**数据结构实验报告**

|  |  |  |  |
| --- | --- | --- | --- |
| 学号-姓名 | 桑龙龙-20030540015 | 实验时间 | 2020年 11 月14 日 |
| 诚信声明 | 本实验及实验报告所写内容为本人所作 | | |
| 实验题目 | 二叉树的运算与应用  题目一、二叉树的遍历运算  题目二、哈夫曼编/译码器 | | |
| 实验过程中遇到的主要问题 | 无 | | |
| 实验小结 | 本次实验进行了二叉树的构建、前中后序遍历，以及huffman树的应用，对树的认识更加深刻。 | | |
| 数据结构  （自定义数据类型） | 1. **struct** huffman\_node { 2. /\* huffman树的节点 \*/ 3. **char** v; 4. /\* 关键字v，只有叶子节点的v才有意义 \*/ 5. **float** f; 6. /\* 关键字v在文本中出现的频率 \*/ 7. huffman\_node\* next[2]; 8. /\* 左右儿子 \*/ 9. huffman\_node( **int** \_v, **float** \_f ) 10. { 11. next[0] = NULL; 12. next[1] = NULL; 13. f   = \_f; 14. v   = \_v; 15. } 16. huffman\_node( huffman\_node\* a, huffman\_node\* b ) 17. { 18. f   = a->f + b->f; 19. next[0] = a; 20. next[1] = b; 21. } 22. }; | | |
| 主要算法  （或算法说明） | 1. /\* 二叉树的遍历运算 \*/ 2. #include <stdio.h> 3. #include <stdlib.h> 4. #include <iostream> 5. #include <algorithm> 6. **using** **namespace** std; 7. **struct** node { 8. node    \* left; 9. node    \* right; 10. **char**    val; 11. node( **int** \_v ) : left( NULL ), right( NULL ), val( \_v ){} 12. }; 14. **void** create( node\* &root ) 15. { 16. /\* 创建二叉树 \*/ 17. **char** v = getchar(); 18. **if** ( v == '.' ) 19. root = NULL; 20. **else**{ 21. root = **new** node( v ); 22. create( root->left ); 23. create( root->right ); 24. } 25. }  28. **void** preorder( node\* root ) 29. { 30. /\* 前序遍历 \*/ 31. **if** ( !root ) 32. **return**; 33. putchar( root->val ); 34. preorder( root->left ); 35. preorder( root->right ); 36. }  39. **void** inorder( node\* root ) 40. { 41. /\* 中序遍历 \*/ 42. **if** ( !root ) 43. **return**; 44. inorder( root->left ); 45. putchar( root->val ); 46. inorder( root->right ); 47. }  50. **void** postorder( node\* root ) 51. { 52. /\* 后序遍历 \*/ 53. **if** ( !root ) 54. **return**; 55. postorder( root->left ); 56. postorder( root->right ); 57. putchar( root->val ); 58. }  61. **int** main() 62. { 63. node\* root = NULL; 64. create( root ); 65. printf( "pre order: " ); 66. preorder( root ); 67. putchar( '\n' ); 69. printf( "in order: " ); 70. inorder( root ); 71. putchar( '\n' ); 73. printf( "post order: " ); 75. postorder( root ); 76. putchar( '\n' ); 77. **return**(0); 78. } 79. /\* 哈夫曼编/译码器 \*/ 80. #include <stdio.h> 81. #include <stdlib.h> 82. #include <iostream> 83. #include <algorithm> 84. #include <queue> 85. #include <vector> 86. #include <string.h> 87. /\* https://paste.ubuntu.com/p/BY2Rv3Rwv3/ \*/ 88. **using** **namespace** std; 89. /\* 90. \* 输入： 91. \* 第一行一个n，表示关键字个数 92. \* 后序n行，每行一个字符c和一个频率 93. \* 之后一行文本，需要我们来解析 94. \* 输出： 95. \* n行一个字符，以及其对应的huffman码 96. \* 之后会输出文本的经过Huffman编码后的文本 97. \* 之后一行会输出经过decode后的原文本 98. \*/ 99. /\* 100. \* 样例1： 101. \* 26 102. \* a 0.830000 103. \* b 0.860000 104. \* c 0.770000 105. \* d 0.150000 106. \* e 0.930000 107. \* f 0.350000 108. \* g 0.860000 109. \* h 0.920000 110. \* i 0.490000 111. \* j 0.210000 112. \* k 0.620000 113. \* l 0.270000 114. \* m 0.900000 115. \* n 0.590000 116. \* o 0.630000 117. \* p 0.260000 118. \* q 0.400000 119. \* r 0.260000 120. \* s 0.720000 121. \* t 0.360000 122. \* u 0.110000 123. \* v 0.680000 124. \* w 0.670000 125. \* x 0.290000 126. \* y 0.820000 127. \* z 0.300000 