第八章 图结构

§8.1 基本概念

§8.2 图的存储结构

§8.3 图的遍历

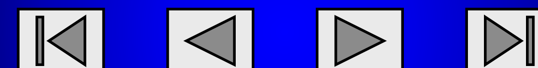
§8.4 连通性及最小生成树

§8.5 有向无环图及应用

§8.6 最短路径

习题

Meng Yingjie



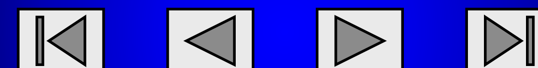
一.概论

1.与**线性结构**、**树形结构**的差异。

2.地位：非常重要的非线性结构。

“图论”已经广泛地应用于许多科学技术领域，诸如电子线路分析、系统工程、寻找最短路径、化学成分分析、统计力学、遗传学、控制论等。除自然科学外还广泛应用于社会科学和人文科学等许多学科。

由于这些领域把图结构作为解决问题的手段之一，因此数据结构需要研究图结构。



3.讨论内容

"图论"是离散数学的主要内容之一。本课程不讨论它的基础理论知识,只是应用它的知识来讨论在计算机上如何表示图结构。

- 计算机中图的表示方法
- 图的遍历
- 求生成树
- 找最短路径
- 拓扑排序
- 求关键路径等

Meng Yingjie

二.图的概念

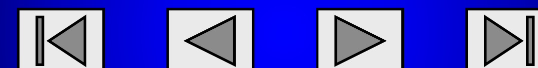
（仅介绍用到的）

1.图(graph)

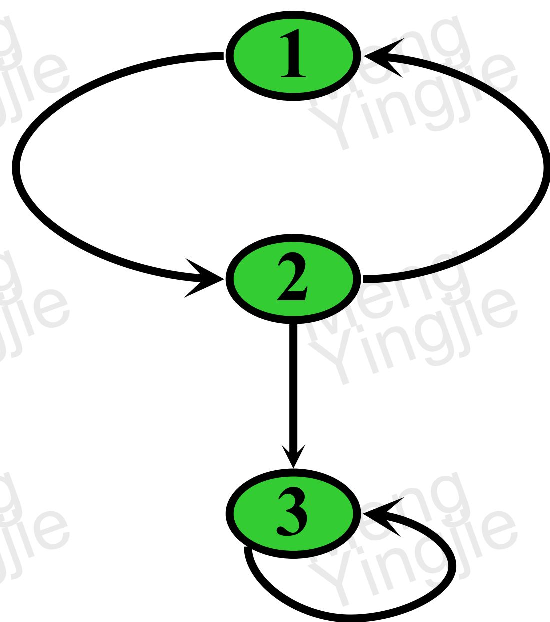
由n (n≥1)个结点$v_1, v_2,\ldots, v_n$ 构成的数据G称为图,若结点集V={$v_1, v_2,\ldots, v_n$}上定义的称为后继的关系E是非自反的。

可表示为G=(V,E)， V称为顶点集，E称为边集。

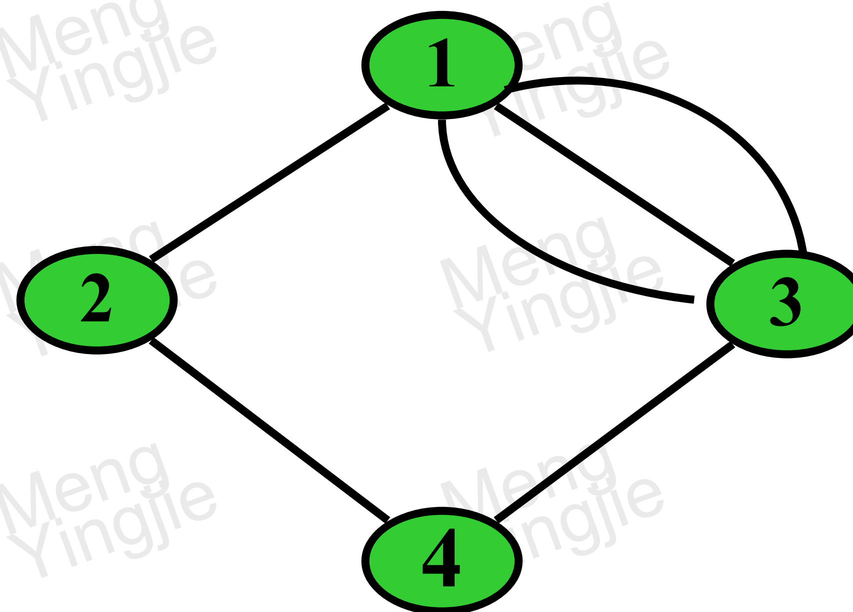
Meng Yingjie



这里不研究的图：



带自身环的图



多重图

与图论中讨论的图的差别：相当于**简单图**，即不包含重边和环(自回路)，目的，适应算法的有限性要求。

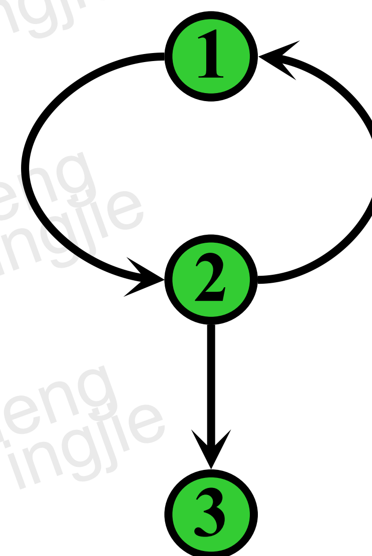
2.有向图(directed graph)

在图G中，若每个关系都是顶点的有序对，则称G为有向图。

一般用尖括弧表示顶点的有序对。在图示表示中，用有向线段指明了次序(方向)。例如：

$V(G)=\{v_1,v_2,v_3\}$ ，

$E(G)=\{<v_1,v_2>, <v_2,v_1>, <v_2,v_3>\}$



Meng Yingjie

在有向图中,若 $<v_i, v_j> \in E(G)$

则称 v_i是边的始点(或尾);

v_j是边的终点(或头);

并称 v_i邻接到 v_j , v_j是从 v_i邻接过来的;

称边 $<v_i, v_j>$ 关联(或依附)于顶点 v_i和 v_j.

推论:在n个结点的有向图中,其最大边数为n×(n–1).

把等于n×(n–1)条边的图称有向完全图。

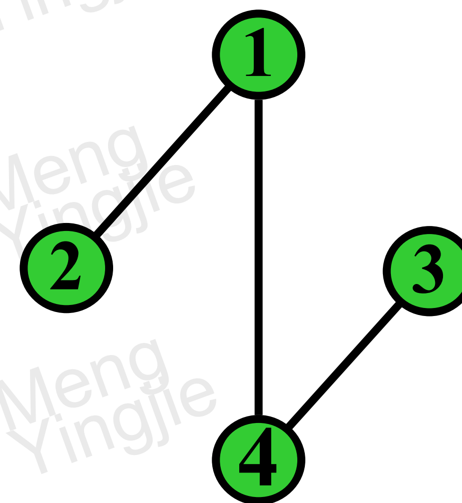
Meng Yingjie

3.无向图(undirected graph)

在图G中,如果每个关系都是顶点的无序对，则称G为无向图。

一般用圆括弧表示顶点的无序对，在图示表示中，用无向线段表示(方向)。

在无向图中(v_1,v_2)和(v_2,v_1)这两个结点偶对表示同一条边。例：



例：图G，$V(G)=\{v_1,v_2,v_3,v_4\}$ ，

$E(G)=\{(v_1,v_2), (v_1,v_4), (v_3,v_4)\}$

Meng Yingjie

在无向图中,若$(v_i,v_j)\in E(G)$,

则称v_i和v_j是邻接的;

并称边(v_i,v_j)关联(或依附)于顶点v_i和v_j.

