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：2023.10.11

班级：2203102

学号：2022111570

姓名：王乾宇

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

一、实验目的

利用树形结构实现对文本的哈夫曼编码与译码，包括对字符的编码译码和对单词的编码译码，在此过程中熟练掌握树形结构的应用

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，分别统计英文文件文本中各字符（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．将哈夫曼编码文件译码为文本文件，并与原文件进行比较。

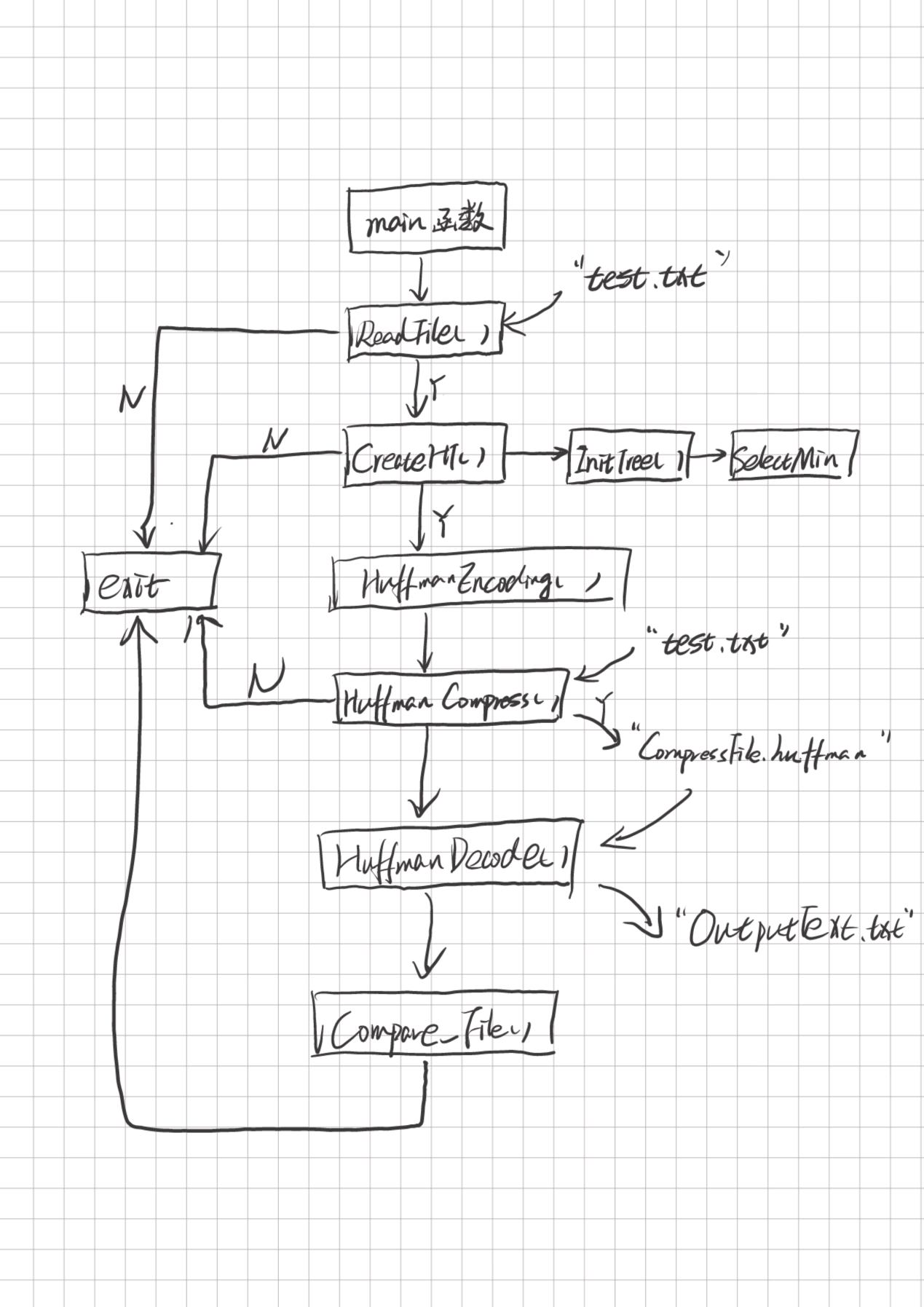
5.计算哈夫曼编码文件的平均长度和压缩率，并与实验结果比较验证。

实验环境：visual studio 2022 Window 10

三、设计思想（本程序中用到的数据类型的定义，主程序的流程图及各程序之间的调用关系、核心算法的主要步骤）

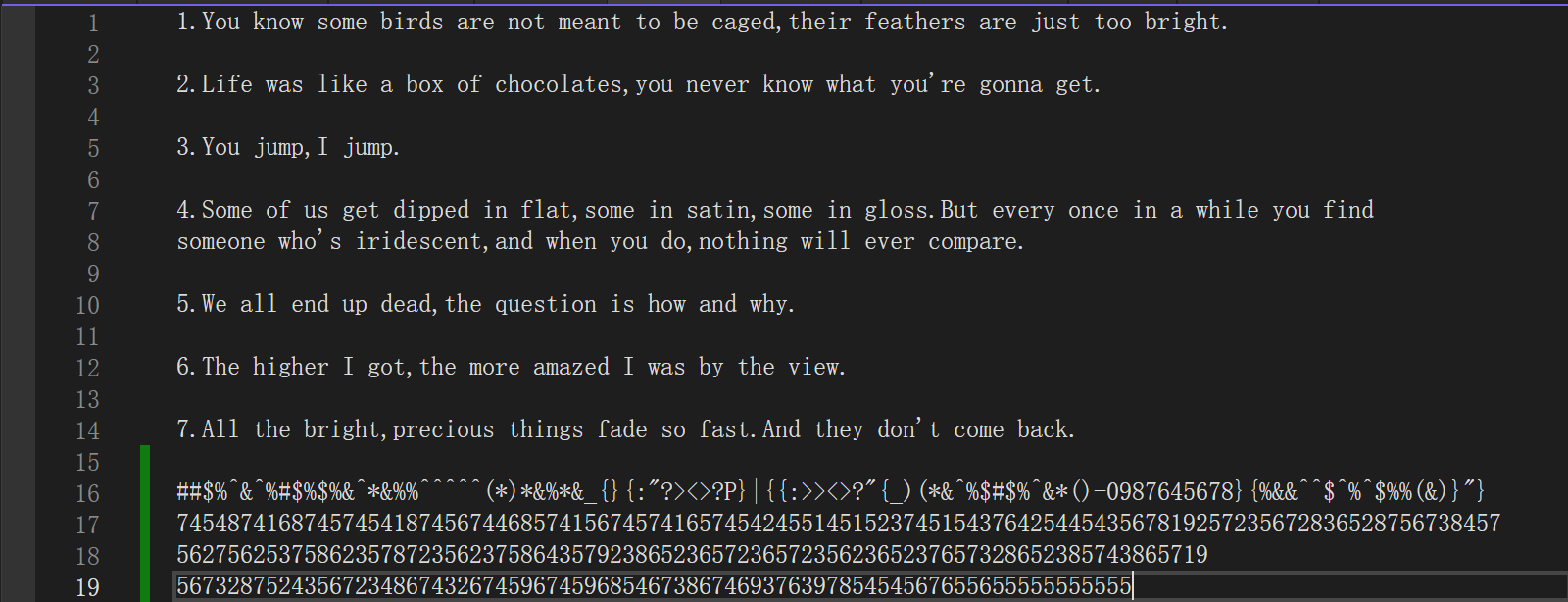
1.宏定义字符总类别数totalch、总字符数countch、haffman树的结点、字符信息的结构体数组，哈夫曼树用线性表存储。

