**数据结构实验报告—哈夫曼树**

设计者姓名：张帆

设计者班级：2班

设计者学号：20192131077

上机环境：DEV-C++

设计日期：2020-12-15

1. **实验题目**

利用哈夫曼编码进行通信可以大大提高信道利用率，缩短信息传输时间，降

低传输成本。但是，这要求在发送端通过一个编码系统对待传输数据预先编码，

在接收端将传来的数据进行译码（复原）。对于双工信道（即可以双向传输信息

的信道），每端都需要一个完整的编/译码系统。试为这样的信息收发站写一个哈

夫曼码的编/译码系统。

一个完整的系统应具有以下功能：

1） I 初始化（Initialization）。从终端读入字符集大小 n，以及 n 个字符和

n 个权值，建立哈夫曼树，并将它存于文件 hfmTree 中。

2） E 编码（Encoding）。利用已建好的哈夫曼树（如不在内存，则从文件

hfmTree 中读入）, 对文件 ToBeTran 中的正文进行编码，然后将结果

存入文件 CodeFile 中。

3） D 译码（Decoding）。 利用已建好的哈夫曼树将文件 CodeFile 中的

代码进行译码，结果存入文件 TextFile 中。

4） P 打印代码文件（Print）。将文件 CodeFile 以紧凑格式显示在终端上，

每行 50 个代码。同时将此字符形式的编码文件写入文件 CodePrint

中

5） T 打印哈夫曼树（Tree Print）。将已在内存中的哈夫曼树以直观的方

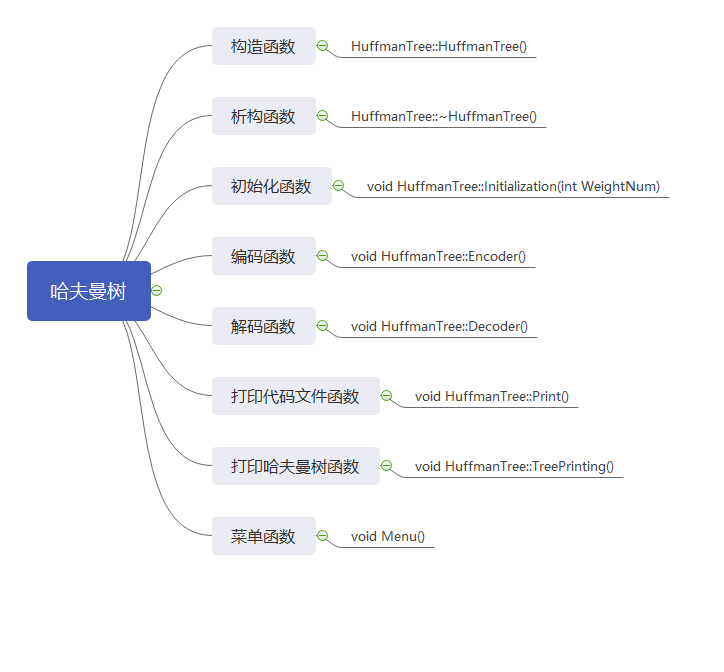
式（树或凹入表）显示在终端上，同时将此字符形式的哈夫曼树写入

文件 TreePrint 中。

1. **实验项目目的**

使学生掌握哈夫曼树结构，熟练使用哈夫曼树进行编码和解码

1. **实验项目的程序结构**



**四、实验项目包含的各个文件中的函数的功能描述**

1. #include <iostream>
2. #include <cstdio>
3. #include <cstdlib>
4. #include <cstring>
5. #include <fstream>
6. #include <algorithm>
7. **using** **namespace** std;
8. /\* run this program using the console pauser or add your own getch, system("pause") or input loop \*/
10. **struct** HuffmanNode   //定义哈夫曼节点
11. {
12. **int** weight;      //权值
13. **int** parent;      //存放父节点位置
14. **int** LeftChild;   //存放左子节点位置
15. **int** RightChild;  //存放右子节点位置
16. };
18. **class** HuffmanTree    //定义哈夫曼树类
19. {
20. **private**:
21. **char** \*Info;         //字符信息
22. **int** LeafNum;        //叶节点数目
23. HuffmanNode\* Node;  //哈夫曼节点
25. **public**:
26. HuffmanTree();        //构造函数
27. ~HuffmanTree();       //析构函数
28. **void** Initialization(**int** WeightNum);   //初始化函数：根据WeightNum个权值建立一棵哈夫曼树
29. **void** Encoder();       //编码函数：利用构造好的哈夫曼树对字符进行编码
30. **void** Decoder();       //译码函数：对二进制串进行译码
31. **void** Print();         //打印文件函数：把已保存好的编码文件显示在屏幕
32. **void** TreePrinting();  //打印哈夫曼树函数：将已在内存中的哈夫曼树以直观的方式显示在终端上
33. };
35. HuffmanTree::HuffmanTree()  //构造函数
36. {
37. Node = NULL;  //将树结点初始化为空
38. Info = NULL;  //将字符数组初始化为空
39. LeafNum = 0;  //将叶子数初始化为0
40. }
42. HuffmanTree::~HuffmanTree()  //析构函数
43. {
44. **delete**[] Info;
45. **delete**[] Node;
46. }
48. //  函数功能：从终端读入字符集大小n，以及n个字符和n个权值,
49. //            建立哈夫曼树,并将它存放在文件hfmTree中.
50. **void** HuffmanTree::Initialization(**int** WeightNum)  //初始化函数
51. {
53. **char** str[] = {' ','A','B','C','D','E','F','G','H','I','J','K','L','M','N',
54. 'O','P','Q','R','S','T','U','V','W','X','Y','Z'}; /\*字符集\*/
