1 实验题目

Huffman 树及 Huffman 编码的算法实现

2 实验目的

1. 了解该树的应用实例，熟悉掌握 Huffman 树的构造方法及 Huffman 编码的应用，
2. 了解 Huffman 树在通信、编码领域的应用过程。

3 实验要求

1. 输入一段100—200字的英文短文，存入一文件 `a` 中。
2. 写函数统计短文出现的字母个数 n 及每个字母的出现次数
3. 写函数以字母出现次数作权值，建 Huffman 树（n 个叶子），给出每个字母的 Huffman 编码。
4. 用每个字母编码对原短文进行编码，码文存入文件 `b` 中。
5. 用 Huffman 树对 `b` 中码文进行译码，结果存入文件 `c` 中，比较 `a`、`c` 是否一致，以检验编码、译码的正确性。

4 实验内容和实验步骤

4.1 需求分析

陈述程序设计的任务，强调程序要做什么，明确规定：

1. 输入的形式和输入值的范围；
2. 输出的形式；
3. 程序所能实现的功能；

4.2 概要设计

4.2.1 数据结构定义

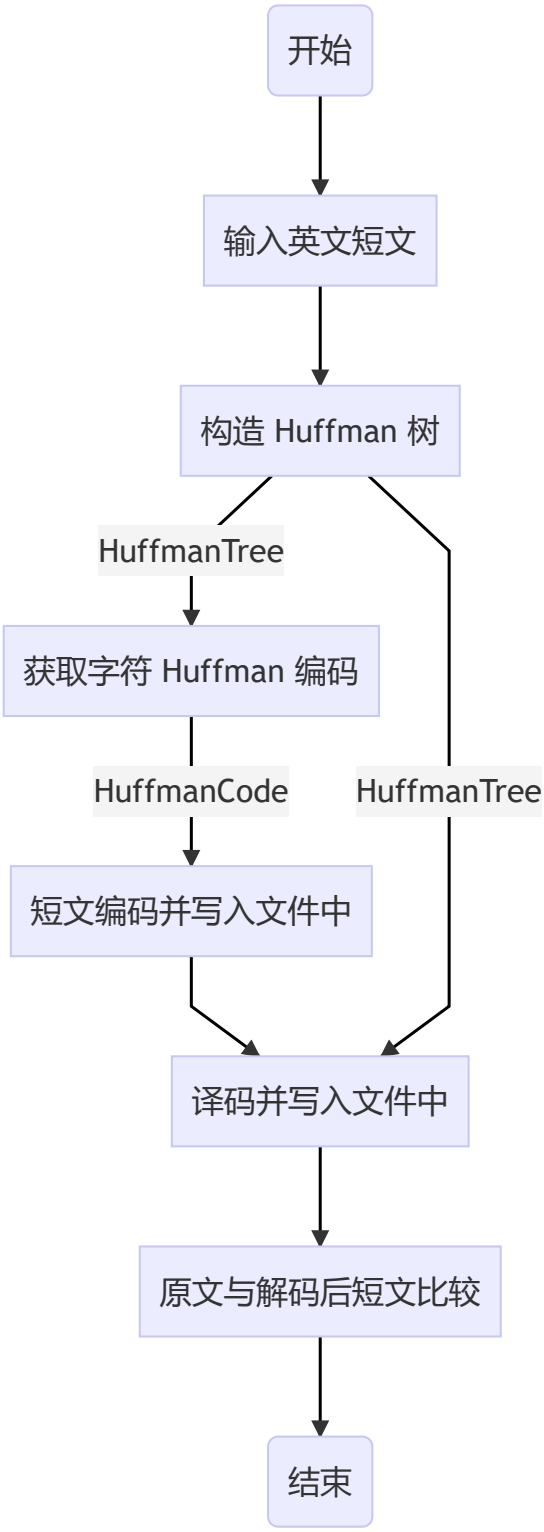
Huffman 树定义

```
typedef struct {
    // 字符
    char letter;
    // 权重
    unsigned int weight;
    // 父节点、左孩子、右孩子下标
    unsigned int parent, left, right;
} Node, *Tree;
```

