数据结构

（C语言版）（第2版）

图

图的遍历

主讲教师：汪红松

教学内容 Contents

图的遍历

遍历定义

从已给的连通图中某一顶点出发，沿着一些边访遍图中所有的顶点，且使每个顶点仅被访问一次，就叫做图的遍历，它是图的基本运算。

遍历实质

找每个顶点的邻接点的过程

图的特点

图中可能存在回路，且图的任一顶点都可能与其它顶点相通，在访问完某个顶点之后可能会沿着某些边又回到了曾经访问过的顶点。

怎样避免重复访问？

解决思路：设置辅助数组 *visited* [n]，用来标记每个被访问过的顶点。

- ✓ 初始状态为0
- ✓ *i* 被访问，改 *visited* [*i*]为1，防止被多次访问

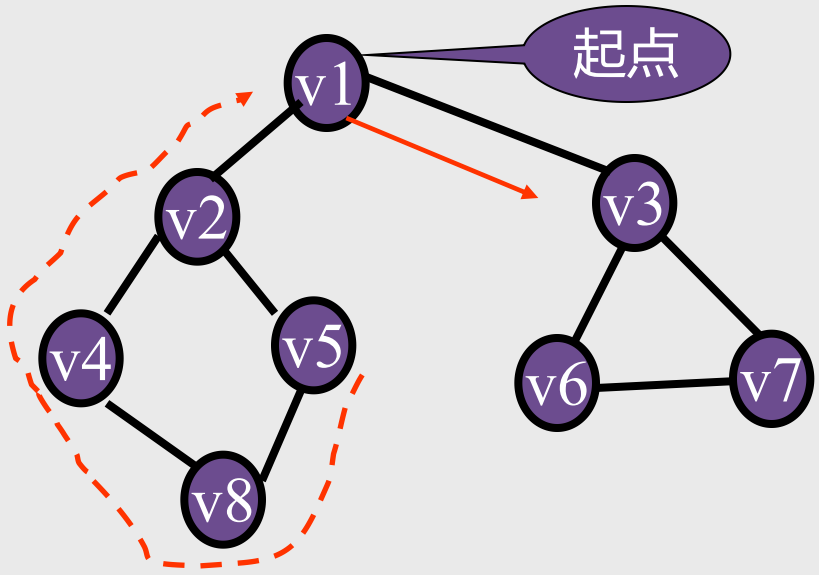
图常用的遍历：

✓深度优先搜索

✓广度优先搜索

深度优先搜索(DFS - Depth_First Search)

基本思想：——仿树的先序遍历过程。



DFS 结果

v1→v2→v4→v8→
v5→v3→v6→v7

深度优先搜索的步骤

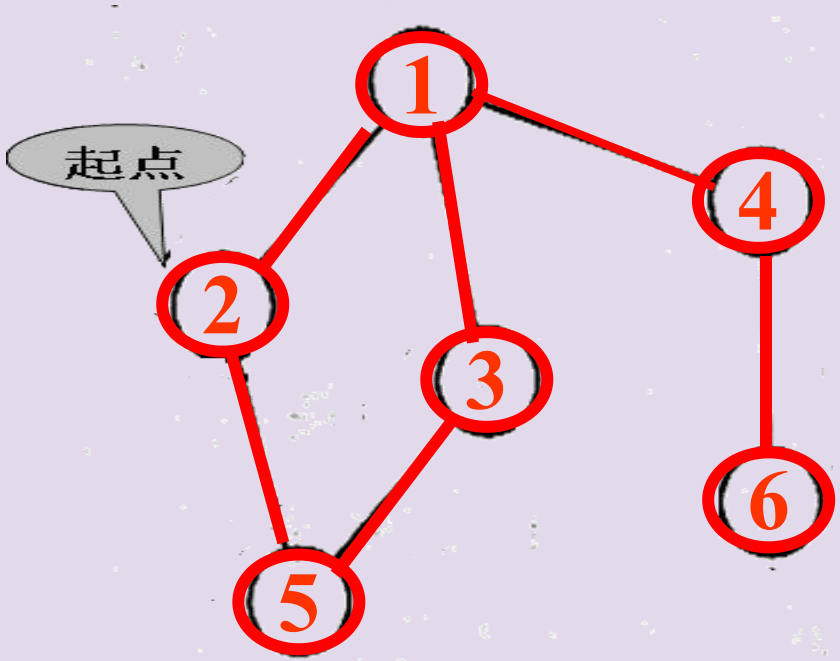
简单归纳：

访问起始点v；

若v的第1个邻接点没访问过，深度遍历此邻接点；

若当前邻接点已访问过，再找v的第2个邻接点重新遍历。

讨论1：计算机如何实现DFS？



DFS 结果

2→1→3→5→4→6

讨论1：计算机如何实现DFS？

辅助数组 *visited* [n]

