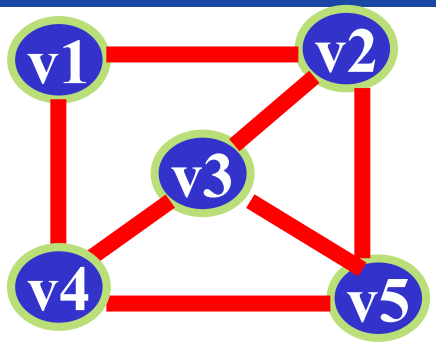
数据结构

主讲：项若曦　助教：申智铭、黄毅
rxxiang@blcu.edu.cn
主楼南329

回顾

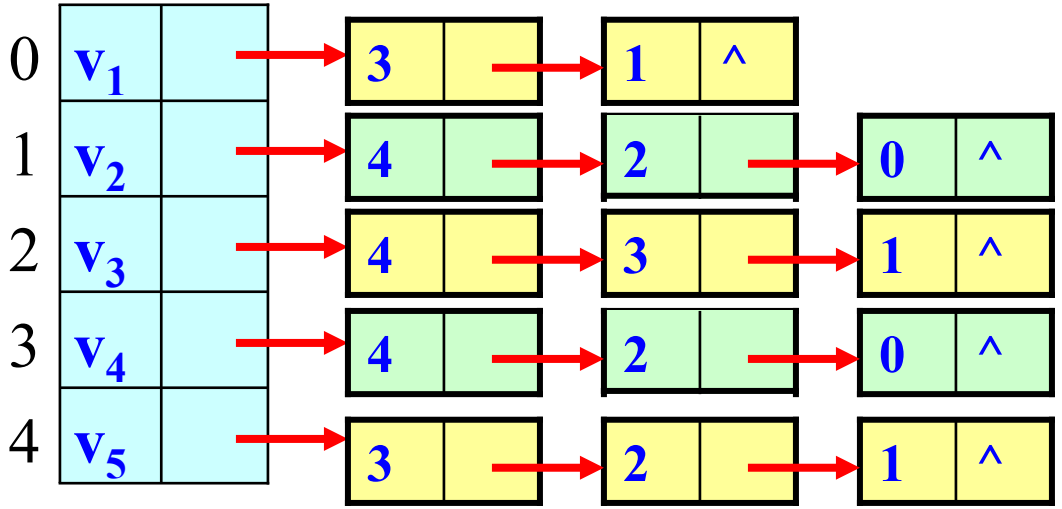
- **图的定义、概念、性质**
- **图的存储**
  - 邻接矩阵
  - 邻接表

邻接矩阵和邻接表的对比



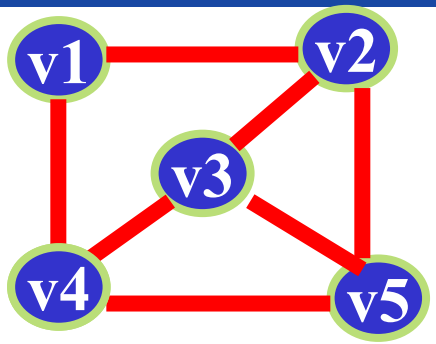
（ v1 v2 v3 v4 v5 ）

0	1	0	1	0	v1
1	0	1	0	1	v2
0	1	0	1	1	v3
1	0	1	0	1	v4
0	1	1	1	0	v5



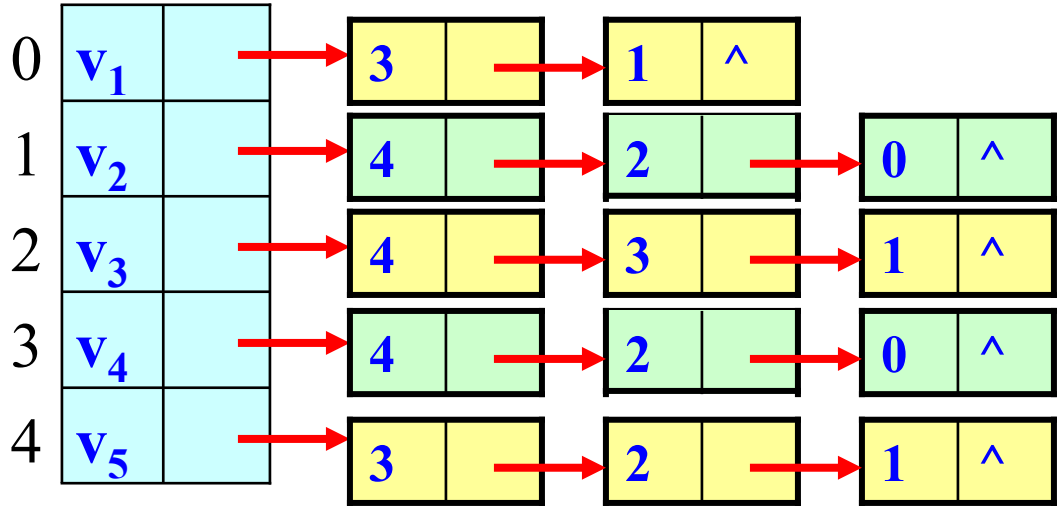
1. 联系：邻接表中每个链表对应于邻接矩阵中的一行，链表中结点个数等于一行中非零元素的个数。

邻接矩阵和邻接表的对比



（ v1 v2 v3 v4 v5 ）

0	1	0	1	0	v1
1	0	1	0	1	v2
0	1	0	1	1	v3
1	0	1	0	1	v4
0	1	1	1	0	v5



区别：

① 对于任一确定的无向图，邻接矩阵是唯一的（行列号与顶点编号一致），但邻接表不唯一（链接次序与顶点编号无关）。

② 邻接矩阵的空间复杂度为$O(n^2)$,而邻接表的空间复杂度为$O(n+e)$。

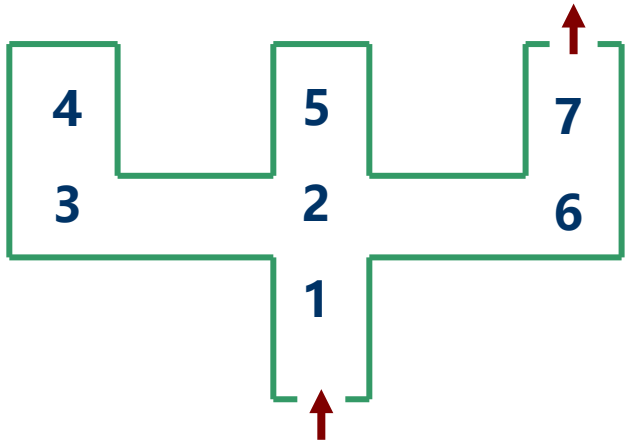
用途：邻接矩阵多用于稠密图；而邻接表多用于稀疏图

本节内容

- **图的遍历**
  - 图与迷宫

例：小型迷宫问题

路口	动作	结果
1 (入口)	正向走	进到 2
2	左拐弯	进到 3
3	右拐弯	进到 4
4 (堵死)	回溯	退到 3
3 (堵死)	回溯	退到 2
2	正向走	进到 5
5 (堵死)	回溯	退到 2
2	右拐弯	进到 6
6	拐弯	进到 7
	(出口)	



小型迷宫的交通路口数据

6 (路口数)

左行 0300077

直行 2500000

右行 0640000

```
1.  bool Traverse(int Pos, struct Maze& M) {                    //迷宫漫游算法
2.          if (Pos > 0) {                                      //路口从 1 开始
3.                  if (Pos == M.EXIT)                          //出口
4.                  {cout<<Pos <<  "<<  "; return 1;}
5.      回溯        else if (Traverse(M.intsec[Pos].left))      //向左
6.                  {cout<<Pos <<  "<<  "; return 1;}
7.      回溯        else if (Traverse(M.intsec[Pos].forwd))     //向前
8.                  {cout<<Pos <<  "<<  "; return 1;}
9.                  else if (Traverse(M.intsec[Pos].right))     //向右
10.                 {cout<<Pos <<  "<<  "; return 1;}
11.         }
12.         return 0;
13.}
```

