数据结构

深圳技术大学
大数据与互联网学院

第七章 图

7.1 图的定义和术语
7.2 图的存储结构
7.3 图的遍历
7.4 图的连通
7.5 有向无环图及其应用
7.6 最短路径

上节复习

- 图的概念，G＝(V, E)
  - V是顶点，E是边或弧
  - 无向图的边(x,y)，有向图的弧<x,y>，带权值的图称为网
  - 根据集合画出图
- 图的术语
  - 顶点的度、入度、出度
  - 邻接、关联
  - 路径、回路（环）、简单路径
- 图的连通
  - 顶点之间有路径则称为连通
  - 无向图的连通包括：连通图、连通分量
    - 连通图是指任意两点之间有路径
  - 有向图的连通：强连通分量、强连通图
  - 生成树，用少的路径包含所有的顶点
  - 无向图多用生成树
  - 有向图多用生成森林

7.2 图的存储结构

一. 邻接矩阵

- 邻接矩阵，Adjacency Matrix，记录图中各顶点之间关系的二维数组
- 对于不带权的图，以1表示两顶点存在边(或弧)(相邻接)，以0表示两顶点不邻接，即

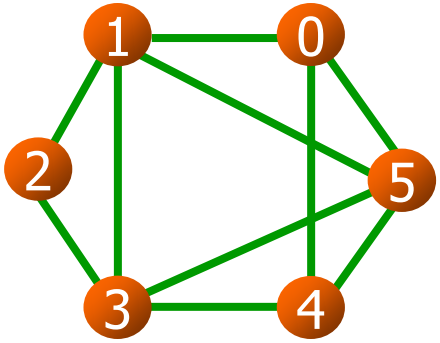
$$A[i][j] = \begin{cases} 1 & \text{如果}(i,j)\in E \text{ 或 } \langle i,j\rangle\in E \\ 0 & \text{其它} \end{cases}$$

7.2 图的存储结构

一. 邻接矩阵

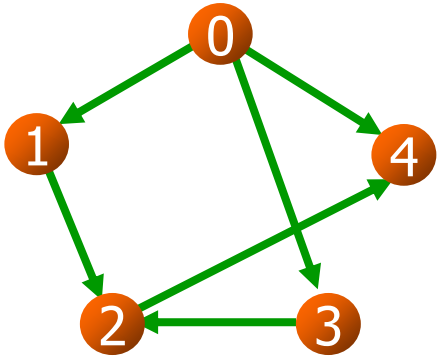
- 无向图的邻接矩阵

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 & 0 \end{pmatrix}$$



有向图的邻接矩阵

$$\begin{pmatrix} 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$



7.2 图的存储结构

一. 邻接矩阵

- 邻接矩阵的性质：
  - 无向图的邻接矩阵是对称的
  - 其第i行1的个数或第i列1的个数，等于顶点i的度TD(i)
  - 有向图的邻接矩阵可能是不对称的
  - 其第i行1的个数等于顶点i的出度OD(i)，第j列1的个数等于顶点j的入度ID(j)

7.2 图的存储结构

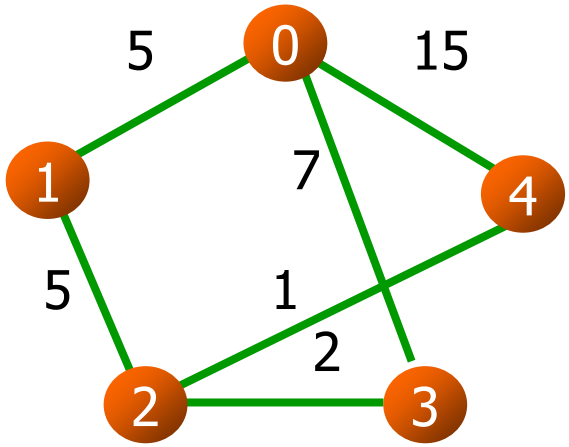
一. 邻接矩阵

- 网络：即权值的图。在网络中，两个顶点如果不邻接，则被视为距离为无穷大；如果邻接，则两个顶点之间存在一个距离值(即权值)

$$A[i][j] = \begin{cases} w_{i,j} & \text{如果}(i,j)\in E \text{ 或 } \langle i,j\rangle \in E \\ \infty & \text{其它} \end{cases}$$