推论:在n个结点的无向图中,其最大边数为$n\times(n-1)/2$.

把等于$n\times(n-1)/2$条边的图称无向完全图。

根据边的数目可以把图划分为稀疏图和稠密图.

稀疏图(Sparse graph):边数 $< n\log_2 n$

稠密图(Dense graph):

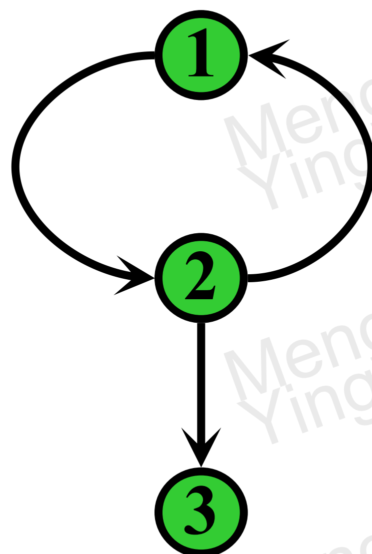
4.顶点的度、入度、出度

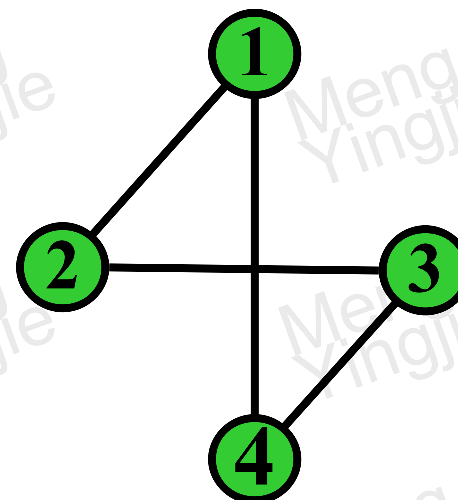
在图中,顶点的度(degree), 就是与该顶点关联的边的数目。

若G为有向图：则

- 把以v_i为终点(头)的边的数目称作v_i的入度(incoming degree ,in-degree)；
- 把以v_i为始点(尾)的边的数目称作v_i的出度(out going degree)。
- v_i的度为v_i的出度和入度之和。

例：图中顶点度的计算：





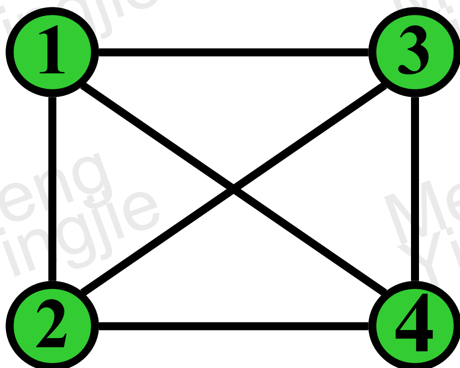
推论:设图G有n个结点，t条边，若d_i为顶点v_i的度，显然有：

$$t=\frac{1}{2}\sum_{i=1}^{n}d_i$$

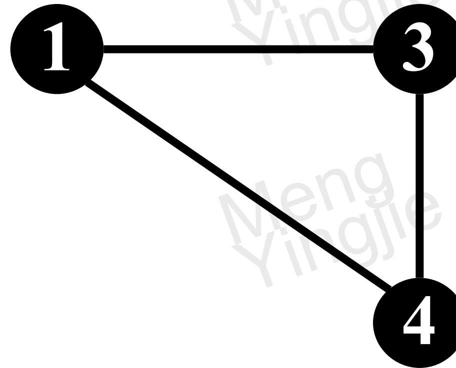
在有向图中,**出度为0**的顶点称**终端顶点**(叶子)。

5.子图(subgraph)

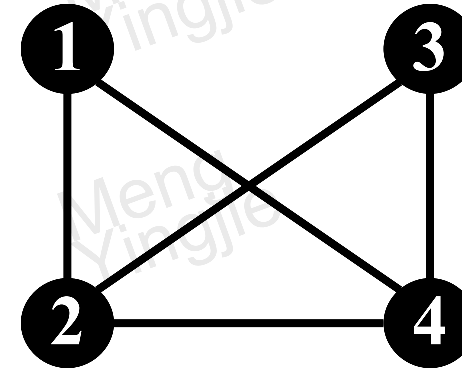
设图G=(V,E)，如果有图G′=(V′,E′)，且E′⊆E, V′⊆V，则称G′为G的子图。



(a)



(b)



(c)

例如，图(b)、(c)设都是(a)的子图。

如果图G的子图包含G的所有顶点，则该子图称为G的生成子图。

6.路径、回路、图根

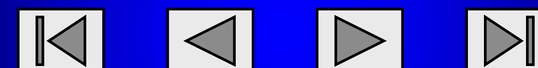
对于图G=(V,E),若存在结点序列$v_p,v_{i1},v_{i2},\ldots,v_{im},v_q$使得$(v_p,v_{i1}),(v_{i1},v_{i2}),\ldots,(v_{im},v_q)$都在E(G)中(对于有向图使得$\langle v_p,v_{i1}\rangle,\langle v_{i1},v_{i2}\rangle,\ldots,\langle v_{im},v_q\rangle$都在E(G)中), 则称从顶点$v_p$到$v_q$存在一条**路径**(或通路，path)，**路径长度**定义为这条路径上边的数目。

(路径：一个特定的点边交替序列)

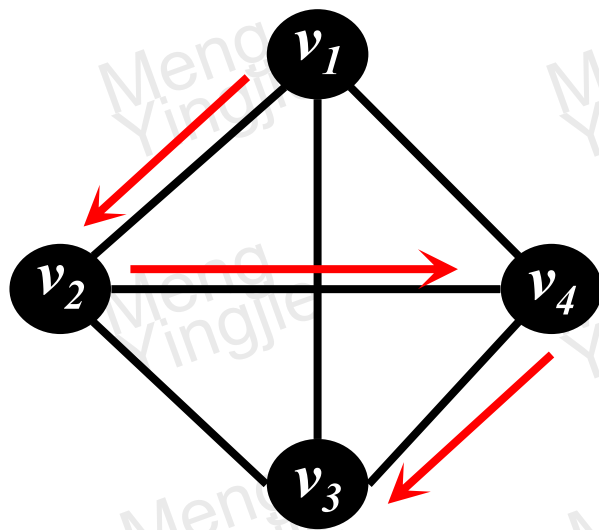
v_p到v_q的路径可以简写为：$v_p,v_{i1},v_{i2},\ldots,v_{im},v_q$

