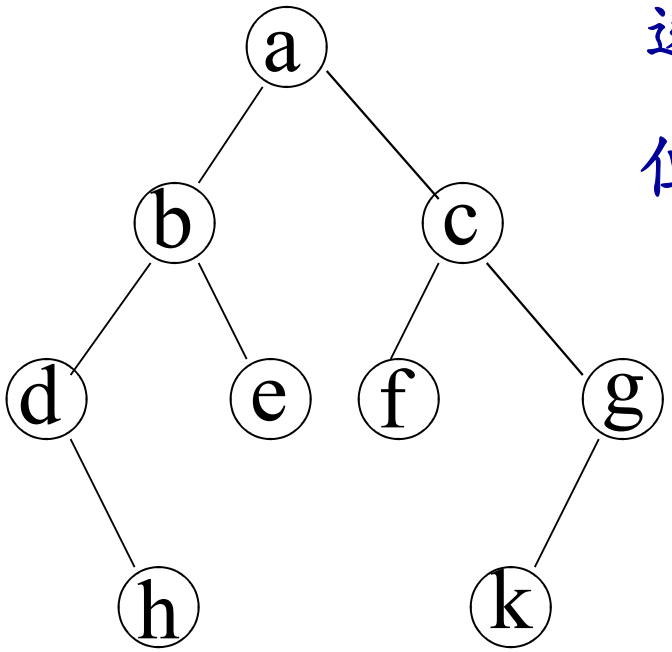
6.3二叉树遍历的非递归算法



递归算法书写简单，

但是效率低

先（根）序遍历算法：

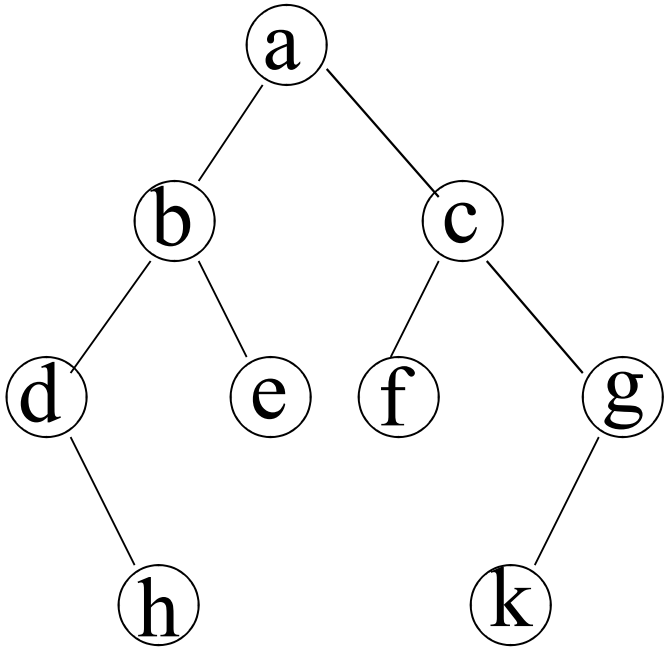
若二叉树为空树，则空操作；否则，

（1）访问根结点；

（2）先序遍历左子树；

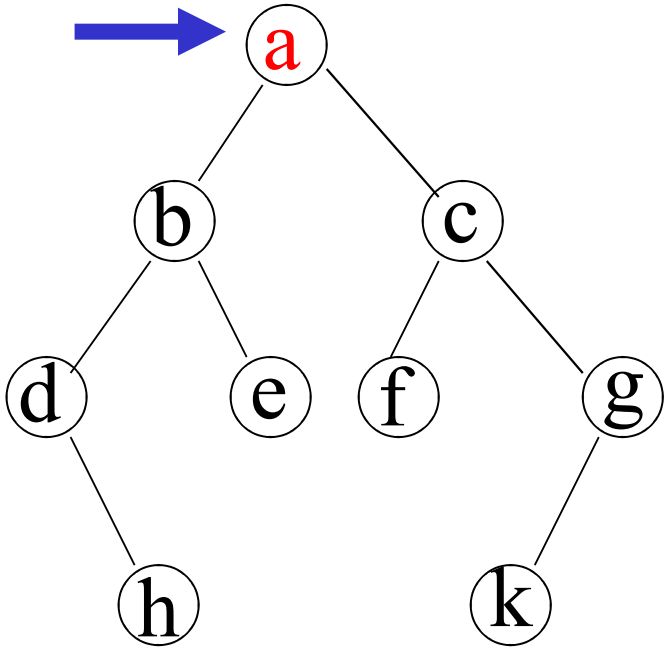
（3）先序遍历右子树。

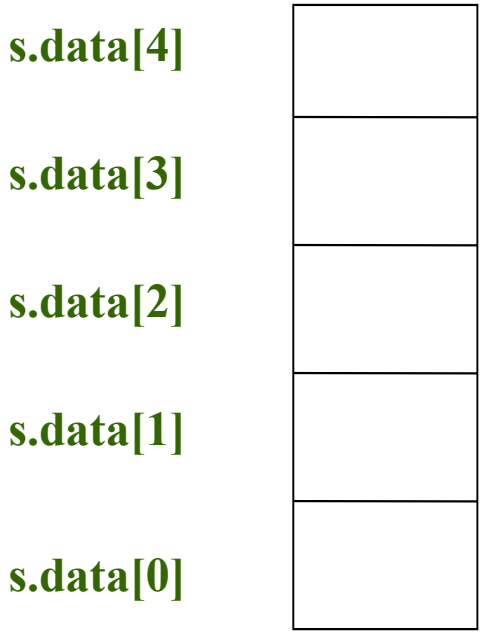
先序遍历：abdhecfgk



二叉树先序遍历的非递归算法实现

a





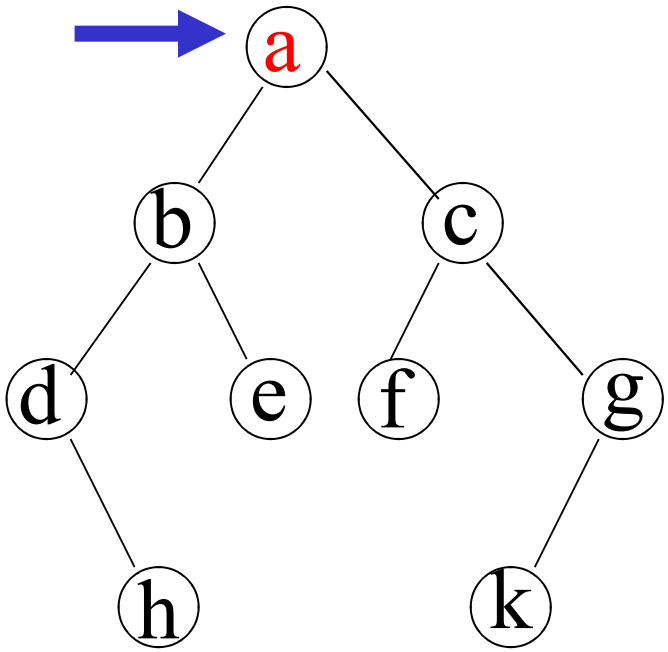
s.top=-1

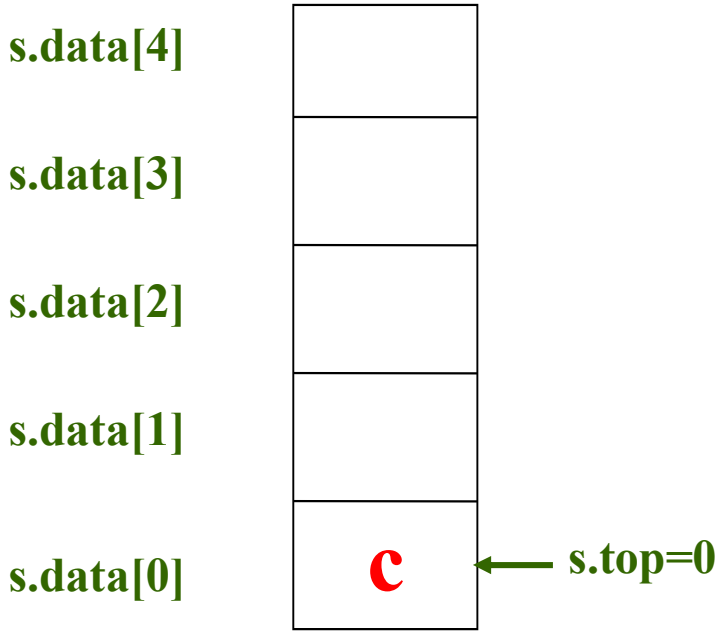
先序遍历非递归算法的实现：访问根结点后，在访问左子树前，应保存其非空右子树

图中a的右子树先于b的右子树被保存，但是其访问要在b的右子树被访问后进行----先保存后访问----先进后出----借助栈实现

二叉树先序遍历的非递归算法实现

a



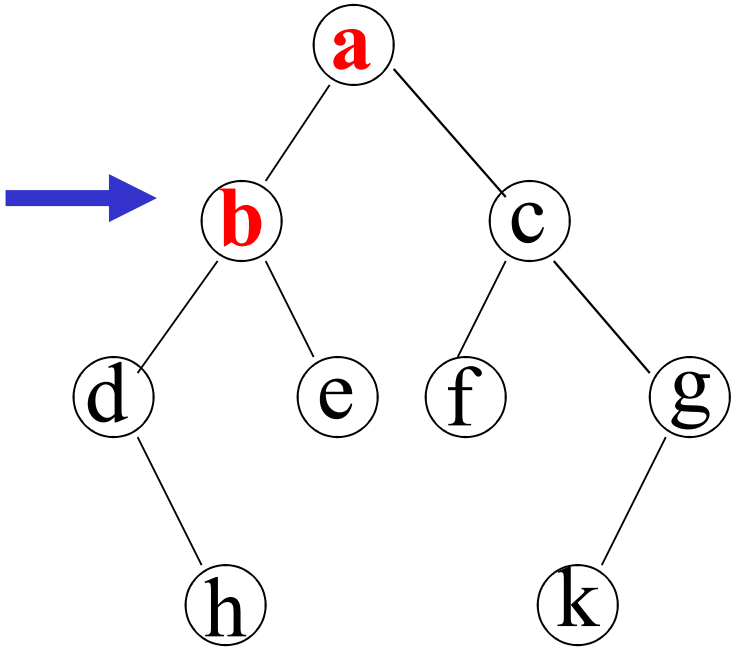


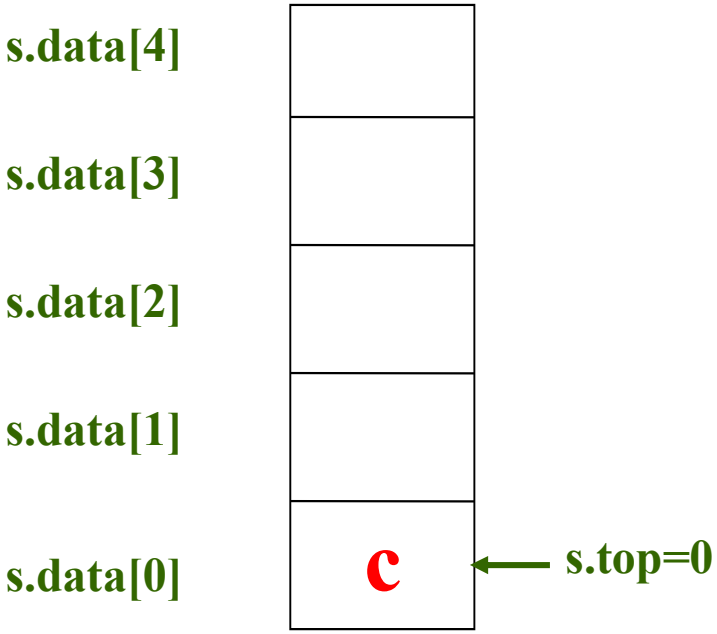
先序遍历非递归算法的实现：访问根结点后，在访问左子树前，应保存其非空右子树

图中a的右子树先于b的右子树被保存，但是其访问要在b的右子树被访问后进行----先保存后访问----先进后出----借助栈实现

二叉树先序遍历的非递归算法实现

ab



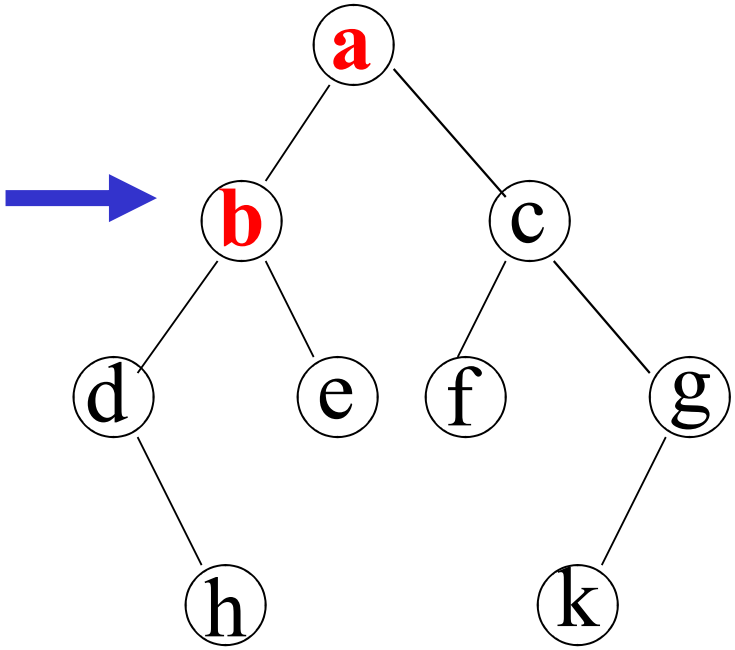


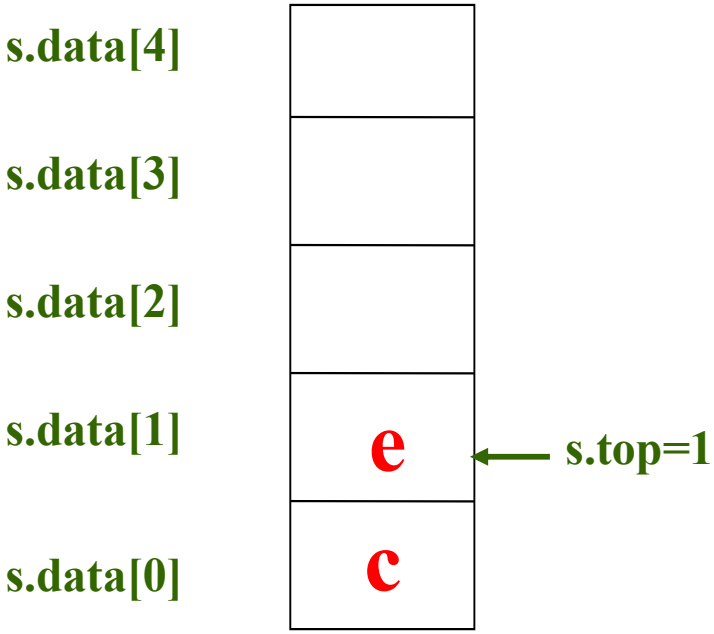
先序遍历非递归算法的实现：访问根结点后，在访问左子树前，应保存其非空右子树

图中a的右子树先于b的右子树被保存，但是其访问要在b的右子树被访问后进行----先保存后访问----先进后出----借助栈实现

二叉树先序遍历的非递归算法实现

ab



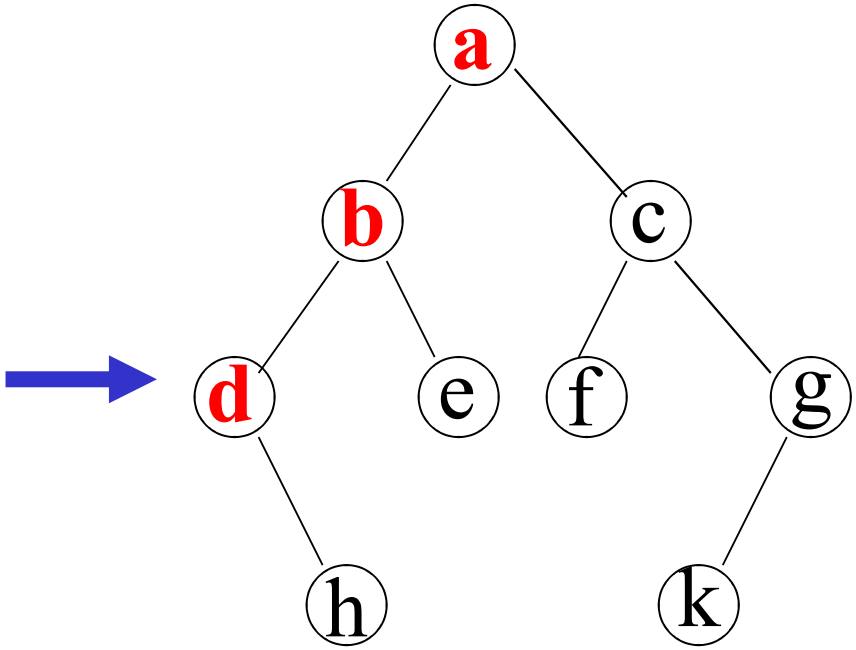


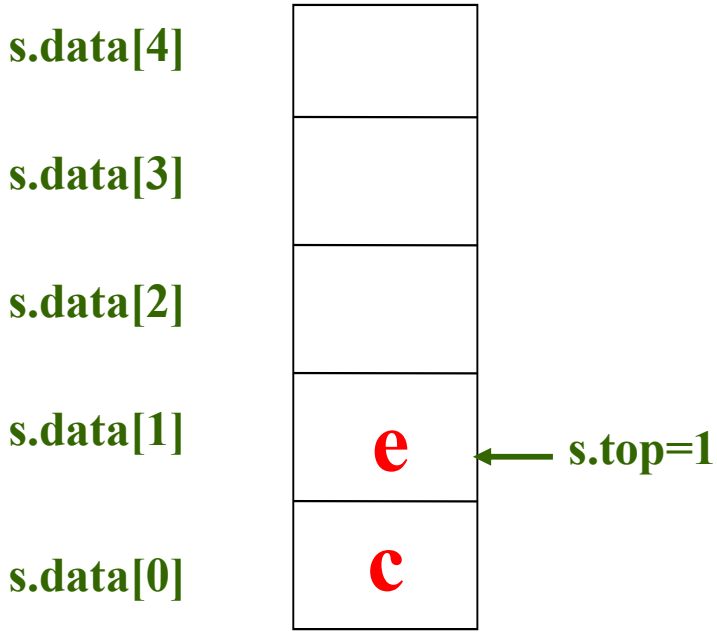
先序遍历非递归算法的实现：访问根结点后，在访问左子树前，应保存其非空右子树

图中a的右子树先于b的右子树被保存，但是其访问要在b的右子树被访问后进行----先保存后访问----先进后出----借助栈实现

二叉树先序遍历的非递归算法实现

abd



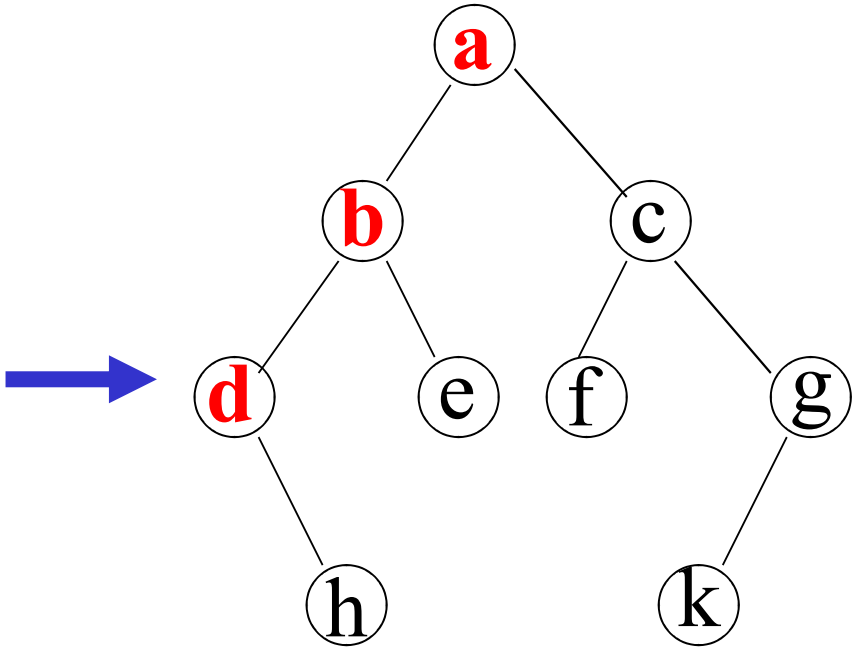


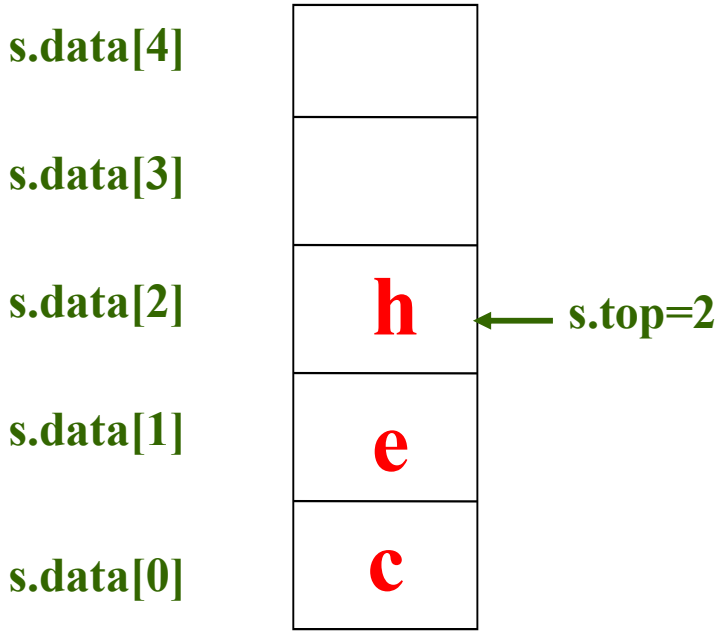
先序遍历非递归算法的实现：访问根结点后，在访问左子树前，应保存其非空右子树

图中a的右子树先于b的右子树被保存，但是其访问要在b的右子树被访问后进行----先保存后访问----先进后出----借助栈实现

二叉树先序遍历的非递归算法实现

abd



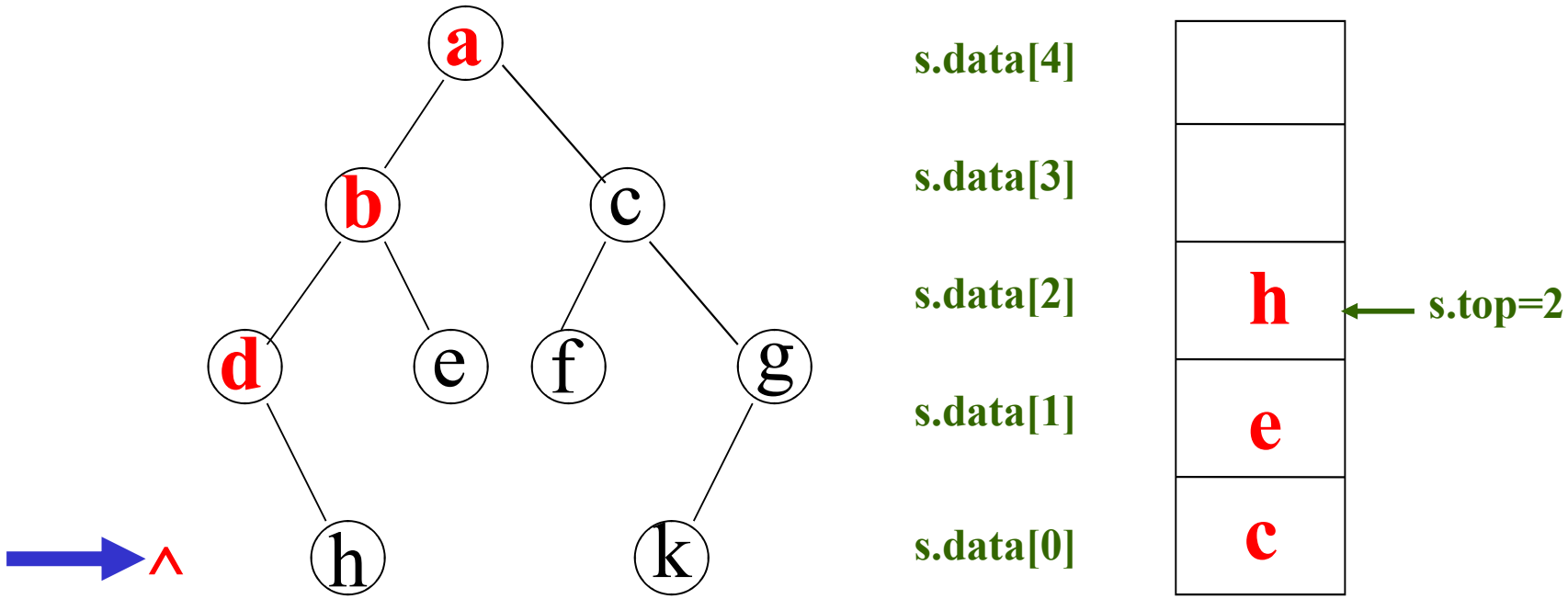


先序遍历非递归算法的实现：访问根结点后，在访问左子树前，应保存其非空右子树

图中a的右子树先于b的右子树被保存，但是其访问要在b的右子树被访问后进行----先保存后访问----先进后出----借助栈实现

二叉树先序遍历的非递归算法实现

abd

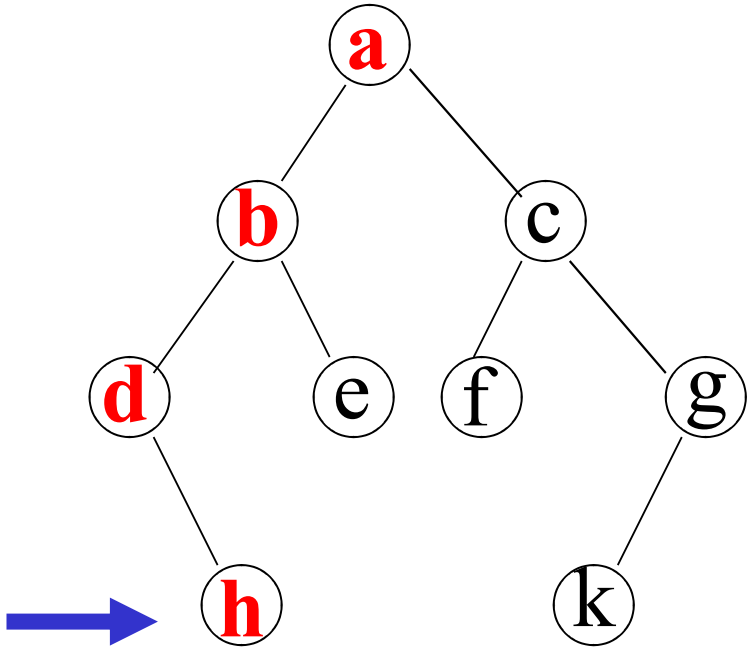


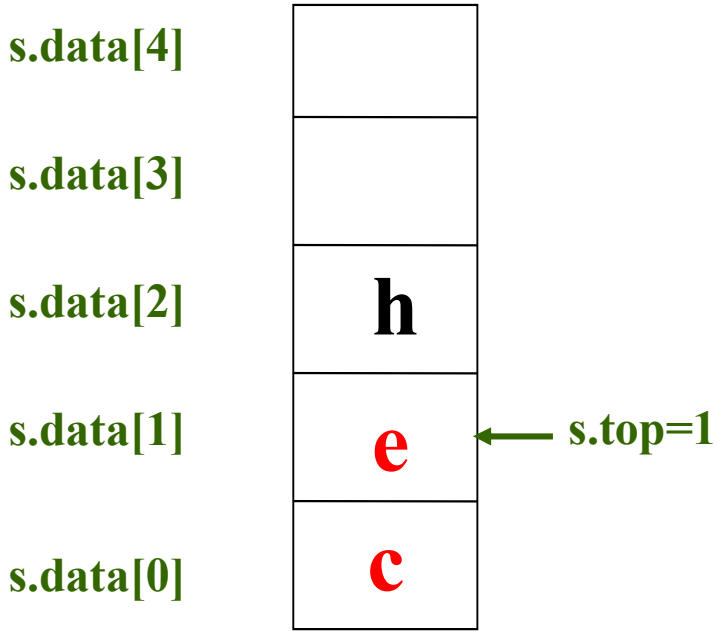
先序遍历非递归算法的实现：访问根结点后，在访问左子树前，应保存其非空右子树

图中a的右子树先于b的右子树被保存，但是其访问要在b的右子树被访问后进行----先保存后访问----先进后出----借助栈实现

二叉树先序遍历的非递归算法实现

abdh



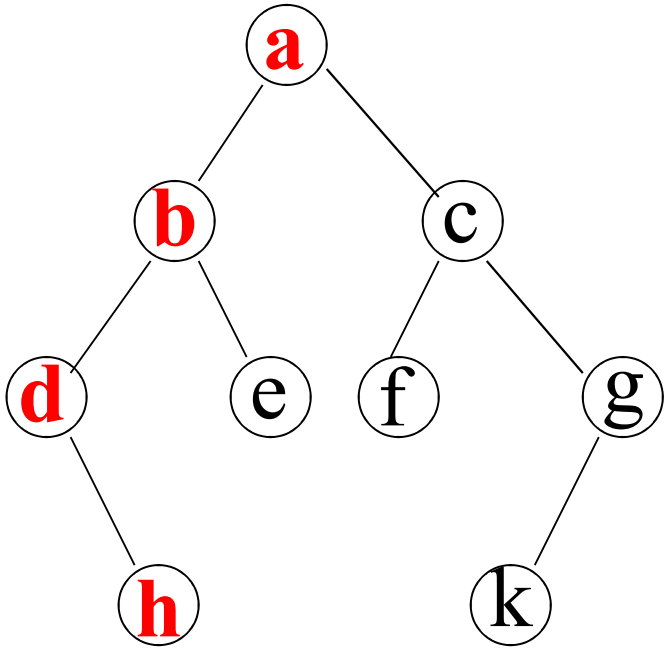


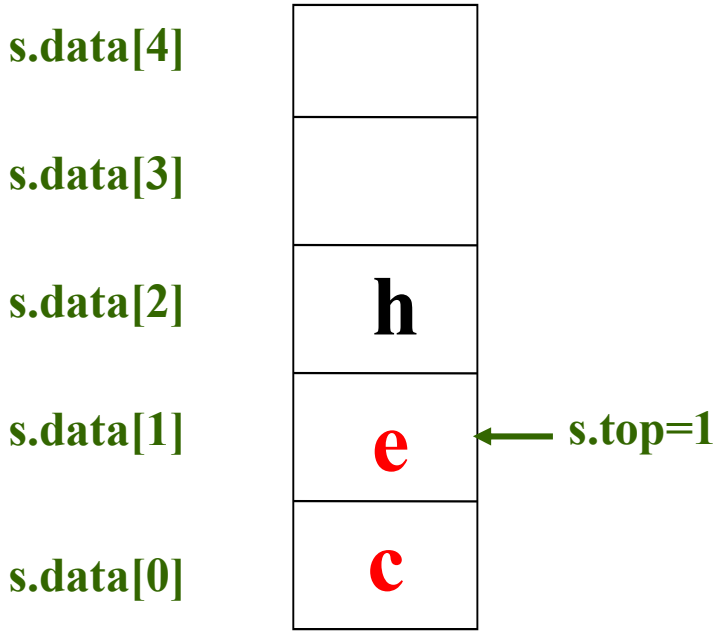
先序遍历非递归算法的实现：访问根结点后，在访问左子树前，应保存其非空右子树

图中a的右子树先于b的右子树被保存，但是其访问要在b的右子树被访问后进行----先保存后访问----先进后出----借助栈实现

二叉树先序遍历的非递归算法实现

abdh



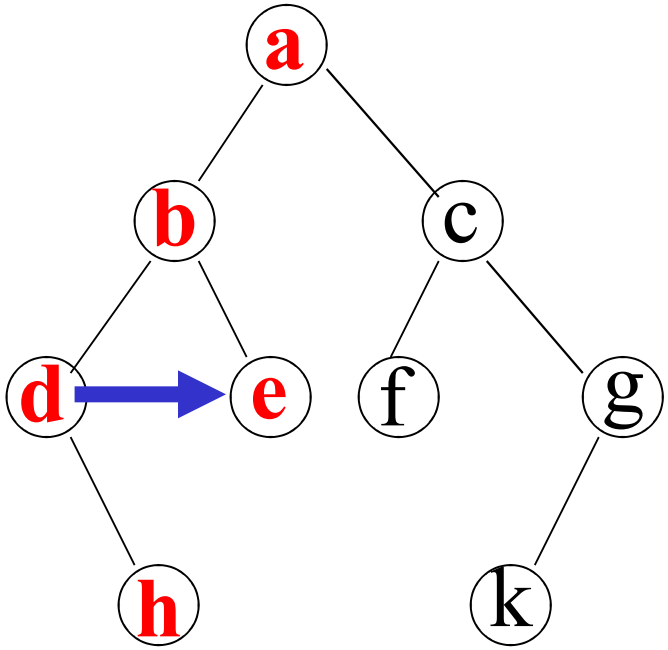


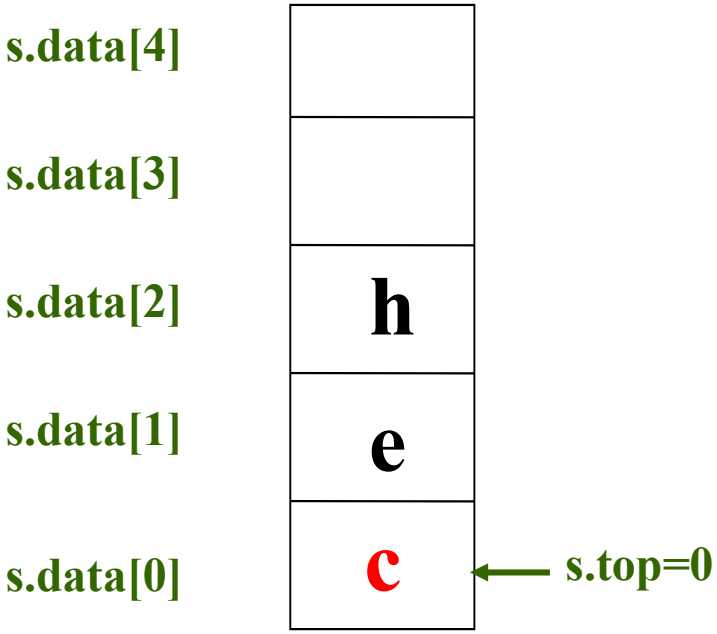
先序遍历非递归算法的实现：访问根结点后，在访问左子树前，应保存其非空右子树

图中a的右子树先于b的右子树被保存，但是其访问要在b的右子树被访问后进行----先保存后访问----先进后出----借助栈实现

二叉树先序遍历的非递归算法实现

abdhe



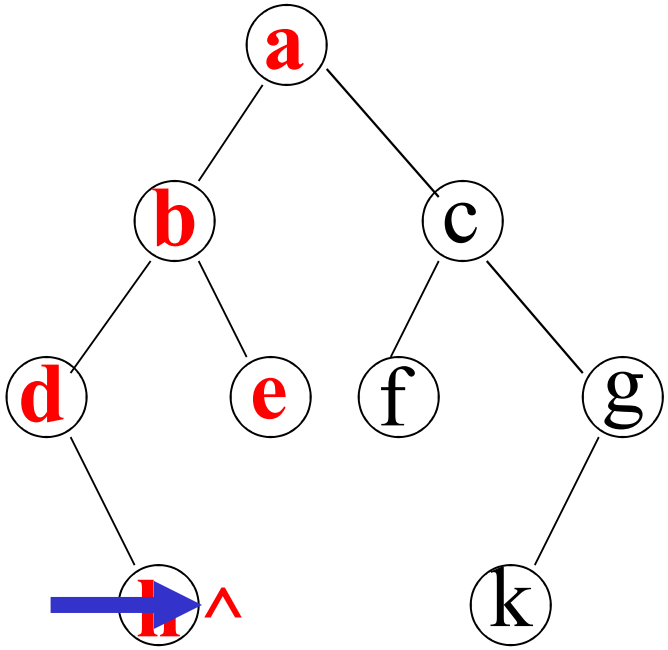


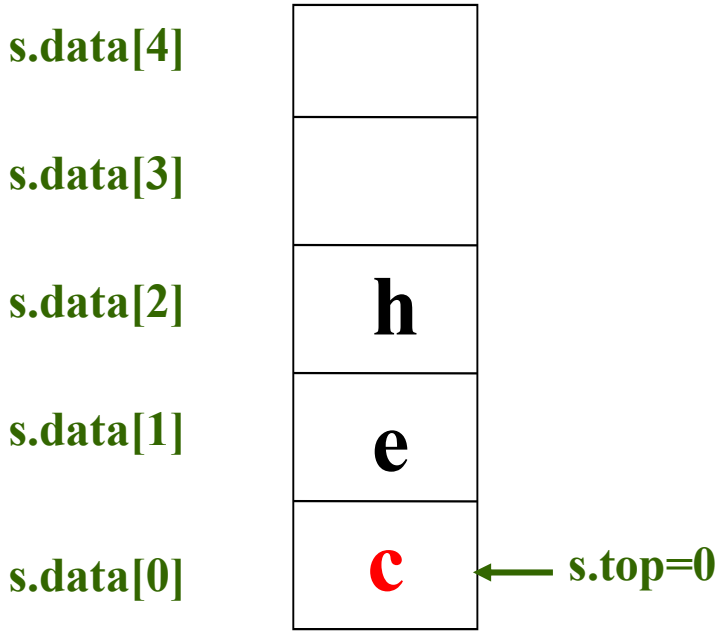
先序遍历非递归算法的实现：访问根结点后，在访问左子树前，应保存其非空右子树

图中a的右子树先于b的右子树被保存，但是其访问要在b的右子树被访问后进行----先保存后访问----先进后出----借助栈实现

二叉树先序遍历的非递归算法实现

abdhe



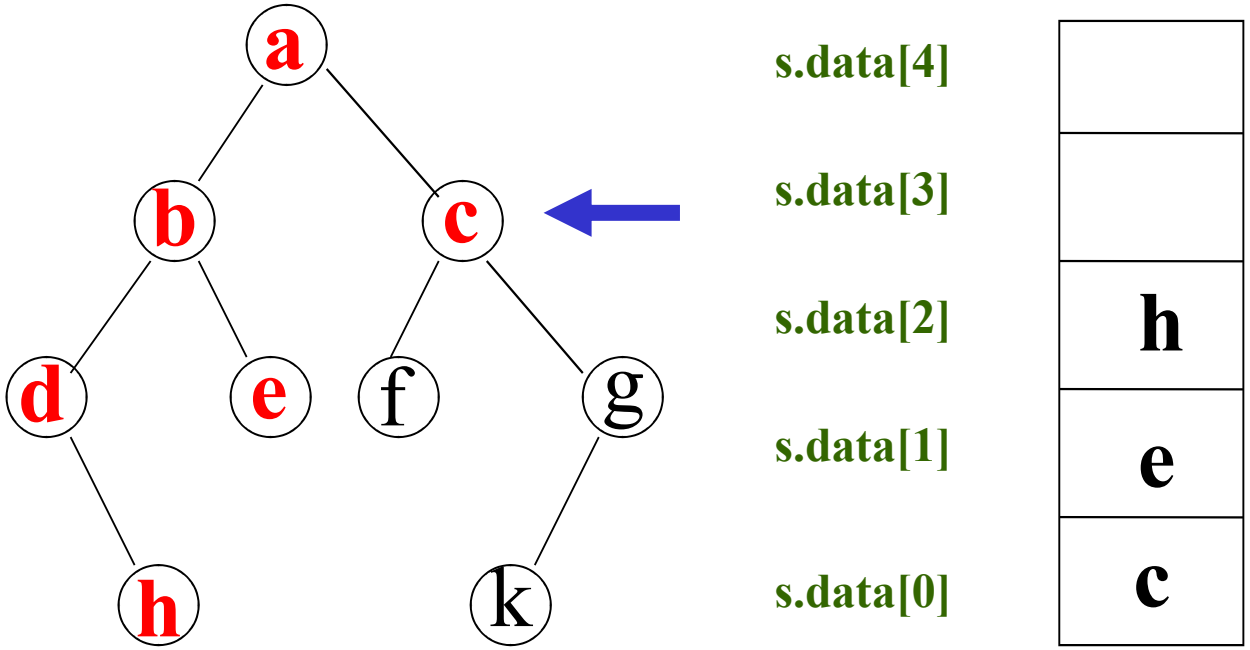


先序遍历非递归算法的实现：访问根结点后，在访问左子树前，应保存其非空右子树

图中a的右子树先于b的右子树被保存，但是其访问要在b的右子树被访问后进行----先保存后访问----先进后出----借助栈实现

二叉树先序遍历的非递归算法实现

abdhec



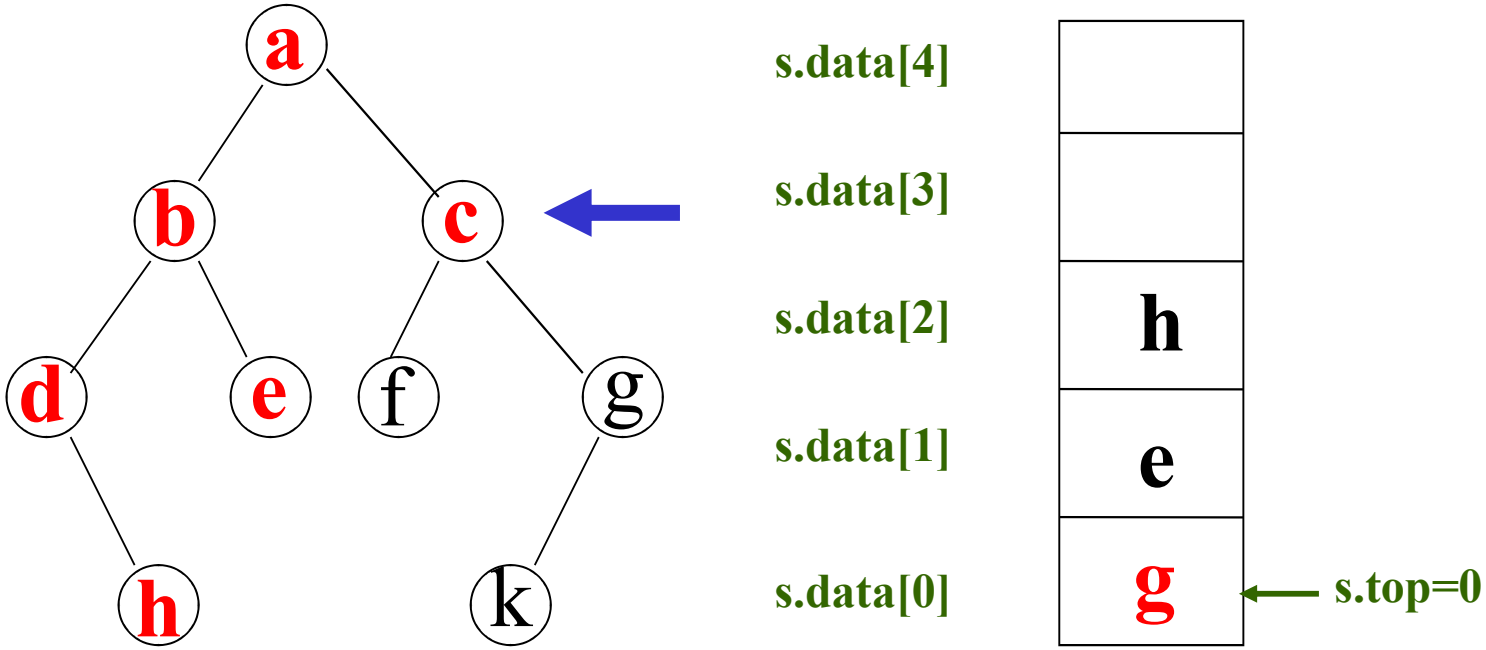
先序遍历非递归算法的实现：访问根结点后，在访问左子树前，应保存其非空右子树

top=-1

图中a的右子树先于b的右子树被保存，但是其访问要在b的右子树被访问后进行----先保存后访问----先进后出----借助栈实现

二叉树先序遍历的非递归算法实现

abdhec

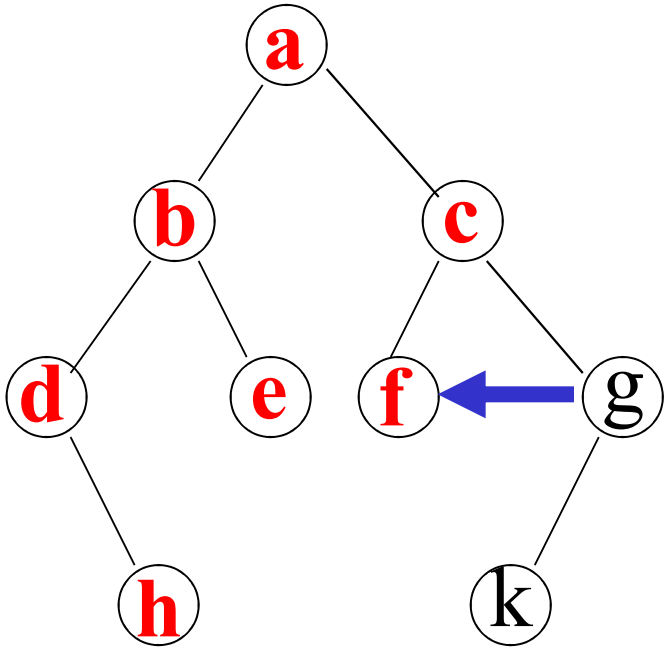


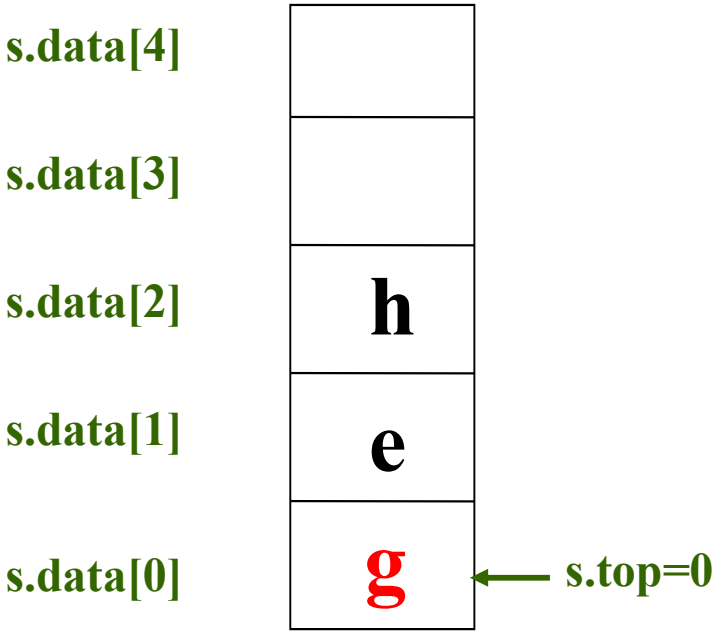
先序遍历非递归算法的实现：访问根结点后，在访问左子树前，应保存其非空右子树

图中a的右子树先于b的右子树被保存，但是其访问要在b的右子树被访问后进行----先保存后访问----先进后出----借助栈实现

二叉树先序遍历的非递归算法实现

abdhecf



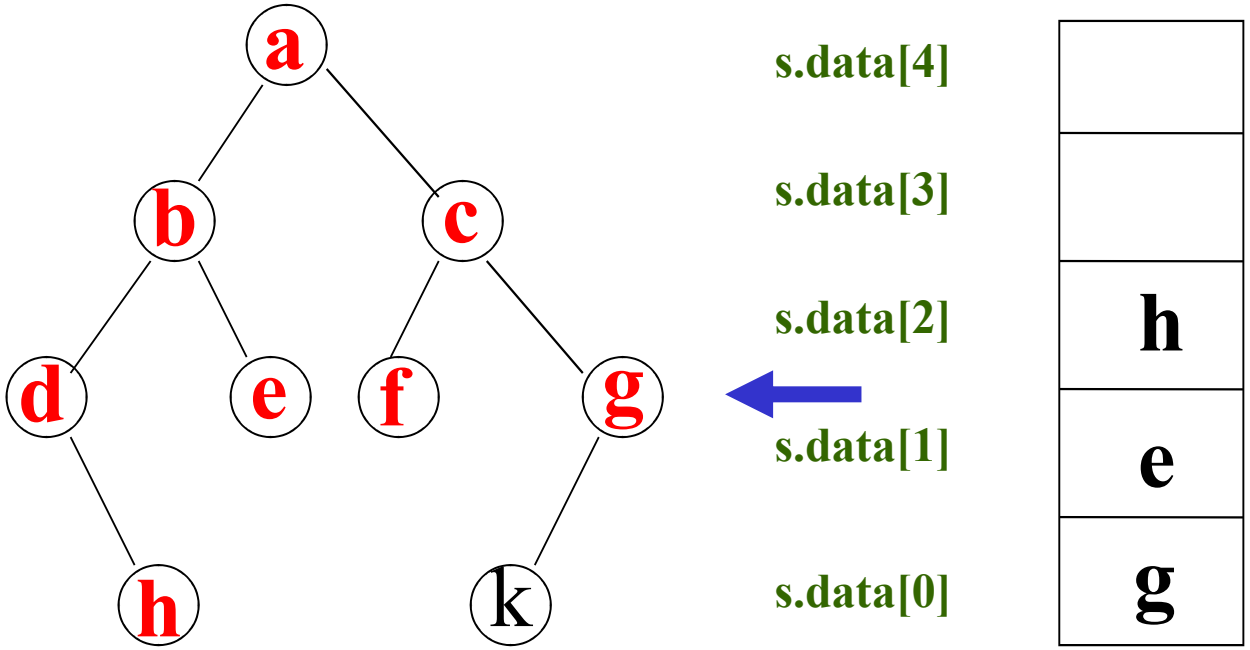


先序遍历非递归算法的实现：访问根结点后，在访问左子树前，应保存其非空右子树

图中a的右子树先于b的右子树被保存，但是其访问要在b的右子树被访问后进行----先保存后访问----先进后出----借助栈实现

二叉树先序遍历的非递归算法实现

abdhecfg

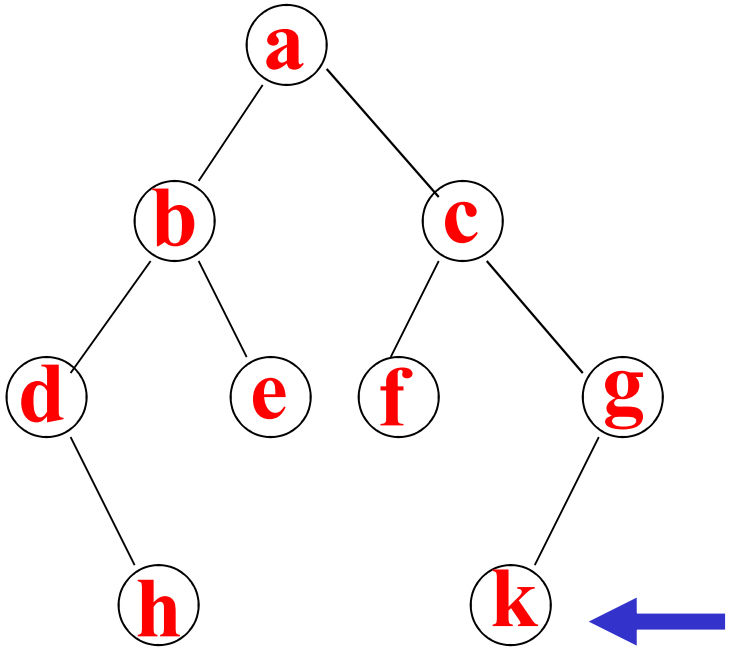


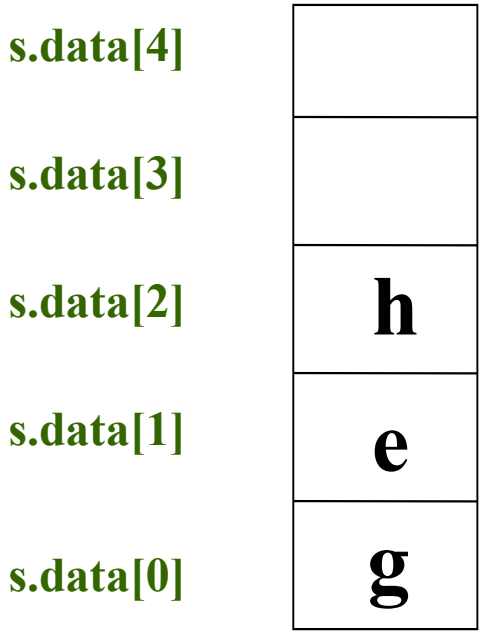
先序遍历非递归算法的实现：访问根结点后，在访问左子树前，应保存其非空右子树 ← s.top=-1

图中a的右子树先于b的右子树被保存，但是其访问要在b的右子树被访问后进行----先保存后访问----先进后出----借助栈实现

二叉树先序遍历的非递归算法实现

abdhecfgk





先序遍历非递归算法的实现：访问根结点后，在访问左子树前，应保存其非空右子树 ← s.top=-1

图中a的右子树先于b的右子树被保存，但是其访问要在b的右子树被访问后进行----先保存后访问----先进后出----借助栈实现

先序遍历的非递归描述

- 采用顺序栈：

```
#define MAX  10000
typedef  struct
{   BiTree  data[MAX];
    int  top;
}SeqStack;
```

- 非递归算法的步骤：

1. p=T; 初始化空栈；//T为二叉树的根结点
2. 若p 存在, 则访问p , 将p的非空右孩子入栈 , p=p→lc,后转至 2；否则转3；
3. 若栈不空，取栈顶元素→p, 转2；否则结束。

先序遍历算法的非递归描述

```
void PreorderTraverse(BiTree T){//T为二叉树的根结点
  SeqStack  s ;
  s.top=-1; p = T;
  while(p){
    while(p){printf("%c",p->data); //访问p结点
              if(p->rc) //将p结点的非空右孩子入栈保存
                if(s.top==MAX-1) exit (0);
                else s.data[++s.top]=p->rc;
                  p =p->lc; //访问p的左孩子
              }
    if (s.top!=-1)  p=s.data[s.top--];
  }
}
```

中（根）序的遍历算法：

若二叉树为空树，则空操作；否则，

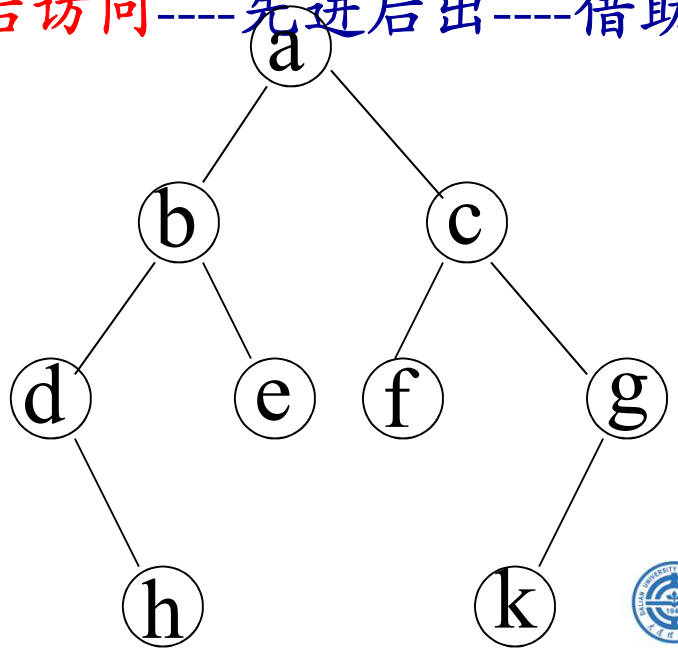
（1）中序遍历左子树；

（2）访问根结点；

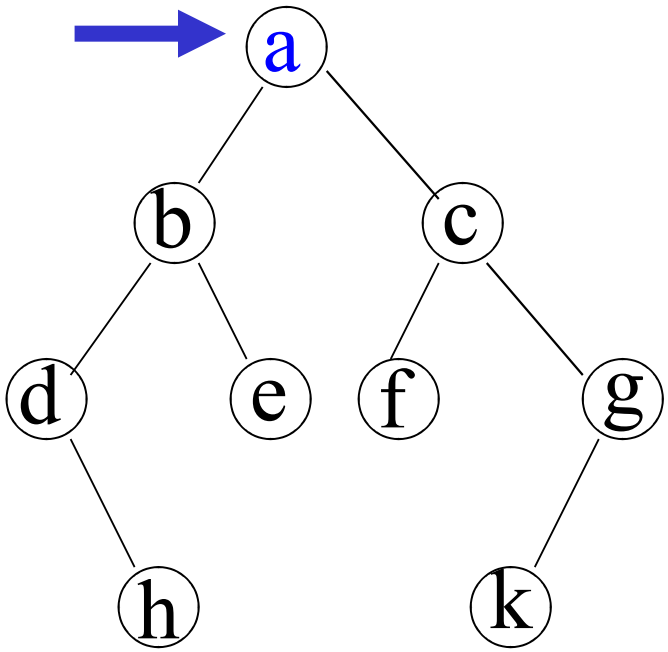
（3）中序遍历右子树。

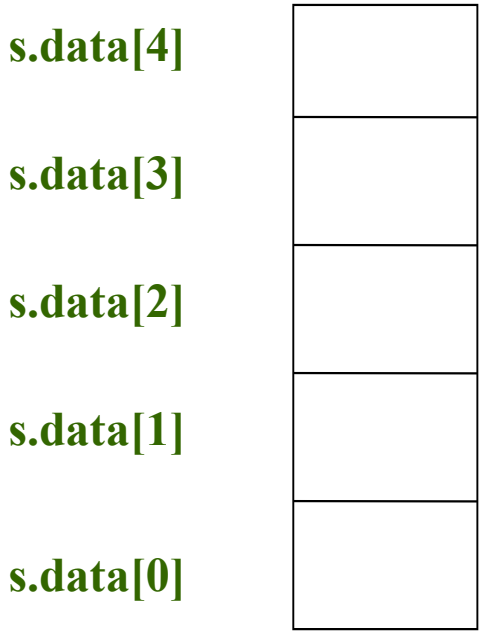
中序遍历的递归算法

1. 中序遍历非递归算法的实现：访问根结点的左子树前，应保存其根结点，以便左子树访问结束后，访问根和根的右子树
2. 图中a结点先于b结点被保存，但是其访问要在b及其右子树被访问后进行----先保存后访问----先进后出----借助栈实现