3.2.4 栈的应用举例-迷宫求解

- **迷宫求解**
  - 问题：找从“入口”到“出口”的路径（所经过的通道方块）
  - 分析：

    1）方块的表示——坐标，当前的状态（障碍、未走的通路、已走的通路）；

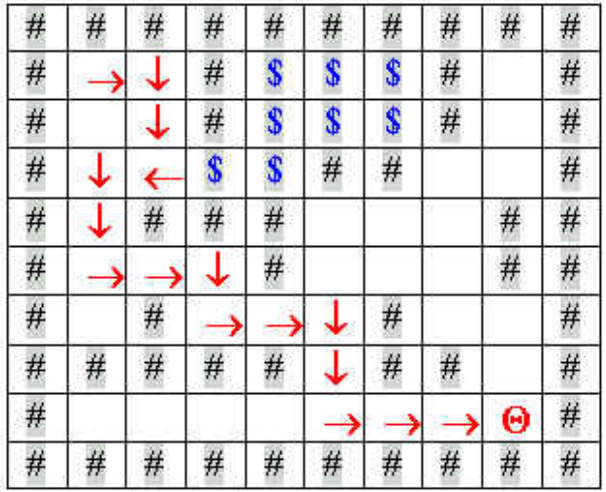
    2）已走的路径：

    A. 路径中各方块的位置及在路径中的序号;

    B. 从各方块出发已探索的方向，注意不能重复（可约定按东、南、西、北的方向顺次探索）；

    C. 从当前方块无路可走时，将已走路径回退一个方块，继续探索其他未走的方向

    栈——存储已走的通道块



3.2.4 求解示例（递归版）


```
void Go(start, end){//伪代码假设end没有设为墙壁
    if (start == end) exit(0);// 起点是终点，成功
    if (Go[start] == 0){      //start可能在某条通路上
        map[x][y] = 8;      //标记当前的路
        Go(start↑, end);
        Go(start←, end);
        Go(start↓, end);
        Go(start→, end);
        map[x][y] = -1;     //start不在通路上，标记
    }
}       //递归求解迷宫问题框架伪码
```

3.2.4 栈的应用举例-迷宫求解

- **迷宫求解**
  - 算法

```
1.  Status MazePath(MazeType &maze,PosType start, PosType end){
2.  //求解迷宫maze中,从入口start到出口end的一条路径
3.     InitStack(s);  PosType curpos = start;
4.     int curstep = 1;                                    //探索第一部
5.     do{
6.        if( Pass(maze,curpos) ){                         //如果当前位置可以通过,即是未曾走到的通道块
7.           FootPrint(maze,curpos);                       //留下足迹
8.           e = CreateSElem(curstep,curpos,1);            //创建元素
9.           Push(s,e);
10.          if( PosEquare(curpos,end) )    return TRUE;
11.          curpos =NextPos(curpos,1);                    //获得下一节点:当前位置的东邻
12.          curstep++;                                    //探索下一步
13.       }else{                                           //当前位置不能通过
14.          if(!StackEmpty(s)){
15.             Pop(s,e);
16.             while(e.di==4 && !StackEmpty(s) ){
17.                MarkPrint(maze,e.seat); Pop(s,e);      //留下不能通过的标记,并退回步
18.             }
19.             if(e.di<4){
20.                e.di++; Push(s,e);                     //换一个方向探索
21.                curpos = NextPos(e.seat,e.di);         //设定当前位置是该方向上的相块
22.             }//if
23.          }//if
24.       }//else
25.    }while(!StackEmpty(s));
26.    return FALSE;
27. }//MazePath
```

北京语言大学
BEIJING LANGUAGE AND CULTURE UNIVERSITY

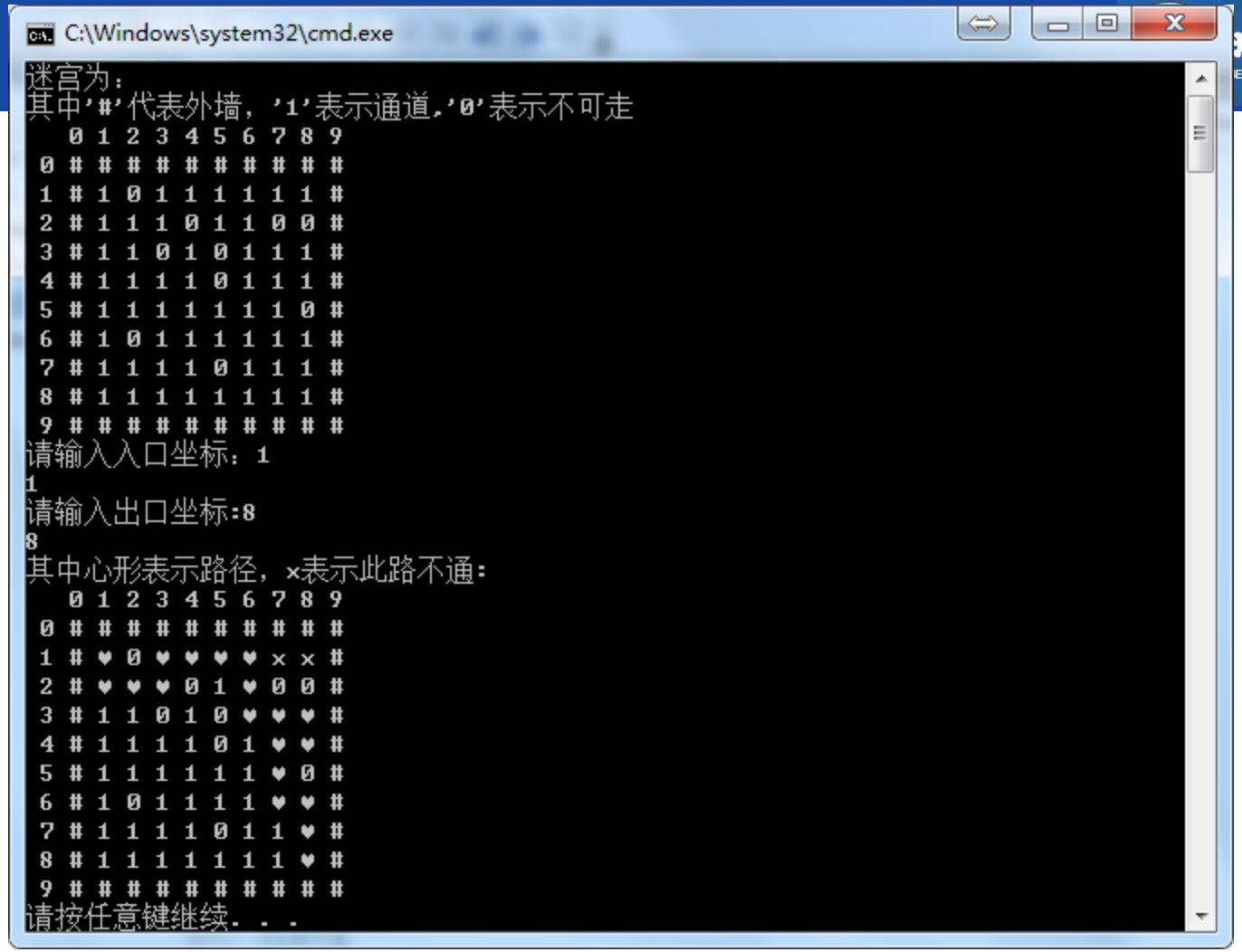
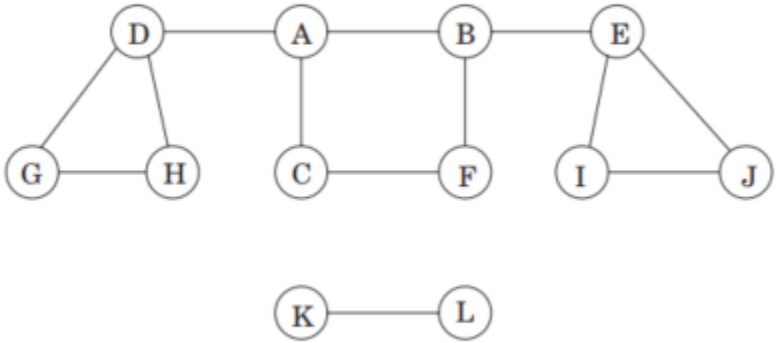
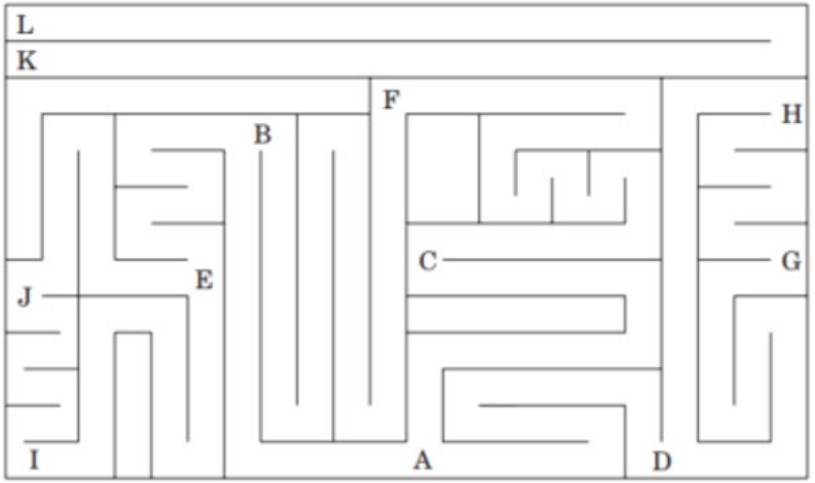


Figure 3.2 Exploring a graph is rather like navigating a maze.





