哈尔滨工业大学计算机科学与技术学院

实验报告

课程名称：数据结构与算法课程

类型：必修

实验项目：树型结构的建立、遍历和应用

实验题目： 二叉树存储结构的建立、遍历和应用

实验日期： 2018.11.23

班级：1703002

学号：1170500913

姓名：熊健羽

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

1. **实验目的**
   1. 熟悉二叉树的逻辑结构、存储结构
   2. 掌握二叉树的前序、中序、后序遍历的递归和非递归的过程和算法
   3. 理解完全二叉树、树的宽度等概念，以遍历算法为基础设计相应的算法。
2. **实验要求及实验环境**
   1. 实验要求
      1. 至少采用两种方法，编写建立二叉树的二叉链表存储结构（左右链表示）的程序，并以适当的形式显示和保存二叉树；
      2. 采用二叉树的二叉链表存储结构，编写程序实现二叉树的先序、中序和后序遍历的递归和非递归算法以及层序遍历算法，并以适当的形式显示和保存二叉树及其相应的遍历序列；
      3. 设计并实现判断任意一棵二叉树是否为完全二叉树的算法；
      4. 设计并实现计算任意一棵二叉树的宽度的（递归或非递归）算法。二叉树的宽度是指其各层结点数的最大值。
   2. 实验环境
      1. 硬件环境

CPU：Intel(R) Core(TM) i5-7200U @ 2.50GHz (64位)

GPU：Intel(R) HD Graphics 620

Nvidia GeForce 940MX

物理内存：8.00GB

磁盘：1TB HDD

128GB SSD

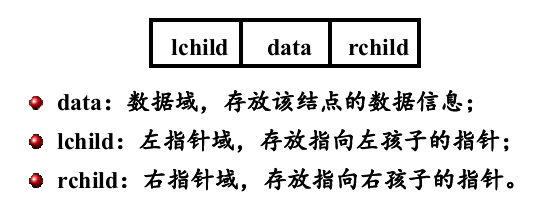
* + 1. 软件环境：

OS: Microsoft Windows 10 家庭中文版

编译环境：MinGW-W64

IDE：VS code、code::blocks

1. **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）
   1. 逻辑设计
      1. 数据结构:



* + 1. 各个函数的功能及其调用关系：
       1. 递归建树

函数原型：BTREE buildRecur(void);

调用函数：buildRecur() (自身)

* + - 1. 非递归建树

函数原型：BTREE buildNoRecur(void);

调用函数：scanf,getchar

* + - 1. 先序遍历（递归）

函数原型：void preRecur(BTREE root);

调用函数：preRecur，printf

* + - 1. 先序遍历（非递归）

函数原型：void preNoRecur(BTREE root);

调用函数：printf

* + - 1. 中序遍历（递归）

函数原型：void inRecur(BTREE root);

调用函数：inRecur，printf

* + - 1. 中序遍历（非递归）

函数原型：void inNoRecur(BTREE root)

调用函数：printf

* + - 1. 后序遍历（递归）

函数原型：void postRecur(BTREE root);

调用函数：postRecur，printf

* + - 1. 后序遍历（非递归）

函数原型：void postNoRecur(BTREE root);

调用函数：printf

* + - 1. 层序遍历

函数原型：void level(BTREE root);

调用函数：printf

* + - 1. 对二叉树root，递归地构造一个布尔数组

函数原型：void signNode(BTREE root, bool num[], int i)；

调用函数：signNode

* + - 1. 判断是否为完全二叉树

函数原型：bool isComplete(BTREE root);

调用函数：signNode

* + - 1. 构造记录每层节点数的数组

函数原型：void setFloor(NODE root, int i, int numOfFloor[]);

调用函数：setFloor(调用自身)

* + - 1. 求树宽度

函数原型：int wide(BTREE root);

调用函数：setFloor，memset

* + - 1. 构造顺序存储二叉树的数组，为输出做准备

函数原型：void buildArray(BTREE root, datatype num[], int i);

调用函数：buildArray(调用自身)

* + - 1. 层序输出二叉树的形态

函数原型：void print(BTREE root);

调用函数：memset, buildArray

* 1. 物理设计
     1. 存储结构：

1. **struct** node
2. {
3. **struct** node \*lchild;
4. **struct** node \*rchild;
5. datatype data;
6. };
   * 1. 函数实现思路：
        1. 递归建树：

较简单，略。

* + - 1. 非递归建树
      2. 先序遍历（非递归）：

1.栈s初始化；

2.循环直到root为空且栈s为空

2.1 当root不空时循环

2.1.1 输出root->data;

