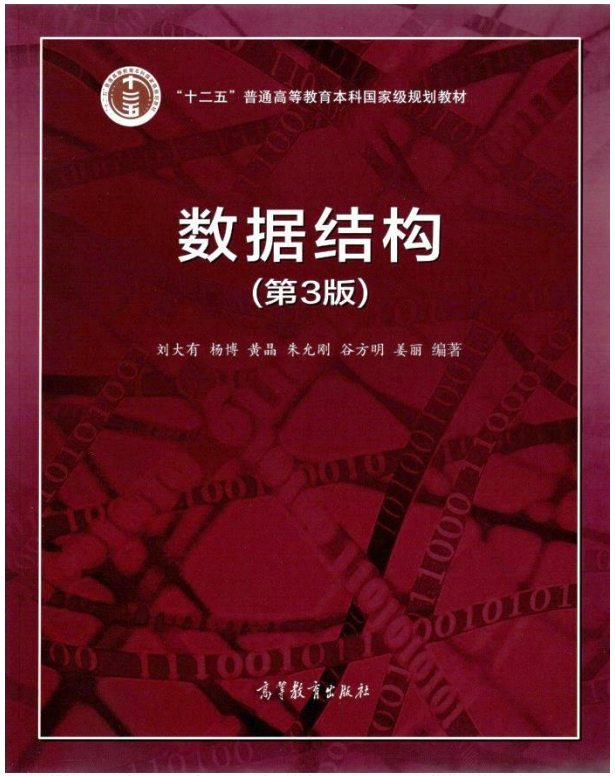


二叉树的路径与分层遍历

- 二叉树的路径
- 二叉树的分层遍历

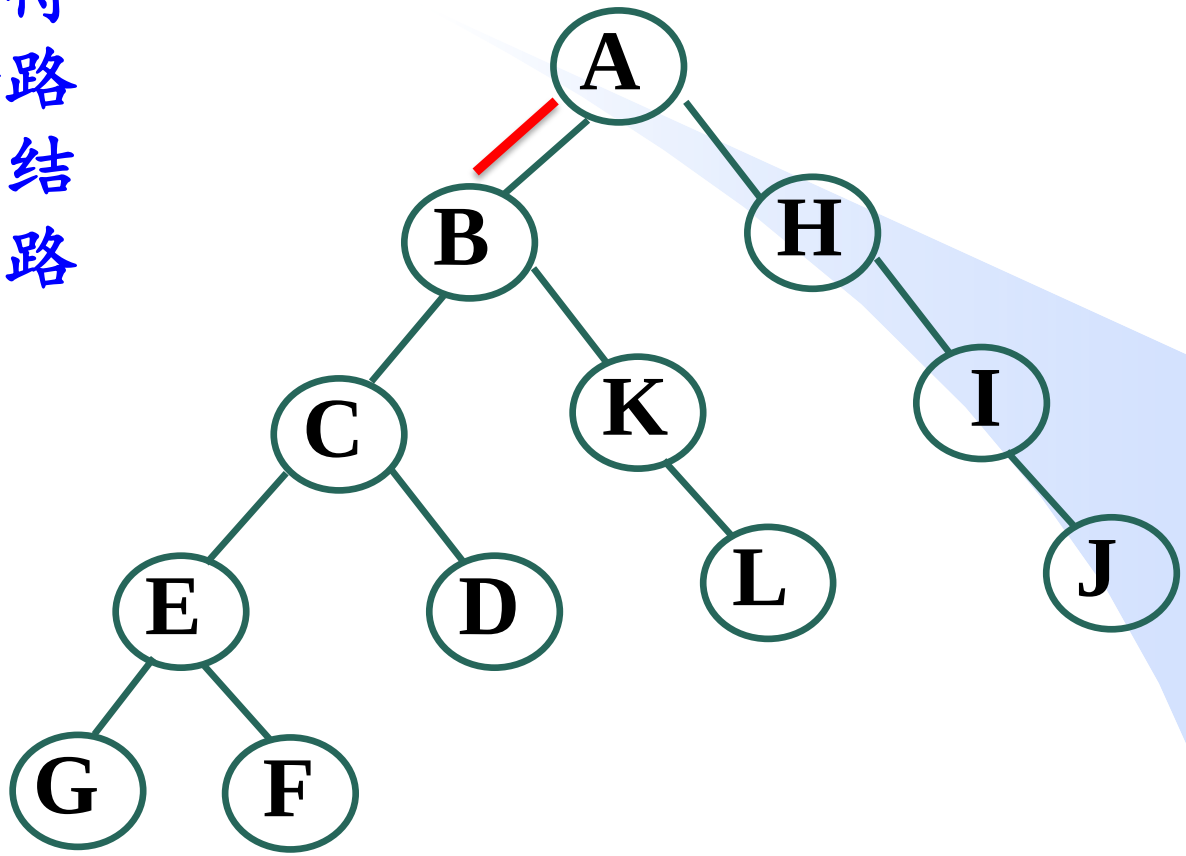
数据之法
结构之美
算法之道

Last updated on 2022.11

zhuyungang@jlu.edu.cn

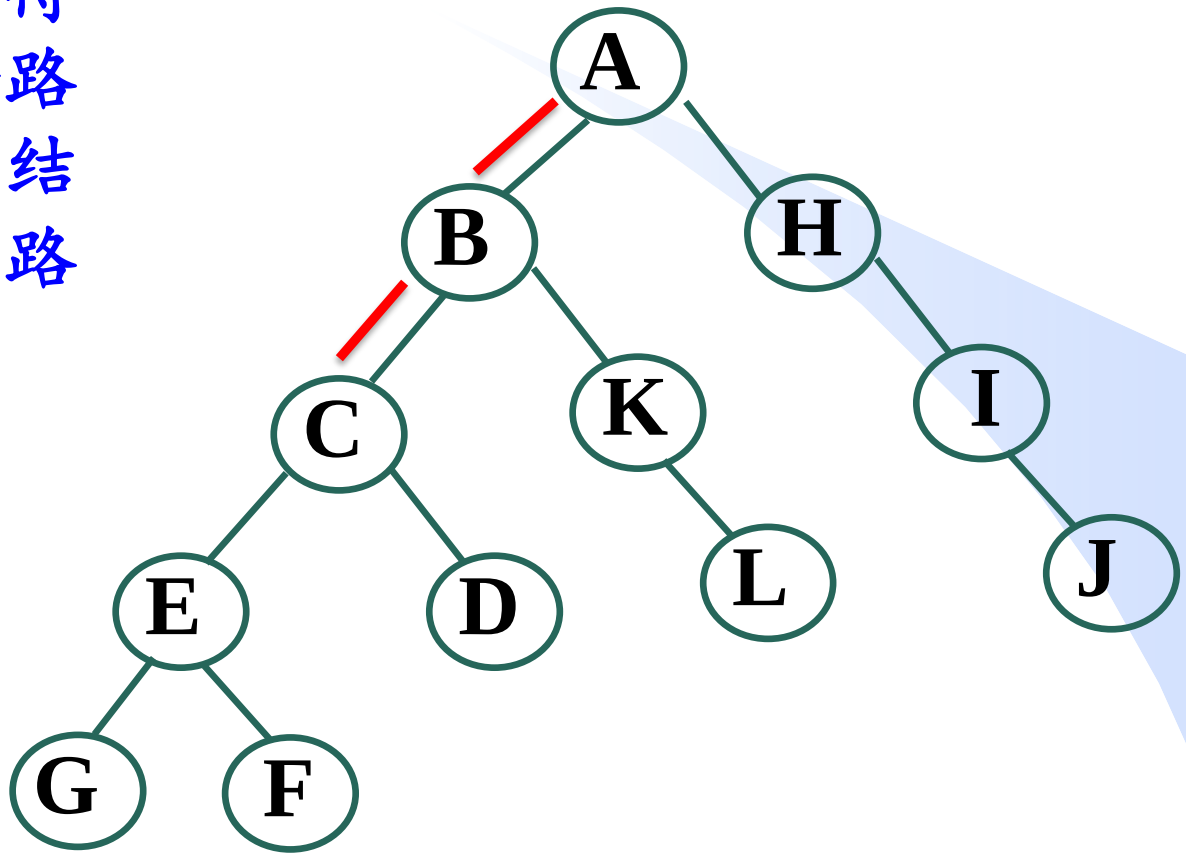
二叉树的路径

先根遍历，保存沿途结点，即得到从根到当前访问结点的一条路径。递归深度对应当前访问的结点的层数，亦对应当前得到的路径的长度。



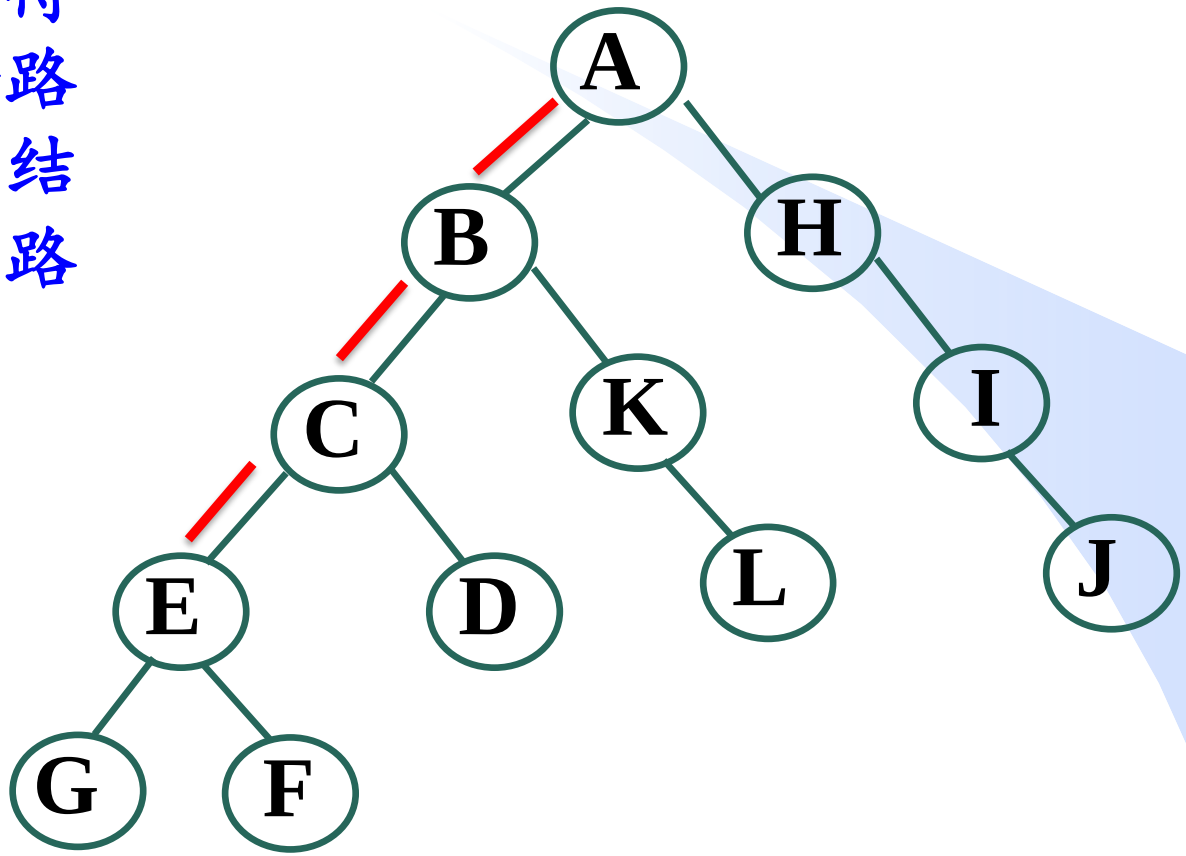
二叉树的路径

先根遍历，保存沿途结点，即得到从根到当前访问结点的一条路径。递归深度对应当前访问的结点的层数，亦对应当前得到的路径的长度。



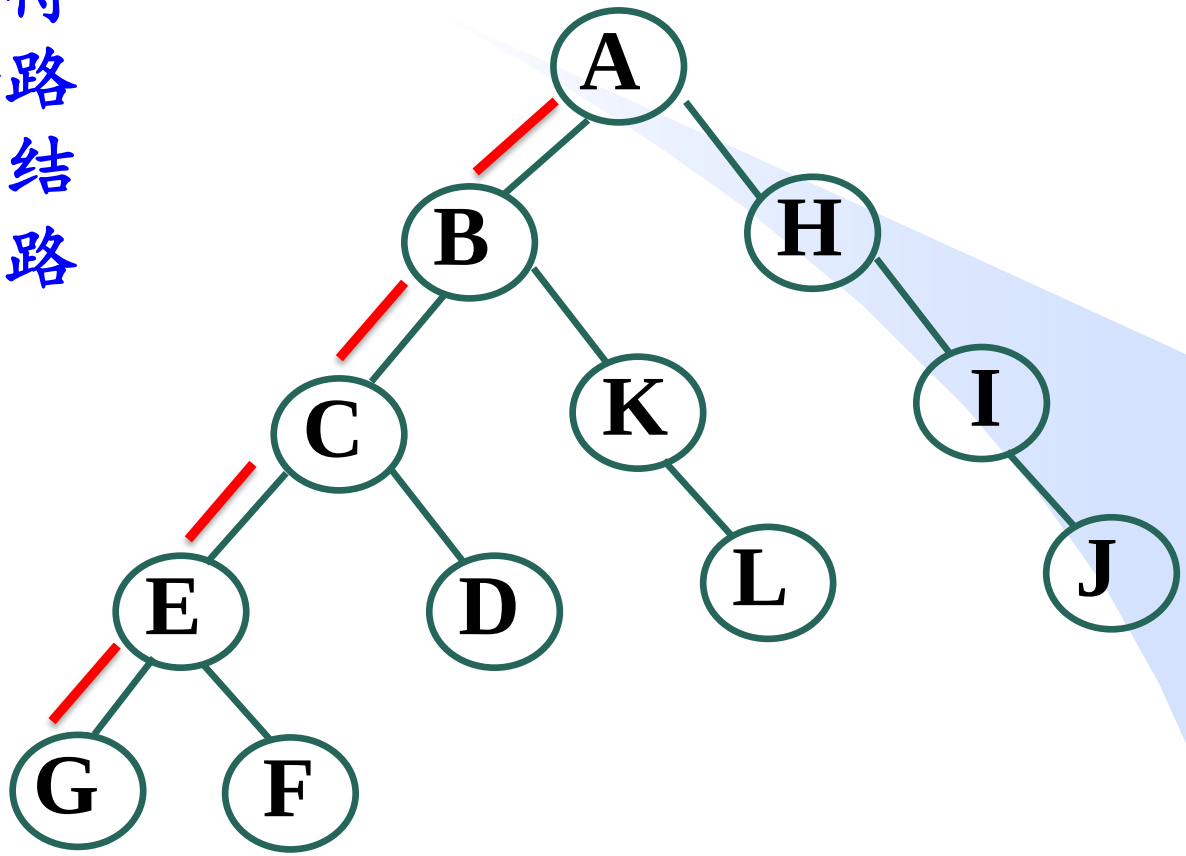
二叉树的路径

先根遍历，保存沿途结点，即得到从根到当前访问结点的一条路径。递归深度对应当前访问的结点的层数，亦对应当前得到的路径的长度。



