8.3 图的遍历

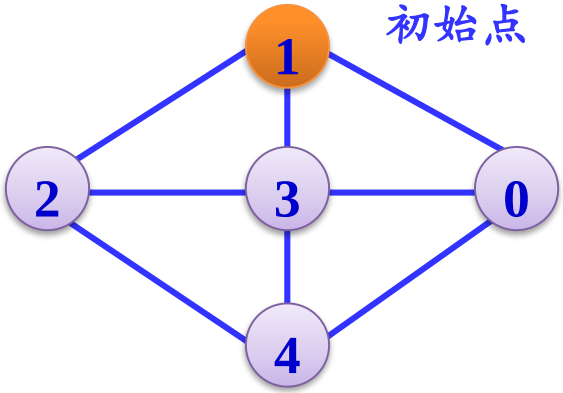
8.3.1 图的遍历的概念

从给定图中任意指定的顶点（称为初始点）出发，按照某种搜索方法沿着图的边访问图中的所有顶点，使每个顶点仅被访问一次，这个过程称为图的遍历。

图的遍历得到的顶点序列称为图遍历序列。

图中顶点之间是多对多的关系，而从一个顶点出发一次只能找另外一个相邻顶点。

例如：



从顶点1出发，访问顶点1，

❶再访问顶点2，4，…

❷再访问顶点2，3，0，…

不同的搜索方法

根据搜索方法的不同，图的遍历方法有两种：

- 深度优先遍历（DFS）。
- 广度优先遍历（BFS）。

8.3.2 深度优先遍历算法

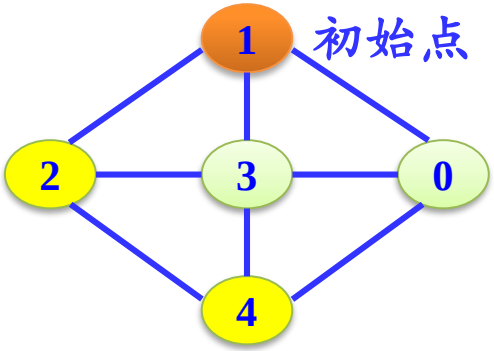
深度优先遍历过程：

（1）从图中某个初始顶点v出发，首先访问初始顶点v。

（2）选择一个与顶点v相邻且没被访问过的顶点w，再从w出发进行深度优先搜索，直到图中与当前顶点v邻接的所有顶点都被访问过为止。

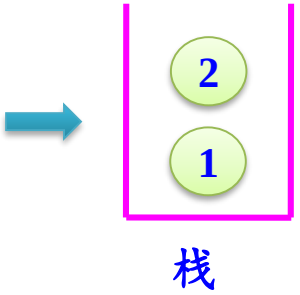
算法设计思路：

- 深度优先遍历的过程体现出后进先出的特点：用栈或递归方式实现。



DFS：1→2 →4 …

用栈保存访问过的顶点



- 如何确定一个顶点是否访问过？设置一个visited[] 全局数组，

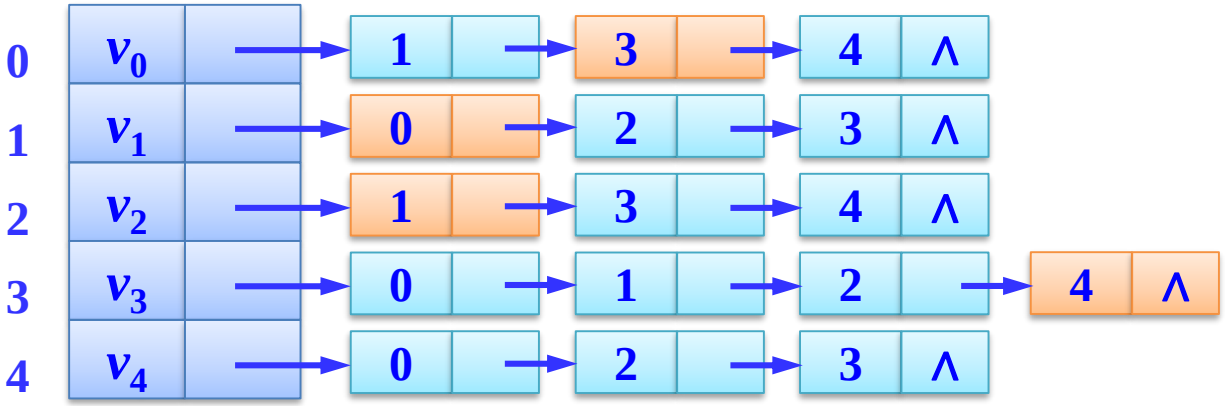
visited[*i*]=0表示顶点*i*没有访问； visited[*i*]=1表示顶点*i*已经访问过。