2.主函数流程图

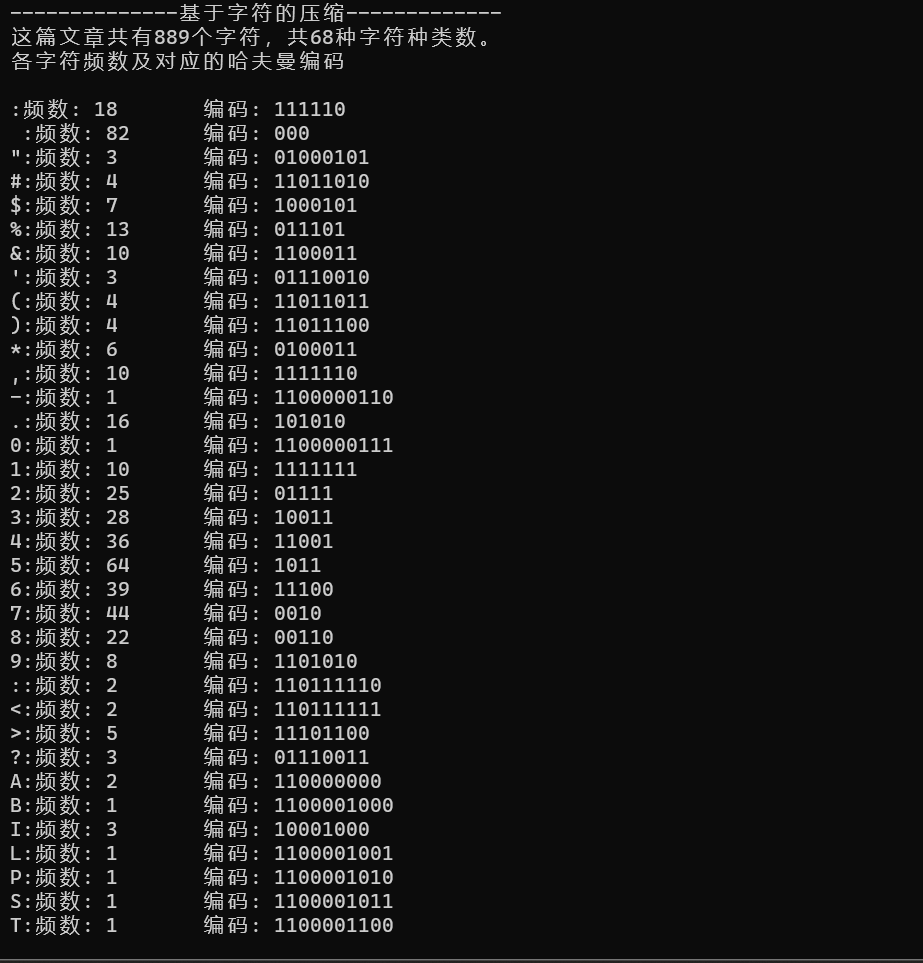


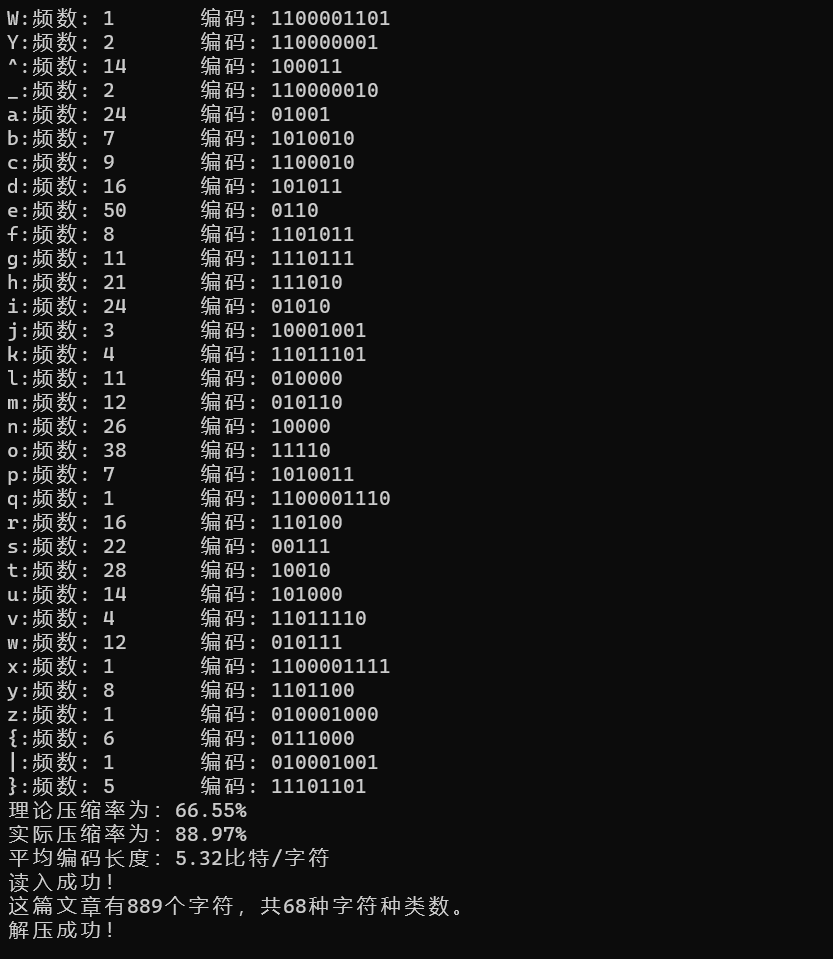
3.测试数据和结果截图

测试文本文件：

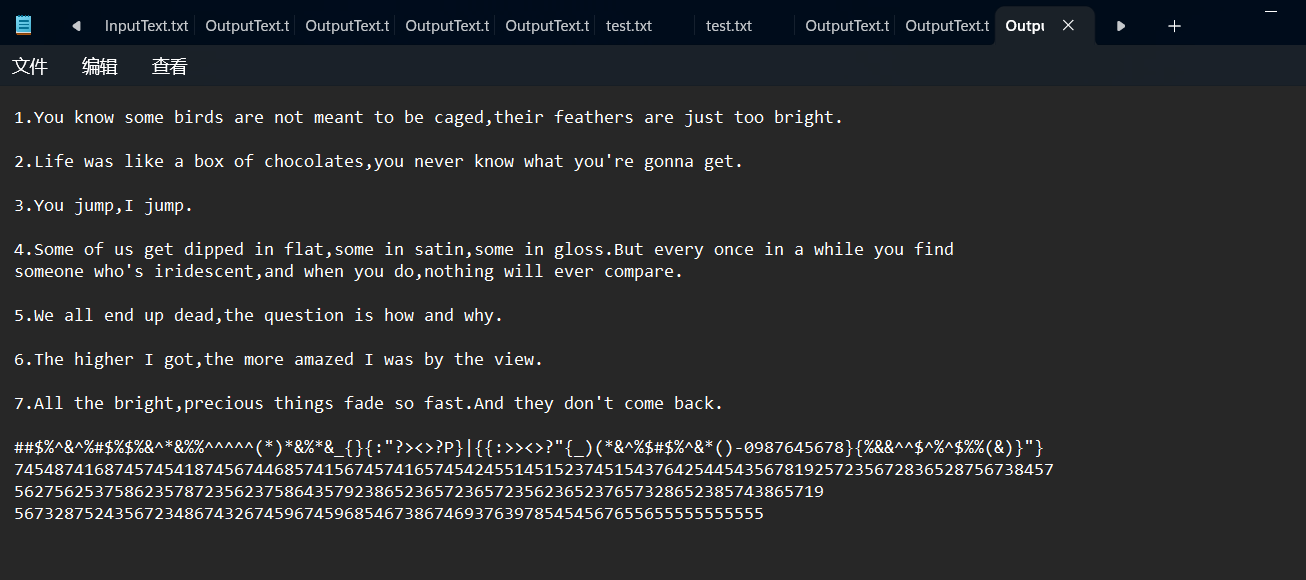


运行结果截图：





译码后文本文件：



四、经验体会与不足

通过这个实验，我深刻的理解了haffman树的存储结构和基于这种结构的haffman编码与译码。但是对于二进制文件的压缩解压缩还存在不足，对于边缘数据的测试考虑有欠缺，应该进一步去学习。

