哈尔滨工业大学计算学部

实验报告

课程名称：数据结构与算法

课程类型：专业核心基础（必修）

实验项目：树形结构及其应用

实验题目：哈夫曼编码与译码方法

实验日期：2022/10/23

班级：2103401

学号：2021112893

姓名：余弦

|  |  |  |
| --- | --- | --- |
| 设计成绩 | 报告成绩 | 指导老师 |
|  |  | 张岩 |

1. **实验目的**

通过构建哈夫曼树，理解哈夫曼编码和解码的具体过程，从而掌握树形结构的构造和应用。

**二、实验要求及实验环境**

1. 从文件中读入任意一篇英文文本文件，分别统计英文文本文件中各字符（包 括标点符号和空格）的使用频率；

2. 根据已统计的字符使用频率构造哈夫曼编码树，并给出每个字符的哈夫曼编 码（字符集的哈夫曼编码表）；

3. 将文本文件利用哈夫曼树进行编码，存储成压缩文件（哈夫曼编码文件）；

4. 计算哈夫曼编码文件的压缩率；

5. 将哈夫曼编码文件译码为文本文件，并与原文件进行比较。

6. 能否利用堆结构，优化的哈夫曼编码算法。

**三、设计思想**（本程序中的用到的所有数据类型的定义，主程序的流程图及各程序模块之间的调用关系、核心算法的主要步骤）

1．逻辑设计

哈夫曼树：从根节点往下如果指向左则为0，指向又则为1。只有叶节点储存所编码的数据。

二叉堆：利用二叉堆来储存节点，便于使用最小的两个节点信息。

2．物理设计

哈夫曼树采用链式存储结构进行构建，节点为结构体，包含了（int）数据内容，（int）节点权重，（节点指针）左指针和（节点指针）右指针。

二叉堆则利用了顺序存储结构，利用c++STL中的vector数组来，并且利用索引的规律，构建出一个二叉堆。

3.函数说明:

Class heap:小根堆类，用于存储节点和返回最小权重的节点

--MinHeap(vector<Node\*> &nodes)：构造函数，创建并初始化小根堆

--void MinHeapify(int i)：利用小根堆的特性对小根堆中的数组元素进行顺序的排列

--Node\* ExtractMin()：返回当前小根堆中的最小节点的指针

--void Insert(Node\* node)：插入节点到当前的小根堆中

--bool Empty()：判断当前小根堆是否为空

void Get\_frequency(int frequency[])：读取文本文件，通过遍历将各个字符的频率输入到frequency数组当中，用于之后哈夫曼树的构造

Tree merge\_nodes(Tree a, Tree b)：将两个树合并到一个新的节点，并返回这一个节点

Tree Construct\_Huffman\_Tree(int frequency[], vector<Node\*> &nodes)利用节点和频率数组来构建哈夫曼树，具体方法为先将所有字符都创建为一个节点，将节点存储到小根堆中，利用循环合并小根堆中最小的两个节点，并将新节点加入到小根堆当中。直到小根堆中只剩下一个节点为止。

void Huffman\_code(Tree T,HASH2 &codes,string s)利用递归函数得到出所有字符的哈夫曼编码

void Print\_codes(HASH2 codes)打印出所有字符的哈夫曼编码

void Output\_codes(HASH2 codes,string outpath, string inpath)将原文本所有字符替换为哈夫曼编码输出到文件当中