55. **int** fnum[] = {186,64,13,22,32,103,21,15,47,57,1,5,32,20,57,
56. 63,15,1,48,51,80,23,8,18,1,16,1}; /\*字符集对应的权值\*/
58. **int** i, j, pos1, pos2, max1, max2;
59. Info = **new** **char**[2 \* WeightNum - 1];
60. //WeightNum权值对应的哈夫曼树中的结点总数为2\*WeightNum-1个
61. Node = **new** HuffmanNode[2 \* WeightNum - 1];
62. **for** (i = 0; i < WeightNum; i++)
63. {
64. //cout << "请输入第" << i + 1 << "个字符值：";
65. //cin >> Info[i];
66. Info[i] = str[i];
67. Node[i].weight = fnum[i];
68. //cout << "请输入该字符的权值或频度：";
69. //cin >> Node[i].weight;    //输入权值
70. Node[i].parent = -1;      //为根结点
71. Node[i].LeftChild = -1;      //无左孩子
72. Node[i].RightChild = -1;      //无右孩子
73. }
74. **for**(i = WeightNum; i < 2 \* WeightNum - 1; i++) //表示需做WeightNum-1次合并
75. {
76. pos1 = -1;          //存放当前最小值的所在单元编号
77. pos2 = -1;          //存放当前次小值的所在单元编号
78. max1 = 32767;       //存放当前找到的最小值
79. max2 = 32767;       //存放当前找到的次小值
80. **for**(j = 0; j < i; j++)   //在未使用节点中选出权值最小的两个
81. {
82. **if**(Node[j].parent == -1)  //是否为根结点
83. {
84. **if**(Node[j].weight < max1)   //是否比最小值要小
85. {
86. max2 = max1;            //原最小值变为次小值
87. max1 = Node[j].weight;  //存放最小值
88. pos2 = pos1;            //修改次小值所在单元编号
89. pos1 = j;               //修改最小值所在单元编号
90. }
91. **else** **if**(Node[j].weight < max2)  //比原最小值大但比原次小值要小
92. {
93. max2 = Node[j].weight;  //存放次小值
94. pos2 = j;               //修改次小值所在的单元编号
95. }
96. }
97. }
98. Node[pos1].parent = i;      //修改父亲位置
99. Node[pos2].parent = i;
100. Node[i].LeftChild = pos1;   //修改儿子位置
101. Node[i].RightChild = pos2;
102. Node[i].parent = -1;        //表示新结点应该是根结点
103. Node[i].weight = Node[pos1].weight + Node[pos2].weight;
104. }
105. LeafNum = WeightNum;
107. **char** ch;
108. cout << "是否要替换原来文件(Y/N):";
109. cin >> ch;
110. **if**(ch == 'Y')
111. {
112. ofstream fop;   //以二进制方式打开hfmTree.dat文件，并当重新运行时覆盖原文件
113. fop.open("hfmTree.txt", ios::out|ios::binary|ios::trunc);
114. **if**(fop.fail())  //文件打开失败
115. {
116. cout << "文件打开失败！" << endl;
117. }
118. //fop << WeightNum;
119. fop.write((**char**\*)&WeightNum, **sizeof**(WeightNum));  //写入WeightNum
120. **for**(i = 0; i < WeightNum; i++)           //把各字符信息写入文件
121. {
122. fop.write((**char**\*)&Info[i], **sizeof**(Info[i]));
123. flush(cout);
124. }
125. **for**(i = 0; i < 2 \* WeightNum - 1; i++)  //把各个节点内容写入文件
126. {
127. fop.write((**char**\*)&Node[i], **sizeof**(Node[i]));
128. flush(cout);
129. }
130. fop.close();            //关闭文件
131. }
132. cout << "哈夫曼树已构造完成!" << endl;
133. }

