5.二叉树

先序遍历

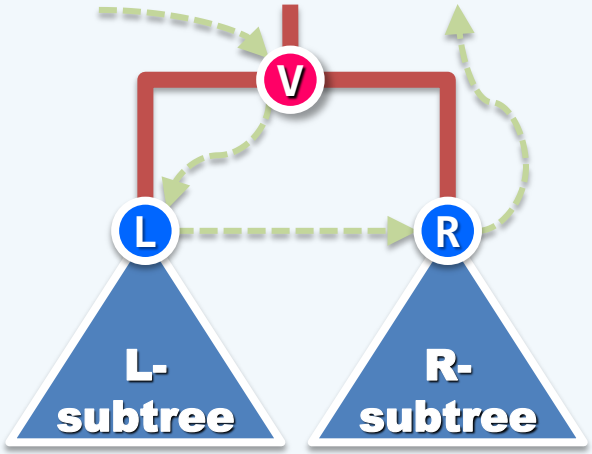
算法A

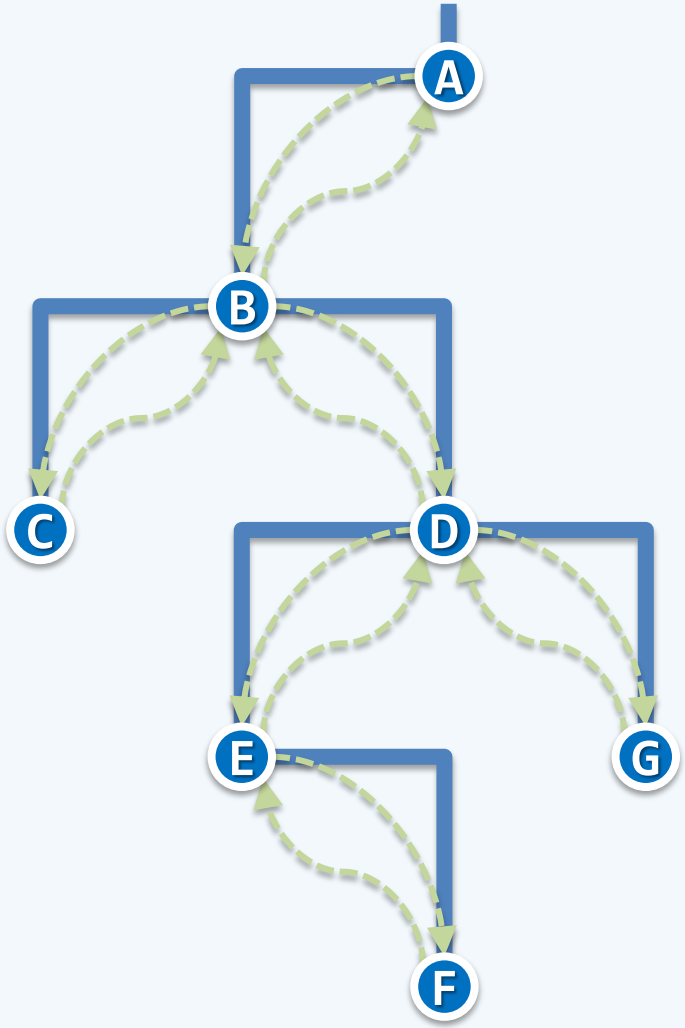
邓俊辉

deng@tsinghua.edu.cn

递归

```
template <typename T, typename VST>
void traverse( BinNodePosi(T) x, VST & visit ) {
    if ( ! x ) return;
    visit( x->data );
    traverse( x->lc, visit );
    traverse( x->rc, visit );
} //T(n) = O(1) + T(a) + T(n - a - 1) = O(n)
```