void Decode(Tree T, string outpath, string inpath利用哈夫曼编码文件进行解码，并将结果输出到解码文本文件当中。

4.流程图:哈夫曼树构建过程已经包含在流程图当中



Figure 流程图

**四、测试结果**

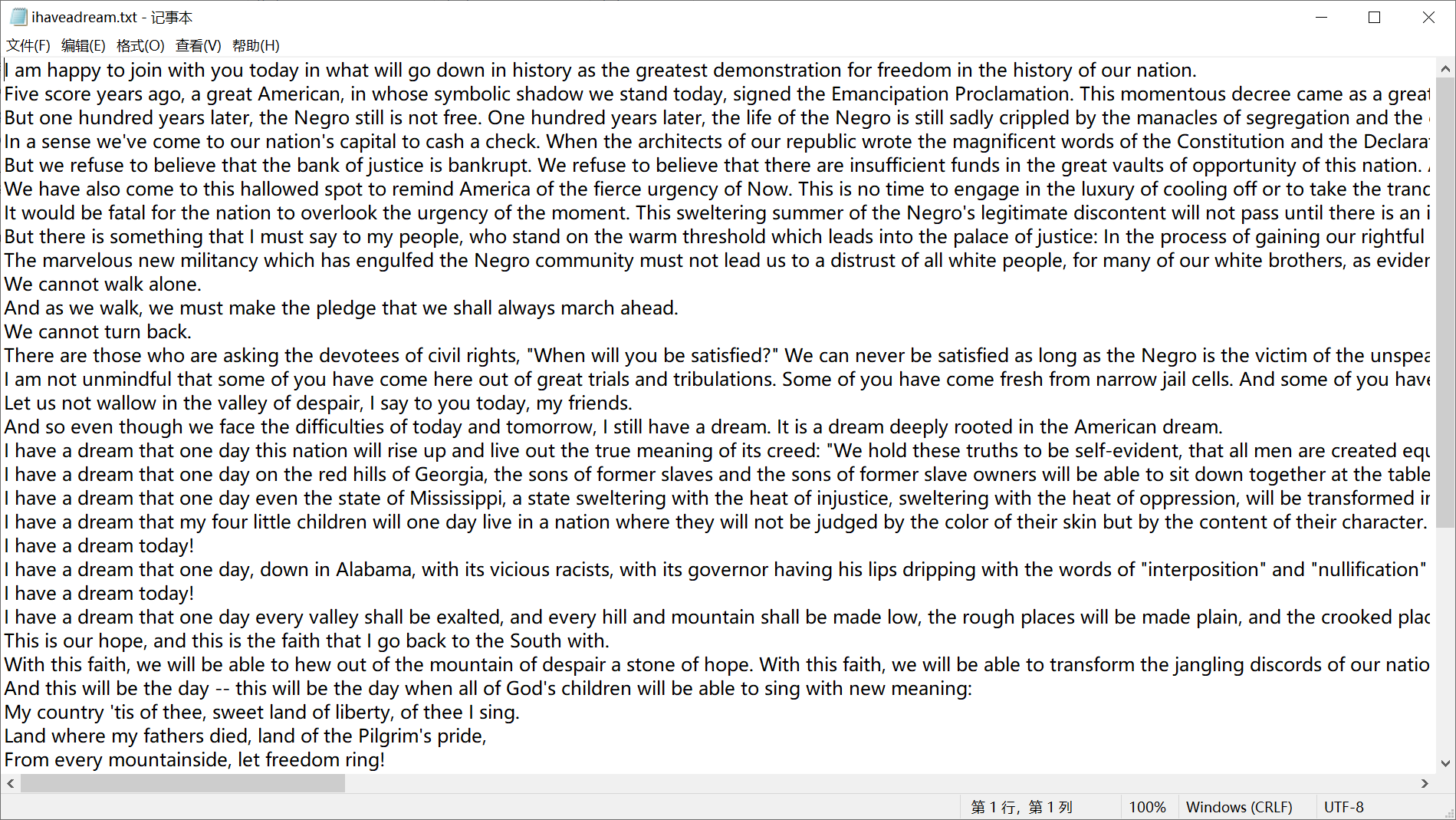
本次实验所编码的文件为马丁路德金的I have a dream，

Figure 文本文件

以下为实验的结果

1. 输出所有字符的哈夫曼编码。由于空格和换行直接显示效果不好，故进行了特判，在显示时以space和/n来分别展示空格和换行符。

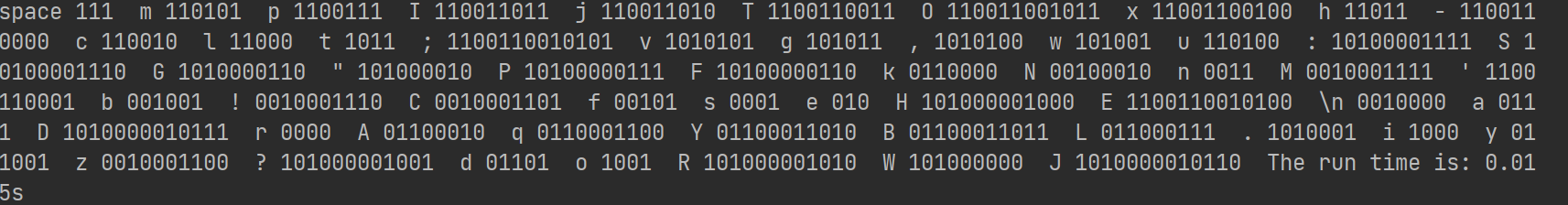


Figure 字符编码结果

1. 、将原文本转换为编码，并输出到文件中，打开文件可以看到如下编码

Figure 文件编码结果

1. 为了程序的简单和展示的方便，这里使用了字符串来对编码进行保存。在真实哈夫曼编码中，应该采取二进制存储的方式来对编码进行存储，从而达到数据压缩的方式。
2. 解码结果展示：将解码的结果输出到文本文件中，与原文本进行对比，完全一致