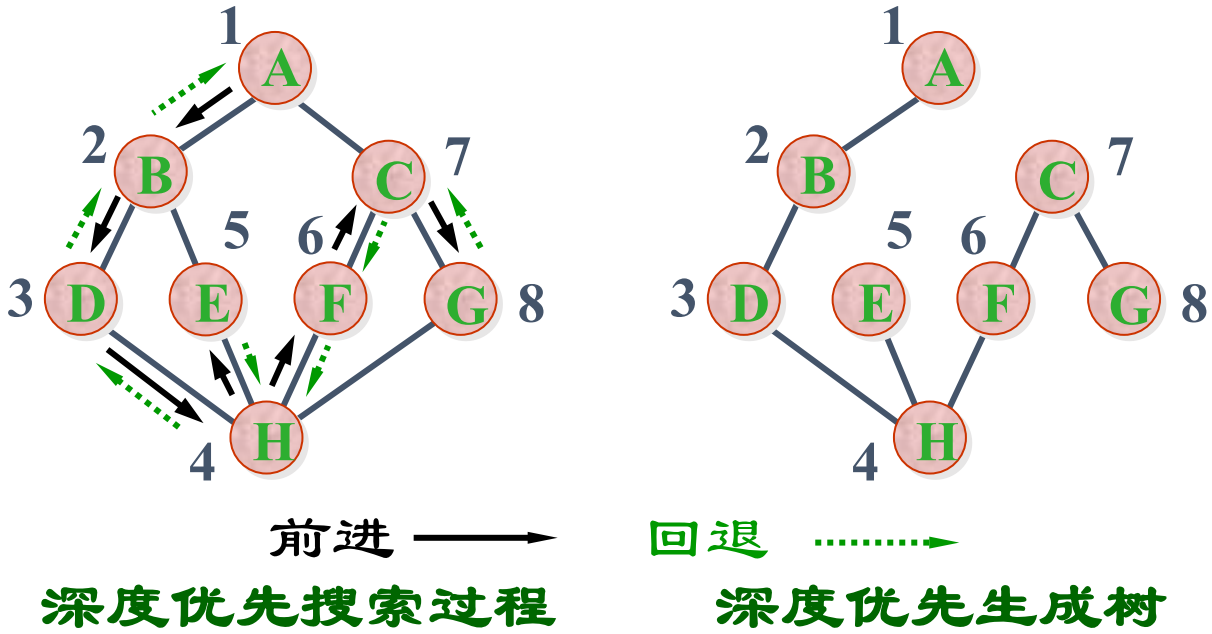
本节内容

- **图的遍历**
  - 深度优先遍历（DFS）
  - 广度优先遍历（BFS）

图的深度优先遍历 DFS(Depth First Search)

- **深度优先遍历：**
  - 从某顶点v 出发，访问该顶点；
  - 然后依次从v 的未被访问的邻接点出发深度优先遍历图；
    - 递归结束后，图中所有和v有路径相通的顶点都将被访问到。
  - 若图中还存在尚未访问过的顶点，则另选图中一个未曾被访问的顶点作起始点，继续重复上述过程
  - 分析
    - 类似于树的先序遍历
    - 引入访问标志数组visit[0:n-1]，区分顶点是否已被访问
    - 非连通图，需引入多个深度优先搜索的起始顶点
    - 递归算法或基于栈的非递归算法皆可实现

深度优先搜索的示例



图的深度优先遍历算法 DFS

```
1.  Bool visited[MAX_VERTEX_NUM];
2.  void  DFS(G,v ){
3.          visited[v] = true;          Visit(G, v);
4.          for ( w = FirstAdjVex( G, v);  w!=-1 ; w = NextAdjVex(G, v, w) )
5.                      if ( !visited[w] )        DFS(G, w);
6.  }
7.  void DFSTraverse(Graph G){
8.        for ( v = 0; v < G.vexnum; ++ v)        visited[v] = false;
9.        //memset(visited, 0, G.vexnum*sizeof(bool);
10.       for ( v = 0; v < G.vexnum; ++ v)
11.                     if ( !visited[v] )        DFS(G, v);
12.}
```

每调用一次 DFS() 就遍历了图的一个连通分量。

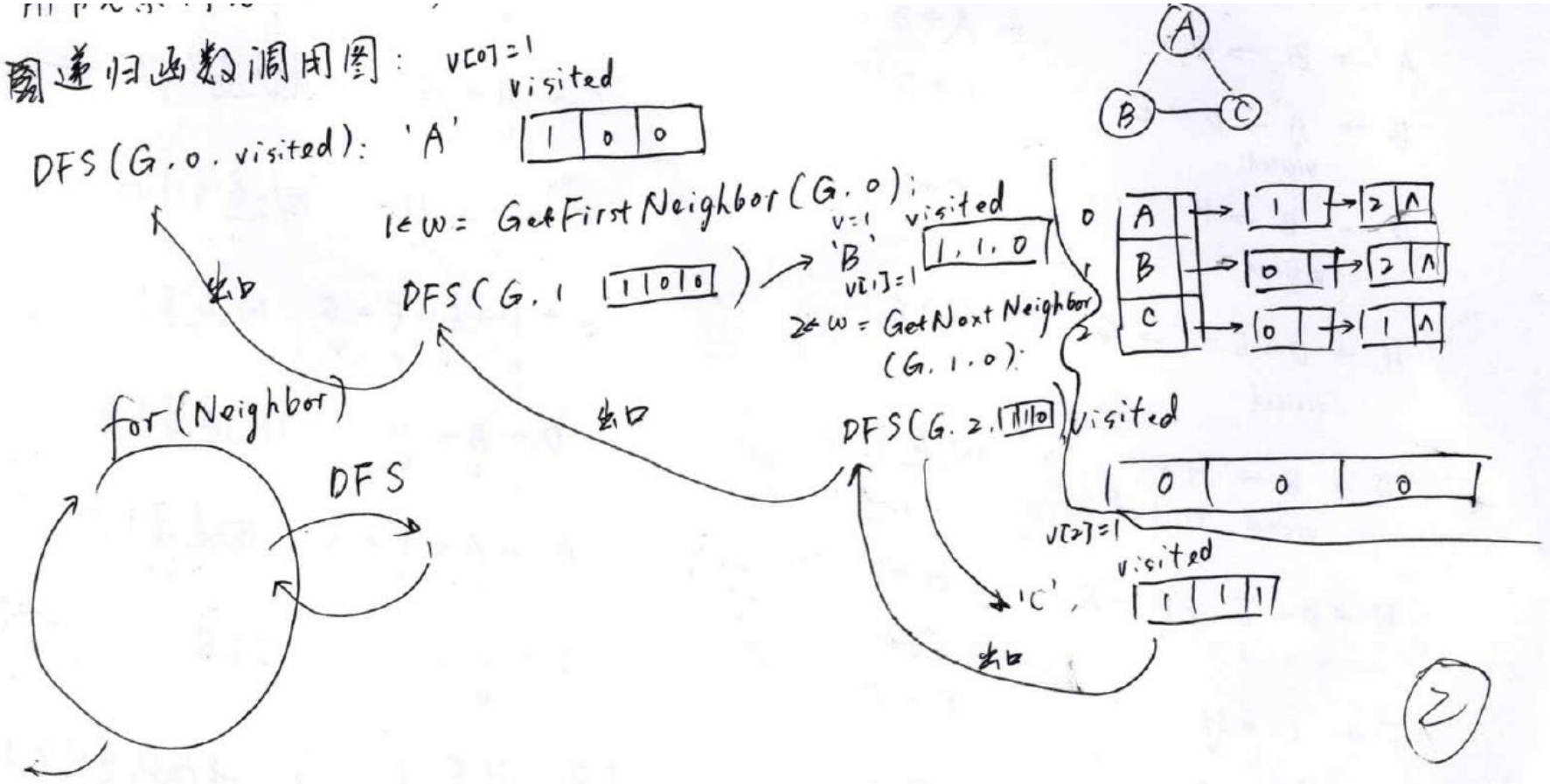
图的深度优先遍历算法

北京语言大学
BEIJING LANGUAGE AND CULTURE UNIVERSITY

```
DFS(Graph G,  v1){
      visited[v1] = TRUE;        Visit(G1, 'v1');
      for ( w =(v2,v3))
            if ( !visited[w] )              DFS(G1, w);
}
```