如果一条路径上的顶点除v_p和v_q可以相同以外其它顶点都不相同，则此路径为一**简单路径**。

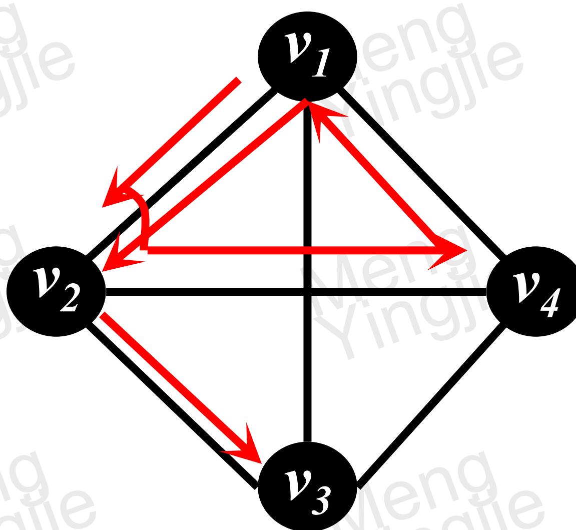
Meng Yingjie



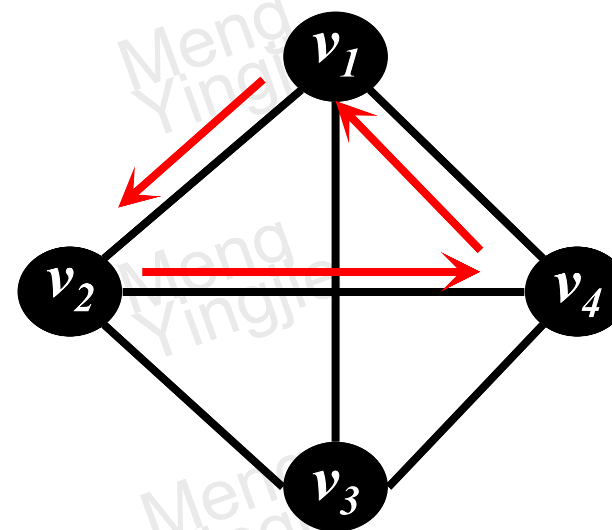
把$v_p=v_q$的简单路径，称作回路(或环，loop)。



v_1,v_2,v_4,v_3
简单路径



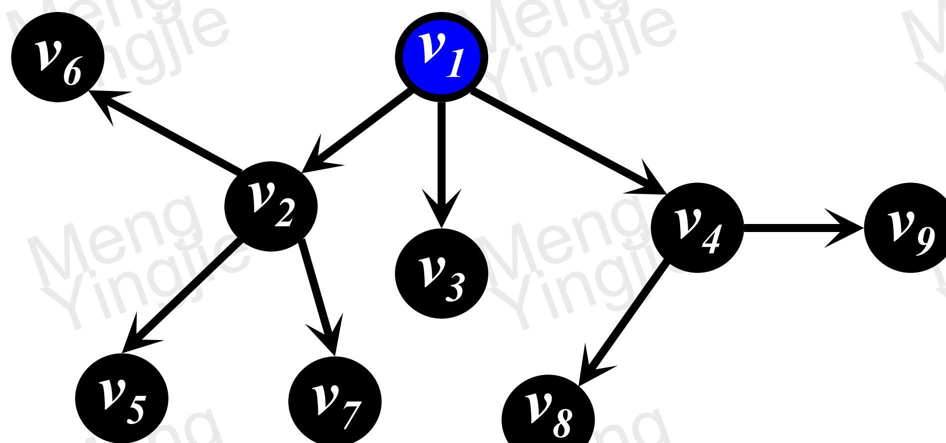
v_1,v_2, v_4,v_1,v_2,v_3
非简单路径

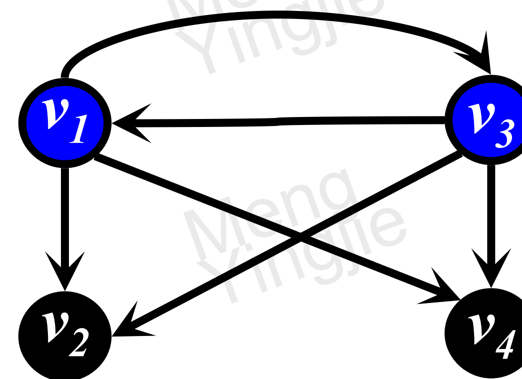


v_1,v_2, v_4,v_1
回路

Meng Yingjie

若在一个有向图中存在一个顶点v_0，从此顶点有路径可以到达图中其它所有顶点,则此有向图称为是**有根的图**， v_0称作**图根**。





Meng Yingjie

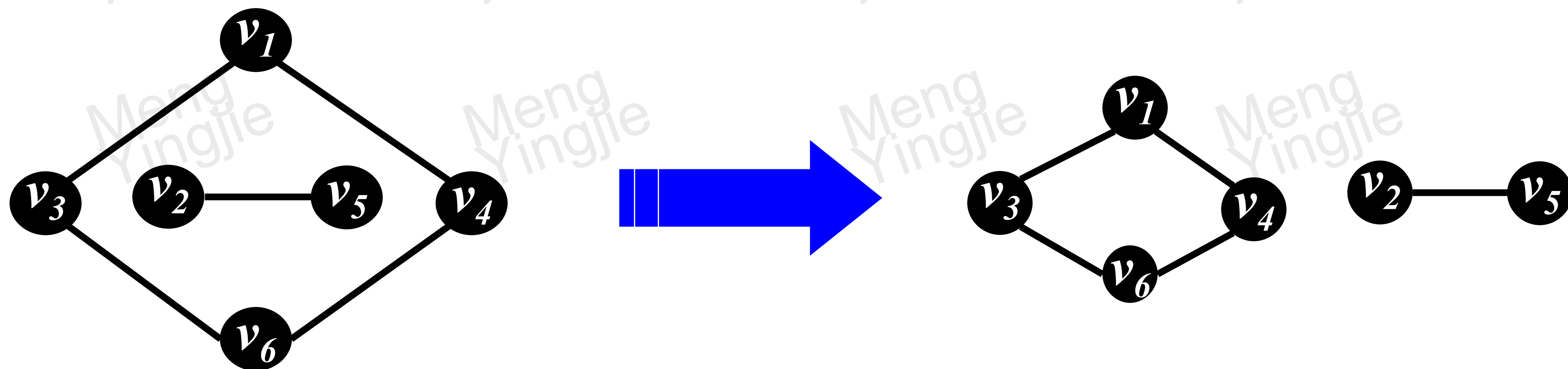
7.连通性

(1).无向图

两个顶点的连通：在无向图G中，顶点u和v(u≠v)之间存在一条路径，则称顶点u和顶点v是连通的。

图的连通：对于无向图G中的任意两个顶点u,v(u≠v)都是连通的则称此无向图是连通的，或称G为连通图。

一个无向图的连通分量定义为该图的最大(或极大)连通子图。



(2).有向图

两个顶点的连通：在有向图G中,顶点*u*和*v*(u≠v)之间存在一条从u到v和v到u的(有向)路径，则称顶点u和顶点v是连通的。

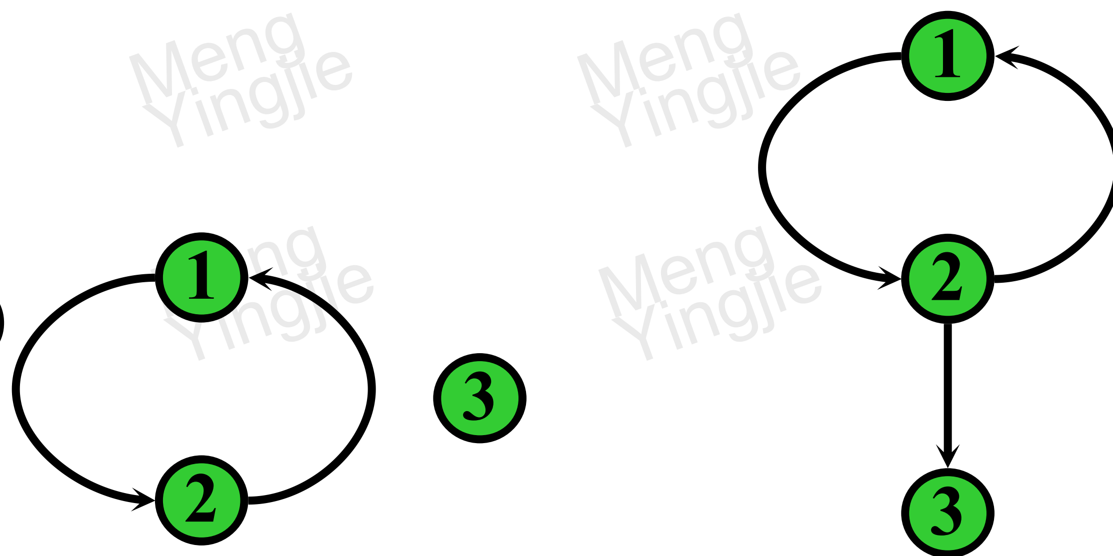
图的强连通：有向图G中的任意两个顶点*u*,*v*(*u*≠*v*)都是连通的则称此有向图是强连通的,或称G为强连通图。

一个有向图的强连通分量定义为该图的最大(或极大)强连通子图。

该图有两个强连通分量。

G_1=({1,2} , {<1,2>,<2,1>})

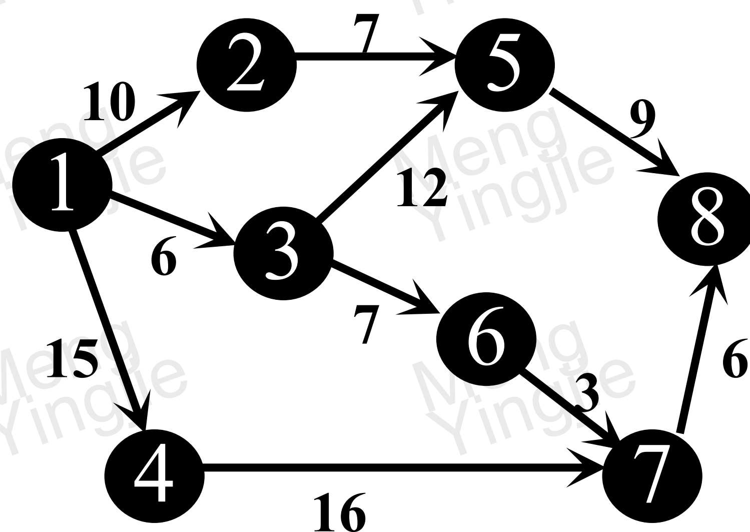
G_2=({3} , Φ)

