# 数据结构大作业

| 小组成员姓名 | 分工 | 打分 |
| --- | --- | --- |
| **吴程锴** | 编写第九题中模拟退火算法，能量函数，可视化代码 | 100 |
| **刘亦高** | 构建第六题中的哈夫曼树，实现编码解码功能 | 100 |
| **蒋易陶** | 构建邻接矩阵，设计交互页面 | 100 |

# 一．题目 9.TSP

## 1.1 问题概述

某旅行团要从南宁坐飞机周游东南亚 7 国，如果八地之间都有直飞航班，已知南宁坐标（378,78），河内（327,119），曼谷（232,266），金边（314,311），吉隆坡（255,477），新加坡（296,513），文莱（510,438），马尼拉（628,246），编程寻找最短周游路径，并显示出来。

## 1.2 问题分析

本题为旅行商（TSP）问题，只能用优化算法找出可能的最优解，为了跳出局部最优，找到全局最优解，本文采用模拟退火算法。

| 编号 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 地名 | 南宁 | 河内 | 曼谷 | 金边 | 吉隆坡 | 新加坡 | 文莱 | 马尼拉 |
| X 坐标 | 378 | 327 | 232 | 314 | 255 | 296 | 510 | 628 |
| Y 坐标 | 78 | 119 | 266 | 311 | 477 | 513 | 438 | 246 |

**表格 1 各地点编号及信息**

如表格 1 所示，对八个地点进行编号，编号结果表格 1 所示。

用 Matlab 画出八个地点的相对位置，如图表 1 所示。代码见附录一。

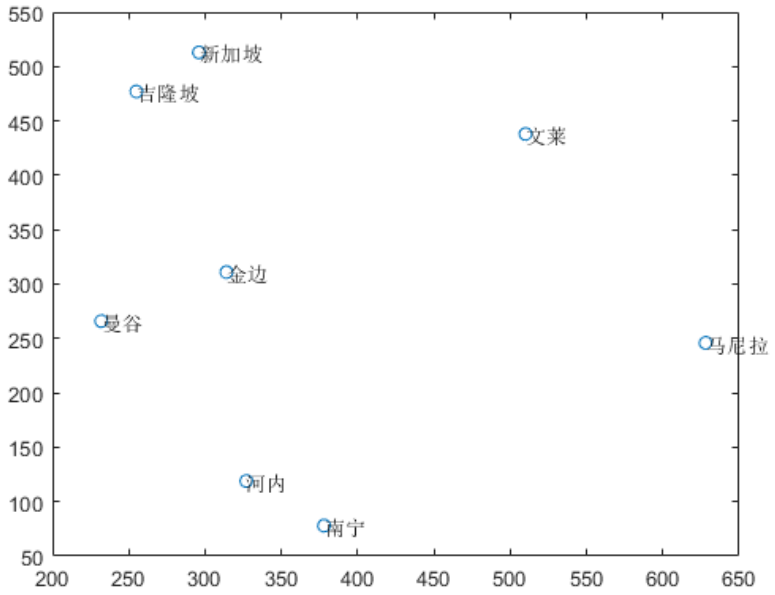



图 1 各地点相对位置

## 1.3 问题求解

### 1.3.1 构建邻接矩阵

根据表格一中各位置编号和坐标构建邻接矩阵。

| | | | |
|---|---|---|---|
| 0 | 65.43699 | 238.0336 | 241.6299 |
| 65.43699 | 0 | 175.0257 | 192.4396 |
| 238.0336 | 175.0257 | 0 | 93.53609 |
| 241.6299 | 192.4396 | 93.53609 | 0 |

表格 2 部分领接矩阵

部分邻接矩阵如图表 2 所示，代码及完整矩阵见附录二。

### 1.3.2 寻找最优解

随机生成初始解，如[1 8 7 2 4 5 3 6 1]，其含义为从 1 地（南宁）出发，先前往 8 地（马尼拉），再前往 7 地（文莱），以此类推，最终回到 1 地（南宁）。

模拟退火算法可以分解为解空间、目标函数和初始解三部分。

(1)初始化：初始温度$T_0$(充分大)，初始解状态$S_0$(是算法迭代的起点)，令$S_{current}=S_{best}=S_{new}=S_0$，其中$S_{current},S_{best},S_{new}$分别为当前解，最优解和新解。把$S_0$代入适应度函数$f$，得到$E_{new}$，令$E_{current}=E_{best}=E_{new}$，其中$E_{current},E_{best},E_{new}$分别为$S_{current},S_{best},S_{new}$对应的适应度函数值。$T$从$T_0$降温到$Tf$，每个$T$值迭代$L$次；