五、附录：源代码（带注释）

1. #pragma once
2. #define \_CRT\_SECURE\_NO\_WARNINGS
3. #include<stdio.h>
4. #include<stdlib.h>
5. #include<sys/stat.h>
6. #include<string.h>
7. #define MAX\_READ 10000
8. #define ASC 128
9. #define m 2\*ASC-1  *//2n-1个结点*
10. int totalch;  *//字符总类别数*
11. long long countch; *//总字符数*
12. *//haffman树的结点，静态三叉链表，每个结点的权重*
13. typedef struct {
14. double weight;
15. int lchild;
16. int rchild;
17. int parent;
18. }HTNode;
19. typedef struct {
20. char ch;  *//字符*
21. int freg; *//频数*
22. char code[m]; *//哈夫曼编码*
23. }Orchar;
24. HTNode HTree[m];
25. Orchar Or[ASC];  *//记录字符信息*
26. void ReadFile(char\* array);
27. void InitHTree();
28. void CreateHT();
29. void SelectMin(int n, int\* p1, int\* p2);
30. void HuffmanCompress();
31. void HuffmanEncoding();
32. void HuffmanDecode();
33. void CompressInformation(char\* array);
34. double CalculateAverageCodeLength();
35. int Compare\_File();
36. #define \_CRT\_SECURE\_NO\_WARNINGS
37. #include"huffman.h"
38. int main()
39. {
40. char array[MAX\_READ];
41. ReadFile(array);
42. CreateHT();
43. HuffmanEncoding();
44. printf("--------------基于字符的压缩-------------\n");
45. printf("这篇文章共有%d个字符，共%d种字符种类数。\n", countch, totalch);
46. printf("各字符频数及对应的哈夫曼编码");
47. for (int i = 0; i < totalch; i++)
48. {
49. printf("\n%c:", Or[i].ch);
50. printf("频数: %d\t编码: %s", Or[i].freg, Or[i].code);
51. }
52. HuffmanCompress();
53. printf("\n");
54. CompressInformation(array);   *//计算压缩率*
55. double averageCodeLength = CalculateAverageCodeLength();
56. printf("平均编码长度：%.2lf比特/字符\n", averageCodeLength);
57. HuffmanDecode();
58. int result = Compare\_File();
59. if (result)
60. printf("原文件与译码文件相同\n");
61. else
62. printf("原文件与译码文件不同\n");
63. printf("----------------------------------------");
64. return 0;
65. }
66. #include"huffman.h"
67. *//分字符读取文件，返回字符总数*
68. void ReadFile(char\* array)
69. {
70. FILE\* file = fopen("test.txt", "r");
71. if (file == NULL)
72. {
73. printf("打开文件失败。\n");
74. return;
75. }
76. int List[ASC] = { 0 };  *//记录频数*
77. char ch;
78. long long i = 0;  *//总数*
79. int total = 0;   *//种类数*
80. *//fgetc()逐字从文本中读取信息*
81. while ((ch = fgetc(file)) != EOF)
82. {
83. *//判断是否为字母字符、标点符号字符、空格、*
84. if (isalpha(ch) || ispunct(ch) || isspace(ch) || isalnum(ch))
85. {
86. List[ch]++;
87. i++;
88. array[i - 1] = ch;
89. }
90. }
91. fclose(file);
92. int p = 0;
93. for (int k = 0; k < ASC; k++)
94. {
95. if (List[k] > 0)
96. {
97. total++;
98. Or[p].ch = (char)k;
99. Or[p].freg = List[k];
100. p++;
101. }
102. }
103. totalch = total;
104. countch = i;
105. }
106. #include"huffman.h"
107. *//创建树*
108. void CreateHT()
109. {
110. int i, p1, p2;
111. InitHTree();
112. int n = totalch;
113. for (i = n; i < 2 \* n - 1; i++)
114. {
115. SelectMin(i, &p1, &p2);
116. HTree[p1].parent = HTree[p2].parent = i;
117. HTree[i].lchild = p1;
118. HTree[i].rchild = p2;
119. HTree[i].weight = HTree[p1].weight + HTree[p2].weight;
120. }
121. }
122. #include"huffman.h"
123. *//初始化各节点值*
124. void InitHTree()