136. //  函数功能：利用已建立好的哈夫曼树（如不在内存，则从文件hfmTree中读入），
137. //            对文件ToBeTran中的正文进行编码,然后将结果代码存(传输)到文件CodeFile中.
138. **void** HuffmanTree::Encoder()  //编码函数
139. {
140. **int** WeightNum = 27;
141. LeafNum = WeightNum;
142. **char** str[] = {' ','A','B','C','D','E','F','G','H','I','J','K','L','M','N',
143. 'O','P','Q','R','S','T','U','V','W','X','Y','Z'}; /\*字符集\*/
144. **int** fnum[] = {186,64,13,22,32,103,21,15,47,57,1,5,32,20,57,
145. 63,15,1,48,51,80,23,8,18,1,16,1}; /\*字符集对应的权值\*/
147. **int** i, j, pos1, pos2, max1, max2;
148. Info = **new** **char**[2 \* WeightNum - 1];
149. //WeightNum权值对应的哈夫曼树中的结点总数为2\*WeightNum-1个
150. Node = **new** HuffmanNode[2 \* WeightNum - 1];
151. **for** (i = 0; i < WeightNum; i++)
152. {
153. //cout << "请输入第" << i + 1 << "个字符值：";
154. //cin >> Info[i];
155. Info[i] = str[i];
156. Node[i].weight = fnum[i];
157. //cout << "请输入该字符的权值或频度：";
158. //cin >> Node[i].weight;    //输入权值
159. Node[i].parent = -1;      //为根结点
160. Node[i].LeftChild = -1;      //无左孩子
161. Node[i].RightChild = -1;      //无右孩子
162. }
163. **for**(i = WeightNum; i < 2 \* WeightNum - 1; i++) //表示需做WeightNum-1次合并
164. {
165. pos1 = -1;          //存放当前最小值的所在单元编号
166. pos2 = -1;          //存放当前次小值的所在单元编号
167. max1 = 32767;       //存放当前找到的最小值
168. max2 = 32767;       //存放当前找到的次小值
169. **for**(j = 0; j < i; j++)   //在未使用节点中选出权值最小的两个
170. {
171. **if**(Node[j].parent == -1)  //是否为根结点
172. {
173. **if**(Node[j].weight < max1)   //是否比最小值要小
174. {
175. max2 = max1;            //原最小值变为次小值
176. max1 = Node[j].weight;  //存放最小值
177. pos2 = pos1;            //修改次小值所在单元编号
178. pos1 = j;               //修改最小值所在单元编号
179. }
180. **else** **if**(Node[j].weight < max2)  //比原最小值大但比原次小值要小
181. {
182. max2 = Node[j].weight;  //存放次小值
183. pos2 = j;               //修改次小值所在的单元编号
184. }
185. }
186. }
187. Node[pos1].parent = i;      //修改父亲位置
188. Node[pos2].parent = i;
189. Node[i].LeftChild = pos1;   //修改儿子位置
190. Node[i].RightChild = pos2;
191. Node[i].parent = -1;        //表示新结点应该是根结点
192. Node[i].weight = Node[pos1].weight + Node[pos2].weight;
193. }
195. /\*
196. if(Node == NULL)       //哈夫曼树不在内存，从文件hfmTree中读入
197. {
198. ifstream fip;        //以二进制方式打开hfmTree.dat文件
199. fip.open("hfmTree.txt",ios::binary|ios::in);
200. if(fip.fail())       //文件打开失败
201. {
202. cout<<"文件打开失败！/n";
203. return;          //结束函数
204. }
205. fip.read((char\*)&LeafNum, sizeof(LeafNum));   //读取叶子数
206. Info = new char[LeafNum];
207. Node = new HuffmanNode[2 \* LeafNum - 1];
208. for(int i = 0; i < LeafNum; i++)              //读取字符信息
209. fip.read((char\*)&Info[i], sizeof(Info[i]));
210. for(int i = 0; i < 2 \* LeafNum - 1; i++)      //读取结点信息
211. fip.read((char\*)&Node[i], sizeof(Node[i]));
212. }\*/
214. **char** \*Tree;          //用于存储需编码内容
215. i = 0;
216. **int** num;
217. cout << "从文件中读取内容ing..." << endl;
218. ifstream fip1("ToBeTran.txt");
219. **if**(fip1.fail())      //文件不存在
220. {
221. cout << "文件打开失败!" << endl;
222. **return**;          //结束函数
223. }
224. **char** ch;
225. **int** k = 0;
226. **while**(fip1.get(ch))
227. {
228. k++;             //计算CodeFile中代码长度
229. }
230. **int** ttt = k;
231. fip1.close();
232. Tree = **new** **char**[k + 1];