128. \* abcdefghigkmnopqrstuvwxyzaaaaaaa 129. \*/ 130. **using** p = pair<**char**, **float**>; 131. p keyword[128]; 132. /\* 用来记录输入关键字及其频率 \*/ 133. **char** code[128][9], path[9], after\_encode[10000]; 134. /\* 135. \* code用来存储ascii码对应的的huffman码 136. \* after\_encode用来存储经过编码后的文本的huffman码 137. \*/ 138. **int** n, pos; 139. **struct** huffman\_node { 140. /\* huffman树的节点 \*/ 141. **char** v; 142. /\* 关键字v，只有叶子节点的v才有意义 \*/ 143. **float** f; 144. /\* 关键字v在文本中出现的频率 \*/ 145. huffman\_node\* next[2]; 146. /\* 左右儿子 \*/ 147. huffman\_node( **int** \_v, **float** \_f ) 148. { 149. next[0] = NULL; 150. next[1] = NULL; 151. f   = \_f; 152. v   = \_v; 153. }  156. huffman\_node( huffman\_node\* a, huffman\_node\* b ) 157. { 158. f   = a->f + b->f; 159. next[0] = a; 160. next[1] = b; 161. } 162. } 163. ; 164. **void** getcode( huffman\_node\* root, **int** step ) 165. { 166. /\* 167. \* 解析每一个关键字的huffman码 168. \* 将其存储起来，方便encode 169. \*/ 170. **if** ( !root->next[0] && !root->next[1] ) 171. { 172. path[step] = 0; 173. memcpy( code[root->v], path, step ); 174. **return**; 175. } 176. path[step] = '0'; 177. getcode( root->next[0], step + 1 ); 178. path[step] = '1'; 179. getcode( root->next[1], step + 1 ); 180. }  183. huffman\_node\* build\_huffman() 184. { 185. /\* 构建huffman树 \*/ 186. huffman\_node    \* a; 187. huffman\_node    \* b; 188. huffman\_node    \* c; 189. **float**       f; 190. **char**        v; 191. auto        cmp = [] (**const** huffman\_node \* a, **const** huffman\_node \* b) { 192. **return**(a->f > b->f); 193. } 194. ; 195. priority\_queue<huffman\_node\*, vector<huffman\_node\*>, decltype( cmp )> Q( cmp ); 196. /\* 利用堆来加快huffman树的构造 \*/ 197. **for** ( **int** i = 0; i < n; i++ ) 198. { 199. scanf( "%c %f\n", &keyword[i].first, &keyword[i].second ); 200. a = **new** huffman\_node( keyword[i].first, keyword[i].second ); 201. Q.push( a ); 202. } 203. **while** ( Q.size() > 1 ) 204. { 205. a   = Q.top(), Q.pop(); 206. b   = Q.top(), Q.pop(); 207. Q.push( **new** huffman\_node( a, b ) ); 208. } 209. getcode( Q.top(), 0 ); 210. **return**(Q.top() ); 211. }  214. **void** encode( huffman\_node\* root ) 215. { 216. /\* encode输入的文本文件 \*/ 217. **char** v; 218. **while** ( (v = getchar() ) != EOF ) 219. { 220. strcat( after\_encode, code[v] ); 221. } 222. }  225. **void** decode( huffman\_node\* root ) 226. { 227. /\* 对huffman码进行解析，恢复为原来的句子 \*/ 228. **if** ( !root->next[0] && !root->next[1] ) 229. { 230. putchar( root->v ); 231. **return**; 232. } 233. decode( root->next[after\_encode[pos++] - '0'] ); 234. }  237. **int** main() 238. { 239. scanf( "%d\n", &n ); 240. **if** ( n <= 1 ) 241. { 242. /\* 如果关键字只有一个就没有必要编码 \*/ 243. printf( "this is no need to encoding\n" ); 244. **return**(0); 245. } 246. huffman\_node\*root = build\_huffman(); 247. **for** ( **int** i = 0; i < n; i++ ) 248. { 249. /\* 输出每个关键字对应的huffman码 \*/ 250. printf( "%c 's huffman code is: %s\n", keyword[i].first, code[keyword[i].first] ); 251. } 252. encode( root ); 253. /\* 编码 \*/ 254. printf( "after\_encode %s\n", after\_encode ); 255. printf( "now will decode:\n" ); 256. **while** ( after\_encode[pos] != 0 ) 257. decode( root ); 258. /\* 解码 \*/ 259. **return**(0); 260. } | | |