125. {
126. for (int i = 0; i < totalch; ++i)
127. {
128. HTree[i].weight = Or[i].freg;
129. }
130. for (int i = 0; i < m; i++)
131. {
132. HTree[i].parent = -1;
133. HTree[i].lchild = -1;
134. HTree[i].rchild = -1;
135. }
136. }
137. #include"huffman.h"
138. *//输出压缩信息,计算理论压缩率和实际压缩率*
139. void CompressInformation(char\* array)
140. {
141. double Inbits = 8 \* countch;
142. double Outbits = 0;
143. for (int i = 0; i < countch; i++)
144. {
145. int t;
146. int j;
147. for (t = 0; t < totalch; t++)
148. {
149. if (Or[t].ch == array[i])  *//查码*
150. break;
151. }
152. for (j = 0; Or[t].code[j]; j++)
153. {
154. Outbits += 1;
155. }
156. }
157. printf("理论压缩率为：%.2lf%%\n", (Outbits / Inbits) \* 100);
158. struct stat In, Out;  *////取得指定文件的文件属性*
159. stat("test.txt", &In);
160. stat("CompressedFile.huffman", &Out);
161. double Inbitst = In.st\_size;  *//以字节为单位的文件容量*
162. double Outbitst = Out.st\_size;
163. printf("实际压缩率为：%.2lf%%\n", (Outbitst / Inbitst) \* 100);
164. }
165. #include"huffman.h"
166. *//输出二进制文件*
167. void HuffmanCompress()
168. {
169. FILE\* file = fopen("test.txt", "r");
170. char ch;
171. if (file == NULL)
172. {
173. printf("无法打开文件\n");
174. return;
175. }
176. FILE\* bfile = fopen("CompressedFile.huffman", "wb");
177. if (bfile == NULL)
178. {
179. printf("无法创建压缩文件\n");
180. fclose(file);
181. return;
182. }
183. fwrite(&totalch, sizeof(int), 1, bfile);  *//把字符总类别数写入*
184. for (int i = 0; i < totalch; i++)
185. {
186. fwrite(&Or[i].ch, sizeof(char), 1, bfile); *//把叶子（字符）写入*
187. }
188. for (int i = totalch; i < 2 \* totalch - 1; i++)
189. {
190. fwrite(&HTree[i].lchild, sizeof(unsigned char), 1, bfile); *//把存储的哈夫曼树写入*
191. fwrite(&HTree[i].rchild, sizeof(unsigned char), 1, bfile);
192. }
193. fwrite(&countch, sizeof(long long), 1, bfile); *//把字符总数写入*
194. int t;
195. unsigned char bin = 0;
196. int index;
197. int binindex = 0;
198. while ((ch = fgetc(file)) != EOF)
199. {
200. for (t = 0; t < totalch; t++)
201. {
202. if (Or[t].ch == ch)
203. break;
204. }
205. for (index = 0; Or[t].code[index]; ++index)*//自上而下输出每个字符的编码*
206. {
207. if (Or[t].code[index] == '0')
208. bin = bin << 1;
209. else
210. bin = (bin << 1) + 1;
211. binindex++;
212. if (binindex == 8)
213. {
214. fwrite(&bin, sizeof(bin), 1, bfile);*//把字符编码写入*
215. binindex = 0;
216. bin = 0;
217. }
218. }
219. }
220. if (binindex)
221. {
222. bin = bin << (8 - binindex);
223. fwrite(&bin, sizeof(unsigned char), 1, bfile);
224. fwrite(&binindex, sizeof(unsigned char), 1, bfile);
225. }
226. fclose(file);
227. fclose(bfile);
228. }
229. #include"huffman.h"
230. *//解码--从二进制文件中解码*
231. void HuffmanDecode()
232. {
233. FILE\* bfile = fopen("CompressedFile.huffman", "rb");
234. FILE\* file = fopen("OutputText.txt", "w");
235. if (bfile == NULL || file == NULL) {
236. printf("无法打开压缩文件或输出文件！\n");
237. return;
238. }
239. int total;
240. long long count;
241. fread(&total, sizeof(int), 1, bfile);
242. for (int i = 0; i < total; ++i) {
243. fread(&Or[i].ch, sizeof(char), 1, bfile);