练习

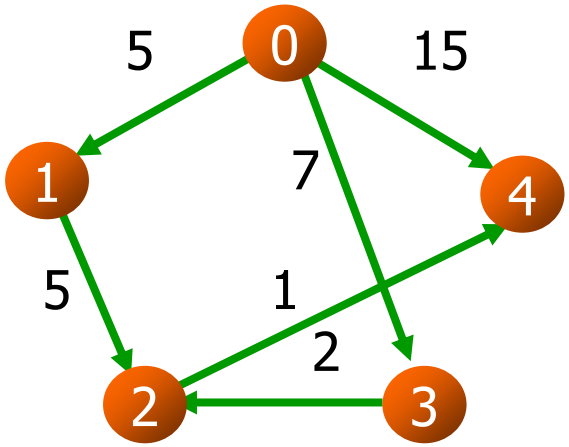
- 已知无向网N={V,E}，V={0,1,2,3,4}，E={(0,1,5)，(0,3,7)，(0,4,15)，(1,2,5)，(2,4,1)，(3,2,2)}，E中每个元组的第三个元素表示权。
  - 画出该网
  - 计算每个结点的度
  - 写出该网的邻接矩阵



$$\begin{pmatrix} \infty & 5 & \infty & 7 & 15 \\ 5 & \infty & 5 & \infty & \infty \\ \infty & 5 & \infty & 2 & 1 \\ 7 & \infty & 2 & \infty & \infty \\ 15 & \infty & 1 & \infty & \infty \end{pmatrix}$$

练习

- 有向网N={V,E}，V={0,1,2,3,4}，E={<0,1,5>，<0,3,7>，<0,4,15>，<1,2,5>，<2,4,1>，<3,2,2> }，E中每个元组的第三个元素表示权。
  - 画出该网
  - 计算每个结点的入度、出度、度
  - 写出该网的邻接矩阵



$$\begin{pmatrix} \infty & 5 & \infty & 7 & 15 \\ \infty & \infty & 5 & \infty & \infty \\ \infty & \infty & \infty & \infty & 1 \\ \infty & \infty & 2 & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \end{pmatrix}$$

