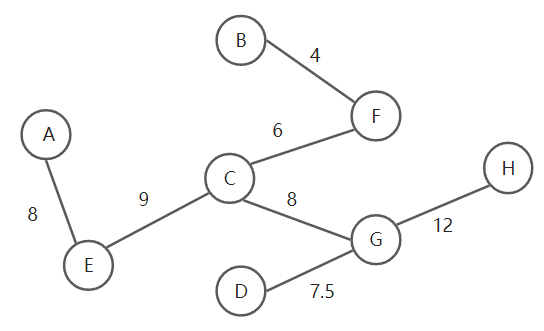
其对应的文本文件内容如下：



现在看一下测试数据2的运行结果：

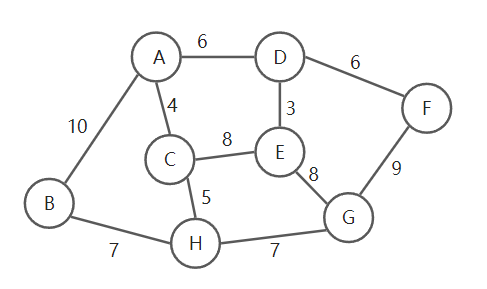


我们还是将结果可视化一下，可得最小生成树：

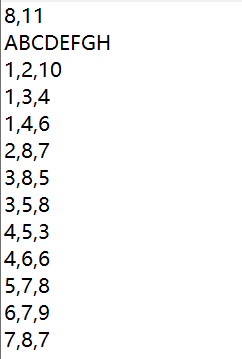


可以看到此架设方案满足我们的要求，因此是正确的。

现在开始测试第三组数据，第三组数据的图为：



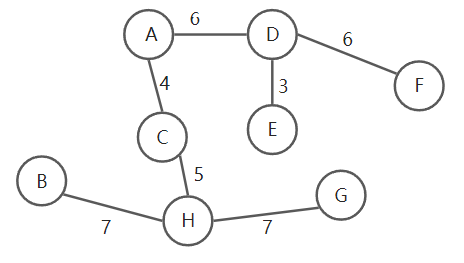
其对应的文本文件为：



现在看一下测试数据3的运行结果：



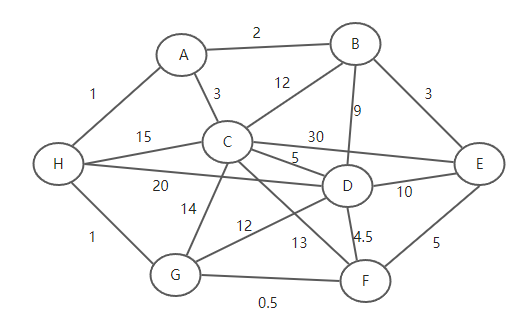
我们将结果可视化一下，可得最小生成树：



可以看到此架设方案满足我们的要求，因此是正确的。

上面三次测试都是对Prim算法的测试，下面三次测试是对Kruskual算法的测试，由于两者都是用于生成最小生成树，并且为了保证测试结果的可比较性，用的三组测试数据和Prim算法的测试数据是一样的。

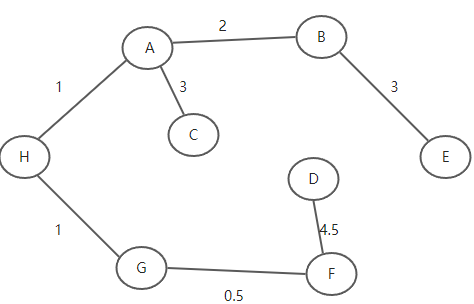
第一组测试数据的图为：



程序生成的架设方案为：

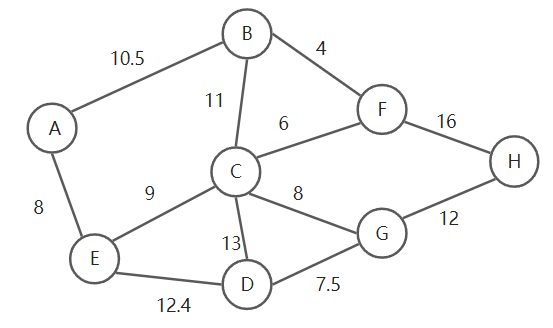


我们可以发现Kruskal算法生成的方案顺序和Prim算法生成的方案顺序是不同的，这是因为算法的原理不同，如果我们把此方案可视化为图，就会发现两者的结果是一样的：