- 先序输出文件树结构：`c:\> tree.com c:\windows`
- 挑战： 不依赖递归机制，能否实现先序遍历？如何实现？效率如何？

思路

❖ 先序遍历任一二叉树T的过程，无非是

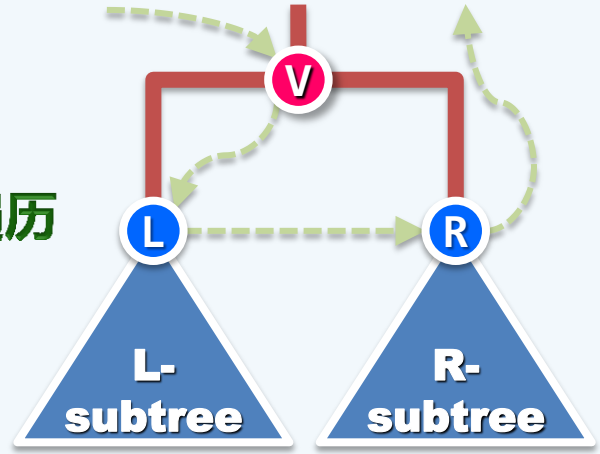
先访问根节点r

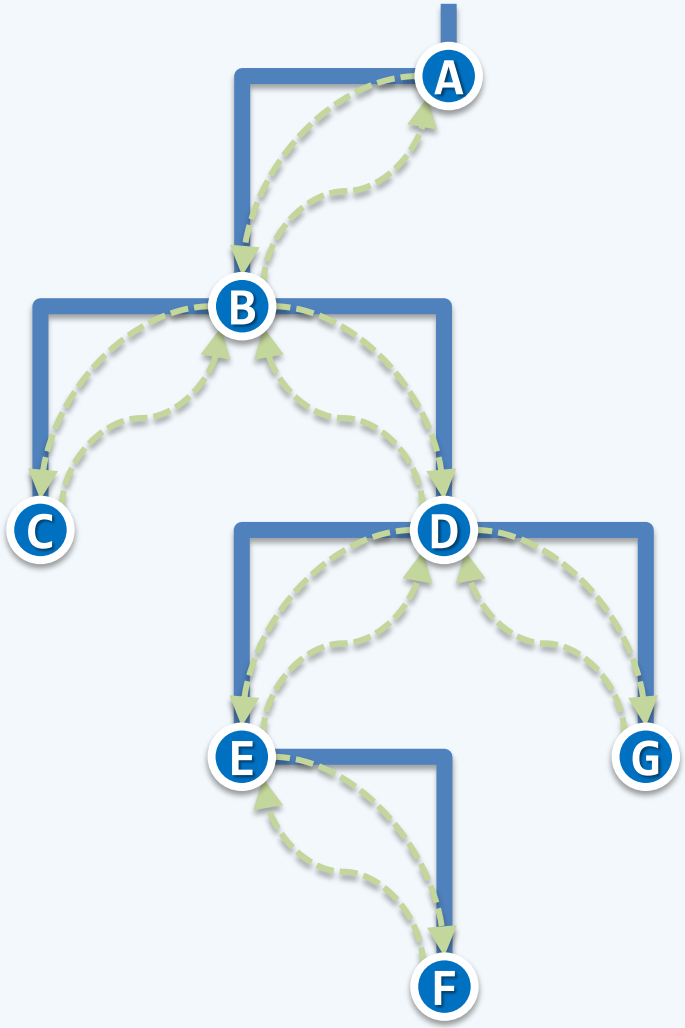
再先后递归地遍历TL和TR

❖ 递归实现中，对左、右子树的递归遍历

都类似于尾递归

故不难直接消除



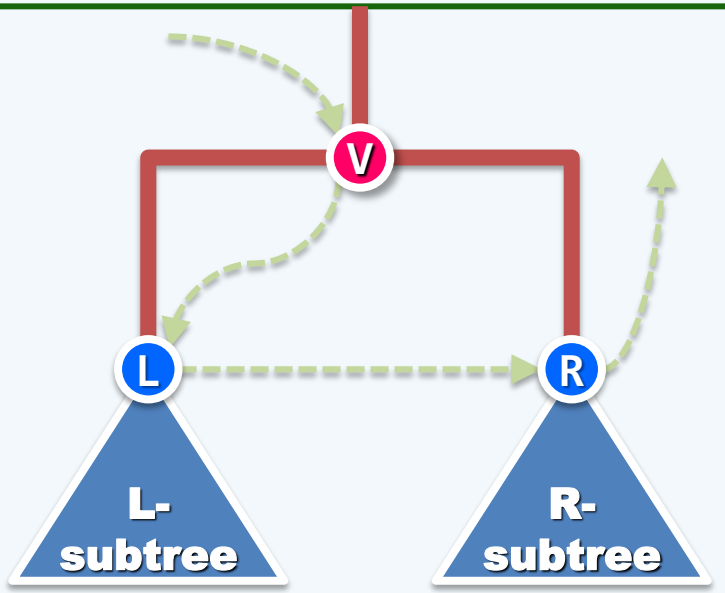


❖ 思路：

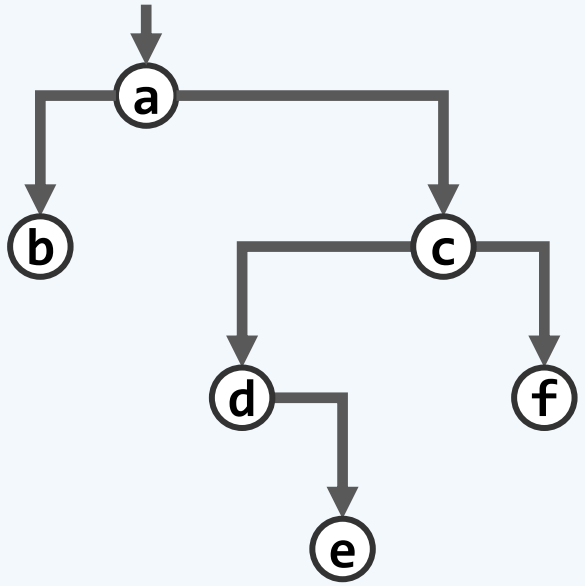
二分递归 → 迭代 + 单递归 → 迭代 + 栈

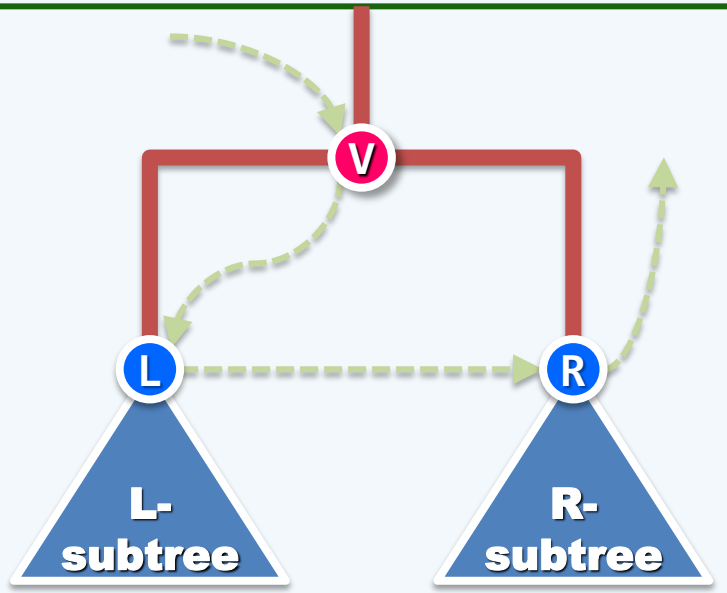