7.2 图的存储结构

二. 邻接表

- 邻接表，Adjacency List，是图的一种链式存储结构
- 在邻接表中，每个顶点设置一个单链表，其每个结点都是依附于该顶点的边（或以该顶点为尾的弧）

7.2 图的存储结构

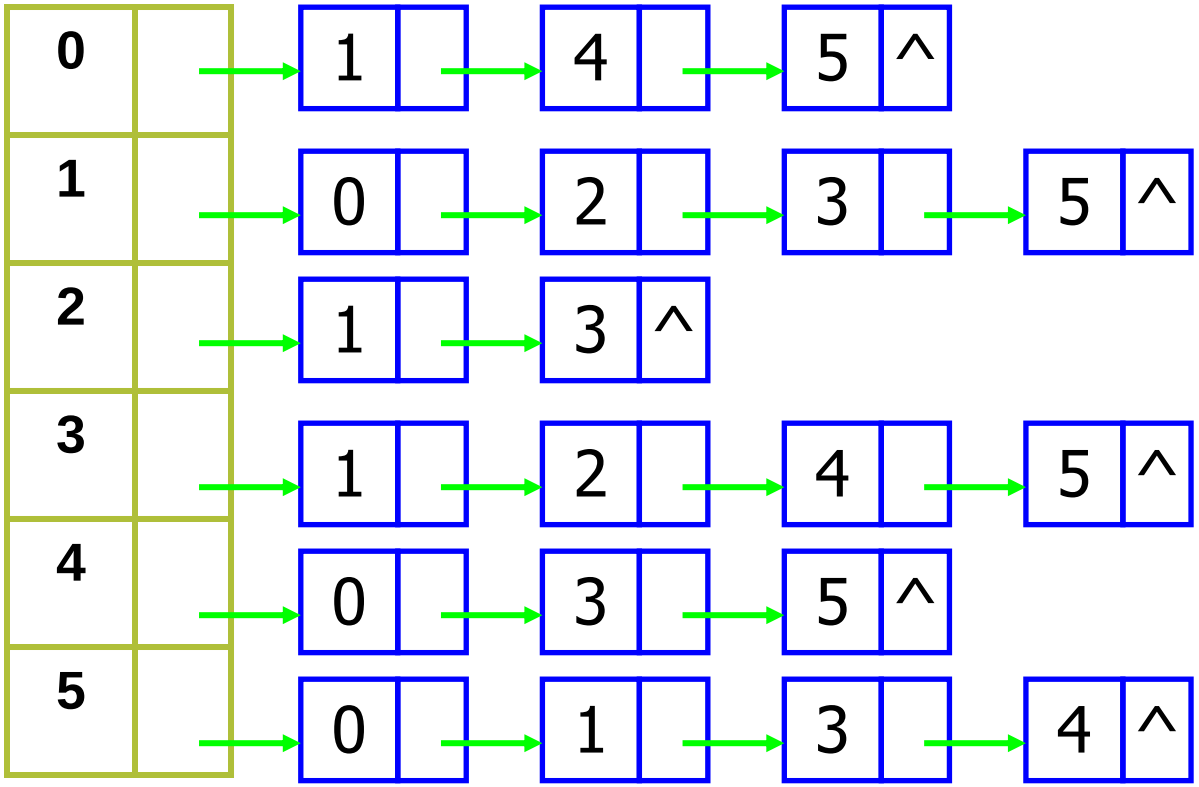
二. 邻接表

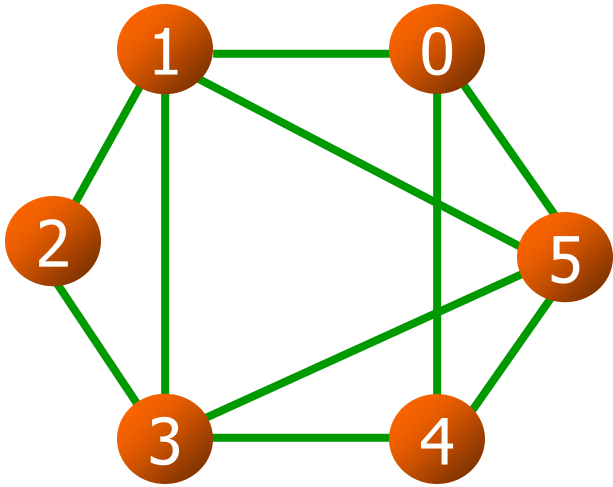
- 边(弧)的结点结构
  - adjvex; // 该边(弧)所指向的顶点的位置
  - nextarc;// 指向下一条边(弧)指针
  - info; // 该边(弧)相关信息的指针或权值
- 顶点的结点结构
  - data; // 顶点信息
  - firstarc;//指向第一条依附该顶点的边(弧)