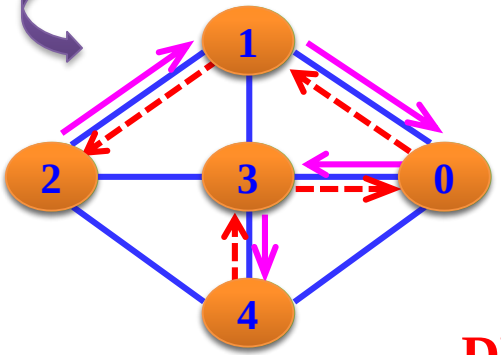
采用邻接表的DFS算法：

```
void DFS(ALGraph *G，int v)
{    ArcNode *p; int w;
     visited[v]=1;                  //置已访问标记
     printf("%d "，v);              //输出被访问顶点的编号
     p=G->adjlist[v].firstarc;//p指向顶点v的第一条边的边头节点
     while (p!=NULL)
     {    w=p->adjvex;
          if (visited[w]==0)
               DFS(G，w);  //若w顶点未访问，递归访问它
          p=p->nextarc;       //p指向顶点v的下一条边的边头节点
     }
}
```

该算法的时间复杂度为$O(n+e)$。

深度优先遍历过程演示





v=2的DFS序列：

2　　1　　0　　3　　4

遍历过程结束

DFS思路：距离初始顶点越远越优先访问！

思考题：

用栈求解迷宫问题，与DFS算法有什么关联？

8.3.3 广度优先遍历算法

广度优先遍历的过程：

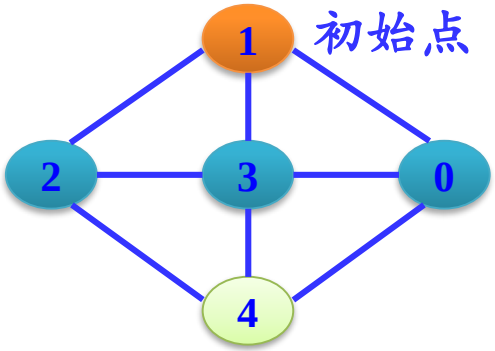
（1）访问初始点v，接着访问v的所有未被访问过的邻接点v_1，v_2，…，v_t。

（2）按照v_1，v_2，…，v_t的次序，访问每一个顶点的所有未被访问过的邻接点。

（3）依次类推，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。

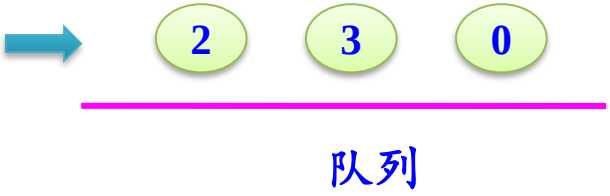
算法设计思路：

- 广度优先搜索遍历体现先进先出的特点，用队列实现。



BFS：1→2 →3 →0 …

用队列保存访问过的顶点



- 如何确定一个顶点是否访问过? 设置一个visited[] 数组，

visited[*i*]=0表示顶点*i*没有访问； visited[*i*]=1表示顶点*i*已经访问过。

采用邻接表的BFS算法：

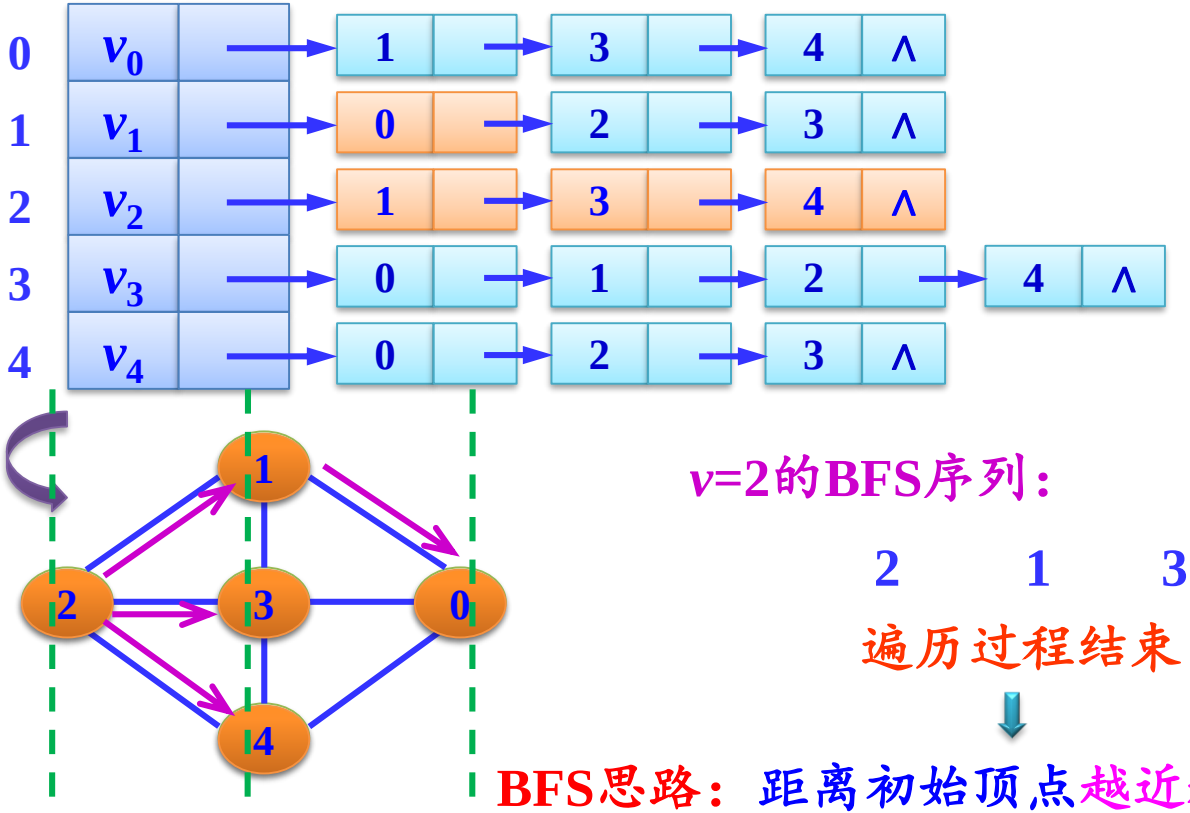
```
void BFS(ALGraph *G，int v)
{    ArcNode *p; int w，i;
     int queue[MAXV]，front=0，rear=0; //定义循环队列
     int visited[MAXV];
     for (i=0;i<G->n;i++)
          visited[i]=0;                    //访问标志数组初始化
     printf("%2d"，v);                     //输出被访问顶点的编号
     visited[v]=1;                         //置已访问标记
     rear=(rear+1)%MAXV;
     queue[rear]=v;                        //v进队
```