(2)对$k=1\sim L$，循环执行(3)至(5)步：

(3)产生新解$S_{new}$；

(4)计算新解的适应度函数值$E_{new}=f(S_{new})$；

(5)若$E_{new}<E_{current}$，则接受$S_{new}$作为新的当前解$S_{current}$。同时，若$E_{new}<E_{best}$，则可使该新解成为最优解，否则以概率$\exp(\frac{-(f(S_{new})-f(S_{current}))}{T})$接受$S_{new}$作为新的当前解，否则令$S_{new}=S_{current}$，目的是跳出局部最优解；

(6) $T$逐渐减少，如果满足终止条件则输出最优解，结束程序。

（2）控制参数的确定：

$T_0$：初始温度，应该比较大，为了所求的解更加接近最优解，本文中令初始温度$T_0=97$；

$T$：温度变化量；

$T_{k+1}=\alpha T_k$：$T$的衰减函数，其中$\alpha=0.99$为$T$的衰减因子。

$L$：Markov链长度，设置为1000；

$Tf$：停止条件$Tf$设置为3，$T\le Tf$时程序停止；

### 1.3.3 结果

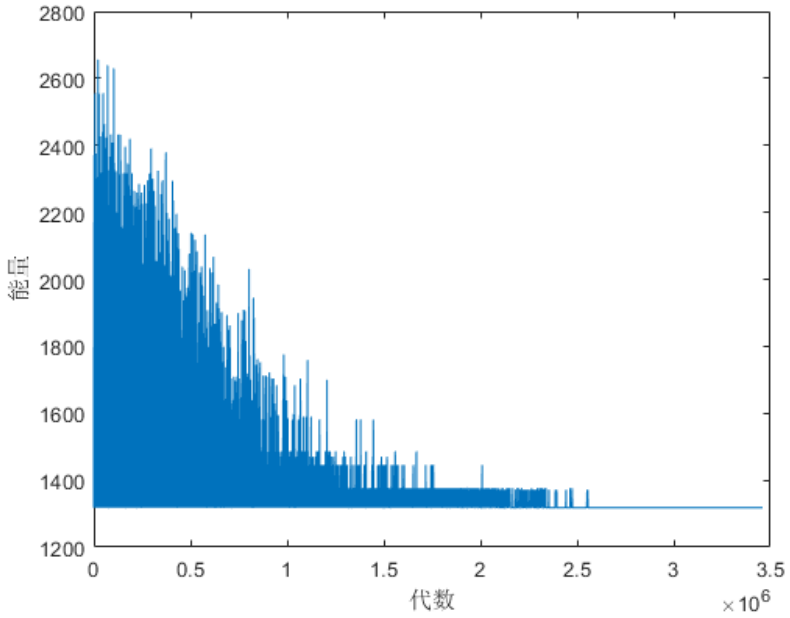



图 2 各代次能量值

图 2 为各代解对应的能量函数，可以看到，能量逐渐趋近最优状态。

```
  1     2     3     4     5     6     7     8     1

1.3181e+03
```

图 3 运行结果

最终运行结果如图 3 所示，虽短路径为 1318.1，最优周游路径为

南宁 → 河内 → 曼谷 → 金边 → 吉隆坡 → 新加坡 → 文莱 → 马尼拉 → 南宁

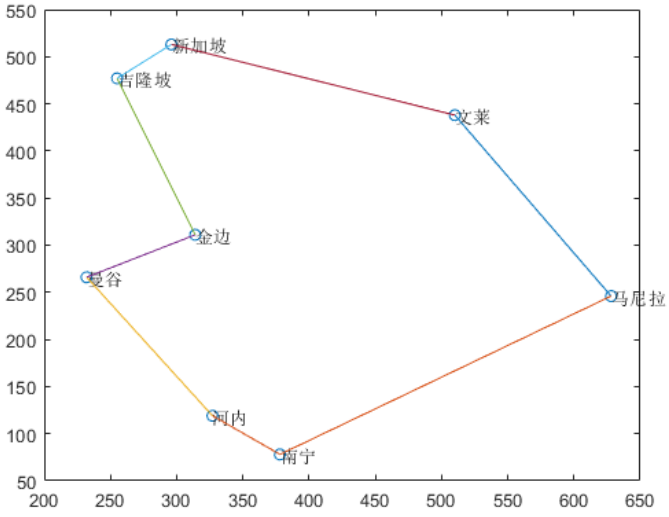



图 4 最短周游路径

## 1.4 总结

本文使用邻接矩阵的目的是：本体最关键是要求路径长度，邻接矩阵能够快速、方便地获得两点之间的距离。

实验心得：用优化算法解决 TSP 问题。如果题目更复杂，不一定能找到全局最优解。优化算法的运用更加熟练了。

## 二、题目 6（哈夫曼树的编/译码器）

### 1.1 问题描述

利用哈夫曼编码进行通讯可以大大提高信道利用率，缩短信息传输时间，减低传输成本。但是，这要求在发送端通过一个编码系统对待传数据进行预先编码；在接受端将传来的数据进行解码（复原)。对于可以双向传输的信道，每端都要有一个完整的编/译码系统。试为这样的信息收发站写一个哈夫曼的编译码系统。