7.2 图的存储结构

二. 邻接表

- 无向图的存储示意图

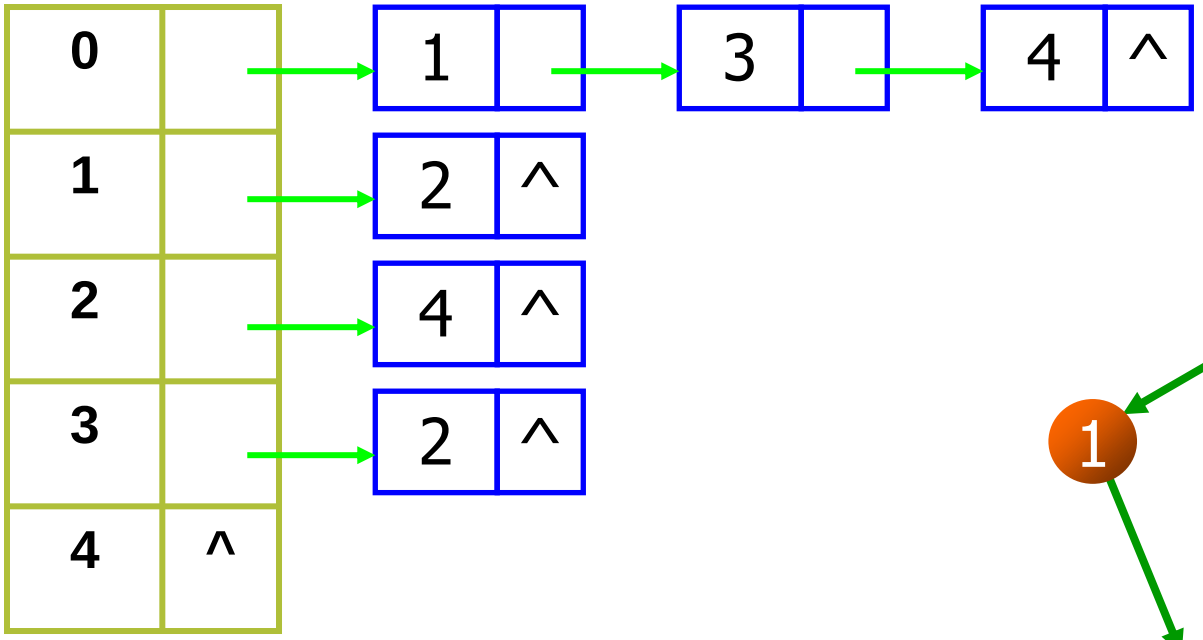


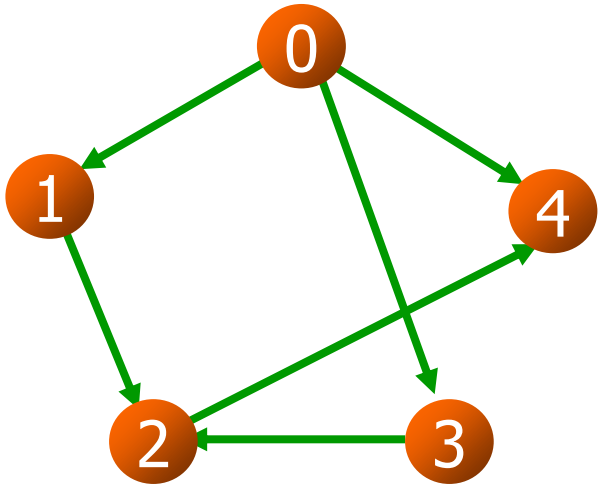


7.2 图的存储结构

二. 邻接表

- 有向图的存储示意图

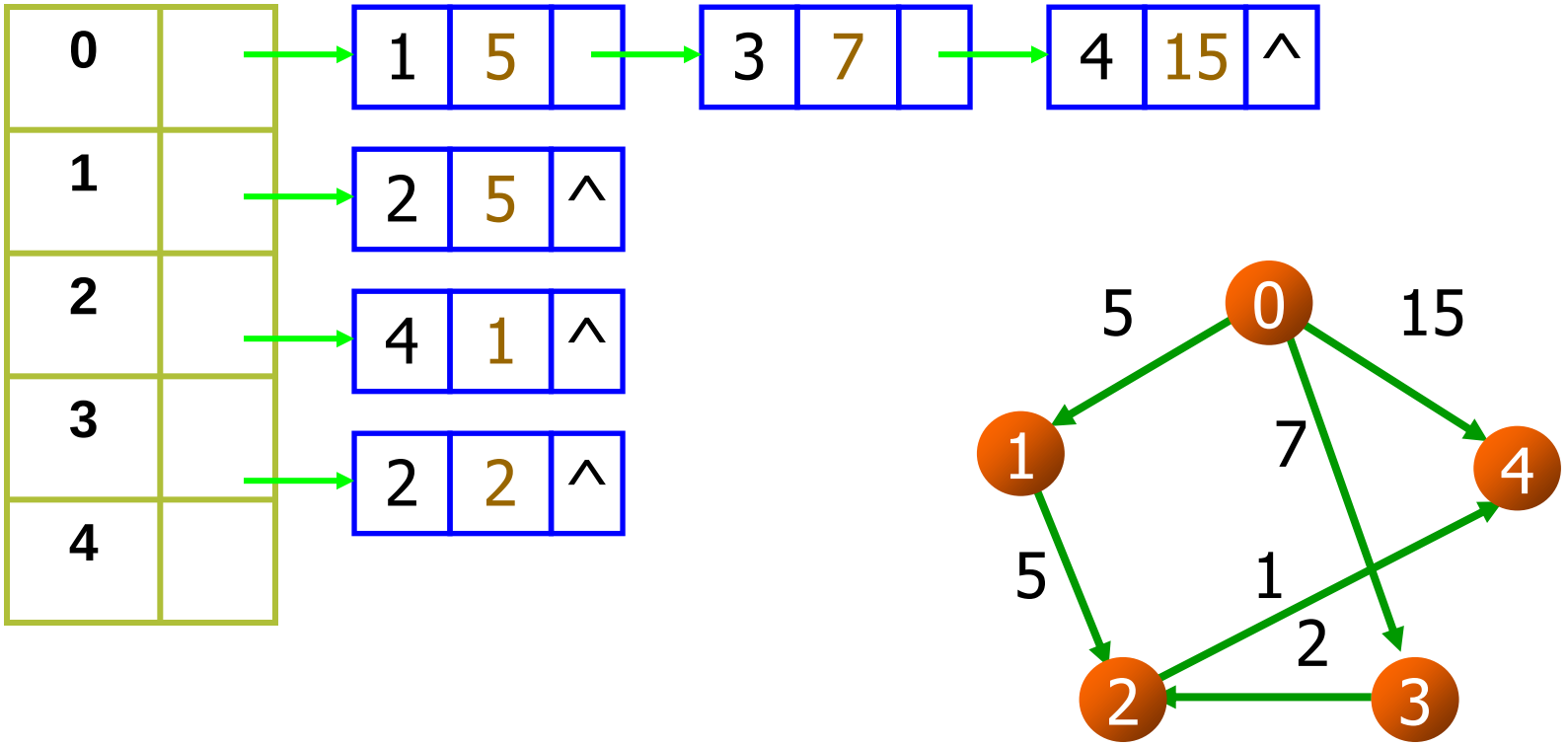




7.2 图的存储结构

二. 邻接表

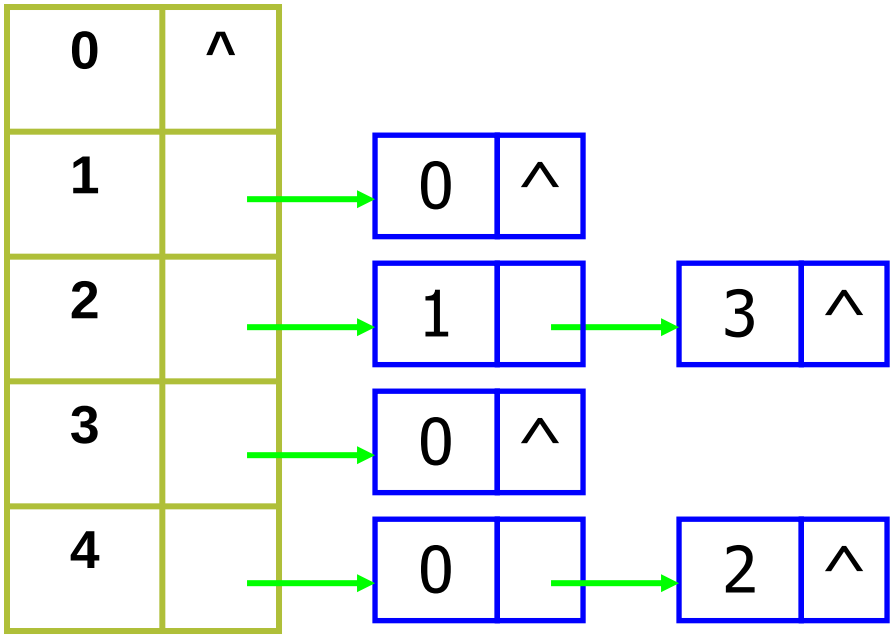
- 网的存储示意图

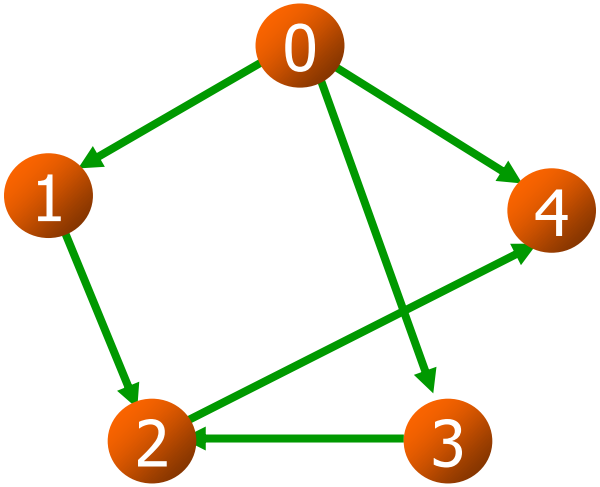


7.2 图的存储结构