- DFS(Graph G, v2){
  - visited[v2] = TRUE; Visit(G1, 'v2');
  - for (w =(v1,v4,v5))
    - if (!visited[w]) DFS(G1, w);
  - }
    - DFS(Graph G, v4){
      - visited[v4] = TRUE; Visit(G1, 'v4');
      - for (w =(v2,v8))
        - if (!visited[w]) DFS(G1, w);
      - }
        - DFS(Graph G, v8){
          - visited[v8] = TRUE; Visit(G1, 'v8');
          - for (w =(v4,v5))
            - if (!visited[w]) DFS(G1, w);
          - }
            - DFS(Graph G, v5){
              - visited[v5] = TRUE; Visit(G1, 'v5');
              - for (w =(v2,v8))
                - if (!visited[w]) DFS(G1, w);
              - }
  - DFS(Graph G, v3){

- 请调试观察递归的变换！
- 另外，如果不用递归，应该怎么写代码？

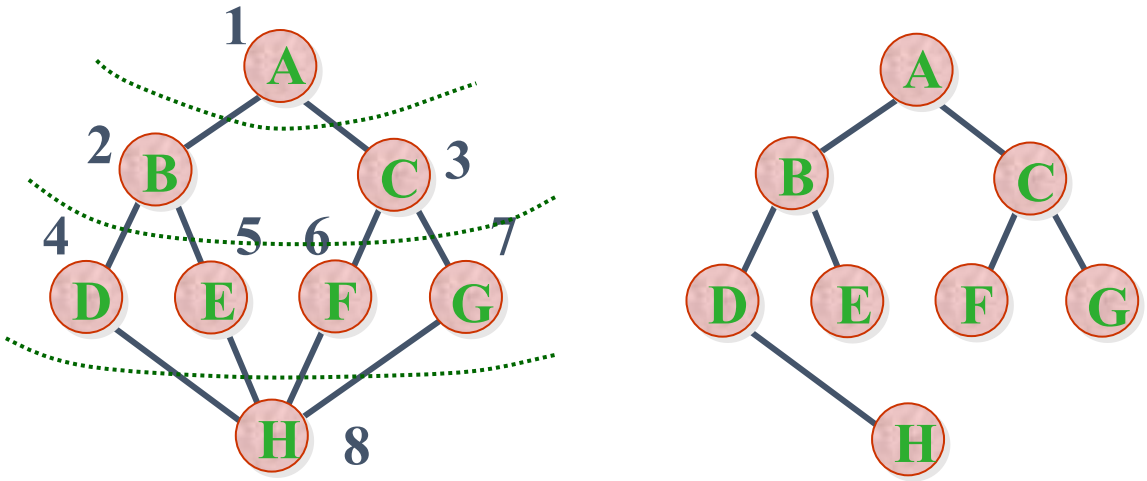

递归函数调用图：
V[0]=1
visited
DFS(G,0,visited): 'A'
1 0 0
A
B
C
if w = GetFirstNeighbor(G,0);
v=1
'B'
v[1]=1
visited
1 1 0
DFS(G,1
1 1 0 1 0
)
出栈
for(Neighbor)
DFS
if w = GetNext Neighbor
(G,1,0);
DFS(G,2,1 1 1 0
出栈
visited
0 0 0
v[2]=1
'C'
visited
1 1 1
出栈
0
A
B
C
1 1 2 ∧
0 0 2 ∧
0 1 ∧
2

图的广度优先遍历

北京语言大学
BEIJING LANGUAGE AND CULTURE UNIVERSITY

- **广度优先遍历：**
  - 从某顶点v 出发，访问该顶点;
  - 然后依次访问v 的所有未曾访问过的邻接点;
  - 然后分别从这些邻接点出发依次访问它们的邻接点，并使“先被访问的顶点的邻接点”先于“后被访问的顶点的邻接点” 被访问，直至图中所有已被访问的顶点的邻接点都被访问到;
  - 若图中还存在尚未访问过的顶点，则另选图中一个未曾被访问的顶点作起始点，继续重复上述过程
  - 分析
    - 类似于树的层次遍历
    - 引入visited访问标志数组
    - 非连通图，需引入多个广度优先搜索的起始顶点
    - 引入队列保存“顶点已访问，但其邻接点未全访问”的顶点编号

广度优先搜索的示例



广度优先搜索过程 ·····► 广度优先生成树

图的遍历—广度优先遍历算法（BFS）

北京语言大学
BEIJING LANGUAGE AND CULTURE UNIVERSITY

```
1.  void BFSTraverse(Graph G){
2.    for ( v = 0; v < G.vexnum; ++ v)     visited[v] = FALSE;
3.    InitQueue(Q);
4.    for ( v = 0; v < G.vexnum; ++ v)
5.        if ( !visited[v] ){
6.            visited[v] = TRUE;  Visit(G, v);
7.             EnQueue(Q, v);
8.             while(!QueueEmpty(Q)){
9.                  DeQueue(Q,u);
10.                 for ( w = FirstAdjVex( G, u) ; w ; w = NextAdjVex(G, u, w))
11.                     if ( !visited[w] ){
12.                             visited[w] = TRUE; Visit(G, w);
13.                             EnQueue(Q, w);
14.                     } //~if
15.             } //~while
16.         } //~if
17. }
```

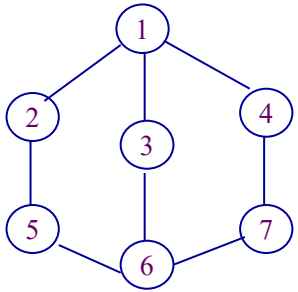
本节内容

- **图**
  - 图的遍历算法的应用
  - AOV/AOE网、拓扑排序

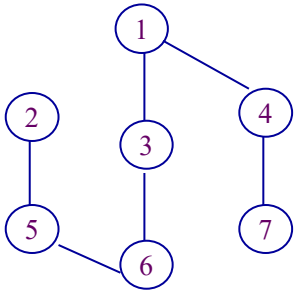
图的遍历算法的应用

1. 求一条包含图中所有顶点的简单路径（DFS的应用）

- 解决方案：
  - 遍历起点的选择：通过for循环，依次选择所有顶点作为起始顶点来尝试；若查找失败，算法应允许回溯从下一个起始顶点继续搜索。
  - 遍历中邻接点的选择：由于邻接点的选取是与顶点和邻接点的存储次序以及算法的搜索次序有关，不可能依据特定的图给出特定的解决算法。因此，在整个搜索中同样应允许有查找失败，此时可采取回溯到上一层的方法，继续查找其他路径。
  - 查找成功的判断：引入计数器count用来记录当前已经遍历过的路径上的顶点数。若count=n，则表示成功查找到其中一条路径。



G



(a)

图的遍历算法的应用

- **1. 求一条包含图中所有顶点的简单路径（DFS的应用）**
  - 算法进一步求精（在DFS算法基础上修改）：
    - 引入数组Path用来保存当前已搜索的简单路径上的顶点，引入计数器n用来记录当前该路径上的顶点数。
    - 对DFS算法的修改如下：
      - 计数器n的初始化，放在visited的初始化前后；
      - 访问顶点时，增加将该顶点序号入数组Path中，计数器n++；判断是否已获得所求路径，是则输出结束，否则继续遍历邻接点；
      - 某顶点的全部邻接点都访问后，仍未得到简单路径，则回溯，将该顶点置为未访问，计数器n--。