233. ifstream fip2("ToBeTran.txt", ios::in | ios::binary);
234. fip2.getline(Tree, k + 1);
235. fip2.close();
236. Tree[k] = '/0';      //结束标志
237. cout << "读取成功！" << endl;
238. ofstream fop("CodeFile.txt",ios::trunc);   //存储编码后的代码,并覆盖原文件
239. **char** \*code;
240. code = **new** **char**[LeafNum];       //为所产生编码分配容量为LeafNum的存储空间
241. //因为不等长编码中最长的编码一定不会超过要求编码的字符个数
242. **for**(**int** k = 0; k < ttt; k++)
243. {
244. **int** j, start = 0;
245. **for**(i = 0; i < LeafNum; i++)
246. {
247. **if**(Info[i] == Tree[k])  //求出该文字所在单元的编号
248. {
249. **break**;
250. }
251. }
252. j = i;
253. **while**(Node[j].parent != -1)     //结点j非树根
254. {
255. j = Node[j].parent;         //非结点j的双亲结点
256. **if**(Node[j].LeftChild == i)  //是左子树，则生成代码0
257. {
258. code[start++] = '0';
259. }
260. **else**
261. {
262. code[start++] = '1';    //是右子树，则生成代码1
263. }
264. i = j;
265. }
266. **for**(i = 0; i < start / 2; i++)  //对二进制序列进行逆置
267. {
268. j = code[i];
269. code[i] = code[start- i - 1];
270. code[start - i - 1] = j;
271. }
272. i = 0;
273. **for** (**int** o = 0; o < start; o++)
274. {
275. fop << code[o];
276. cout << code[o];
277. }
278. }
279. cout << endl;
280. fop.close();
281. cout << "已编码！且存到文件CodeFile.txt中！" << endl << endl;
282. }
284. //  函数功能：利用已建好的哈夫曼树,对传输到达的CodeFile中的数据代码进行译码,
285. //            将译码结果存入文件TextFile中.
286. **void** HuffmanTree::Decoder()  //译码函数
287. {
288. **int** WeightNum = 27;
289. LeafNum = WeightNum;
290. **char** str[] = {' ','A','B','C','D','E','F','G','H','I','J','K','L','M','N',
291. 'O','P','Q','R','S','T','U','V','W','X','Y','Z'}; /\*字符集\*/
292. **int** fnum[] = {186,64,13,22,32,103,21,15,47,57,1,5,32,20,57,
293. 63,15,1,48,51,80,23,8,18,1,16,1}; /\*字符集对应的权值\*/
295. **int** i, j, pos1, pos2, max1, max2;
296. Info = **new** **char**[2 \* WeightNum - 1];
297. //WeightNum权值对应的哈夫曼树中的结点总数为2\*WeightNum-1个
298. Node = **new** HuffmanNode[2 \* WeightNum - 1];
299. **for** (i = 0; i < WeightNum; i++)
300. {
301. //cout << "请输入第" << i + 1 << "个字符值：";
302. //cin >> Info[i];
303. Info[i] = str[i];
304. Node[i].weight = fnum[i];
305. //cout << "请输入该字符的权值或频度：";
306. //cin >> Node[i].weight;    //输入权值
307. Node[i].parent = -1;      //为根结点
308. Node[i].LeftChild = -1;      //无左孩子
309. Node[i].RightChild = -1;      //无右孩子
310. }
311. **for**(i = WeightNum; i < 2 \* WeightNum - 1; i++) //表示需做WeightNum-1次合并
312. {
313. pos1 = -1;          //存放当前最小值的所在单元编号
314. pos2 = -1;          //存放当前次小值的所在单元编号
315. max1 = 32767;       //存放当前找到的最小值
316. max2 = 32767;       //存放当前找到的次小值
317. **for**(j = 0; j < i; j++)   //在未使用节点中选出权值最小的两个
318. {
319. **if**(Node[j].parent == -1)  //是否为根结点
320. {
321. **if**(Node[j].weight < max1)   //是否比最小值要小
322. {
323. max2 = max1;            //原最小值变为次小值
324. max1 = Node[j].weight;  //存放最小值
325. pos2 = pos1;            //修改次小值所在单元编号
326. pos1 = j;               //修改最小值所在单元编号
327. }
328. **else** **if**(Node[j].weight < max2)  //比原最小值大但比原次小值要小
329. {
330. max2 = Node[j].weight;  //存放次小值
331. pos2 = j;               //修改次小值所在的单元编号
332. }
333. }
334. }
335. Node[pos1].parent = i;      //修改父亲位置
336. Node[pos2].parent = i;
337. Node[i].LeftChild = pos1;   //修改儿子位置
338. Node[i].RightChild = pos2;
339. Node[i].parent = -1;        //表示新结点应该是根结点