二. 邻接表

- 有向图的逆邻接表，弧的箭头向内





7.2 图的存储结构

二. 邻接表

- 对于有向图的邻接表，其第i个链表中结点的个数只是该顶点的出度；逆邻接表则表示入度
- 要判定两个顶点i和j是否有边（或弧），必须搜索整个第i个和第j个链表，比不上邻接矩阵方便

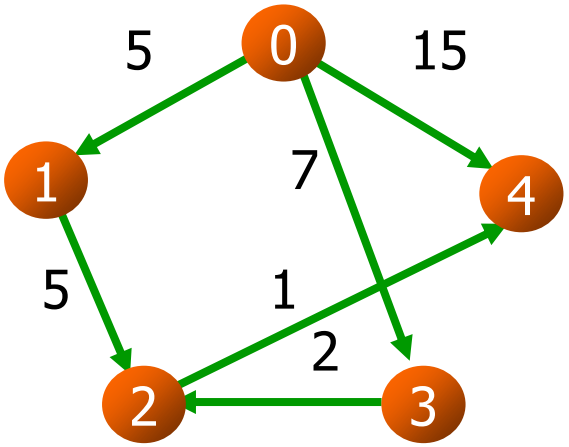
7.2 图的存储结构

二. 邻接表

- 邻接表优点：能直接得到与顶点i相邻接的其它顶点
- 邻接表缺点：要搜索才能知道两顶点间是否有连线
- 邻接矩阵优点：直接知道两顶点间是否有连线
- 邻接矩阵缺点：要搜索才能得到与顶点i相邻接的其它顶点

练习

- 有向网N={V,E}，V={0,1,2,3,4}，E={<0,1,5>，<0,3,7>，<0,4,15>，<1,2,5>，<2,4,1>，<3,2,2> }，E中每个元组的第三个元素表示权。
  - 画出该网
  - 计算每个结点的入度、出度
  - 写出该网的邻接表和逆邻接表



一. 遍历的含义

- 从图的某一顶点开始，访遍图中其余顶点，且使每一个顶点仅被访问一次
- 图的遍历主要应用于无向图

实验简介

二. 深度优先搜索

- 图的深度优先搜索，DFS-Depth First Search，是树的先根遍历的推广
- 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