图的遍历算法的应用

北京语言大学 BEIJING LANGUAGE AND CULTURE UNIVERSITY

- 求一条包含图中所有顶点的简单路径（DFS的应用）

```
1. int Path[n],n;//算法: 邻接矩阵表示法, 红色字部分为在深度优先遍历上的增加或修改的步骤
2. void  Hamilton(MGraph G){
3.         for(i=0; i<G.vexnum; i++)
4.                 visited[i]=FALSE;
5.          n=0;
6.         for(i=0; i<G.vexnum; i++)
7.                 if ( !visited[i])     DFS (G, i);
8. }
9. void  DFS(MGraph G, int i){
10.        visited[i]=TRUE;
11.        Path[n]=i;
12.        n++;
13.        if(n==G.vexnum)      Print(Path);              //符合条件，输出该简单路径
14.       for(j=0; j<G.vexnum; j++)
15.                if( G.arcs[i][j].adj && !visited[j])
16.                DFS( G, j);
17.        visited[i]=FALSE;                              //查找失败时，回溯处理
18.        n--;
19.}
```

图的遍历算法的应用

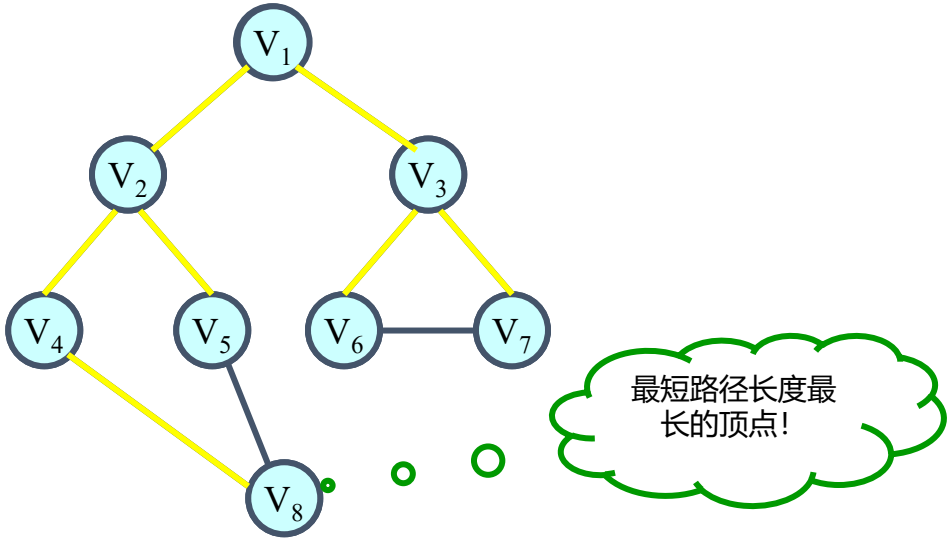
1. 求一条包含图中所有顶点的简单路径（DFS的应用）

- 思考
  - 若图中存在多条符合条件的路径，本算法是输出一条，还是输出全部？（全部）
  - 如何修改算法，变成判断是否有包含全部顶点的简单路径？
  - （print函数输出true而非路径）
  - 如何修改算法，输出包含全部顶点的简单路径的条数？
  - （加计数器）
  - 如何修改算法，输出所有的简单回路？

图的遍历算法的应用

- **2. 求距v_0最短路径长度最长的一个顶点 （BFS的应用）**
  - 思路：
    - 由于题意强调为最短路径，因此应当考虑BFS的算法应用。本问题的求解转变成：从v0出发进行BFS，最后一层的顶点距离v0的最短路径长度最长。

图的遍历算法的应用



广度优先搜索(BFS)

$V_1\ V_2\ V_3\ V_4\ V_5\ V_6\ V_7\ V_8$

2. 求距v_0最短路径长度最长的一个顶点 （BFS的应用）

- 对BFS算法进行适当修改：
  - 由于BFS类似于树的按层次遍历，原算法中引入队列用来保存本身已访问但其邻接点尚未全部访问的顶点。BFS遍历中最后一层的顶点一定是最后出队的若干顶点，队列中最后一个出队的顶点必定是符合题意的顶点。

这样，只需调用BFS的算法，将最后出队的元素返回即可。

图的遍历算法的应用

北京语言大学 BEIJING LANGUAGE AND CULTURE UNIVERSITY

- **2. 求距v_0最短路径长度最长的一个顶点 （BFS的应用）**

```
1. int  MaxDistance(MGraph G, int v0){
2.          for(i=0; i<G.vexnum; i++)
3.                  visited[i]=FALSE;
4.          InitQueue(Q);
5.          EnQueue(Q, v0);
6.          visited[v0] = TRUE;
7.          while( !QueueEmpty(Q)){
8.                  DeQueue(Q, v);
9.                  for(w = 0; w < G.vexnum; w++)
10.                         if(G.arcs[v][w].adj && !visited[w] ){
11.                                 visited[w]=TRUE;
12.                                 EnQueue(Q, w);
13.                         } // ~for_if
14.         } //~while
15.         return(v);
16.}
```

连通分量 (Connected component)

连通分量

- 当无向图为非连通图时，从图中某一顶点出发，利用深度优先搜索算法或广度优先搜索算法不可能遍历到图中的所有顶点，只能访问到该顶点所在的最大连通子图（连通分量）的所有顶点。
- 若从无向图的每一个连通分量中的一个顶点出发进行遍历，可求得无向图的所有连通分量。
- 求连通分量的算法需要对图的每一个顶点进行检测：若已被访问过，则该顶点一定是落在图中已求得的连通分量上；若还未被访问，则从该顶点出发遍历图，可求得图的另一个连通分量。

图的深度优先遍历算法

北京语言大学
BEIJING LANGUAGE AND CULTURE UNIVERSITY

```
1.  Boolean     visited[MAX_VERTEX_NUM];
2.  void  DFS(G,v ){
3.        visited[v] = TRUE;        Visit(G, v);
4.        for ( w = FirstAdjVex( G, v);  w!=NIL ; w = NextAdjVex(G, v, w) )
5.                  if ( !visited[w] )        DFS(G, w);
6.  }
7.  void DFSTraverse(Graph G){
8.      for ( v = 0; v < G.vexnum; ++ v)      visited[v] = FALSE;
9.      for ( v = 0; v < G.vexnum; ++ v)
10.                 if ( !visited[v] )        DFS(G, v);
11. }
```

每调用一次 DFS() 就遍历了图的一个连通分量。

最小生成树（minimum cost spanning tree）

最小生成树

- 使用不同的遍历方法，可得到不同的生成树；从不同的顶点出发，也可能得到不同的生成树。
- 按照生成树的定义，n 个顶点的连通带权图的生成树有 n 个顶点、n-1 条边。
- 构造最小生成树的准则
  - 必须使用且仅使用该带权图中的n-1 条边来联结网络中的 n 个顶点；
  - 不能使用产生回路的边；
  - 各边上的权值的总和达到最小。
- 问题解法
  - Prim算法（贪心算法，从某个顶点出发）
  - Kruskal算法（贪心算法，选边，直到形成一个连通分量）

预告

算法导论

- **最小生成树（Minimum Spanning Tree， MST）**
  - 适用条件：无向图
  - 最小生成树的性质
  - 最小生成树的两个贪心算法
    - Prim算法
    - Kruskal算法

最短路径 (Shortest Path)

- **最短路径问题：**
  - 如果从图中某一顶点(称为源点)到达另一顶点(称为终点)的路径可能不止一条，如何找到一条路径使得沿此路径上各边上的权值总和达到最小。
- **问题解法**
  - 单源最短路径：
    - 边上权值非负情形的单源最短路径问题
      — Dijkstra算法（迪杰斯特拉算法，贪心）
    - 权值可取负值
      — Bellman-Ford算法
  - 所有顶点之间的最短路径
    — Floyd算法

预告

算法导论

- **最短路径（Shortest Path，SP）**
  - 广度优先遍历BFS与单源最短路径
  - BFS推广到带权图——Dijkstra算法(Greedy)
  - 含有负边的图——Bellman-Ford算法(DP)
  - 有向无环图DAG上的最短路径——DAG-DP
  - 所有顶点对间的最短路径——Floyd算法（DP）