340. Node[i].weight = Node[pos1].weight + Node[pos2].weight;
341. }
343. i = 0;
344. **int** k = 0;
345. j = LeafNum \* 2 - 2;      //表示从根结点开始往下搜索
346. **char**\* BitStr;
348. ifstream fip1("CodeFile.txt"); //利用已建好的哈夫曼树将文件CodeFile中的代码进行译码
349. **if**(fip1.fail())          //文件打开失败，还未编码
350. {
351. cout << "请先编码！" << endl;
352. **return**;
353. }
354. cout << "经译码,原内容为:";
355. **char** ch;
356. **while**(fip1.get(ch))
357. {
358. k++;                  //计算CodeFile中代码长度
359. }
360. fip1.close();
362. BitStr = **new** **char**[k + 1];
363. ifstream fip2("CodeFile.txt");
364. k = 0;
365. **while**(fip2.get(ch))
366. {
367. BitStr[k] = ch;       //读取文件内容
368. k++;
369. }
370. fip2.close();
371. **int** num = k;
372. //BitStr[k] = '/0';        //结束标志符
373. **if**(Node == NULL)         //还未建哈夫曼树
374. {
375. cout << "请先编码!" << endl;
376. **return**;
377. }
378. ofstream fop("TextFile.txt",ios::out|ios::binary|ios::trunc);        //将字符形式的编码文件写入文件CodePrin中
379. **for** (**int** i = 0; i < num; i++)
380. {
381. **if**(BitStr[i] == '0')
382. j = Node[j].LeftChild;       //往左走
383. **else**
384. j = Node[j].RightChild;      //往右走
385. **if**(Node[j].RightChild == -1)     //到达叶子结点
386. {
387. cout << Info[j];             //输出叶子结点对应的字符
388. j = LeafNum \* 2 - 2;         //表示重新从根结点开始往下搜索
389. fop << Info[j];              //存入文件
390. }
391. }
392. fop.close();
393. cout << endl << "译码成功且已存到文件TextFile.txt中！" << endl << endl;
394. }
396. //  函数功能：将文件CodeFile以紧凑格式显示在终端上，
397. //            每行50个代码。同时将此字符形式的编码文件写入文件CodePrin中。
398. **void** HuffmanTree::Print()  //打印代码文件函数
399. {
400. **char** ch;
401. **int** i = 1;  //计数
402. ifstream fip("CodeFile.txt");  //读取文件
403. ofstream fop("CodePrin.txt");  //存储文件
404. **if**(fip.fail())
405. {
406. cout<<"没有文件，请先编码！/n";
407. **return**;
408. }
409. **while**(fip.get(ch))
410. {
411. cout << ch;        //读取文件内容
412. fop << ch;         //存到文件中
413. **if**(i == 50)        //每行输出50个字符
414. {
415. cout << endl;
416. i = 0;
417. }
418. i++;
419. }
420. cout << endl << endl;
421. fip.close();           //关闭CodeFile.txt文件
422. fop.close();           //关闭CodePrin.txt文件
423. }
425. //  函数功能：将已在内存中的哈夫曼树以直观的方式(树或凹入表的形式)显示在终端上，
426. //            同时将此字符形式的哈夫曼树写入文件TreePrint中。
427. **void** HuffmanTree::TreePrinting()  //打印哈夫曼树函数
428. {
429. **int** WeightNum = 27;
430. **char** str[] = {' ','A','B','C','D','E','F','G','H','I','J','K','L','M','N',
431. 'O','P','Q','R','S','T','U','V','W','X','Y','Z'}; /\*字符集\*/
432. **int** fnum[] = {186,64,13,22,32,103,21,15,47,57,1,5,32,20,57,
433. 63,15,1,48,51,80,23,8,18,1,16,1}; /\*字符集对应的权值\*/
435. **int** i, j, pos1, pos2, max1, max2;
436. Info = **new** **char**[2 \* WeightNum - 1];
437. //WeightNum权值对应的哈夫曼树中的结点总数为2\*WeightNum-1个
438. Node = **new** HuffmanNode[2 \* WeightNum - 1];
439. **for** (i = 0; i < WeightNum; i++)
440. {
441. //cout << "请输入第" << i + 1 << "个字符值：";
442. //cin >> Info[i];
443. Info[i] = str[i];
444. Node[i].weight = fnum[i];
445. //cout << "请输入该字符的权值或频度：";
446. //cin >> Node[i].weight;    //输入权值
447. Node[i].parent = -1;      //为根结点
448. Node[i].LeftChild = -1;      //无左孩子
449. Node[i].RightChild = -1;      //无右孩子
450. }
451. **for**(i = WeightNum; i < 2 \* WeightNum - 1; i++) //表示需做WeightNum-1次合并