算法

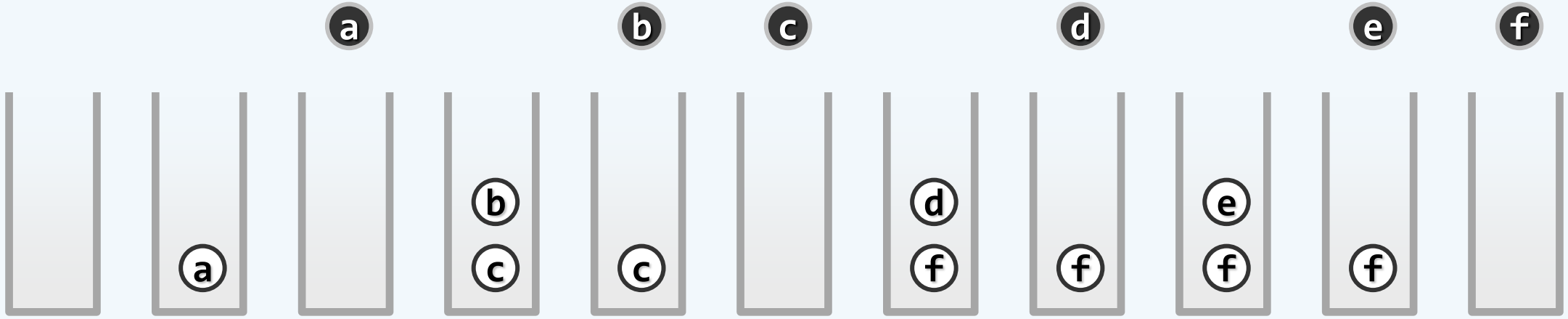
```
❖ template <typename T, typename VST>
  void travPre_I1( BinNodePosi(T) x, VST & visit ) {
     Stack < BinNodePosi(T) > S; //辅助栈
     if (x) S.push( x ); //根节点入栈
     while ( ! S.empty() ) { //在栈变空之前反复循环
        x = S.pop(); visit( x->data ); //弹出并访问当前节点
        if ( HasRChild( *x ) ) S.push( x->rc ); //右孩子先入后出