二叉树中序遍历的非递归算法实现

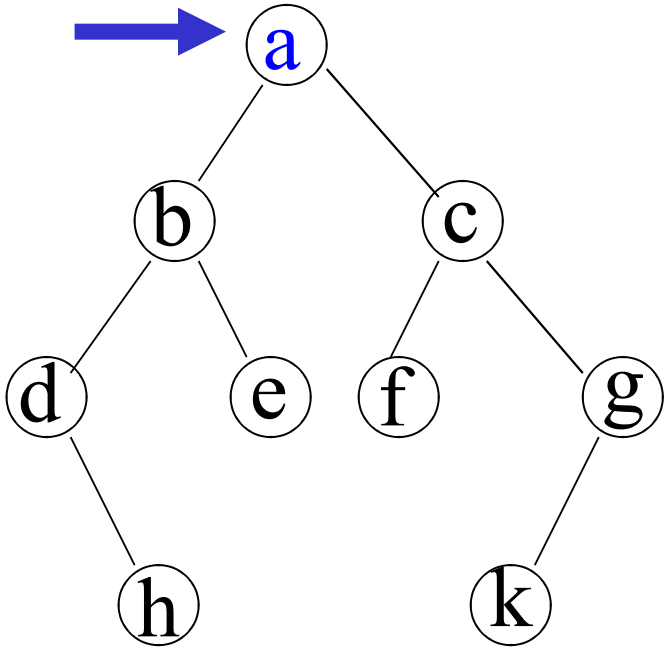


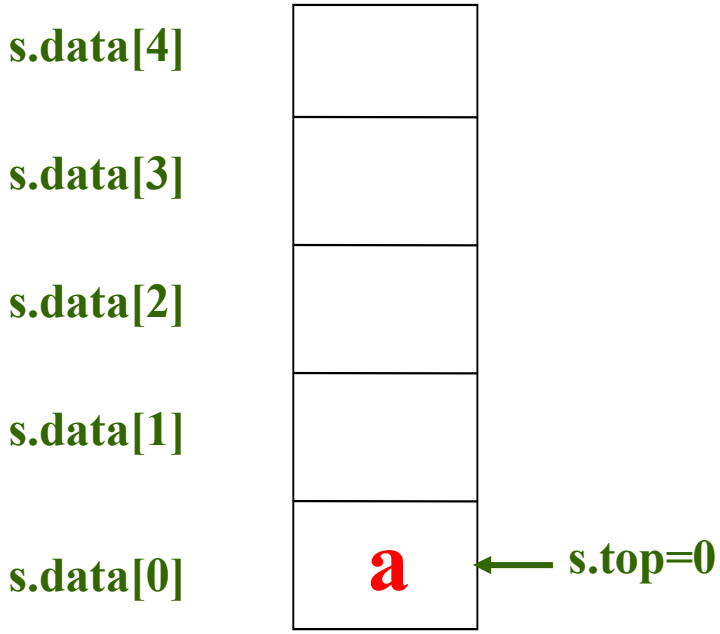


s.top=-1

中序遍历非递归算法的实现：在访问左子树前，应保存根结点，如此左子树访问结束后，可以找到根结点，访问根结点和根的右子树

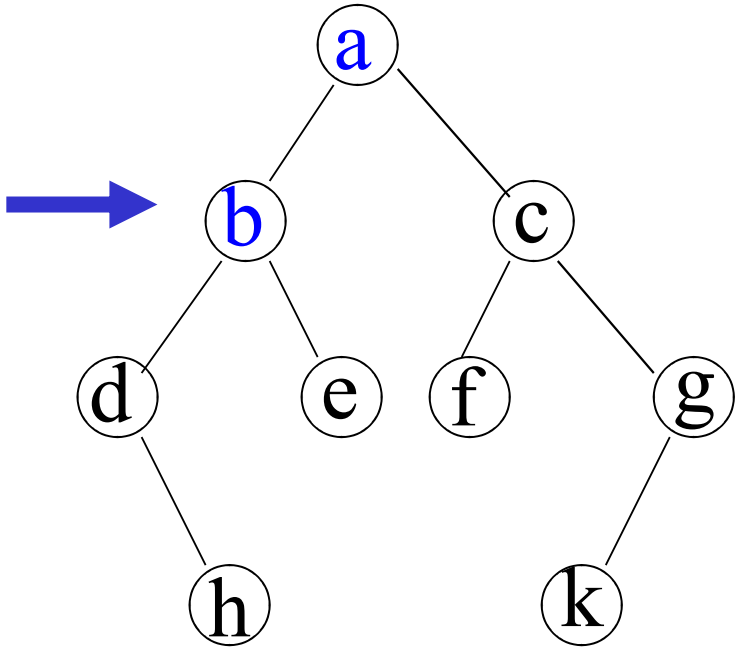
二叉树中序遍历的非递归算法实现

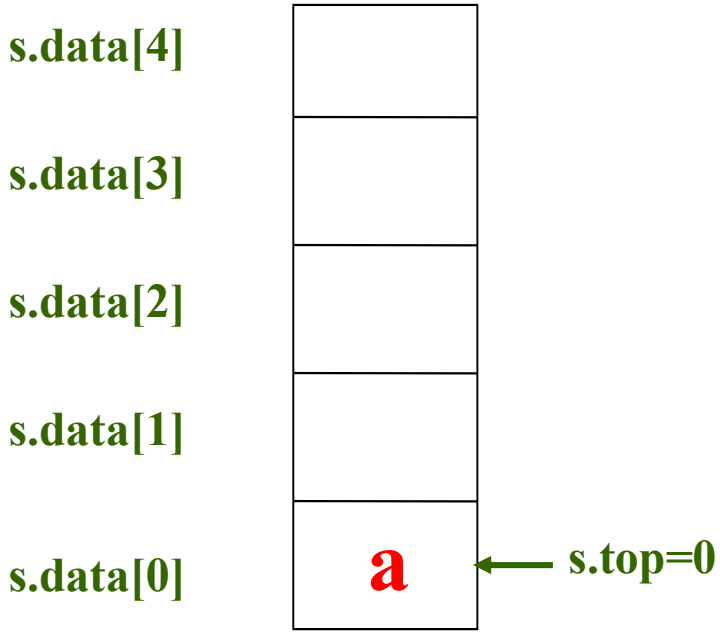




中序遍历非递归算法的实现：在访问左子树前，应保存根结点，如此左子树访问结束后，可以找到根结点，访问根结点和根的右子树

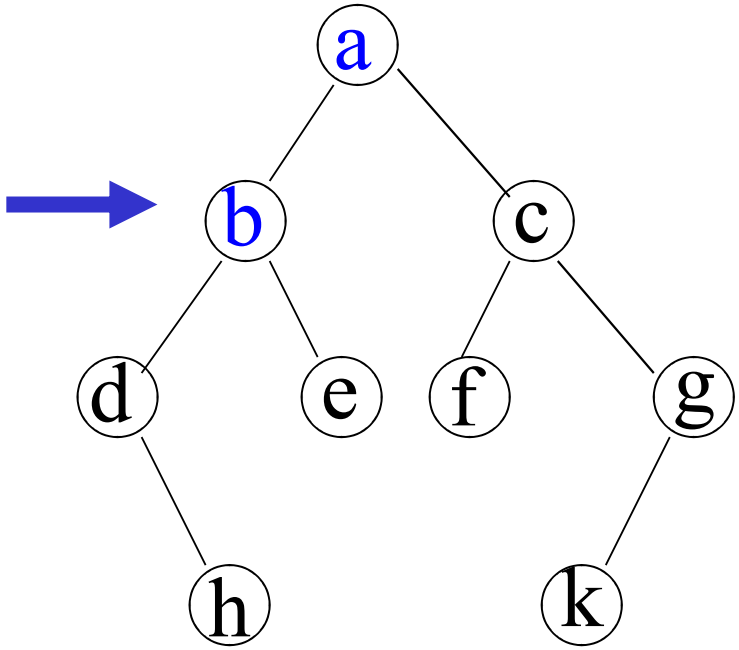
二叉树中序遍历的非递归算法实现

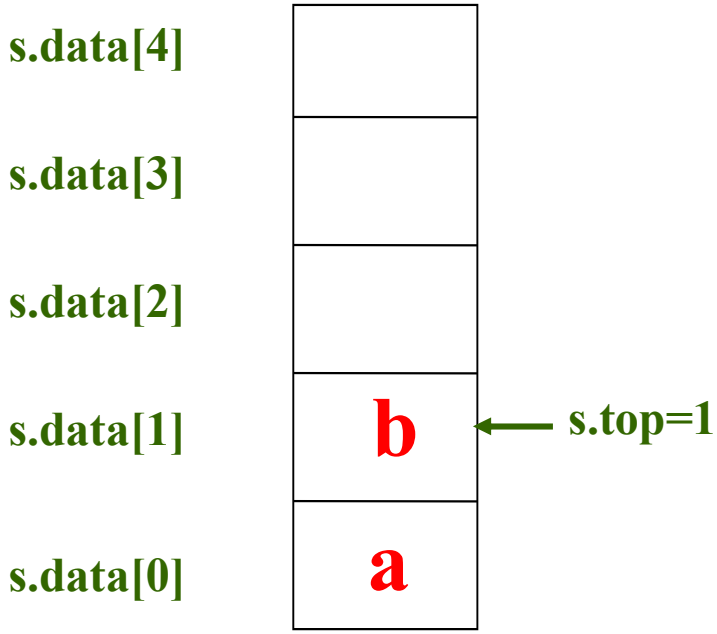




中序遍历非递归算法的实现：在访问左子树前，应保存根结点，如此左子树访问结束后，可以找到根结点，访问根结点和根的右子树

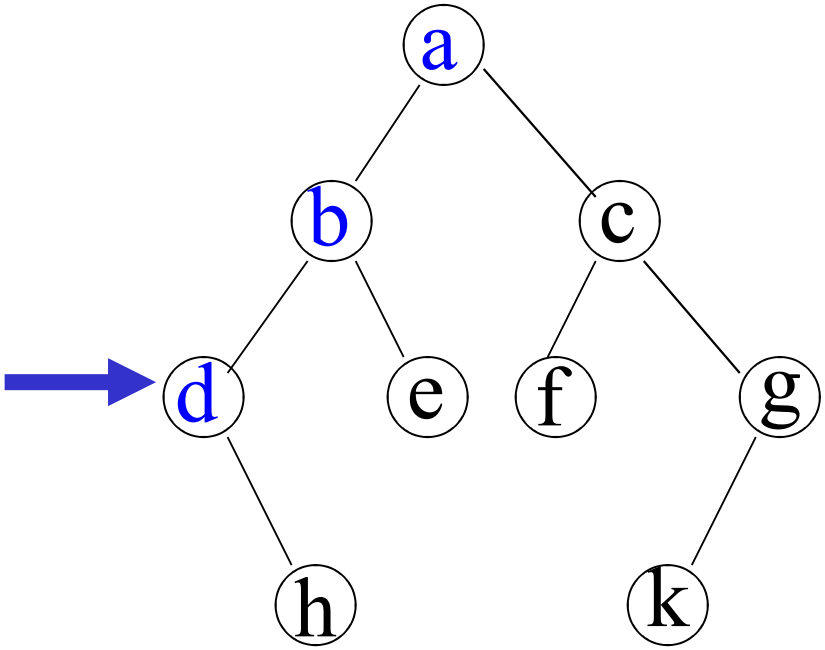
二叉树中序遍历的非递归算法实现

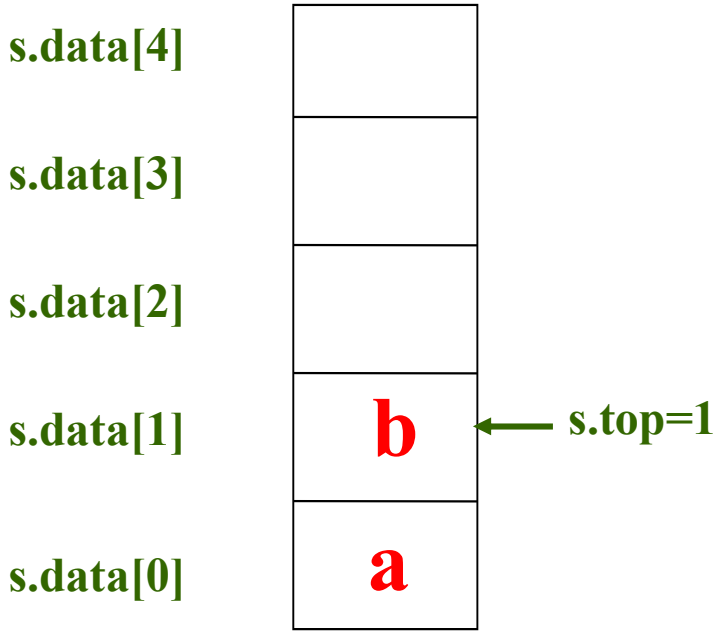




中序遍历非递归算法的实现：在访问左子树前，应保存根结点，如此左子树访问结束后，可以找到根结点，访问根结点和根的右子树

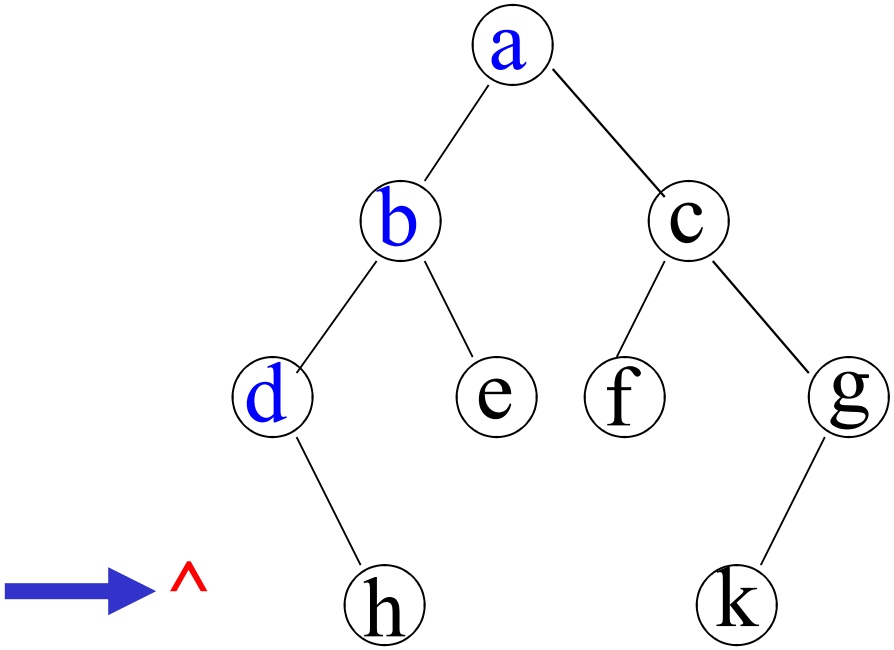
二叉树中序遍历的非递归算法实现

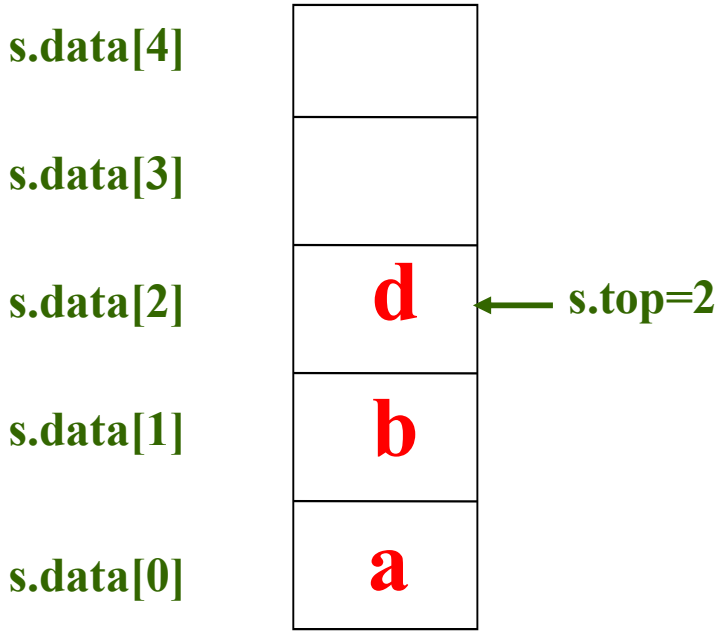




中序遍历非递归算法的实现：在访问左子树前，应保存根结点，如此左子树访问结束后，可以找到根结点，访问根结点和根的右子树

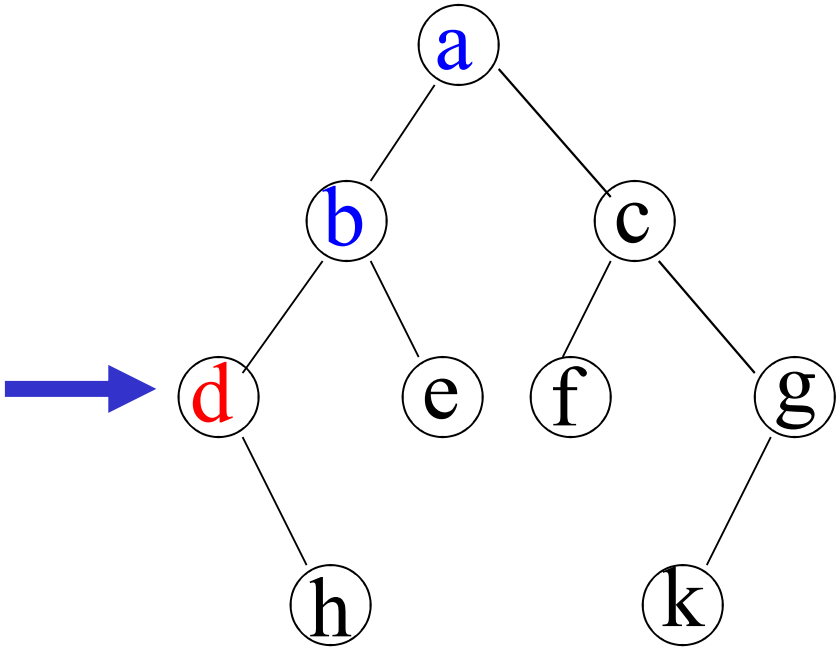
二叉树中序遍历的非递归算法实现

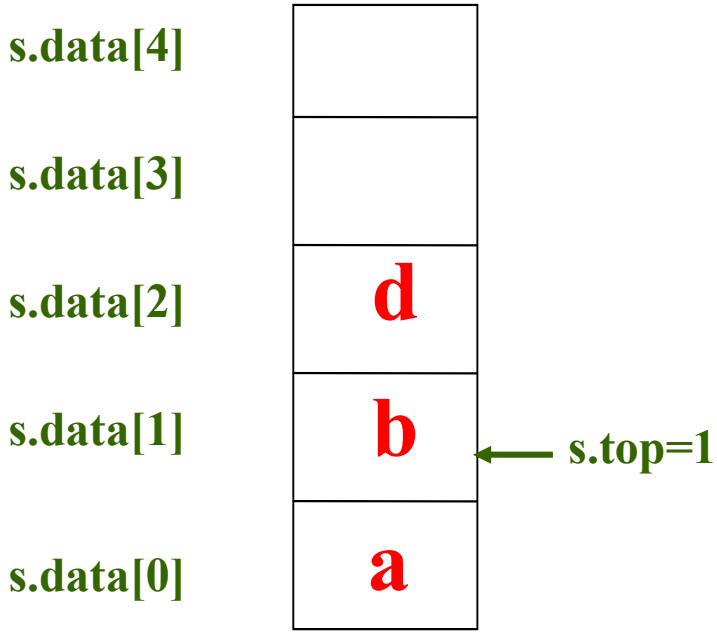




中序遍历非递归算法的实现：在访问左子树前，应保存根结点，如此左子树访问结束后，可以找到根结点，访问根结点和根的右子树

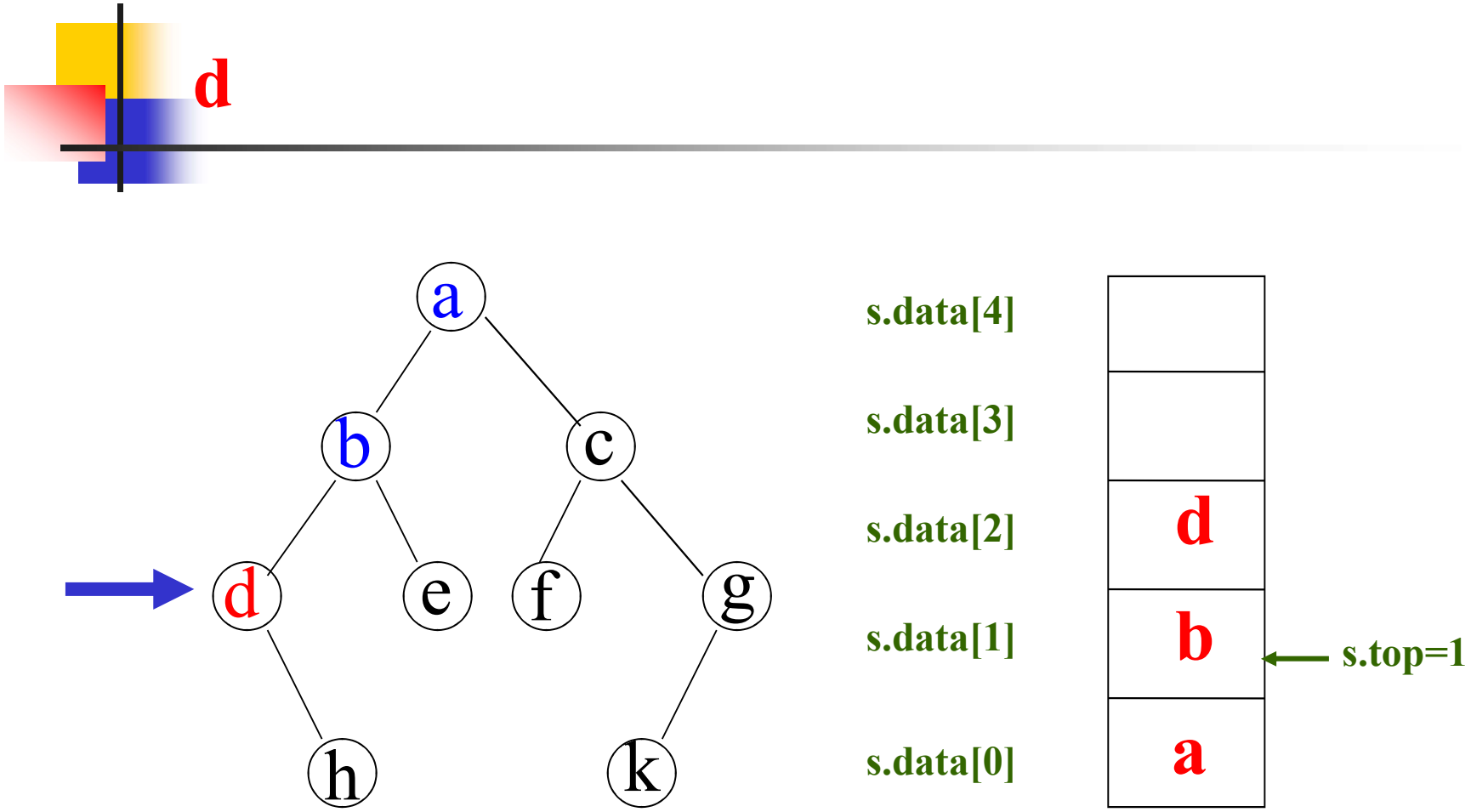
二叉树中序遍历的非递归算法实现





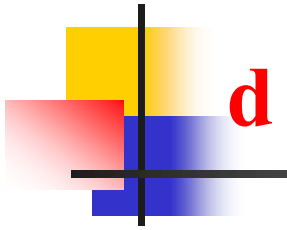
中序遍历非递归算法的实现：在访问左子树前，应保存根结点，如此左子树访问结束后，可以找到根结点，访问根结点和根的右子树