思考题： 为什么visited数组不需要设置为全局变量？

```
    while (front!=rear)                          //队列不空时循环
    {     front=(front+1)%MAXV;
          w=queue[front];                        //出队并赋给w
          p=G->adjlist[w].firstarc;              //找w的第一个的邻接点
          while (p!=NULL)
          {     if (visited[p->adjvex]==0)
                {   printf("%2d",  p->adjvex); //访问之
                    visited[p->adjvex]=1;
                    rear=(rear+1)%MAXV;    //相邻顶点进队
                    queue[rear]=p->adjvex;
                }
                p=p->nextarc;                    //找下一个邻接顶点
          }
    }
}
```

该算法的时间复杂度为$O(n+e)$。

广度优先遍历过程演示



v=2的BFS序列：

2　　1　　3　　4　　0

遍历过程结束

BFS思路：距离初始顶点越近越优先访问！

思考题：

❶ 用队列求解迷宫问题，与BFS算法有什么关联？

❷ 图采用邻接矩阵存储时，DFS和BFS算法如何实现，时间复杂度分别是多少？

8.3.4 非连通图的遍历

- 无向连通图：调用一次DFS或BFS，能够访问到图中的所有顶点。

- 无向非连通图：调用一次DFS或BFS，只能访问到初始点所在连通分量中的所有顶点，不可能访问到其他连通分量中的顶点。

  可以分别遍历每个连通分量，才能够访问到图中的所有顶点。

采用深度优先遍历方法遍历非连通图的算法如下：

```
void  DFS1(ALGraph *G)
{    int i;
     for (i=0;i<G->n;i++)     //遍历所有未访问过的顶点
          if (visited[i]==0)
               DFS(G，i);
}
```

非连通图：调用DFS()的次数恰好等于连通分量的个数

采用广度优先遍历方法遍历非连通图的算法如下：

```
void  BFS1(ALGraph *G)
{     int i;
      for (i=0;i<G->n;i++)     //遍历所有未访问过的顶点
          if (visited[i]==0)
              BFS(G，i);
}
```

非连通图：调用BFS()的次数恰好等于连通分量的个数

【例8-1】 假设图G采用邻接表存储，设计一个算法，判断无向图G是否连通。若连通则返回true；否则返回false。

求解思路

- 采用某种遍历方式来判断无向图G是否连通。这里用深度优先遍历方法，先给visited[]数组（为全局变量）置初值0，然后从0顶点开始遍历该图。
- 在一次遍历之后，若所有顶点*i*的visited[*i*]均为1，则该图是连通的；否则不连通。

判断无向图G是否连通的算法如下：

```
int  visited[MAXV];
bool Connect(ALGraph *G) //判断无向图G的连通性
{    int i;
     bool flag=true;
     for (i=0;i<G->n;i++)        //visited数组置初值
          visited[i]=0;
     DFS(G,0);                   //调用前面的中DSF算法,从顶点0开始深度优先遍历
     for (i=0;i<G->n;i++)
          if (visited[i]==0)
          {      flag=false;
                 break;
          }
     return flag;
}
```

图搜索算法设计一般方法

提示：两个遍历算法是图搜索算法的基础，必须熟练掌握！

——本讲完——