由此可说明程序结果的正确性。

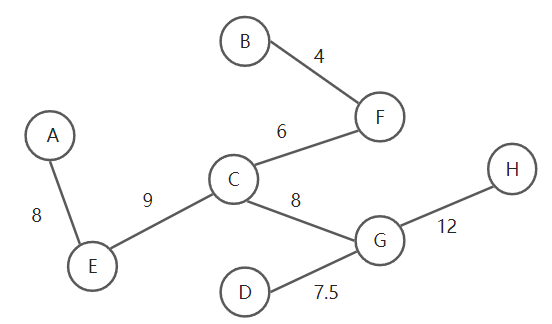
第二组测试数据为：



程序生成的架设方案为：

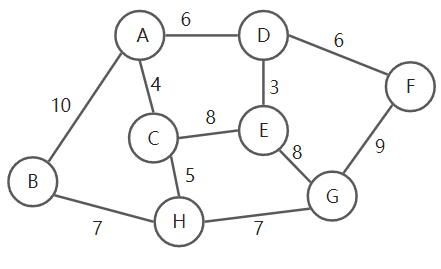


我们将此方案可视化一下，可得：



此方案的确是最经济的方案并且连通所有居民区，说明程序是正确的。

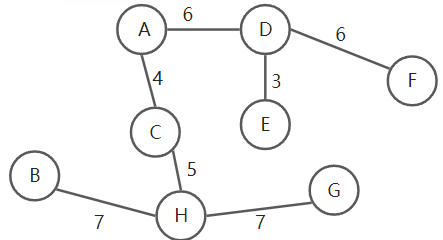
第三组测试数据为：



程序的运行结果为：



我们将此方案可视化一下，可得：

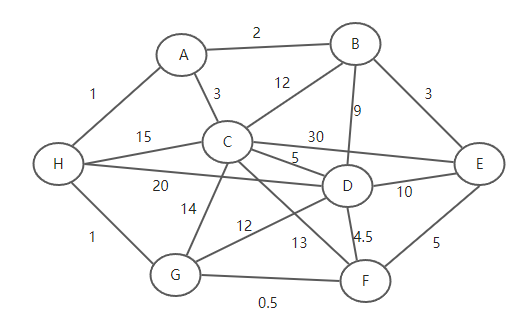


此方案的确是最经济的方案并且连通所有居民区，说明程序是正确的。

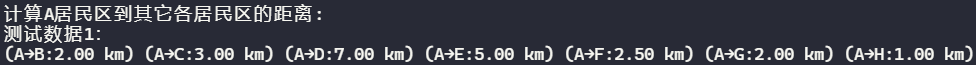
以上就是使用Prim算法和Kruskual算法来生成最小生成树以解决架设自来水管道的问题，每个算法的三组测试结果都是正确的。

下面开始测试求居民区A到其余各居民区的距离这一问题，其主要用到Dijkstra算法生成最短路径，上面测试架设自来水管道的三组例子在这里是可以通用的，所以我们采用上面三组测试示例来测试这个问题。

我们输入指令3，就可以进行测试。首先是第一组数据：

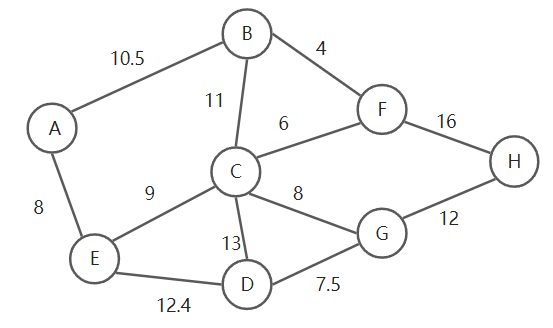


我们运行一下程序，程序输出如下：



因为实际中两个居民区的距离肯定是最短距离，这里我们就输出了居民区A到其它各居民区的最短距离，经过实际的计算，可以说明程序输出的距离都是正确的。

下面开始第二组测试，测试数据为：

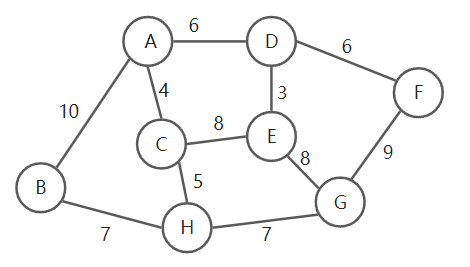


第二组的程序运行结果为：



同样，经过实际的计算，可以说明程序输出的距离都是正确的。

下面开始第三组测试，测试数据为：



第三组的程序运行结果为：

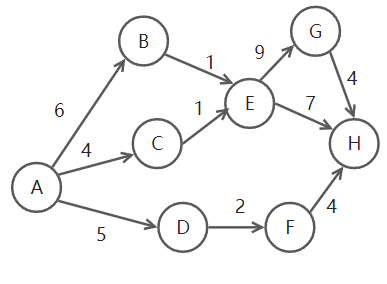


同样，经过实际的计算，可以说明程序输出的距离都是正确的。

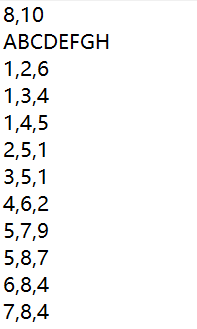
以上就是使用Dijkstra算法生成居民区A到其它各居民区的最短路径的测试结果，通过实际的计算可以发现每组数据的输出都是正确的。