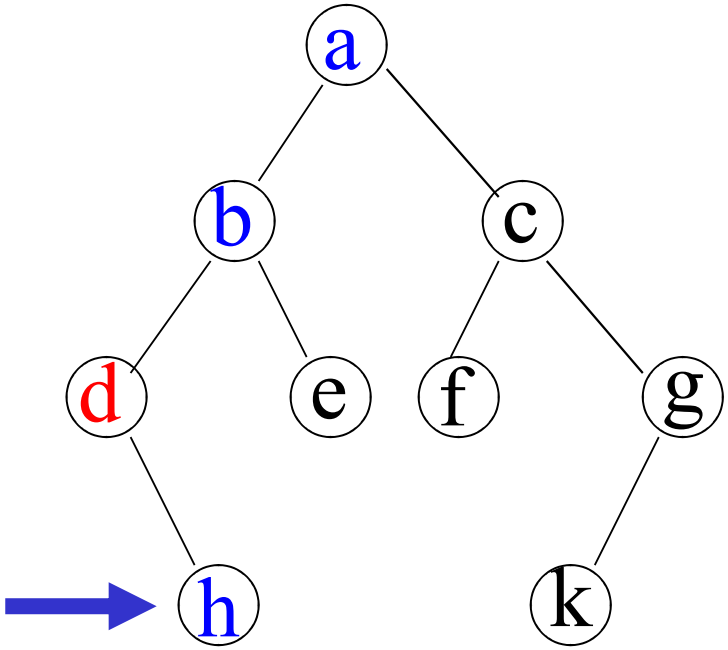
二叉树中序遍历的非递归算法实现

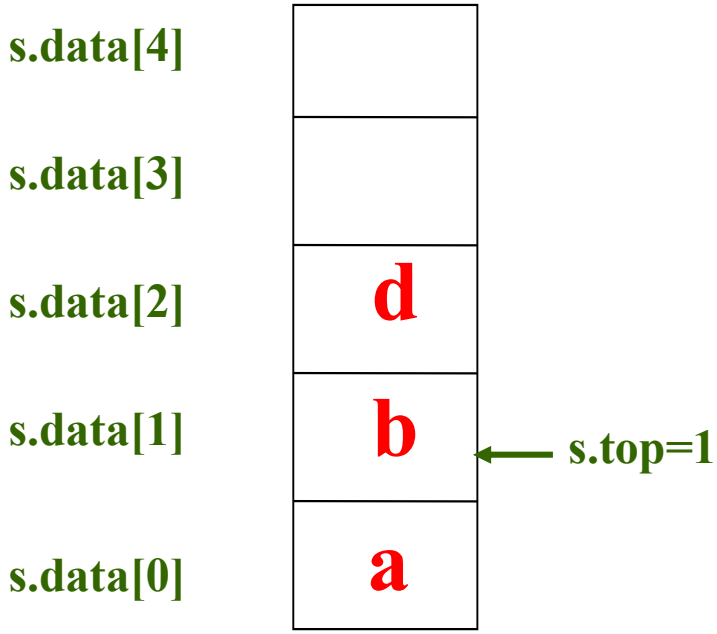


中序遍历非递归算法的实现：在访问左子树前，应保存根结点，如此左子树访问结束后，可以找到根结点，访问根结点和根的右子树

二叉树中序遍历的非递归算法实现

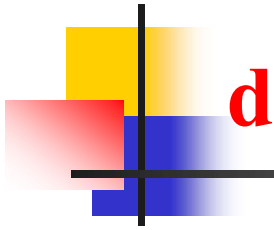


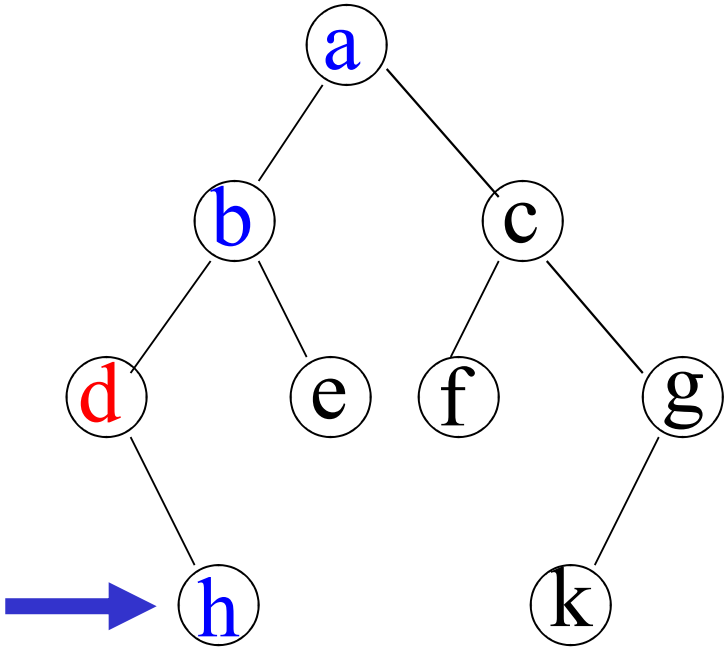


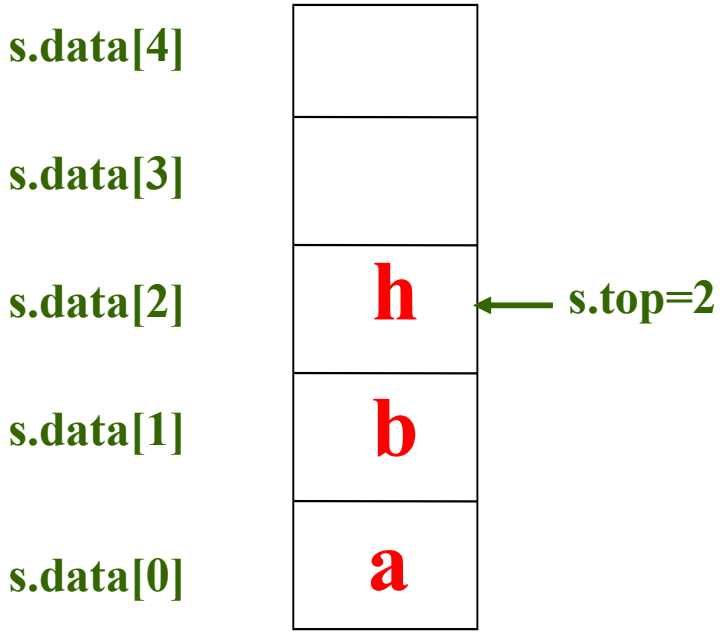


中序遍历非递归算法的实现：在访问左子树前，应保存根结点，如此左子树访问结束后，可以找到根结点，访问根结点和根的右子树

二叉树中序遍历的非递归算法实现



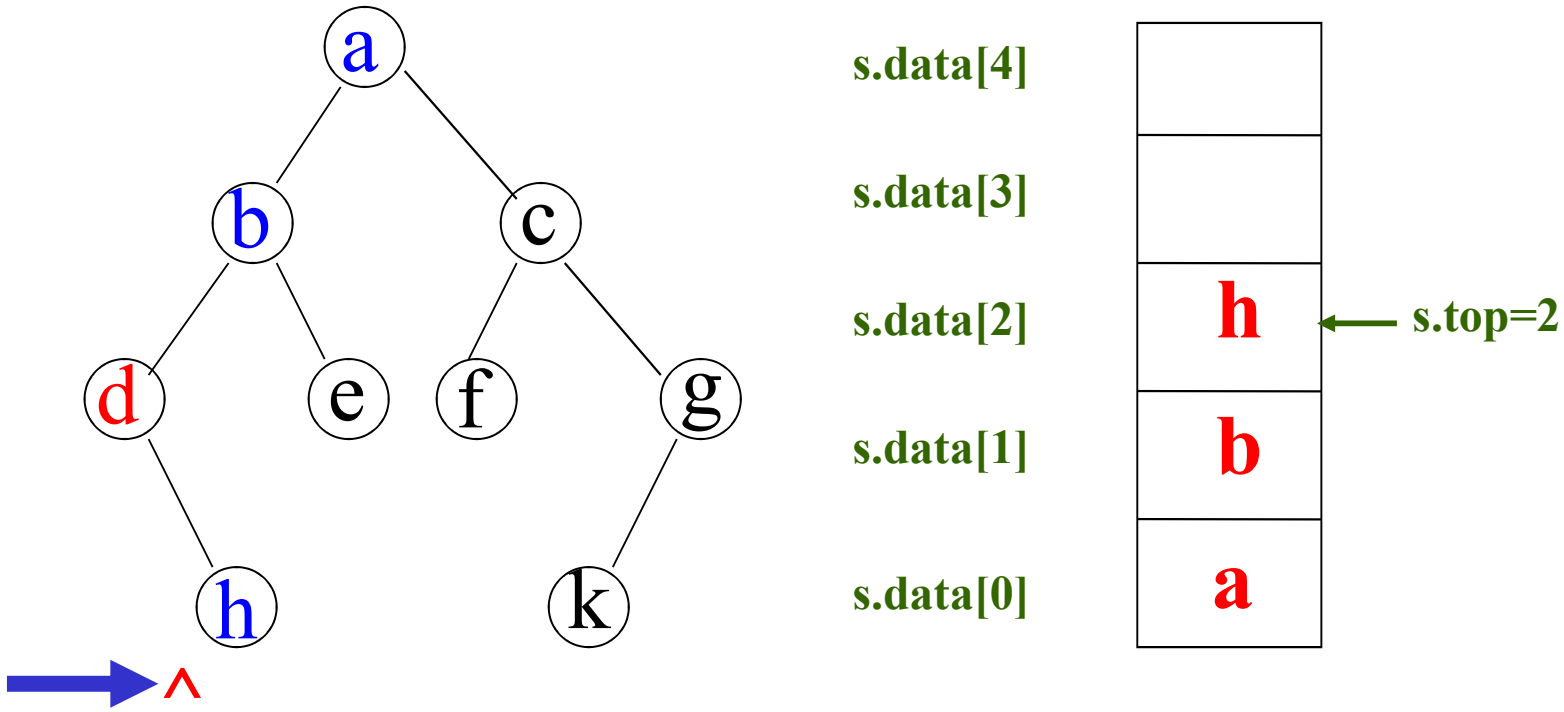




中序遍历非递归算法的实现：在访问左子树前，应保存根结点，如此左子树访问结束后，可以找到根结点，访问根结点和根的右子树

二叉树中序遍历的非递归算法实现

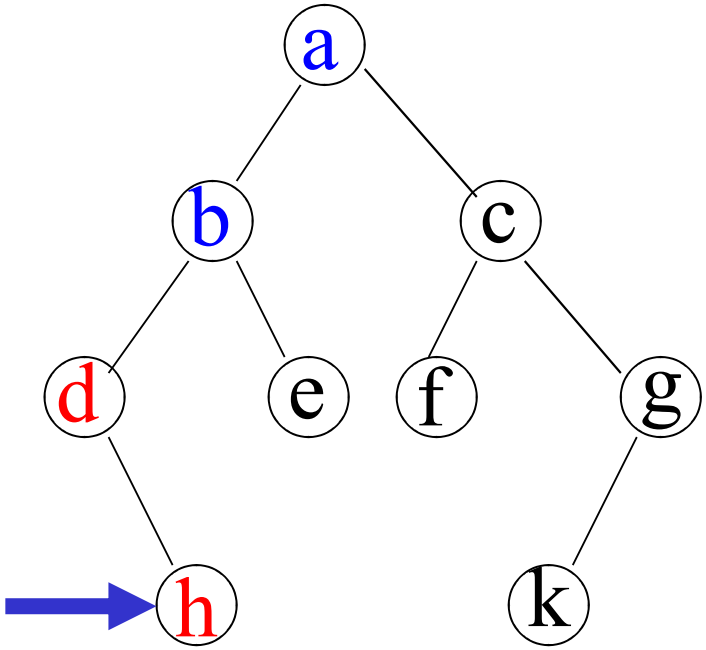
d

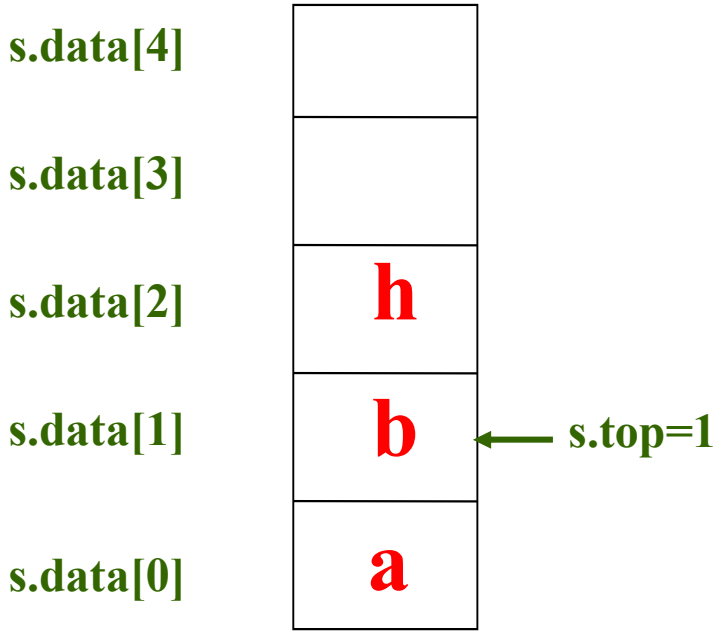


中序遍历非递归算法的实现：在访问左子树前，应保存根结点，如此左子树访问结束后，可以找到根结点，访问根结点和根的右子树

二叉树中序遍历的非递归算法实现

d

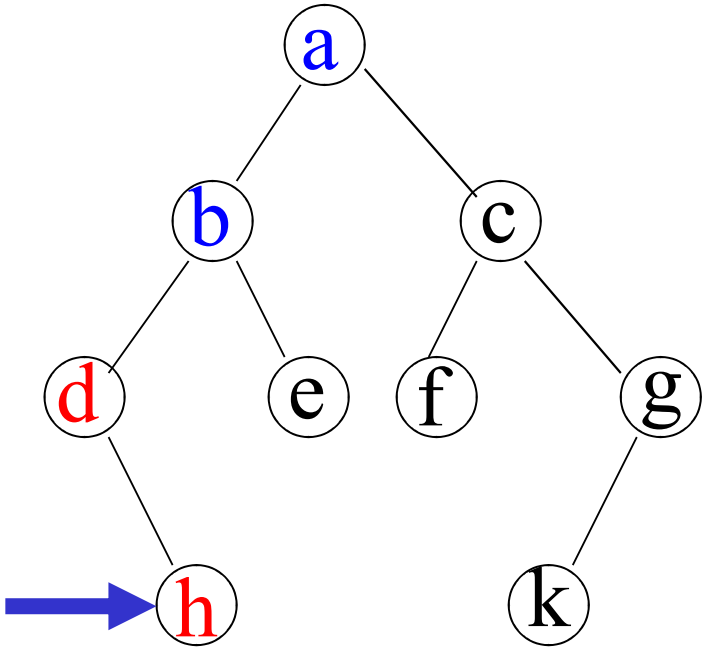


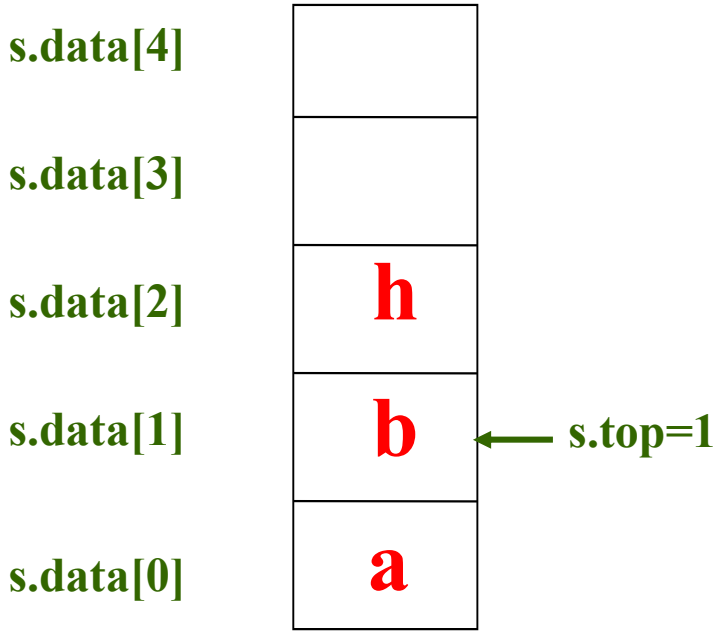


中序遍历非递归算法的实现：在访问左子树前，应保存根结点，如此左子树访问结束后，可以找到根结点，访问根结点和根的右子树

二叉树中序遍历的非递归算法实现

dh

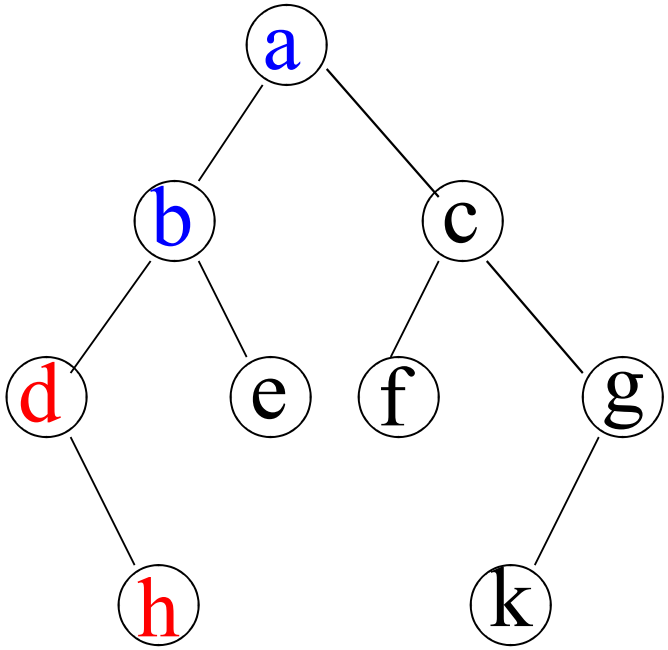


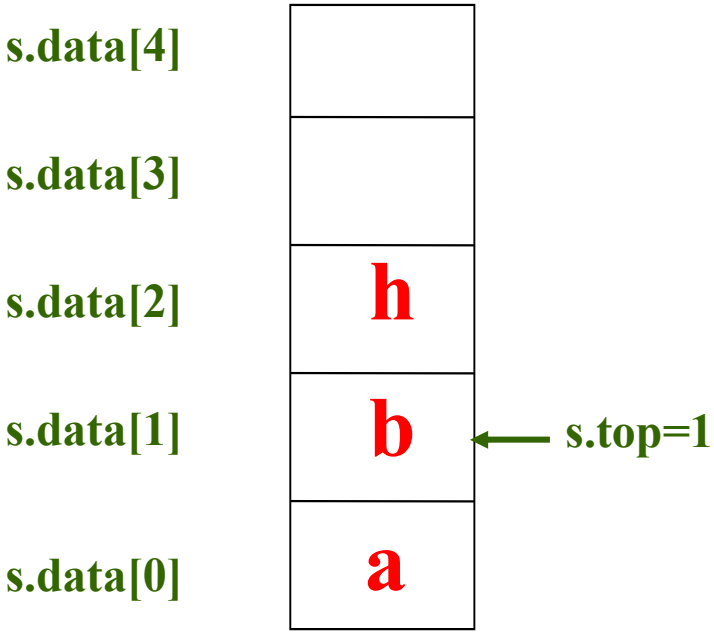


中序遍历非递归算法的实现：在访问左子树前，应保存根结点，如此左子树访问结束后，可以找到根结点，访问根结点和根的右子树

二叉树中序遍历的非递归算法实现

dh

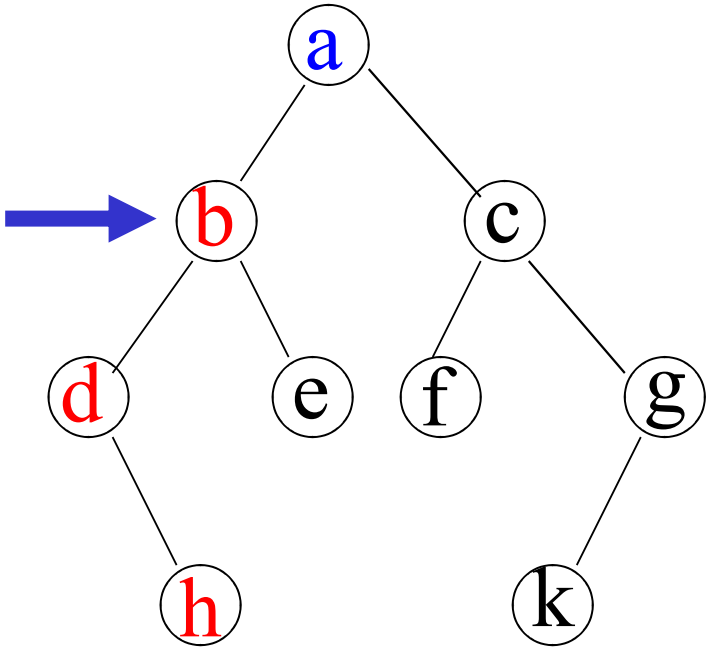


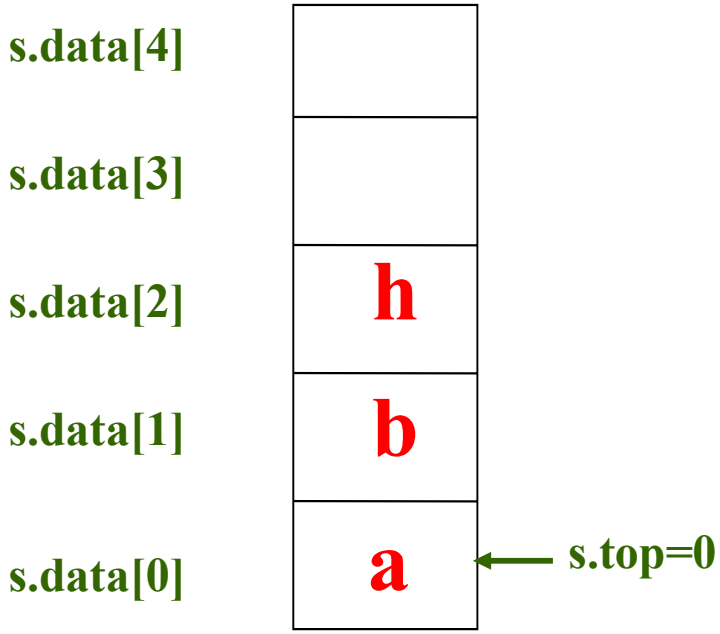


中序遍历非递归算法的实现：在访问左子树前，应保存根结点，如此左子树访问结束后，可以找到根结点，访问根结点和根的右子树

二叉树中序遍历的非递归算法实现

dh

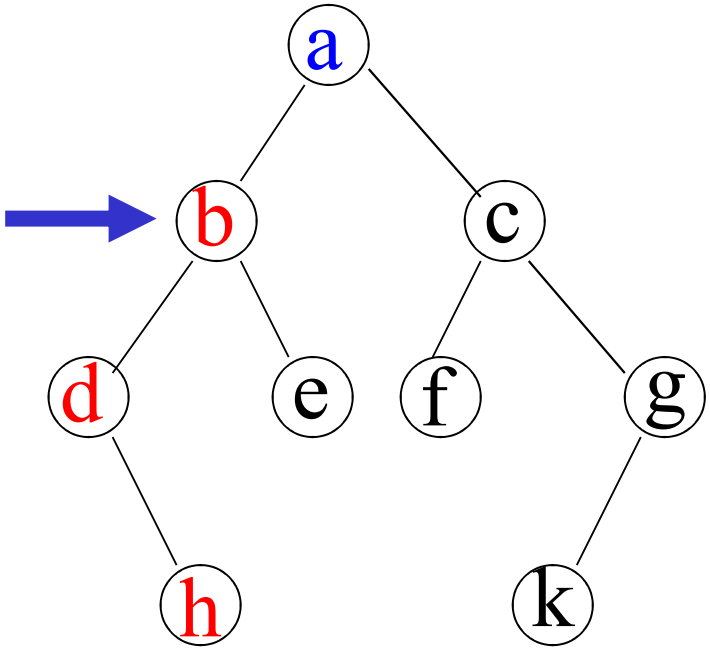


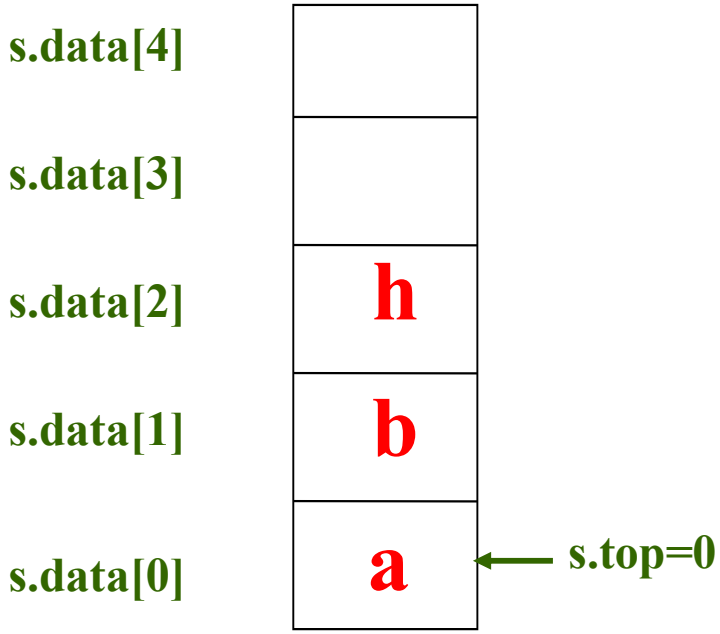


中序遍历非递归算法的实现：在访问左子树前，应保存根结点，如此左子树访问结束后，可以找到根结点，访问根结点和根的右子树

二叉树中序遍历的非递归算法实现

dhb

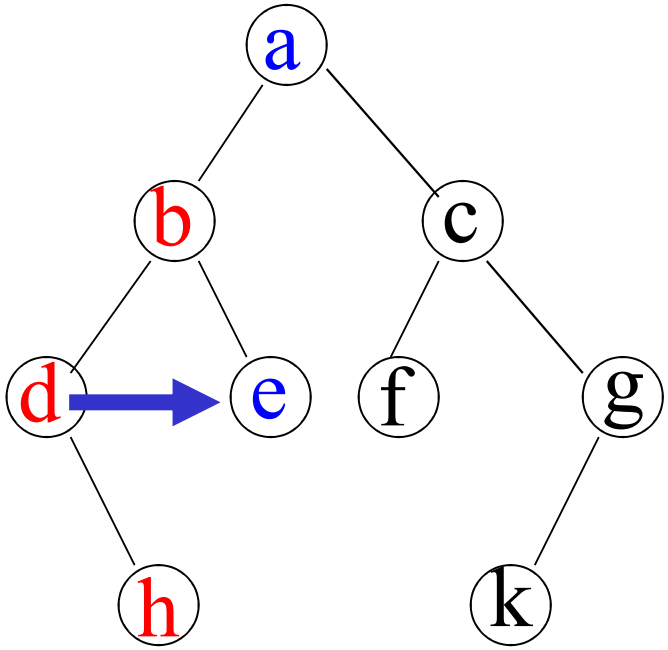


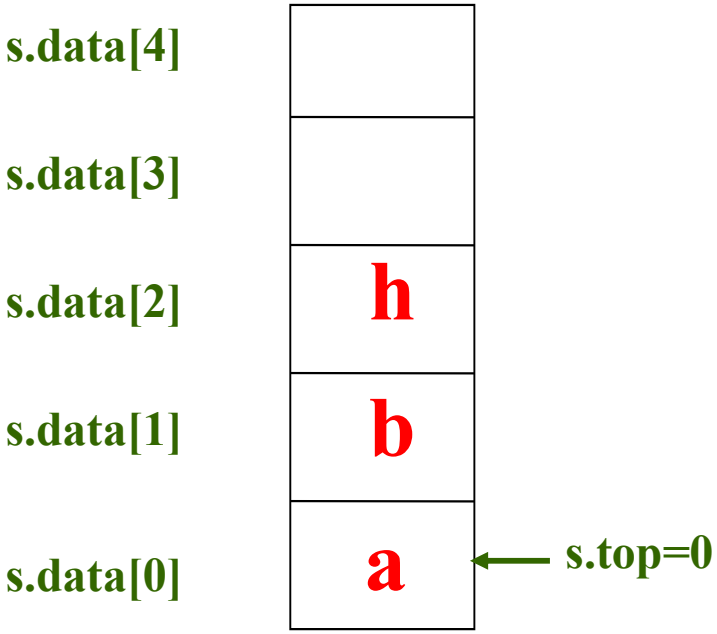


中序遍历非递归算法的实现：在访问左子树前，应保存根结点，如此左子树访问结束后，可以找到根结点，访问根结点和根的右子树

二叉树中序遍历的非递归算法实现

dhb

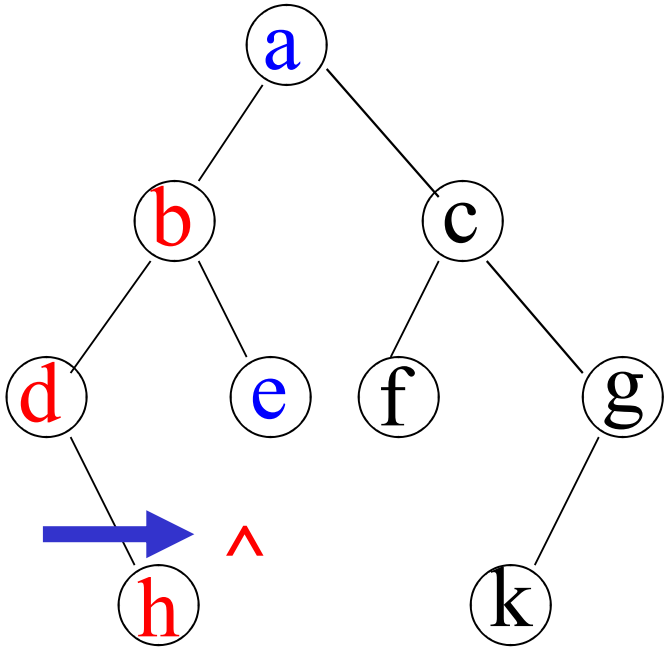


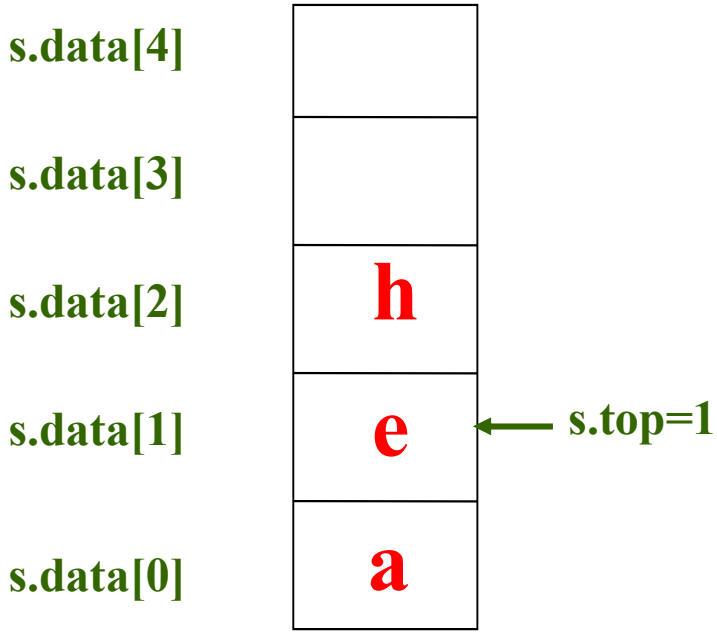


中序遍历非递归算法的实现：在访问左子树前，应保存根结点，如此左子树访问结束后，可以找到根结点，访问根结点和根的右子树

二叉树中序遍历的非递归算法实现

dhb

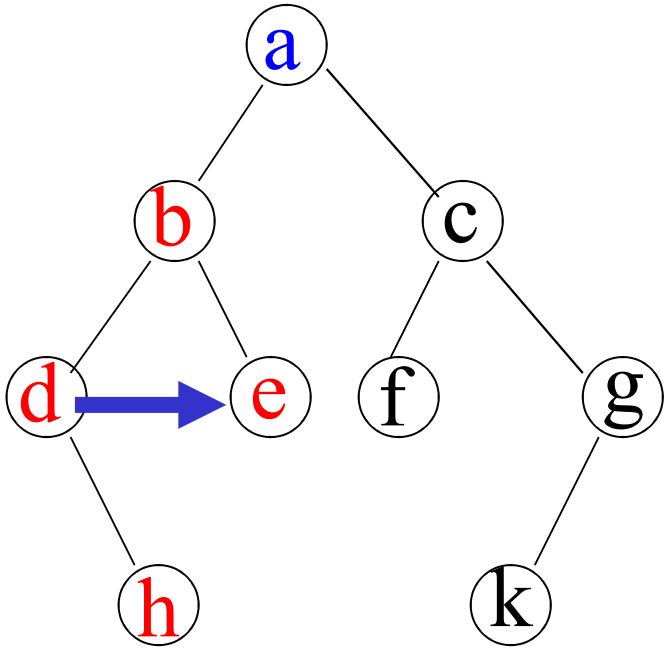


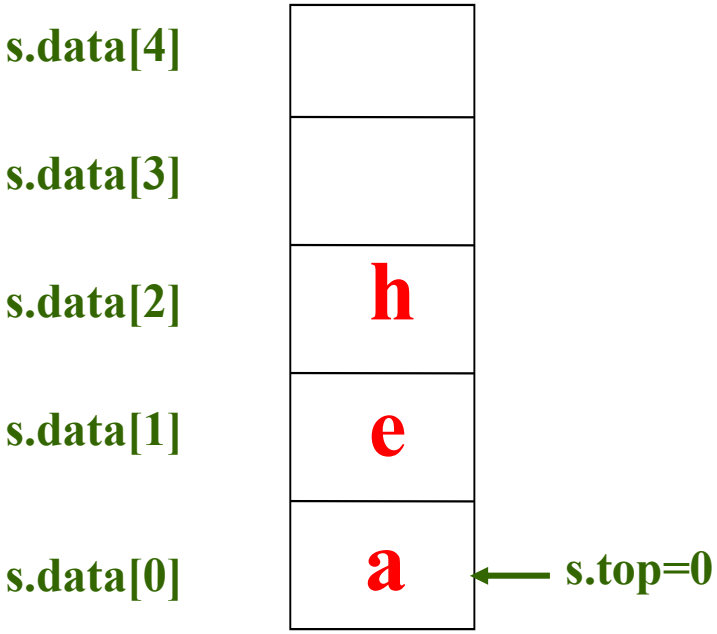


中序遍历非递归算法的实现：在访问左子树前，应保存根结点，如此左子树访问结束后，可以找到根结点，访问根结点和根的右子树

二叉树中序遍历的非递归算法实现

dhb

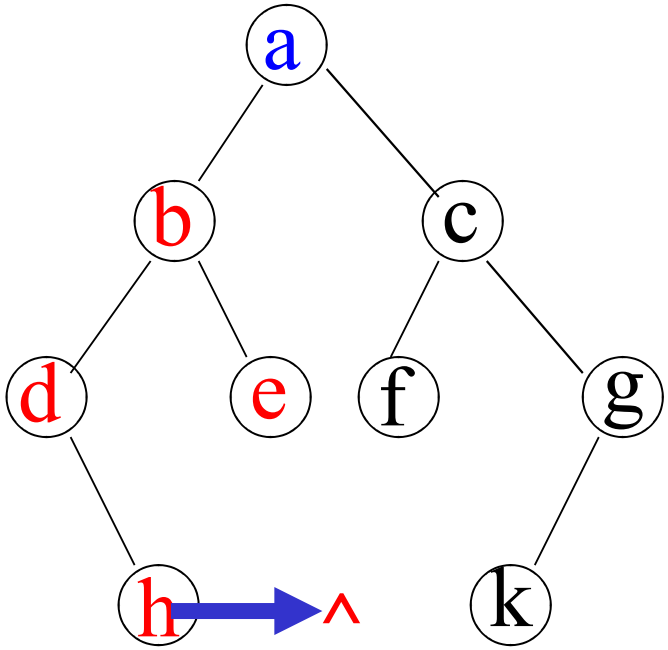


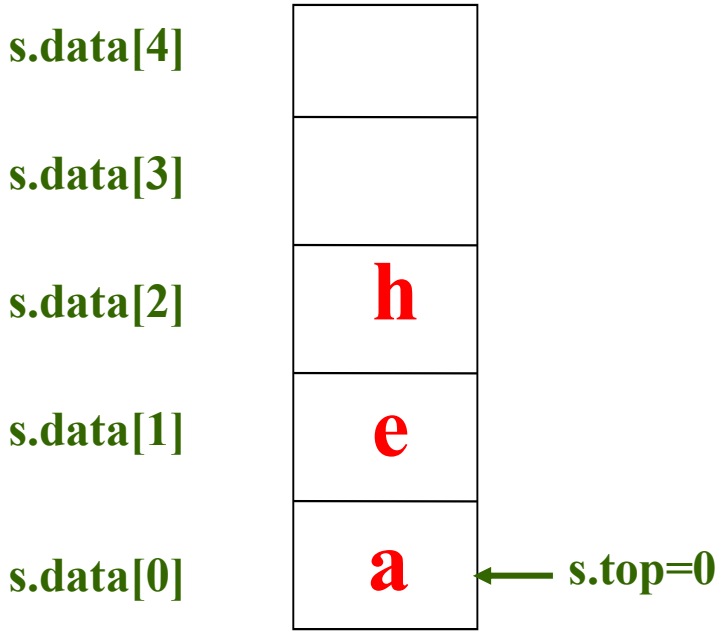


中序遍历非递归算法的实现：在访问左子树前，应保存根结点，如此左子树访问结束后，可以找到根结点，访问根结点和根的右子树

二叉树中序遍历的非递归算法实现

dhbe

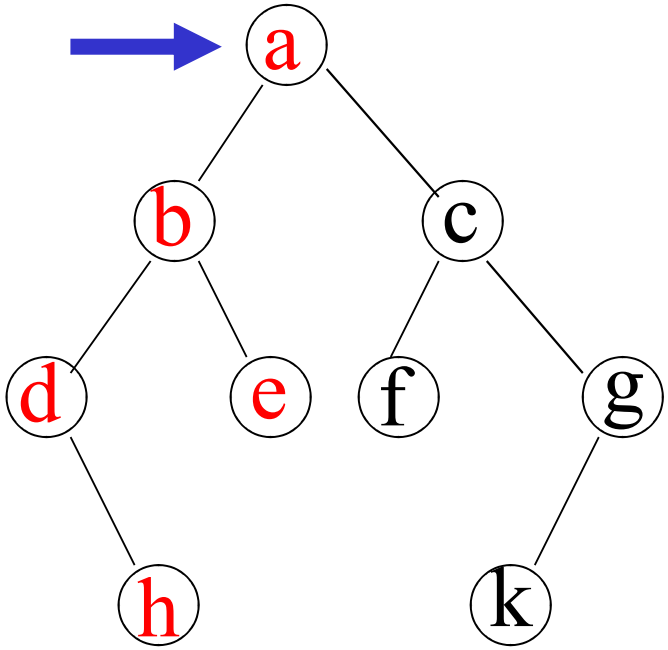


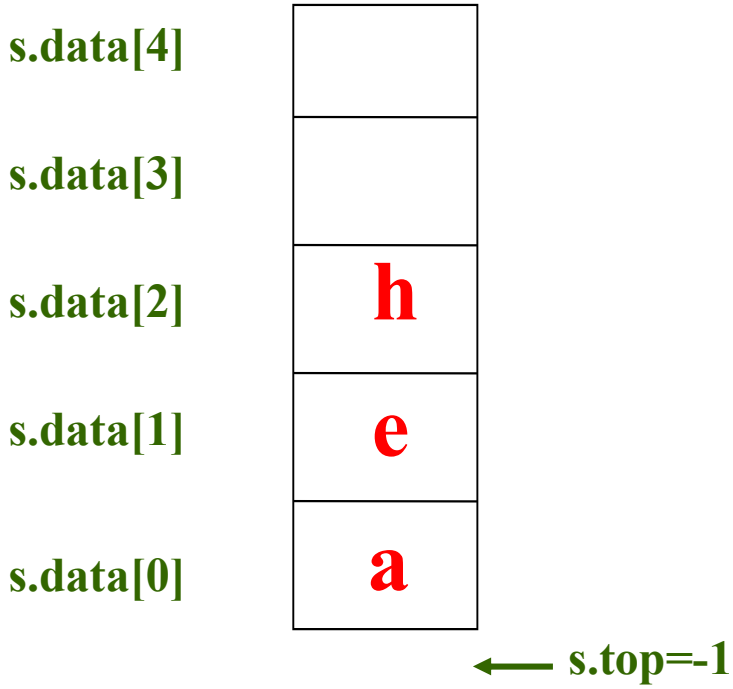


中序遍历非递归算法的实现：在访问左子树前，应保存根结点，如此左子树访问结束后，可以找到根结点，访问根结点和根的右子树

二叉树中序遍历的非递归算法实现

dhbe

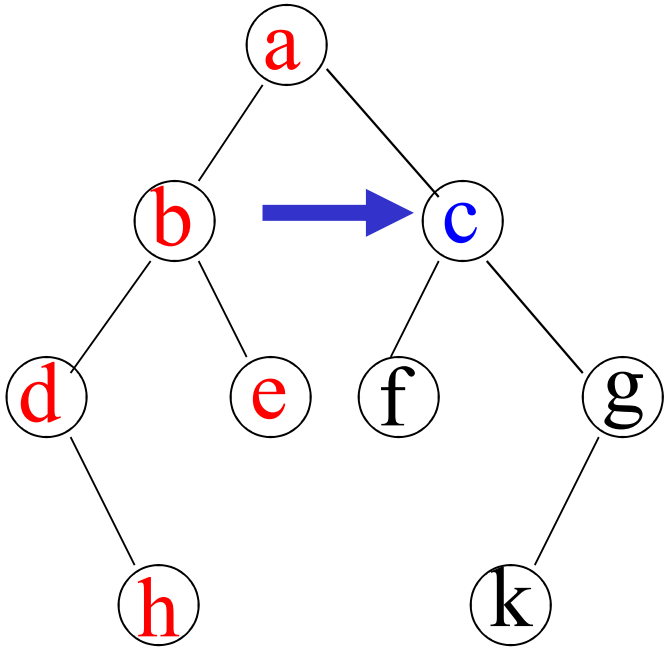


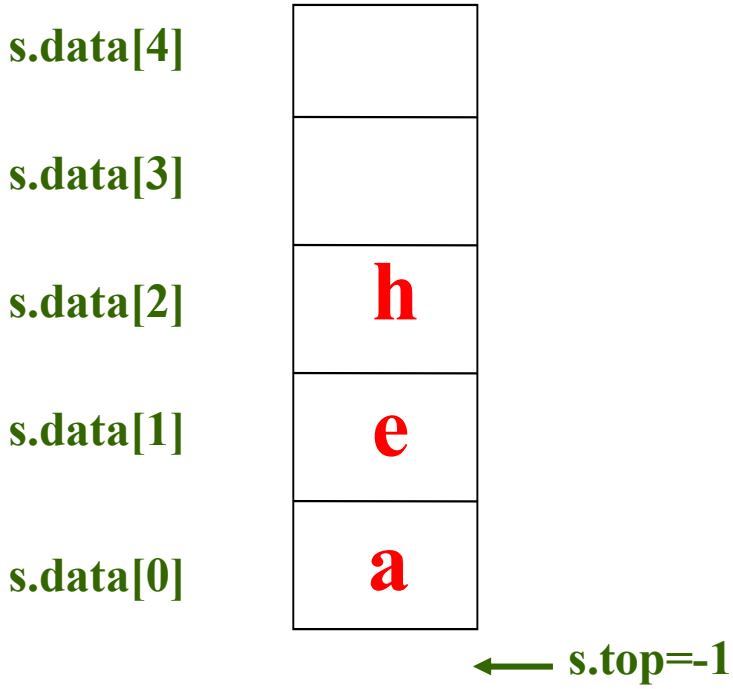


中序遍历非递归算法的实现：在访问左子树前，应保存根结点，如此左子树访问结束后，可以找到根结点，访问根结点和根的右子树

二叉树中序遍历的非递归算法实现

dhbea

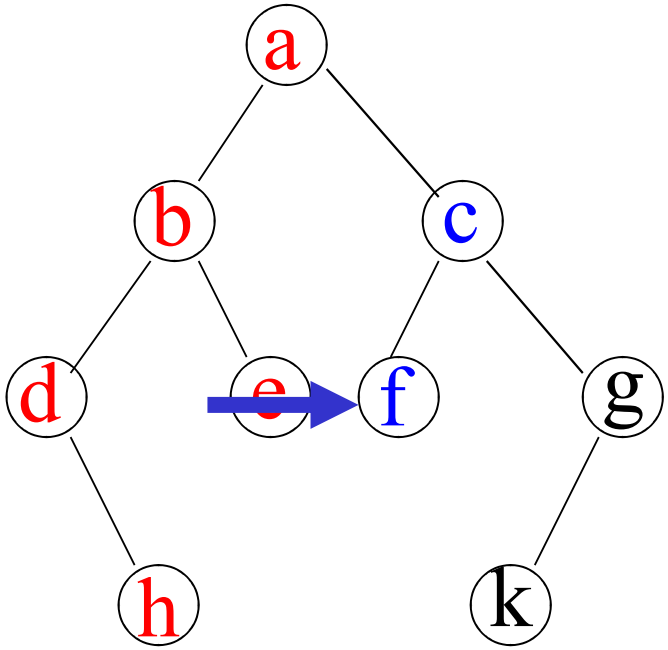


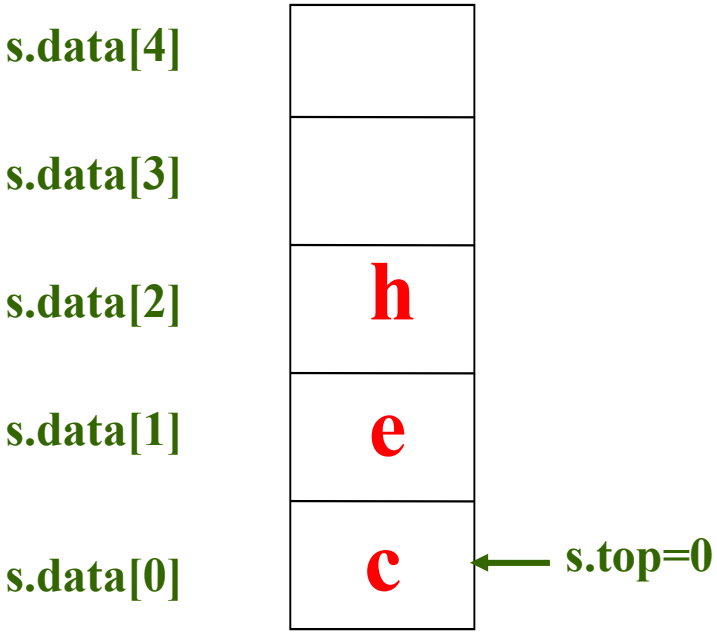


中序遍历非递归算法的实现：在访问左子树前，应保存根结点，如此左子树访问结束后，可以找到根结点，访问根结点和根的右子树

二叉树中序遍历的非递归算法实现

dhbea

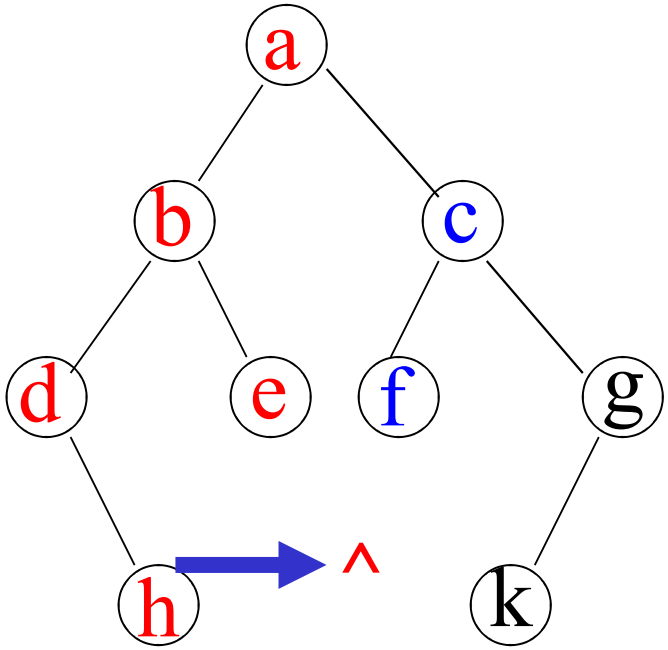


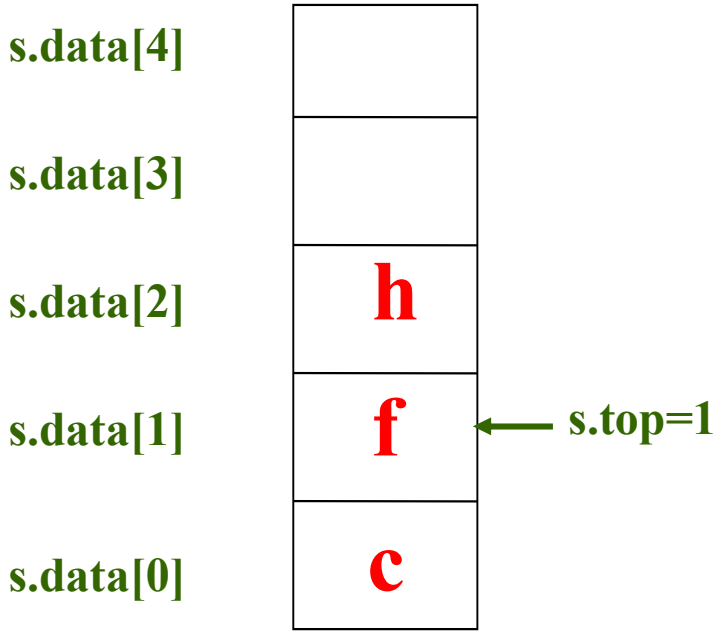


中序遍历非递归算法的实现：在访问左子树前，应保存根结点，如此左子树访问结束后，可以找到根结点，访问根结点和根的右子树

二叉树中序遍历的非递归算法实现

dhbea

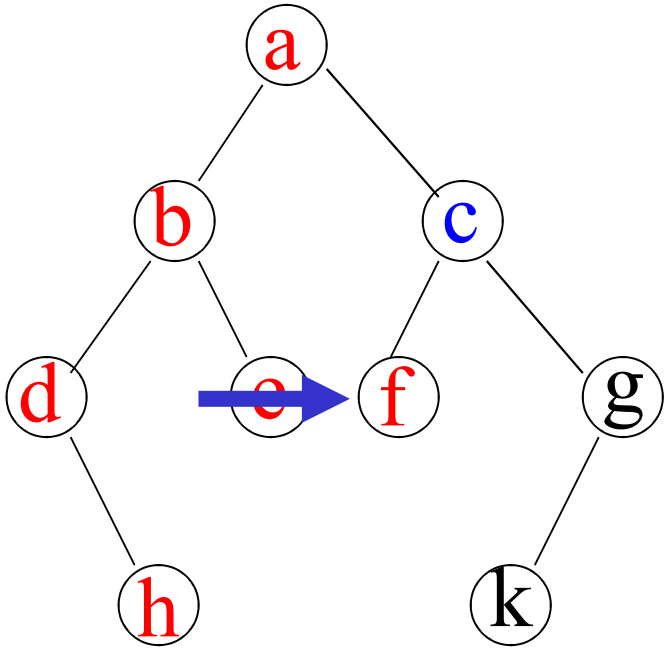


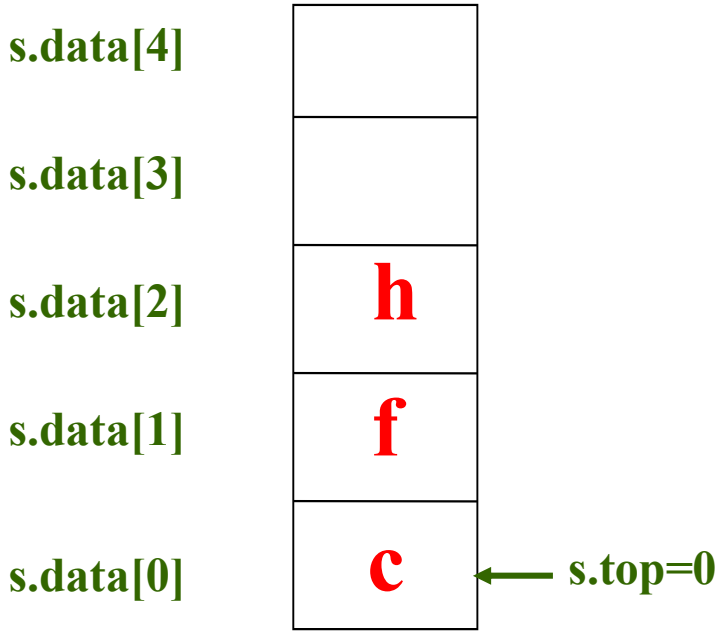


中序遍历非递归算法的实现：在访问左子树前，应保存根结点，如此左子树访问结束后，可以找到根结点，访问根结点和根的右子树

二叉树中序遍历的非递归算法实现

dhbea

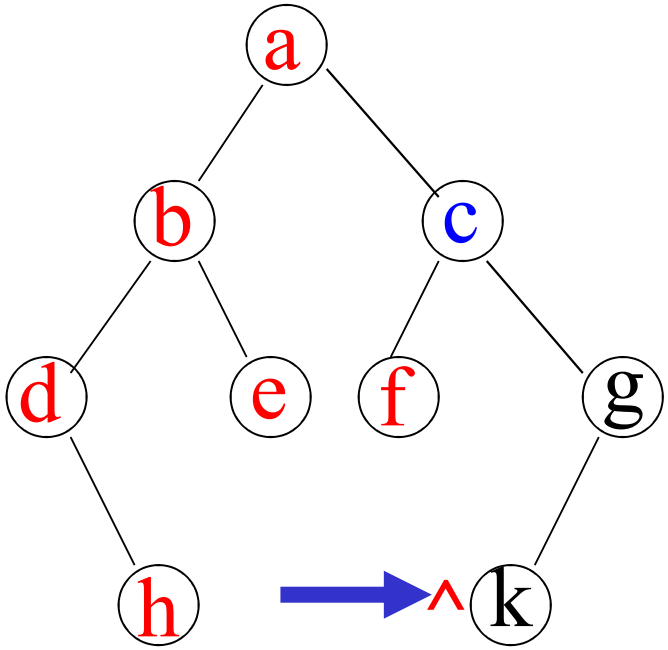


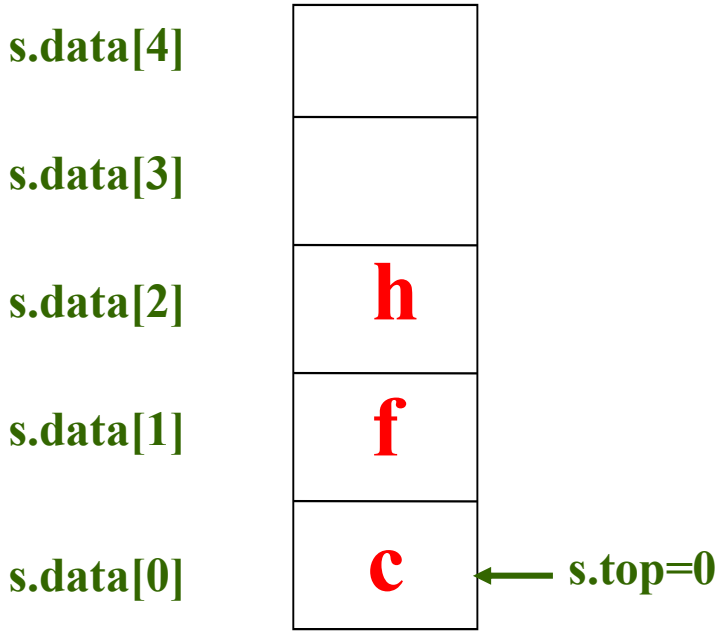


中序遍历非递归算法的实现：在访问左子树前，应保存根结点，如此左子树访问结束后，可以找到根结点，访问根结点和根的右子树

二叉树中序遍历的非递归算法实现

dhbeaf

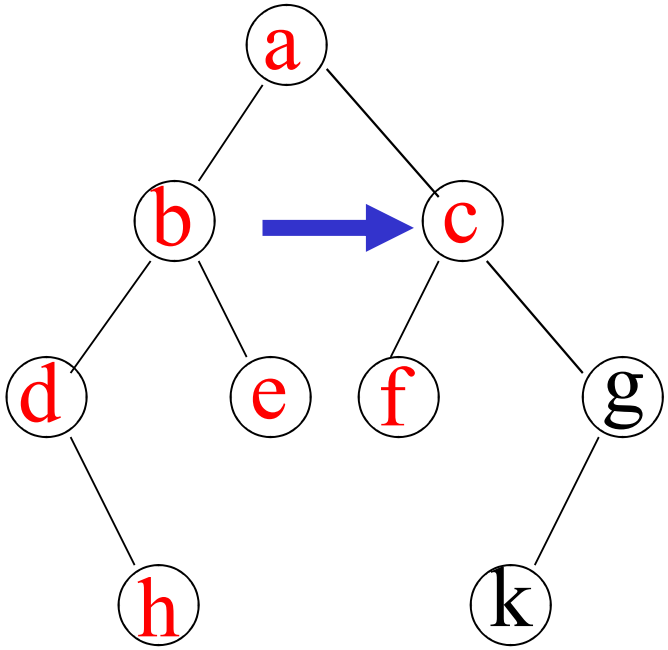


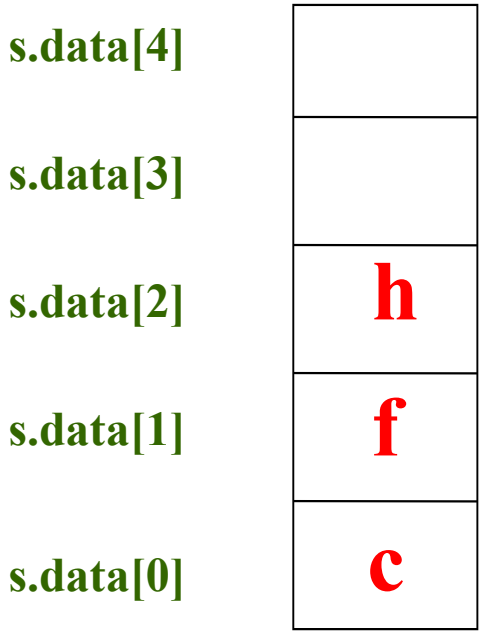


中序遍历非递归算法的实现：在访问左子树前，应保存根结点，如此左子树访问结束后，可以找到根结点，访问根结点和根的右子树

二叉树中序遍历的非递归算法实现

dhbeaf



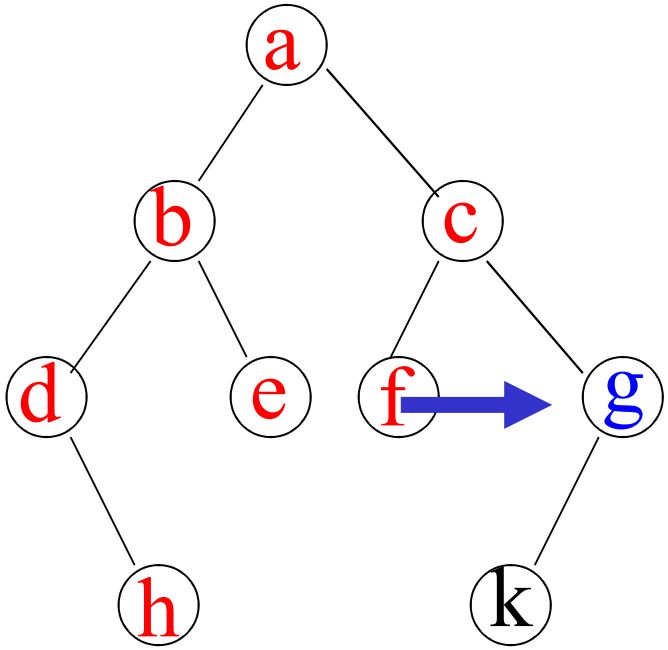


