

二叉树

层次遍历：算法及分析

05-H1

邓俊辉

deng@tsinghua.edu.cn

算法实现

❖ template <typename T> template <typename VST>

```
void BinNode<T>::travLevel( VST & visit ) { //二叉树层次遍历
```

```
    Queue< BinNodePosi<T> > Q; //引入辅助队列
```

```
    Q.enqueue( this ); //根节点入队
```

```
    while ( ! Q.empty() ) { //在队列再次变空之前，反复迭代
```

```
        BinNodePosi<T> x = Q.dequeue(); //取出队首节点，并随即
```

```
        visit( x->data ); //访问之
```

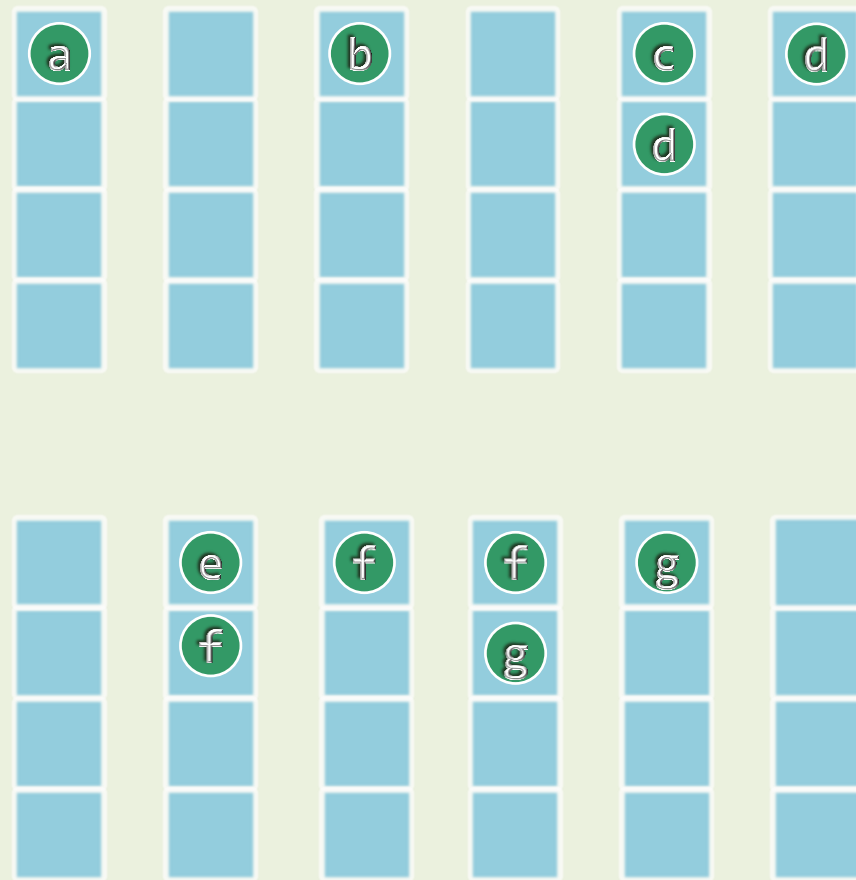
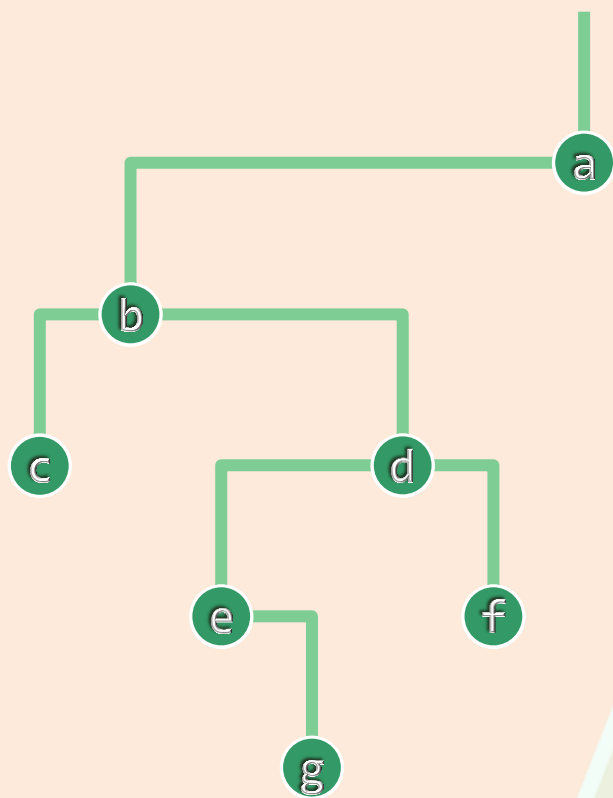
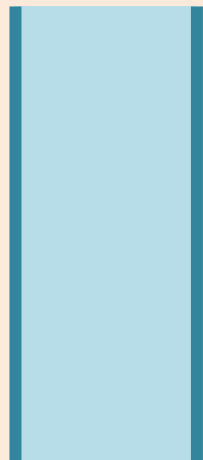
```
        if ( HasLChild( * x ) ) Q.enqueue( x->lc ); //左孩子入队
```

```
        if ( HasRChild( * x ) ) Q.enqueue( x->rc ); //右孩子入队
```