244. }
245. for (int i = total; i < 2 \* total - 1; ++i) {
246. unsigned char p1, p2;
247. fread(&p1, sizeof(unsigned char), 1, bfile);
248. fread(&p2, sizeof(unsigned char), 1, bfile);
249. HTree[i].lchild = (int)p1;
250. HTree[i].rchild = (int)p2;
251. }
252. fread(&count, sizeof(long long), 1, bfile);
253. printf("读入成功！\n");
254. printf("这篇文章有%lld个字符，共%d种字符种类数。\n", count, total);
255. unsigned char b1;
256. fread(&b1, sizeof(unsigned char), 1, bfile); *//读一个字节*
257. int i = 0;
258. int j = 2 \* total - 2; *//保证只有一个节点时也能输出*
259. int ccount = 0;
260. while (ccount < count) {
261. int bit = (b1 >> (7 - i)) & 1;
262. if (bit) *//bit=1*
263. j = HTree[j].rchild;
264. else
265. j = HTree[j].lchild;
266. ++i;
267. if (j < total) {
268. fprintf(file, "%c", Or[j].ch);
269. ccount++;
270. j = 2 \* total - 2;
271. }
272. if (i == 8) {
273. i = 0;
274. fread(&b1, sizeof(unsigned char), 1, bfile);
275. }
276. }
277. fclose(bfile);
278. fclose(file);
279. printf("解压成功！\n");
280. }
281. #include"huffman.h"
282. *//编码*
283. void HuffmanEncoding()
284. {
285. int c, p, i;  *//c--孩子,p--双亲*
286. char cd[1000]; *//存放临时编码*
287. cd[totalch] = '\0';
288. int start;  *//编码在cd中的位置*
289. for (i = 0; i < totalch; i++)
290. {
291. start = totalch;*//自下而上，逆溯哈夫曼码*
292. c = i;
293. while ((p = HTree[c].parent) >= 0)
294. {
295. cd[--start] = (HTree[p].lchild == c) ? '0' : '1';
296. c = p;
297. }
298. strcpy(Or[i].code, &cd[start]);
299. }
300. }
301. #include"huffman.h"
302. *//选取两个最小数*
303. *//现选择两个无父结点的结点，再循环选取最小值*
304. void SelectMin(int n, int\* p1, int\* p2) {
305. int i, j;
306. for (i = 0; i < n; ++i) {
307. if (HTree[i].parent == -1) {
308. \*p1 = i;
309. break;
310. }
311. }
312. for (j = i + 1; j < n; ++j) {
313. if (HTree[j].parent == -1) {
314. \*p2 = j;
315. break;
316. }
317. }
318. for (i = \*p1; i < n; ++i) {
319. if ((HTree[\*p1].weight > HTree[i].weight) && (HTree[i].parent == -1) && (\*p2 != i)) {
320. \*p1 = i;
321. }
322. }
323. for (j = \*p2; j < n; ++j) {
324. if ((HTree[\*p2].weight > HTree[j].weight) && (HTree[j].parent == -1) && (\*p1 != j)) {
325. \*p2 = j;
326. }
327. }
328. }
329. #include"huffman.h"
330. double CalculateAverageCodeLength()
331. {
332. double totalbits = 0;
333. for (int i = 0; i < totalch; i++)
334. {
335. totalbits += strlen(Or[i].code) \* Or[i].freg;
336. }
337. return totalbits / countch;
338. }
339. #include"huffman.h"
340. int Compare\_File() {
341. FILE\* inputfile = fopen("test.txt", "r");
342. FILE\* outputfile = fopen("OutputText.txt", "r");
343. if (inputfile == NULL || outputfile == NULL) {
344. printf("Error opening files. Please check if the files exist and try again.\n");
345. exit(1);
346. }
347. char ch1;
348. char ch2;
349. while ((ch1 = fgetc(inputfile)) != EOF && (ch2 = fgetc(outputfile)) != EOF) {
350. if (ch1 != ch2) {
351. fclose(inputfile);
352. fclose(outputfile);
353. return 0;
354. }
355. }
356. fclose(inputfile);
357. fclose(outputfile);
358. return 1;
359. }