2.1.2 将指针root的值保存到栈中；

2.1.3 继续遍历root的左子树；

2.2 如果栈s不空，则

2.2.1 将栈顶元素弹出至root；

2.2.2 遍历root的右子树；

* + - 1. 中序遍历（非递归）

1.栈s初始化；

2.循环直到root为空且栈s为空

2.1 当root不空时循环

2.1.1 将指针root的值保存到栈中；

2.1.2 继续遍历root的左子树；

2.2 如果栈s不空，则

2.2.1 将栈顶元素弹出至root；

2.2.2 输出root->data;

2.2.3 遍历root的右子树；

* + - 1. 后序遍历（非递归）

1. 栈s初始化；

2. 循环直到root为空且栈s为空

2.1 当root非空时循环

2.1.1 将root连同标志flag=1 入栈；

2.1.2 继续遍历root的左子树；

2.2 当栈s 非空且栈顶元素标志为2 时，出栈并输出栈顶结点;

2.3 若栈非空，将栈顶元素标志改为2，遍历栈顶结点的右子树。root

* + - 1. 层序遍历

1. 队列Q初始化；

2. 如果二叉树非空，将根指针入队；

3. 循环直到队列Q为空

3.1 q=队列Q的队头元素出队；

3.2 访问结点q的数据域；

3.3 若结点q存在左孩子，则将左孩子指针入队；

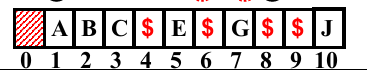
3.4 若结点q存在右孩子，则将右孩子指针入队；

* + - 1. 对二叉树root，递归地构造一个布尔数组：

参数为(BTREE root, bool num[], int i)，若root不为空，则把数组下标为i的元素置为true，于是，根据孩子与双亲节点的下标关系可写出下列递归调用：

1. **if** (root != NULL && root->data != '$')
2. {
3. num[i] = 1;
4. signNode(root->lchild, num, i \* 2);
5. signNode(root->rchild, num, i \* 2 + 1);
6. }
   * + 1. 判断是否为完全二叉树：

调用上述函数构造布尔数组。如果该二叉树为完全二叉树（总节点数为n个），则前面n个元素全都是true，否则就不是完全二叉树（如图）。



1. **int** wide(BTREE root) //求宽度
2. {
3. **int** numOfFloor[MAX];
4. **int** i, max;
5. memset(numOfFloor, 0, **sizeof**(numOfFloor));
6. setFloor(root, 1, numOfFloor);
7. i = 1;
8. max = 0;
9. **while** (numOfFloor[i] != 0)
10. {
11. max = (numOfFloor[i] > max) ? numOfFloor[i] : max;
12. i++;
13. }            //打擂算法找出最大的宽度
14. **return** max;
15. }
    * + 1. 构造记录每层节点数的数组

参数为(NODE root, int i, int numOfFloor[])，若root非空，给下标为i的元素加1.容易写出如下的递归调用：

1. **void** setFloor(NODE root, **int** i, **int** numOfFloor[])  //递归遍历每个节点，构造记录每层节点数的数组
2. {
3. **if** (root != NULL)
4. {
5. numOfFloor[i]++;
6. setFloor(root->lchild, i + 1, numOfFloor);
7. setFloor(root->rchild, i + 1, numOfFloor);
8. }
9. }
   * + 1. 求树宽度

调用上述函数，构造记录每层节点数的数组，找出其最大值即可。

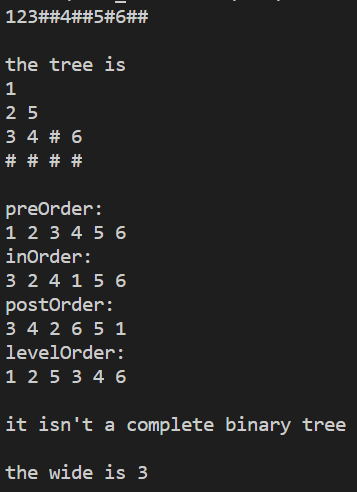
1. **int** wide(BTREE root) //求宽度
2. {
3. **int** numOfFloor[MAX];
4. **int** i, max;
5. memset(numOfFloor, 0, **sizeof**(numOfFloor));
6. setFloor(root, 1, numOfFloor);
7. i = 1;
8. max = 0;
9. **while** (numOfFloor[i] != 0)
10. {
11. max = (numOfFloor[i] > max) ? numOfFloor[i] : max;
12. i++;
13. }            //打擂算法找出最大的宽度
14. **return** max;
15. }
    * + 1. 构造顺序存储二叉树的数组，为输出做准备，空节点用’#’填充