下面开始测试第三个问题，由于要求居民区是DAG图，所以测试数据是三组DAG网，用于生产拓扑排序和关键路径。

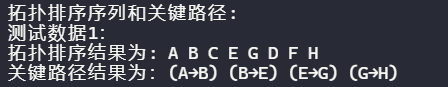
首先是第一组测试数据，其图为：



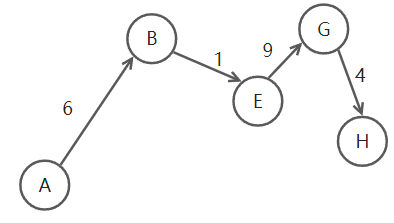
其对应的文本文件如下：



我们开始运行程序，对于一组测试数据，程序会输出一组拓扑排序和关键路径，第一组测试数据的运行结果为：

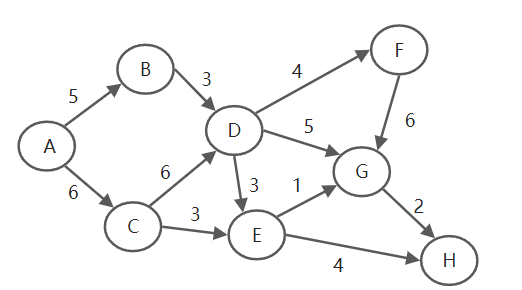


由于拓扑排序不为一，题目也只要求输出一组，我们根据此结果进行验证，最后的结果说明此拓扑排序是正确的，根据此拓扑排序生成的关键路径结果为：

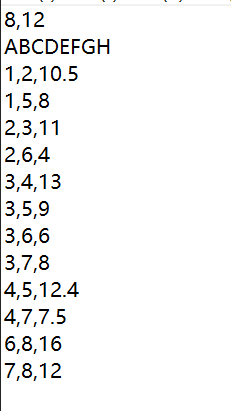


经过验证也是正确的。

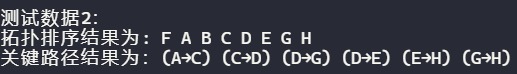
下面我们开始测试第二组数据，第二组数据的图如下：



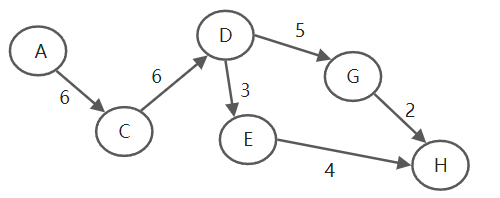
其文本文件如下：



我们开始运行程序，第二组测试数据的运行结果为：

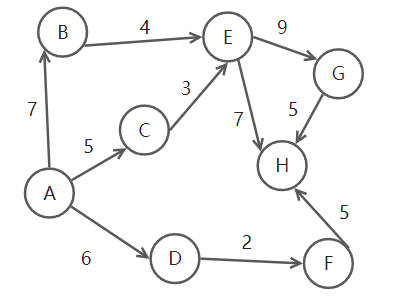


关键路径的图为：

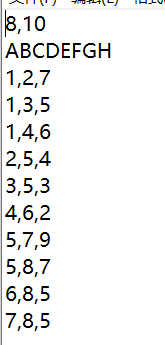


经过验证，拓扑排序和关键路径的结果都是正确的。

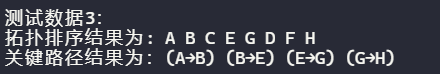
下面我们开始测试第三组数据，第三组数据的图如下：



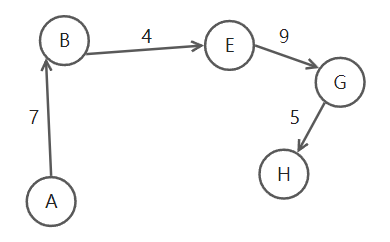
其文本文件如下：



我们开始运行程序，第三组测试数据的运行结果为：



关键路径的图为：



经过验证，拓扑排序和关键路径的结果都是正确的。

以上就是课题一的全部测试数据和运行结果。

## 3.2 课题二测试数据和运行结果

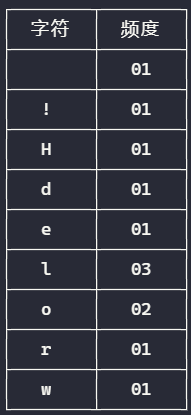
运行程序后，程序会提示我们输入文本：



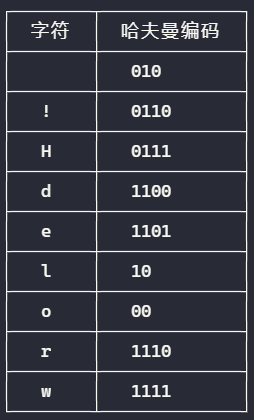
下面开始测试，首先我们输入第一组测试数据‘Hello world!’:



程序运行后，首先会输出字符串中各字符及其对应的频度：



我们计算一下频度，并且和频度表里的数据对照一下，可以验证此频度表的输出是正确的。接下来，程序会自动输出各字符对应的哈夫曼编码以及字符串对应的哈夫曼编码。各字符的哈夫曼编码以清晰的表格输出，输出结果如下：



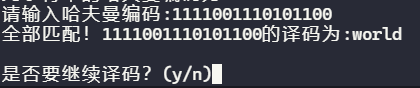
虽然哈夫曼编码不唯一，但是我们仍然可以验证此输出的正确性，可以发现频度高的字符的哈夫曼编码较短，频度低的字符的哈夫曼编码较长。字符串的哈夫曼编码输出如下：



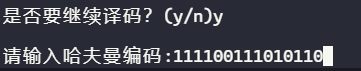
我们对照一下上表，可以验证其正确性。接下来，程序会提示我们输入要进行解码的哈夫曼码：



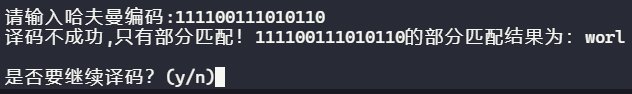
假如我们输入对应world的哈夫曼编码1111001110101100,程序应该输出world，我们来验证一下，在程序中输入1111001110101100:



可以看到程序提示匹配成功，并且给出了上述哈夫曼码的译码为world，是正确的，并且提示我们是否要继续译码，我们再输入一个不匹配的，比如去掉上述哈夫曼编码最后一个数字，然后输入：



程序的输出结果如下：



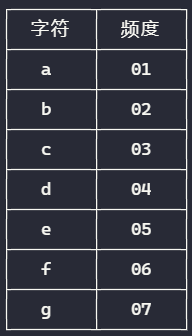
如果编码不匹配，程序就会输出“译码不成功，只有部分匹配！”的错误信息，并且会给出这种不匹配下的部分匹配结果。