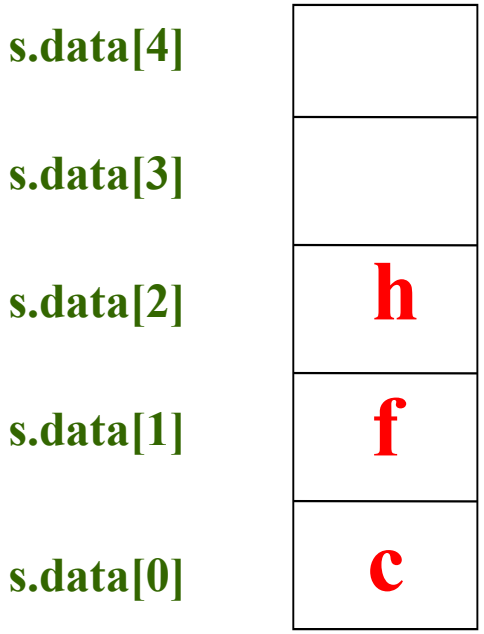
s.top=-1

中序遍历非递归算法的实现：在访问左子树前，应保存根结点，如此左子树访问结束后，可以找到根结点，访问根结点和根的右子树

二叉树中序遍历的非递归算法实现

dhbeafc



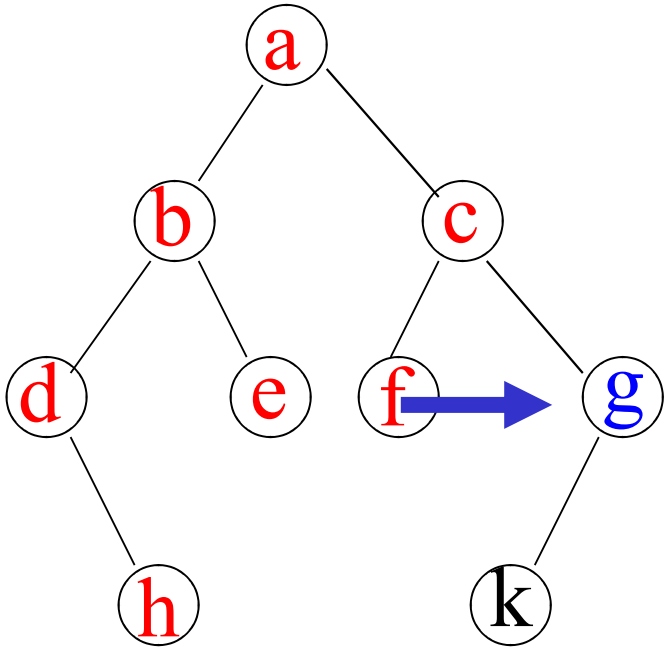


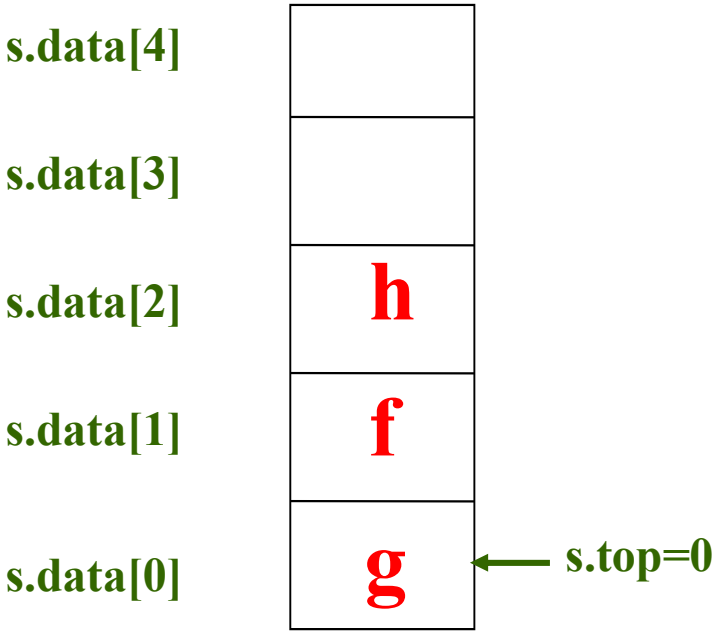
s.top=-1

中序遍历非递归算法的实现：在访问左子树前，应保存根结点，如此左子树访问结束后，可以找到根结点，访问根结点和根的右子树

二叉树中序遍历的非递归算法实现

dhbeafc

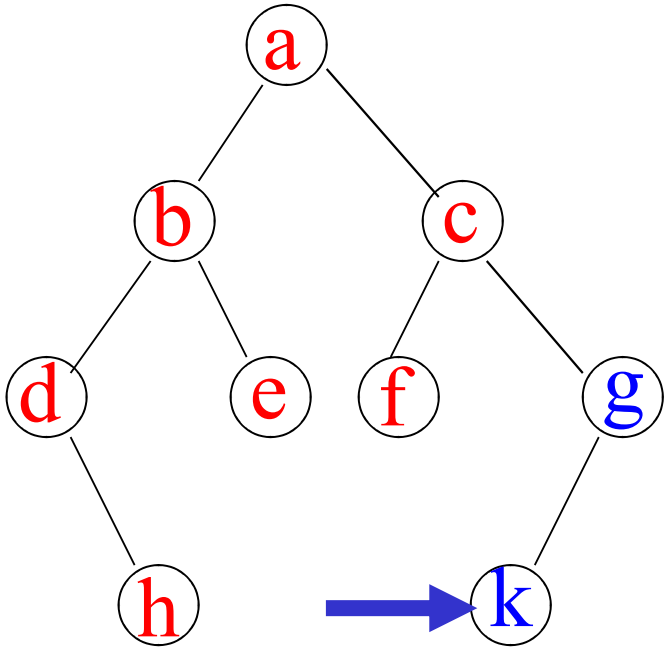


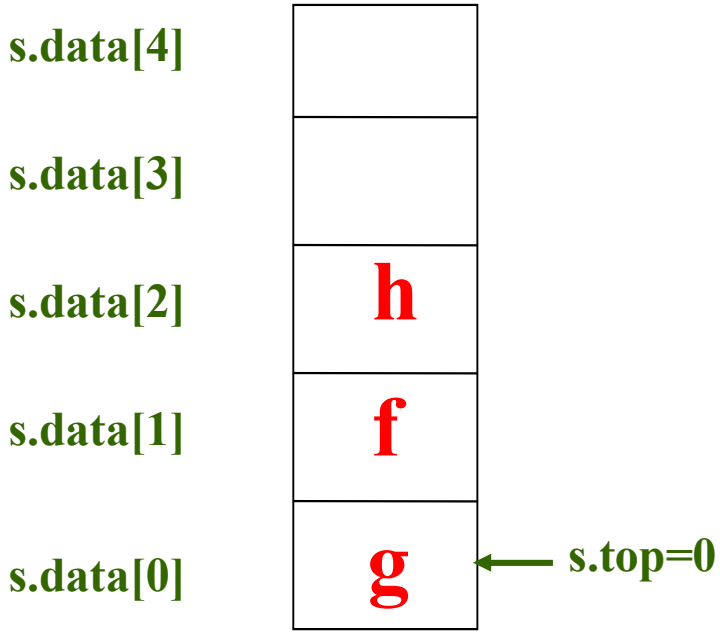


中序遍历非递归算法的实现：在访问左子树前，应保存根结点，如此左子树访问结束后，可以找到根结点，访问根结点和根的右子树

二叉树中序遍历的非递归算法实现

dhbeafc

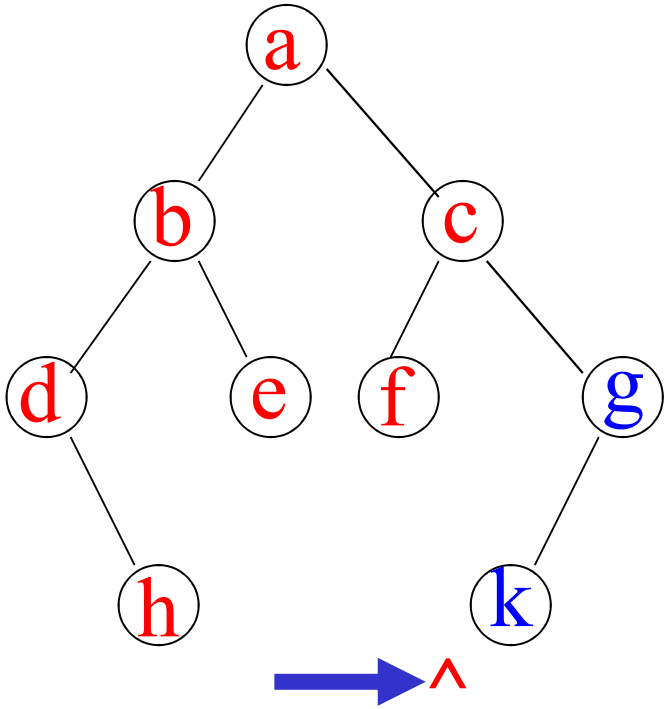


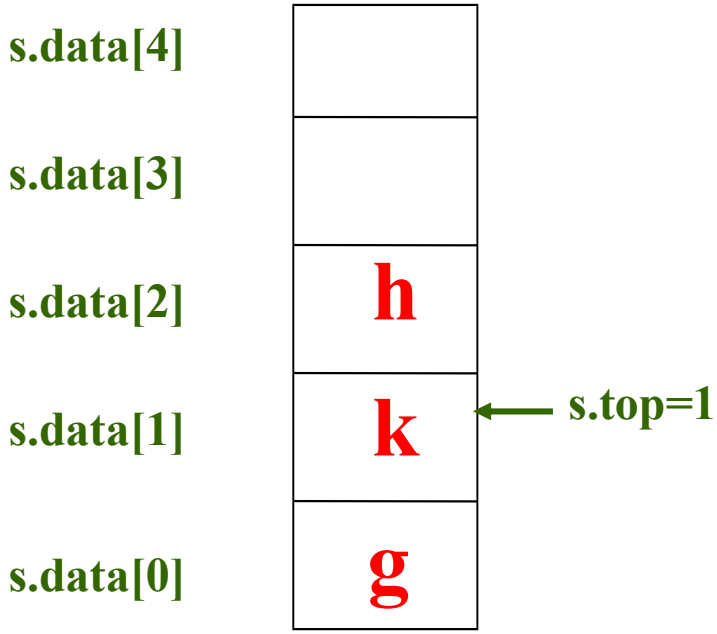


中序遍历非递归算法的实现：在访问左子树前，应保存根结点，如此左子树访问结束后，可以找到根结点，访问根结点和根的右子树

二叉树中序遍历的非递归算法实现

dhbeafc

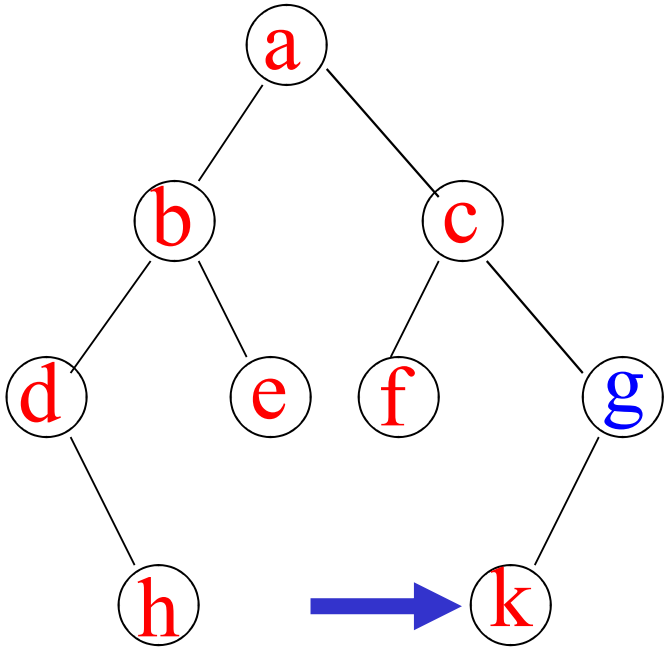


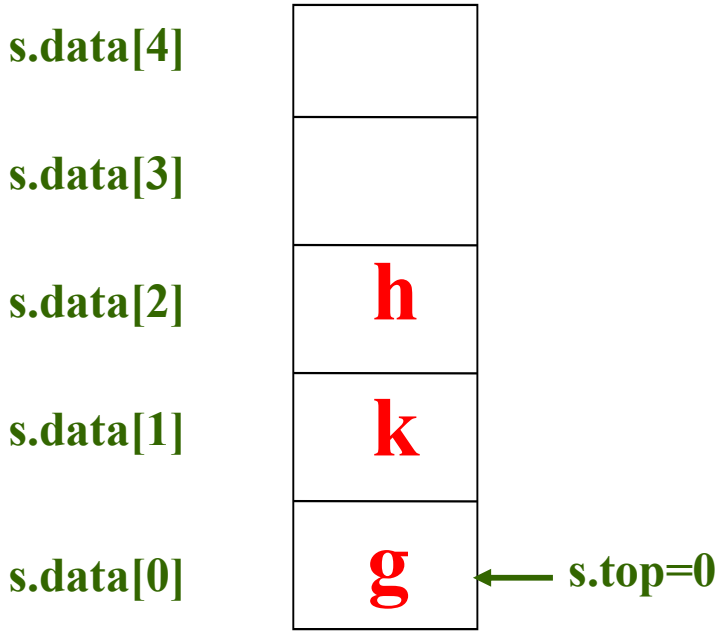


中序遍历非递归算法的实现：在访问左子树前，应保存根结点，如此左子树访问结束后，可以找到根结点，访问根结点和根的右子树

二叉树中序遍历的非递归算法实现

dhbeafc

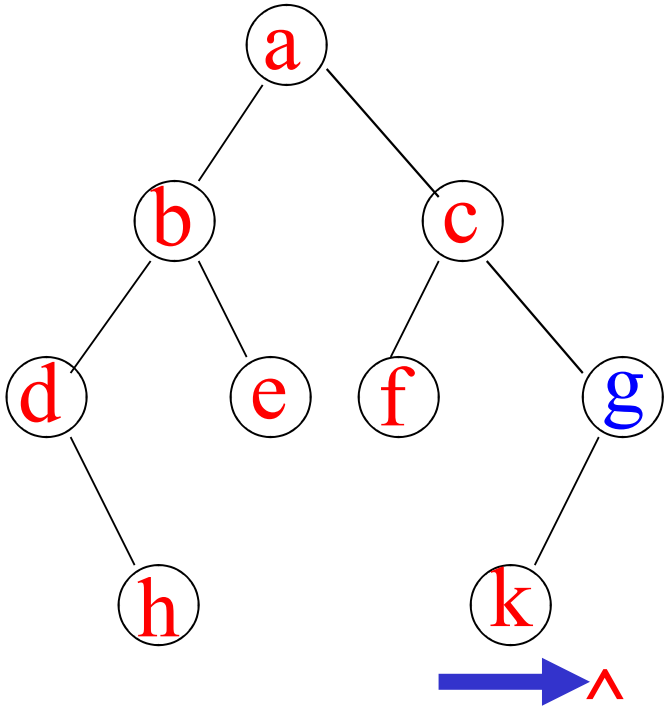


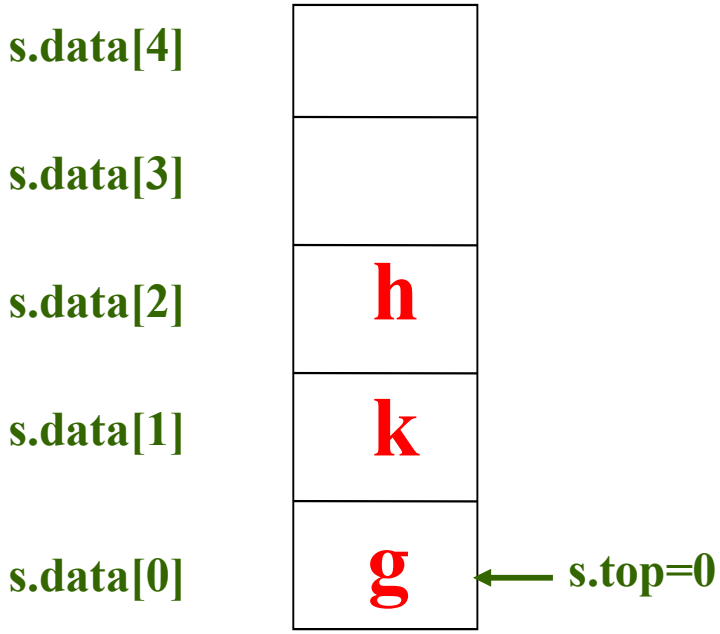


中序遍历非递归算法的实现：在访问左子树前，应保存根结点，如此左子树访问结束后，可以找到根结点，访问根结点和根的右子树

二叉树中序遍历的非递归算法实现

dhbeafck

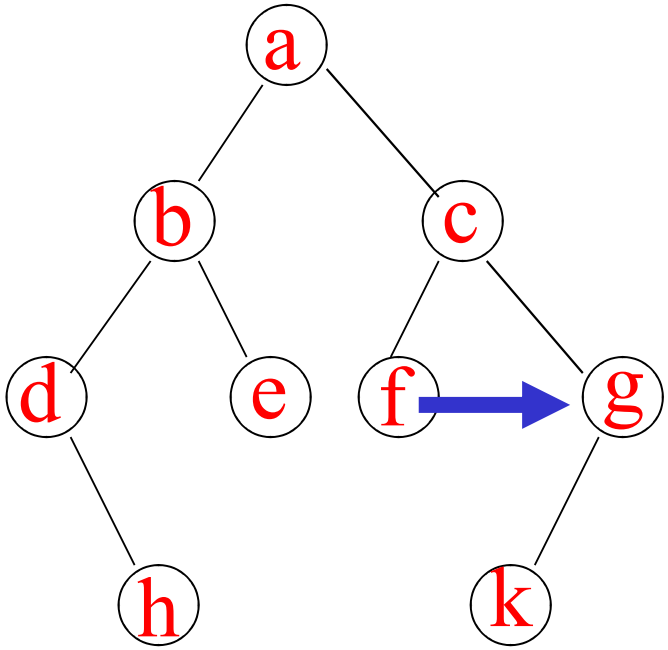


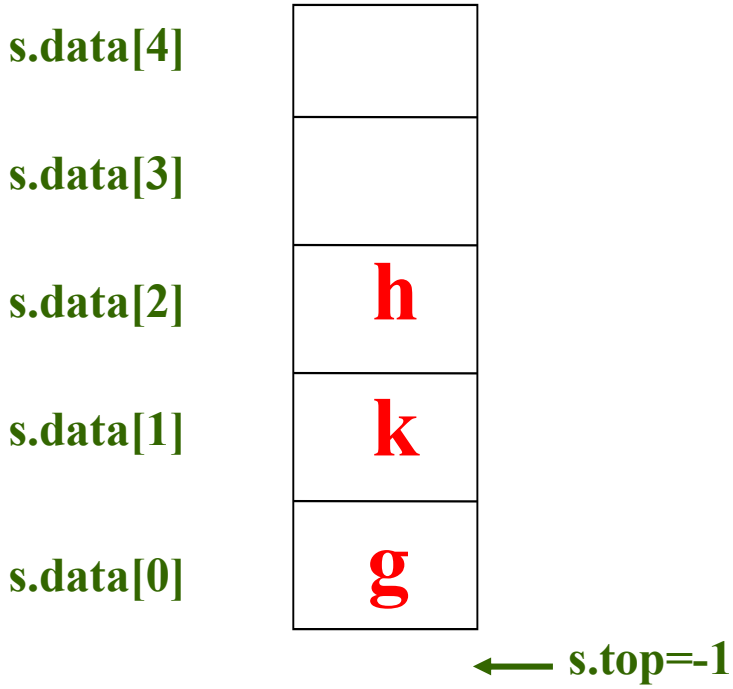


中序遍历非递归算法的实现：在访问左子树前，应保存根结点，如此左子树访问结束后，可以找到根结点，访问根结点和根的右子树

二叉树中序遍历的非递归算法实现

dhbeafck

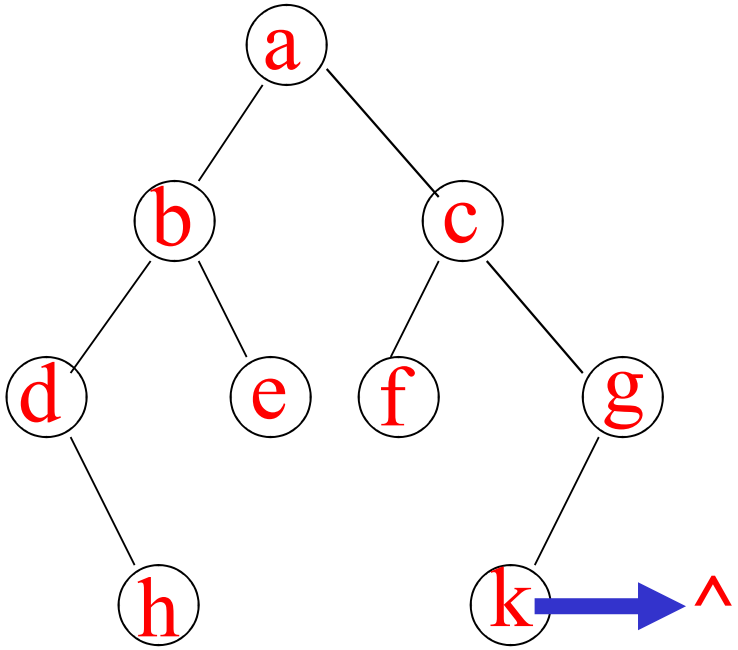


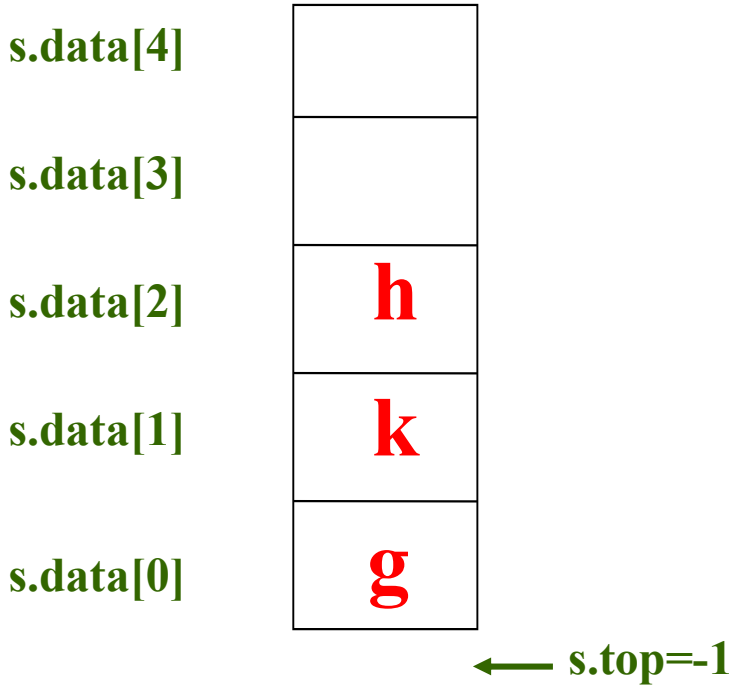


中序遍历非递归算法的实现：在访问左子树前，应保存根结点，如此左子树访问结束后，可以找到根结点，访问根结点和根的右子树

dhbeafckg

当前访问的结点为空指针，并且栈空----算法结束





中序遍历非递归算法的实现：在访问左子树前，应保存根结点，如此左子树访问结束后，可以找到根结点，访问根结点和根的右子树

中序遍历算法的非递归描述

```
void InorderTraverse(BiTree T){
  SeqStack  s ;
  s.top=-1; p = T;
  while(p||(s.top!=-1)){
    while(p){  if(s.top==MAX-1) exit (0);
               s.data[++s.top]=p; //p结点入栈
               p =p->lc; //进入p的左子树
             }
    if (s.top!=-1)
        {p=s.data[s.top--];
         printf("%c",p->data); //访问p结点
         p = p->rc;   }
  }
}
```

后（根）序的遍历算法：

若二叉树为空树，则空操作；否则，

（1）后序遍历左子树；

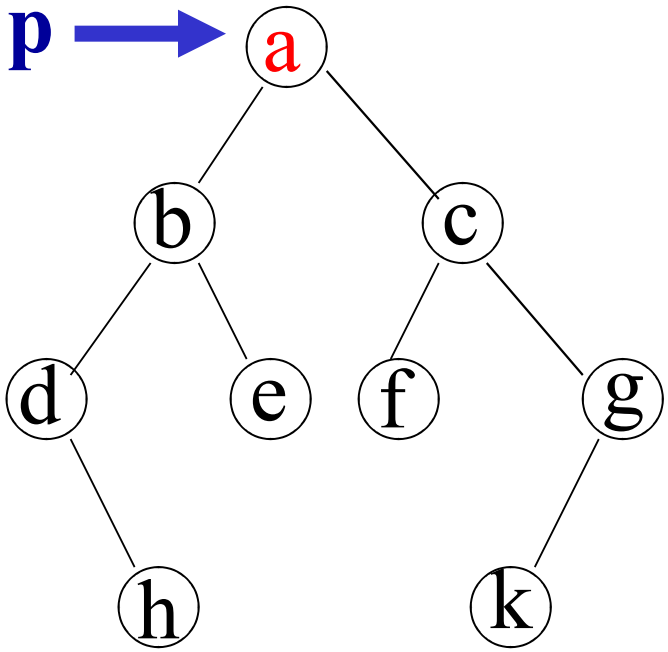
（2）后序遍历右子树；

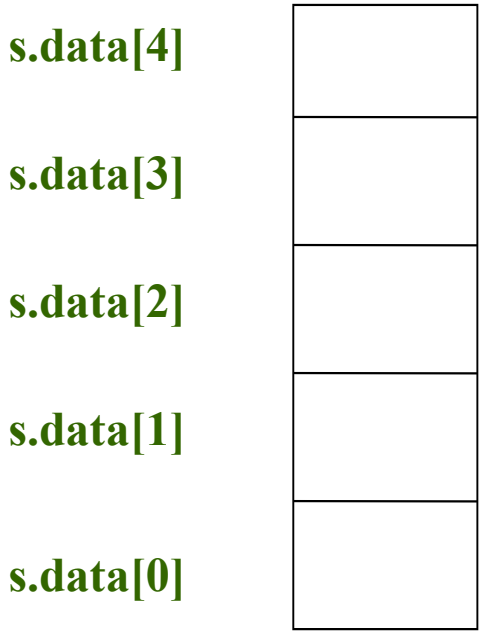
（3）访问根结点。

后序遍历非递归算法的实现：访问根结点的左子树前，应保存其根结点，以便左子树访问结束后，访问根的右子树和根

图中a结点先于b结点被保存，但是其访问要在b及其右子树被访问后进行----先保存后访问----先进后出----借助栈实现

图中结点只有在其左、右子树被访问后才能被访问





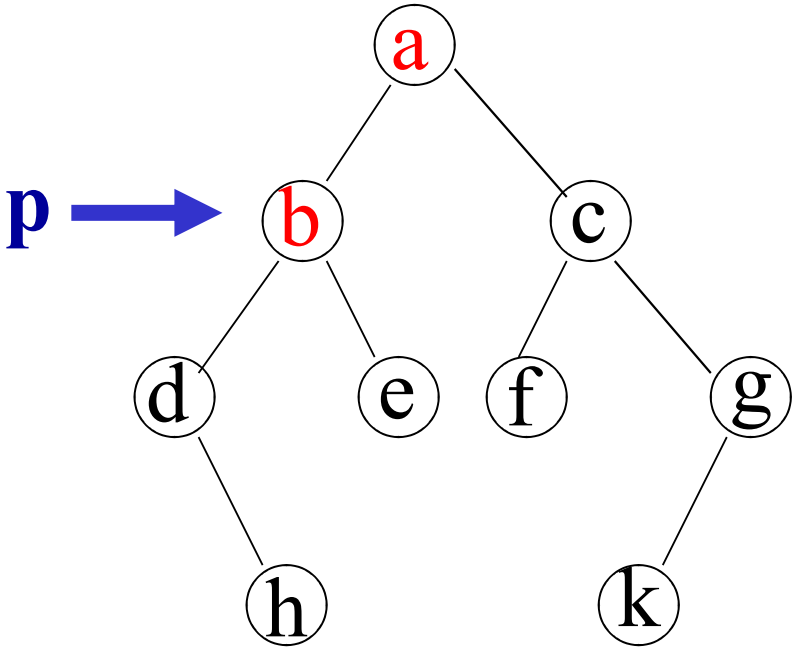
s.top=-1

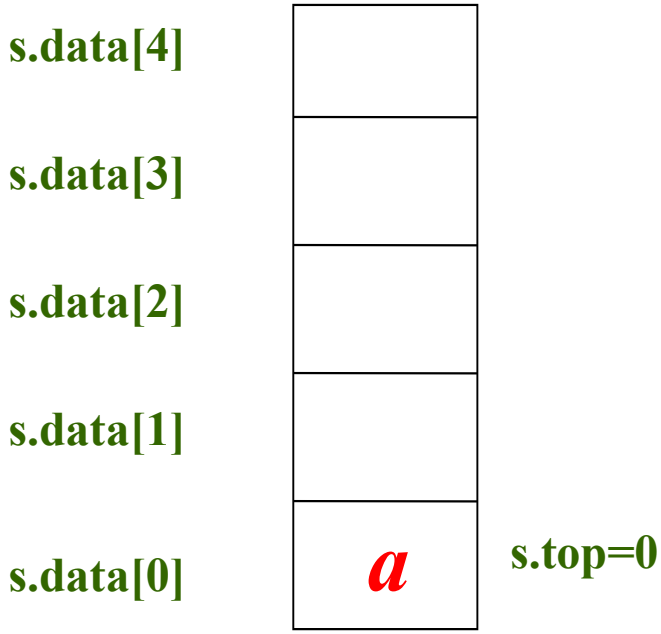
二叉树后序遍历的非递归算法实现

后序遍历非递归算法的实现：访问根结点的左子树前，应保存其根结点，以便左子树访问结束后，访问根的右子树和根

图中a结点先于b结点被保存，但是其访问要在b及其右子树被访问后进行----先保存后访问----先进后出----借助栈实现

图中结点只有在其左、右子树被访问后才能被访问



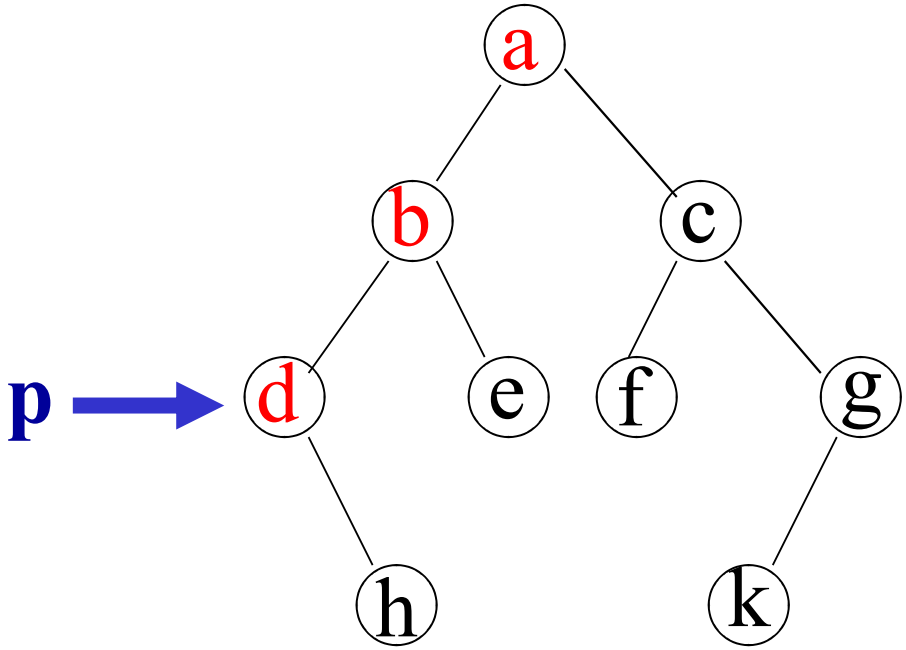


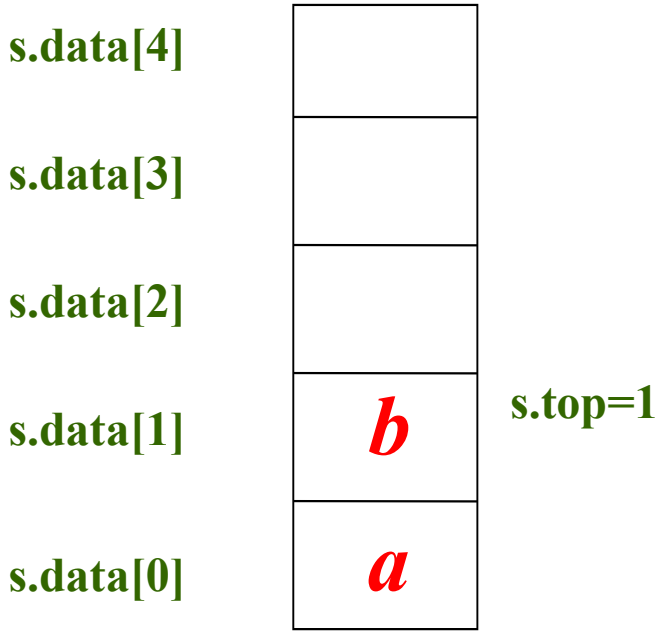
二叉树后序遍历的非递归算法实现

后序遍历非递归算法的实现：访问根结点的左子树前，应保存其根结点，以便左子树访问结束后，访问根的右子树和根

图中a结点先于b结点被保存，但是其访问要在b及其右子树被访问后进行----先保存后访问----先进后出----借助栈实现

图中结点只有在其左、右子树被访问后才能被访问



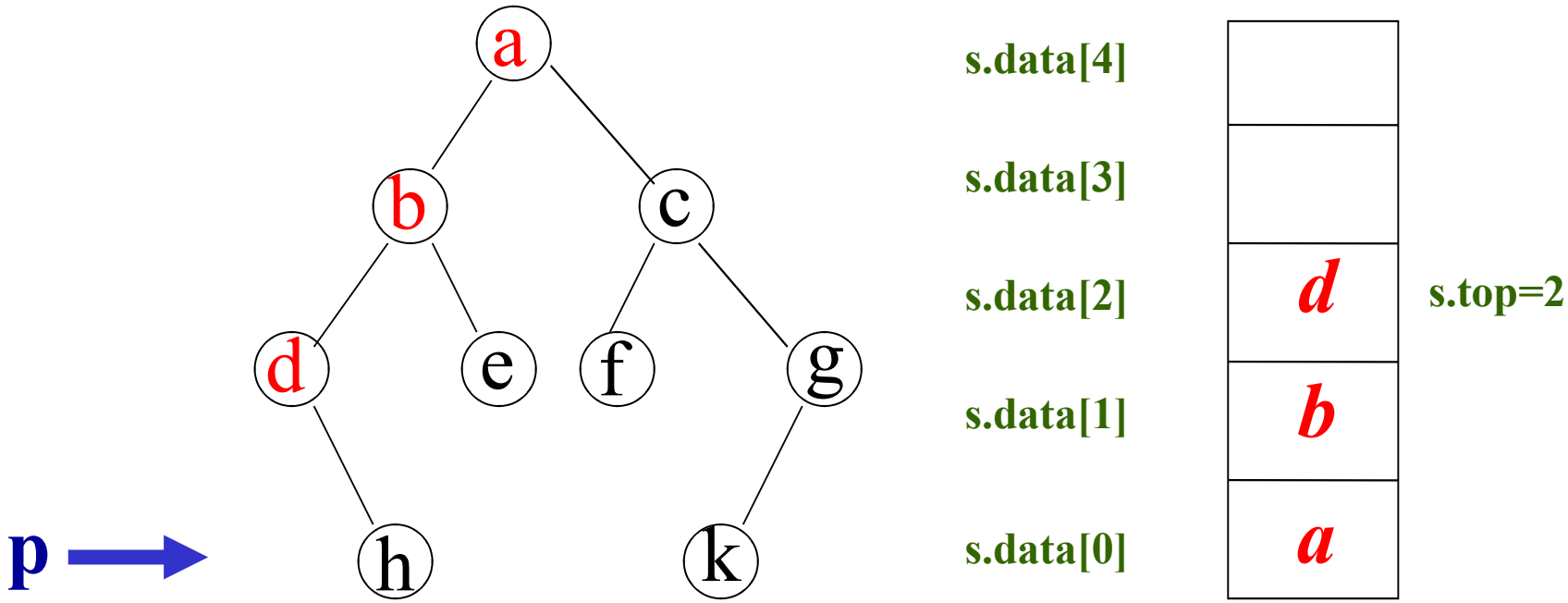


二叉树后序遍历的非递归算法实现

后序遍历非递归算法的实现：访问根结点的左子树前，应保存其根结点，以便左子树访问结束后，访问根的右子树和根

图中a结点先于b结点被保存，但是其访问要在b及其右子树被访问后进行----先保存后访问----先进后出----借助栈实现

图中结点只有在其左、右子树被访问后才能被访问



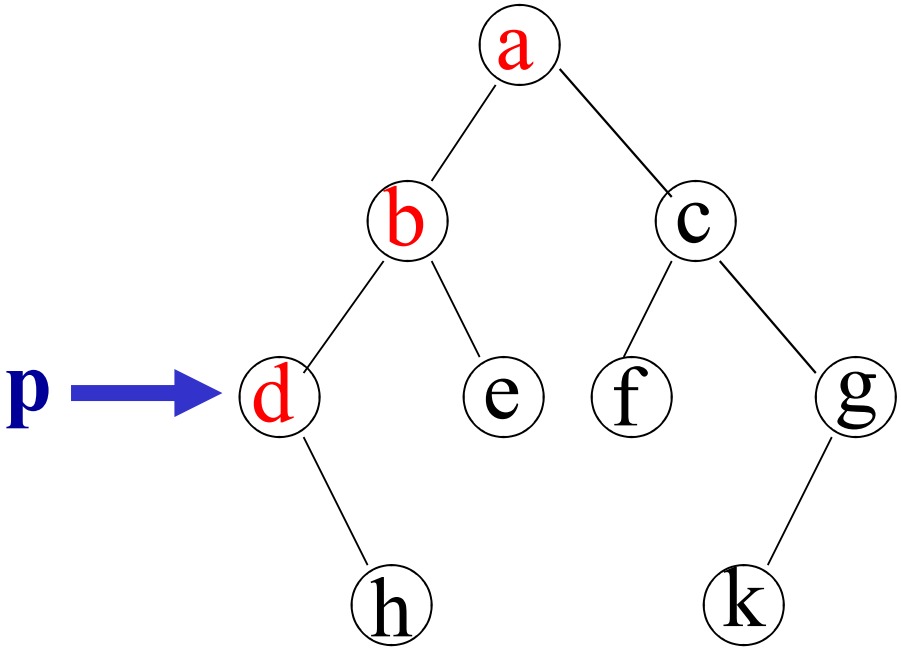
二叉树后序遍历的非递归算法实现

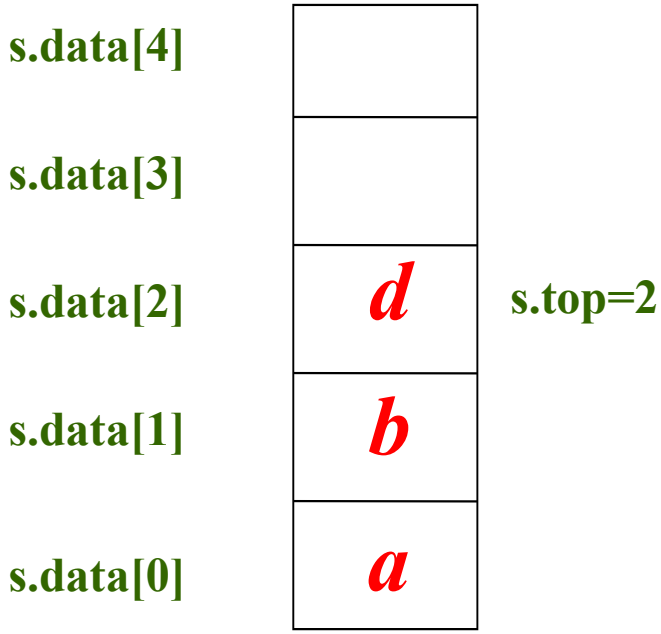
后序遍历非递归算法的实现：访问根结点的左子树前，应保存其根结点，以便左子树访问结束后，访问根的右子树和根

图中a结点先于b结点被保存，但是其访问要在b及其右子树被访问后进行----先保存后访问----先进后出----借助栈实现

图中结点只有在其左、右子树被访问后才能被访问

图中一个结点的左子树访问结束，回到该结点----》右子树，右子树访问结束后回到该结点，才能访问该结点

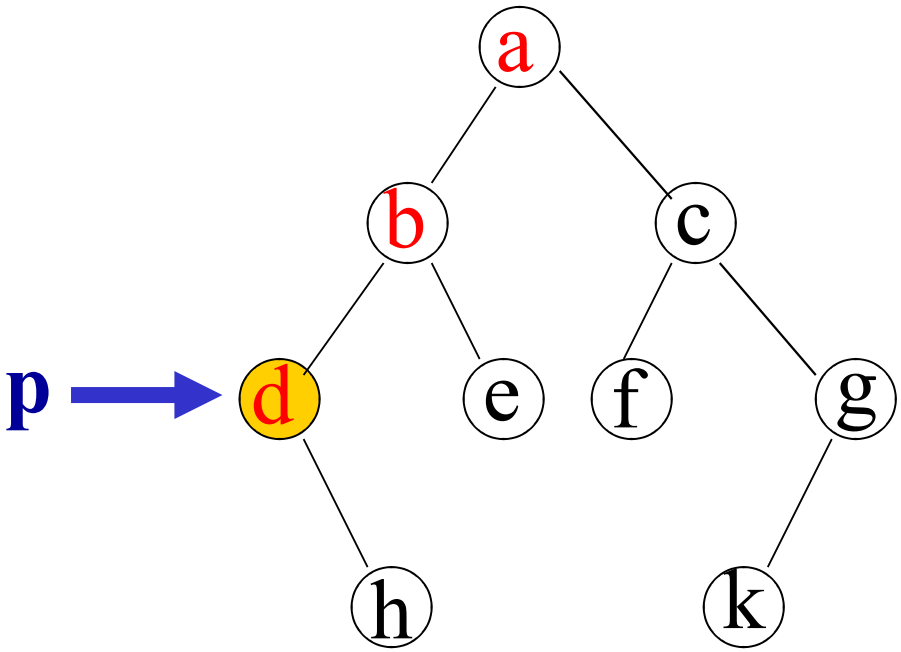


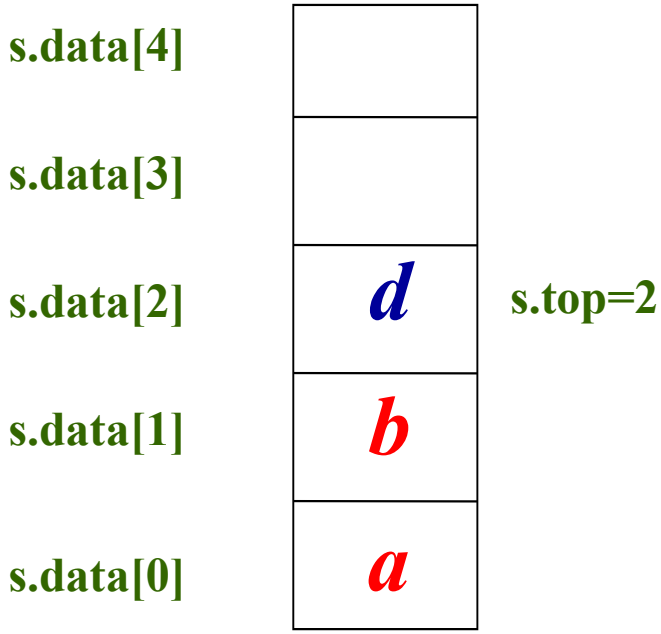


栈中结点要设标志域，指示该结点目前是被访问其左子树还是右子树

二叉树后序遍历的非递归算法实现

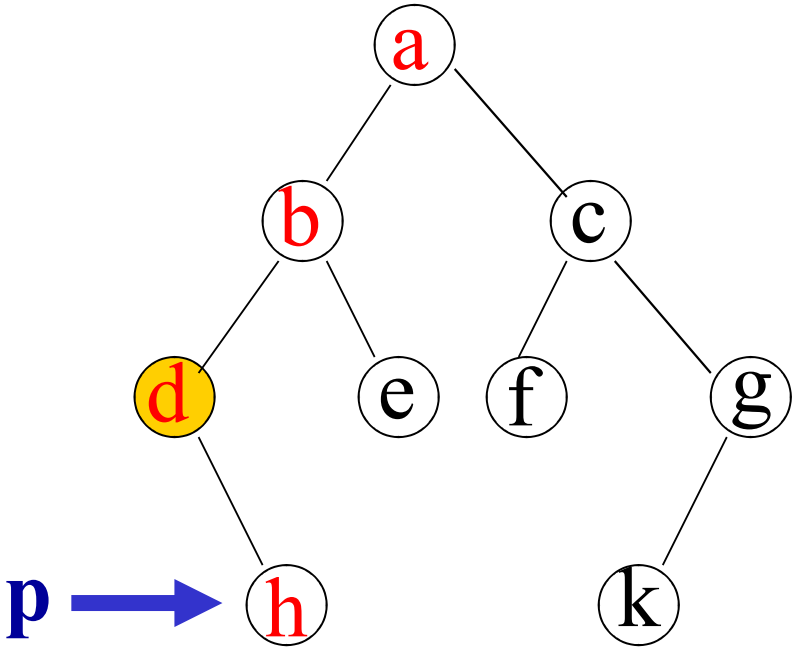
栈中结点红色标志指示该结点目前是被访问其左子树
蓝色标志指示该结点目前是被访问右子树