### 1.2 问题分析

此题为哈夫曼树的编码问题，包括了哈夫曼树的构造、编码和译码。哈夫曼树编译码的主要用途是实现数据压缩，要对电文中出现的每一个字符进行二进制编码，需遵循两个原则：

（1） 发送方传输的二进制编码，到接收方解码后必须具有唯一性，即解码结果与发送方发送的电文完全一致；

（2） 发送的二进制编码尽可能短。

根据这两点要求，本题所设计的算法需满足以下要求：

（1） 根据输入的字符和字符相应的频度建立哈夫曼树；

（2） 能将建立好的哈夫曼树用简单易懂的方式输出；

（3） 实现哈夫曼树的编码和译码功能，电文内容与编码内容一一对应；

（4） 采用不等长编码，根据不同字符的出现频率基于不等长编码，使用频度较高的字符分配较短的编码，使用频度较低的字符分配较长的编码；

（5） 操作界面简洁。

### 1.3 问题求解

### 1.3.1 建立哈夫曼树

首先，定义哈夫曼树的储存结构，该哈夫曼树的结构体需要一个整型变量储存结点的权值 weight，由两个指针分别储存结点的左右孩子，此外，由于输入的权值存在重复，需要额外用一个整型变量 id 加以区分。

其次，构造哈夫曼树，构造哈夫曼树的哈夫曼算法如下：

（1） 由给定的 n 个权值，构造具有 n 棵二叉树的森林，其中每一棵二叉树只有一个带有权值的根结点，其左、右结点均为空；

（2） 在森林中选取两棵根结点权值最小和次小的二叉树，作为左右子树构造一棵二叉树，其根结点的权值即为其左、右子树的根结点的权值之和；

（3） 从森林中删除已构成新二叉树的左右子树的两棵二叉树，并将新构成的二叉树放入森林中；

（4） 重复（2）和（3），直到 F 中仅剩一颗二叉树，即哈夫曼树。

根据算法可知，一个由 n 个叶子结点组成的初始集合，要生成哈夫曼树需要进行 n-1 次合并，产生 n-1 个新结点，最终所得的哈夫曼树共有 2n-1 个结点。

C 语言描述如下：

```
// 哈夫曼树结点结构体
typedef struct HuffmanTree
{
    int weight;
    int id;              // id 用来主要用以区分权值相同的结点
    struct HuffmanTree* lchild;
    struct HuffmanTree* rchild;
```

```
}HuffmanNode;


// 构建哈夫曼树
HuffmanNode* createHuffmanTree(int* a, int n)
{
    int i, j;
    HuffmanNode **temp, *hufmTree;
    temp = (HuffmanNode**)malloc(n * sizeof(HuffmanNode));
    for (i = 0; i < n; ++i)
    {
        temp[i] = (HuffmanNode*)malloc(sizeof(HuffmanNode));
        temp[i]->weight = a[i]; // 将数组 a 中的权值赋给结点中的 weight
        temp[i]->id = i;
        temp[i]->lchild = temp[i]->rchild = NULL;
    }
    for (i = 0; i < n - 1; ++i)          // 构建哈夫曼树需要 n-1 合并
    {
        int small1 = -1, small2;          // small1、small2 分别作为最小和次小权值
的下标
        for (j = 0; j < n; ++j)              // 先将最小的两个下标赋给 small1、
small2
        {
```

```
        if (temp[j] != NULL && small1 == -1)
        {
            small1 = j;
            continue;
        }
        else if (temp[j] != NULL)
        {
            small2 = j;
            break;
        }
    }

    for (j = small2; j < n; ++j)
    {
        if (temp[j] != NULL)
        {
            if (temp[j]->weight < temp[small1]->weight)
            {
                small2 = small1;
                small1 = j;
            }
            else if (temp[j]->weight < temp[small2]->weight)
```

```
                {
                    small2 = j;
                }
// 比较权值，挪动 small1 和 small2 使之分别成为最小和次小权值的下标

            }
        }
        hufmTree = (HuffmanNode*)malloc(sizeof(HuffmanNode));
        hufmTree->weight = temp[small1]->weight + temp[small2]->weight;
        hufmTree->lchild = temp[small1];
        hufmTree->rchild = temp[small2];

        temp[small1] = hufmTree;
        temp[small2] = NULL;
    }
    free(temp);
    return hufmTree;
}
```