我们输入n后，程序就会自动退出。

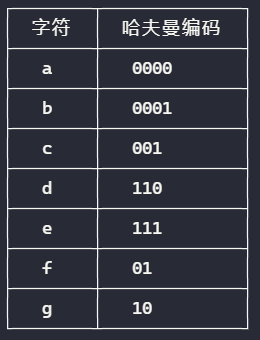
接下来开始第二组数据的测试，我们输入各字符频度不重复的字符串：



频度表如下：



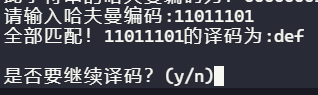
各字符对应的哈夫曼编码为：



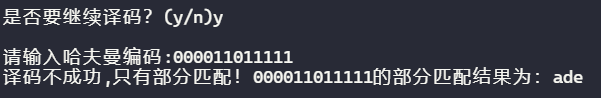
字符串对应的哈夫曼编码为：



我们再输入输入“def”对应的哈夫曼码”11011101”，看看程序是否能译码成功：



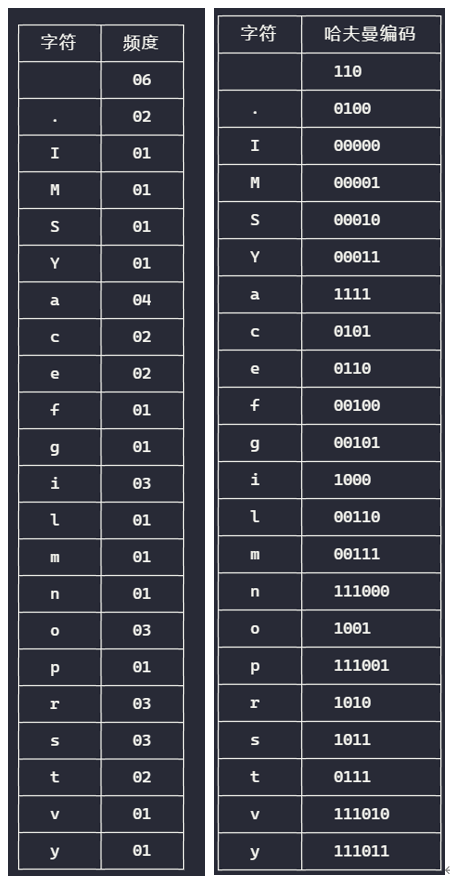
程序译码是正确的，我们再输入一个不匹配的码串，再来查看输出结果：



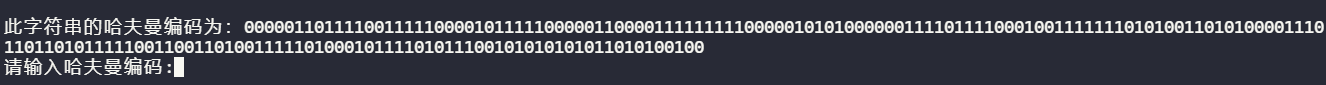
下面开始第三组数据的测试，我们输入一个普通的文本：



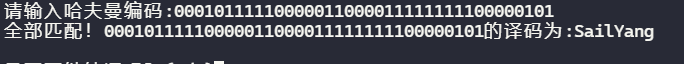
程序运行后，输出字符串中各字符及其对应的频度和哈夫曼编码（为了展示方便，放在一行）：



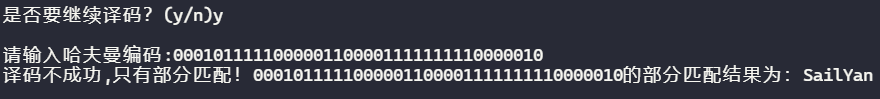
字符串对应的哈夫曼编码输出为：



我们输入“SailYang”对应的哈夫曼编码”00010111110000011000011111111100000101”：



可以看到程序是可以成功匹配的。接下来，我们去掉上面编码的最后一个数字，再次输入：



程序会给出错误信息。

以上就是课题二全部的测试数据和运行结果。

# 四、体会与总结

## 4.1 课题一的体会与总结

课题一是在8个居民区之间以最小的经济代价架设自来水管道，第一步就是分析问题，这个过程中锻炼了我的问题分析能力，即如何把一个实际的工程问题转化为逻辑上的数据结构，然后选取适当的存储结构来存储数据。这次课题的题目可以很好地对应图这种数据结构，如果是更复杂的工程问题，可能会对使用更多更复杂的数据结构，因此还需要加强对数据结构的学习，以及扩展知识。

在以图这种数据结构为基础后，要解决实际问题就需要图的相关算法。对于架设自来水管道方案的题目，需要使用生成最小生成树的Prim算法和Kruskal算法，在实际的编写代码过程中，我掌握了生成树的概念，以及Prim算法和Kruskal算法的原理及两者的区别，从而得出了合理的自来水管道架设方案。

对于求最短路径的题目，需要使用Dijkstra算法来生成居民区A到其它居民区的最短距离，在分析这个问题以及解决问题的过程中，我掌握了Dijkstra算法的原理与使用。

在最后DAG图的两个应用拓扑排序和关键路径题目中，我首先复习了DAG图的概念，也就是有向无环图。然后掌握了拓扑排序的方法和以拓扑排序为基础给关键路径。

在设计此课题时，巩固了图的数据结构及其相关算法的知识，并且将数据结构和算法应用于实际问题后，我对数据结构和算法的学习热情进一步提高。

## 4.2 课题二的体会与总结

课题2是根据一组文本生成各字符的频度表，然后生成一个哈夫曼树，生成各字符对应的哈夫曼编码，然后根据哈夫曼树进行编码与解码。

首先第一个问题是统计字符的频度，我使用的方法是开辟一个元素个数为ASCII码个数的数组（当然可以进一步优化），全部置0，然后遍历文本，在对应位置上+1，最后将不是0的ASCII码提取到一个表中，这个表是结构体数组，存储了每个字符及其对应的频度和哈夫曼编码。

然后就是根据各字符的频度生成哈夫曼树，我用到了一个单链表来存放森林，以此来生成哈夫曼树，过程还算顺利。

接下来就是根据哈夫曼树得出各个字符的哈夫曼编码，这个遇到的比较困难的问题就是如果哈夫曼树的一些叶子结点是重复的，相同频度的字符会有一样编码的问题。针对此问题，我设置了一些flag标志位和一个表，已经被访问过的权值会加入到表中，flag标志位用来记录一些被访问过的叶子结点，最后完美地解决了这个问题。

哈夫曼编码的译码问题相对比较简单，只要在哈夫曼树中一个数字一个数字的遍历即可，遇到叶子结点，说明一个字符的译码成功。如果最后还剩字符没有对应，则输出相关错误信息。