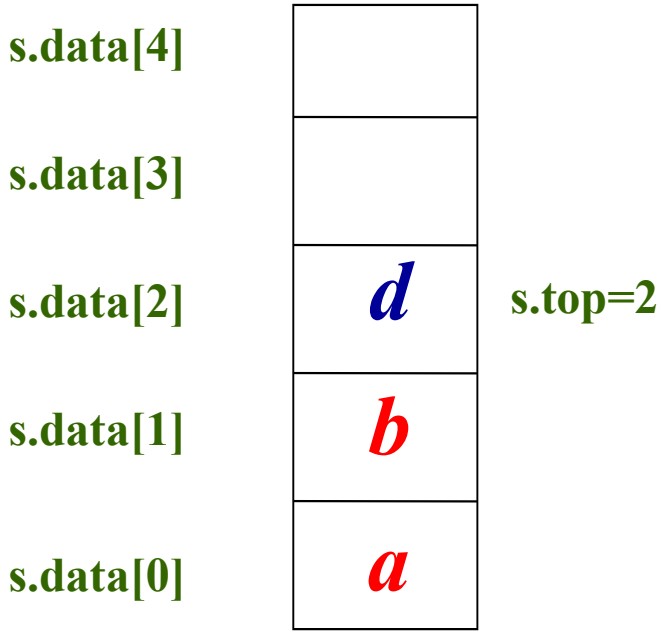




二叉树后序遍历的非递归算法实现

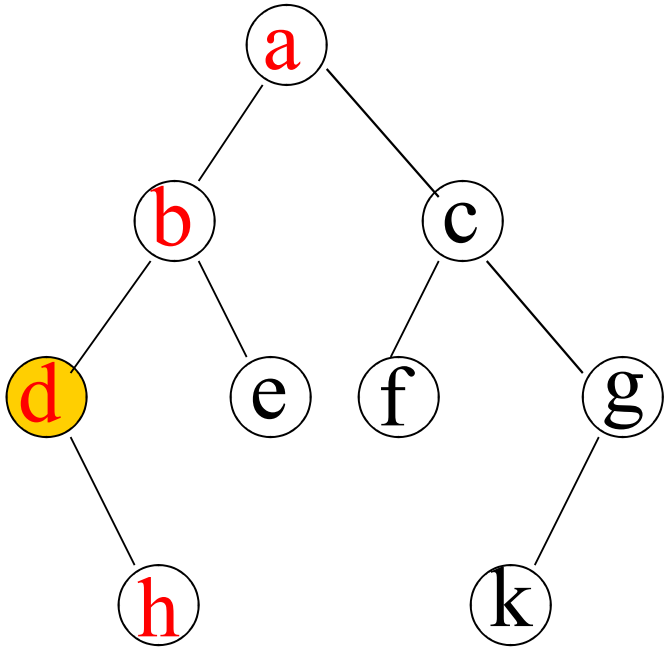
栈中结点红色标志指示该结点目前是被访问其左子树
蓝色标志指示该结点目前是被访问右子树

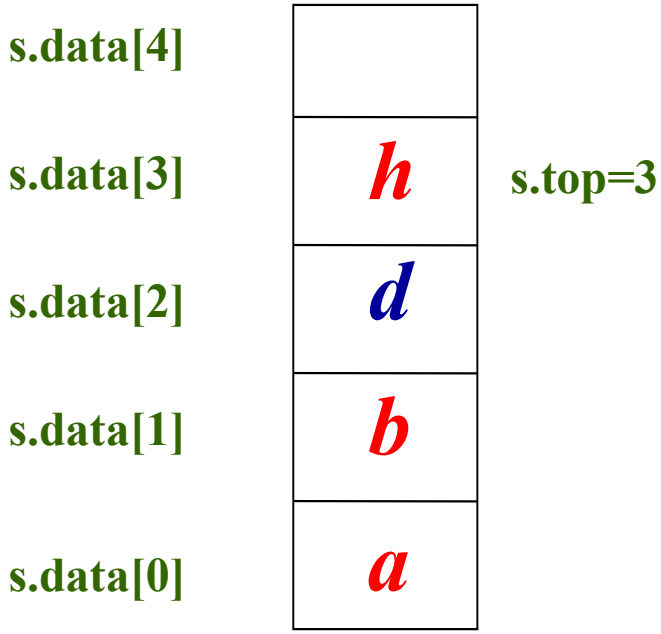




二叉树后序遍历的非递归算法实现

栈中结点红色标志指示该结点目前是被访问其左子树
蓝色标志指示该结点目前是被访问右子树

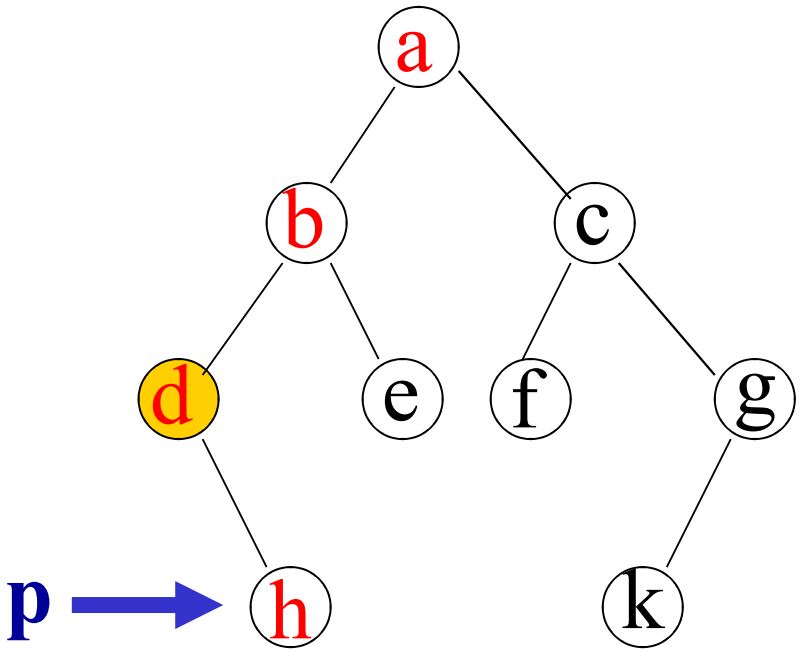


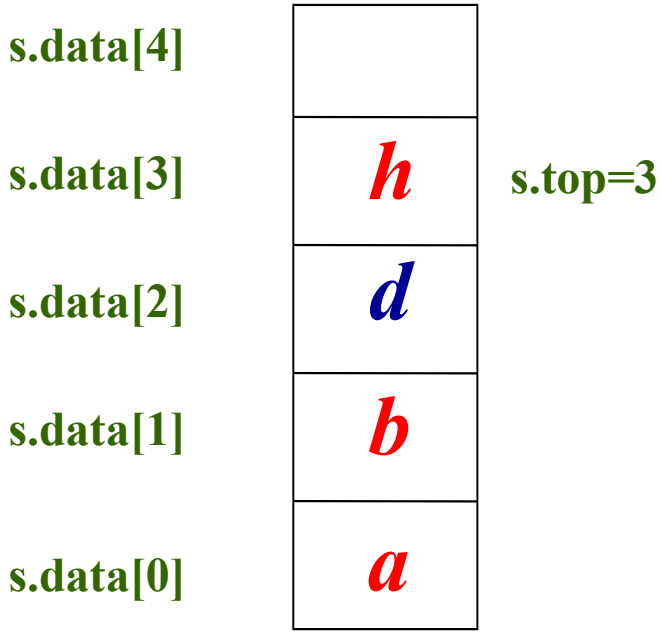


p →

二叉树后序遍历的非递归算法实现

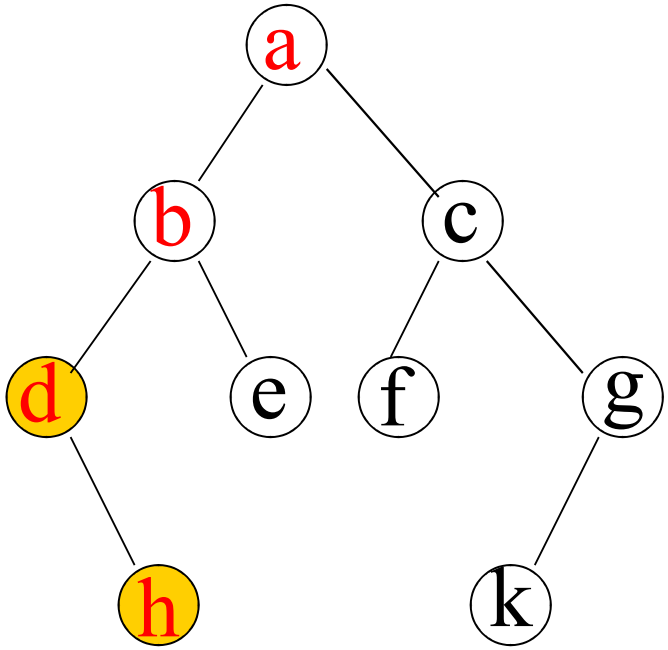
栈中结点红色标志指示该结点目前是被访问其左子树
蓝色标志指示该结点目前是被访问右子树