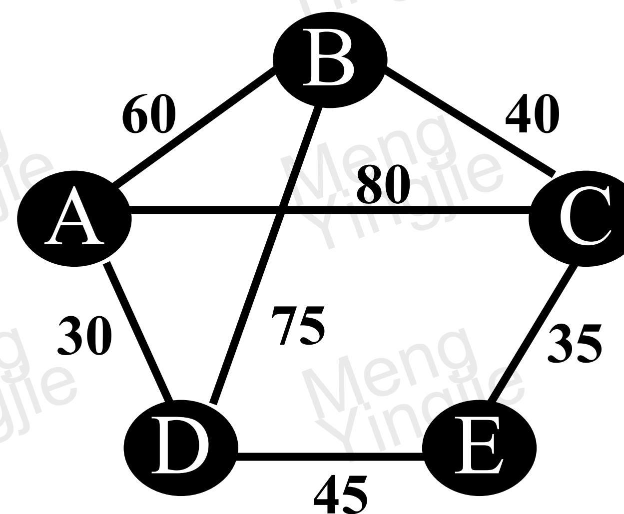


8.网

给图的每一条边加一个(非负的)数，这个与图的边相关的数值称为权(weight)。带权的图称为网。

带权的连通图称为网络。





Meng Yingjie

本节结束

引言：

由于图结构比树结构更复杂,故采用多重链表较为适宜,每个结点除一个数据项外,包含若干指针项。

但这种结构不便于进行图的各种运算，而且指针域的设置较为困难，指针关系复杂。

因此，要针对具体的图和具体的运算来灵活选取存储方式。

Meng Yingjie

最常用的有三种方式

它们都侧重于研究某种关系。

- 邻接矩阵

  侧重于处理顶点与顶点之间的关系，顶点具体情况不太关心。

- 邻接表

  侧重于处理顶点信息，同时必须关注顶点到顶点之间的关系。

- 邻接多重表

  侧重于处理一条完整边的处理(顶点、边)。

Meng Yingjie

一.邻接矩阵

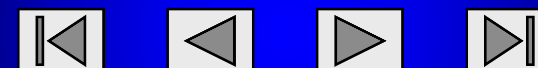
定义： 表示顶点之间邻接关系的矩阵称邻接矩阵 (adjacency matrix)。

1.图非网时 （用boolean矩阵表示。）

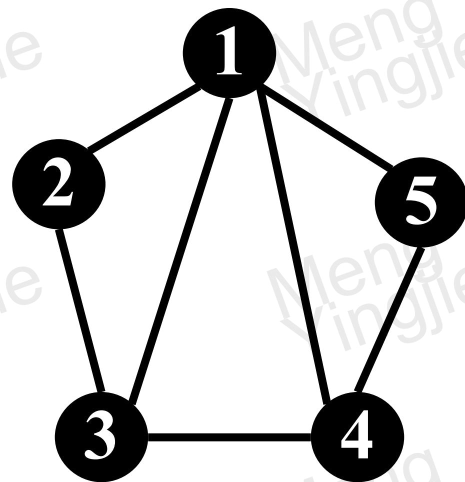
设图有n≥1个顶点，则图G的邻接矩阵定义为n阶方阵M，M满足下列性质：

$$m_{ij}=\begin{cases}1, & 若(v_i,v_j)或<v_i,v_j>\in E(G)\\ 0, & 反之\end{cases}$$

Meng Yingjie



(1).对于无向图：



$$\begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{array}\begin{array}{c} \begin{array}{ccccc} v_1 & v_2 & v_3 & v_4 & v_5 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

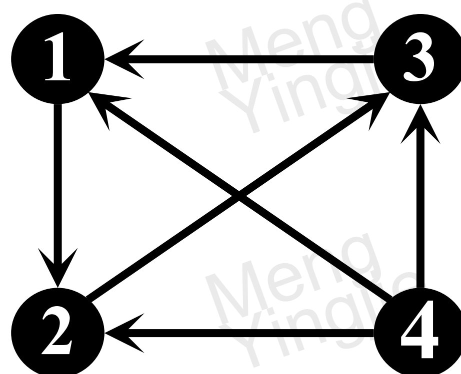
特点：

①.为对称矩阵

②.矩阵第i行之和是顶点v_i的度.