### 1.3.2 哈夫曼树的输出

关于哈夫曼树的输出，可使用广义表的形式输出，可采用递归的方法。

C 语言描述如下：

```
// 以广义表的形式打印哈夫曼树
void PrintHuffmanTree(HuffmanNode* hufmTree)
{
    if (hufmTree)
    {
        printf("%d", hufmTree->weight);
        if (hufmTree->lchild != NULL || hufmTree->rchild != NULL)
        {
            printf("(");
            PrintHuffmanTree(hufmTree->lchild);
            printf(",");
            PrintHuffmanTree(hufmTree->rchild);
            printf(")");
        }
    }
}
```

### 1.3.3 对每个字符进行编码输出

此哈夫曼树编码方式为从根节点出发，左子树的路径代码设为‘0‘，右子树的路径设为‘1’，查找叶子结点，将每个字符储存在数组 string 中，将路径上的‘0’和‘1‘储存在数组 code 中，通过递归的方式逐层查找该哈夫曼树的叶子结点，若该结点的左右子树非空，在 code 中输入‘0’并查找左子树，在 code

中输入‘1’并查找右子树，若该结点的左右子树均为空，则该结点为叶子结点，用 code 输出该结点相应的字符的编码。

C 语言描述如下：

```
// 递归进行哈夫曼编码
void HuffmanCode(HuffmanNode* hufmTree, int depth,char string[])
{
    static int code[10];
    if (hufmTree)
    {
        if (hufmTree->lchild == NULL && hufmTree->rchild == NULL)
        {
            printf("%c 的哈夫曼编码为：  ",string[hufmTree->id]);
            int i;
            for (i = 0; i < depth; ++i)
            {
                printf("%d", code[i]);
            }
            printf("\n");
        }
        else
        {
            code[depth] = 0;
```

```
            HuffmanCode(hufmTree->lchild, depth + 1,string);
            code[depth] = 1;
            HuffmanCode(hufmTree->rchild, depth + 1,string);
        }
    }
}
```

### 1.3.4 对输入的字符串进行编码

先将新输入的字符串储存在 string 中，调用建立哈夫曼树时所用的字符数组 ch，逐个从 ch 中查找到与 string 相同的字符，匹配之后，将该字符的权值代入 Encode 函数中进行编码，逐个输出 string 中字符的编码。

C 语言描述如下：

```
//哈夫曼编码
void Encode(HuffmanNode* hufmTree, int depth, int w)
{
    static int num[10];
    if (hufmTree)
    {
        if (hufmTree->lchild == NULL && hufmTree->rchild == NULL &&
hufmTree->weight == w)
        {
            int i;
```

```
            for (i = 0; i < depth; ++i)
            {
                printf("%d", num[i]);
            }
        }
        else
        {
            num[depth] = 0;
            Encode(hufmTree->lchild, depth + 1, w);
            num[depth] = 1;
            Encode(hufmTree->rchild, depth + 1, w);
        }
    }
}

void HuffmanEncode(HuffmanNode* hufmTree, char string[], char ch[], int* a)
{
    int i, j;
    for (i = 0; i < strlen(string); i++)
    {
        for (j = 0; j < strlen(ch); j++)
        {
```

```
            if (string[i] == ch[j])
            {
                Encode(hufmTree, 0, a[j]);
            }
        }
    }
}
```

### 1.3.5 对输入的编码进行译码

取 ch 为要解码的编码串，string 是结点对应的字符串，以 ch 所给的编码值作为查找路径，从根结点出发，查找叶子结点，并输出该叶子结点所代表的字符，当查找到字符时，返回根结点，进行下一个编码值的查找。

C 语言描述如下：

```
// 哈夫曼解码
void HuffmanDecode(char ch[], HuffmanNode* hufmTree, char string[])
{
    int i;
    int num[100];
    HuffmanNode* tempTree = NULL;
    for (i = 0; i < strlen(ch); ++i)
    {
        if (ch[i] == '0')
```

```
            num[i] = 0;
        else
            num[i] = 1;
    }
    if (hufmTree)
    {
        i = 0;
        while (i < strlen(ch))
        {
            tempTree = hufmTree;
            while (tempTree->lchild != NULL && tempTree->rchild != NULL)
            {
                if (num[i] == 0)
                {
                    tempTree = tempTree->lchild;
                }
                else
                {
                    tempTree = tempTree->rchild;
                }
                i++;
            }
```

```
            printf("%c", string[tempTree->id]); // 输出解码后对应结点的字符
        }
    }
}
```