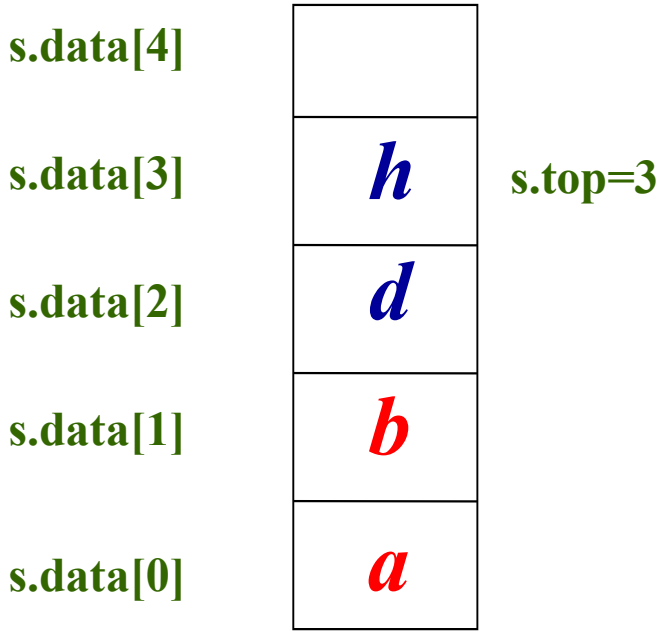




二叉树后序遍历的非递归算法实现

栈中结点红色标志指示该结点目前是被访问其左子树
蓝色标志指示该结点目前是被访问右子树

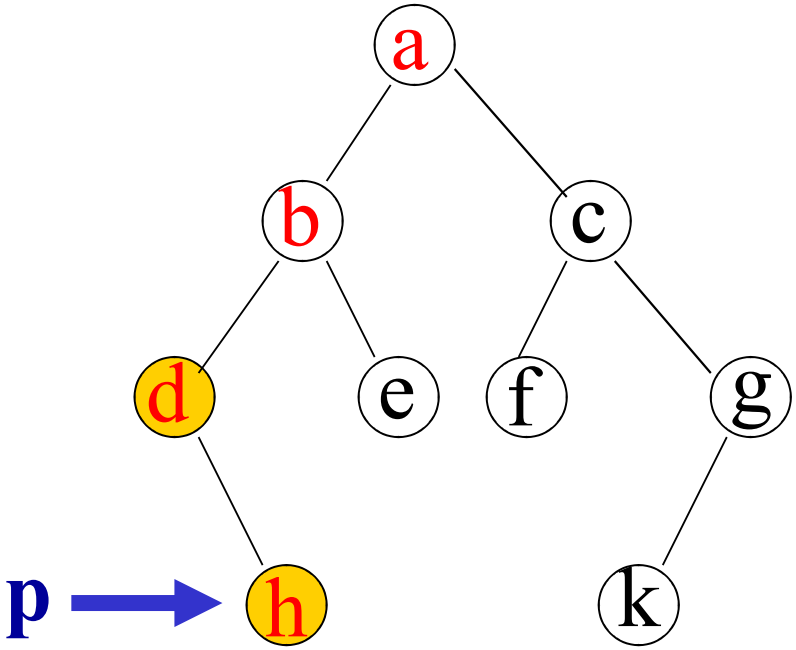


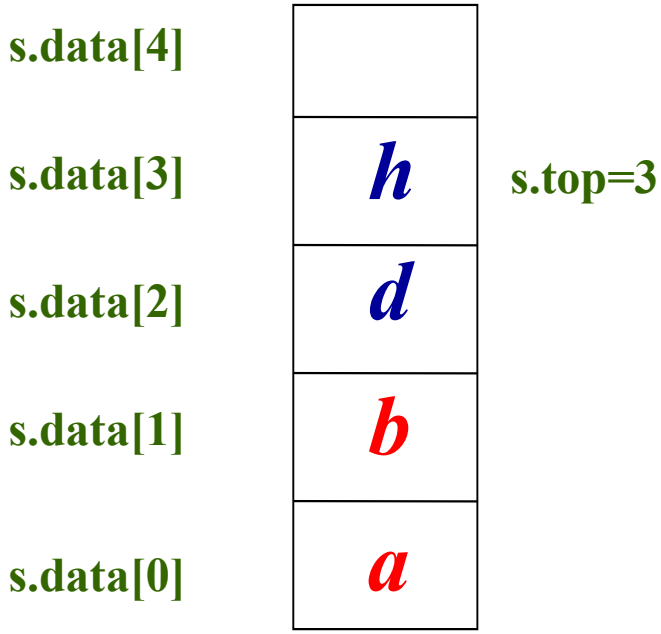


p →

二叉树后序遍历的非递归算法实现

栈中结点红色标志指示该结点目前是被访问其左子树
蓝色标志指示该结点目前是被访问右子树

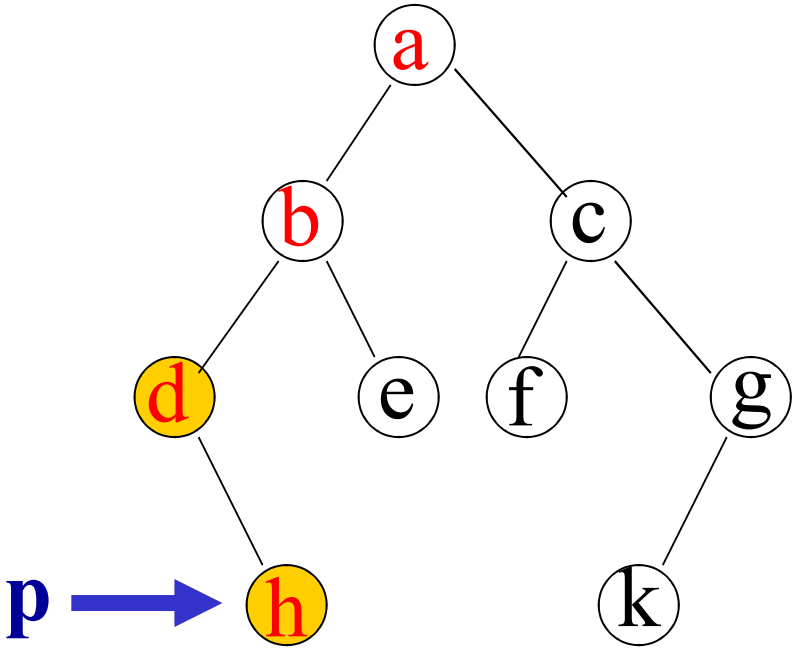


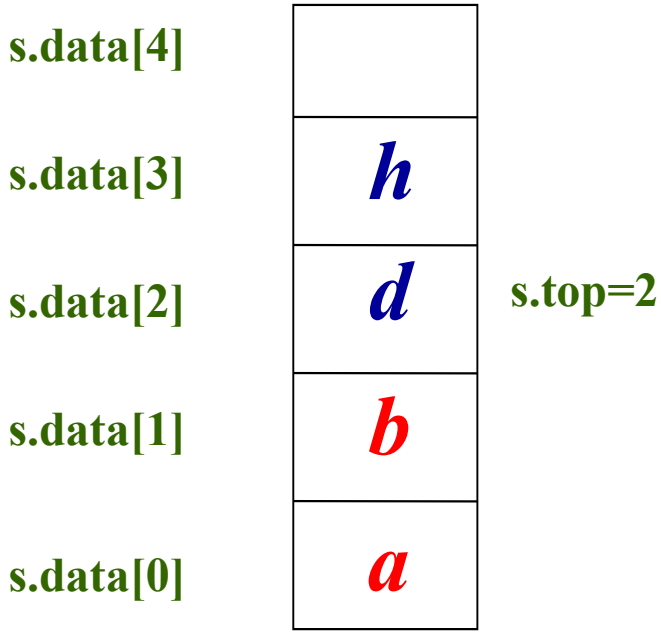


二叉树后序遍历的非递归算法实现

栈中结点红色标志指示该结点目前是被访问其左子树

h 蓝色标志指示该结点目前是被访问右子树

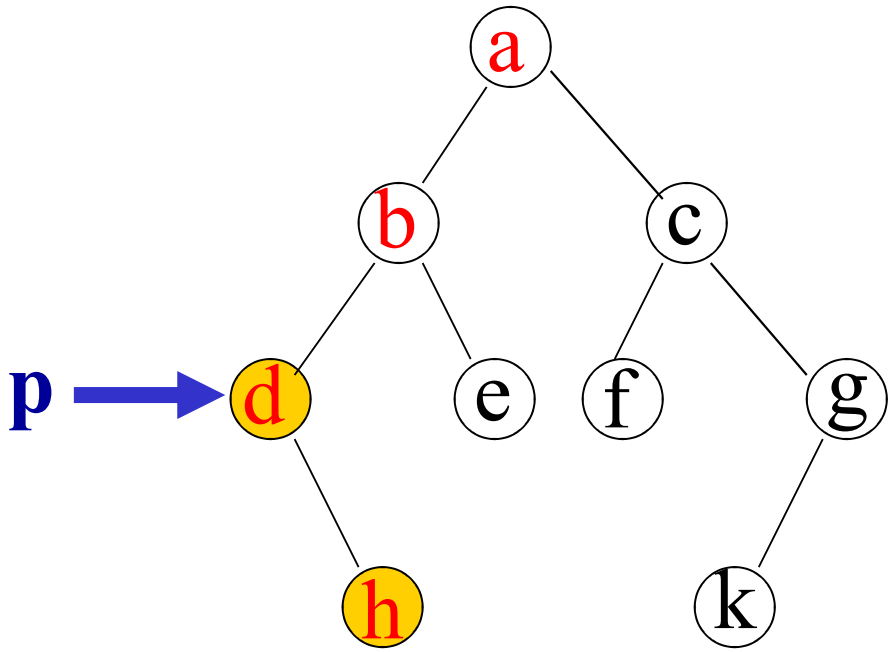


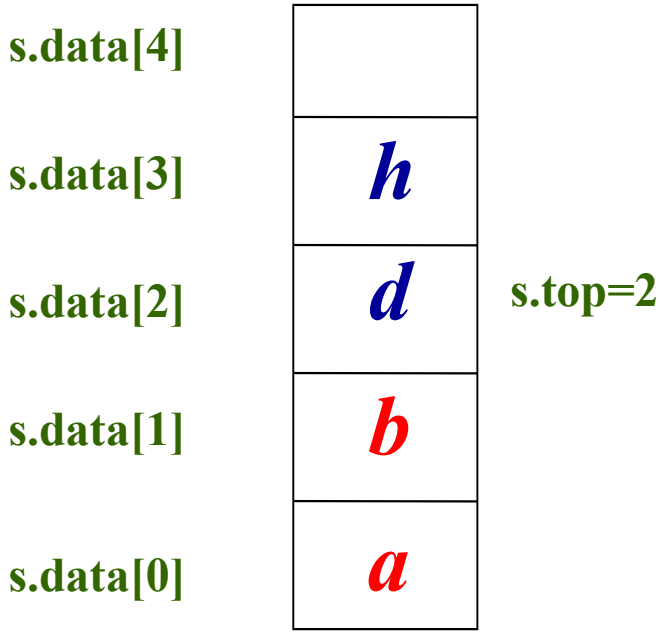


二叉树后序遍历的非递归算法实现

栈中结点红色标志指示该结点目前是被访问其左子树

h 蓝色标志指示该结点目前是被访问右子树

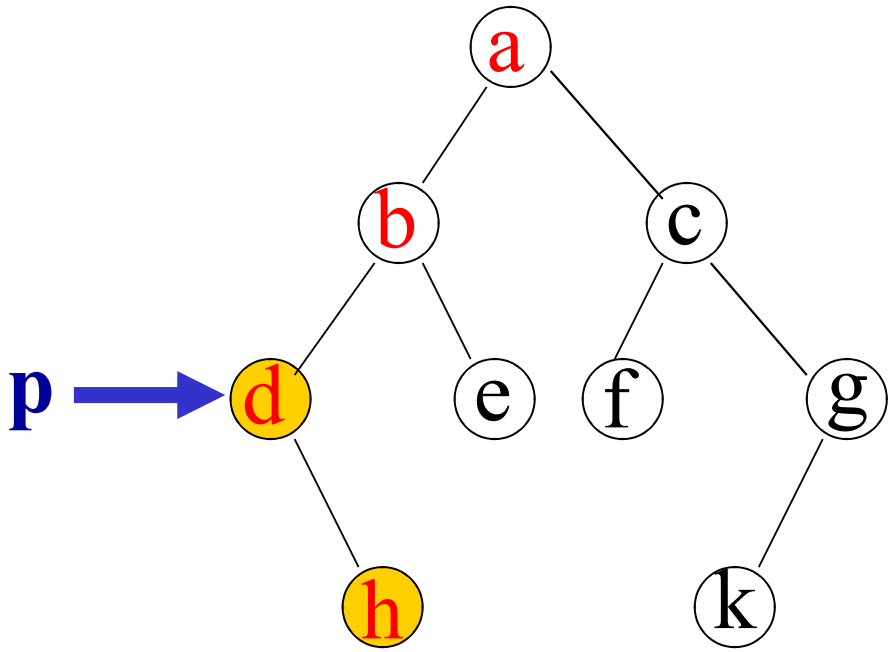


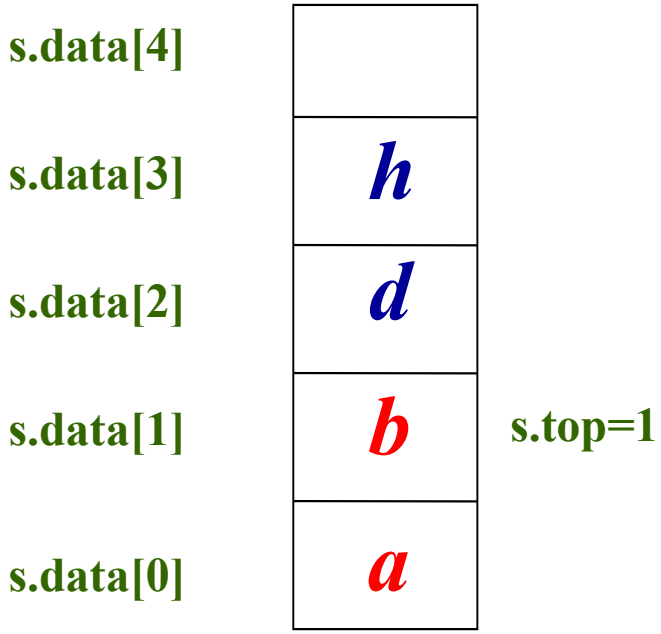


二叉树后序遍历的非递归算法实现

栈中结点红色标志指示该结点目前是被访问其左子树
蓝色标志指示该结点目前是被访问右子树

hd

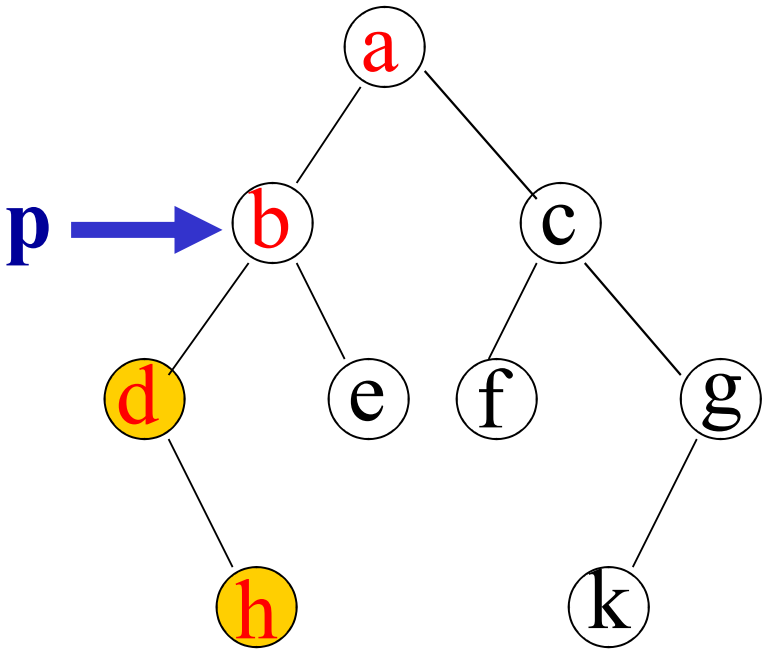


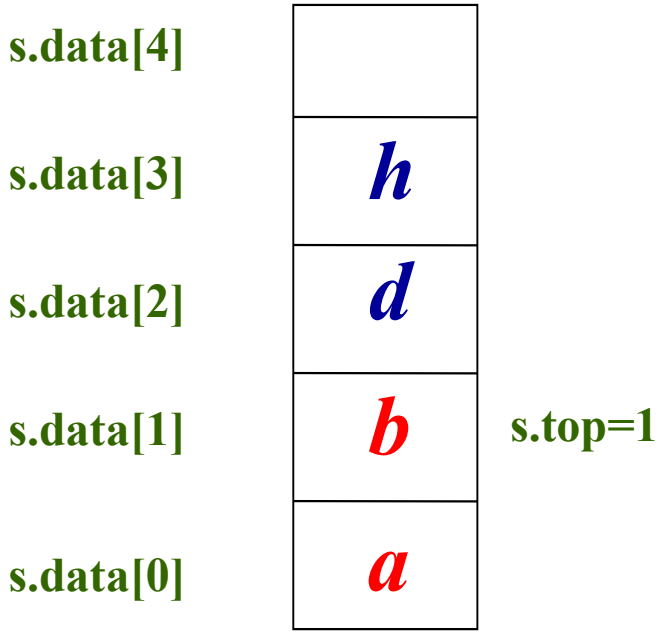


二叉树后序遍历的非递归算法实现

栈中结点红色标志指示该结点目前是被访问其左子树
蓝色标志指示该结点目前是被访问右子树

hd

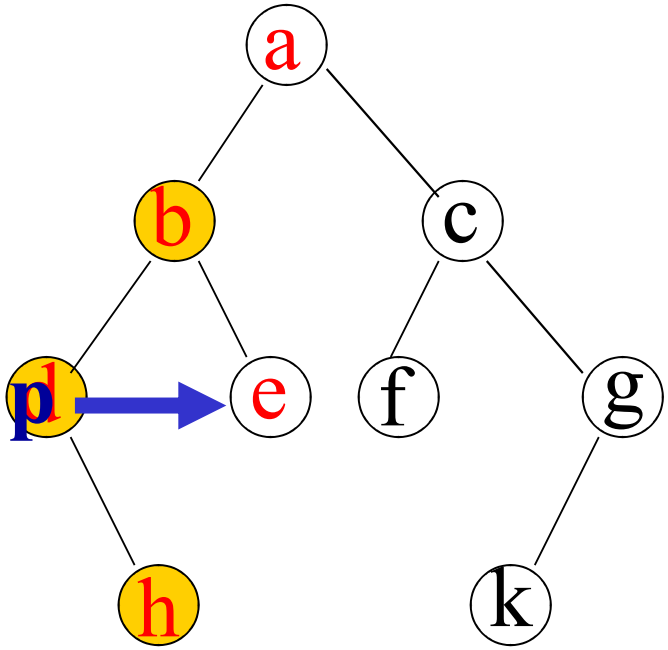




二叉树后序遍历的非递归算法实现

栈中结点红色标志指示该结点目前是被访问其左子树
蓝色标志指示该结点目前是被访问右子树

hd

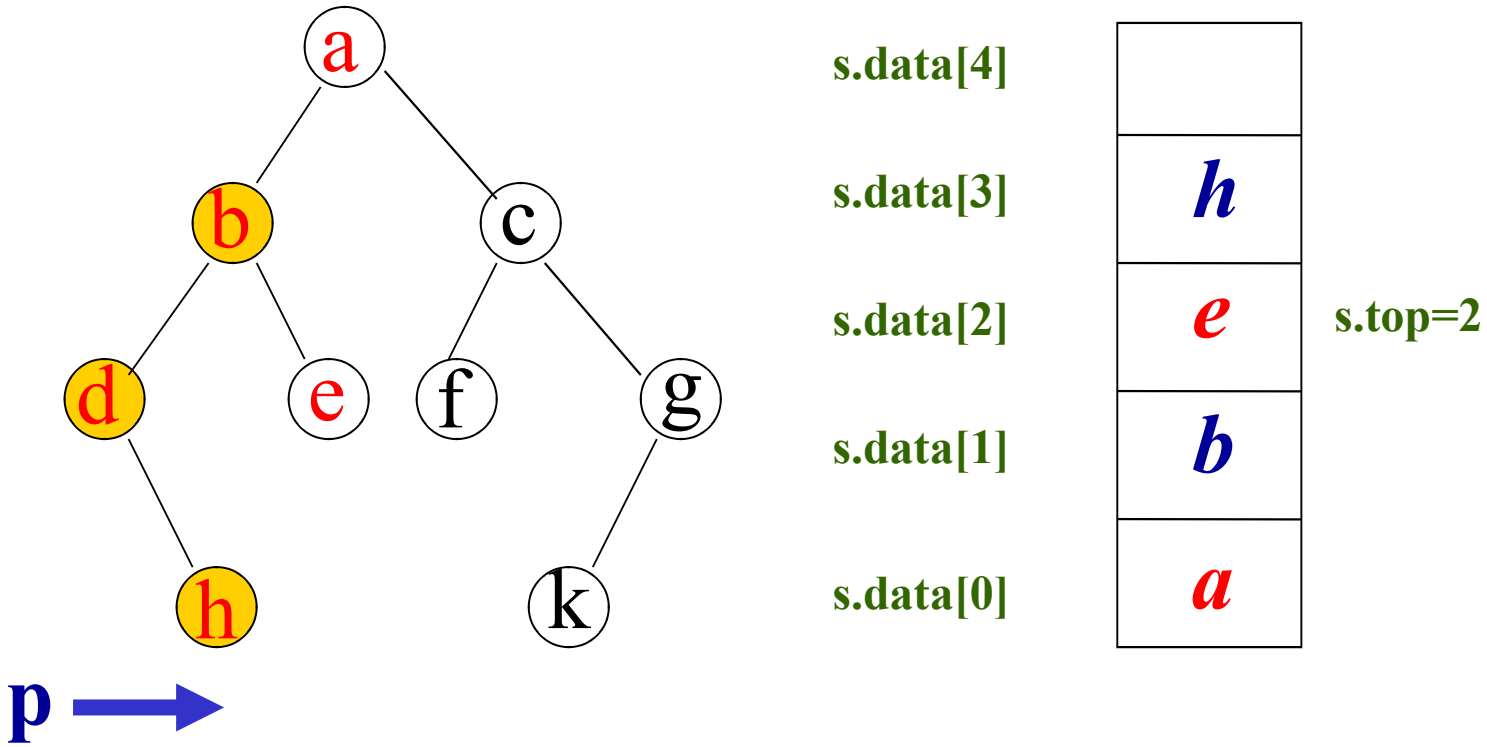


s.data[4]		
s.data[3]	*h*	
s.data[2]	*d*	
s.data[1]	*b*	s.top=1
s.data[0]	*a*	

二叉树后序遍历的非递归算法实现

栈中结点红色标志指示该结点目前是被访问其左子树
蓝色标志指示该结点目前是被访问右子树

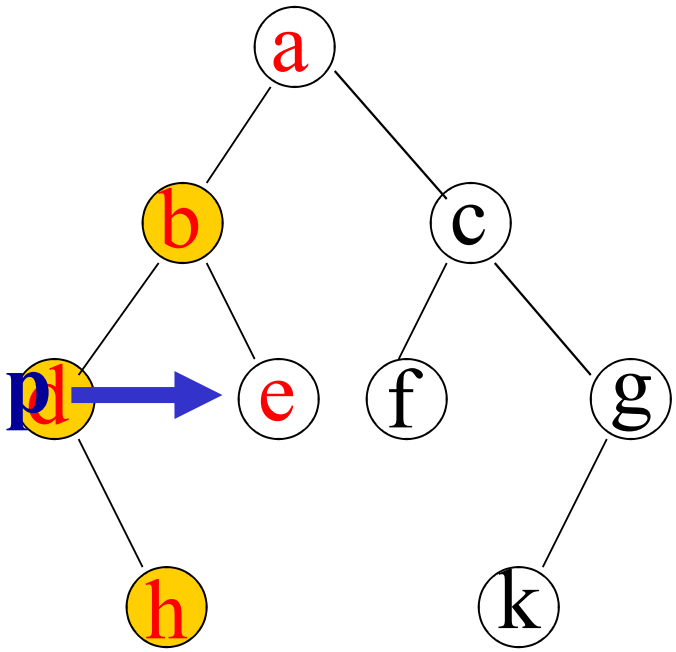
hd



二叉树后序遍历的非递归算法实现

栈中结点红色标志指示该结点目前是被访问其左子树
蓝色标志指示该结点目前是被访问右子树

hd

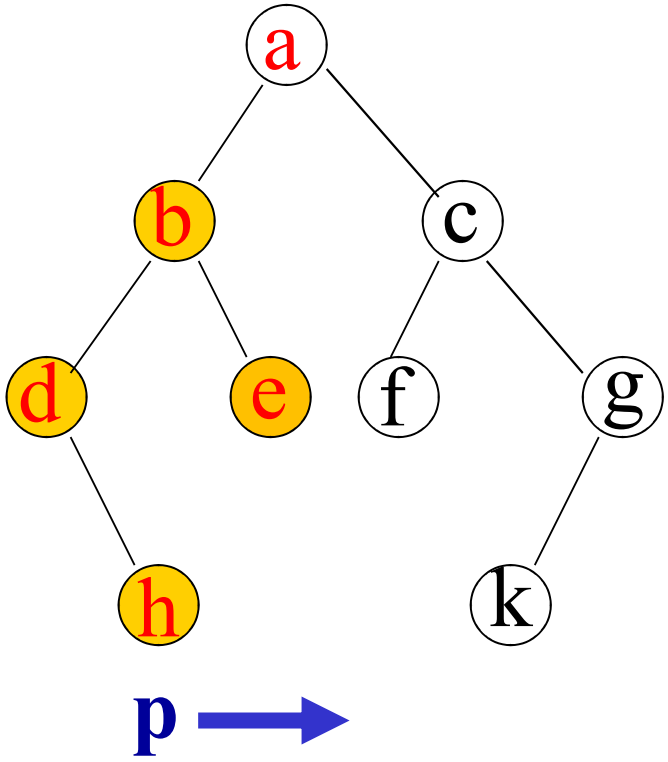


s.data[4]	
s.data[3]	*h*
s.data[2]	*e*
s.data[1]	*b*
s.data[0]	*a*

s.top=2

二叉树后序遍历的非递归算法实现

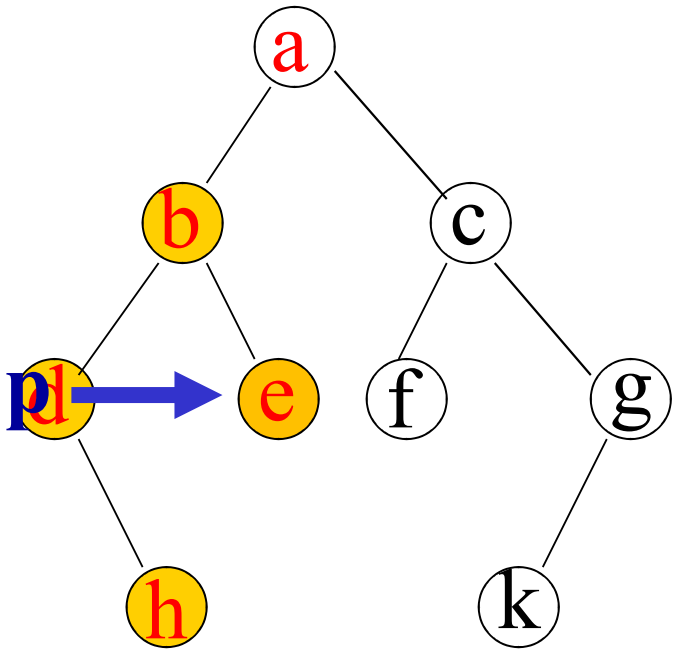
hd



s.data[4]		
s.data[3]	*h*	
s.data[2]	*e*	s.top=2
s.data[1]	*b*	
s.data[0]	*a*	

二叉树后序遍历的非递归算法实现

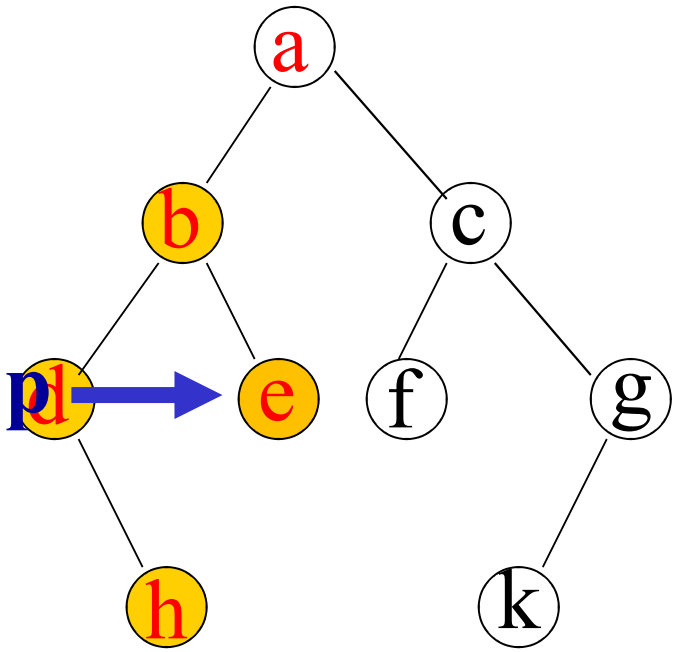
hd



s.data[4]		
s.data[3]	*h*	
s.data[2]	*e*	s.top=2
s.data[1]	*b*	
s.data[0]	*a*	

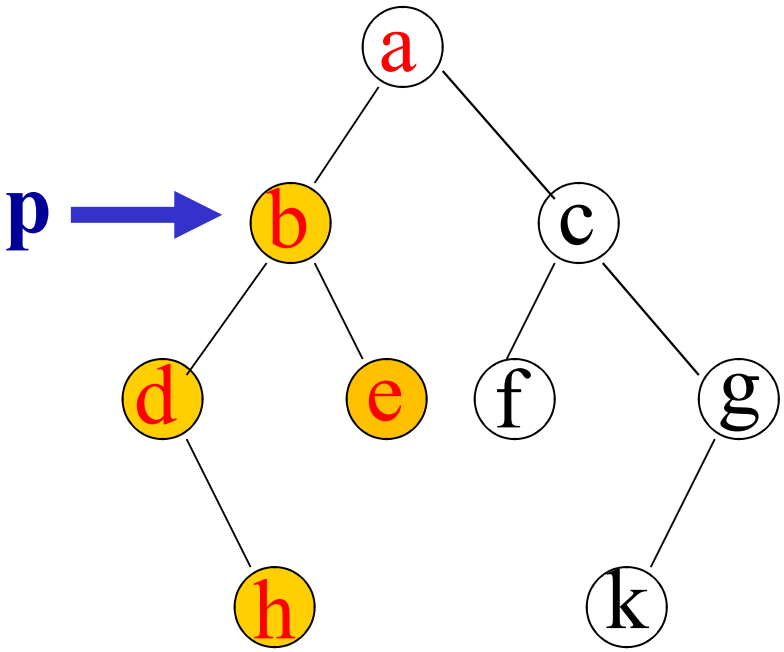
二叉树后序遍历的非递归算法实现

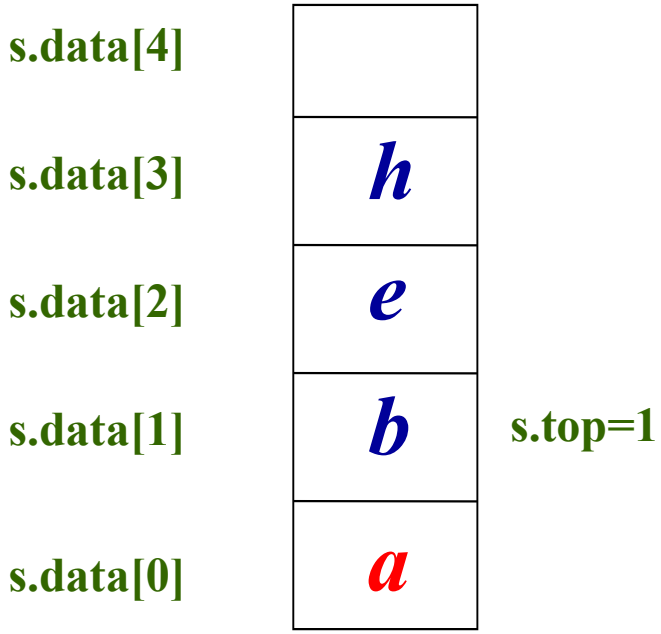
hde



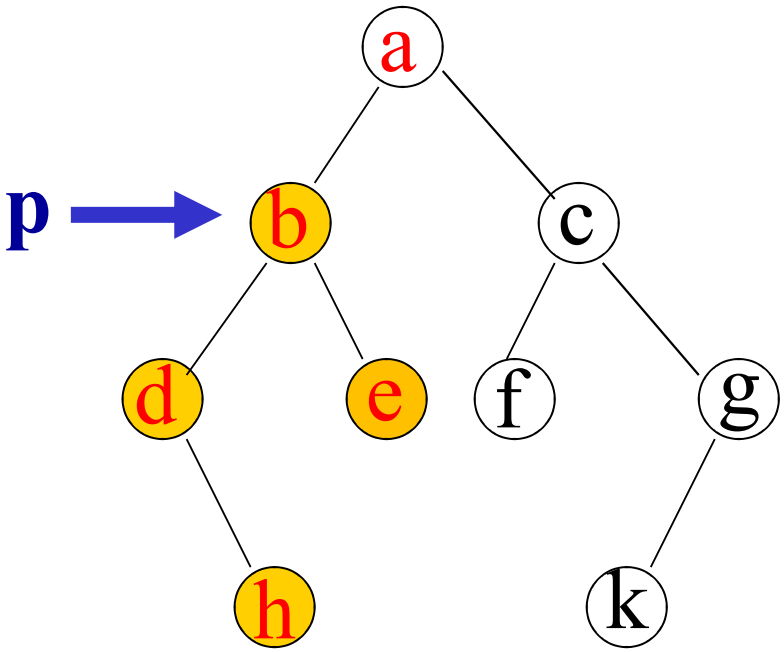
s.data[4]		
s.data[3]	*h*	
s.data[2]	*e*	
s.data[1]	*b*	s.top=1
s.data[0]	*a*	