Meng Yingjie

(2).对于有向图:



$$
\begin{array}{c c}
 & \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array} \\
\begin{array}{c} v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} &
\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \end{bmatrix}
\end{array}
$$

特点:

①.为非对称矩阵

②.矩阵第i行之和是顶点v_i的出度.

③.矩阵第i列之和是顶点v_i的入度.

Meng Yingjie

(3).邻接矩阵需注意的问题

①矩阵与结点的标定次序有关，例如v_1和v_2对换，新的邻接矩阵是原邻接矩阵第1、2行对调。实际应用中不考虑这种情形。

②矩阵有许多重要特性，例如：

易于(随机访问)确定结点之间是否有边；

两个结点之间是否有长度为m的路径的数目可以通过邻接矩阵运算得出：

设A(G)是图G的邻接矩阵，则$(A(G))^m$的元素$a_{ij}^{(m)}$等于G中连接v_i与v_j的长度为m的路径的数目。

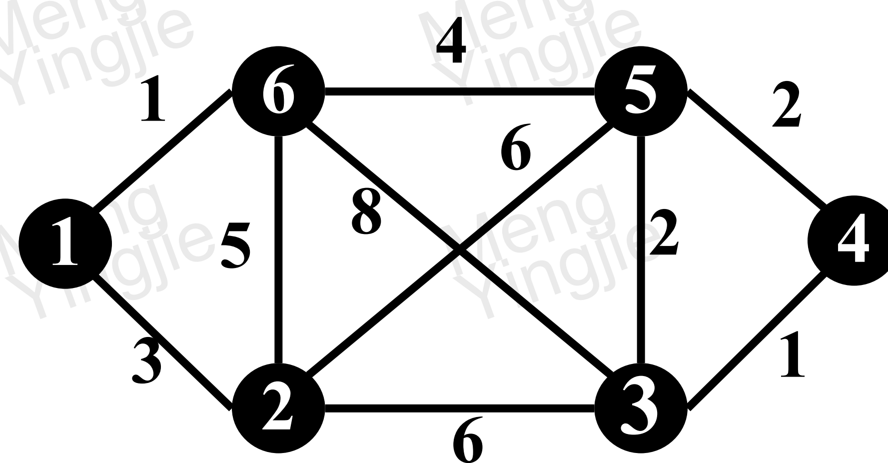
但这里不重点研究这些特性。

2.网的邻接矩阵

设图有n≥1个顶点，则图G的邻接矩阵定义为n阶方阵M，M满足下列性质：

$$m_{ij}=\begin{cases} w_{ij}, & \text{若}(v_i,v_j)\text{或}\langle v_i,v_j\rangle\in E(G) \\ 0 \text{ 或 } \infty, & \text{反之} \end{cases}$$

$$\begin{bmatrix} \infty & 3 & \infty & \infty & \infty & 1 \\ 3 & \infty & 6 & \infty & 6 & 5 \\ \infty & 6 & \infty & 1 & 2 & 8 \\ \infty & \infty & 1 & \infty & 2 & \infty \\ \infty & 6 & 2 & 2 & \infty & 4 \\ 1 & 5 & 8 & \infty & 4 & \infty \end{bmatrix}$$



二.邻接表(Adjacency List)