本节内容

- 有向无环图(DAG)
- 拓扑排序：AOV网络、 AOE网络

- 有向无环图(DAG)
- 拓扑排序：AOV网络

有向无环图

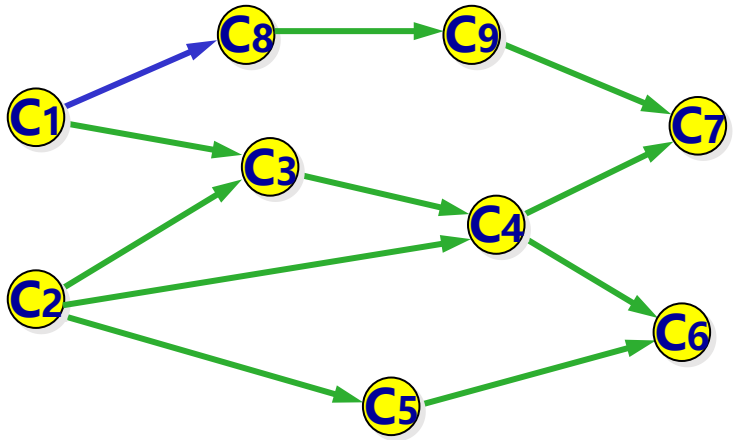
- 一个无环的有向图(DAG, directed acycline graph)，简称DAG图。
- 应用之一：是描述工程、系统进行过程的有效工具。
- 例如，我们接下来讲的活动网络(Activity Network)。

活动网络

- 计划、施工过程、生产流程、程序流程等都是“工程”。除了很小的工程外，一般都把工程分为若干个叫做“活动”的子工程。完成了这些活动，这个工程就可以完成了。
- 例如，计算机专业学生的学习就是一个工程，每一门课程的学习就是整个工程的一些活动。其中有些课程要求先修课程，有些则不要求。这样在有的课程之间有领先关系，有的课程可以并行地学习。

活动网络(Activity Network)

课程代号	课程名称	先修课程
C_1	高等数学	
C_2	程序设计基础	
C_3	离散数学	C_1，C_2
C_4	数据结构	C_3，C_2
C_5	高级语言程序设计	C_2
C_6	编译方法	C_5，C_4
C_7	操作系统	C_4，C_9
C_8	普通物理	C_1
C_9	计算机原理	C_8



学生课程学习工程图

AOV网络(Activity On Vertices)

AOV网络

AOV——结点表示活动

- 可以用有向图表示一个工程。在这种有向图中，用顶点表示活动，用有向边<V_i, V_j>表示活动V_i必须先于活动V_j进行。这种有向图叫做顶点表示活动的**AOV网络** (Activity On Vertices)。
- 在AOV网络中不能出现有向回路，即有向环。如果出现了有向环，则意味着某项活动应以自己作为先决条件。
- 因此，对给定的AOV网络，必须先判断它是否存在有向环。

拓扑排序(Topological Sort)

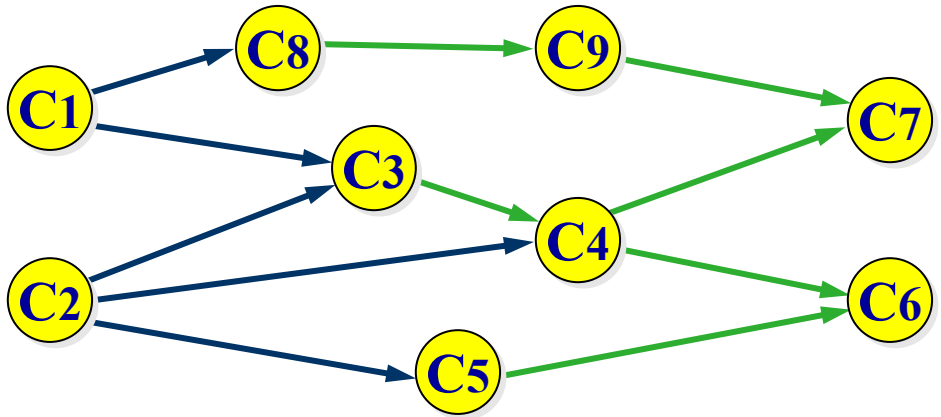
- 检测有向环的一种方法是对AOV网络构造它的**拓扑有序序列**。即将各个顶点(代表各个活动)排列成一个线性有序的序列，使得AOV网络中所有应存在的前驱和后继关系都能得到满足。
- 这种构造AOV网络全部顶点的拓扑有序序列的运算就叫做**拓扑排序**。
- 如果通过拓扑排序能将AOV网络的所有顶点都排入一个拓扑有序的序列中，则该网络中必定不会出现有向环。
- 如果AOV网络中存在有向环，此AOV网络所代表的工程是不可行的。

拓扑排序(Topological Sort)

- 例如，对学生选课工程图进行拓扑排序，得到的拓扑有序序列为

C_1 , C_2 , C_3 , C_4 , C_5 , C_6 , C_8 , C_9 , C_7

或 C_1 , C_8 , C_9 , C_2 , C_5 , C_3 , C_4 , C_7 , C_6



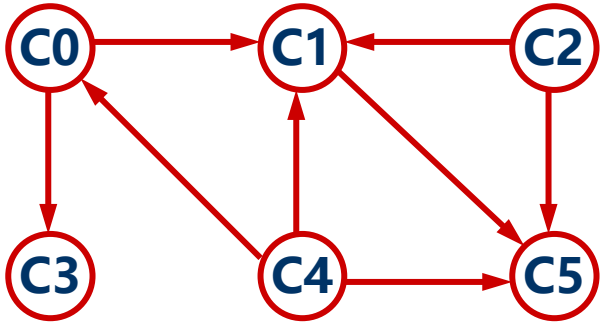
学生课程学习工程图

拓扑排序(Topological Sort)

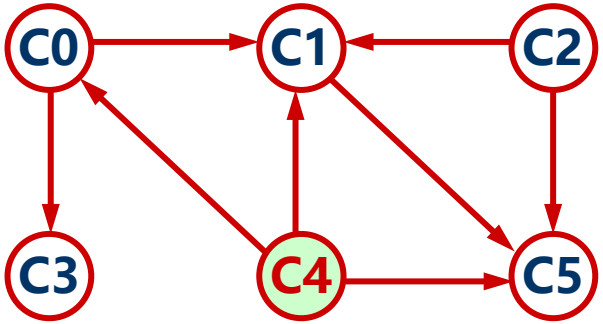
拓扑排序方法

- 1. 输入AOV网络。令 n 为顶点个数。
- 2. 在AOV网络中选一个没有直接前驱的顶点，并输出之;
- 3. 从图中删去该顶点，同时删去所有它发出的有向边;
- 4. 重复以上 2、3步, 直到
  - 全部顶点均已输出，拓扑有序序列形成，拓扑排序完成；或
  - 图中还有未输出的顶点, 但已跳出处理循环。说明图中还剩下一些顶点, 它们都有直接前驱。这时网络中必存在有向环。

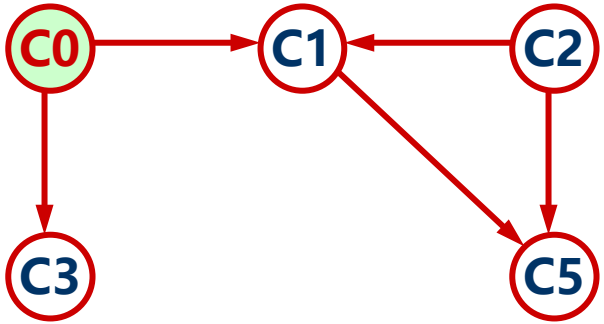
拓扑排序(Topological Sort)过程(1)



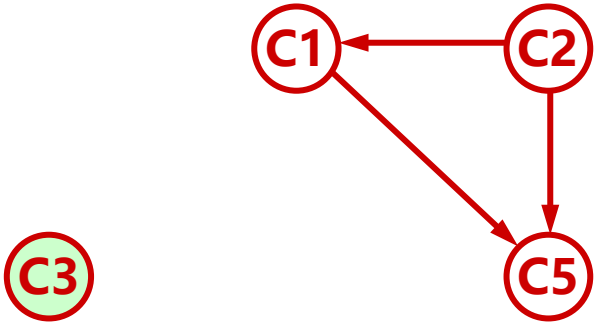
(a) 有向无环图



(b) 输出顶点C4

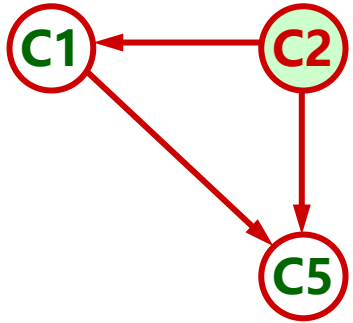


(c) 输出顶点C0

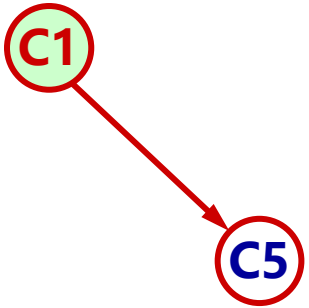


(d) 输出顶点C3

拓扑排序(Topological Sort)过程(2)