实验简介

二. 深度优先搜索

- 算法流程：
  - 所有顶点访问标志visited[]设置为FALSE
  - 从某顶点v_0开始，设$v=v_0$

1. 如果visited[v]==FALSE，则访问该顶点v，并将顶点v压入栈中，且设visited[v]=TRUE

2. 如果找到当前顶点的一个新的相邻顶点w

（即visited[w] == FALSE），设v=w，重复1

3. 否则(说明当前顶点的所有相邻顶点都已被访问过，或者当前顶点没有相邻顶点)，如果当前顶点是v_0，退出;否则返回上一级顶点(即从栈中弹出一个顶点)，重复2

实验简介

二. 深度优先搜索

- 算法实现：

```
void DFSTraverse(Graph G, Status (*Visit)(int v)) {
    // 算法7.4
    // 对图G作深度优先遍历。
    int v;
    VisitFunc = Visit;
    // 使用全局变量VisitFunc，使DFS不必设函数指针参数
    for (v=0; v<G.vexnum; ++v) visited[v] = false;
    // 访问标志数组初始化
    for (v=0; v<G.vexnum; ++v)
    if (!visited[v]) DFS(G, v);
    // 对尚未访问的顶点调用DFS
}
```

实验简介

二. 深度优先搜索

- 算法实现：

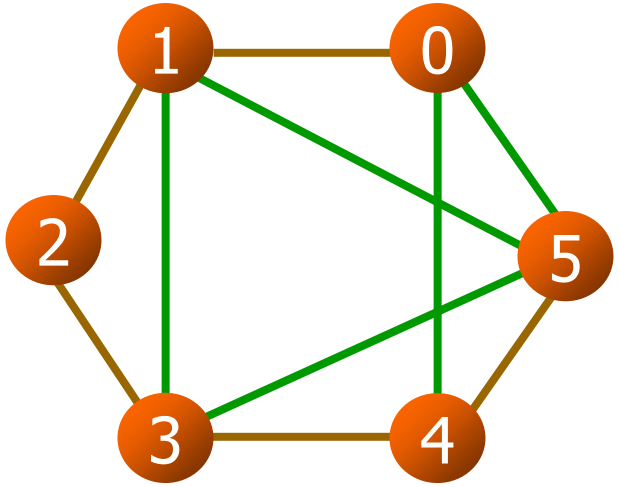
```
bool visited[MAX_VERTEX_NUM];    // 访问标志数组
Status (* VisitFunc)(int v);     // 函数变量
void DFS(Graph G, int v) {  // 算法7.5
   // 从第v个顶点出发递归地深度优先遍历图G。
   int w;
   visited[v] = true;
   VisitFunc(v);     // 访问第v个顶点
   for (w=FirstAdjVex(G, v);w!=-1;w=NextAdjVex(G, v, w))
      if (!visited[w])
    // 对v的尚未访问的邻接顶点w递归调用DFS
         DFS(G, w);
}
```

实验简介

二. 深度优先搜索

- 采用以下链表存储结构时，DFS次序为0,1,2,3,4,5

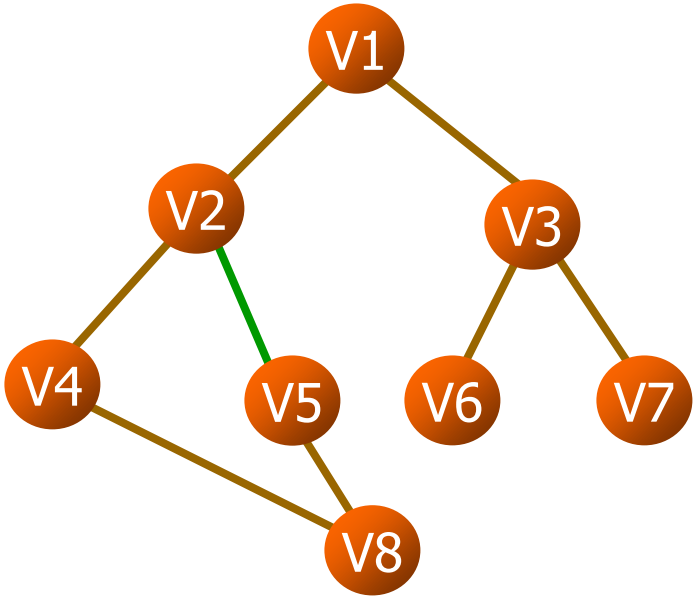
$$\begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 & 0 \end{pmatrix}$$