在这种表示中，对图的**每个顶点建立一个链表**，也称邻接链表表示。

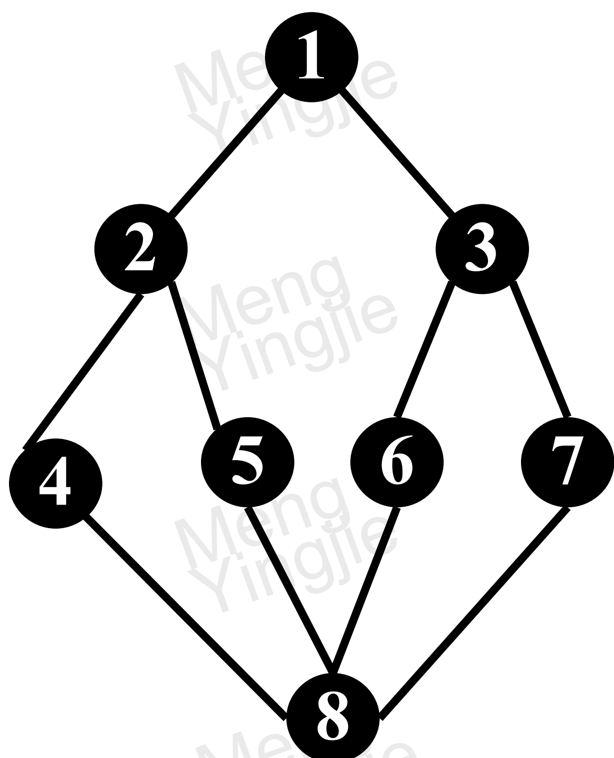
即有n个结点时有n个链表，第i个链表中的结点：

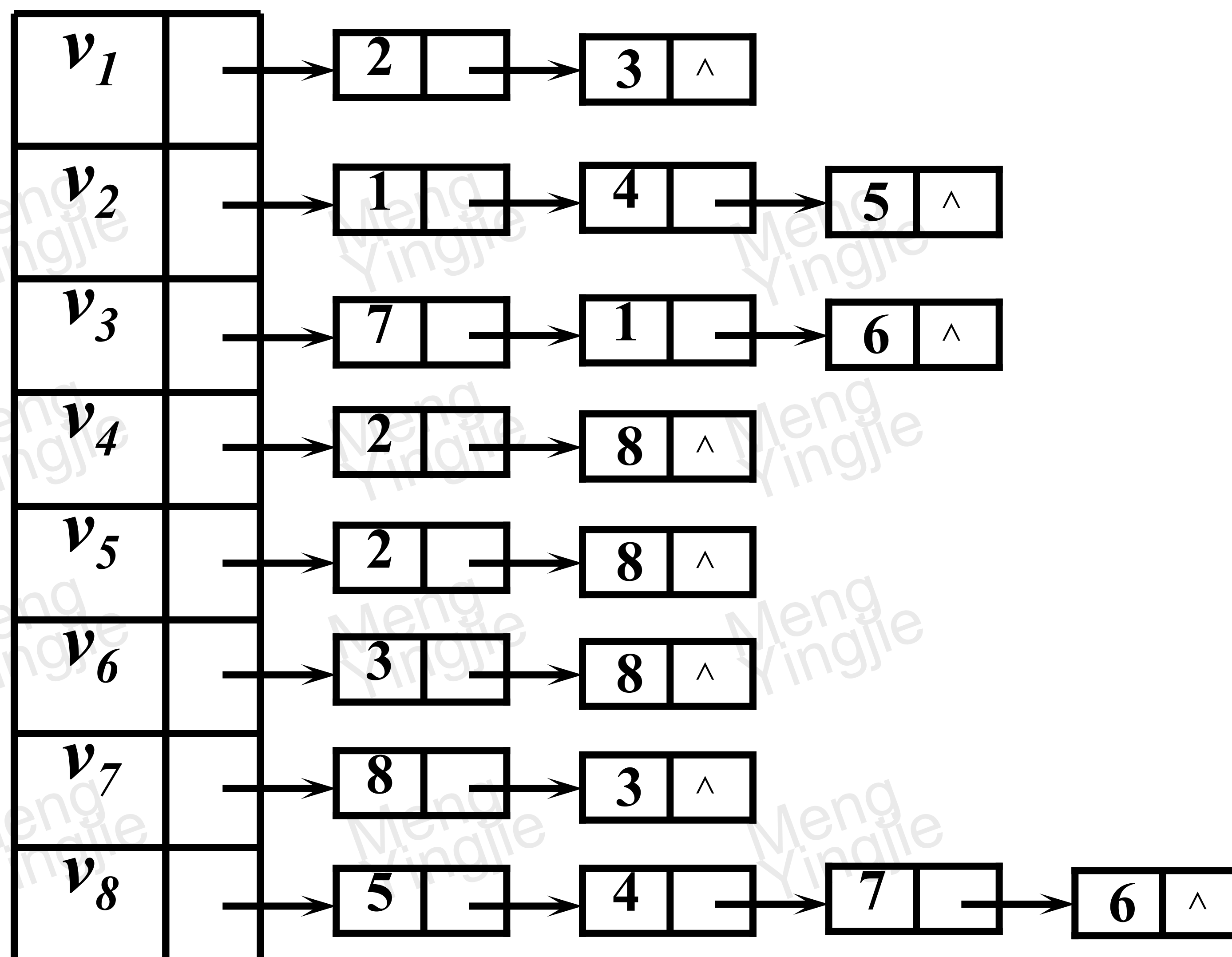
在无向图中：是与v_i邻接的所有结点的收集。

在有向图中：是以v_i为始点的所有终点的收集。

Meng Yingjie

举例：无向图





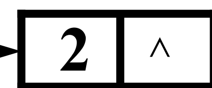
第i个链表中的结点个数： v_i的度。

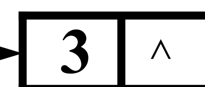
总表结点数目：2e

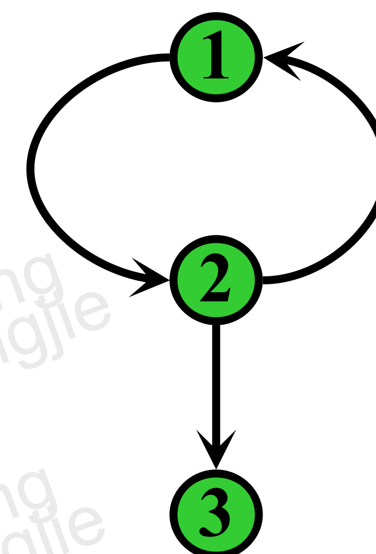
结点间是否有边判别比邻接矩阵困难。

举例： 有向图

v_1	→
v_2	→
v_3	^







第i个链表中的结点个数： v_i的出度。

总表结点数目： e； 总空间： n+e

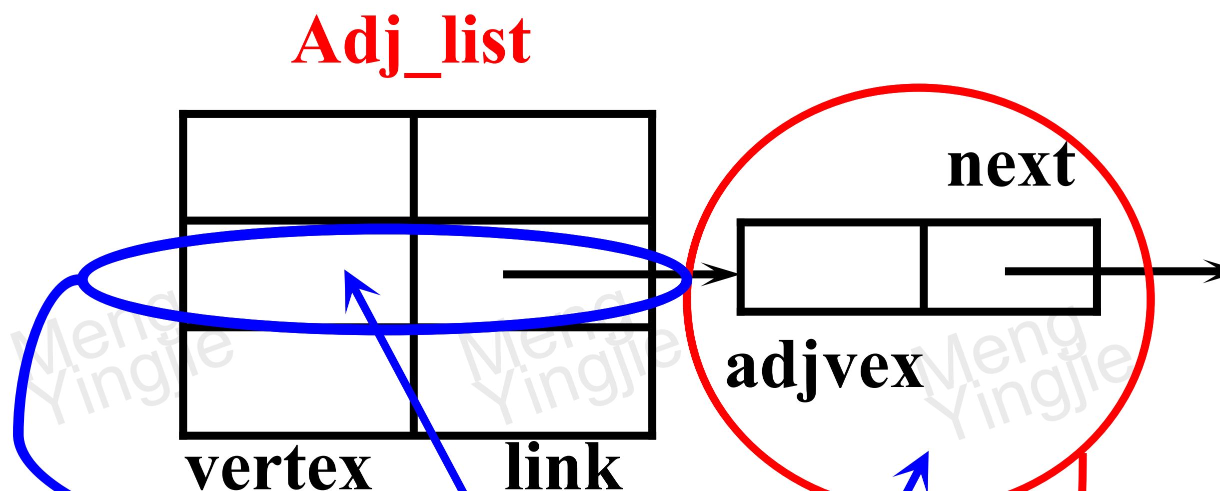
结点间是否有边判别比邻接矩阵更困难。

为获取邻接过来的结点可以建立逆邻接表。

v_1	→	2 ^
v_2	→	1 ^
v_3	→	2 ^

对于网的邻接表，表结点可以再增加一个数据项。

类型定义：

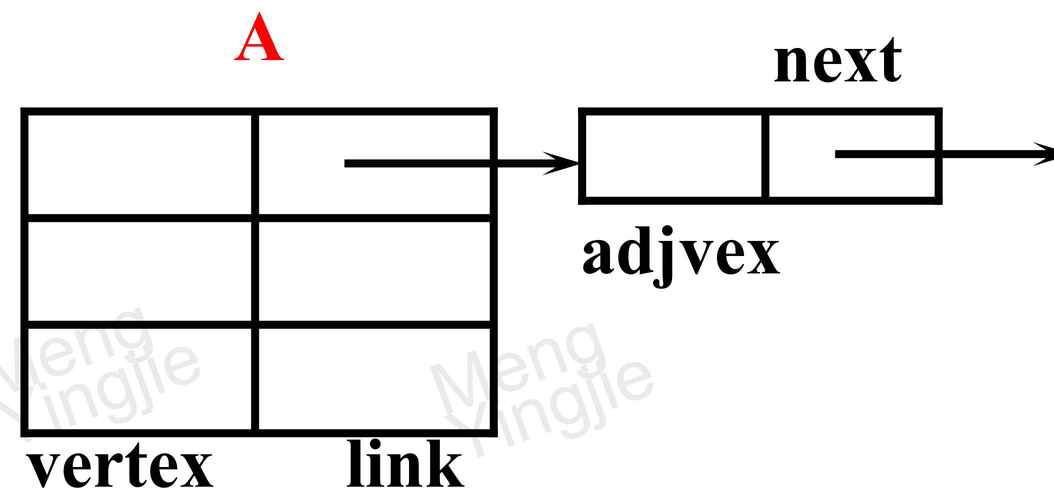



```
TYPE  Adjlink=↑node1;
          node1=RECORD
                  adjvex: 1..max;
                  next: adjlink
              END;
          node2=RECORD
                  vertex:VertexType;
                  link: adjlink
              END;
      Adj_list=ARRAY [1..max] OF node2;
```