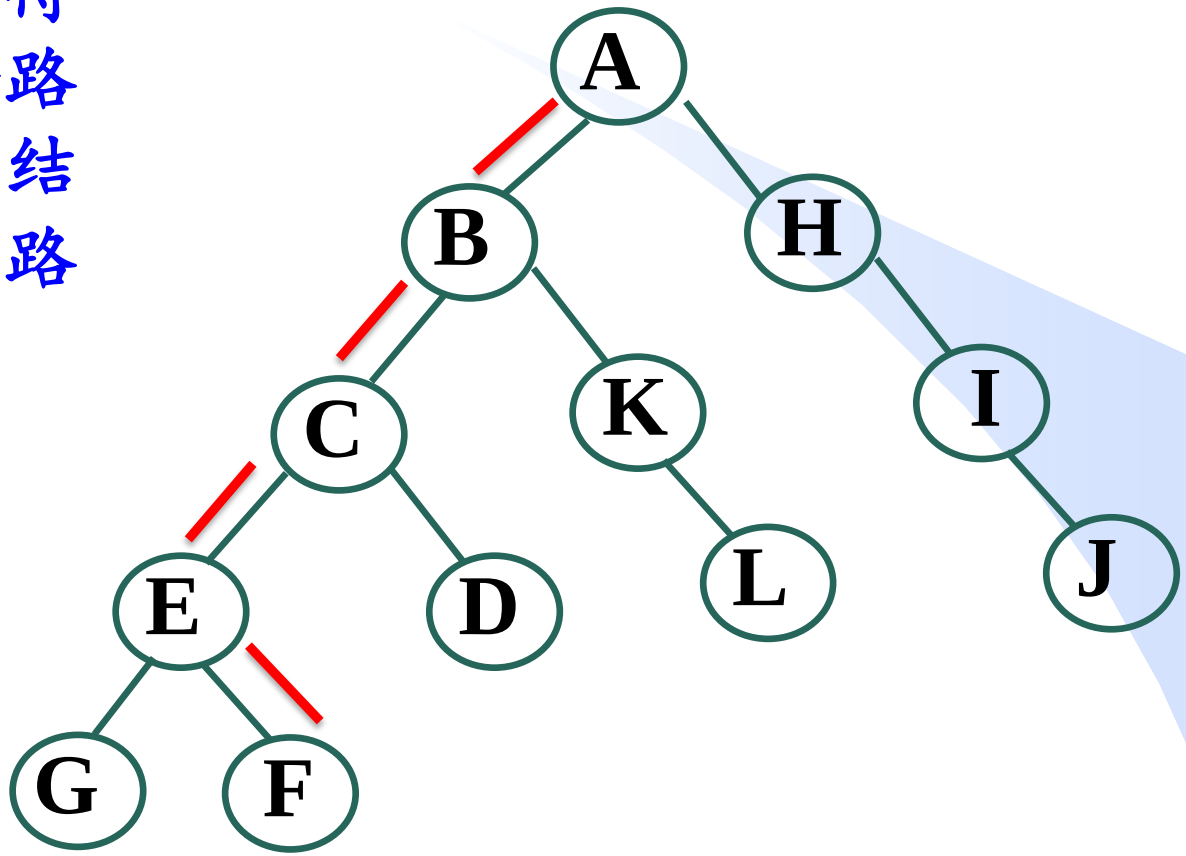
二叉树的路径

先根遍历，保存沿途结点，即得到从根到当前访问结点的一条路径。递归深度对应当前访问的结点的层数，亦对应当前得到的路径的长度。



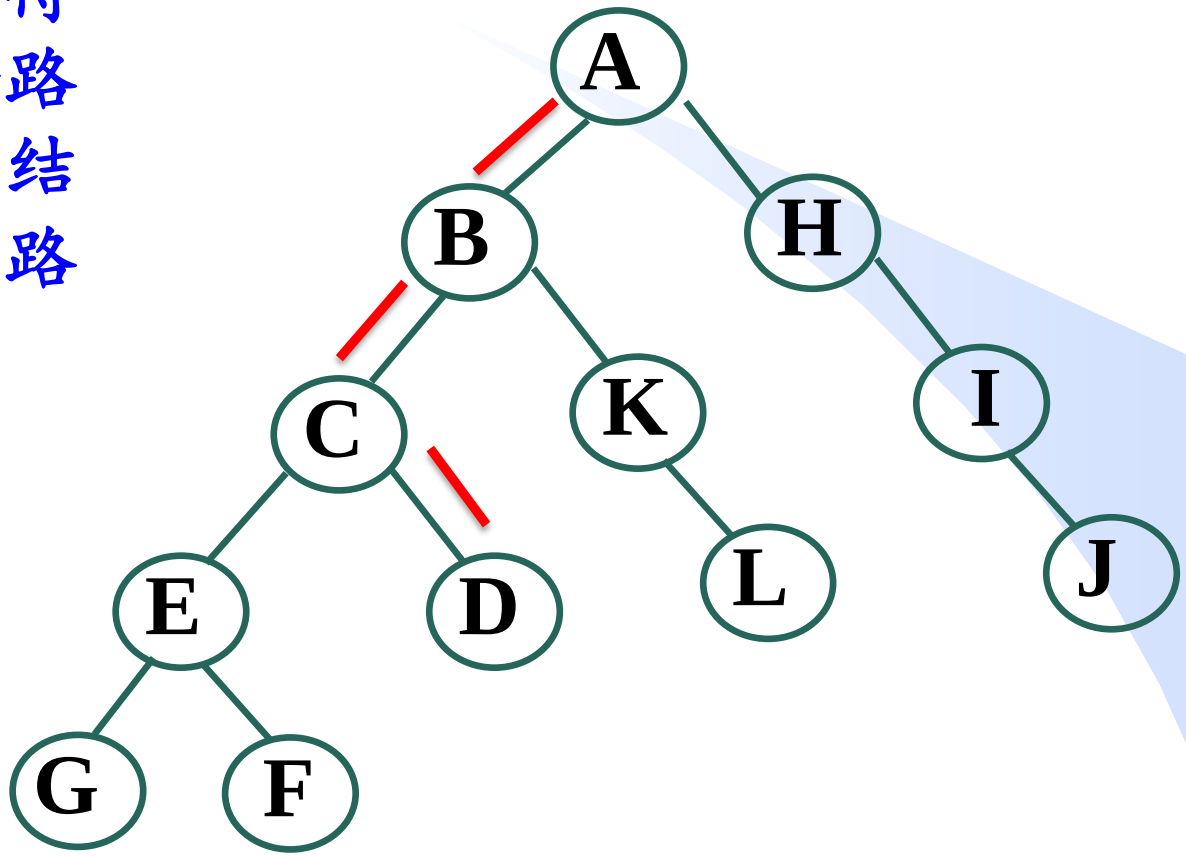
二叉树的路径

先根遍历，保存沿途结点，即得到从根到当前访问结点的一条路径。递归深度对应当前访问的结点的层数，亦对应当前得到的路径的长度。



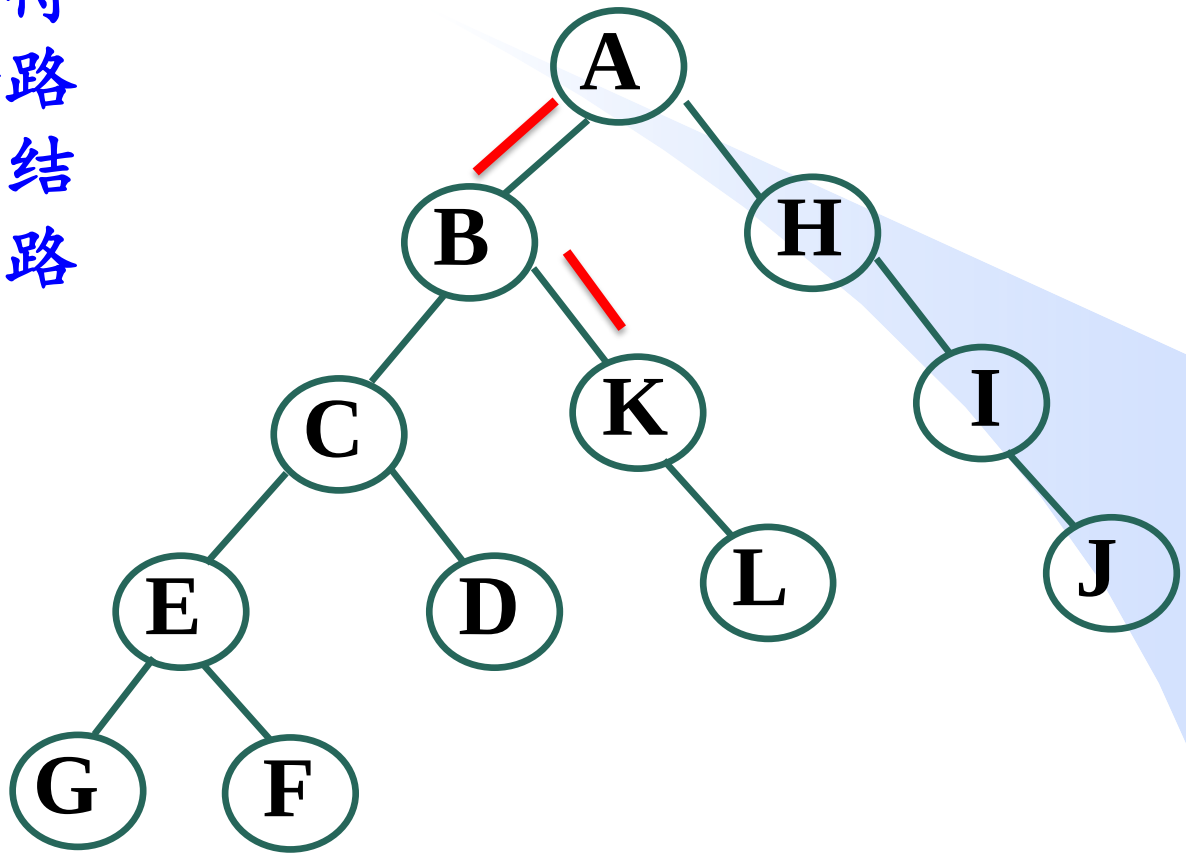
二叉树的路径

先根遍历，保存沿途结点，即得到从根到当前访问结点的一条路径。递归深度对应当前访问的结点的层数，亦对应当前得到的路径的长度。



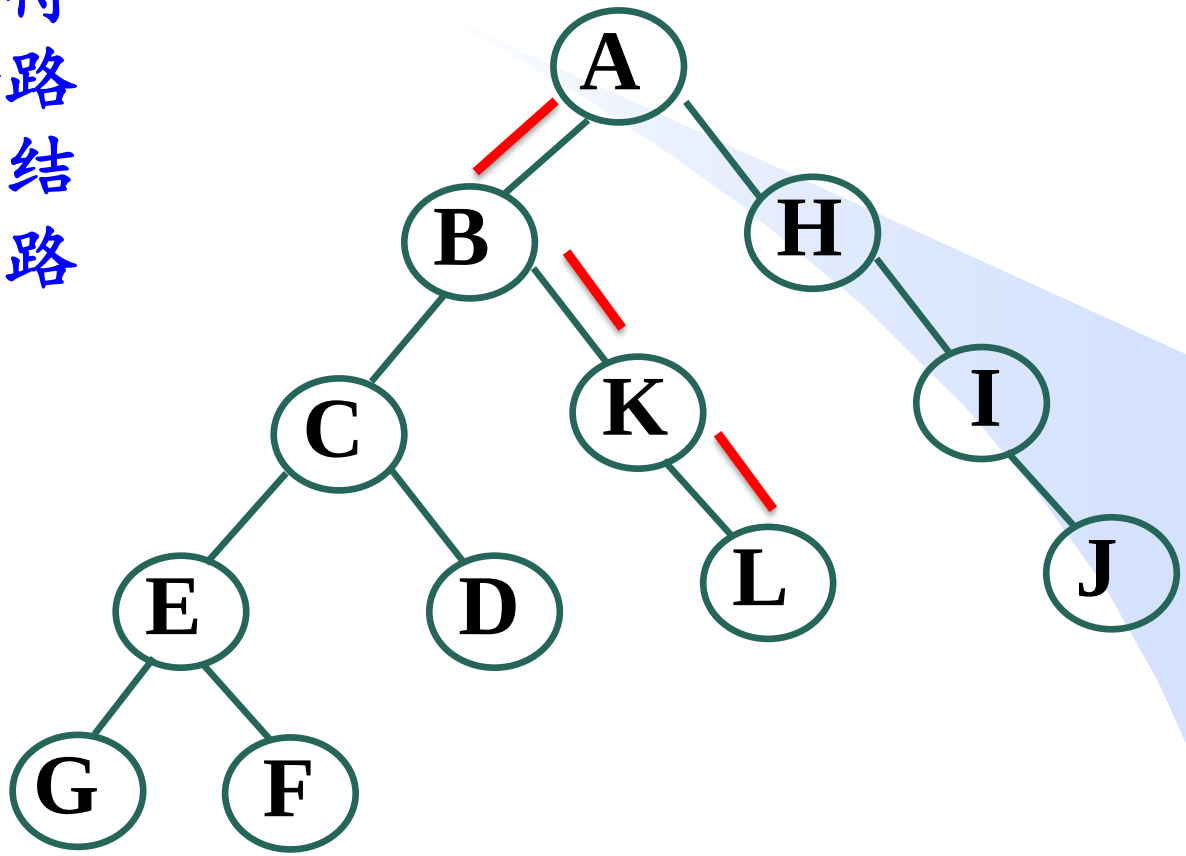
二叉树的路径

先根遍历，保存沿途结点，即得到从根到当前访问结点的一条路径。递归深度对应当前访问的结点的层数，亦对应当前得到的路径的长度。



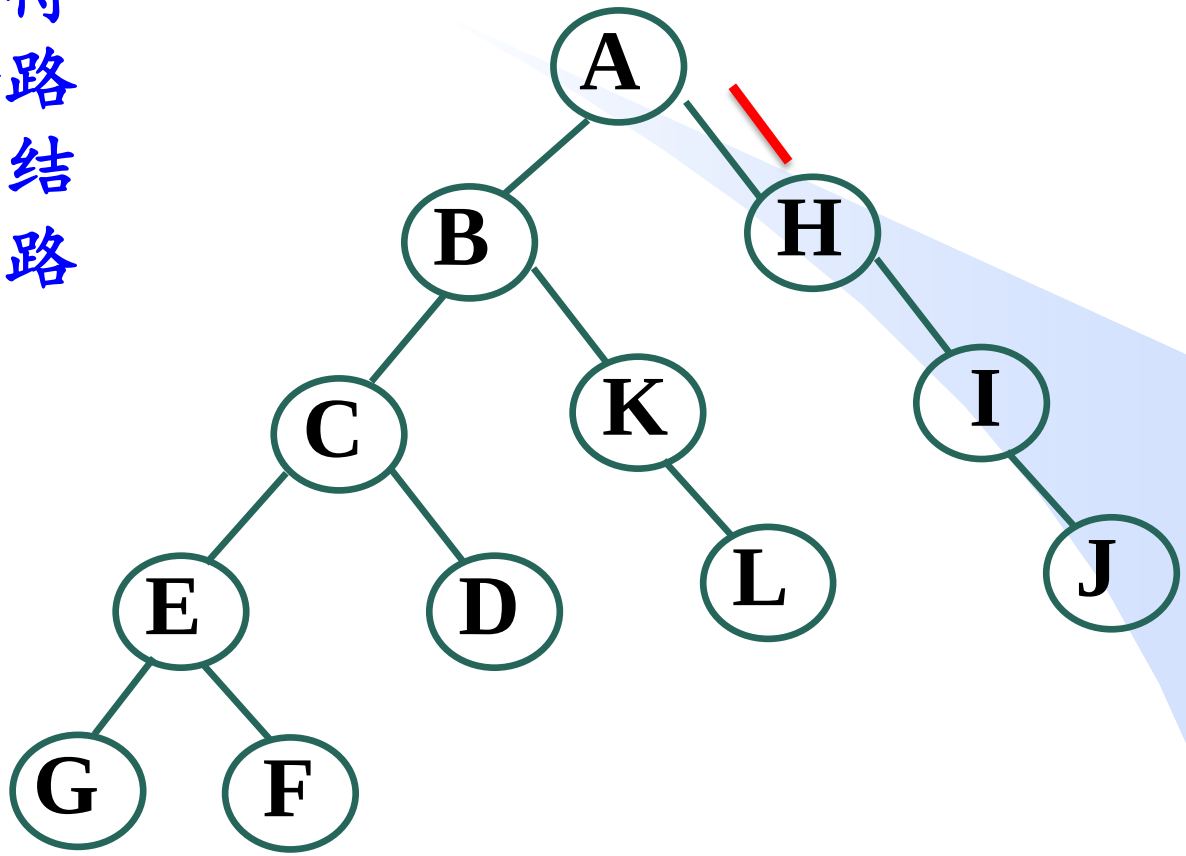
二叉树的路径

先根遍历，保存沿途结点，即得到从根到当前访问结点的一条路径。递归深度对应当前访问的结点的层数，亦对应当前得到的路径的长度。