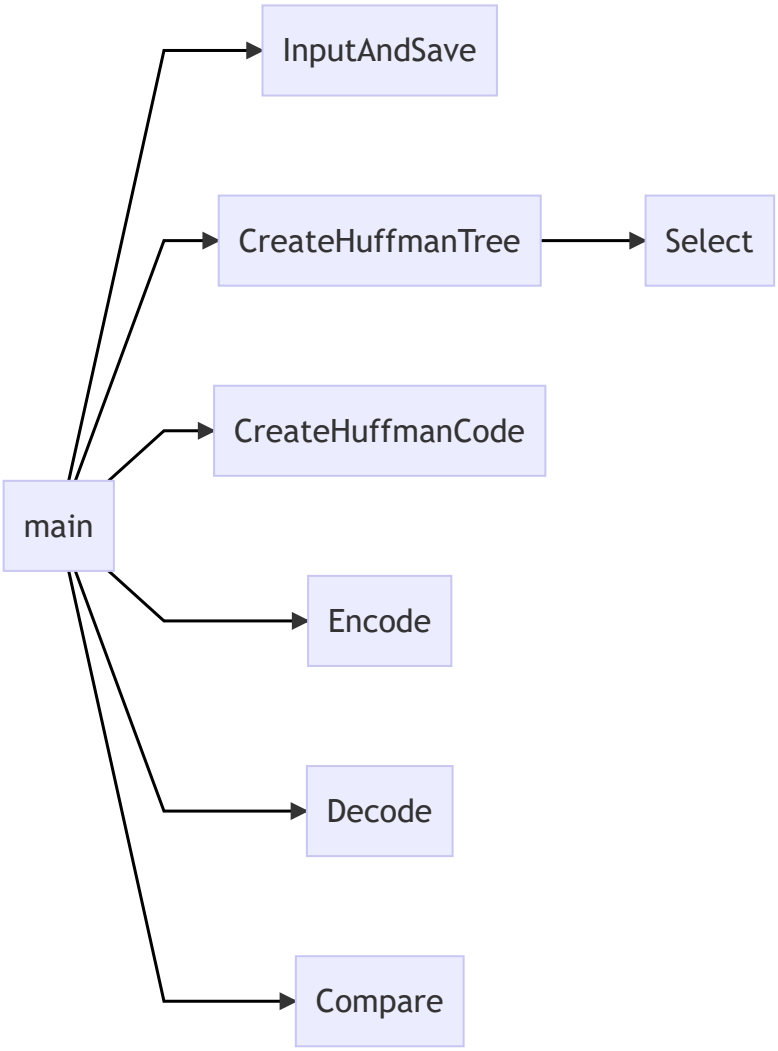
Huffman 编码定义

```
typedef char *Code;
```

4.2.2 主程序流程



4.2.3 各程序模块之间的调用关系



4.3 详细设计

4.3.1 主程序入口

```
int main() {
    // 输入英文文章并保存到 a.txt 中
    InputAndSave("a.txt");

    // 构造 Huffman 树
    CreateHuffmanTree("a.txt", ht, n);

    // 获取 Huffman 编码
    CreateHuffmanCode(hc, ht, n);

    // 短文编码
    Encode("a.txt", "b.txt", hc);

    // 短文解码
```

```
    Decode("b.txt", "c.txt", ht, n);

    // 原文与解码后短文比较
    printf("a.txt is %s to c.txt",
           Compare("a.txt", "c.txt") ? "equal" : "NOT equal");

    // 释放空间
    free(ht);
    return 0;
}
```

4.3.2 文章读入

```
void InputAndSave(const char *filename) {
    // 新建 / 打开文件
    FILE *fp = fopen(filename, "w");
    printf("Please input an essay, end with an enter: ");
    while (true) {
        // 从 stdin 中读取一个字符
        char ch = getchar();
        // 若字符为'\n'，退出循环
        if (ch == '\n')
            break;
        // 写入文件中
        fputc(ch, fp);
    }
    // 关闭文件
    fclose(fp);
}
```