(e) 输出顶点C2



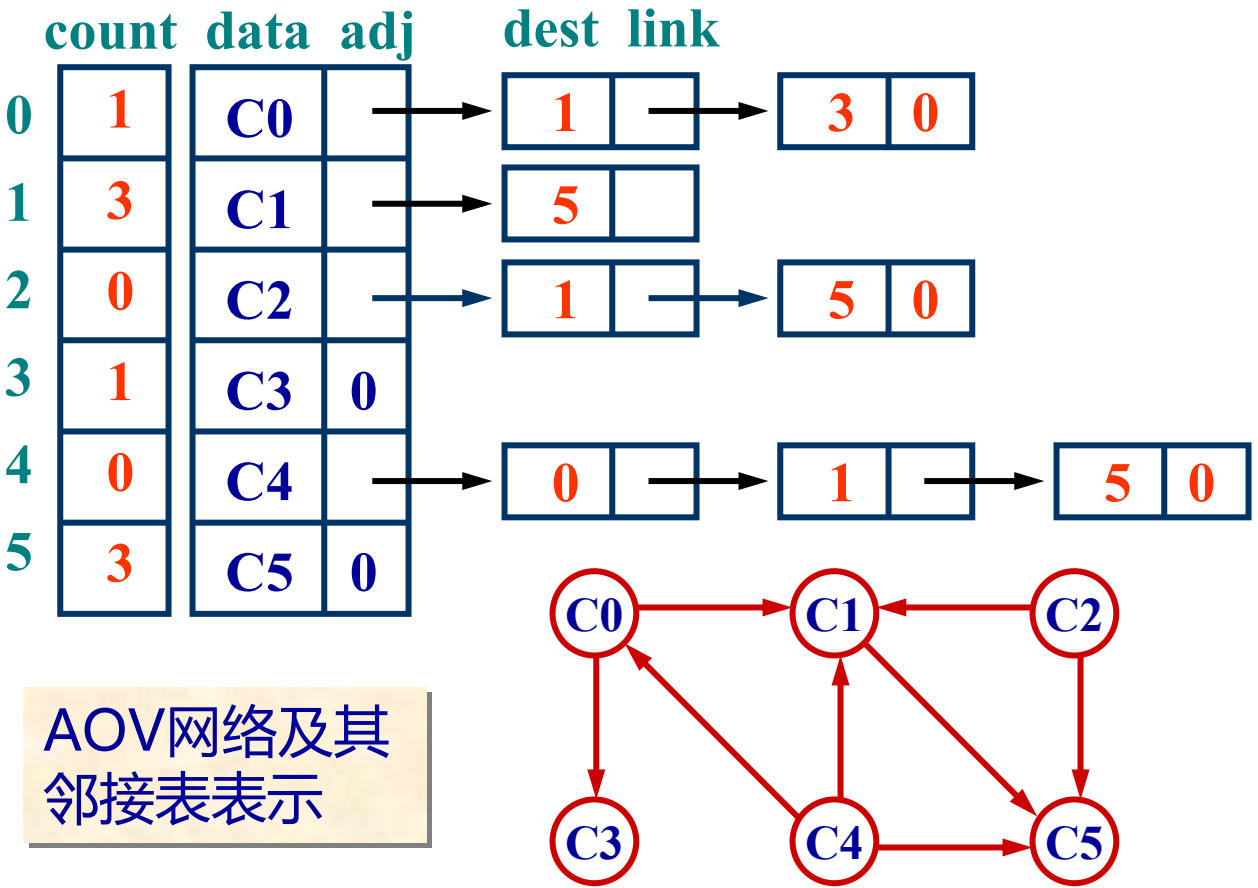
(f) 输出顶点C1

(g) 输出顶点C5

(h) 拓扑排序完成

- 最后得到的拓扑有序序列为 C4, C0, C3, C2, C1, C5。它满足图中给出的所有前驱和后继关系，对于本来没有这种关系的顶点，如C4和C2，也排出了先后次序关系。

AOV网络及其邻接表表示



AOV网络及其邻接表表示

7.2.2 图的存储结构-邻接表

- **邻接表：通过把顶点的所有邻接点组织成一个单链表来描述边**

```
#define MVNum 100                          //最大顶点数
typedef struct ArcNode{                    //边结点
    int adjvex;                            //该边所指向的顶点的位置
    struct ArcNode * nextarc;              //指向下一条边的指针
    EdgeData cost;                         //改造结点，typedef int EdgeData;
}ArcNode;
typedef struct VNode{
    VerTexType data;                       //顶点信息
    ArcNode * firstarc;                    //指向第一条依附该顶点的边的指针
}VNode, AdjList[MVNum];                    //AdjList表示邻接表类型
typedef struct{
    AdjList vertices;                      //邻接表
    int vexnum, arcnum;                    //图的当前顶点数和边数
}ALGraph;
```

AOV网络及其邻接表表示

- 在邻接表中增设一个数组count[]，记录各顶点入度。入度为零的顶点即无前驱顶点。
- 在输入数据前，顶点表VexList[]和入度数组count[]全部初始化。在输入数据时，每输入一条边<j,k>，就需要建立一个边结点，并将它链入相应边链表中，统计入度信息：

```
ArcNode *p = new ArcNode;
p->adjvex = k;                    //建立边结点
p->nextarc = G. AdjList[j]. firstarc;
G. AdjList[j]. firstarc = p;      //链入顶点 j 的边链表的前端
count[k]++;                       //顶点 k 入度加一
```

拓扑排序算法

- 在算法中，使用一个存放入度为零的顶点的链式栈，供选择和输出无前驱的顶点。
- 拓扑排序算法可描述如下：

a) 建立入度为零的顶点栈;
b) 当入度为零的顶点栈不空时，重复执行
   - 从顶点栈中退出一个顶点，并输出之;
   - 从AOV网络中删去这个顶点和它发出的边，边的终顶点入度减一;
   - 如果边的终顶点入度减至0，则该顶点进入度为零的顶点栈;
- 如果输出顶点个数少于AOV网络的顶点个数，则报告网络中存在有向环。

拓扑排序算法


```
1.  void TopologicalSort(AdjGraph& G) {
2.      int i, j, k;   stack S;   InitStack(S);
3.      //入度为零的顶点栈初始化
4.      for (i = 0; i < G.n; i++)
5.          if (count[i] == 0)  Push(S, i);
6.          //入度为零顶点进栈
7.      for (i = 0; i < n; i++)                    //期望输出 n 个顶点
8.          if (StackEmpty(S)) {                   //中途栈空,转出
9.              cout << "网络中有回路!\n"; return;
10.         }
11.         else {                                 //继续拓扑排序
```

拓扑排序算法


```
1.          Pop(S, j);                                  //退栈
2.          cout << j << endl;                          //输出拓扑序列
3.           ArcNode * p = G. AdjList[j]. firstarc;
4.          while (p != NULL) {                         //扫描出边表
5.              k = p-> nextarc;                        //另一顶点
6.              if (--count[k] == 0)                    //顶点入度减一
7.                  Push(S, k);                         //入度减至零, 进栈
8.              p = p->nextarc;
9.          }
10.     }
11.}
```

AOE网络

关键路径与AOE网络

AOE——结点表示事件，边代表活动

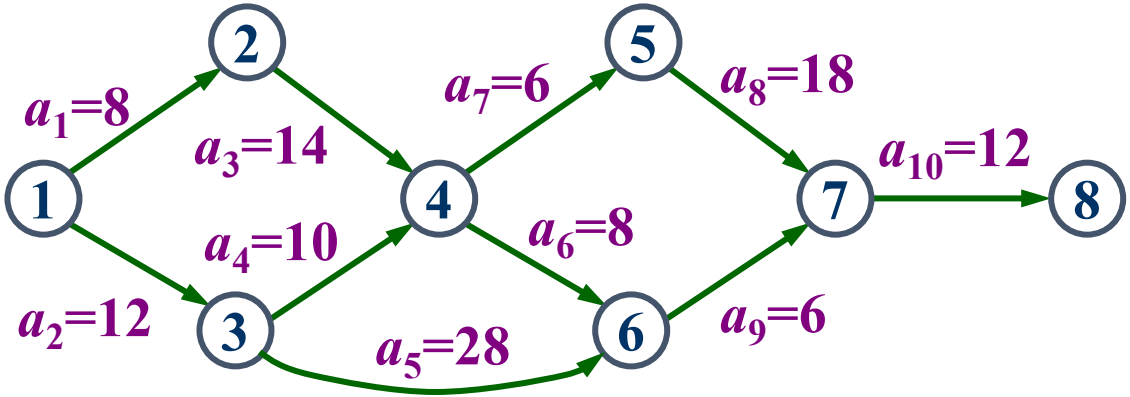
- 如果在无有向环的带权有向图中，用有向边表示一个工程中的**活动**(Activity)，用边上权值表示活动**持续时间** (Duration)，用顶点表示**事件**(Event)，则这样的有向图叫做用边表示活动的网络，简称 **AOE (Activity On Edges) 网络。**
- AOE网络在某些工程估算方面非常有用。例如，可以使人们了解：
  - 完成整个工程至少需要多少时间（假设网络中没有环）？
  - 为缩短完成工程所需的时间，应当加快哪些活动?