hde





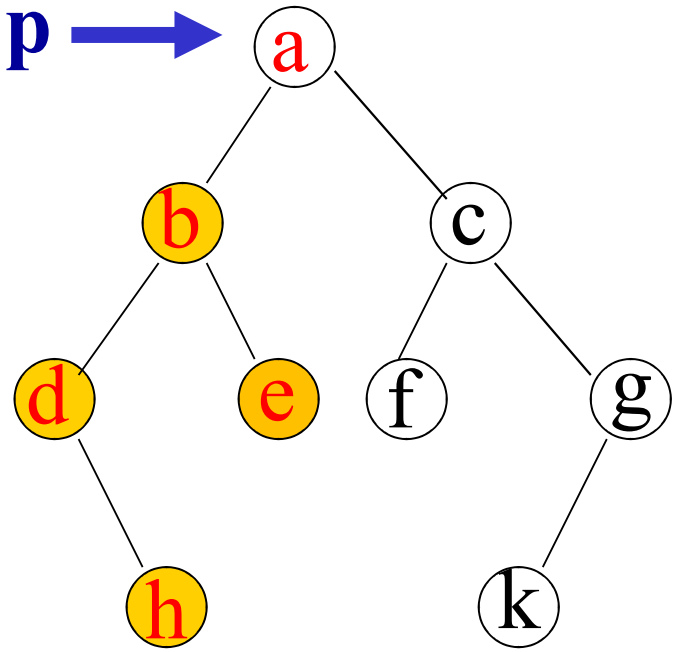
hdeb

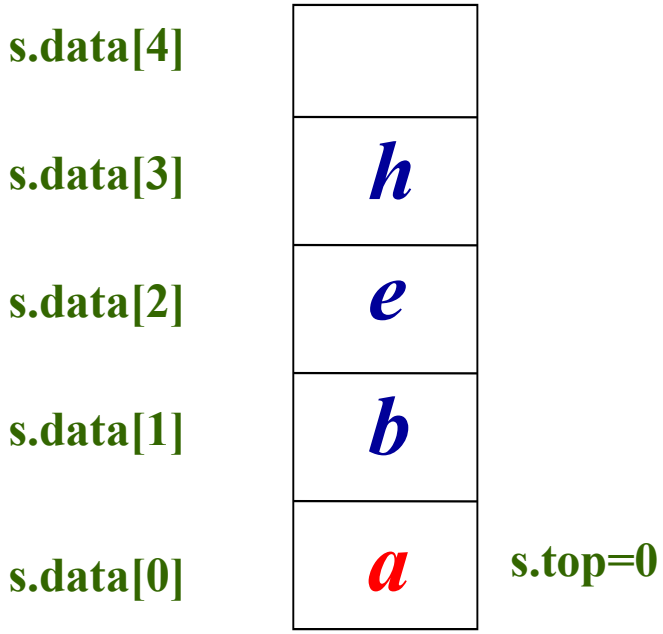


s.data[4]		
s.data[3]	*h*	
s.data[2]	*e*	
s.data[1]	*b*	
s.data[0]	*a*	s.top=0

二叉树后序遍历的非递归算法实现

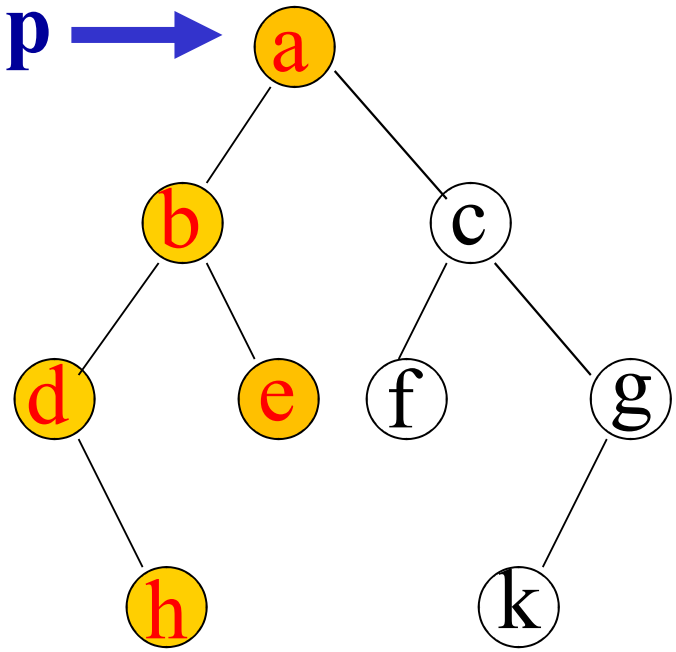
hdeb

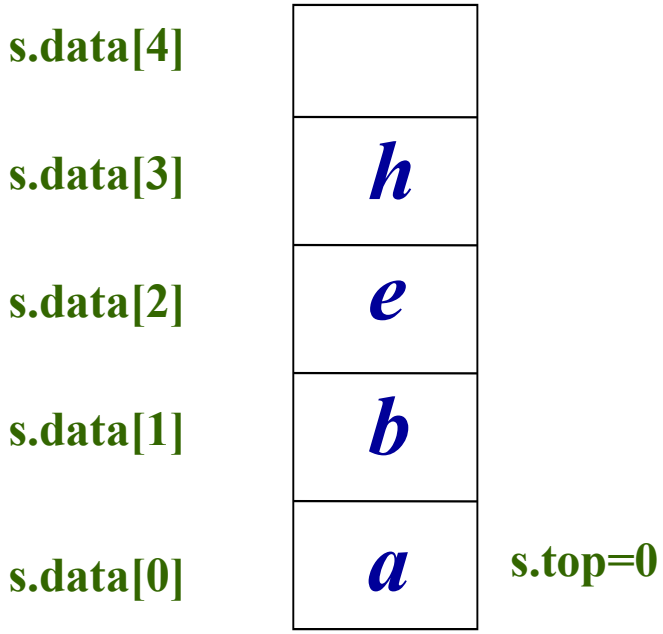




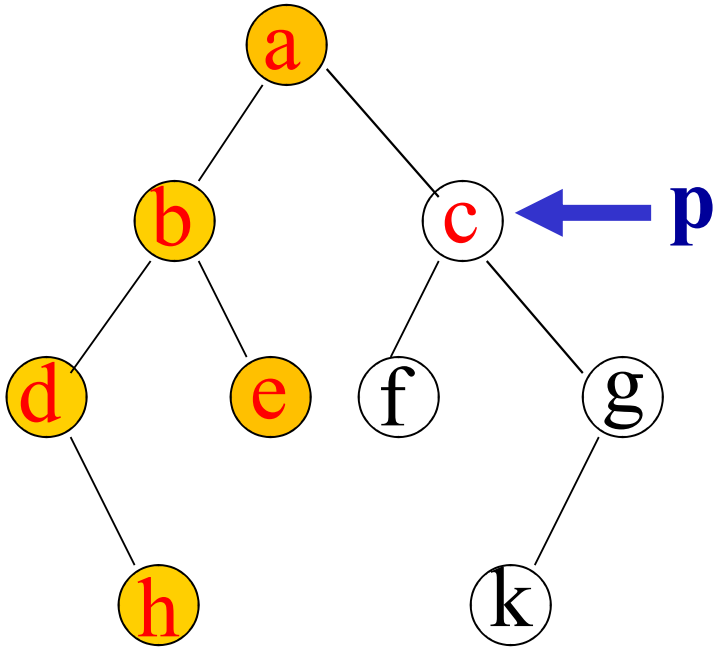
二叉树后序遍历的非递归算法实现

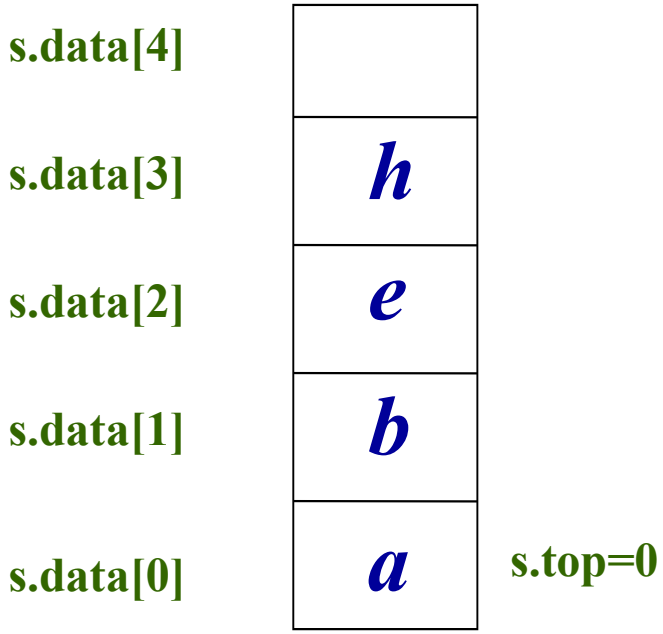
hdeb





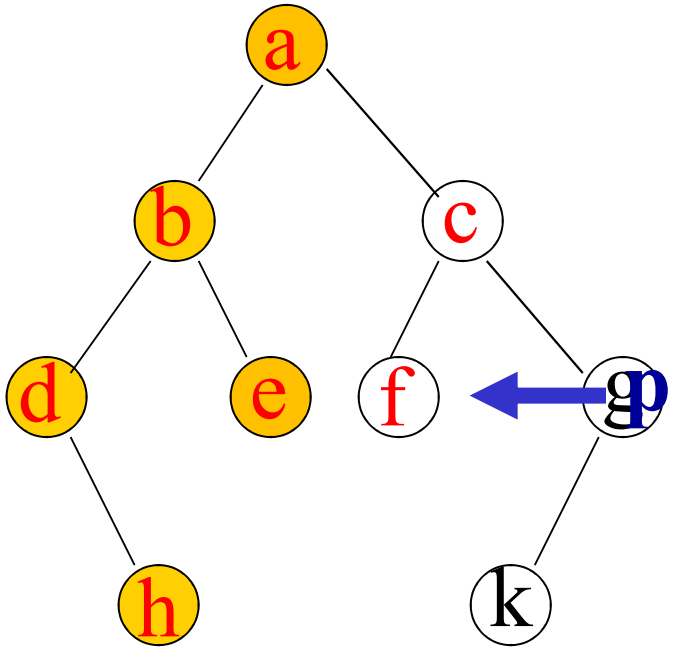
hdeb





二叉树后序遍历的非递归算法实现

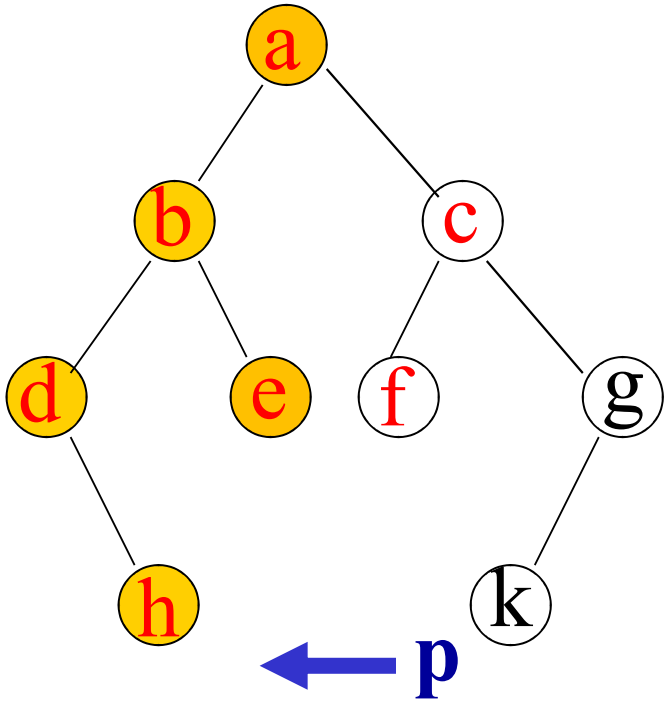
hdeb



s.data[4]		
s.data[3]	*h*	
s.data[2]	*e*	
s.data[1]	*c*	s.top=1
s.data[0]	*a*	

二叉树后序遍历的非递归算法实现

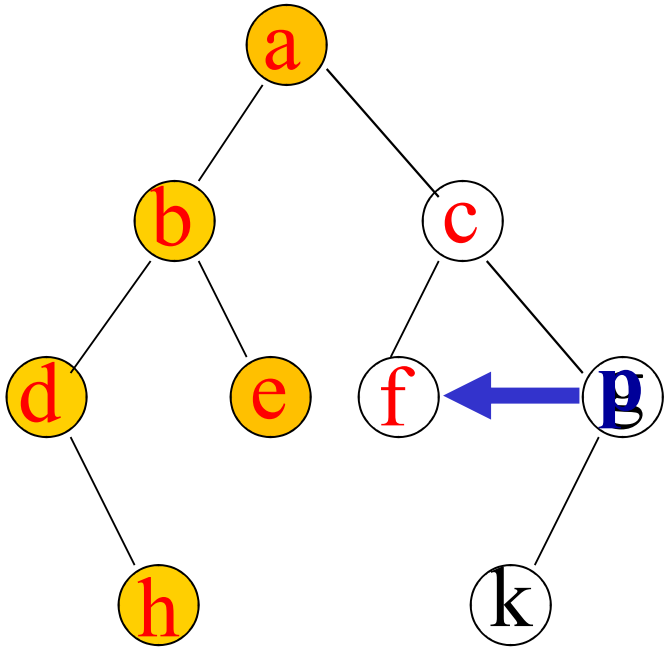
hdeb



s.data[4]		
s.data[3]	*h*	
s.data[2]	*f*	s.top=2
s.data[1]	*c*	
s.data[0]	*a*	

二叉树后序遍历的非递归算法实现

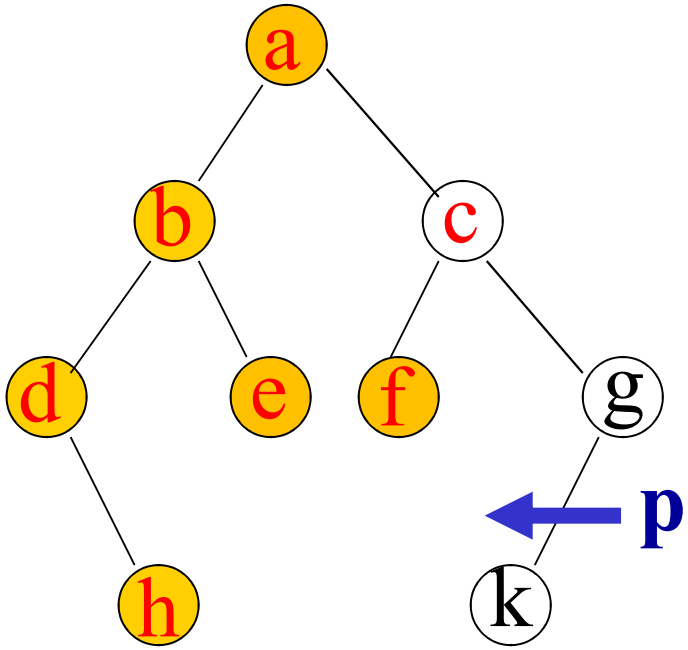
hdeb

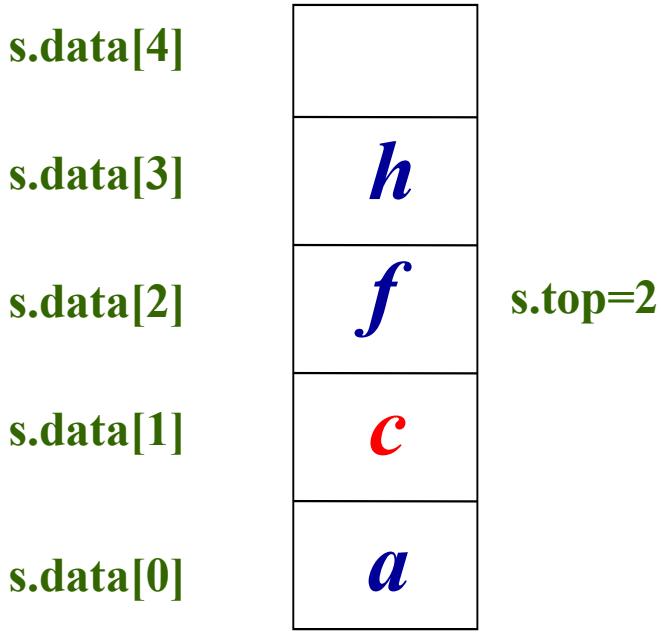


s.data[4]		
s.data[3]	*h*	
s.data[2]	*f*	s.top=2
s.data[1]	*c*	
s.data[0]	*a*	

二叉树后序遍历的非递归算法实现

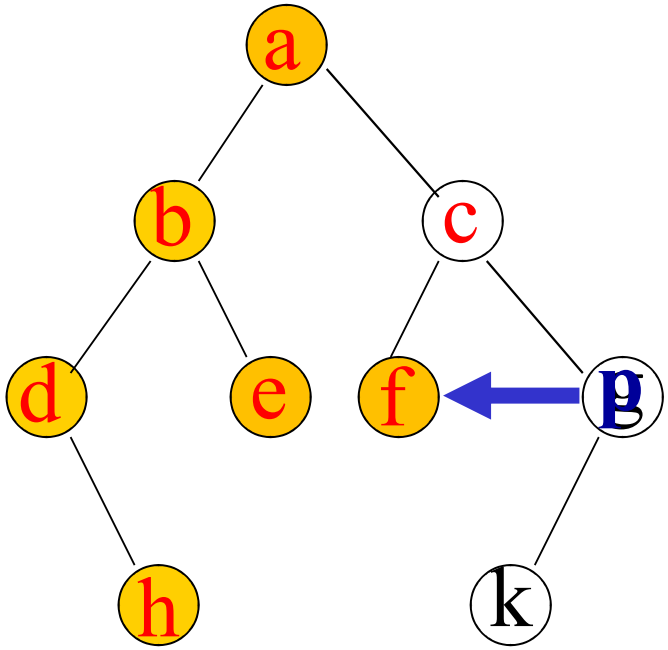
hdeb





二叉树后序遍历的非递归算法实现

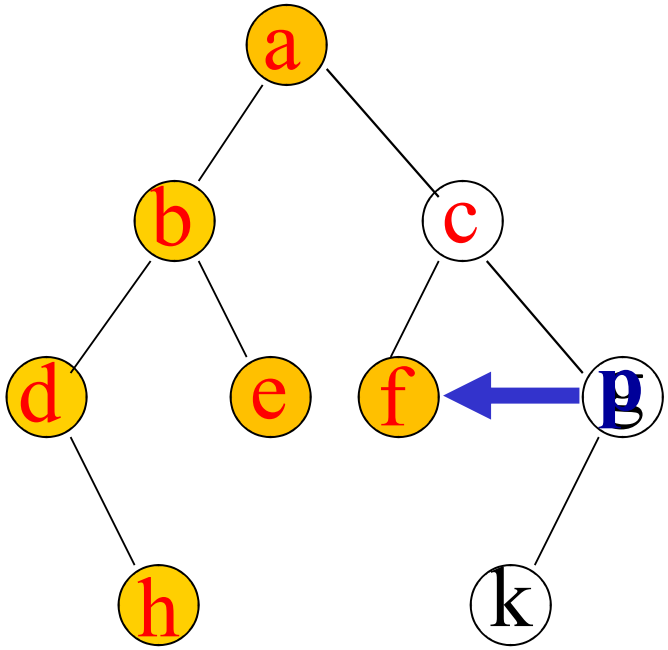
hdeb



s.data[4]		
s.data[3]	*h*	
s.data[2]	*f*	s.top=2
s.data[1]	*c*	
s.data[0]	*a*	

二叉树后序遍历的非递归算法实现

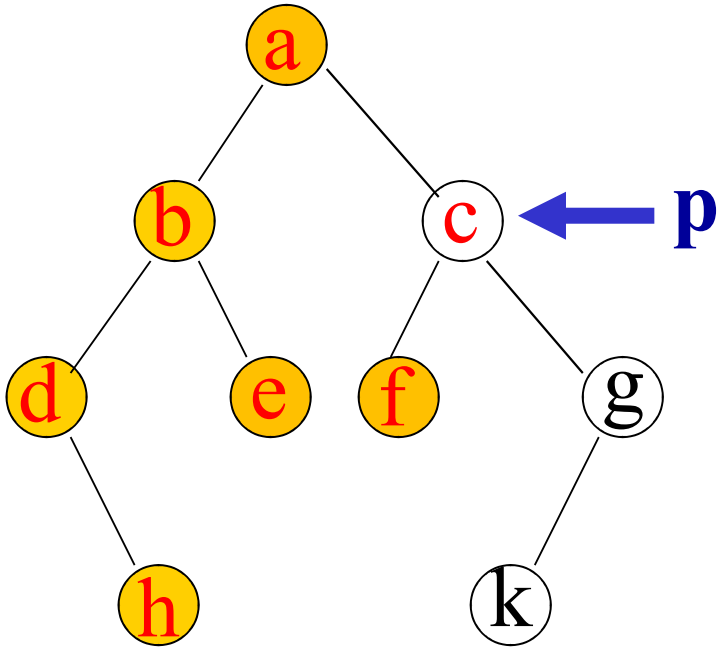
hdebf

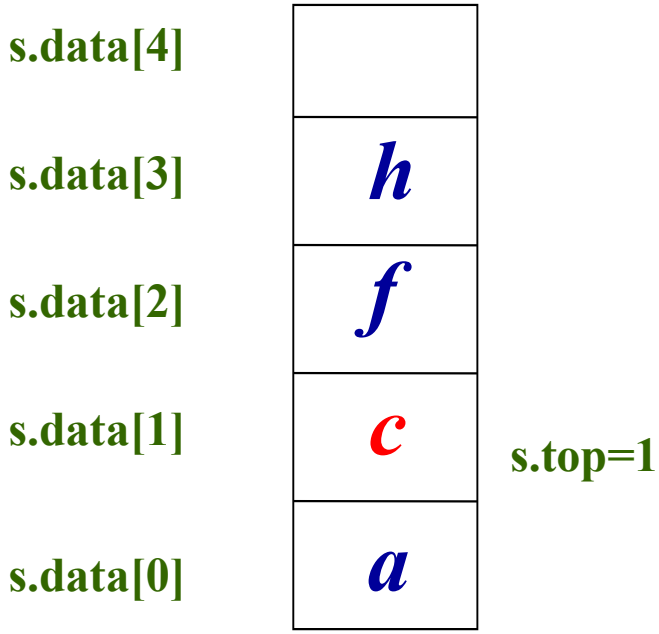


s.data[4]		
s.data[3]	*h*	
s.data[2]	*f*	
s.data[1]	*c*	s.top=2
s.data[0]	*a*	

二叉树后序遍历的非递归算法实现

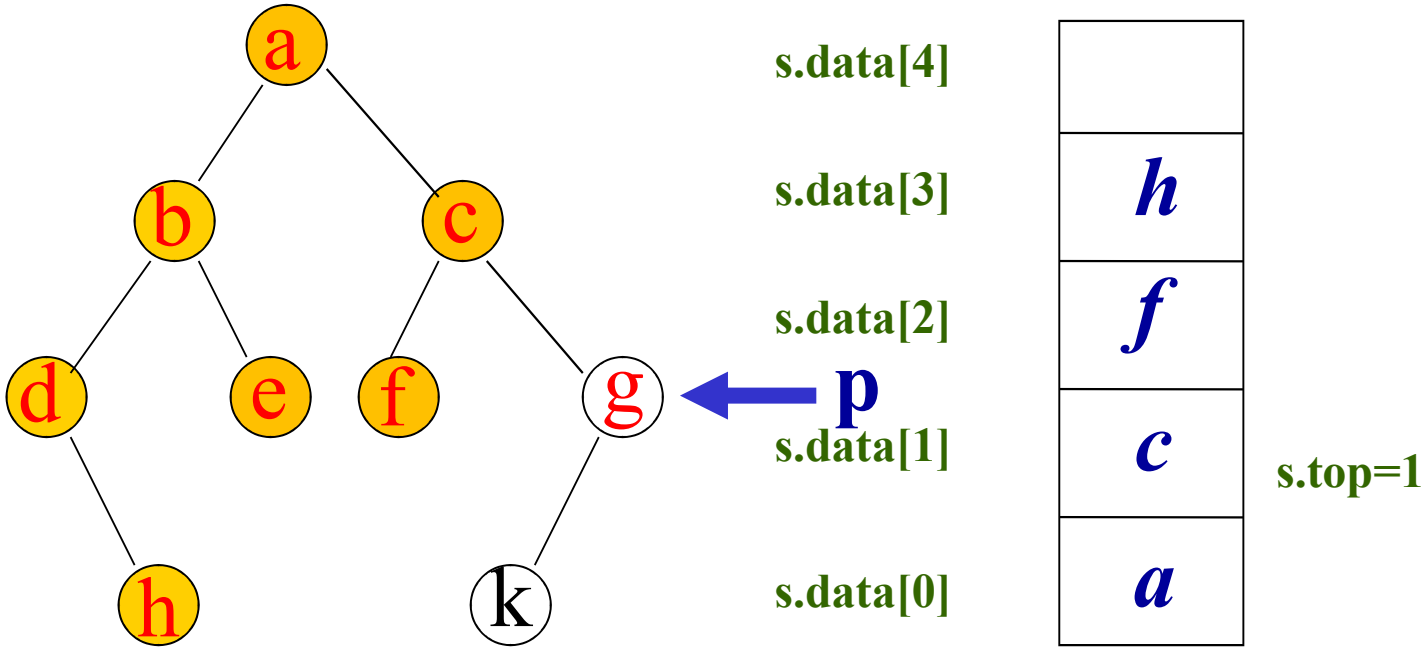
hdebf





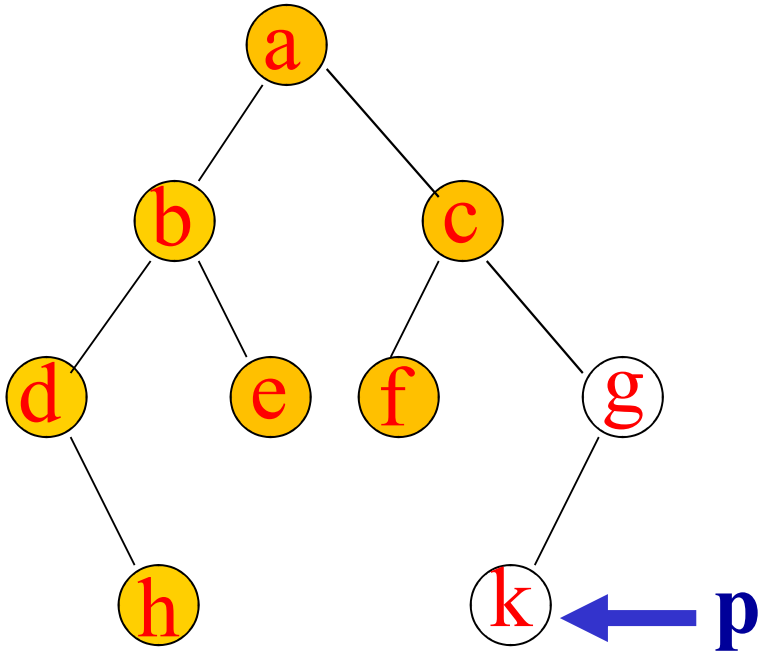
二叉树后序遍历的非递归算法实现

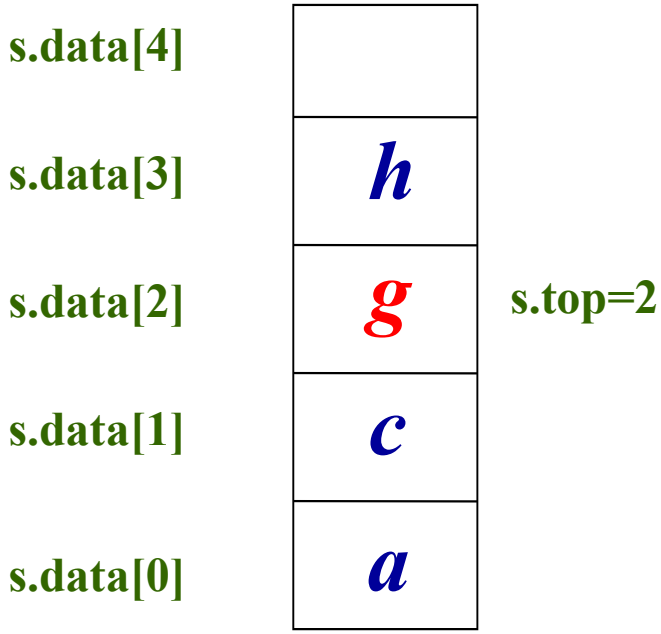
hdebf



二叉树后序遍历的非递归算法实现

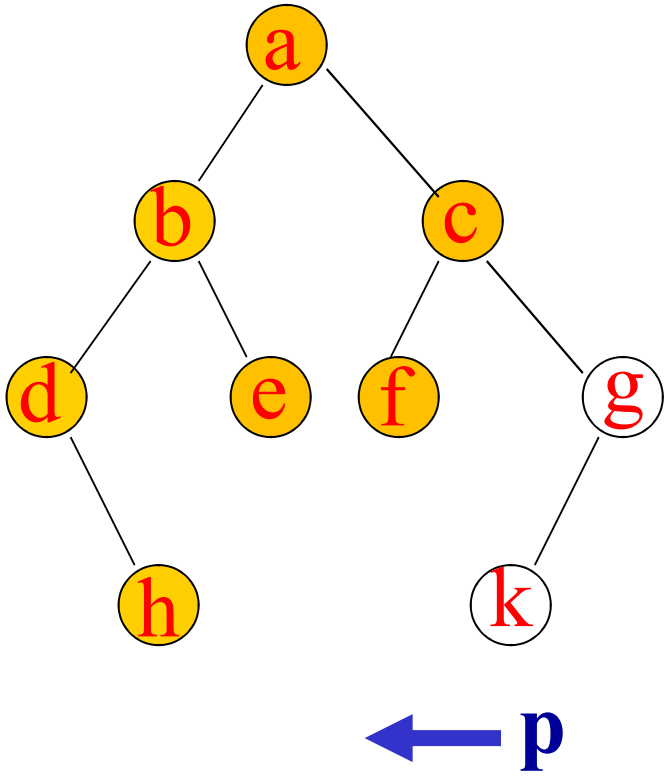
hdebf

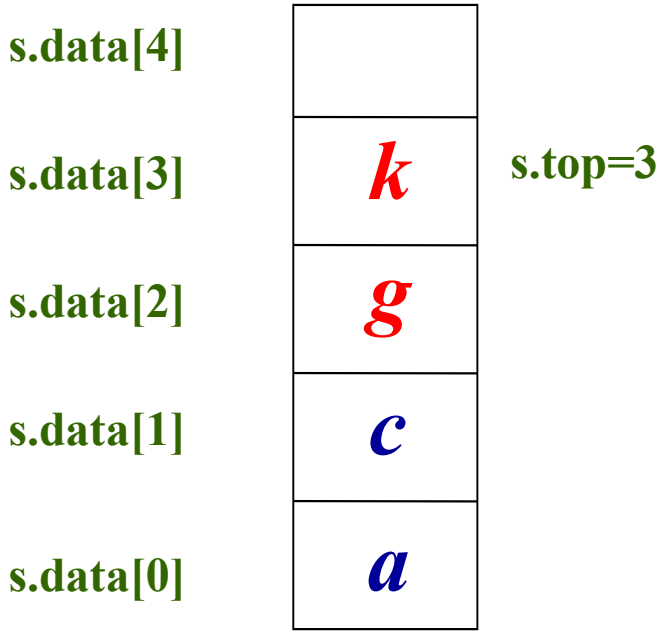




二叉树后序遍历的非递归算法实现