4.3.3 构造 Huffman 树

以 `Hello, World!` 为例

index	letter	weight	parent	left	right
1	空格	1	11	0	0
2	!	1	11	0	0
3	,	1	12	0	0
4	H	1	12	0	0
5	W	1	13	0	0
6	d	1	13	0	0
7	e	1	14	0	0
8	l	3	17	0	0
9	o	2	15	0	0
10	r	1	14	0	0
11		2	15	2	1
12		2	16	4	3
13		2	16	6	5
14		2	17	10	7
15		4	18	11	9
16		4	18	13	12
17		5	19	14	8
18		8	19	16	15
19		13	0	17	18

```
void CreateHuffmanTree(const char *filename, Tree &ht, int &n)
{
    // 从文件中逐个读取字符并计数
    FILE *fp = fopen(filename, "r");
    int count[128] = {0};
    n = 0;
    while (true) {
        char ch = fgetc(fp);
        if (ch == EOF)
            break;
        if (count[ch] == 0)
```

```
            n += 1;
        count[ch] += 1;
    }
    fclose(fp);

    // 申请 Huffman 树空间
    ht = (Tree)malloc(2 * n * sizeof(Node));

    // 不使用 0 号单元
    // 将字符及权重存入树中
    // 置 parent, left, right 为 0
    Tree p = ht + 1;
    for (int i = 0; i < 128; i++) {
        // ......
    }

    // 构建 Huffman 树
    for (int i = n + 1; i < 2 * n; i++) {
        // 在 ht[1..i-1] 中选择 parent=0 且 weight 最小的两个结点
        int s1, s2;
        Select(ht, i - 1, s1, s2);

        // 更新子节点与父节点的信息
        // ......
    }
}

void Select(Tree ht, int n, int &s1, int &s2) {
    // ......
}
```

4.3.4 获取 Huffman 编码

```
void CreateHuffmanCode(Code hc[], Tree ht, int n) {
    // 编码临时存储空间
    char *cd = (char *)malloc(n * sizeof(char));
    cd[n - 1] = '\0';

    // 从叶子节点逆向求编码
    // ......

    // 释放临时存储空间
    free(cd);
}
```

4.3.5 短文编码

```
void Encode(const char *src, const char *dst, Code hc[]) {
    FILE *fsrc = fopen(src, "r");
    FILE *fdst = fopen(dst, "w");

    while (true) {
        // 从 src 中读取字符
        char ch = fgetc(fsrc);
        if (ch == EOF)
            break;
        // 查询编码后写入 dst
        // ......
    }

    fclose(fsrc);
    fclose(fdst);
}
```

4.3.6 短文解码

```
void Decode(const char *src, const char *dst, Tree ht, int n)
{
    FILE *fsrc = fopen(src, "r");
    FILE *fdst = fopen(dst, "w");

    // 寻找叶子节点
    // 到达叶子结点后将对应字符写入 dst 中
    // ......

    fclose(fsrc);
    fclose(fdst);
}
```

4.3.7 原文与解码后短文比较

```
bool Compare(const char *first, const char *second) {
    FILE *f1 = fopen(first, "r");
    FILE *f2 = fopen(second, "r");

    // 逐字符比较两个文件
    while (!feof(f1) && !feof(f2)) {
        char c1 = fgetc(f1);
        char c2 = fgetc(f2);
```

```
        // 字符不相同，跳出循环
        if (c1 != c2)
            break;
    }

    // 两个文件未同时到达末尾
    int res = true;
    if (!feof(f1) || !feof(f2))
        res = false;

    fclose(f1);
    fclose(f2);
    return res;
}
```

4.4 调试分析

1. 调试过程中所遇到的问题及解决方法
2. 算法的时空分析

时间复杂度	函数名
$O(1)$	
$O(n)$	
$O(n^2)$	

5 实验用测试数据和相关结果分析

5.1 实验结果

列出对于给定的输入所产生的输出结果。若可能，测试随输入规模的增长所用算法的实际运行时间的变化。

5.2 实验总结

有关实验过程中的感悟和体会、经验和教训等。