根据孩子与双亲节点的下标关系可写出下列递归调用：

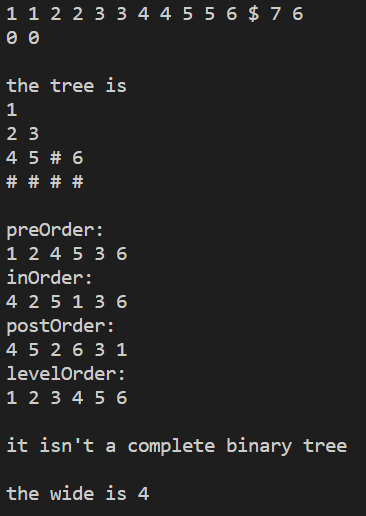
1. **int** c = 0;  //记录应该输出字符的个数
2. **void** buildArray(BTREE root, datatype num[], **int** i)  //构造顺序存储二叉树的数组，为输出做准备
3. {
4. **if** (root != NULL && root->data != '$')
5. {
6. num[i] = root->data;
7. c++;
8. buildArray(root->lchild, num, i \* 2);
9. buildArray(root->rchild, num, i \* 2 + 1);
10. }
11. **else**
12. {
13. num[i] = '#';  //空节点用#填充
14. c++;
15. }
16. }
    * + 1. 层序输出二叉树的形态

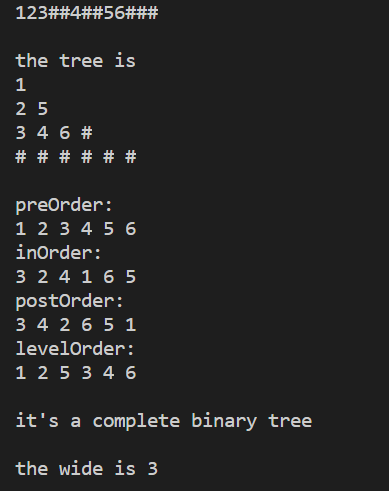
调用上述函数，初始化输出数组。

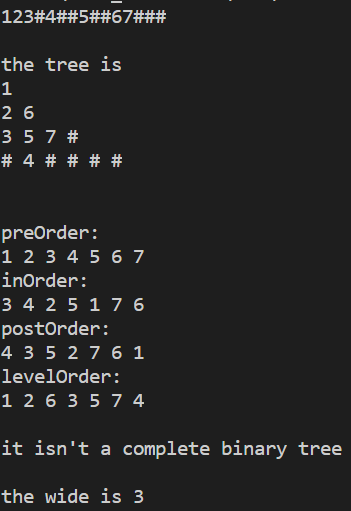
1. **void** print(BTREE root) //层序输出
2. {
3. datatype num[MAX];
4. memset(num, 0, **sizeof**(num));
5. buildArray(root, num, 1);
6. **int** i;
7. **for** (i = 1; i < c + 1; i++)
8. {
9. printf("%c ", num[i]);
10. **if**((i + 1 & i )== 0)
11. printf("\n");//当i + 1为2的整数次方时，换行
12. }
13. }
14. **测试结果**
    1. 递归建树：



* 1. 非递归建树：







1. **经验体会与不足**

经验体会：

* 1. 熟悉了二叉树的建立、先序、中序、后序的递归、非递归算法，体会到栈、队列等数据结构在非递归算法中的作用；
  2. 通过编写宽度、完全二叉树鉴定的函数，深入理解了“二叉树遍历算法是有关二叉树算法中最核心的算法”这句话；
  3. 认识到二叉树的链式、顺序存储结构的优缺点，熟练使用两者，能简化算法。

不足：

* 1. 多次用到顺序存储结构，未进行复用，某种程度上增加了算法的无意义操作；
  2. 由于时间关系，二叉树的输出还不够直观，只是简单的输出了每一层节点。

1. **附录：源代码（带注释）**
2. #include <stdio.h>
3. #include <stdlib.h>
4. #include <stdbool.h>
5. #include <string.h>
6. #include <math.h>
7. #define MAX 1000
9. **typedef** **char** datatype;
11. **struct** node
12. {
13. **struct** node \*lchild;
14. **struct** node \*rchild;
15. datatype data;
16. };
18. **typedef** **struct** node \*NODE;
19. **typedef** **struct** node \*BTREE;

