5.二叉树

（e4）层次遍历

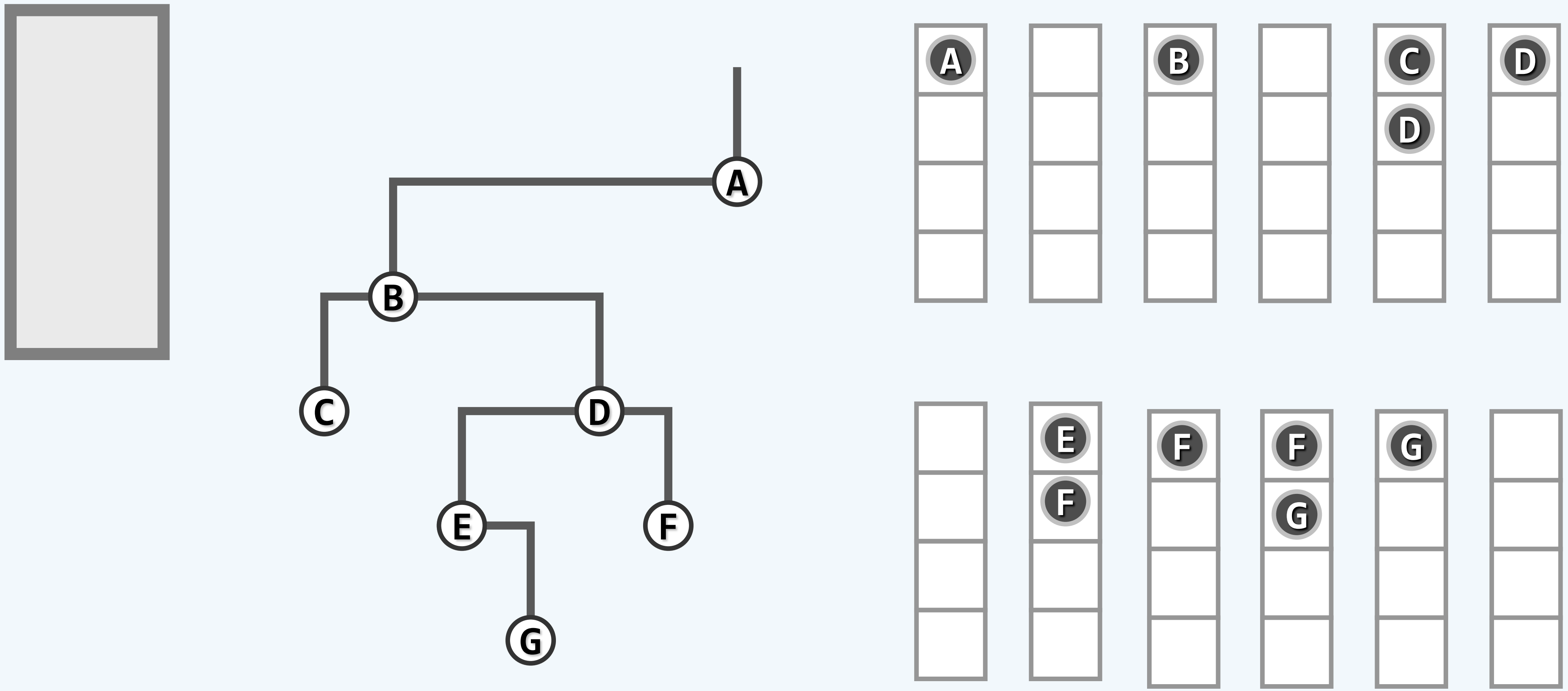
邓俊辉

deng@tsinghua.edu.cn

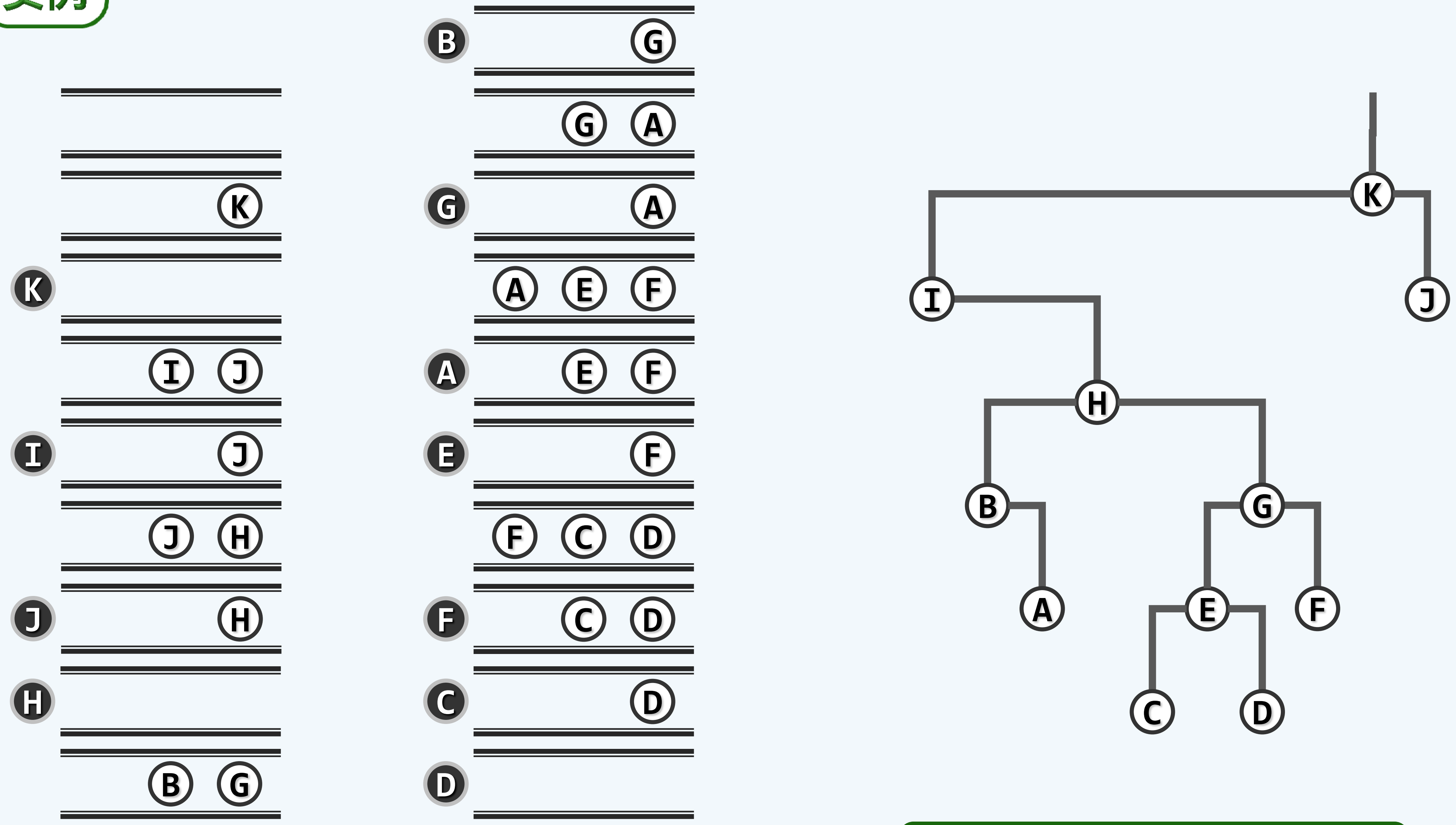
实现

```
❖ 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->lChild ); //左孩子入队
        if ( HasRChild(*x) ) Q.enqueue( x->rChild ); //右孩子入队
     }
  }
```