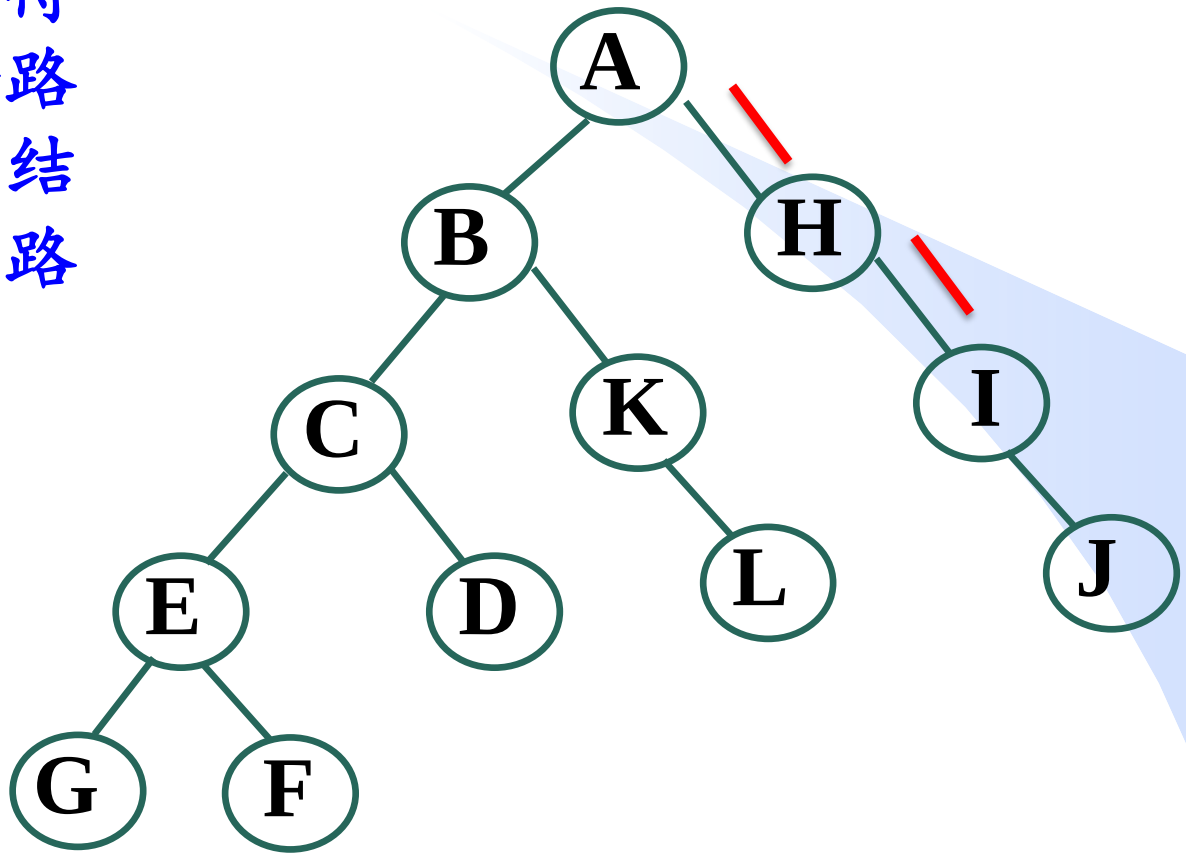
二叉树的路径

先根遍历，保存沿途结点，即得到从根到当前访问结点的一条路径。递归深度对应当前访问的结点的层数，亦对应当前得到的路径的长度。



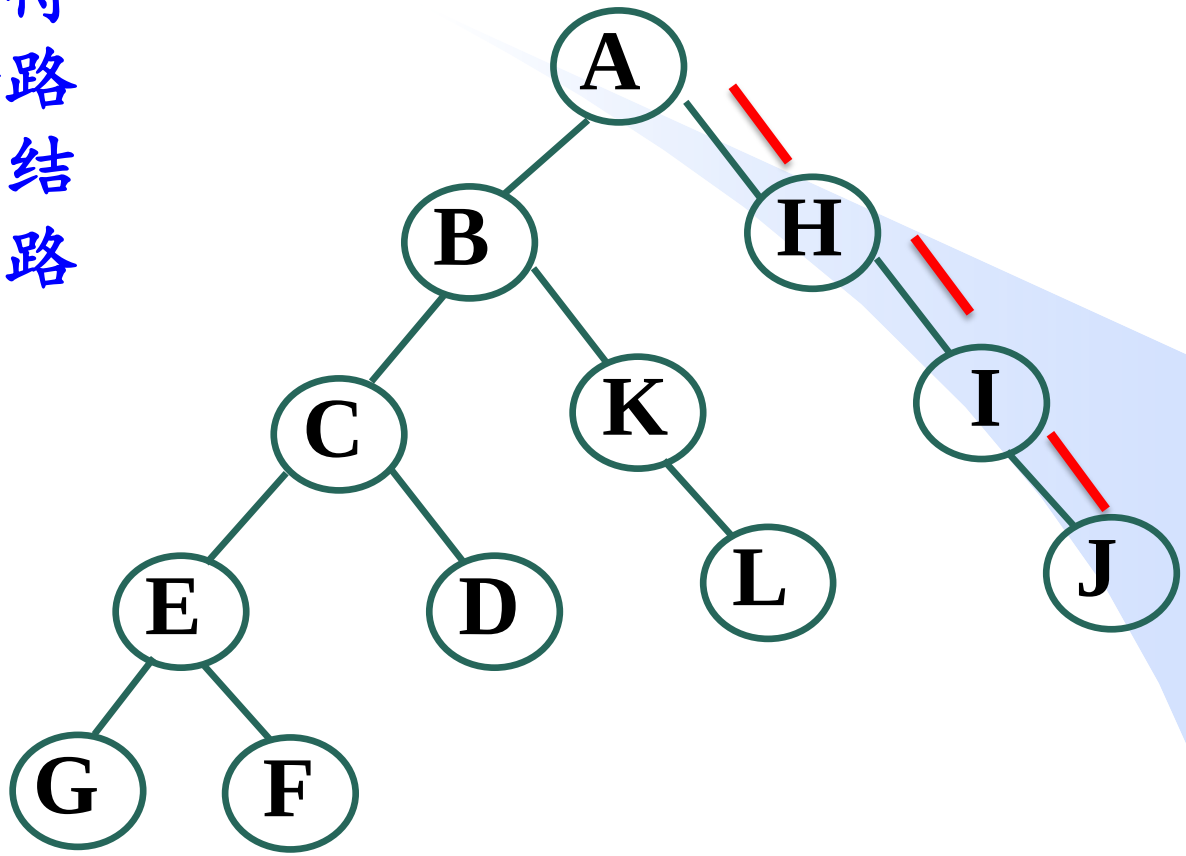
二叉树的路径

先根遍历，保存沿途结点，即得到从根到当前访问结点的一条路径。递归深度对应当前访问的结点的层数，亦对应当前得到的路径的长度。



二叉树的路径

先根遍历，保存沿途结点，即得到从根到当前访问结点的一条路径。递归深度对应当前访问的结点的层数，亦对应当前得到的路径的长度。



二叉树的路径

```
bool findPath(node *t, int path[], int k, int x, int &len){
   //找从根到x的一条路径存入path数组，len为路径长度，k为递归深度
   path[k] = t->data;
   if(t->data == x) { len = k; return true;}
   if(t->left != NULL)
      if(findPath(t->left, path, k+1, x, len)) return true;
