7.3 图的遍历

- 给定一个图G=(V,E)和其中的任一顶点*v*，从顶点*v*出发，访问图G中的所有顶点而且每个顶点仅被访问一次，这一过程称为图的遍历。
- 为了避免同一顶点被访问多次，在遍历图的过程中，必须记下每个已访问过的顶点。
- 为此设一个辅助数组visited[]，用以标记顶点是否被访问过，其初态应为0（false）。一旦一个顶点*i*被访问，则visited[*i*]=1（true）。

7.3.1 深度优先遍历算法

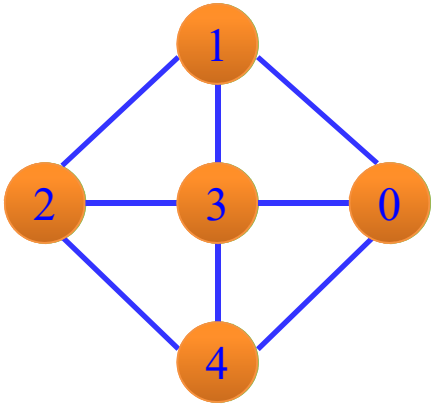
深度优先遍历（Depth First Search，简称DFS）：

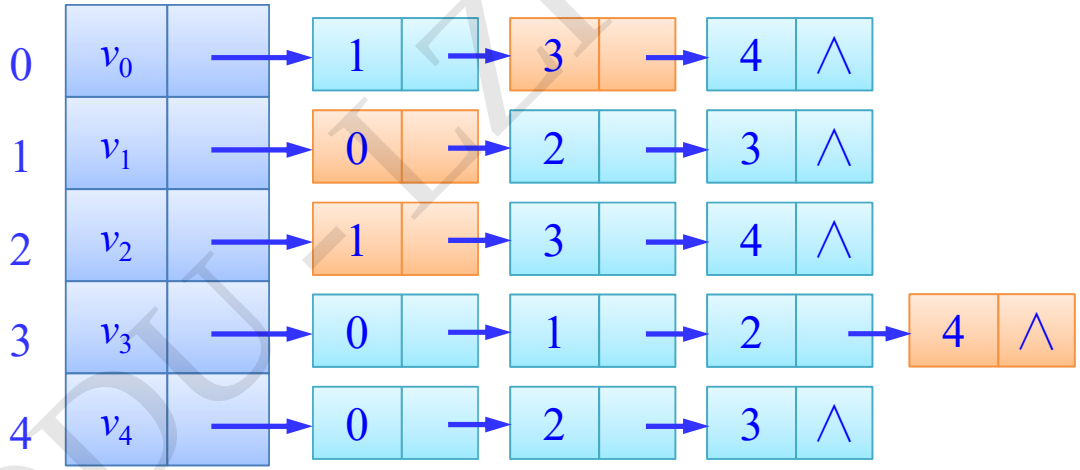
① 访问顶点v；

② 选择一个与顶点v相邻且没被访问过的顶点w，从w出发深度优先遍历。

③ 直到图中与v相邻的所有顶点都被访问过为止。

例如，对于下图的邻接表，从顶点2出发进行深度优先遍历。





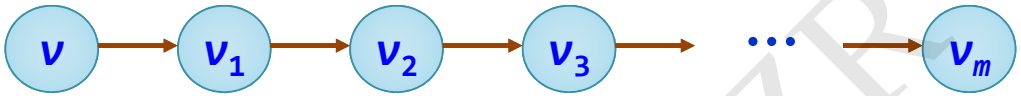
v=2的DFS序列：

2　　1　　0　　3　　4

实现深度优先遍历的递归算法如下：

```
visited[MAXVEX]={0};              //全局变量
void DFS(AdjGraph *G,int v)
{
   int w;
   ArcNode *p;
   printf("%d ",v);               //访问v顶点
   visited[v]=1;
   p=G->adjlist[v].firstarc;      //找v的第一个邻接点
   while (p!=NULL)                //找v的所有邻接点
   {    w= p->adjvex;             //顶点v的邻接点w
        if (visited[w]==0)        //顶点w未访问过
            DFS(G,w);             //从w出发深度优先遍历
        p=p->nextarc;             //找v的下一个邻接点
   }
}
```