显然，须要定义两种类型的结点

Meng Yingjie

建立初始邻接表



对有向图G,建立其邻接表A。

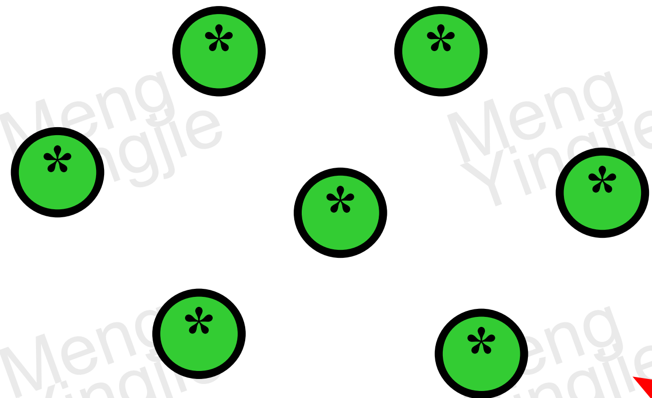
方法步骤分析(人工产生图分两个阶段):

(1) 逐个画出图的全部结点

(2) 逐个画出图的每条边，即线段

将以上过程细化对应到计算机模拟

(1)逐个画出图的全部结点，即：



相当于先不考虑邻接关系(link)逐个输入每个结点(vertex)，此处不妨假设结点信息就是节点编号。

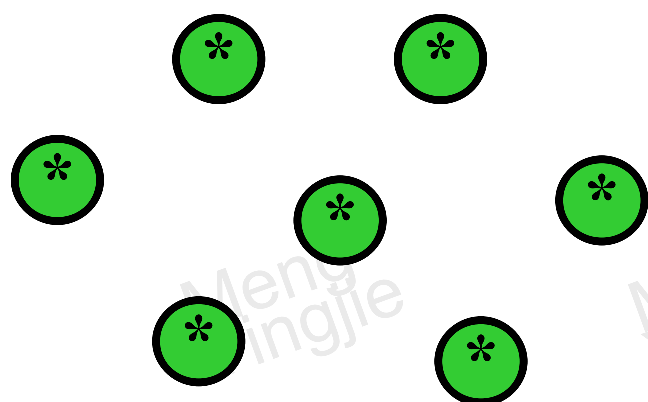
A

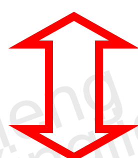
	vertex	link
1	*	nil
2	*	nil
i	*	nil
n	*	nil

用动作来体现的话，就是一个从1→n的重复如下工作:

A[i].vertex ← 第 i 个顶点的数据

A[i].link ← *nil*





A

	vertex	link
1	*	nil
2	*	nil
i	*	nil
n	*	nil

用动作来体现的话，就是一个从1→n的重复如下工作：

A[i].vertex ← 第 i 个顶点的数据

A[i].link ← *nil*

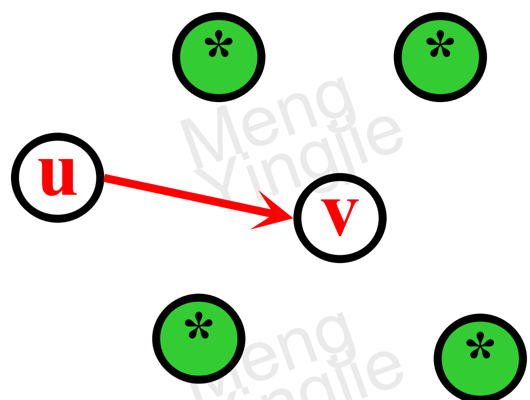
假设节点数据简单化：用编号表示。

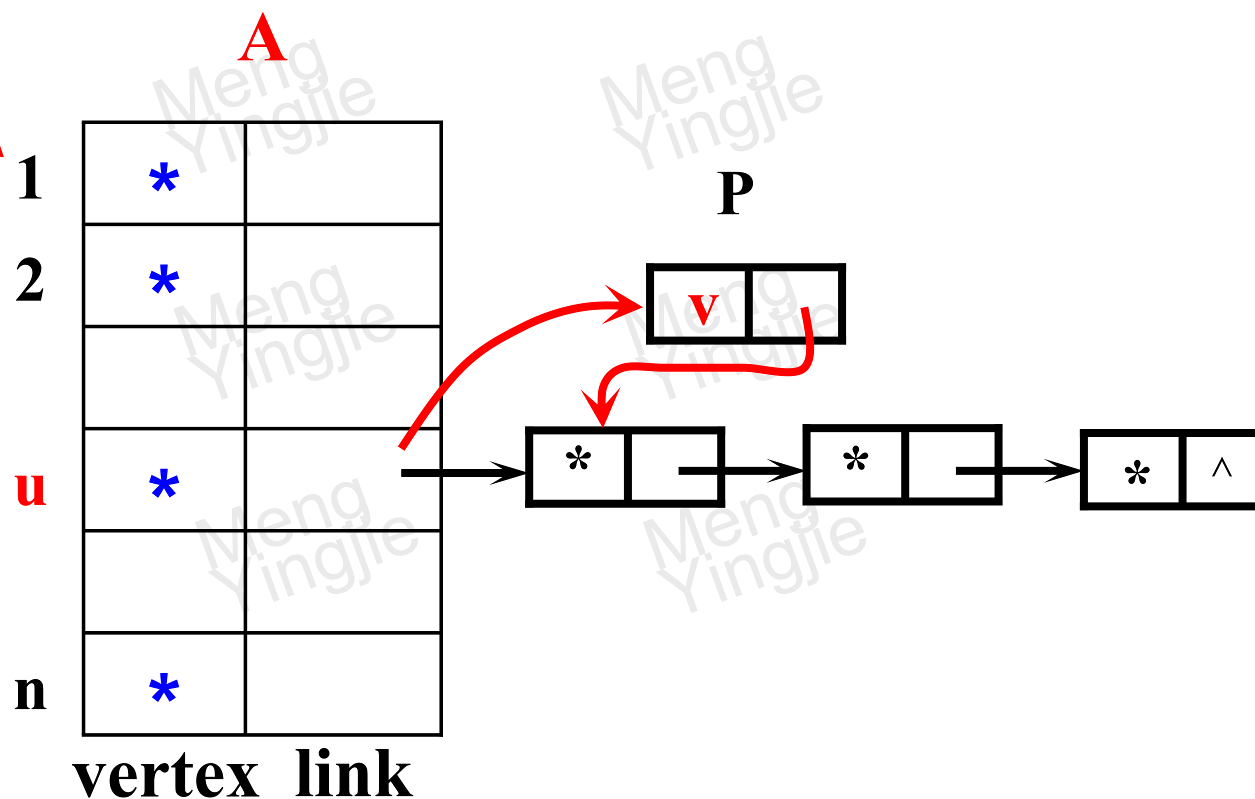
FOR i=1 TO n DO

【 A[i].vertex)← i ;