在此课题中，我体会到了哈夫曼编码的巧妙，哈夫曼编码可以根据字母出现的频度进行编码，频度高的字符码较短，频度低的字符码较长，在电报中，这样进行编码，可以大大地节省成本。

# 五、源程序

## 5.1 课题一

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAXVERTEX 10

#define VertexType char

#define INFINITY 32767.0

#define MAXSIZE 100

/\*结构体声明\*/

/\*------邻接矩阵------\*/

typedef struct

{

VertexType vexs[MAXVERTEX]; //存放顶点信息V

double arcs[MAXVERTEX][MAXVERTEX]; //存放关系集合E

int vexnum,arcnum; //当前顶点数和弧数

}MGraph;

/\*--------邻接表-------\*/

typedef struct ArcNode

{

int adjvex;

struct ArcNode \*nextarc;

double weight;

}ArcNode;

typedef struct VNode

{

VertexType data;

int indegree;

ArcNode \*firstarc;

}VNode,AdjList[MAXVERTEX];

typedef struct

{

AdjList vertices;

int vexnum,arcnum;

}AlGraph;

/\*---最小生成树相关结构体---\*/

typedef struct

{

int adjvex;

double lowcost;

}PEdge;

typedef struct

{

int begin;

int end;

double weight;

}KEdge;

typedef struct

{

int stack[MAXVERTEX];

int top;

}Stack;

/\*------函数声明--------\*/

void MCreateUDN(MGraph \*G\_ptr,char filename[MAXSIZE]); //创建无向网

void ACreateDN(AlGraph \*G\_ptr,char filename[MAXSIZE]); //创建DAG网

void Menu();

//Prim算法

void Prim\_MinSpanTree(MGraph G,int u);

int Get\_MinVex(MGraph G,PEdge closedge[MAXVERTEX]);

//Kruskal算法

void Kruskal\_MinSpanTree(MGraph G);

void OperationEdge(MGraph G, KEdge \*edges);

int Find(int \*parent,int f);

//快速排序，用于对边集数组排序

void QuickSort(KEdge \*edges ,int l,int h);

int Partition(KEdge \*edges,int l,int h);

//Dijkstra算法求最小路径

void Dijkstra(MGraph G,int source);

//DAG相关

void TopoSort(AlGraph G);

int TopoOrder(AlGraph G,Stack \*T\_ptr,double \*ve);

void CriticalPath(AlGraph G);

/\*-----main-----\*/

int main()

{

/\*铺设自来水管道\*/

MGraph MG1,MG2,MG3;

AlGraph AG1,AG2,AG3;

MCreateUDN(&MG1,"udn1.txt");

MCreateUDN(&MG2,"udn2.txt");

MCreateUDN(&MG3,"udn3.txt");

ACreateDN(&AG1,"dag1.txt");

ACreateDN(&AG2,"dag2.txt");

ACreateDN(&AG3,"dag3.txt");

char order;

do{

Menu();

order = getchar();

fflush(stdin);

system("cls");

switch(order)

{

case '1': printf("用Prim算法生成自来水管道架设方案:\n");printf("测试数据1：");Prim\_MinSpanTree(MG1,1);printf("测试数据2：");Prim\_MinSpanTree(MG2,1);printf("测试数据3：");Prim\_MinSpanTree(MG3,1);system("pause");break;

case '2': printf("用Kruskal算法生成自来水管道架设方案:\n");printf("测试数据1：");Kruskal\_MinSpanTree(MG1);printf("测试数据2：");Kruskal\_MinSpanTree(MG2);printf("测试数据3：");Kruskal\_MinSpanTree(MG3);system("pause");break;

case '3': printf("计算A居民区到其它各居民区的距离:\n");printf("测试数据1：\n");Dijkstra(MG1,1);printf("测试数据2：\n");Dijkstra(MG2,1);printf("测试数据3：\n");Dijkstra(MG3,1);system("pause");break;

case '4': printf("拓扑排序序列和关键路径:\n");printf("测试数据1：\n");TopoSort(AG1);CriticalPath(AG1);printf("测试数据2：\n");TopoSort(AG2);CriticalPath(AG2);printf("测试数据3：\n");TopoSort(AG3);CriticalPath(AG3);system("pause");system("cls");break;

default: printf("程序结束！");return 0;

}

system("cls");

}while(1);

return 0;

}

/\*------end-----\*/

/\*函数定义\*/

void Menu()

{

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*Menu\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n\n");

printf(" 1.用Prim算法生成自来水管道架设方案\n\n");

printf(" 2.用Kruskal算法生成自来水管道架设方案\n\n");

printf(" 3.计算A居民区到其它各居民区的距离\n\n");

printf(" 4.拓扑排序序列和关键路径\n\n");

printf(" 5.其他键退出程序\n\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*Menu\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n");

printf("请输入指令:");

}

void MCreateUDN(MGraph \*G\_ptr,char filename[MAXSIZE])

{

/\*从1开始存储\*/

FILE \*fp = fopen(filename,"r");

fscanf(fp,"%d,%d",&G\_ptr->vexnum,&G\_ptr->arcnum);

char space;fscanf(fp,"%c",&space);//去掉'\n'

int i,j,k,vnum = G\_ptr->vexnum,anum = G\_ptr->arcnum;double weight;

/\*初始化\*/

for(i=1;i<=vnum;i++) fscanf(fp,"%c",&G\_ptr->vexs[i]);

for(i=1;i<=vnum;i++)

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

G\_ptr->arcs[i][j]=INFINITY;

for(k=1;k<=anum;k++)

{

fscanf(fp,"%d,%d,%lf",&i,&j,&weight);

G\_ptr->arcs[i][j]=weight;

G\_ptr->arcs[j][i]=weight;

}

}

void ACreateDN(AlGraph \*G\_ptr,char filename[MAXSIZE])