452. {
453. pos1 = -1;          //存放当前最小值的所在单元编号
454. pos2 = -1;          //存放当前次小值的所在单元编号
455. max1 = 32767;       //存放当前找到的最小值
456. max2 = 32767;       //存放当前找到的次小值
457. **for**(j = 0; j < i; j++)   //在未使用节点中选出权值最小的两个
458. {
459. **if**(Node[j].parent == -1)  //是否为根结点
460. {
461. **if**(Node[j].weight < max1)   //是否比最小值要小
462. {
463. max2 = max1;            //原最小值变为次小值
464. max1 = Node[j].weight;  //存放最小值
465. pos2 = pos1;            //修改次小值所在单元编号
466. pos1 = j;               //修改最小值所在单元编号
467. }
468. **else** **if**(Node[j].weight < max2)  //比原最小值大但比原次小值要小
469. {
470. max2 = Node[j].weight;  //存放次小值
471. pos2 = j;               //修改次小值所在的单元编号
472. }
473. }
474. }
475. Node[pos1].parent = i;      //修改父亲位置
476. Node[pos2].parent = i;
477. Node[i].LeftChild = pos1;   //修改儿子位置
478. Node[i].RightChild = pos2;
479. Node[i].parent = -1;        //表示新结点应该是根结点
480. Node[i].weight = Node[pos1].weight + Node[pos2].weight;
481. }
482. LeafNum = WeightNum;
484. **if**(Node == NULL)  //未建立哈夫曼树
485. {
486. cout << "请先建立哈夫曼树！" << endl;
487. **return**;
488. }
489. ofstream fop("TreePrint.txt",ios::trunc);
490. cout << "结点位置(权值)  " << "编码  " << "左孩子  " << "编码" << "右孩子('^'表示叶子)" << endl;
491. fop << "结点位置(权值)  " << "编码  " << "左孩子  " << "编码" << "右孩子('^'表示叶子)" << endl;
492. **for**(i = (2 \* LeafNum - 2); i > LeafNum - 1; i--)   //输出哈夫曼树
493. {
494. cout << i << "(" << Node[i].weight << ")" << "--1--"
495. << Node[i].LeftChild << "(" << Node[Node[i].LeftChild].weight << ")" << "--0--"
496. << Node[i].RightChild << "(" << Node[Node[i].RightChild].weight << ")" << endl;
497. fop << i << "(" << Node[i].weight << ")" << "--1--"
498. << Node[i].LeftChild << "(" << Node[Node[i].LeftChild].weight << ")" << "--0--"
499. << Node[i].RightChild << "(" << Node[Node[i].RightChild].weight << ")" << endl;
500. }
501. **for**(; i >= 0; i--)
502. {
503. cout << i << ":" << Node[i].weight << "(" << Info[i] << ")---^/n";
504. fop << i << ":" << Node[i].weight << "(" << Info[i] << ")---^/n";
505. }
506. cout << endl << endl;
507. }
509. **void** Menu()  //菜单函数
510. {
511. **while**(1)
512. {
513. **int** choice;
514. HuffmanTree HT;
515. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;
516. cout << "\* 哈夫曼码的编/译码系统 \*" << endl;
517. cout << "\*       1、初始化       \*" << endl;
518. cout << "\*        2、编码        \*" << endl;
519. cout << "\*        3、译码        \*" << endl;
520. cout << "\*    4、打印代码文件    \*" << endl;
521. cout << "\*    5、打印哈夫曼树    \*" << endl;
522. cout << "\*        6、退出        \*" << endl;
523. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;
524. cout << "请选择要操作的选项：";
525. cin >> choice;
526. **if** (choice == 1)
527. {
528. **int** n;
529. cout << "请输入字符集大小：";
530. cin >> n;
531. HT.Initialization(n);
532. }
533. **else** **if** (choice == 2)
534. {
535. HT.Encoder();
536. }
537. **else** **if** (choice == 3)
538. {
539. HT.Decoder();
540. }
541. **else** **if** (choice == 4)
542. {
543. HT.Print();
544. }
545. **else** **if** (choice == 5)
546. {
547. HT.TreePrinting();
548. }
549. **else** **break**;
550. }
551. }
553. **int** main(**int** argc, **char**\*\* argv)
554. {
555. Menu();  //菜单函数
556. **return** 0;
557. }

