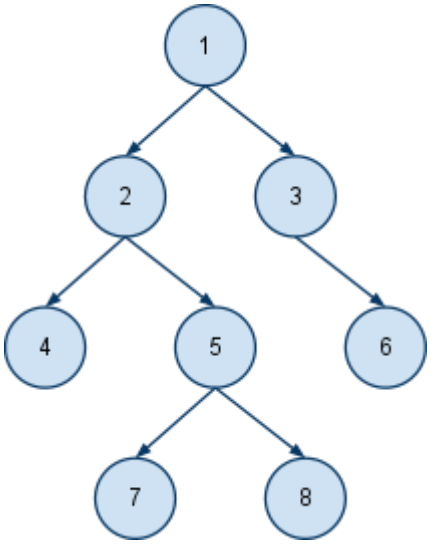
之前重温本书写书评时，也尝试找寻更好的编程解法。今天把另一个问题的实现和大家分享。

问题定义

给定一棵二叉树，要求按分层遍历该二叉树，即从上到下按层次访问该二叉树(每一层将单独输出一行)，每一层要求访问的顺序为从左到右，并将节点依次编号。下面是一个例子：



输出：

1

2 3

4 5 6

7 8

节点的定义：

```
1       struct  Node {
2               Node *pLeft;
3               Node *pRight;
4               int  data;
5       };
```

书上的解法

书上举出两个解法。第一个解法是用递归方式，搜寻并打印某一层的节点，再打印下一层的节点。这方法简单但时间效率不高(但不需要额外空间)，因此书中亦提供了第二个解法。

书中第二个解法，使用 *vector* 容器来储存 *n* 个节点信息，并用一个游标变量 *last* 记录前一层的访问结束条件，实现如下：

```
1   void  PrintNodeByLevel(Node* root) {
```

```
2           vector<Node*> vec; // 这里我们使用 STL 中的 vector 来代替数组，可利用到其
3           vec.push_back(root);
4           int  cur = 0;
5           int  last = 1;
6           while(cur < vec.size()) {
7                   Last = vec.size(); // 新的一行访问开始，重新定位 last 于当前行最
8 置
9                   while(cur < last) {
10                          cout << vec[cur] -> data << " "; // 访问节点
11                          if(vec[cur] -> lChild) // 当前访问节点的左节点不为空
12                                  vec.push_back(vec[cur] -> lChild);
13                          if(vec[cur] -> rChild) // 当前访问节点的右节点不为空
14 的访问顺序不能颠倒
15                                  vec.push_back(vec[cur] -> rChild);
16                          cur++;
17                  }
18                  cout << endl; // 当 cur == last 时,说明该层访问结束，输出换行符
            }
    }
```

广度优先搜索

书中没有提及，本问题其实是以[广度优先搜索](*breath-first search*, *BFS*)去遍历一个树结构。

广度优先搜索的典型实现是使用队列*(queue)*。其伪代码如下*:*

```
1   enqueue(Q, root)
2   do
3           node = dequeue(Q)
4           process(node) //如把内容列印
5           for  each child of node
6                   enqueue(Q, child)
7   while  Q is not empty
```

书上的解法，事实上也使用了一个队列。但本人认为，使用 *vector* 容器，较不直觉，而且其空间复杂度是 *O(n)*。

如果用队列去实现 *BFS*，不处理换行，能简单翻译伪代码为 *C++*代码*:*

```
1   void  PrintBFS(Node* root) {
2           queue<Node*> Q;
3           Q.push(root);
4           do  {
5                   Node *node = Q.front();
6                   Q.pop();
7                   cout << node->data << " ";
```

```
8                   if  (node->pLeft)
9                       Q.push(node->pLeft);
10                  if  (node->pRight)
11                      Q.push(node->pRight);
12          }
13          while  (!Q.empty());
14  }
```

本人觉得这样的算法实现可能比较清楚，而且空间复杂度只需 *O(m)*，*m* 为树中最多节点的层的节点数量。最坏的情况是当二叉树为完整，*m = n/2*。

之后的难点在于如何换行。

本人的尝试之一

第一个尝试，利用了两个队列，一个储存本层的节点，另一个储存下层的节点。遍历本层的节点，把其子代节点排入下层队列。本层遍历完毕后，就可换行，并交换两个队列。

```
1   void  PrintNodeByLevel(Node* root) {
2           deque<Node*> Q1, Q2;
3           Q1.push_back(root);
4           do  {
5                   do  {
6                           Node* node = Q1.front();
7                           Q1.pop_front();
8                           cout << node->data << " ";
9                           if  (node->pLeft)
10                              Q2.push_back(node->pLeft);
11                          if  (node->pRight)
12                              Q2.push_back(node->pRight);
13                  } while  (!Q1.empty());
14                  cout << endl;
15                  Q1.swap(Q2);
16          } while(!Q1.empty());
17  }
```

本实现使用 *deque* 而不是 *queue*，因为 *deque* 才支持 *swap()*操作。注意，*swap()*是 *O(1)* 的操作，实际上只是交换指针。

这实现要用两个循环*(*书上的实现也是*)*，并且用了两个队列。能够只用一个循环、一个队列么*?*

本人的尝试之二

换行问题其实在于如何表达一层的结束。书上采用了游标，而第一个尝试则用了两个队列。本人想到第三个可行方案，是把一个结束信号放进队列里。由于使用 *queue<Node*>*，可以插入一个空指针去表示一层的遍历结束。

```
1   void  PrintNodeByLevel(Node* root) {
2           queue<Node*> Q;
3           Q.push(root);
4           Q.push(0);
5           do  {
6                   Node* node = Q.front();
7                   Q.pop();
8                   if  (node) {
9                           cout << node->data << " ";
10                          if  (node->pLeft)
11                                  Q.push(node->pLeft);
12                          if  (node->pRight)
13                                  Q.push(node->pRight);
14                  }
15                  else  if  (!Q.empty()) {
16                          Q.push(0);
17                          cout << endl;
18                  }
19          } while  (!Q.empty());
20  }
```

这个实现的代码很贴近之前的 *PrintBFS()*，也只有一个循环。注意一点，当发现空指针*(*结束信号*)*时，要检查队列内是否还有节点，如果没有的话还插入新的结束信号，则会做成死循环。

测试代码

```
1   void  Link(Node* nodes, int  parent, int  left, int  right) {
2           if  (left != -1)
3                   nodes[parent].pLeft = &nodes[left];
4
5           if  (right != -1)
6                   nodes[parent].pRight = &nodes[right];
7   }
8
9   void  main()
10  {
11          Node test1[9] = { 0 };
12
13          for  (int  i = 1; i < 9; i++)
```

```
14                  test1[i].data = i;
15
16          Link(test1, 1, 2, 3);
17          Link(test1, 2, 4, 5);
18          Link(test1, 3, 6, -1);
19          Link(test1, 5, 7, 8);
20
21          PrintBFS(&test1[1]);
22          cout << endl << endl;
23
24          PrintNodeByLevel(&test1[1]);
25          cout << endl;
26  }
```

结语

第一个尝试是几个月前做的，没想到今晚写博文又想到了第二个尝试。两个尝试难分优劣，但两种思维或许也可以解决其他问题。还有其他方法么？