{

FILE \*fp = fopen(filename,"r");

fscanf(fp,"%d,%d",&G\_ptr->vexnum,&G\_ptr->arcnum);

char space;fscanf(fp,"%c",&space);

int i,j,k,vnum = G\_ptr->vexnum,anum = G\_ptr->arcnum;ArcNode \*p;double weight;

for(i=1;i<=vnum;i++)

{

fscanf(fp,"%c",&G\_ptr->vertices[i].data);

G\_ptr->vertices[i].firstarc = NULL;

G\_ptr->vertices[i].indegree = 0;

}

for(k=1;k<=G\_ptr->arcnum;k++)

{

fscanf(fp,"%d,%d,%lf",&i,&j,&weight);

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

p->adjvex = j;p->weight = weight;

p->nextarc = G\_ptr->vertices[i].firstarc;

G\_ptr->vertices[i].firstarc = p;

G\_ptr->vertices[j].indegree++;

}

fclose(fp);

}

void Prim\_MinSpanTree(MGraph G,int u)

{

PEdge closedge[MAXVERTEX];int i,j,k;

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

{

if(i!=u)

{

closedge[i].adjvex=u;

closedge[i].lowcost = G.arcs[u][i];

}

}

closedge[u].lowcost=0;

for(i=2;i<=G.vexnum;i++)

{

k=Get\_MinVex(G,closedge);

printf("(%c,%c) ",G.vexs[closedge[k].adjvex],G.vexs[k]);

closedge[k].lowcost=0;

for(j=1;j<=G.vexnum;j++)

{

if(G.arcs[k][j]<closedge[j].lowcost)

{

closedge[j].adjvex=k;

closedge[j].lowcost=G.arcs[k][j];

}

}

}

printf("\n");

}

int Get\_MinVex(MGraph G,PEdge closedge[MAXVERTEX])

{

int i,index;

double lowedge=INFINITY;

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

{

if(closedge[i].lowcost&&closedge[i].lowcost<lowedge)

{

lowedge = closedge[i].lowcost;

index = i;

}

}

return index;

}

void OperationEdge(MGraph G, KEdge\* edges)

{

int i,j,k=1;

//无向图对称，只需转换一半即可

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

{

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

{

if (G.arcs[i][j] != INFINITY)

{

edges[k].begin = i;

edges[k].end = j;

edges[k].weight = G.arcs[i][j];

k++;

}

}

}

QuickSort(edges,1,G.arcnum);

}

void QuickSort(KEdge \*edges,int l,int h)

{

int k;

if(l<h)

{

k = Partition(edges,l,h);

QuickSort(edges,l,k-1);

QuickSort(edges,k+1,h);

}

}

int Partition(KEdge \*edges,int l,int h)

{

int i=l,j=h;edges[0] = edges[i];

while(i<j)

{

while(i<j&&edges[j].weight>=edges[0].weight) j--;

edges[i] = edges[j];

while(i<j&&edges[i].weight<=edges[0].weight) i++;

edges[j] = edges[i];

}

edges[i] = edges[0];

return i;

}

void Kruskal\_MinSpanTree(MGraph G)

{

int i,n,m;KEdge edges[MAXSIZE]; int parent[MAXSIZE];

OperationEdge(G,edges);

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

parent[i] = 0;

for (i = 1;i <= G.arcnum;i++)

{//循环每一条边

n = Find(parent,edges[i].begin);

m = Find(parent,edges[i].end);

if (n!=m) //没有成环

{

parent[n] = m;

printf("(%c,%c) ",G.vexs[edges[i].begin],G.vexs[edges[i].end]);

}

}

printf("\n");

}

int Find(int \*parent,int f)

{

while(parent[f] > 0)

f = parent[f];

return f;

}

void Dijkstra(MGraph G,int source)

{

int i,j,min\_node;

int \*Final = (int\*)malloc(sizeof(int)\*(G.vexnum+1));

double \*Dist = (double\*)malloc(sizeof(double)\*(G.vexnum+1)),min\_path;

for(j=1;j<=G.vexnum;j++)

{

Dist[j] = G.arcs[source][j];

Final[j] = 0;

}

Final[source] = 1; //源点到源点的最短路径确定

Dist[source] = 0; //源点到源点的最短路径为0

for(i=2;i<=G.vexnum;i++)

{

min\_path = INFINITY;

for(j=1;j<=G.vexnum;j++)

if(!Final[j]&&Dist[j]<min\_path)

{

min\_node = j;

min\_path = Dist[j];

}

Final[min\_node] = 1;//离source最近的点Min\_node被加到Final里了

for(j=1;j<=G.vexnum;j++)

if(!Final[j]&&(min\_path+G.arcs[min\_node][j]<Dist[j]))

Dist[j] = min\_path+G.arcs[min\_node][j];

}

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

{

if(i!=source)

printf("(%c→%c:%.2lf km) ",G.vexs[source],G.vexs[i],Dist[i]);

}

printf("\n");

}

void TopoSort(AlGraph G)

{

printf("拓扑排序结果为:\t");

Stack S;

ArcNode \*p;

S.top = 0;

int i,j,count=0;

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

if(!G.vertices[i].indegree)

S.stack[S.top++] = i;

while(S.top)

{

i = S.stack[--S.top];

printf("%c ",G.vertices[i].data);

count++;

for(p=G.vertices[i].firstarc;p;p=p->nextarc)

{

j = p->adjvex;

if(!(--G.vertices[j].indegree))

S.stack[S.top++] = j;

}

}

printf("\n");

if(count < G.vexnum)

printf("网中存在环\n");

}

int TopoOrder(AlGraph G,Stack \*T\_ptr,double \*ve)

{

Stack S;S.top = 0;

ArcNode \*p;

int i,j,count=0;

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

if(!G.vertices[i].indegree)

S.stack[S.top++] = i;

T\_ptr->top = 0;

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

ve[i] = 0;

while(S.top)

{

i = S.stack[--S.top];

T\_ptr->stack[T\_ptr->top++] = i;

count++;

for(p=G.vertices[i].firstarc;p;p=p->nextarc)