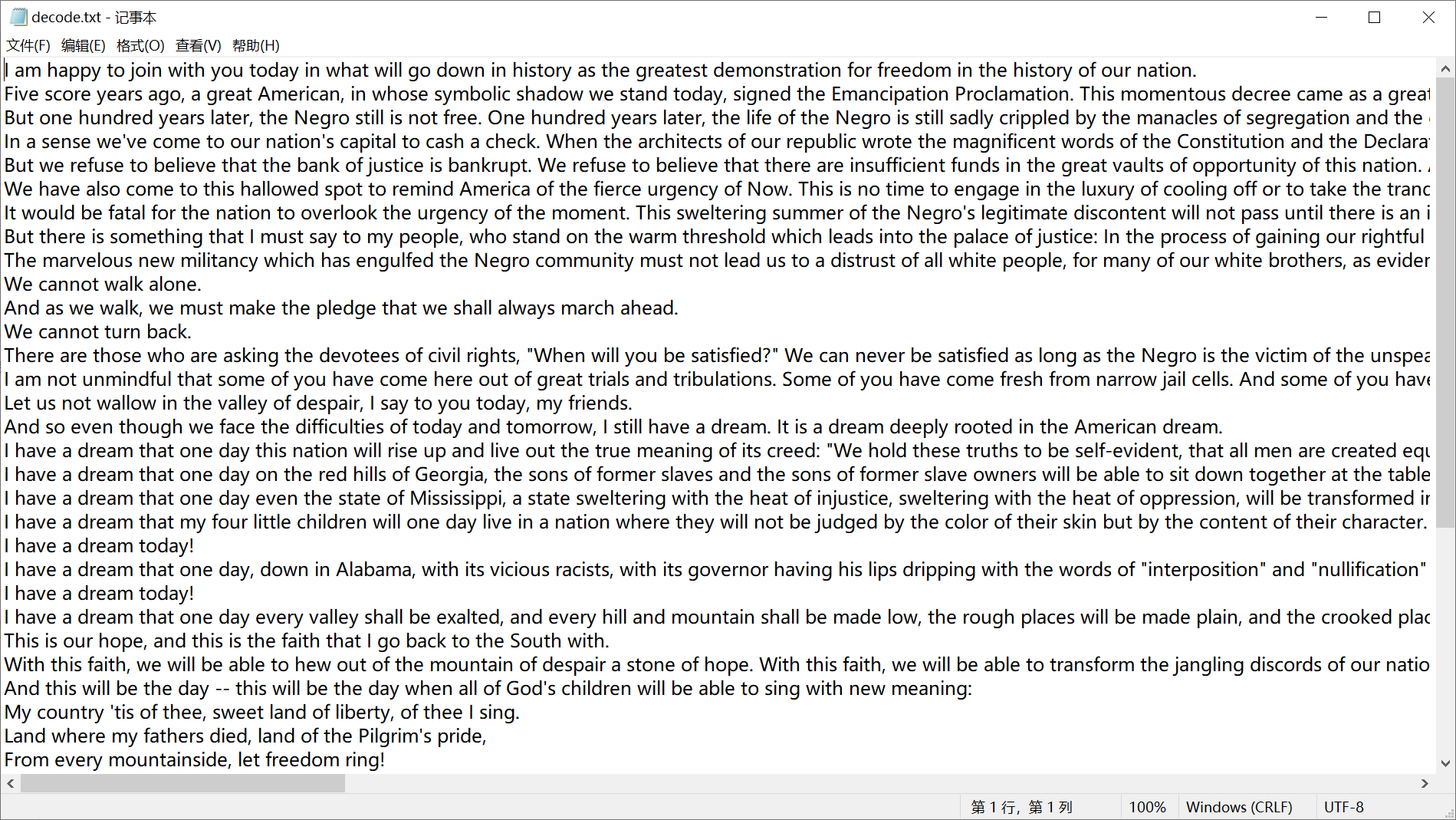


Figure 文件解码结果

1. 压缩率的计算：

原文本的文件大小为9kb，编码文件的大小为39kb。由于在模拟过程中采取了char来进行存储，原本1bit就能储存的通过字符来储存变成了8bits，故在计算压缩率时需要进行转换