### 1.3.6 操作界面设置

本程序的操作界面分为四个部分：建立哈夫曼树的输入部分；输出哈夫曼树部分；输入字符相应编码值部分；编码译码选择部分。

将程序的编码译码选择部分设计成一个选择页面，分为“编码”、“译码”、“退出”三个部分，通过输入‘e‘、’d‘、’q‘进行操作的选择。

C 语言描述如下：

```
//主程序
int main()
{
    printf("创建哈夫曼树:");
    int i, n;
    printf("请输入字符的个数：\n");
    while (1)
    {
        scanf("%d", &n);
        if (n > 1)
            break;
```

```
        else
            printf("输入错误，请重新输入！");
    }

    int* arr;
    arr = (int*)malloc(n * sizeof(int));
    printf("请输入%d 个字符出现的频度：\n", n);
    for (i = 0; i < n; ++i)
    {
        scanf("%d", &arr[i]);
    }

    char ch[100], string[100],c;
    printf("请连续输入这%d 个频度各自所代表的字符：\n", n);
    fflush(stdin);          // 强行清除缓存中的数据，也就是上面输入权值结束
时的回车符
    gets(string);

    HuffmanNode* hufmTree = NULL;
    hufmTree = createHuffmanTree(arr, n);

    printf("此哈夫曼树的广义表形式为：\n");
```

```
PrintHuffmanTree(hufmTree);


printf("\n 各字符的哈夫曼编码为：\n");
HuffmanCode(hufmTree, 0, string);


printf("\ne:编码(Encode)\nd:解码(Decode)\nq:退出(Quit))\n");
printf("请选择操作：");
scanf("%c",&c);
do
{
    switch (c)
    {
        case'e':
        {
            printf("请输入想要编码的文本：");
            fflush(stdin);
            gets(ch);
            printf("编码结果为：\n");
            HuffmanEncode(hufmTree, ch, string, arr);
            printf("\n");
            printf("\ne:编码(Encode)\nd:解码(Decode)\nq:退出(Quit))\n");
            printf("请选择操作：");
```

```
        break;
    }
    case’d’:
    {
        printf("请输入想要解码的电文：");
        fflush(stdin);
        gets(ch);
        printf("解码结果为：\n");
        HuffmanDecode(ch, hufmTree, string);
        printf("\n");
        printf("\ne:编码(Encode)\nd:解码(Decode)\nq:退出(Quit))\n");
        printf("请选择操作：");
        break;
    }
    case’q’:
    {
        exit(0);
        break;
    }
    default:
    {
        exit(0);
```

```
                    break;
                }
            }
            scanf("%c",&c);
        } while (c != 'q');

        free(arr);
        free(hufmTree);

        return 0;
}
```

## 1.4 程序运行

所给数据：

| 字符 | A | B | C | D | E | F | G | H | I | J | K | L | M | N |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 频度 | 64 | 13 | 22 | 32 | 103 | 21 | 15 | 47 | 57 | 1 | 5 | 32 | 20 | 57 |
| 字符 | O | P | Q | R | S | T | U | V | W | X | Y | Z | 空格 | |
| 频度 | 63 | 15 | 1 | 48 | 51 | 80 | 23 | 8 | 18 | 1 | 16 | 1 | 168 | |

运行结果：

一共有 27 个字符需要输入

输入操作如下：

```
创建哈夫曼树:请输入字符的个数:
27
请输入27个字符出现的频度:
64 13 22 32 103 21 15 47 57 1 5 32 20 57 63 15 1 48 51 80 23 8 18 1 16 1 168
请连续输入这27个频度各自所代表的字符:
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

哈夫曼树输出和字符编码输入如下：

```
此哈夫曼树的广义表形式为:
982(408(191(92(45(22,23),47),99(48,51)),217(103,114(57,57))),574(250(122(59(28(13,15),31(15,16)
),63),128(64,64(32,32))),324(156(76(35(17(8,9(4(2(1,1),2(1,1)),5)),18),41(20,21)),80),168)))
各字符的哈夫曼编码为:
C的哈夫曼编码为:  00000
U的哈夫曼编码为:  00001
H的哈夫曼编码为:  0001
R的哈夫曼编码为:  0010
S的哈夫曼编码为:  0011
E的哈夫曼编码为:  010
I的哈夫曼编码为:  0110
N的哈夫曼编码为:  0111
B的哈夫曼编码为:  100000
G的哈夫曼编码为:  100001
P的哈夫曼编码为:  100010
Y的哈夫曼编码为:  100011
O的哈夫曼编码为:  1001
A的哈夫曼编码为:  1010
D的哈夫曼编码为:  10110
L的哈夫曼编码为:  10111
V的哈夫曼编码为:  1100000
J的哈夫曼编码为:  1100001000
Q的哈夫曼编码为:  1100001001
X的哈夫曼编码为:  1100001010
Z的哈夫曼编码为:  1100001011
K的哈夫曼编码为:  11000011
W的哈夫曼编码为:  110001
M的哈夫曼编码为:  110010
F的哈夫曼编码为:  110011
T的哈夫曼编码为:  1101
 的哈夫曼编码为:  111
```

操作选择页面如下：