实例



实例



K
K
I J
I J
J H
J H
H
B G
B G
G A
G A
A E F
A E F
E F
E F
F C D
F C D
C D
C D
D
K
I J
H
B G
A E F
C D

分析

❖ 正确性何以见得？是的，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

❖ 效率如何？

❖ 每次迭代，都有一个节点出队并接受访问，但可能有两个节点入队

更精确地，每个节点入、出队各恰好一次——整体效率 = $\mathcal{O}(n)$

应用：表达式树

❖ 表达式树（expression tree）：由前缀表达式创建表达式树

a + b * (c - d) - e / f

❖ postorder（postfix = RPN）

a b c d - * + e f / -

❖ preorder（prefix，求值并不便捷）

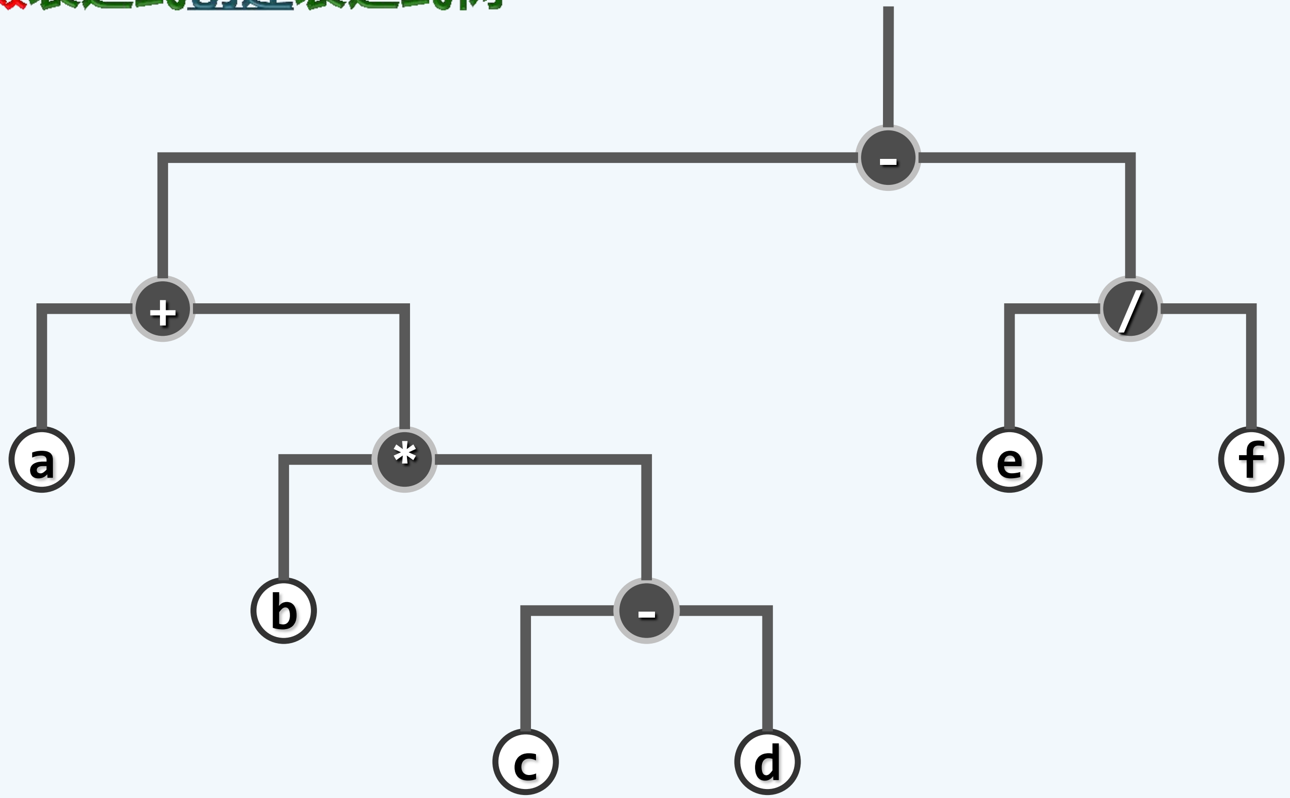
- + a * b - c d / e f

❖ inorder（infix，无优先级，有歧义）

a + b * c - d - e / f

❖ breadth-first（?）

- + / a * e f b - c d



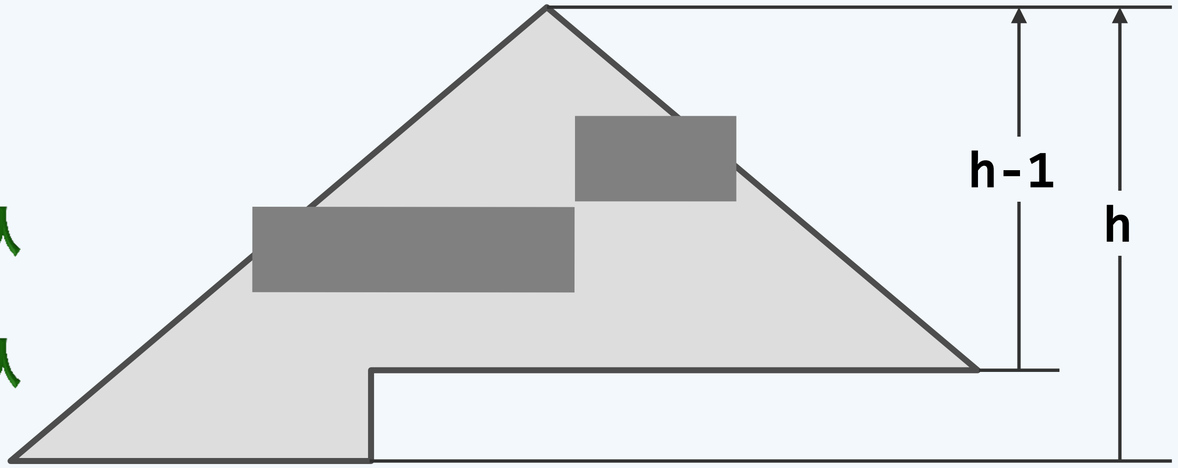
完全二叉树

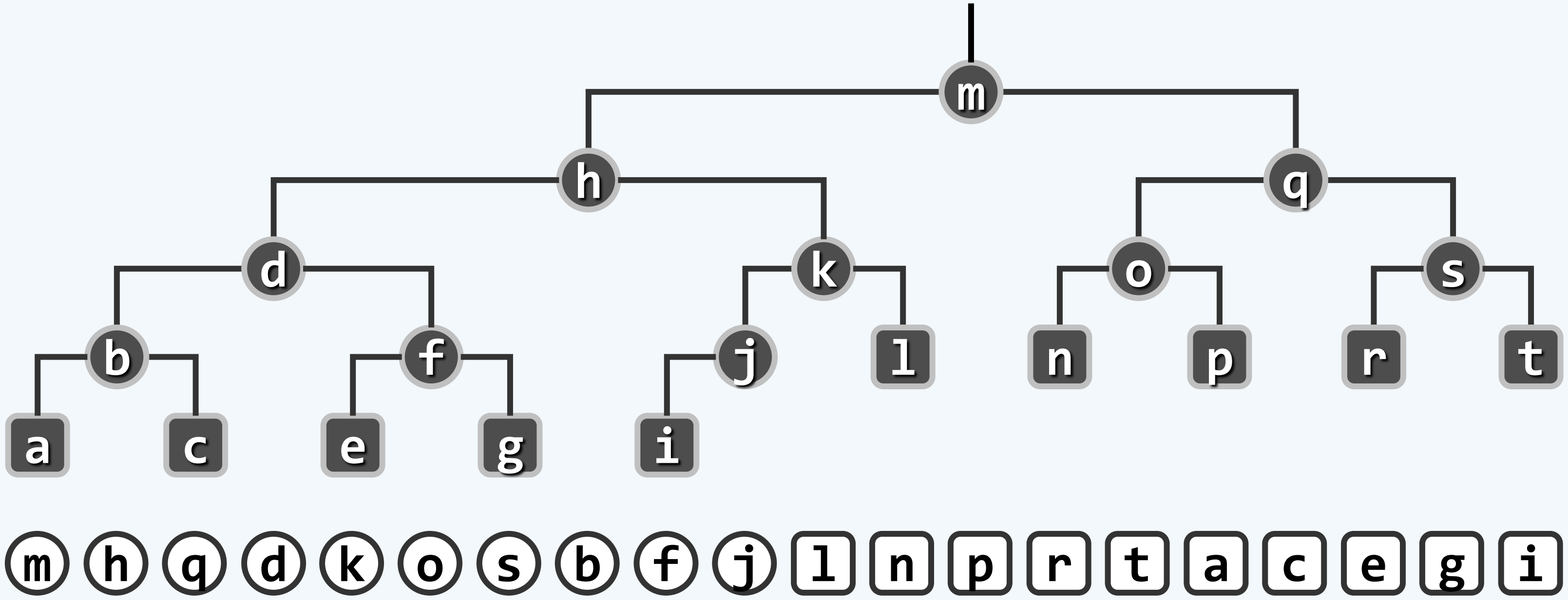
❖ 考察层次遍历中的n次迭代...