   if(t->right != NULL)
      if(findPath(t->right, path, k+1, x, len)) return true;
   return false;
}
初始调用 findPath(root, path, 0, x, len);//保证root不为空
```

二叉树的路径

```
bool findPath(node *t, int path[], int k, int x, int &len){
   //找从根到x的一条路径存入path数组，len为路径长度，k为递归深度
   if(t==NULL) return false;
   path[k] = t->data;
   if(t->data == x) { len = k; return true;}
   if(findPath(t->left, path, k+1, x, len)) return true;
   if(findPath(t->right, path, k+1, x, len)) return true;
   return false;
}
```

初始调用 findPath(root, path, 0, x, len);

课堂作业：找从根开始的最长路径

```
void findpath(node *t,int path[],int k, int &max,int maxpath[])
{
    int i;
    if(t==NULL) return;
    path[k]=t->data;

    if(t->left==NULL && t->right==NULL)
    {
        if(k > max)
        {
            max=k;
            for( i=0;i<=k;i++)
                maxpath[i]=path[i];
        }
        return;
    }

    findpath(t->left,path,k+1,max,maxpath);
    findpath(t->right,path,k+1,max,maxpath);
}
```

```
int path[100],maxpath[100];
int max=-1;
findpath(root,path,0,max,maxpath);
```