```
e:编码(Encode)
d:解码(Decode)
q:退出(Quit))
请选择操作:
```

- 选择“编码”选项，并输入：THIS PROGRAM IS MY FAVORITE

输出结果如下：

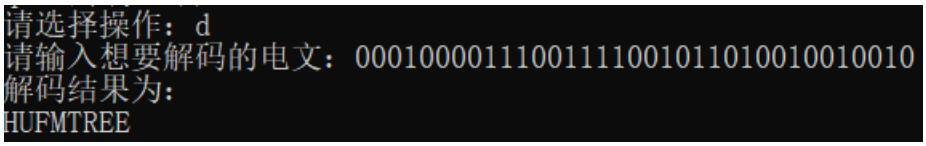
```
请选择操作: e

请输入想要编码的文本: THIS PROGRAM IS MY FAVORITE
编码结果为:
11010001011001110011111100001100010001010011000011000100010101011001011101100111001111111001010
00111111100111010110000001001001001100111101010
```

输出结果经比较，与字符编码相符，编码正确。

- 选择“解码”选项，并输入：00010000111001111001011010010010010

输出结果如下：




输出结果经比较，我字符相符，解码正确。

- 选择“退出”选项，程序退出。

## 1.5 总结

利用哈夫曼树，将字符转换成一组最优前缀编码，从而实现数据的压缩，降低传输的难度，提高传输的效率。

实验心得：通过学习哈夫曼树的定义和原理，基本掌握了构造哈夫曼树的意义以及算法思想，通过实际上机实验，具体构造哈夫曼树，进一步理解了构造哈夫曼树编码的意义。

## 附录：

完整代码如下：

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>


// 哈夫曼树结点结构体

typedef struct HuffmanTree

{

    int weight;
```

```
    int id;              // id 用来主要用以区分权值相同的结点
    struct HuffmanTree* lchild;
    struct HuffmanTree* rchild;
}HuffmanNode;

// 构建哈夫曼树
HuffmanNode* createHuffmanTree(int* a, int n)
{
    int i, j;
    HuffmanNode **temp, *hufmTree;
    temp = (HuffmanNode**)malloc(n * sizeof(HuffmanNode));
    for (i = 0; i < n; ++i)          // 将数组 a 中的权值赋给结点中的 weight
    {
        temp[i] = (HuffmanNode*)malloc(sizeof(HuffmanNode));
        temp[i]->weight = a[i];
        temp[i]->id = i;
        temp[i]->lchild = temp[i]->rchild = NULL;
    }
    for (i = 0; i < n - 1; ++i)              // 构建哈夫曼树需要 n-1 合并
    {
        int small1 = -1, small2;           // small1、small2 分别作为最小和次小权值
的下标
```

```
        for (j = 0; j < n; ++j)                // 先将最小的两个下标赋给 small1、
small2
        {
            if (temp[j] != NULL && small1 == -1)
            {
                small1 = j;
                continue;
            }
            else if (temp[j] != NULL)
            {
                small2 = j;
                break;
            }
        }

        for (j = small2; j < n; ++j)
        {
            if (temp[j] != NULL)
            {
                if (temp[j]->weight < temp[small1]->weight)    // 比较权值，挪
动 small1 和 small2 使之分别成为最小和次小权值的下标
                {
```

```
                small2 = small1;
                small1 = j;
            }
            else if (temp[j]->weight < temp[small2]->weight)    // 比较权值，挪动 small1 和 small2 使之分别成为最小和次小权值的下标
            {
                small2 = j;
            }
        }
    }
    hufmTree = (HuffmanNode*)malloc(sizeof(HuffmanNode));
    hufmTree->weight = temp[small1]->weight + temp[small2]->weight;
    hufmTree->lchild = temp[small1];
    hufmTree->rchild = temp[small2];

    temp[small1] = hufmTree;
    temp[small2] = NULL;
  }
  free(temp);
  return hufmTree;
}
```

```
// 以广义表的形式打印哈夫曼树
void PrintHuffmanTree(HuffmanNode* hufmTree)
{
    if (hufmTree)
    {
        printf("%d", hufmTree->weight);
        if (hufmTree->lchild != NULL || hufmTree->rchild != NULL)
        {
            printf("(");
            PrintHuffmanTree(hufmTree->lchild);
            printf(",");
            PrintHuffmanTree(hufmTree->rchild);
            printf(")");
        }
    }
}

// 递归进行哈夫曼编码
void HuffmanCode(HuffmanNode* hufmTree, int depth,char string[])        //
depth 为哈夫曼树的深度
{
    static int code[10];
```

```
    if (hufmTree)
    {
        if (hufmTree->lchild == NULL && hufmTree->rchild == NULL)
        {
            printf("%c 的哈夫曼编码为:  ",string[hufmTree->id]);
            int i;
            for (i = 0; i < depth; ++i)
            {
                printf("%d", code[i]);
            }
            printf("\n");
        }
        else
        {
            code[depth] = 0;
            HuffmanCode(hufmTree->lchild, depth + 1,string);
            code[depth] = 1;
            HuffmanCode(hufmTree->rchild, depth + 1,string);
        }
    }
}
```

```
//哈夫曼编码
