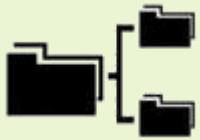
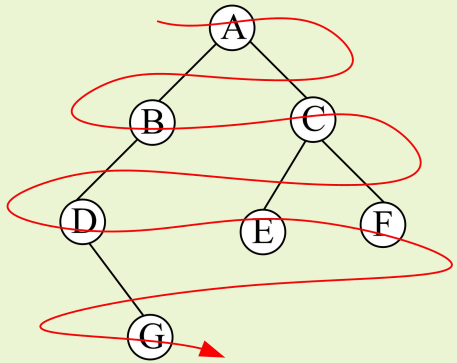


本节主题：

层次遍历算法

层次遍历

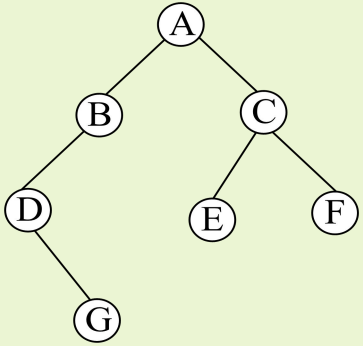
- 做法
  - 逐层进行访问
  - 对某一层的节点访问完后，再按照其访问次序对各个节点的左、右孩子顺序访问
- 数据结构设计
  - 先访问的节点，其左、右孩子也要先访问
  - 先进先出
  - 用队列实现
  - 用环形队列
- 层次遍历过程
  - 先将根节点进队
  - 节点出队并访问，再将左右孩子入队
  - 循环，直到队空



算法实现

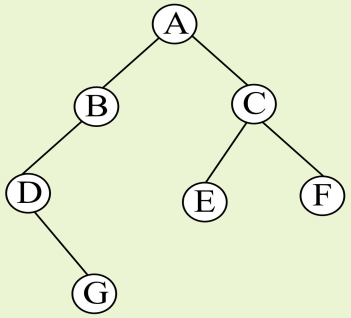
```
void LevelOrder(BTNode *b)
{
   BTNode *p;
   BTNode *qu[MaxSize];
   int front,rear;
   front=rear=-1;
   rear++;   //根节点入队
   qu[rear]=b;
   while (front!=rear)
   {
      //从队列中出列一个节点*p，访问它；
      //若它有左孩子节点，将左孩子节点进队；
      //若它有右孩子节点，将右孩子节点进队。
   }
}
```

```
front=(front+1)%MaxSize;
p=qu[front];
printf("%c ",p->data);
if (p->lchild!=NULL)
{
   rear=(rear+1)%MaxSize;
   qu[rear]=p->lchild;
}
if (p->rchild!=NULL)
{
   rear=(rear+1)%MaxSize;
   qu[rear]=p->rchild;
}
```



例：路径之逆

- 问题：二叉树采用二叉链存储结构，设计算法输出从根节点到每个叶子节点的路径之逆
- 解：本例采用层次遍历的思路解决。
  - 采用非环形顺序队列qu
  - 层次遍历二叉树
  - 将所有已访问过的节点指针进队，并在队列中保存双亲节点的位置。
  - 当找到一个叶子节点时，在队列中通过双亲节点的位置输出根节点到该叶子节点的路径之逆。



qu	node	parent
[0]	A	-1
[1]	B	0
[2]	C	0
[3]	D	1
[4]	E	2
[5]	F	2
[6]	G	3
[7]		
[8]		

算法实现

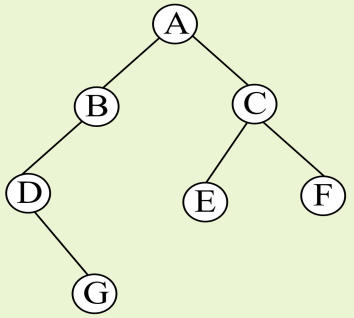
```
void AllPath2(BTNode *b)
{
   BTNode *q;
   //定义队列，根节点入队
   while (front!=rear) //队列不为空
   {
      front++;
      q=qu[front].node;
      //如果是叶节点，输出路径之逆
      //左孩子入队
      //右孩子入队
   }
}
```

```
struct snode
{
   BTNode *node;
   int parent;
} qu[MaxSize];
int front,rear,p;
front=rear=-1;
rear++;
qu[rear].node=b;
qu[rear].parent=-1;
```

```
if (q->lchild==NULL && q->rchild==NULL)
{
   p=front;
   while (qu[p].parent!=-1)
   {
      printf("%c->",qu[p].node->data);
      p=qu[p].parent;
   }
   printf("%c\n",qu[p].node->data);
}
```

```
if (q->rchild!=NULL)
{
   rear++;
   qu[rear].node=q->rchild;
   qu[rear].parent=front;
}
```

```
if (q->lchild!=NULL)
{
   rear++;
   qu[rear].node=q->lchild;
   qu[rear].parent=front;
}
```



	node	parent
[0]	A	-1
[1]	B	0
[2]	C	0
[3]	D	1
[4]	E	2
[5]	F	2
[6]	G	3
[7]		
[8]		

思考题

- 求一个节点的祖先节点有哪些方法？