DFS思路：



一步一步向前走，当没有可走的相邻顶点时便回退。

企业面试题：

无向图G=（V，E），其中

V={a,b,c,d,e,f}

E={(a,b),(a,e),(a,c),(b,e),(c,f),(f,d),(e,d)}

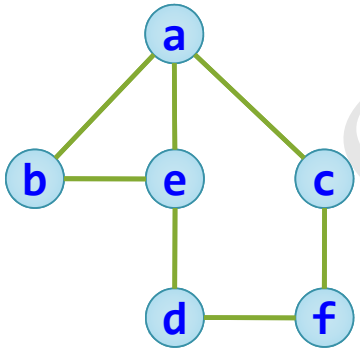
对该图进行深度优先排序，得到的顶点序列正确的是（ ）。

A. a，b，e，c，d，f

B. a，c，f，e，b，d

C. a，e，b，c，f，d

D. a，e，d，f，c，b



用R表示回退一次

A. a,a→b,b→e,e→d,不应该到c

B. a,a→c,c→f,f→d,不应该到e

C. a,a→e,e→b,R,e→d,不应该到c

D. a,a→e,e→d,d→f,f→c,R,R,R,e→b √

7.3.2 广度优先遍历算法

广度优先遍历（Breadth First Search，简称BFS）：

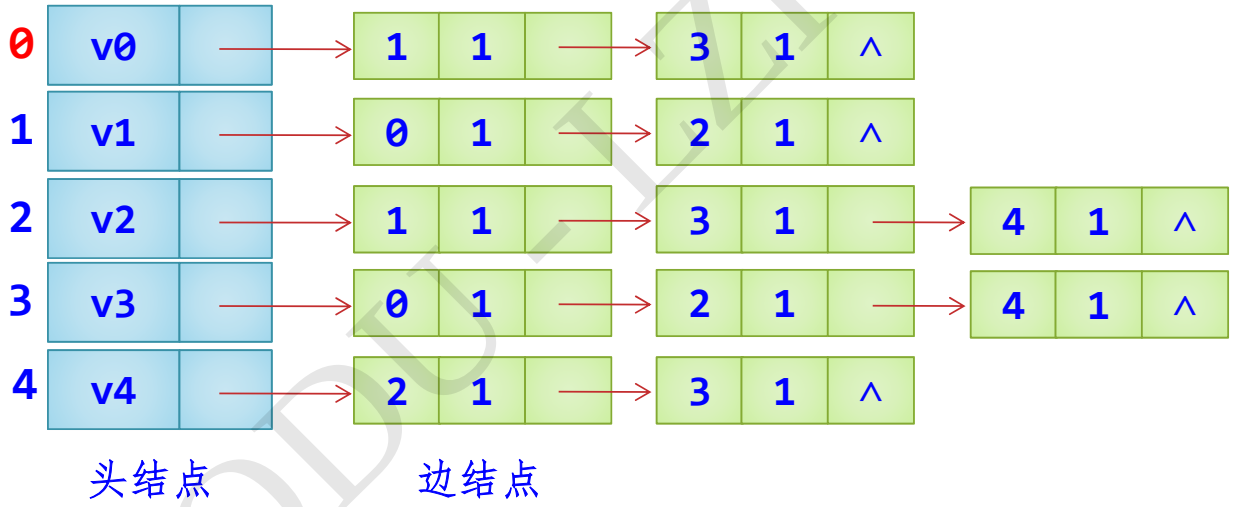
① 访问顶点v；

② 访问顶点v的所有未被访问过的邻接点，假设访问次序是v_{i1}，v_{i2}，…，v_{it}。

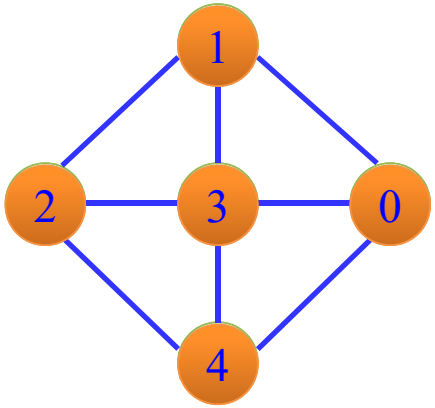
③ 按v_{i1}，v_{i2}，…，v_{it}的次序，访问每个顶点的所有未被访问过的邻接点，直到图中所有和初始点v有路径相通的顶点都被访问过为止。

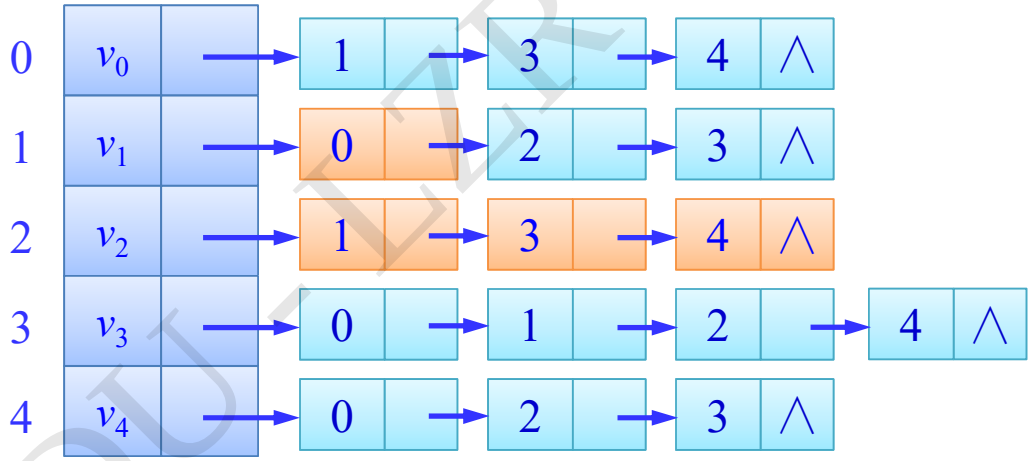
顺序一致，用队列实现

例如，现有邻接表，从顶点0出发的广度优先遍历序列是0、1、3、2、4。



例如，对于下图的邻接表，从顶点2出发进行广度优先遍历。





v=2的BFS序列：

2　　1　　3　　4　　0

实现广度优先搜索的算法如下：

```
void BFS(ALGraph *G,int vi)
{
   int i,v,visited[MAXVEX];   ArcNode *p;
   int Q[MAXVEX],front=0,rear=0;        //定义一个循环队列Q
   for (i=0;i<G->n;i++) visited[i]=0;  //visited数组置初值0
   printf("%d ",vi);                    //访问初始顶点
   visited[vi]=1;
   rear=(rear+1)%MAXVEX;Q[rear]=vi;     //初始顶点进队
   while (front!=rear)                  //队不为空时循环
   {  front=(front+1) % MAXVEX;
      v=Q[front];                       //出队顶点v
      p=G->adjlist[v].firstarc;         //查找v的第一个邻接点
      while (p!=NULL)                   //查找v的所有邻接点
      {  if (visited[p->adjvex]==0)     //未访问过则访问之
         {  printf("%d ",p->adjvex);    //访问该点并进队
            visited[p->adjvex]=1;
            rear=(rear+1) % MAXVEX;
            Q[rear]=p->adjvex;
         }
         p=p->nextarc;                  //查找v的下一个邻接点
      }
   }
}
```

END