void Encode(HuffmanNode* hufmTree, int depth, int w)
{
    static int num[10];
    if (hufmTree)
    {
        if (hufmTree->lchild == NULL && hufmTree->rchild == NULL &&
hufmTree->weight == w)
        {
            int i;
            for (i = 0; i < depth; ++i)
            {
                printf("%d", num[i]);
            }
        }
        else
        {
            num[depth] = 0;
            Encode(hufmTree->lchild, depth + 1, w);
            num[depth] = 1;
            Encode(hufmTree->rchild, depth + 1, w);
        }
```

```
    }
}

void HuffmanEncode(HuffmanNode* hufmTree, char string[], char ch[], int* a)
{
    int i, j;
    for (i = 0; i < strlen(string); i++)
    {
        for (j = 0; j < strlen(ch); j++)
        {
            if (string[i] == ch[j])
            {
                Encode(hufmTree, 0, a[j]);
            }
        }
    }
}

// 哈夫曼解码
void HuffmanDecode(char ch[], HuffmanNode* hufmTree, char string[])        //
ch 是要解码的 01 串，string 是结点对应的字符
{
```

```
int i;
int num[100];
HuffmanNode* tempTree = NULL;
for (i = 0; i < strlen(ch); ++i)
{
    if (ch[i] == '0')
        num[i] = 0;
    else
        num[i] = 1;
}
if (hufmTree)
{
    i = 0;
    while (i < strlen(ch))
    {
        tempTree = hufmTree;
        while (tempTree->lchild != NULL && tempTree->rchild != NULL)
        {
            if (num[i] == 0)
            {
                tempTree = tempTree->lchild;
            }
```

```
                else
                {
                    tempTree = tempTree->rchild;
                }
                i++;
            }
            printf("%c", string[tempTree->id]);        // 输出解码后对应结点的字符
        }
    }
}

//主程序
int main()
{
    printf("创建哈夫曼树:");
    int i, n;
    printf("请输入字符的个数：\n");
    while (1)
    {
        scanf("%d", &n);
        if (n > 1)
```

```
            break;
        else
            printf("输入错误，请重新输入！");
    }

    int* arr;
    arr = (int*)malloc(n * sizeof(int));
    printf("请输入%d 个字符出现的频度：\n", n);
    for (i = 0; i < n; ++i)
    {
        scanf("%d", &arr[i]);
    }

    char ch[100], string[100],c;
    printf("请连续输入这%d 个频度各自所代表的字符：\n", n);
    fflush(stdin);          // 强行清除缓存中的数据，也就是上面输入权值结束
时的回车符
    gets(string);

    HuffmanNode* hufmTree = NULL;
    hufmTree = createHuffmanTree(arr, n);
```

```
printf("此哈夫曼树的广义表形式为：\n");
PrintHuffmanTree(hufmTree);

printf("\n 各字符的哈夫曼编码为：\n");
HuffmanCode(hufmTree, 0, string);

printf("\ne:编码(Encode)\nd:解码(Decode)\nq:退出(Quit))\n");
printf("请选择操作：");
scanf("%c",&c);
do
{
    switch (c)
    {
        case'e':
        {
            printf("请输入想要编码的文本：");
            fflush(stdin);
            gets(ch);
            printf("编码结果为：\n");
            HuffmanEncode(hufmTree, ch, string, arr);
            printf("\n");
            printf("\ne:编码(Encode)\nd:解码(Decode)\nq:退出(Quit))\n");
```

```
        printf("请选择操作：");
        break;
    }
    case’d’:
    {
        printf("请输入想要解码的电文：");
        fflush(stdin);
        gets(ch);
        printf("解码结果为：\n");
        HuffmanDecode(ch, hufmTree, string);
        printf("\n");
        printf("\ne:编码(Encode)\nd:解码(Decode)\nq:退出(Quit))\n");
        printf("请选择操作：");
        break;
    }
    case’q’:
    {
        exit(0);
        break;
    }
    default:
    {
```

```
                    exit(0);
                    break;
                }
            }
            scanf("%c",&c);
        } while (c != 'q');
        free(arr);
        free(hufmTree);

        return 0;
}
```

# 附录

## 附录一

```
clc,clear
data=xlsread('data.xlsx');
x=data(:,1);
y=data(:,2);
label={'南宁','河内','曼谷','金边','吉隆坡','新加坡','文莱','马尼拉'};
plot(x,y,'o')
for i=1:length(x)
   text(x(i),y(i),label{i})
end
A=[x,y];
xlswrite('data.xlsx',A)
```