若多条路径满足条件，则输出最右边的路径。怎么办？

课堂作业：找从根开始的最长路径

```
void findpath(node *t,int path[],int k, int &max,int maxpath[])
{
    int i;
    if(t==NULL) return;
    path[k]=t->data;

    if(t->left==NULL && t->right==NULL)
    {
        if(k > max)
        {
            max=k;
            for( i=0;i<=k;i++)
                maxpath[i]=path[i];
        }
        return;
    }

    findpath(t->left,path,k+1,max,maxpath);
    findpath(t->right,path,k+1,max,maxpath);
}
```

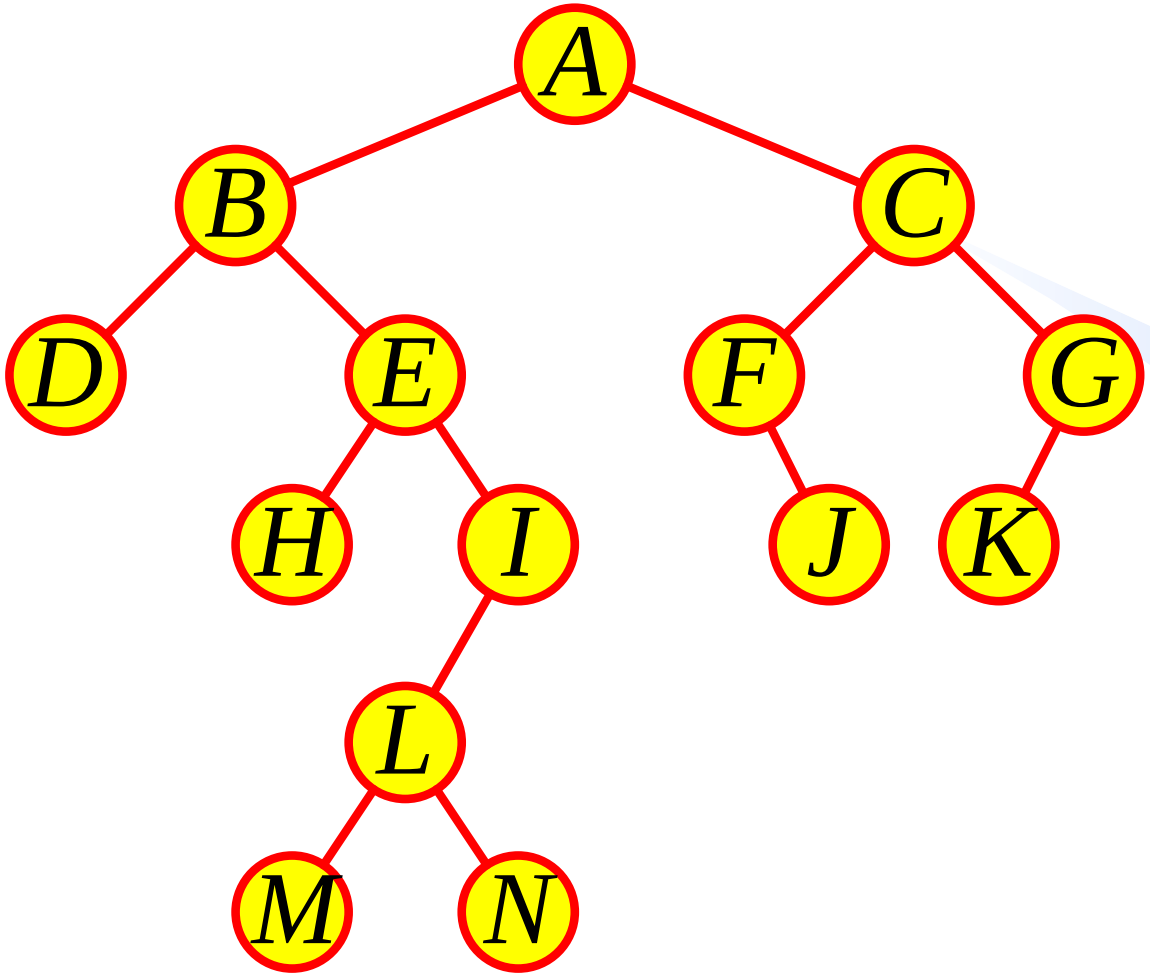
```
int path[100],maxpath[100];
int max=-1;
findpath(root,path,0,max,maxpath);
```