关键路径与AOE网络

- 从开始点到各个顶点，以至从开始点到结束点的有向路径可能不止一条。这些路径的长度也可能不同。完成不同路径的活动所需的时间虽然不同，但只有各条路径上所有活动都完成了，整个工程才算完成。
- 因此，完成整个工程所需的时间取决于从源点到汇点的最长路径长度，即在这条路径上所有活动的持续时间之和。这条路径长度最长的路径就叫做**关键路径**(**Critical Path**)。
- 要找出关键路径，必须找出关键活动，即不按期完成就会影响整个工程完成的活动。

关键路径与AOE网络

- 关键路径上的所有活动都是关键活动。因此，只要找到了关键活动，就可以找到关键路径。例如，下图就是一个AOE网。



- 顶点 1 是整个工程的开始点，顶点 8 是整个工程的结束点。a_i = ...是各边上的活动及持续时间。

几个与计算关键活动有关的量

1. 事件V_i的最早可能开始时间Ve(i)

   它是从开始点V_1到顶点V_i的最长路径长度。

2. 事件V_i的最迟允许开始时间Vl[i]

   它是在保证结束点V_n在Ve[n]时刻完成的前提下，事件V_i的允许的最迟开始时间。

3. 活动a_k的最早可能开始时间 e[k]

   设活动a_k在边<V_i，V_j>上，则e[k]是从开始点V_1到顶点V_i的最长路径长度。因此

   $$e[k] = Ve[i]。$$

关键路径与AOE网络

4. 活动a_k的最迟允许开始时间 l[k]

   它是在不会引起时间延误的前提下，该活动允许的最迟开始时间。

   l[k] = Vl[j] - dur(<i, j>)。

   其中，dur(<i, j>)是完成 a_k 所需的时间。

5. 时间余量 l[k] - e[k]

   表示活动 a_k 的最早可能开始时间和最迟允许开始时间的时间余量。l[k] == e[k] 表示活动 a_k 是没有时间余量的关键活动。

- 为找出关键活动，需要求各个活动的 e[k] 与 l[k]，以判别是否 l[k] == e[k]。

关键路径与AOE网络

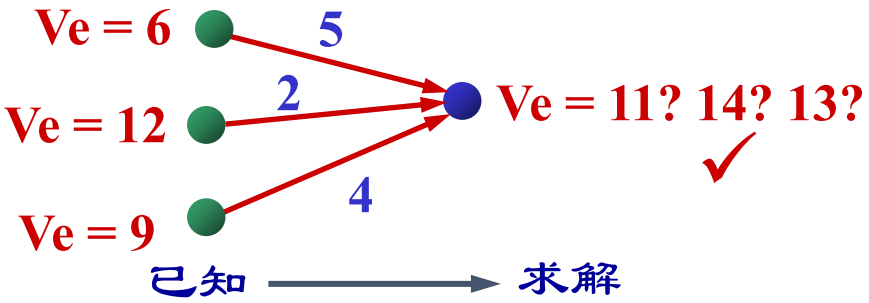
- 为求得e[k]与l[k]，需要先求得从开始点V1 到各个顶点Vi 的Ve[i] 和Vl[i]。
- 求Ve[i]的递推公式

a) 从Ve[1] = 0开始，向前递推

$$Ve[i] = \max_{j}\{Ve[j] + dur(<V_j, V_i>)\},$$

$<V_j, V_i> \in S2$，　i = 2, 3, …, n

S2 是所有指向V_i的有向边$< V_j , V_i >$的集合。

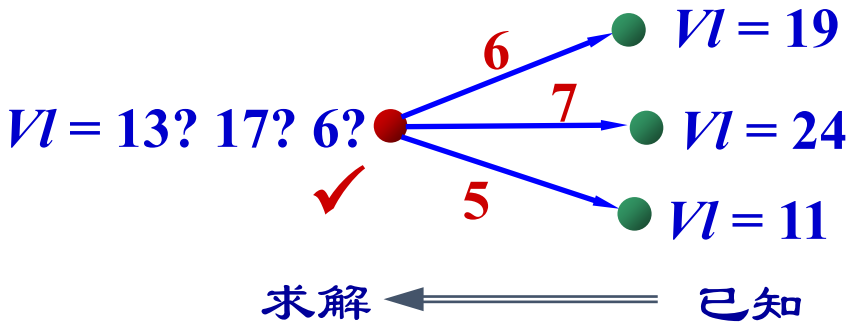


b) 从Vl[n] = Ve[n]开始，反向递推

$<V_i, V_j> \in S1, i = n-1, n-2, \ldots, 1$

$$Vl[i] = \min_j \{Vl[j] - dur(<V_i, V_j>)\},$$

S1是所有源自V_i的有向边$< V_i , V_j >$的集合。



- 这两个递推公式的计算必须分别在拓扑有序及逆拓扑有序的前提下进行。

关键路径与AOE网络

- 设活动a_k (k = 1, 2, ..., e) 在带权有向边<V_i, V_j> 上，其持续时间用dur (<V_i , V_j >) 表示，则有

  e[k] = Ve[i];

  l[k] = Vl[j] - dur(<V_i , V_j >)； k = 1, 2, ..., e。

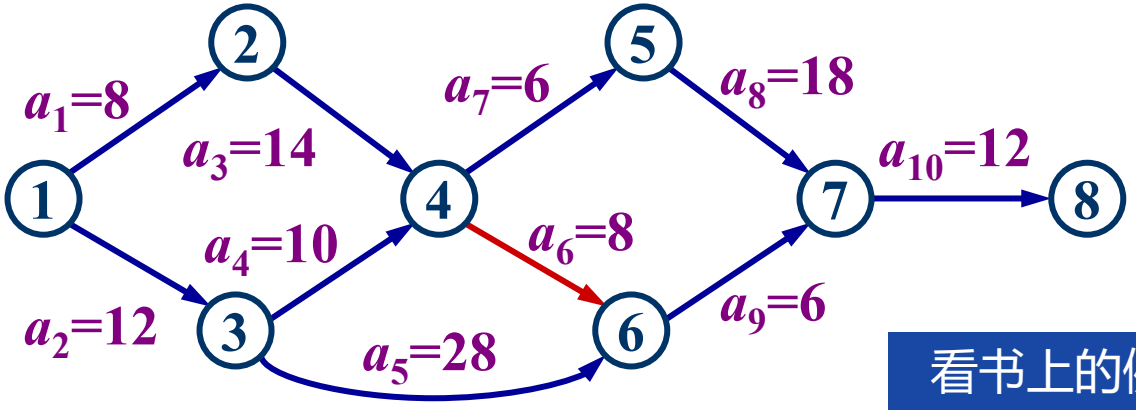
这样就得到计算关键路径的算法。

- 为了简化算法，假定在求关键路径之前已经对各顶点实现了拓扑排序，并按拓扑有序的顺序对各顶点重新进行了编号。

关键路径与AOE网络

	1	2	3	4	5	6	7	8
Ve	0	8	12	22	28	40	46	58
Vl	0	8	12	22	28	40	46	58

	1	2	3	4	5	6	7	8	9	10
e	0	0	8	12	12	22	22	28	40	46
l	0	0	8	12	12	32	22	28	40	46



看书上的例子更清晰

注意

- 所有顶点按拓扑有序的次序编号
- 仅计算 Ve[i] 和 Vl[i] 是不够的，还须计算 e[k] 和 l[k]。
- 不是任一关键活动加速一定能使整个工程提前。
- 想使整个工程提前，要考虑各个关键路径上所有关键活动。

作业预告

- 图的存储、创建及遍历~