## 附录二

```
clc,clear
n=8;
graph=zeros(n);
pos=xlsread('data.xlsx');
for i=1:n
    for j=i+1:n
        graph(i,j)=distance(pos(i,1),pos(i,2),pos(j,1),pos(j,2));
    end
end
graph=graph+graph';
xlswrite('graph.xlsx',graph);
```

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 65.43699 | 238.0336 | 241.6299 | 417.5284 | 442.6613 | 383.4371 | 301.2042 |
| 65.43699 | 0 | 175.0257 | 192.4396 | 365.1685 | 395.2177 | 367.7635 | 326.6956 |
| 238.0336 | 175.0257 | 0 | 93.53609 | 212.2499 | 255.1568 | 326.9067 | 396.5047 |
| 241.6299 | 192.4396 | 93.53609 | 0 | 176.1732 | 202.8004 | 233.5487 | 320.6571 |
| 417.5284 | 365.1685 | 212.2499 | 176.1732 | 0 | 54.56189 | 257.9651 | 438.7368 |
| 442.6613 | 395.2177 | 255.1568 | 202.8004 | 54.56189 | 0 | 226.762 | 426.0434 |
| 383.4371 | 367.7635 | 326.9067 | 233.5487 | 257.9651 | 226.762 | 0 | 225.3619 |
| 301.2042 | 326.6956 | 396.5047 | 320.6571 | 438.7368 | 426.0434 | 225.3619 | 0 |

## 附录三

```
clc,clear;
global graph
global n
graph=xlsread('graph.xlsx');%read graph
n=size(graph,1);
%随机生成处解
sol_new=[1:n 1];
for i=1:10
    m=ceil(rand()*(n-1)+1);
    n=ceil(rand()*(n-1)+1);
    temp=sol_new(m);
    sol_new(m)=sol_new(n);
    sol_new(n)=temp;
end
sol_best=sol_new;
sol_current=sol_new;
Etemp=fun(sol_new);
Ebest=Etemp;
Ecurrent=Etemp;
%init
t0=97;tf=3;L=10000;t=t0;at=0.99;
%main code
tic
i=1;
while t>=tf
    for k=1:L
        if(rand()<0.5)
            %two exchange
            c1=ceil(rand()*(n-1)+1);
            c2=ceil(rand()*(n-1)+1);
            while(c1==c2)
                c1=ceil(rand()*(n-1)+1);
                c2=ceil(rand()*(n-1)+1);
            end
            temp=sol_new(c1);
            sol_new(c1)=sol_new(c2);
            sol_new(c2)=temp;
        else
            %three exchange
            %make c1!=c2!=c3
            c1=ceil(rand()*(n-1)+1);
            c2=ceil(rand()*(n-1)+1);
            c3=ceil(rand()*(n-1)+1);
            while(c1==c2)||(c2==c3)||(c1==c3)||(abs(c1-c2)==1)
```

```
                c1=ceil(rand()*(n-1)+1);
                c2=ceil(rand()*(n-1)+1);
                c3=ceil(rand()*(n-1)+1);
            end
            %make c1<c2<C3
            temp1=c1;
            temp2=c2;
            temp3=c3;
            if (c1<c2)&&(c2<c3)
            elseif (c1<c3)&&(c3<c2)
                c2=temp3;c3=temp2;
            elseif (c2<c1)&&(c1<c3)
                c1=temp2;c2=temp1;
            elseif (c2<c3)&&(c3<c1)
                c1=temp2;c2=temp3;c3=temp1;
            elseif (c3<c1)&&(c1<c2)
                c1=temp3;c2=temp1;c3=temp2;
            elseif (c3<c2)&&(c2<c1)
                c1=temp3;c2=temp2;c3=temp1;
            end
            templist=sol_new((c1+1):(c2-1));
            sol_new((c1+1):(c1+c3-c2+1))=sol_new(c2:c3);
            sol_new((c1+c3-c2+2):c3)=templist;
        end
        Etemp=fun(sol_new);
        %记录历代能量
        Ehis(i)= Ecurrent;
        i=i+1;
        if(Etemp<Ecurrent)
            Ecurrent=Etemp;
            sol_current=sol_new;
            if(Etemp<Ebest)
                Ebest=Etemp;
                sol_best=sol_new;
            end
        else
            if rand<exp(-(Etemp-Ecurrent)./t)
                Ecurrent=Etemp;
                sol_current=sol_new;
            else
                sol_new=sol_current;
            end
        end
    end
```

```
    t=t.*at;
end
toc
%存结果
xlswrite('result_sol.xlsx',sol_best);
xlswrite('result_E.xlsx',Ebest);
plot(1:i-1,Ehis);
xlabel('代数')
ylabel('能量')
disp(sol_best)
disp(Ebest)
function E=fun(circle)
global graph
global n
n=8;
length=0;
for i=1:n
    length=length+graph(circle(i),circle(i+1));
end
E=length;
```