        if ( HasLChild( *x ) ) S.push( x->lc  ); //左孩子后入先出
     } //体会以上两句的次序
  }
```



实例

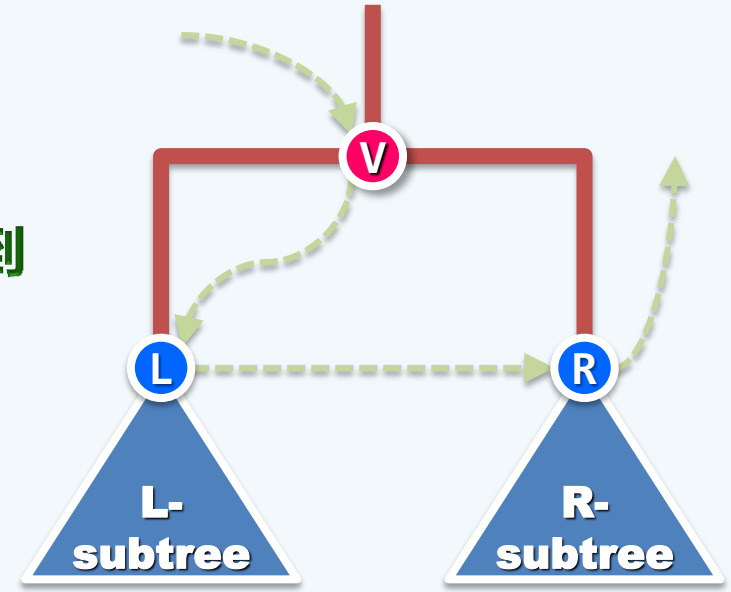






正确性

- **无遗落**：
  - 每个节点都会被访问到
  - 归纳假设：若深度为d的节点都能被正确访问到，则深度为d+1的也是
- **根先**：
  - 对于任一子树，根被访问后才会访问其它节点
  - 只需注意到：若u是v的真祖先，则u必先于v被访问到
- **左先右后**：
  - 同一节点的左子树，先于右子树被访问



效率

- $O(n)$
  - 每步迭代，都有一个节点出栈并被访问
  - 每个节点入/出栈一次且仅一次
  - 每步迭代只需$O(1)$时间
- 以上消除尾递归的思路不易推广

  需要另寻他法...