**五、经验体会与不足**

本次实验仅仅使用了01字符串对编码进行了模拟，并没有使用真正的位来储存编码，主要原因是因为编码的知识还储存的不够。这是有待改进的地方。另外有些算法并不是最优的算法，其时空复杂度有待改进。

通过这次实验我体会了哈夫曼编码作为一种可变长的编码方式的优势。同时理解并掌握了树形结构的应用。

**六、附录：源代码（带注释）**

**main.cpp**

1. #include <algorithm>
2. #include <string>
3. #include <vector>
4. #include <unordered\_map>
5. #include "Huffman\_Tree.h"
6. #include "iostream"
7. **using** **namespace** std;
8. **const** **int** N = 128;
9. **int** frequency[N];
11. **void** Get\_frequency(**int** frequency[]);
12. Tree Construct\_Huffman\_Tree(**int** frequency[], vector<Node\*> &nodes);
13. **void** Huffman\_code(Tree T,HASH2 &codes,string s);
14. **void** Output\_codes(HASH2 codes,string outpath, string inpath);
15. **void** Decode(Tree T, string outpath, string inpath);
16. **int** main() {
17. HASH2 code;
18. vector<Node\*> nodes;
19. Get\_frequency(frequency);
20. Tree T = Construct\_Huffman\_Tree(frequency, nodes);
21. Huffman\_code(T, code, "");
22. Output\_codes(code, "C:/Users/PC/Desktop/things/datastructure/exp/Huffman/code.txt", "C:/Users/PC/Desktop/things/datastructure/exp/Huffman/ihaveadream.txt");
23. Decode(T, "C:/Users/PC/Desktop/things/datastructure/exp/Huffman/decode.txt", "C:/Users/PC/Desktop/things/datastructure/exp/Huffman/code.txt");
24. **return** 0;
25. }