实验简介

二. 深度优先搜索

- DFS次序为V1,V2,V4,V8,V5,V3,V6,V7
  - V1、V2、V4、V8、V5依次访问并压入堆栈
  - 弹出V5、V8、V4、V2，查找V1相邻的新结点，找到V3
  - 访问V3、V6、V7
  - 弹出V7、V6、V3
  - 最后弹出V1，结束遍历



实验简介

三. 广度优先搜索

- 广度优先搜索，BFS-Broad First Search,是一种分层搜索方法
- BFS每向前走一步可能访问一批顶点，不存在往回退的情况
- BFS不是一个递归的过程，BFS使用队列

实验简介

三. 广度优先搜索

- 广度优先搜索算法
  - 所有顶点访问标志visited[]设置为FALSE
  - 从某顶点v_0开始，访问v_0，visited[v_0]=TRUE，将v_0插入队列Q

1. 如果队列Q不空，则从队列Q头上取出一个顶点v，否则结束
2. 依次找到顶点v的所有相邻顶点v’，如果visited[v′]==FALSE，则访问该顶点v’，然后将v′插入队列Q，并使visited[v′]=TRUE，
3. 重复1，2

7.3 图的遍历

三. 广度优先搜索

- BFS算法实现

```
void BFSTraverse(Graph G, Status (*Visit)(int v )) {//
算法7.6
  // 按广度优先非递归遍历图G。使用辅助队列Q和访问标志数组
visited。
  QElemType v,w;
  queue Q;
  QElemType u;
  for (v=0; v<G.vexnum; ++v) visited[v] = FALSE;
  InitQueue(Q);                          // 置空的辅助队列Q
  // ……接下半部分
```

7.3 图的遍历

三. 广度优先搜索

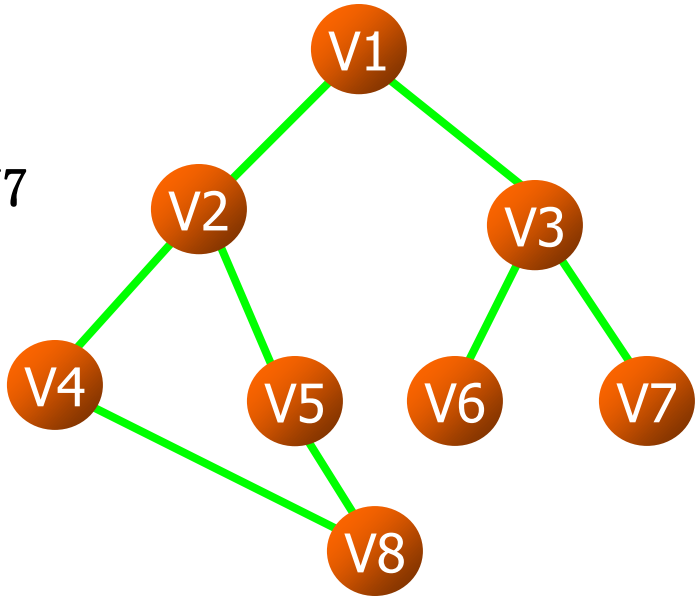
- BFS算法实现

```
void BFSTraverse(Graph G, Status (*Visit)(int v )) {//  算法
// ……接上半部分
  for (v=0;  v<G.vexnum;  ++v)
    if (!visited[v]) {                    // v尚未访问
      visited[v] = TRUE;  Visit(v); // 访问v
      EnQueue(Q, v);                      // v入队列
      while (!QueueEmpty(Q)) {
        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]) {
          // u的尚未访问的邻接顶点w入队列Q
            visited[w] = TRUE;  Visit(w);
            EnQueue(Q, w);
          }//if
      }//while
    }//if
} // BFSTraverse
```

7.3 图的遍历

三. 广度优先搜索

- 无向图举例
  - V1入队
  - V1出队，V2、V3入队，队列：V2 V3
  - V2出队，V4、V5入队，队列：V3 V4 V5
  - V3出队，V6、V7入队，队列：V4 V5 V6 V7
  - V4出队，V8入队，队列：V5 V6 V7 V8
  - V5 V6 V7 V8依次出队，结束



BFS为
v1, v2, v3, v4,
v5, v6, v7, v8