{

j = p->adjvex;

if(--G.vertices[j].indegree==0)

S.stack[S.top++] = j;

if((ve[i]+p->weight)>ve[j])

ve[j] = ve[i]+p->weight;

}

}

if(count < G.vexnum)

return 0;

else

return 1;

}

void CriticalPath(AlGraph G)

{

printf("关键路径结果为：");

int i,j;char tag;

double \*ve = (double\*)malloc(sizeof(double)\*(G.vexnum+1)),dut,ee,el;

double \*vl = (double\*)malloc(sizeof(double)\*(G.vexnum+1));

Stack T;

ArcNode \*p;

if(!TopoOrder(G,&T,ve)){

printf("有环！");

return;

}

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

vl[i] = ve[G.vexnum];

while(T.top)

{

for(i=T.stack[--T.top],p=G.vertices[i].firstarc;p;p=p->nextarc)

{

j = p->adjvex;

dut = p->weight;

if(vl[j]-dut<vl[i])

vl[i]=vl[j]-dut;

}

}

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

for(p=G.vertices[i].firstarc;p;p=p->nextarc)

{

j = p->adjvex;

dut = p->weight;

ee = ve[i];el = vl[j]-dut;

/\*tag = (ee==el)?'\*':' ';\*/

if(ee==el)

printf("(%c→%c) ",G.vertices[i].data,G.vertices[j].data);

/\*printf(" %d\t%d\t%.2lf\t%.2lf\t%.2lf\t%c\n",i,j,dut,ee,el,tag);\*/

}

printf("\n\n");

}

## 5.2 课题二

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#define MAX\_STRSIZE 200

#define MAX\_CODESIZE 100

/\*结构体声明\*/

typedef struct Node

{

struct Node \*parent,\*lchild,\*rchild;

int weight;

int flag;

}Node,\*HfmTree;

typedef struct HC

{

char character; //字符

int fequency; //字符的频度

char code[MAX\_CODESIZE];//该字符对应的哈夫曼编码

}HC;

typedef struct list

{

HC \*HCList;

int Len;

}List,\*ListPtr;

typedef struct leaf

{

Node \*node;

struct leaf \*next;

}leaf,\*Leaves;

typedef struct stack

{

Node \*node;

int flag;

}Stack;

/\*函数声明\*/

List GetFrequency(char \*str); //获取输入字符串的各字符频度

void PrintFrequency(List L); //打印各字符的频度

HfmTree CreateHuffmanTree(List L); //构造哈夫曼树

Node \*GetMinLeaf(Leaves \*LeafList\_ptr); //获取最小权值的叶子，并且从叶子列表中删除此节点

void InsertNewLeaf(Leaves \*LeafList\_ptr,Node \*root);//往叶子列表中插入新的叶子节点，权值为最小两个的和

void ClearLeaves(Leaves \*LeafList\_ptr); //释放叶子列表

leaf \*GetPreNode(Leaves LeafList,leaf \*p); //获取叶子列表中一个结点的前一个结点

void GetHC(HfmTree T,ListPtr L\_ptr); //获取Hfm编码(HC)

int InVisited(int weight,int \*visited,int vlen);

void Decode(HfmTree T,List L);

void PrintHc(List L,char \*str);

int main()

{

char str[MAX\_STRSIZE];

List L = GetFrequency(str);PrintFrequency(L);

HfmTree T = CreateHuffmanTree(L);

GetHC(T,&L);

PrintHc(L,str);

char order;

do

{

Decode(T,L);

printf("\n是否要继续译码？(y/n)");

order = getchar();

fflush(stdin);

}while(order=='y'||order=='Y');

return 0;

}

/\*函数定义\*/

List GetFrequency(char \*str)

{

List L;L.Len=0;

printf("请输入要进行哈夫曼编码的字符串：");

gets(str);

int i,j = 0,Len = strlen(str),ascii[128]={0},ascii\_code;

/\*统计出现的字符的频度\*/

for(i=0;i<Len;i++)

{

ascii\_code = (int) str[i];

ascii[ascii\_code]++;

}

/\*获得L的长度\*/

for(i=0;i<128;i++)

{

if(ascii[i]!=0)

L.Len++;

}

L.HCList = (HC\*)malloc(L.Len\*sizeof(HC));

for(i=0;i<L.Len;i++)

{

while(ascii[j]==0) j++;//定位不是0的ascii码

L.HCList[i].fequency = ascii[j];L.HCList[i].character = (char) j;

j++;

}

return L;

}

void PrintFrequency(List L)

{ /\*─│┌ ┐└ ┘├ ┤┬ ┴ ┼\*/

int i;

printf("\n┌────────┬────────┐\n");printf("│ 字符 │ 频度 │\n");

for(i=0;i<L.Len;i++)

{

printf("├────────┼────────┤\n");

printf("│ %c │ %02d │\n",L.HCList[i].character,L.HCList[i].fequency);

}

printf("└────────┴────────┘\n");

}

HfmTree CreateHuffmanTree(List L)

{

HfmTree T;Node \*minNode1,\*minNode2,\*root;int i;

Leaves LeafList = (Leaves)malloc(sizeof(leaf));leaf \*p = LeafList,\*q;

/\*初始化叶子结点的列表\*/

for(i=0;i<L.Len;i++)

{

q = (Leaves)malloc(sizeof(leaf));

q->node = (Node\*)malloc(sizeof(Node));

q->node->weight = L.HCList[i].fequency;

q->node->lchild = NULL;q->node->rchild = NULL;

p->next = q;

p = q;

}

p->next = NULL;

/\*开始构建哈夫曼树\*/

if(LeafList->next == NULL)

return NULL;

while(LeafList->next->next!=NULL)