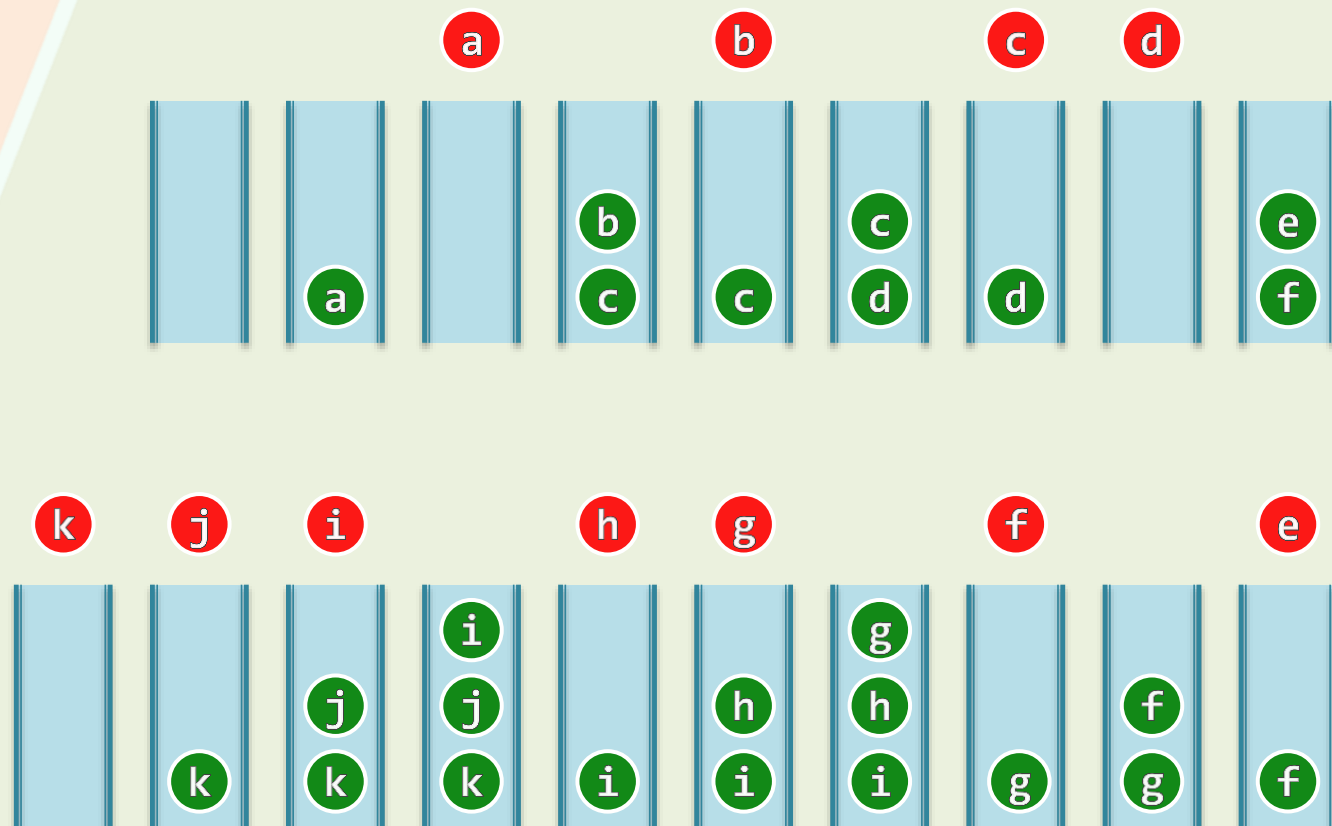
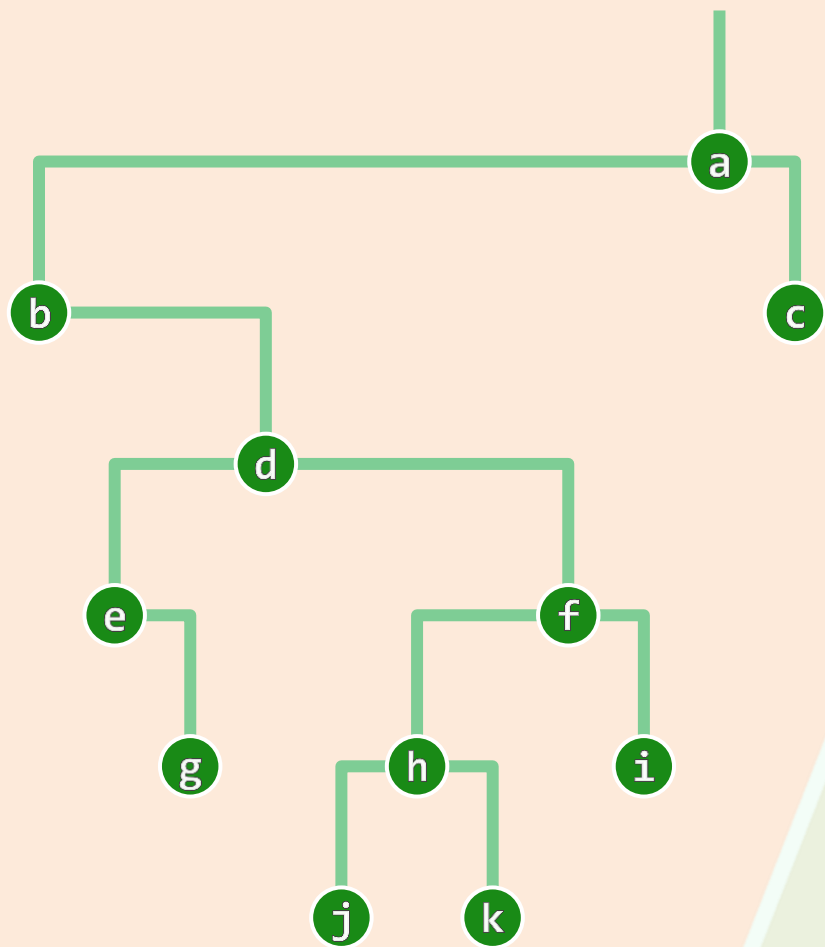
```
    }
```

```
}
```

实例



实例



分析

- ❖ 正确性何以见得？以上迭代算法符合广度优先遍历的规则...
- ❖ 每次迭代，入队节点（若存在）都是出队节点的孩子，深度增加一层
- ❖ 任何时刻，队列中各节点按深度**单调**排列，而且（相邻）节点的**深度**相差不超过**1**层
- ❖ 进一步地，所有节点迟早都会入队，而且
更高/低的节点，更早/晚入队；更左/右的节点，更早/晚入队
- ❖ 每次迭代，都有**恰好一个节点出队**并接受访问，同时有**不超过两个节点入队**
每个节点入、出队恰好各一次，故知整体只需 $O(n)$ 时间