❖ 前 $\boxed{\lceil n/2 \rceil - 1}$ 次迭代中，都有**右**孩子入队

前 $\boxed{\lfloor n/2 \rfloor}$ 次迭代中，都有**左**孩子入队

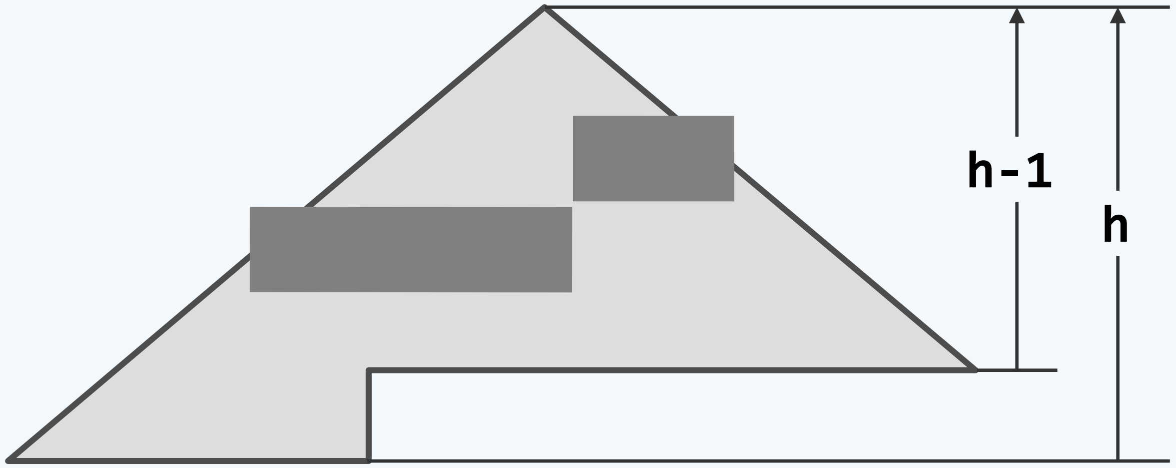
累计 $\boxed{n - 1}$ 次入队

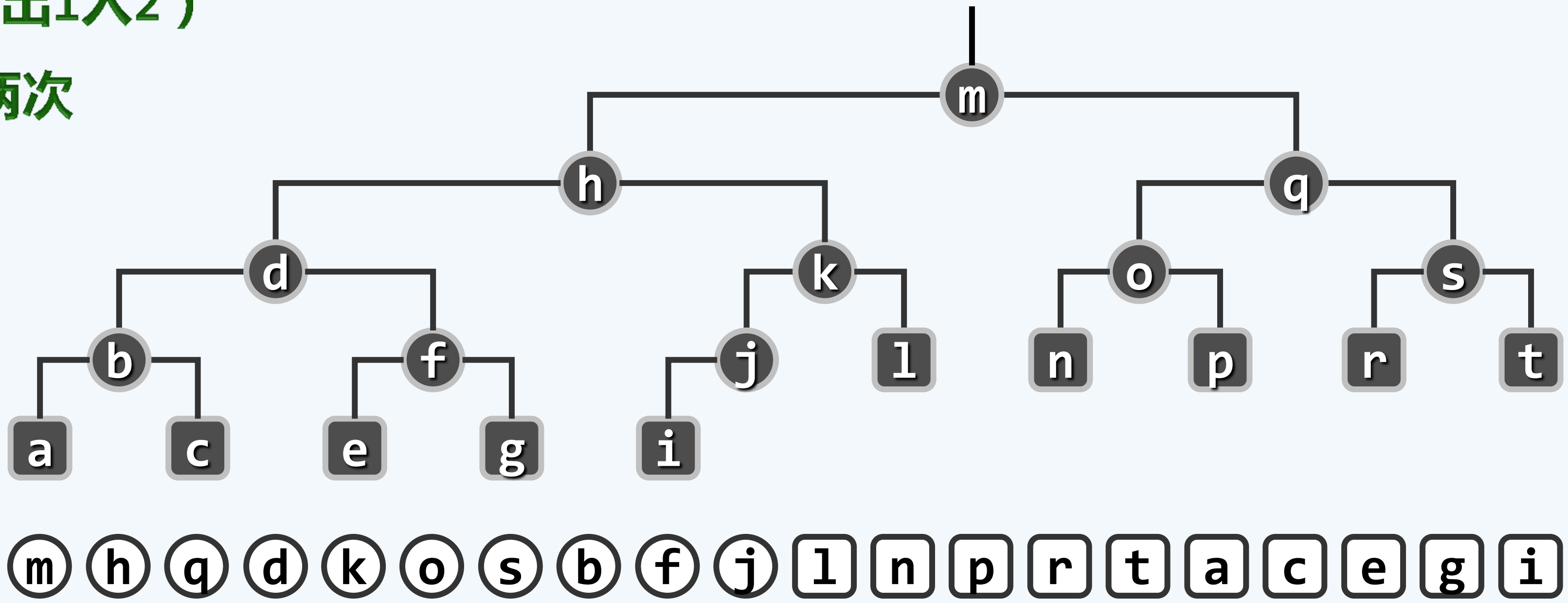




完全二叉树

- 考察层次遍历中的n次迭代...
- 辅助队列的规模
  1） 先增后减，单峰对称
  2） 最大规模 = $\lceil n/2 \rceil$
     （前 $\lceil n/2 \rceil - 1$ 次均出1入2）
  3） 最大规模可能出现两次





完全二叉树

❖ 叶节点仅限于最低两层

底层叶子，均居于次底层叶子左侧

除末节点的父亲，内部节点均有双子

❖ 叶节点

不致少于内部节点，但

至多多出一个