22. **int** counter = 0;       //计数器，记录树的节点个数
23. BTREE buildRecur(**void**) //以先序顺序递归建树
24. {
25. BTREE t;
26. datatype ch;
27. scanf("%c", &ch);
28. **if** (ch == '#')   //若输入'#'证明遇到空指针
29. {
30. t = NULL;
31. }
32. **else**             //否则建立一个新的节点
33. {
34. t = (BTREE)malloc(**sizeof**(**struct** node));  //开辟新节点
35. t->data = ch;                         //填入数据
36. counter++;                           //计数器加一
37. t->lchild = buildRecur();
38. t->rchild = buildRecur();            //建立当前的左节点和右节点
39. }
40. **return** t;                               //返回建立的节点
41. }
43. BTREE buildNoRecur(**void**)  //非递归建树
44. {
45. BTREE t;         //储存根节点
46. NODE s[MAX], p;
47. **int** i, j;
48. datatype ch;
49. scanf("%d %c", &i, &ch);
50. getchar();
51. **while** (i != 0 && ch != 0)
52. {
53. p = (BTREE)malloc(**sizeof**(**struct** node));
54. p->data = ch;
55. counter++;
56. s[i] = p;
57. p->lchild = NULL;
58. p->rchild = NULL;
59. **if** (i == 1)   //如果是父节点，
60. t = p;
61. **else**
62. {
63. j = i / 2;   // 不是父节点，则父节点为i / 2
64. **if** (i % 2 == 0)
65. s[j]->lchild = p;   //是左孩子
66. **else**
67. s[j]->rchild = p;   //是右孩子
68. }
69. scanf("%d %c", &i, &ch);
70. getchar();
71. }
72. **return** t;
73. }
75. **void** preRecur(BTREE root)    //先序遍历（递归）
76. {
77. **if** (root != NULL)
78. {
79. **if** (root->data != '$')
80. printf("%c ", root->data);
81. preRecur(root->lchild);
82. preRecur(root->rchild);
83. }
84. }
85. **void** preNoRecur(BTREE root)  //先序遍历（非递归）
86. {
87. NODE stack[MAX];
88. **int** top = -1;
89. **while** (top != -1 || root != NULL)
90. {
91. **while** (root != NULL)
92. {
93. **if** (root->data != '$')
94. printf("%c ", root->data);
95. **if** (root->rchild != NULL)
96. {
97. stack[++top] = root;
98. }
99. root = root->lchild;
100. }
101. **if** (top != -1)
102. {
103. root = stack[top--];
104. root = root->rchild;
105. }
106. }
107. }
109. **void** inRecur(BTREE root)       //中序遍历（递归）
110. {
111. **if** (root != NULL)
112. {
113. inRecur(root->lchild);
114. **if** (root->data != '$')
115. printf("%c ", root->data);
116. inRecur(root->rchild);
117. }
118. }
120. **void** inNoRecur(BTREE root)    //中序遍历（非递归）
121. {
122. NODE stack[MAX];
123. **int** top = -1;
124. **while** (root != NULL || top != -1)
125. {
126. **while** (root != NULL)
127. {
128. // if (root->rchild != NULL)
129. stack[++top] = root;
130. root = root->lchild;
131. }
133. **if** (top != -1)
134. {
135. root = stack[top--];
136. **if** (root->data != '$')
137. printf("%c ", root->data);
138. root = root->rchild;
139. }
140. }
141. }
143. **void** postRecur(BTREE root)    //后序遍历（递归）
144. {
145. **if** (root != NULL)
146. {
147. postRecur(root->lchild);
148. postRecur(root->rchild);
149. **if** (root->data != '$')
150. printf("%c ", root->data);
151. }
152. }
154. **void** postNoRecur(BTREE root)  //后序遍历（非递归）
155. {
156. **typedef** **struct**
157. {
158. **int** flag;
159. NODE node;
160. } STACK;
162. STACK stack[MAX];
163. NODE temp;
164. **int** top = -1;
165. **while** (root != NULL || top != -1)   //循环直到root为空且栈s为空
167. {
168. **while** (root != NULL)    //当root非空时循环
169. {
170. stack[++top].node = root;
171. stack[top].flag = 1;    //将root连同标志flag=1入栈
172. root = root->lchild;   //继续遍历root的左子树
173. }
174. **while** (top != -1 && stack[top].flag == 2)     //当栈顶标志为2循环
175. {
176. temp = stack[top--].node;
177. **if** (temp->data != '$')
178. printf("%c ", temp->data);     //出栈并输出
179. }
180. **if** (top != -1)        //若栈不为空
181. {
182. root = stack[top].node;  //读取栈顶
183. stack[top].flag = 2;    //栈顶标志置2
184. root = root->rchild;    //遍历栈的右孩子
185. }
186. }
187. }
188. **void** level(BTREE root)    //层序遍历
189. {
190. NODE Q[MAX];
191. **int** front = 0, rear = 0;  //初始化队列
192. **if** (root == NULL)        //如果树空，直接返回
193. **return**;
194. Q[rear++] = root;       //根节点入队
195. **while** (rear != front)   //循环直到队列为空
196. {
197. root = Q[front++];           //出队
198. **if** (root->data != '$')
199. printf("%c ", root->data);  //访问
200. **if** (root->lchild != NULL)
201. Q[rear++] = root->lchild;   //左孩子入队
202. **if** (root->rchild != NULL)
203. Q[rear++] = root->rchild;   //右孩子入队
204. }
205. }

