哈夫曼编码

一、算法思路

1、以编码字符集中每个字符c的出现频率f(c)，作为贪心选择依据，对字符集进行由小到大的排序，每个字符对应于一个只包含一个结点的子树。

2、先合并最小频率的2个字符对应的子树，计算合并后的子树中这个字符出现的频率总和。

3、重新排序各个子树。

4、对上诉排序后的子树序列进行合并。

5、重复上述过程，将全部结点合并成1棵完整的二叉树，称为编码树T。

6、对二叉树中的边赋予0、1，得到各字符的变长编码。

二、程序实现

1、全局变量及结构体

1. **typedef** **char** \*\*HuffmanCode; //哈夫曼编码类型
2. **typedef** **struct**
3. {
4. **char** data;                  //编码树结点的字符
5. **int** weight;                 //当前字符的出现频率
6. **int** parent, lchild, rchild; //当前结点的父亲、左儿子和右儿子
7. } HuffmanNode;                  //哈夫曼树结点类型
9. **typedef** **struct**
10. {
11. HuffmanNode arr[N]; //编码树结点数组
12. **int** n, count;       // n表示字符的种类数，count表示字符的总数目
13. } HuffmanTree;          //哈夫曼树类型
15. **int** root, match[N]; // root表示根结点，match用来存储字符-下标映射关系
16. **bool** flag[N];       //标记当前字符是否为第一次出现

2、函数定义

1. **void** initHuffmanTree(HuffmanTree &T);                //初始化哈夫曼树
2. **void** createHuffmanTree(HuffmanTree &T);              //构造哈夫曼树
3. **void** createHuffmanCode(HuffmanTree T, HuffmanCode &HC); //构造哈夫曼编码
4. **void** printHuffmanInfo(HuffmanTree T, HuffmanCode HC); //打印相关信息
5. **bool** findTwoMin(HuffmanTree T, **int** &m1, **int** &m2, **int** num); //寻找当前最小频率的两个字符结点