邻接矩阵A

	1	2	3	4	5	6
1	0	1	1	1	0	0
2	1	0	0	0	1	0
3	1	0	0	0	1	0
4	1	0	0	0	0	1
5	0	1	1	0	0	0
6	0	0	0	1	0	0

1	0	0	1	1	1	1	1
2	0	1	1	1	1	1	1
3	0	0	0	1	1	1	1
4	0	0	0	0	0	1	1
5	0	0	0	0	1	1	1
6	0	0	0	0	0	0	1

2→1→ 3→ 5→ 4→ 6

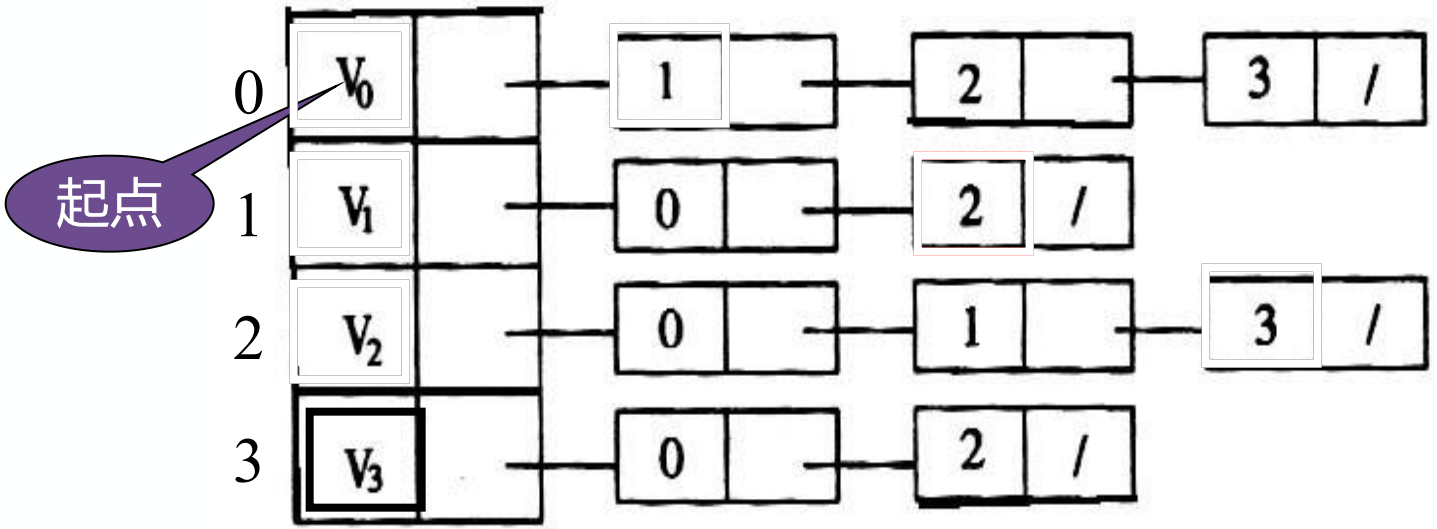
讨论2：DFS算法如何编程？

—用递归实现遍历算法

```
void DFS(AMGraph G, int v){          //图G为邻接矩阵类型
  cout<<v;  visited[v] = true;       //访问第v个顶点
  for(w = 0; w< G.vexnum; w++)       //依次检查邻接矩阵v所在的行
       if((G.arcs[v][w]!=0)&& (!visited[w]))
          DFS(G, w);
     //w是v的邻接点，如果w未访问，则递归调用DFS
}
```

讨论3：在图的邻接表中如何进行DFS？

—借用*visited* [n] ！



辅助数组 *visited* [n]

0	0	1	1	1	1
1	0	0	1	1	1
2	0	0	0	1	1
3	0	0	0	0	1

DFS 结果

v0 → v1 → v2 → v3

讨论4：邻接表的DFS算法如何编程？

——用递归实现遍历算法

```
void DFS(ALGraph G, int v){        //图G为邻接表类型
  cout<<v;  visited[v] = true;     //访问第v个顶点
  p= G.vertices[v].firstarc;       //p指向v的边链表的第一个边结点
  while(p!=NULL){                  //边结点非空
        w=p->adjvex;               //表示w是v的邻接点
        if(!visited[w])  DFS(G, w); //如果w未访问，则递归调用DFS
        p=p->nextarc;              //p指向下一个边结点
   }
}
```