208. **void** signNode(BTREE root, **bool** num[], **int** i)  //对二叉树root，递归地构造一个布尔数组
209. {
210. **if** (root != NULL && root->data != '$')
211. {
212. num[i] = 1;
213. signNode(root->lchild, num, i \* 2);
214. signNode(root->rchild, num, i \* 2 + 1);
215. }
216. }
218. **bool** isComplete(BTREE root)   //判断是否为完全二叉树
219. {
220. **bool** num[MAX];
221. **int** i;
222. memset(num, 0, **sizeof**(num));  //数组初始化为false
223. signNode(root, num, 1);
224. i = 1;
225. **while** (num[i] == 1)
226. i++;
227. **if** (i - 1 == counter)
228. **return** **true**;    //如果数组的第1 ~ counter项都为true，则为完全二叉树
229. **else**
230. **return** **false**;
231. }
233. **void** setFloor(NODE root, **int** i, **int** numOfFloor[])  //递归遍历每个节点，构造记录每层节点数的数组
234. {
235. **if** (root != NULL)
236. {
237. numOfFloor[i]++;
238. setFloor(root->lchild, i + 1, numOfFloor);
239. setFloor(root->rchild, i + 1, numOfFloor);
240. }
241. }

244. **int** wide(BTREE root)
245. {
246. **int** numOfFloor[MAX];
247. **int** i, max;
248. memset(numOfFloor, 0, **sizeof**(numOfFloor));
249. setFloor(root, 1, numOfFloor);
250. i = 1;
251. max = 0;
252. **while** (numOfFloor[i] != 0)
253. {
254. max = (numOfFloor[i] > max) ? numOfFloor[i] : max;
255. i++;
256. }            //打擂算法找出最大的宽度
257. **return** max;
258. }


262. **int** c = 0;  //记录应该输出字符的个数
263. **void** buildArray(BTREE root, datatype num[], **int** i)  //构造顺序存储二叉树的矩阵，为输出做准备
264. {
265. **if** (root != NULL && root->data != '$')
266. {
267. num[i] = root->data;
268. c++;
269. buildArray(root->lchild, num, i \* 2);
270. buildArray(root->rchild, num, i \* 2 + 1);
271. }
272. **else**
273. {
274. num[i] = '#';            //空节点用#填充
275. c++;
276. }
277. }
279. **void** print(BTREE root)     //层序输出
280. {
281. datatype num[MAX];
282. memset(num, 0, **sizeof**(num));
283. buildArray(root, num, 1);
284. **int** i;
285. **for** (i = 1; i < c + 1; i++)
286. {
287. printf("%c ", num[i]);
288. **if**((i + 1 & i )== 0)
289. printf("\n");     //当i + 1为2的整数次方时，换行
290. }
291. }
293. **int** main()
294. {
295. BTREE t;
297. /\*build\*/
298. // t = buildNoRecur();
299. t = buildRecur();
301. printf("\nthe tree is \n");
303. print(t);
305. printf("\n\n");
306. /\*traverse\*/
308. printf("preOrder:\n");
309. preRecur(t);
310. // preNoRecur(t);
311. printf("\n");
313. printf("inOrder:\n");
314. inRecur(t);
315. // inNoRecur(t);
316. printf("\n");
318. printf("postOrder:\n");
319. postRecur(t);
320. // postNoRecur(t);
321. printf("\n");
323. printf("levelOrder:\n");
324. level(t);
325. printf("\n\n");
327. /\*is it complete?\*/
328. **if** (isComplete(t))
329. printf("it\'s a complete binary tree \n");
330. **else**
331. printf("it isn\'t a complete binary tree \n");
332. printf("\n");
334. /\*figue out the wide\*/
335. printf("the wide is %d\n\n", wide(t));
336. getchar();getchar();
337. **return** 0;
338. }