1. 设计说明书：
   1. 数据结构设计：
2. class hf\_node：
3. 作用：

该类实现对文件中的256个ascii码出现次数进行统计，构建哈夫曼树和存储字符对应的编码。

1. 成员变量：
2. int ascii；//用于记录当前节点的ascii码值
3. int weight；//用于记录当前ascii码在文件中现的次数。
4. hf\_node\* rchild；//当前节点在哈夫曼树中的右孩子
5. hf\_node\* lchild；//左孩子
6. string code；//当前节点对应的哈夫曼编码
7. 成员函数：
8. hf\_node(int ascii\_num, int weight)；//带参构造函数，创建一个记录对应ascii\_num的节点；
9. hf\_node()；//默认构造函数，置空；
10. struct cmp\_g：
11. 作用：

该结构本身不是一个数据结构，不存放数据，只是作为一个实现priority\_queue中的比较方式的仿函数。

1. 成员变量：

无；

1. 成员函数：
2. bool operator()(hf\_node\* a, hf\_node\* b);//重载（）运算符，实现优先队列中两个hf\_node比较大小的仿函数。
3. struct temp\_node：
4. 作用：

该结构用于解压时临时存储各个ascii码对应的码值以及出现次数，相当于一个简化版的hf\_node，方便构造hf\_node优先队列。

1. 成员变量：
2. int ascii\_num;//用于记录当前节点ascii码值
3. int times;//功能同weight
4. 成员函数：

无；

* 1. 主要技术难点及解决方案：

1. 难点：
2. 什么是哈夫曼树：

由哈夫曼树的定义，树中权重最大（即出现次数最多的）应更靠近树根，即每次取出一组元素中最小的两个元素相加形成一个新的元素再并删除这刚才两个元素，再将新的元素放回组中，依次操作，最后能得到一个以最大权重到最小权重为叶子节点的哈夫曼树。

1. 采用什么什么方法构建哈夫曼树：

由哈夫曼树的定义，可知我们每次对一组元素进行操作之后，必须对则组元素重新排序，由此可想到自动排序的数据结构——优先队列。