若多条路径满足条件，则输出最右边的路径。怎么办？

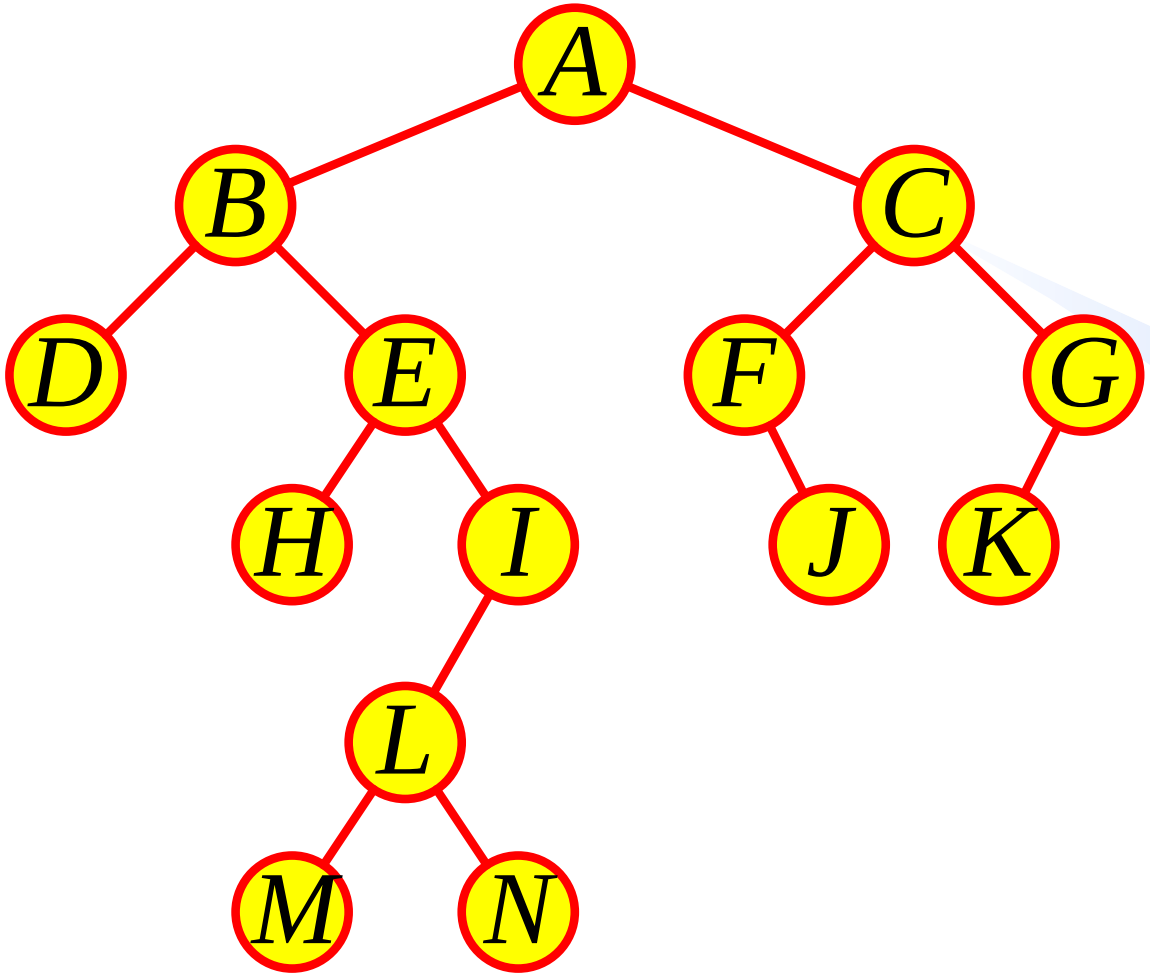
统计二叉树每层信息

例：统计一棵二叉树中每层叶结点的数目。

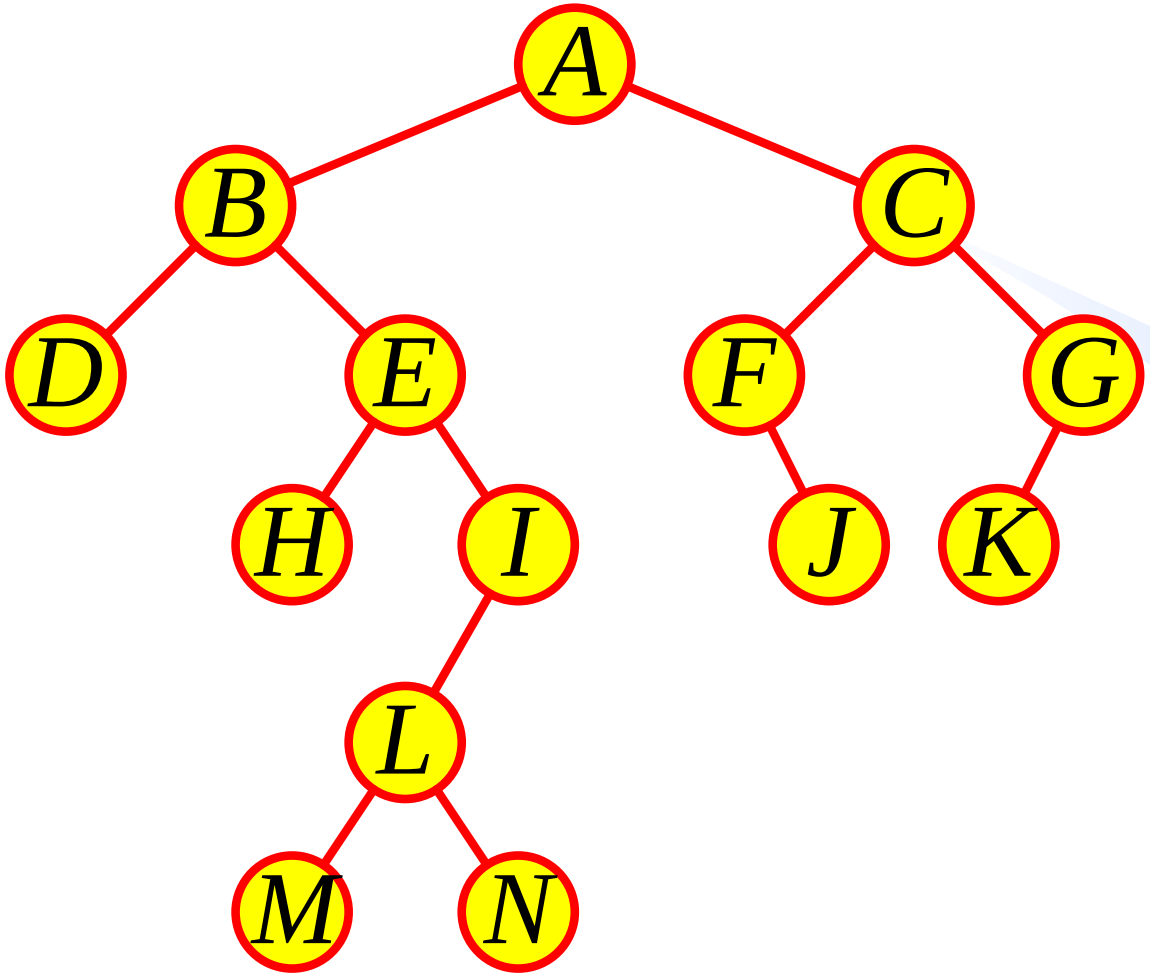
- 层次遍历，需要识别每层的结束（每层结束后，输出该层信息）。
- 可以设置一个不同于队列中其他结点的特殊结点（比如空结点NULL）来表示每层的结束。
- 遍历第一层时，先将根结点入队，再将NULL入队。在遍历过程中，当NULL出队时，表示已经遍历完一层，将NULL再入队，此时NULL即为下一层的结尾。