FUNC.cpp

1. //
2. // Created by PC on 2022/10/12.
3. //
4. #include <iostream>
5. #include <fstream>
6. #include <string>
7. #include <vector>
8. #include "Huffman\_Tree.h"
9. #include <algorithm>
10. #include <unordered\_map>
11. #include <bitset>
13. **using** **namespace** std;
14. //实现小根堆
15. **class** MinHeap{
16. **public**:
17. //储存的元素
18. vector<Node\*> heap;
19. //类的函数
20. MinHeap(vector<Node\*> &nodes){
21. **for**(**int** i = 0; i < nodes.size(); i++){
22. heap.push\_back(nodes[i]);
23. }
24. **for**(**int** i = heap.size() / 2 - 1; i >= 0; i--){
25. MinHeapify(i);
26. }
27. }
28. //插入节点,i是插入节点的位置
29. **void** MinHeapify(**int** i){
30. **int** l = 2 \* i + 1;
31. **int** r = 2 \* i + 2;
32. **int** smallest = i;
33. **if**(l < heap.size() && heap[l]->weight < heap[i]->weight){
34. smallest = l;
35. }
36. **if**(r < heap.size() && heap[r]->weight < heap[smallest]->weight){
37. smallest = r;
38. }
39. **if**(smallest != i){
40. swap(heap[i], heap[smallest]);
41. MinHeapify(smallest);//递归调用
42. }
43. }
44. //提取最小的节点并删除
45. Node\* ExtractMin(){
46. Node\* min = heap[0];
47. heap[0] = heap[heap.size() - 1];
48. heap.pop\_back();
49. MinHeapify(0);
50. **return** min;
51. }
53. **void** Insert(Node\* node){
54. heap.push\_back(node);
55. **int** i = heap.size() - 1;
56. **while**(i > 0 && heap[(i - 1) / 2]->weight > heap[i]->weight){
57. swap(heap[i], heap[(i - 1) / 2]);
58. i = (i - 1) / 2;
59. }
60. }
62. **bool** Empty(){
63. **return** heap.empty();
64. }
66. };
68. **void** Get\_frequency(**int** frequency[]){//统计字符频率
69. //读取文件
70. ifstream infile;
71. infile.open("C:/Users/PC/Desktop/things/datastructure/exp/Huffman/ihaveadream.txt");
72. **if**(!infile.is\_open()){
73. cout<<"文件打开失败"<<endl;
74. **return**;
75. }
76. //统计字符频率
77. **char** ch;
78. **while**(infile.get(ch)){
79. frequency[ch]++;
80. }
81. }
83. **bool** compare(Node\* &a,Node\* &b){
84. **return** (a -> weight) < (b -> weight);
85. }
87. Tree merge\_nodes(Tree a, Tree b){//合并两个节点
88. Tree res = nullptr;
89. res=(Tree) malloc(**sizeof**(Node));
90. **if**(!res) **return** nullptr;
91. res -> weight = (a -> weight) + (b -> weight);
92. res -> data = -1;
93. res -> left = a;
94. res -> right = b;
95. **return** res;
96. }
98. Tree Construct\_Huffman\_Tree(**int** frequency[], vector<Node\*> &nodes){//构造哈夫曼树
99. //将nodes转换为小根堆
101. **for**(**int** i = 0 ; i < 128 ; i++){
102. **if**(frequency[i]){//如果频率不为0
103. Node\* nn = (Node\*)malloc(**sizeof**(Node));
104. nn -> data = i;
105. nn -> weight = frequency[i];
106. nn -> left = nullptr;
107. nn -> right = nullptr;
108. nodes.push\_back(nn);//将节点加入到nodes中
109. }
110. }
111. MinHeap heap(nodes);//构造小根堆
112. **while**(!heap.Empty()){
113. Tree a = heap.ExtractMin();
114. Tree b = heap.ExtractMin();
115. Tree c = merge\_nodes(a, b);
116. **if** (heap.Empty()) **return** c;
117. heap.Insert(c);//将合并后的节点插入到小根堆中
119. }
120. **return** nodes[0];
121. }
123. **void** Huffman\_code(Tree T,HASH2 &codes,string s){
124. **if**(T -> data != -1) {
125. codes.insert({T -> data, s});
126. **return**;
127. }
128. Huffman\_code(T -> left, codes, s + '0');
129. Huffman\_code(T -> right, codes, s + '1');
130. }
131. //打印哈夫曼编码
132. **void** Print\_codes(HASH2 codes) {
133. //输出字符的个数
134. cout << codes.size() << endl;
135. **for** (auto it = codes.begin(); it != codes.end(); it++) {
136. //如果是换行符，打印为\\n
137. **if** (it->first == '\n') {
138. cout << "\\n" << " " << it->second << "  ";
139. } **else** **if** (it->first == ' ') {
140. cout << "space" << " " << it->second << "  ";
141. } **else** {
142. cout << it->first << " " << it->second << "  ";
143. }
144. }
145. }
147. **void** Output\_codes(HASH2 codes,string outpath, string inpath){
148. //输出编码
149. ofstream outfile;
150. outfile.open(outpath,ios::binary);
151. **if**(!outfile.is\_open()){
152. cout<<"文件打开失败"<<endl;
153. **return**;
154. }
155. ifstream infile;
156. infile.open(inpath);
157. **char** ch;
158. **while**(infile.get(ch)){
159. outfile<<codes[ch];
160. }
161. //打印哈夫曼编码
162. Print\_codes(codes);
164. }
166. **void** Decode(Tree T, string outpath, string inpath){
167. ofstream outfile;
168. outfile.open(outpath);
169. **if**(!outfile.is\_open()){
170. cout<<"文件打开失败"<<endl;
171. **return**;
172. }
173. ifstream infile;
174. infile.open(inpath);
175. **char** temp;
176. Tree p = T;
177. **while**(infile.get(temp)){
178. **if**(temp == '0'){
179. p = p -> left;
180. }
181. **if**(temp == '1'){
182. p = p -> right;
183. }
184. **if**(p -> data != -1){
185. outfile << **char**(p -> data);
186. p = T;
187. }
188. }
189. }

Huffman\_Tree.h

1. #ifndef HUFFMAN\_HUFFMAN\_TREE\_H
2. #define HUFFMAN\_HUFFMAN\_TREE\_H
4. **typedef** **struct** node{
5. **int** data;//储存字符
6. **int** weight;//储存权重
7. node\* left;
8. node\* right;
9. }Node;
10. **typedef** node\* Tree;
11. #define HASH2 unordered\_map<char, string>
13. #endif //HUFFMAN\_HUFFMAN\_TREE\_H