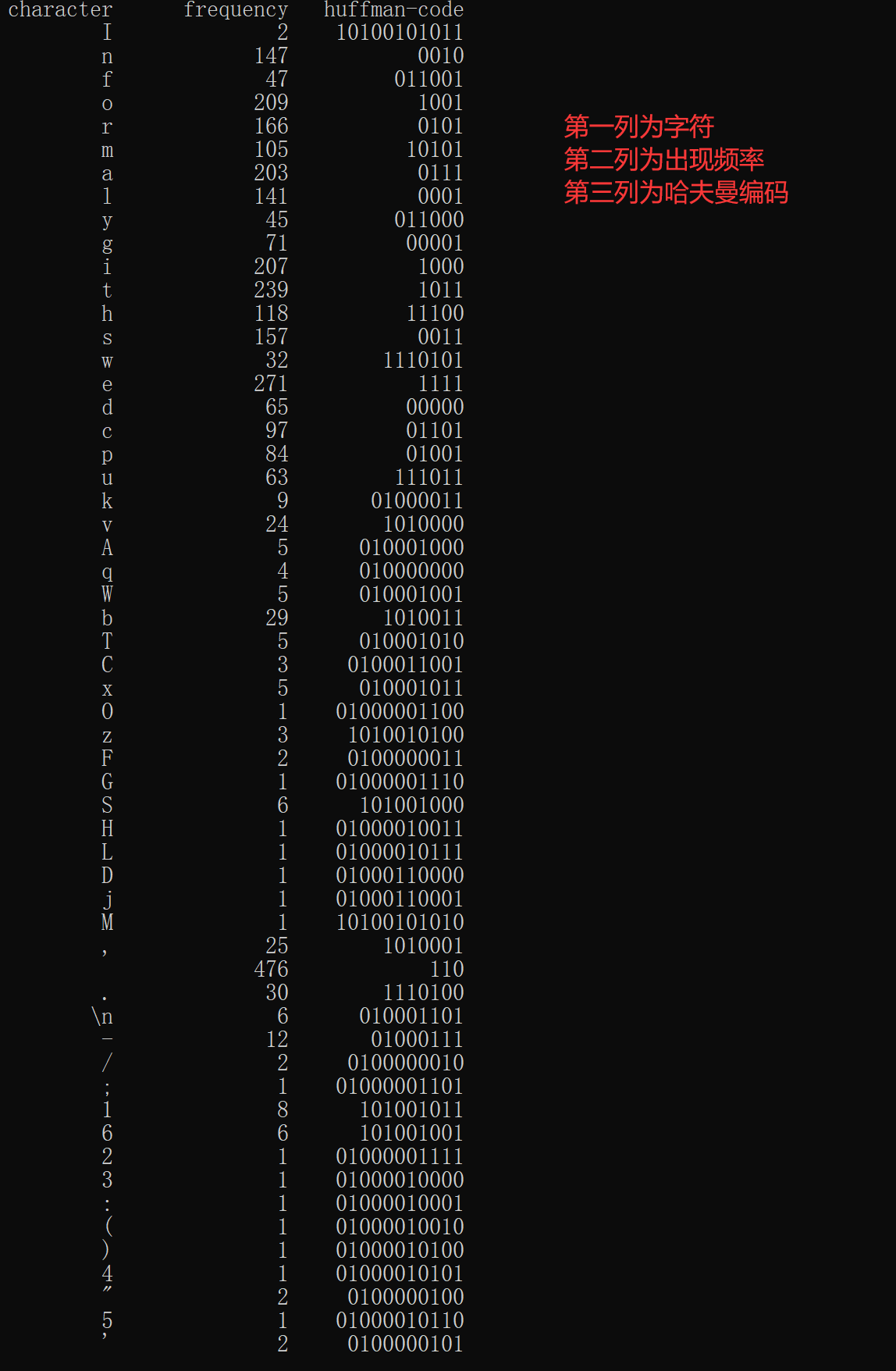
3、核心算法

1. //构造哈夫曼树
2. **void** createHuffmanTree(HuffmanTree &T)
3. {
4. **int** m1 = 0, m2 = 0; //最小频率的两个结点
5. **int** i = T.n;
7. //循环寻找最小频率的两个结点
8. **while** (findTwoMin(T, m1, m2, i))
9. {
10. //合并两个结点
11. T.arr[m1].parent = T.arr[m2].parent = i;
12. T.arr[i].lchild = m1, T.arr[i].rchild = m2;
13. T.arr[i].parent = -1;
14. T.arr[i].weight = T.arr[m1].weight + T.arr[m2].weight;
15. i++;
16. }
18. root = m1; //哈夫曼树的根结点
19. }
21. //构造哈夫曼编码
22. **void** createHuffmanCode(HuffmanTree T, HuffmanCode &HC)
23. {
24. HC = (**char** \*\*)malloc(**sizeof**(**char** \*) \* (T.n));
26. **char** tmp[N];
27. **for** (**int** i = 0; i < T.n; i++)
28. {
29. //自底向上构建字符的逆向哈夫曼编码
30. **int** p = T.arr[i].parent;
31. **int** c = i, top = 0;
33. **while** (p != -1)
34. {
35. **if** (T.arr[p].lchild == c)
36. tmp[top++] = '0';
37. **else**
38. tmp[top++] = '1';
39. c = p, p = T.arr[p].parent;
40. }
42. HC[i] = (**char** \*)malloc(**sizeof**(**char**) \* N);
44. **int** j = 0;
45. //倒置处理，获取正向的哈夫曼编码
46. **while** (top != 0)
47. HC[i][j++] = tmp[--top];
48. HC[i][j] = '\0';
49. }
50. }