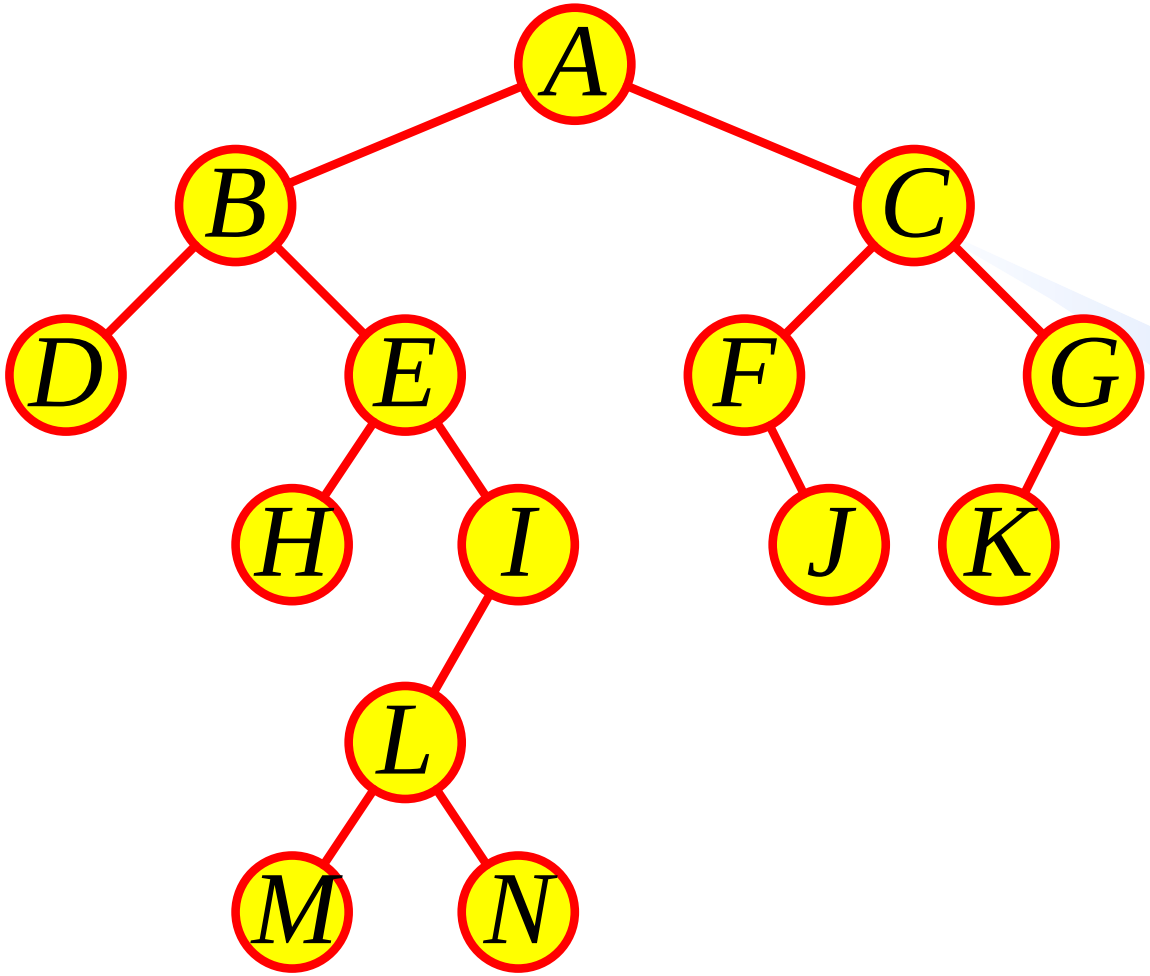
A	Λ



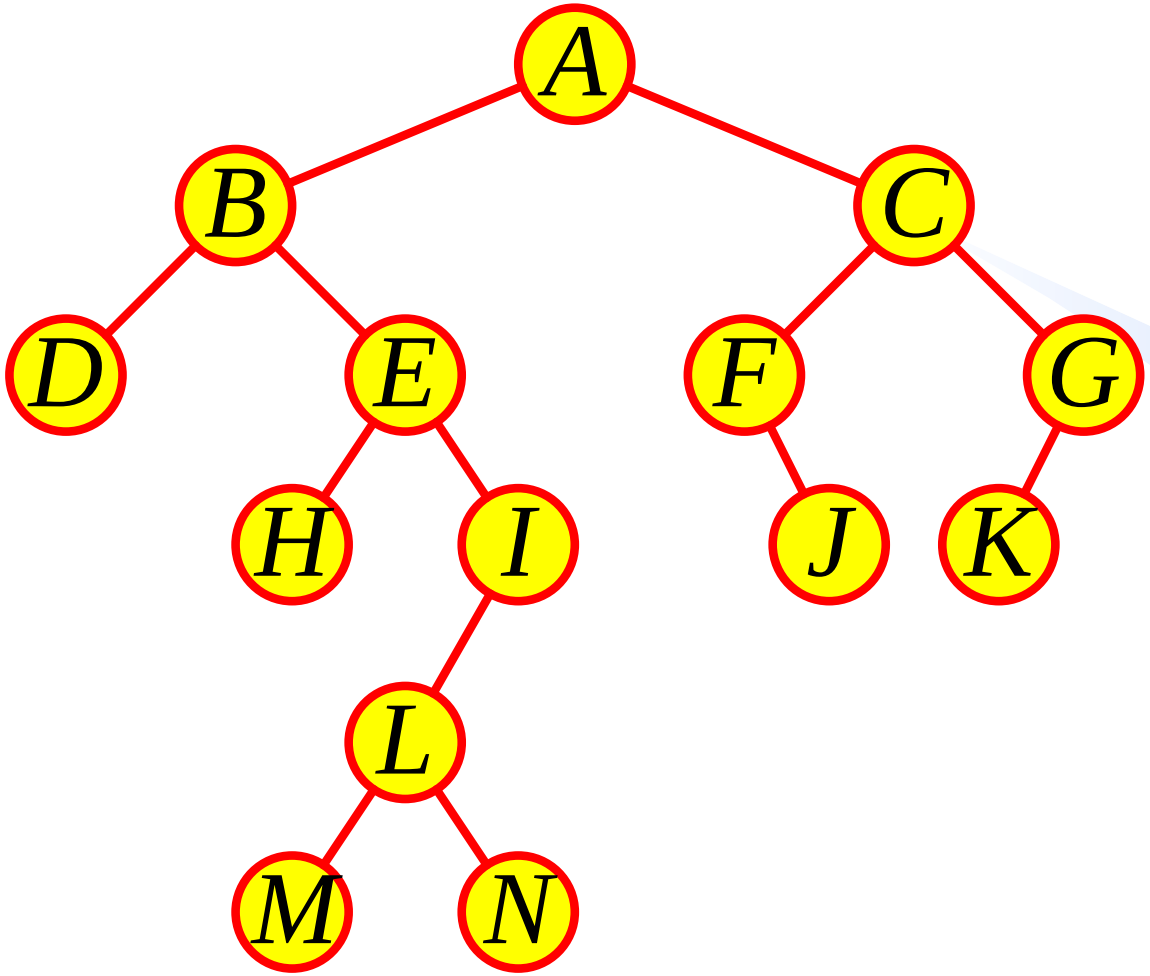
Λ	**B**	**C**



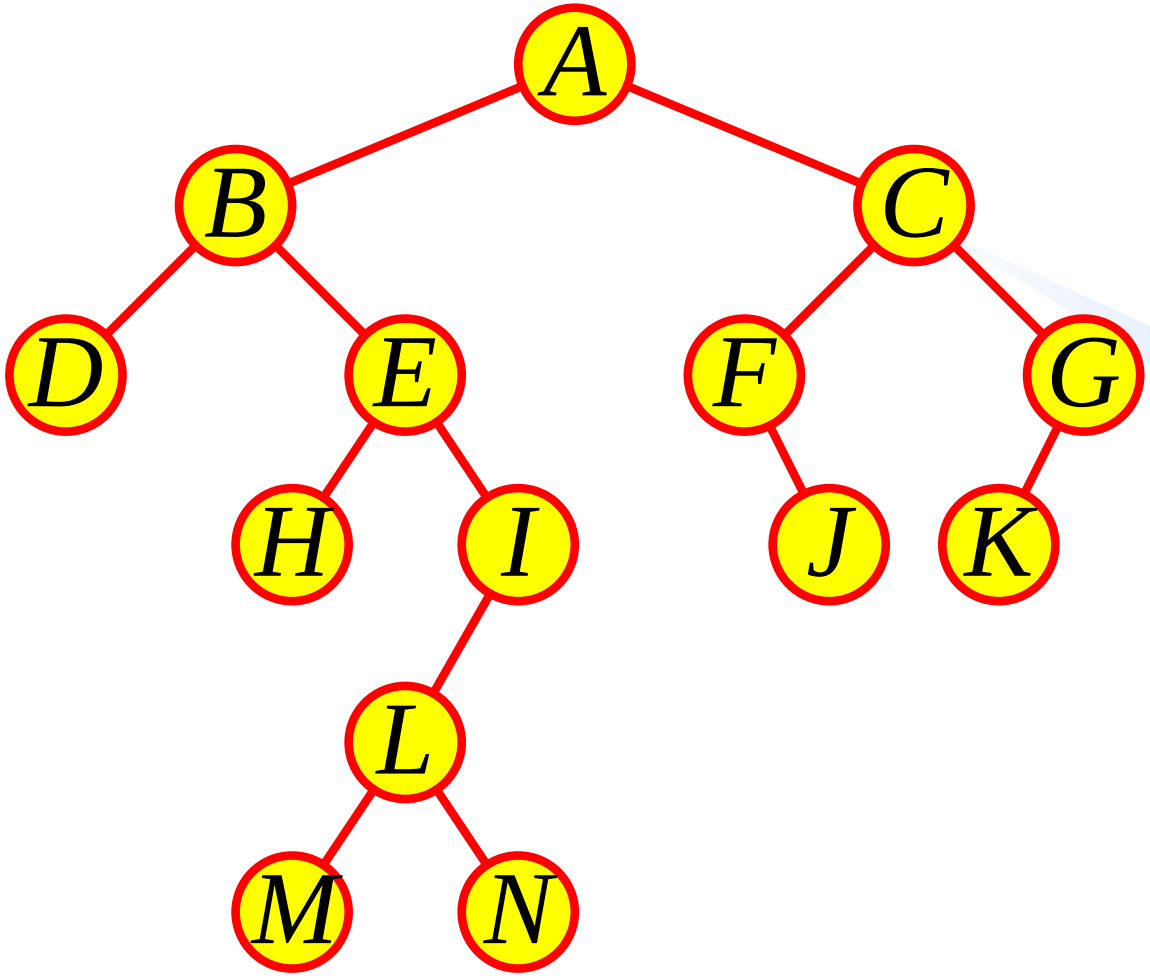
B	**C**	Λ



C	Λ	**D**	**E**



Λ	D	E	F	G



D	**E**	**F**	**G**	Λ

```
void LeafInEachLevel (Node *root){
     if(root==NULL) return;
     Queue q ; int level=0, cnt=0;
     q.ENQUEUE(root);                                  //根和NULL入队
     while (!q.IsEmpty()){
          Node* p=q.DEQUEUE(); //出队一个结点
                                                叶结点
                                                level,c
                                                     ++;
               if(p->left!=NULL) q.ENQUEUE(p->left);
               if(p->right!=NULL) q.ENQUEUE(p->right);

     }
};
```

```
void LeafInEachLevel (Node *root){
    if(root==NULL) return;
    Queue q ; int level=0, cnt=0;
    q.ENQUEUE(root);     q.ENQUEUE(NULL); //根和NULL入队
    while (!q.IsEmpty()){
        Node* p=q.DEQUEUE(); //出队一个结点
        if(p==NULL) {              //本层结束，输出叶结点数
            printf("第%d层包含%d个叶结点。\n",level,cnt);
            level++;    cnt=0;
            if(!q.IsEmpty()) q.ENQUEUE(NULL);
        }
        else{ if(p->left==NULL && p->right==NULL) cnt++;
            if(p->left!=NULL) q.ENQUEUE(p->left);
            if(p->right!=NULL) q.ENQUEUE(p->right);
        }
    }
};
```

统计每层的信息（另一种方案）

采用先根遍历，用变量k标识递归深度，每进入一层递归，k加1，递归深度k其实就是当前结点所在的层数，用数组记录第k层的所有结点，先根遍历的顺序即从左到右的顺序。这样任意某层的所有结点都可以得到，无论统计什么信息，都很容易了。