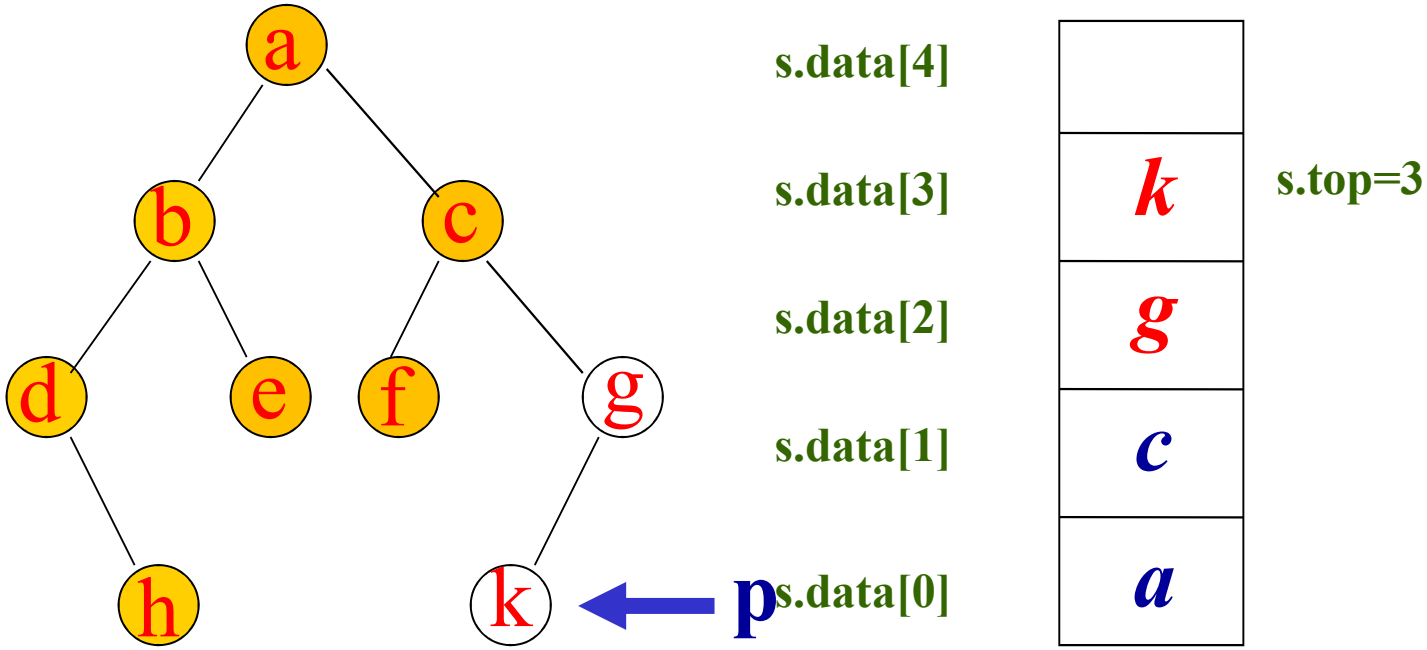
hdebf





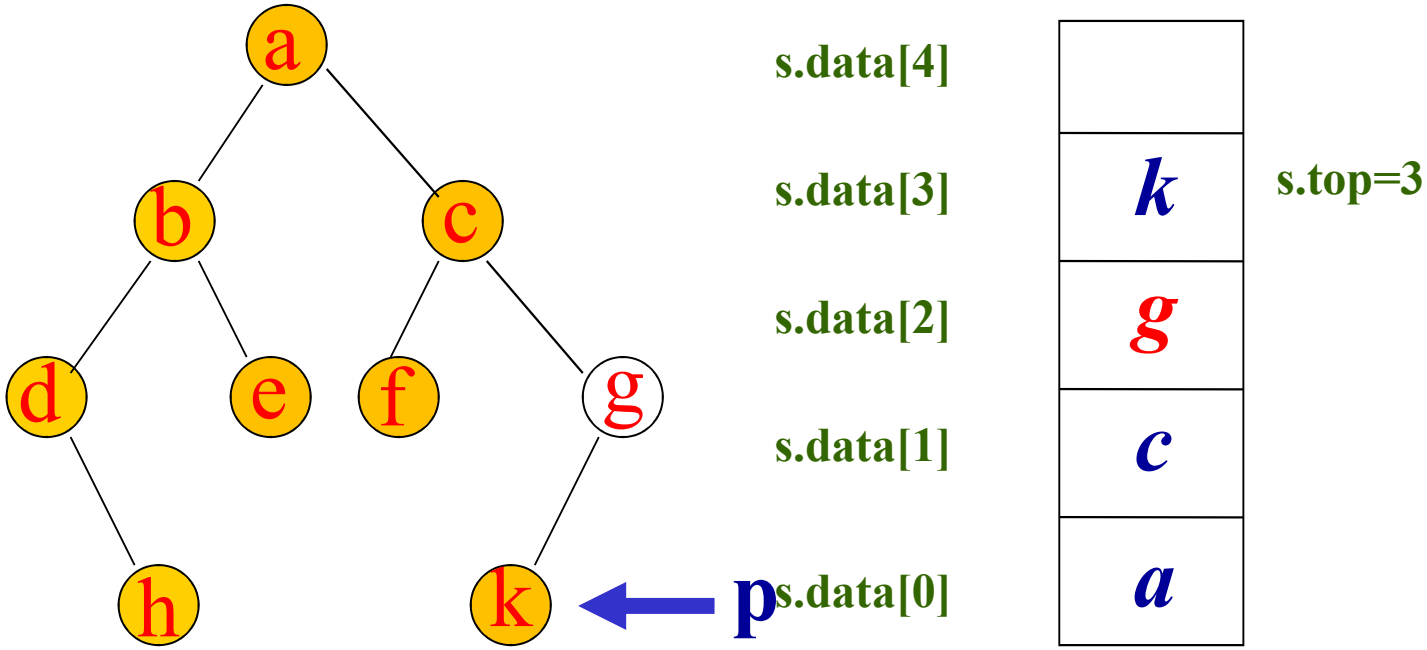
二叉树后序遍历的非递归算法实现

hdebf



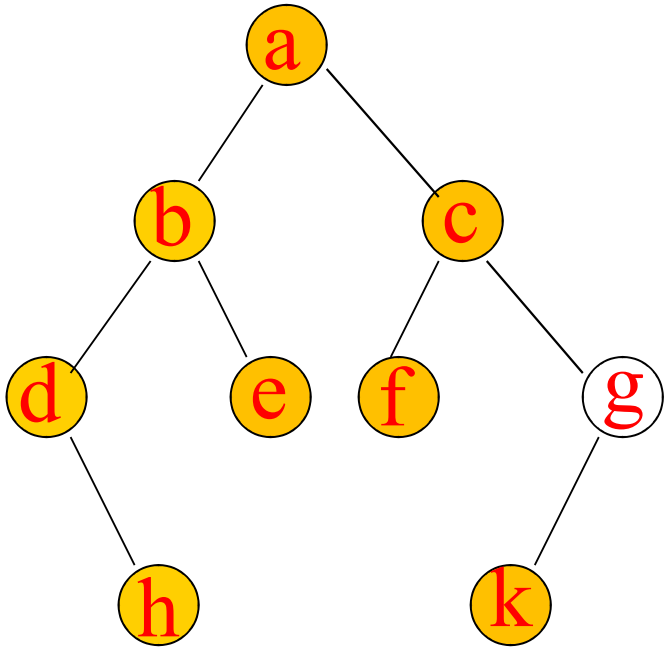
二叉树后序遍历的非递归算法实现

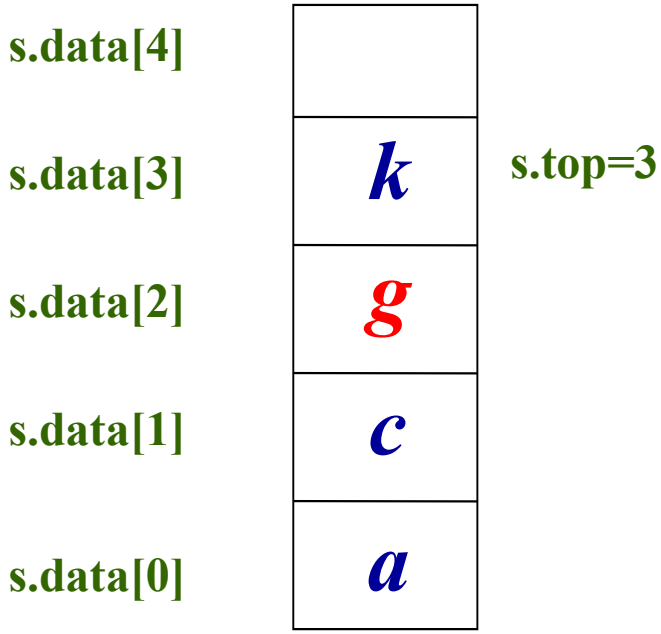
hdebf



二叉树后序遍历的非递归算法实现

hdebf

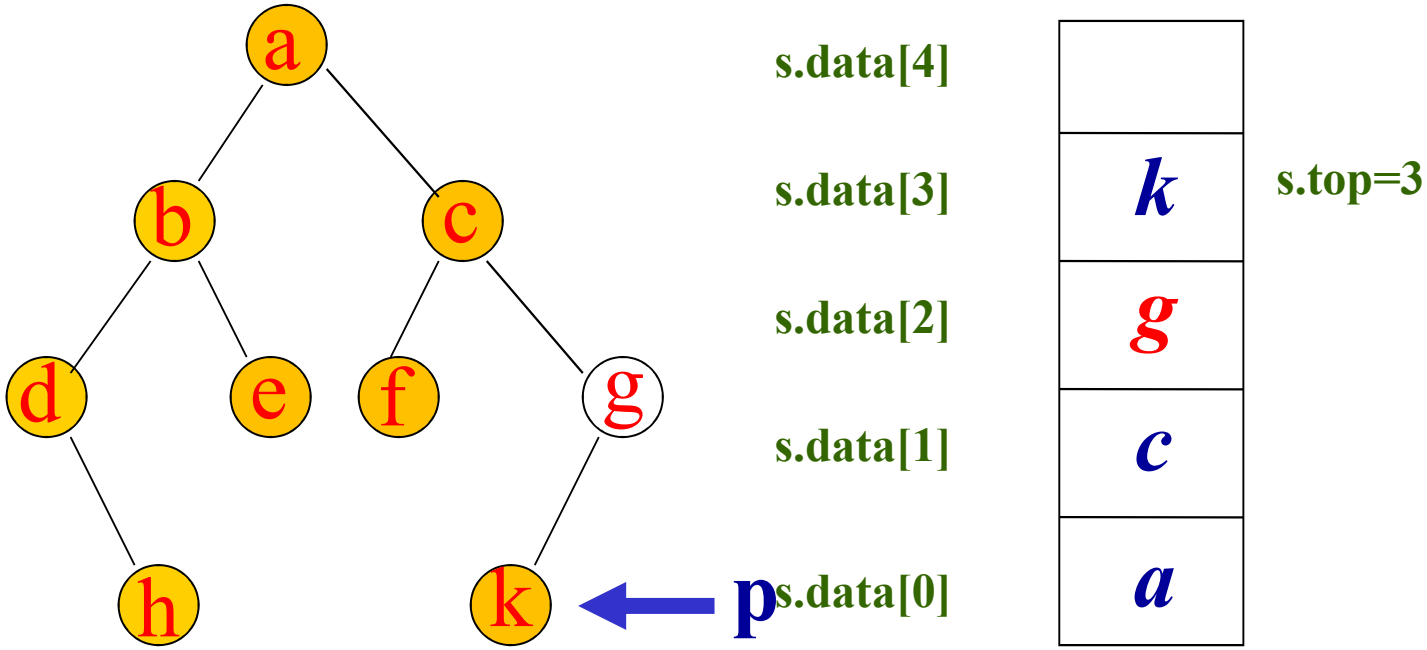




← p

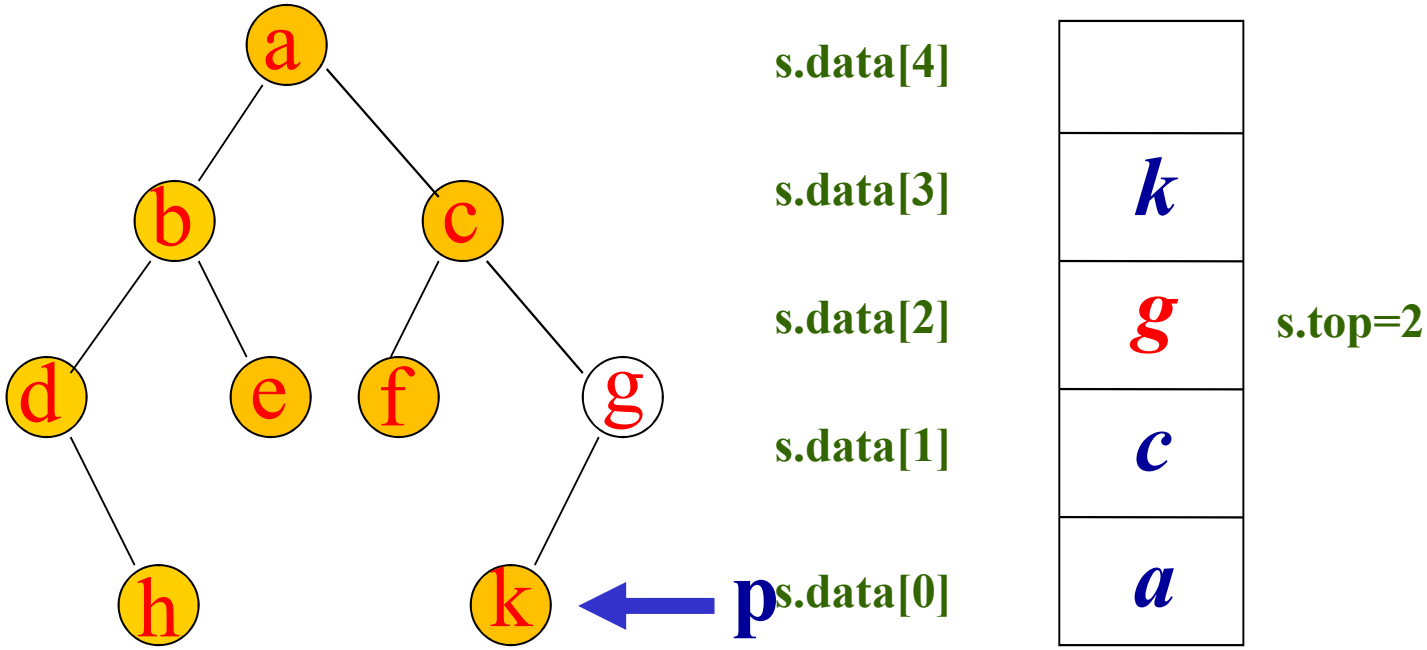
二叉树后序遍历的非递归算法实现

hdebf



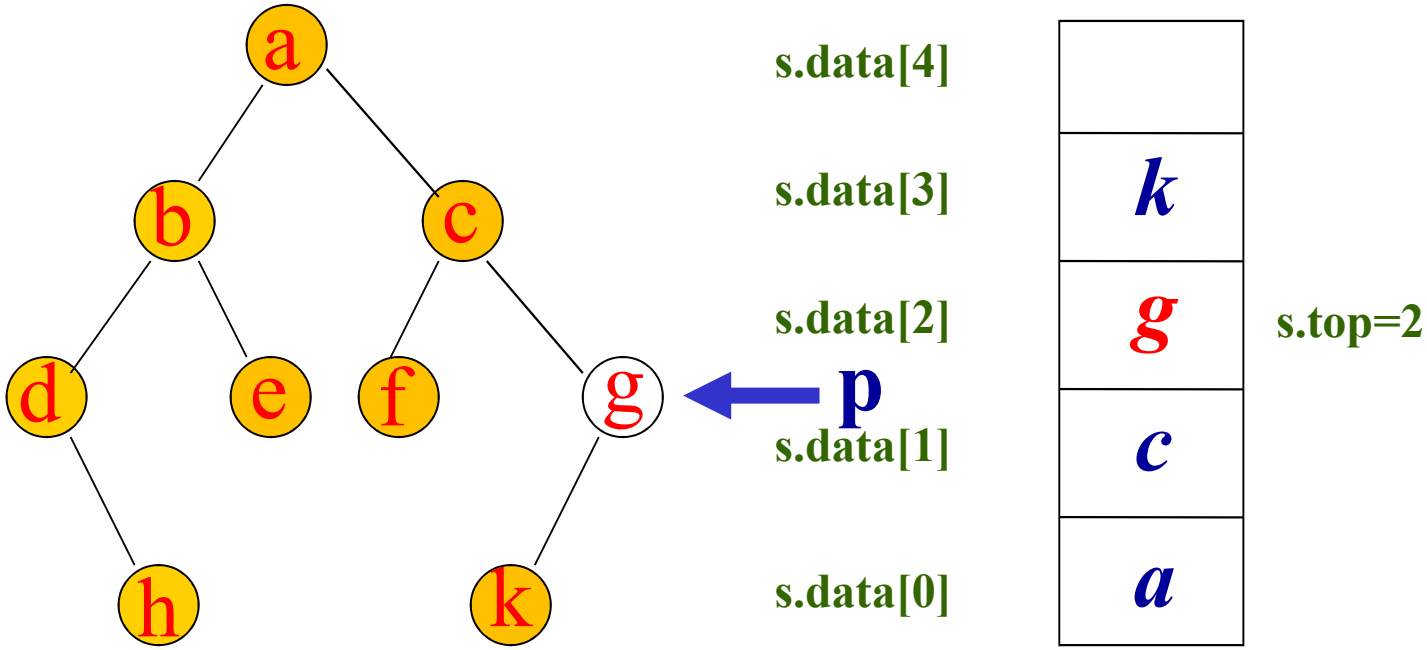
二叉树后序遍历的非递归算法实现

hdebfk



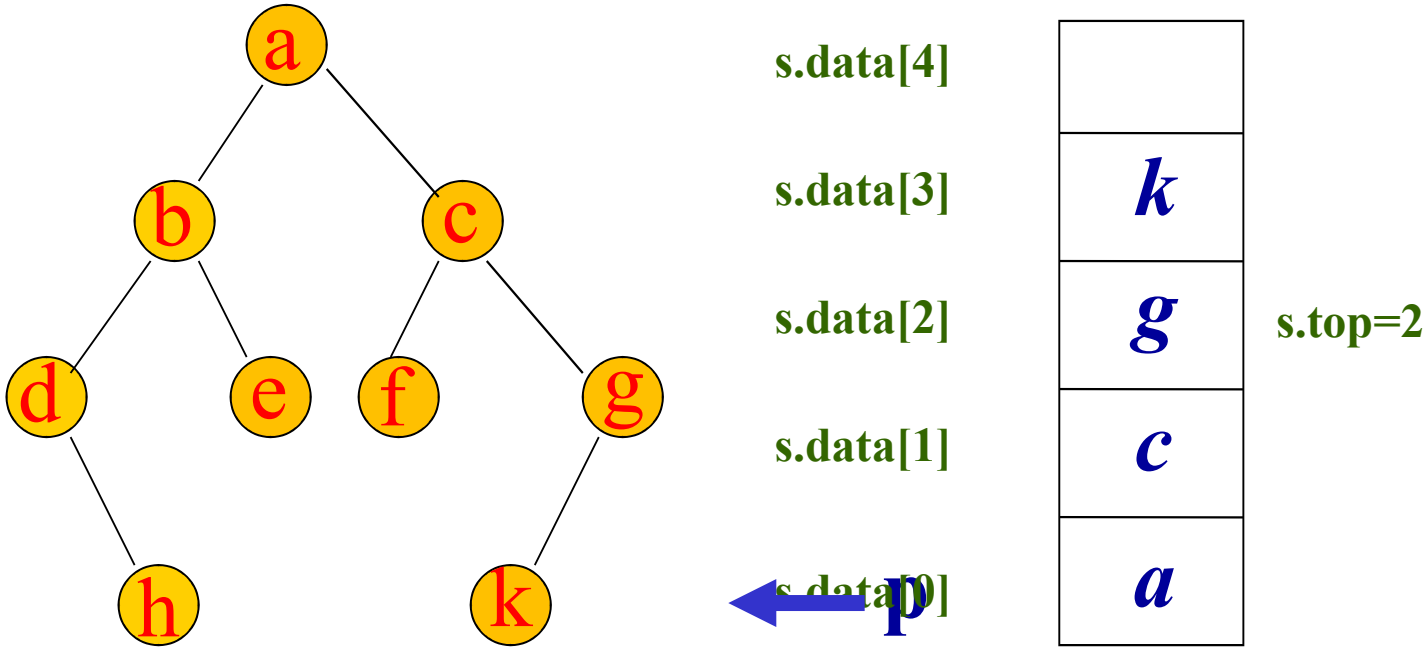
二叉树后序遍历的非递归算法实现

hdebfk



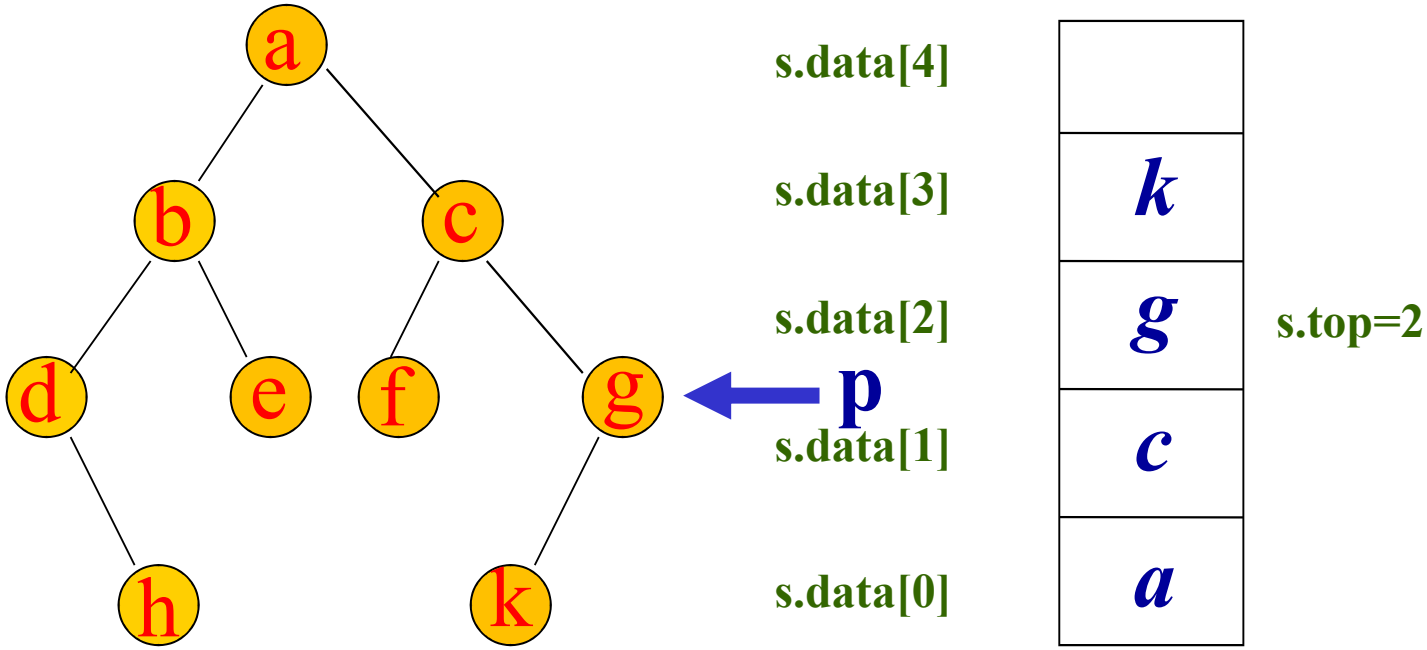
二叉树后序遍历的非递归算法实现

hdebfk



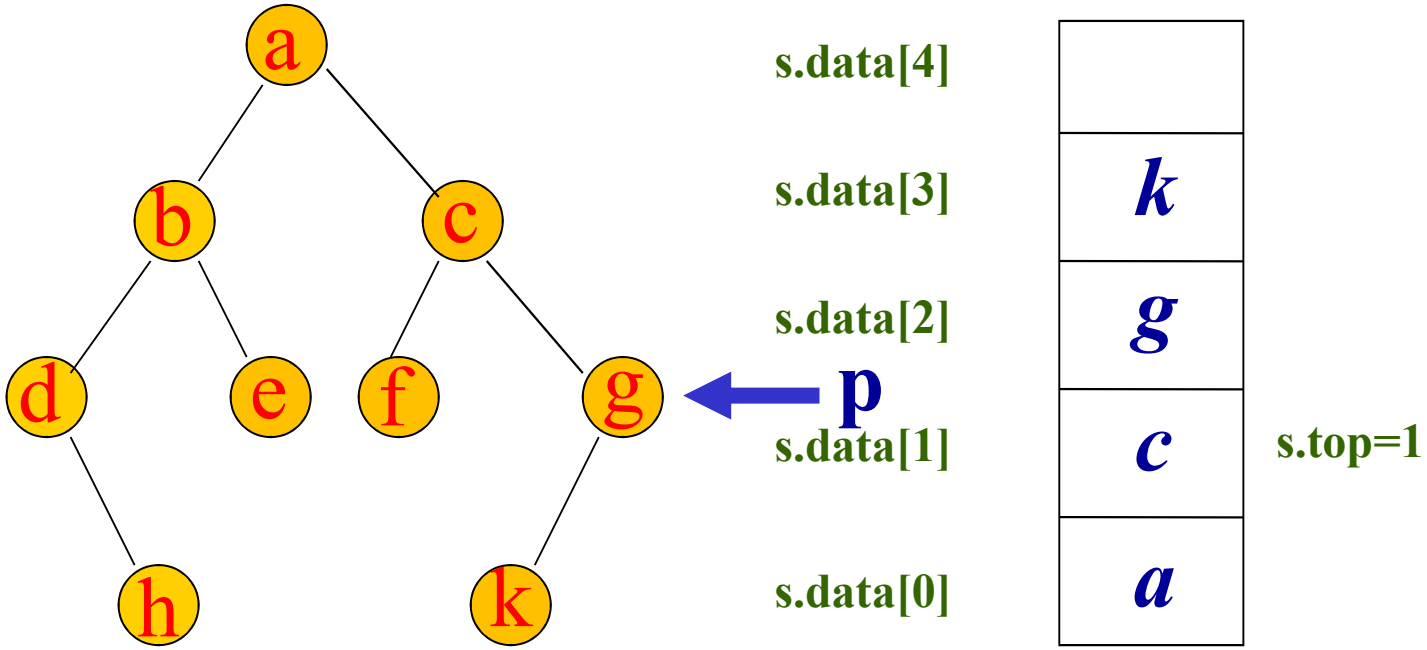
二叉树后序遍历的非递归算法实现

hdebfk



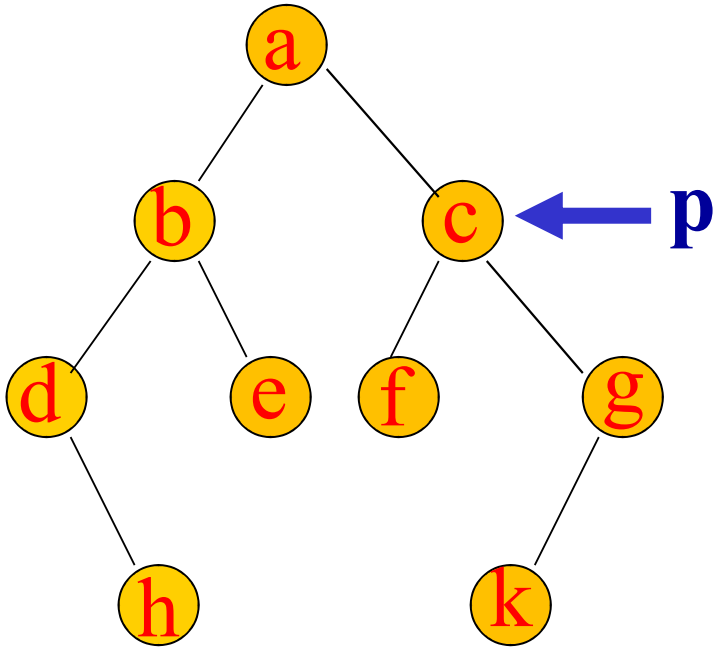
二叉树后序遍历的非递归算法实现

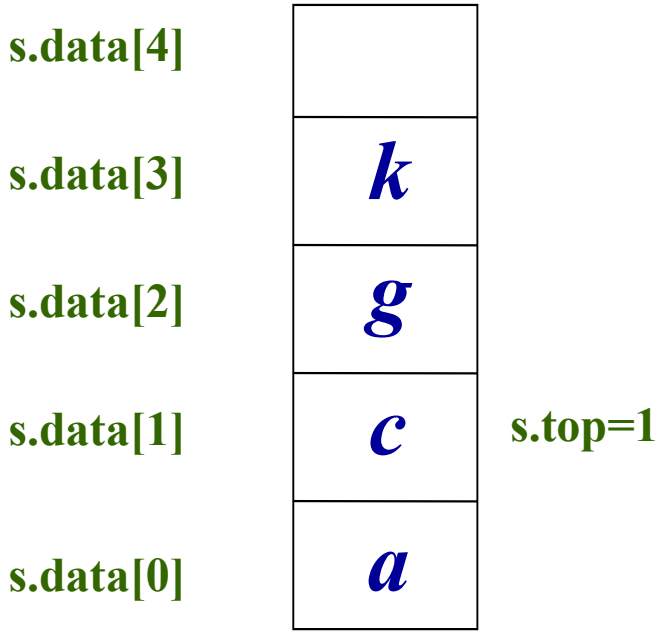
hdebfkg



二叉树后序遍历的非递归算法实现

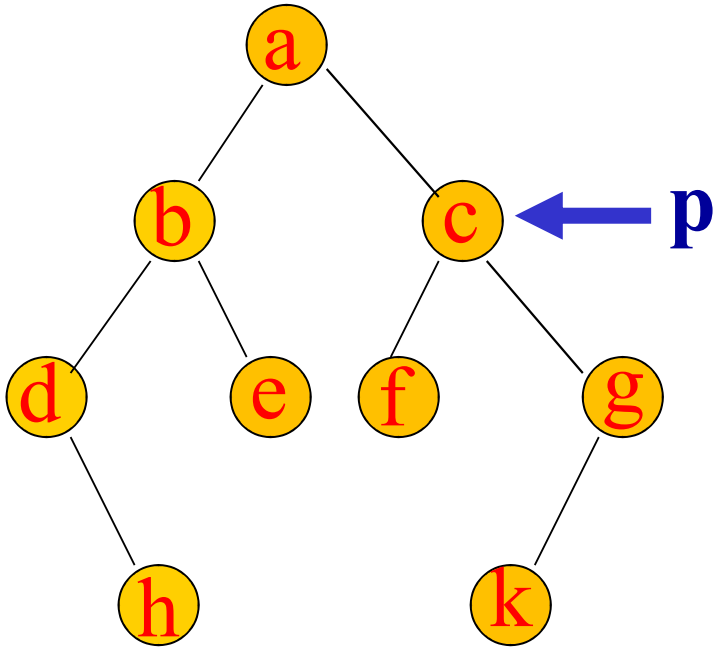
hdebfkg

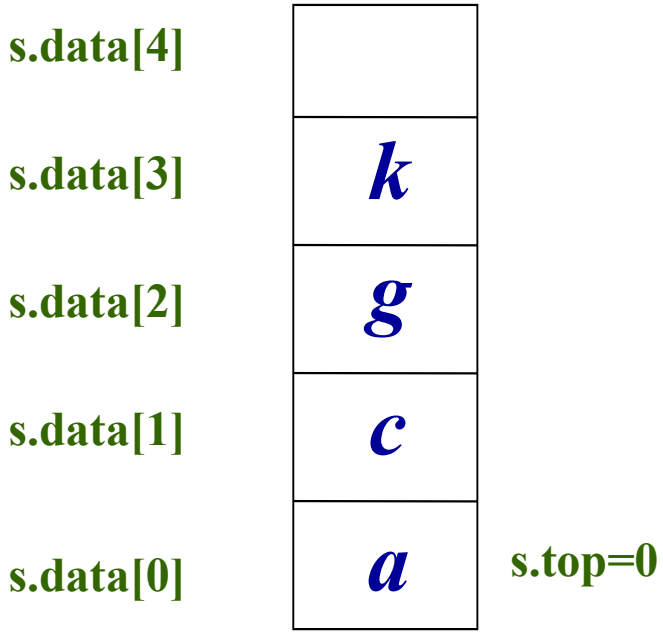




二叉树后序遍历的非递归算法实现

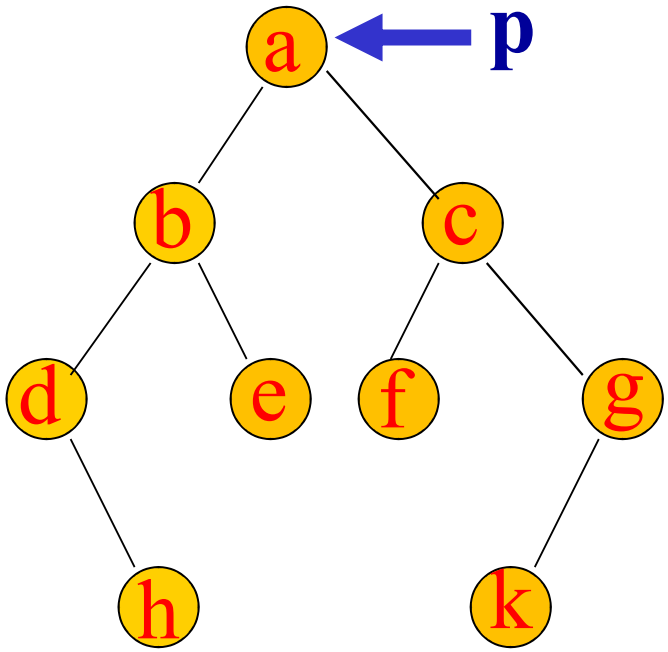
hdebfkgc

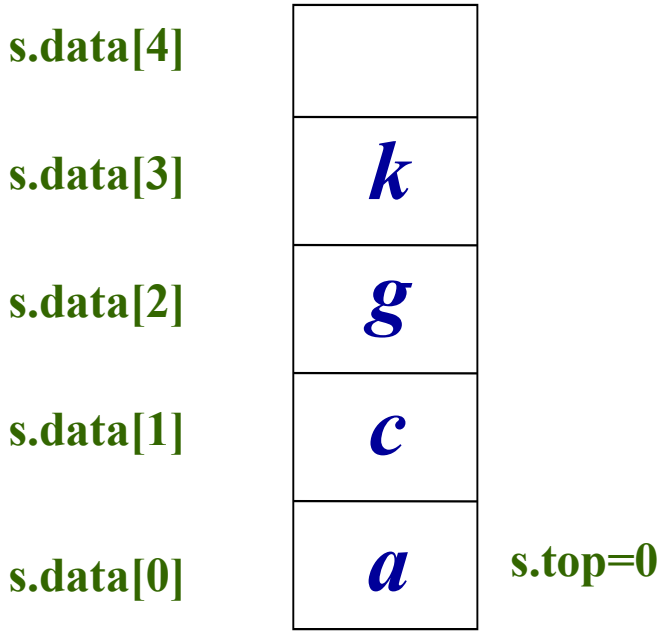




二叉树后序遍历的非递归算法实现

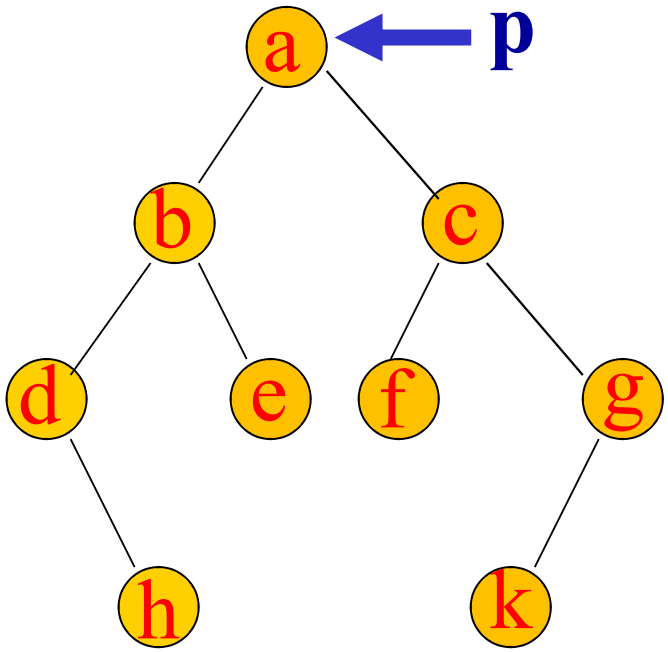
hdebfkgc





二叉树后序遍历的非递归算法实现

hdebfkgca



s.data[4]	
s.data[3]	*k*
s.data[2]	*g*
s.data[1]	*c*
s.data[0]	*a*

s.top=-1

二叉树后序遍历的非递归算法实现

顺序栈

```
#define MAX  10000
typedef  struct {BiTree d; int flag;}dataelem;
typedef  struct
  {dataelem data[MAX];
   int  top;
  }SeqStack2;
```

后序遍历的非递归描述

```
void postorder(BiTree T)
{SeqStack2 s;
  s.top=-1;
  p=T;
 do{while(p!=NULL)   {s.data[++s.top].d=p; s.data[s.top].flag=0; p=p->lc;}
      while((s.top>-1)&&(s.data[s.top].flag==1))
      { p=s.data[s.top--].d;   printf("%c",p->data); }
      if(s.top>-1)
       {s.data[s.top].flag=1;p=s.data[s.top].d;  p=p->rc;}
  }while(s.top>-1);
}
```