**五、算法描述或流程图**

假设有n个权值，则构造出的哈夫曼树有n个叶子结点。 n个权值分别设为 w1、w2、…、wn，则哈夫曼树的构造规则为：

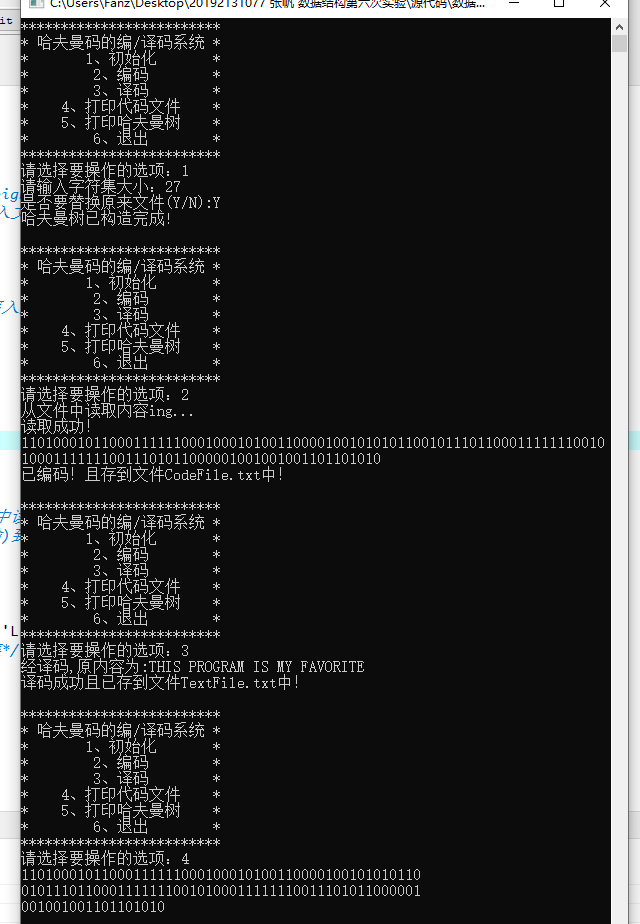
(1) 将w1、w2、…，wn看成是有n 棵树的森林(每棵树仅有一个结点)；

(2) 在森林中选出两个根结点的权值最小的树合并，作为一棵新树的左、右子树，且新树的根结点权值为其左、右子树根结点权值之和；

(3)从森林中删除选取的两棵树，并将新树加入森林；

(4)重复(2)、(3)步，直到森林中只剩一棵树为止，该树即为所求得的哈夫曼树。

**六、实验数据和实验结果分析**





1. **实验体会**

这次实验，让我更加理解了哈夫曼树的存储操作，对哈夫曼树的编码和解码有了一个更深入的了解。