DFS算法效率分析

用邻接矩阵来表示图，遍历图中每一个顶点都要从头扫描该顶点所在行，时间复杂度为$O(n^2)$。

用邻接表来表示图，虽然有 $2e$ 个表结点，但只需扫描 e 个结点即可完成遍历，加上访问 n个头结点的时间，时间复杂度为$O(n+e)$。

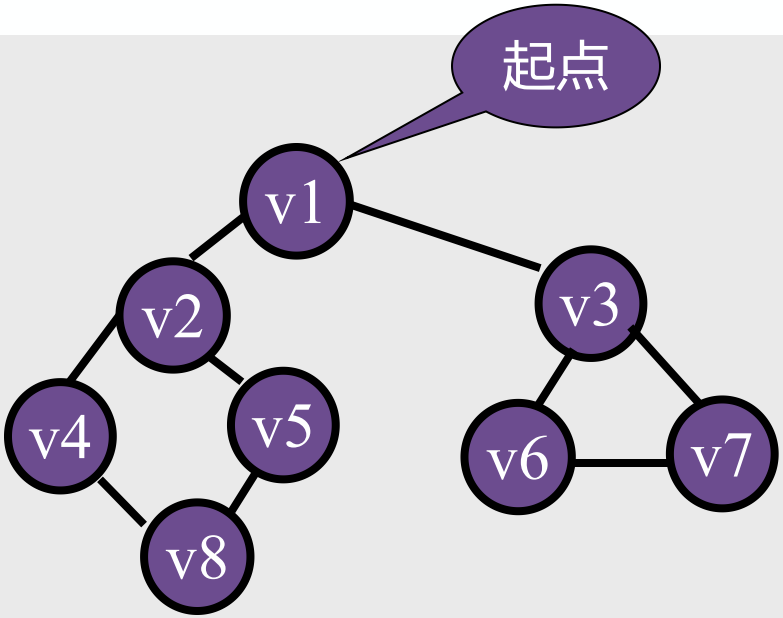
结论：

稠密图适于在邻接矩阵上进行深度遍历；

稀疏图适于在邻接表上进行深度遍历。

广度优先搜索(BFS - Breadth_First Search)

基本思想：——仿树的层次遍历过程



BFS 结果

v1→v2→v3→

v4→v5→v6→v7→v8

广度优先搜索的步骤

简单归纳：

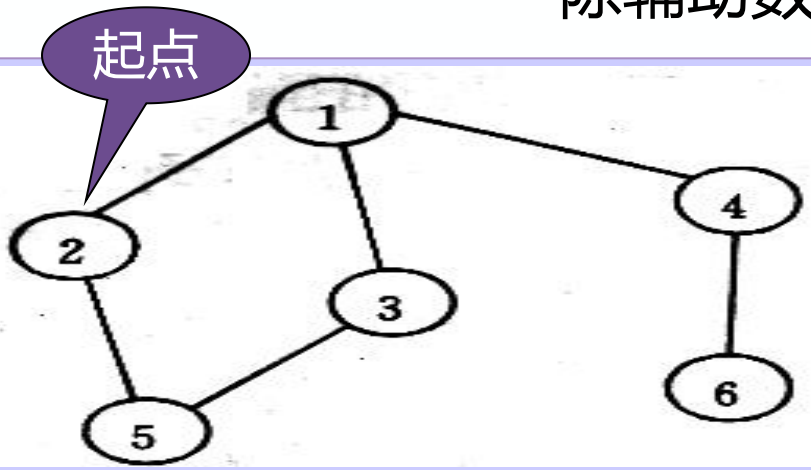
- 在访问了起始点v之后，依次访问 v的邻接点；
- 然后再依次访问这些顶点中未被访问过的邻接点；
- 直到所有顶点都被访问过为止。