{

minNode1 = GetMinLeaf(&LeafList);

minNode2 = GetMinLeaf(&LeafList);

root = (Node\*)malloc(sizeof(Node));

root->weight = minNode1->weight+minNode2->weight;

if(minNode1->weight<=minNode2->weight)

{

root->lchild = minNode1;

root->rchild = minNode2;

}else

{

root->lchild = minNode2;

root->rchild = minNode1;

}

root->flag = 0;

minNode1->parent = root;minNode2->parent = root;

InsertNewLeaf(&LeafList,root);

}

/\*最后一个结点就是root\*/

T = (HfmTree)malloc(sizeof(Node));

T->lchild = LeafList->next->node->lchild;

T->rchild = LeafList->next->node->rchild;

T->parent = NULL;

T->weight = LeafList->next->node->weight;

T->flag = 0;

/\*清空叶子列表\*/

ClearLeaves(&LeafList);

return T;

}

Node \*GetMinLeaf(Leaves \*LeafList\_ptr)

{

Leaves p = (\*LeafList\_ptr)->next;leaf \*q,\*minNode=p,\*pre\_minNode;

int minweight = p->node->weight;

p = p->next;

while(p!=NULL)

{

if(p->node->weight<minweight)

{

minNode = p;minweight = p->node->weight;

}

p = p->next;

}

pre\_minNode = GetPreNode(\*LeafList\_ptr,minNode);

pre\_minNode->next = minNode->next;

Node \*HfmNode = (Node\*)malloc(sizeof(Node));

HfmNode->lchild = minNode->node->lchild;

HfmNode->rchild = minNode->node->rchild;

HfmNode->weight = minNode->node->weight;

HfmNode->flag = 0;

free(minNode);

return HfmNode;

}

leaf \*GetPreNode(Leaves LeafList,leaf \*p)

{

leaf \*pre\_p = LeafList;

while(pre\_p->next!=p)

pre\_p = pre\_p->next;

return pre\_p;

}

void InsertNewLeaf(Leaves \*LeafList\_ptr,Node \*root)

{

leaf \*p = (\*LeafList\_ptr),\*new\_node;

new\_node = (leaf\*)malloc(sizeof(leaf));

new\_node->node = root;

new\_node->next = p->next;

p->next = new\_node;

}

void ClearLeaves(Leaves \*LeafList\_ptr)

{

leaf \*p = (\*LeafList\_ptr),\*q;

while(p!=NULL)

{

q = p->next;

free(p);

p = q;

}

}

void GetHC(HfmTree T,ListPtr L\_ptr)

{

Node \*p = T;

Stack s[100];int top = -1,strlen=0;

int visited[MAX\_STRSIZE] = {0},vlen=0; //记录已经被访问过的叶子结点

int i,weight;

if(T->lchild==NULL&&T->lchild==NULL){

strcpy(L\_ptr->HCList[i].code,"0");return;

}

for(i=0;i<L\_ptr->Len;i++)

{

p = T;top = -1;strlen=0;

strcpy(L\_ptr->HCList[i].code,"\0");

weight = L\_ptr->HCList[i].fequency;

while(1)

{

if(p->lchild==NULL&&p->rchild==NULL){

if(p->weight==weight&&p->flag==0){

if(!InVisited(weight,visited,vlen))

visited[vlen++] = weight;

p->flag = 1;

break;

}else{

if(s[top].flag==0){

p = s[top].node->rchild;

L\_ptr->HCList[i].code[strlen-1] = '\0';

strcat(L\_ptr->HCList[i].code,"1");

s[top].flag=1;

}else

{

L\_ptr->HCList[i].code[--strlen] = '\0';

while(s[top].flag==1)

{

L\_ptr->HCList[i].code[--strlen] = '\0';

top--;

}

p = s[top].node->rchild;

s[top].flag = 1;

strcat(L\_ptr->HCList[i].code,"1");

strlen++;

}

}

}else{

s[++top].node = p;

s[top].flag = 0;

strcat(L\_ptr->HCList[i].code,"0");

strlen++;

p = p->lchild;

}

}

}

printf("\n┌────────┬──────────────┐\n");printf("│ 字符 │ 哈夫曼编码 │\n");

for(i=0;i<L\_ptr->Len;i++)

{

printf("├────────┼──────────────┤\n");

printf("│ %c │ %-8s │\n",L\_ptr->HCList[i].character,L\_ptr->HCList[i].code);

}

printf("└────────┴──────────────┘\n");

}

int InVisited(int weight,int \*visited,int vlen)

{

int i;

for(i=0;i<vlen;i++)

{

if(visited[i]==weight)

return 1;

}

return 0;

}

void Decode(HfmTree T,List L)

{

char str[MAX\_STRSIZE],result[MAX\_STRSIZE] = {'\0'},path[MAX\_STRSIZE]={'\0'};int i=0,j=0;

Node \*p = T;

printf("\n请输入哈夫曼编码:");

gets(str);

while(i<=strlen(str))

{

if(p->lchild==NULL&&p->rchild==NULL){

j=0;

while(strcmp(path,L.HCList[j].code)!=0)

j++;

result[strlen(result)] = L.HCList[j].character;

strcpy(path,"\0");

p = T;

}else

{

if(str[i]=='0'){

strcat(path,"0");

p = p->lchild;

}else if(str[i]=='1'){

strcat(path,"1");

p = p->rchild;

}else if(str[i]=='\0'){

break;

}else{

printf("\n输入数据只能包含1和0！\n");

return;

}

i++;

}

}

if(strcmp(path,"\0")==0) printf("全部匹配！%s的译码为:%s\n",str,result);

else printf("译码不成功,只有部分匹配！%s的部分匹配结果为：%s\n",str,result);

}

void PrintHc(List L,char \*str)

{

int i,j;

printf("\n此字符串的哈夫曼编码为：");

for(i=0;i<strlen(str);i++)

for(j=0;j<L.Len;j++)

if(str[i]==L.HCList[j].character)

printf("%s",L.HCList[j].code);

}