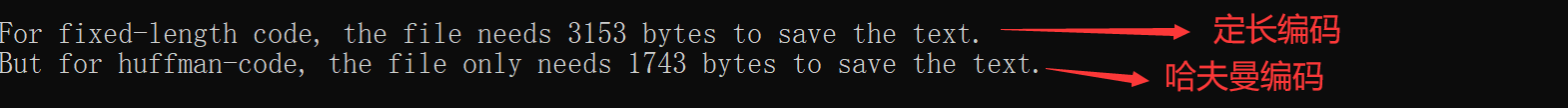
三、算法输出

1、方案1（不替换）

（1）出现频率和哈夫曼编码

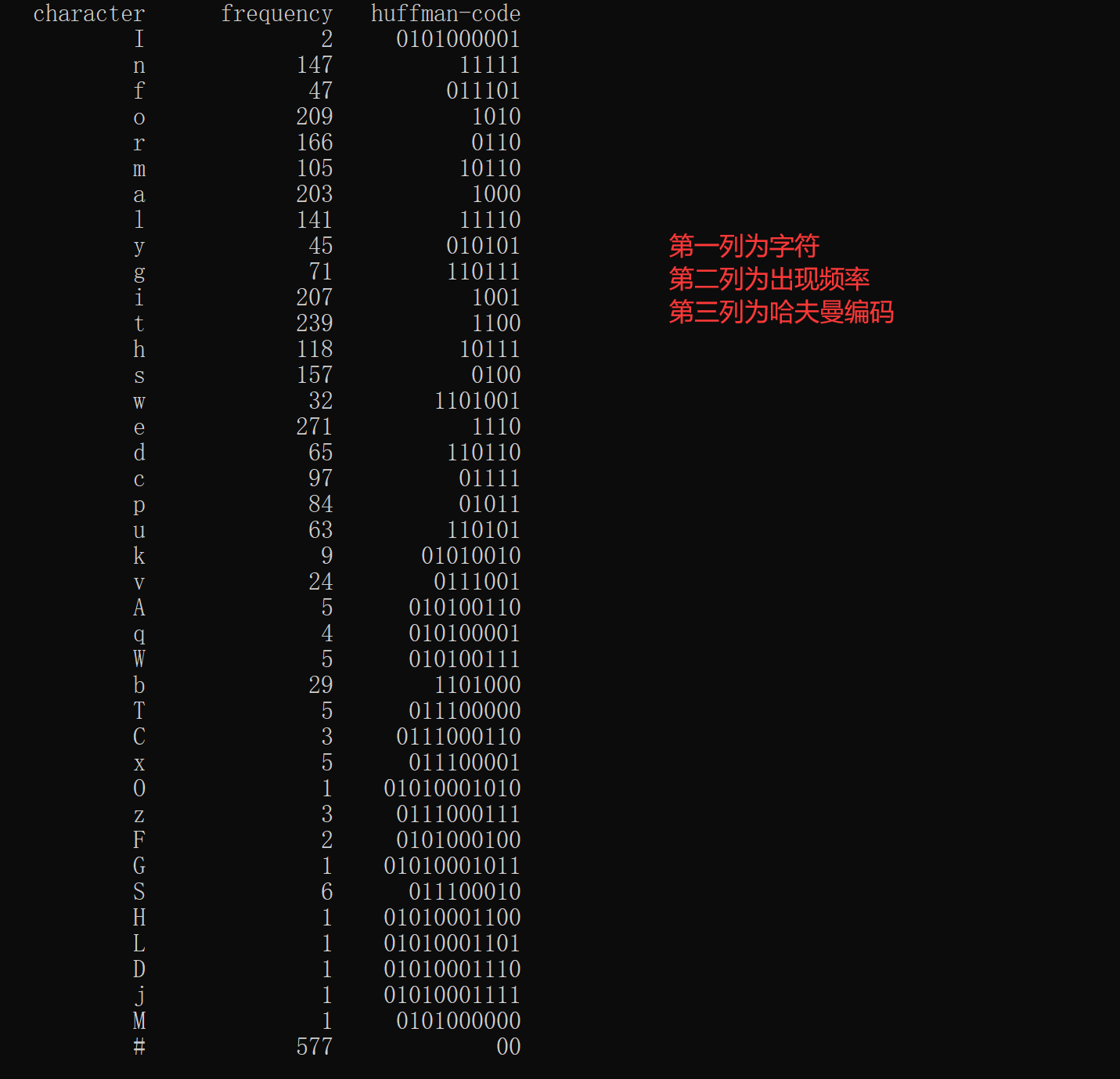


（2）定长编码、哈夫曼编码所需的存储比特数

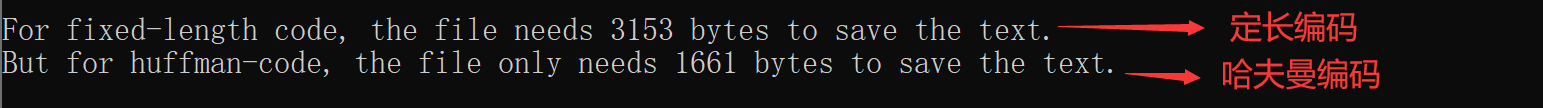


2、方案2（替换）

（1）出现频率和哈夫曼编码



（2）定长编码、哈夫曼编码所需的存储比特数



四、时空复杂度分析

算法初始化哈夫曼树需要O(n)，合并两个最小频率的结点需要O(nlogn)，故时间复杂度为O(nlogn)；存储编码数组需要二维数组，故空间复杂度为O(n^2)。