✓ 广度优先搜索是一种分层的搜索过程，每向前走一步可能访问一批顶点，不像深度优先搜索那样有回退的情况。

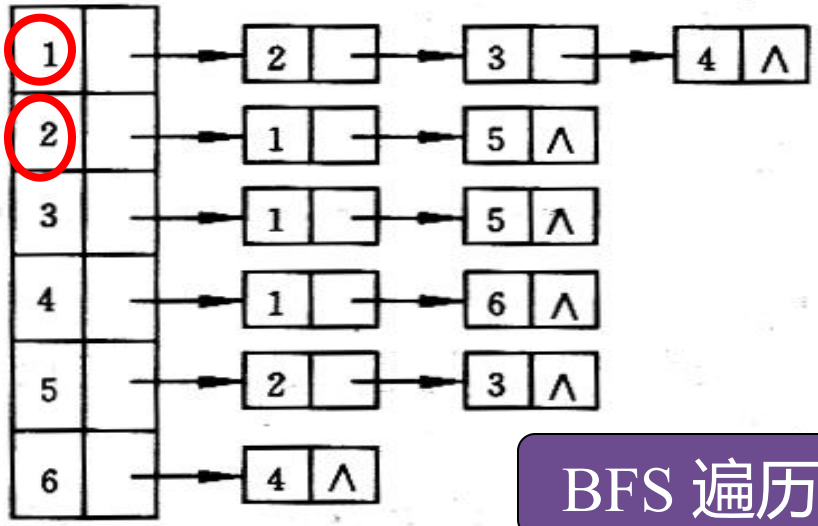
✓ 因此，广度优先搜索不是一个递归的过程，其算法也不是递归的。

讨论1：计算机如何实现BFS？

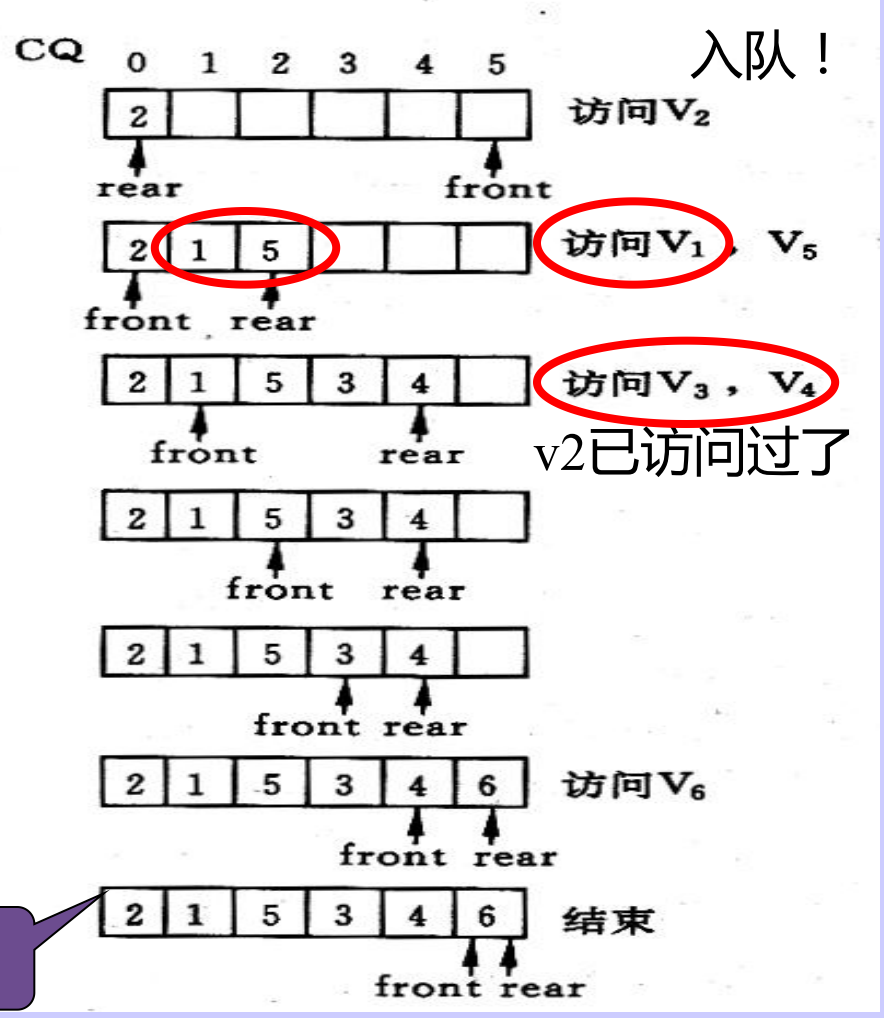
—除辅助数组*visited* [n]外，还需再开一辅助队列



邻接表



辅助队列



BFS 遍历结果

【算法描述】

```
void BFS (Graph G, int v){
   //按广度优先遍历连通图G
   cout<<v; visited[v] = true;                    //访问第v个顶点
   InitQueue(Q);                                  //辅助队列Q初始化，置空
   EnQueue(Q, v);                                 //v进队
   while(!QueueEmpty(Q)){                         //队列非空
     DeQueue(Q, u);                               //队头元素出队并置为u
     for(w = FirstAdjVex(G, u); w>=0; w = NextAdjVex(G, u, w))
     if(!visited[w]){                             //w为u的尚未访问的邻接顶点
          cout<<w; visited[w] = true;EnQueue(Q, w); //w进队
       }//if
   }//while
}//BFS
```

BFS算法效率分析

如果使用邻接矩阵，则BFS对于每一个被访问到的顶点，都要循环检测矩阵中的整整一行（ n 个元素），总的时间代价为$O(n^2)$。

用邻接表来表示图，虽然有 2e 个表结点，但只需扫描 e 个结点即可完成遍历，加上访问n个头结点的时间，时间复杂度为O(n+e)。

小结

1. 图的深度优先遍历方法和广度优先遍历方法
2. 针对图的邻接矩阵和邻接表两种存储结构分别实现了上述两种不同的遍历算法
3. 分析了不同存储结构图的遍历算法的时间复杂度