A[i].link ← *nil* 】

(2)逐个画出图的每条边，即线段，每画一个相当于产生了一个邻接关系，对于边<u，v>来说，顶点u的邻接链表中应当增加一个顶点 v，即插入一个结点P





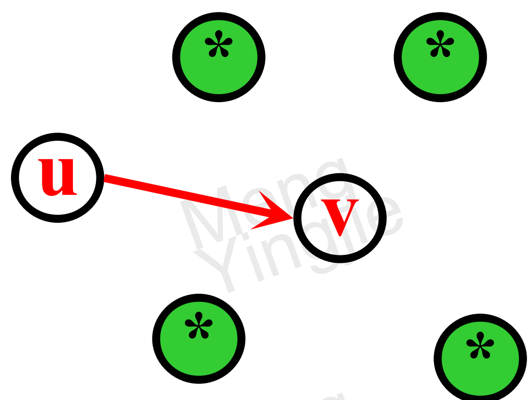
用动作来体现的话：

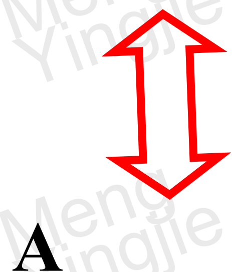
①输入u,v

②申请空间，存储v

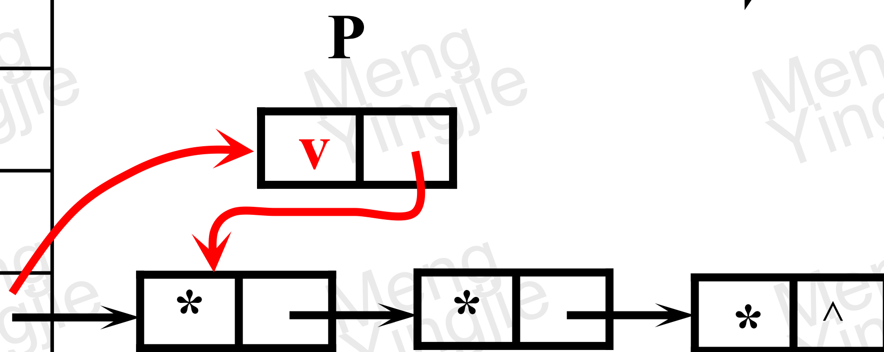
③完成插入

Meng Yingjie





A



用动作来体现的话：

①输入u,v

②申请空间，存储v

③完成插入

① read(u,v);

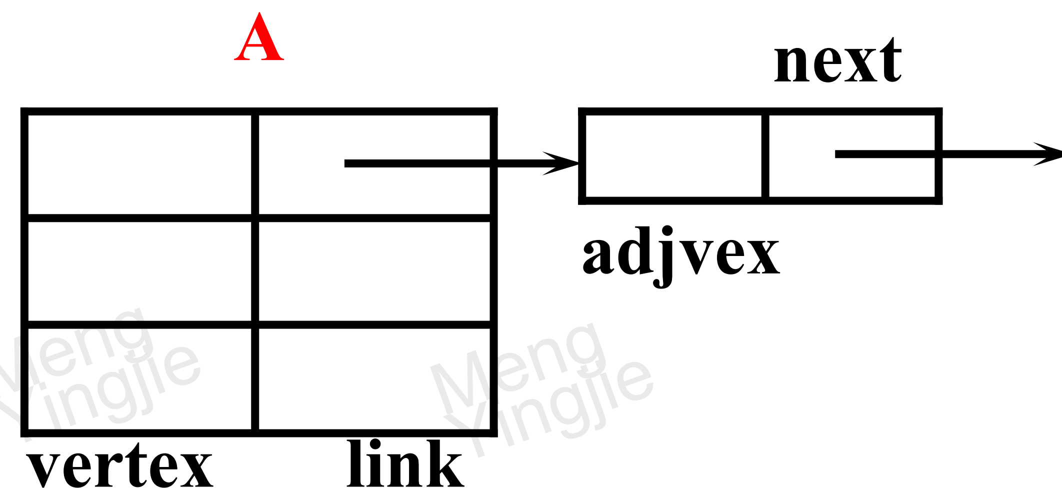
② new(P); p↑.adjvex←v ;

③ p↑.next←A[u].link;

A[u].link←p

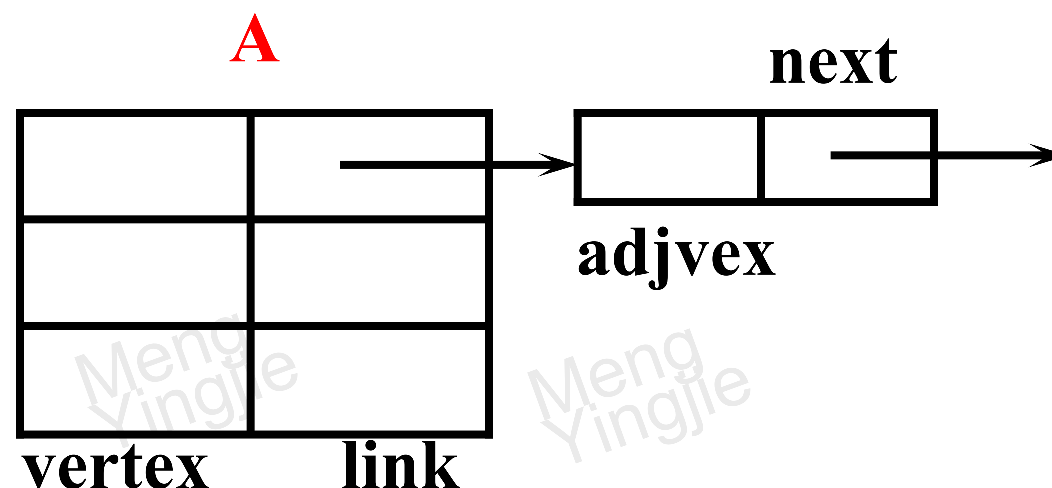
Meng Yingjie

具体算法过程：




```
PROC BuildAdjlist(VAR A: adj_list;e,n:integer);
BEGIN
   FOR i←1 TO n DO【A[i].vertex←i;  A[i].link←nil】 ;
   FOR i←1 TO e DO
       【read(u,v);  {读入有向边<u,v>}
        new(P);      p↑.adjvex←v;
        p↑.next←A[u].link;    A[u].link←p】
END;
```

建立初始邻接表



对无向图G,建立其邻接表A。

```
PROC  BuildAdjlist(VAR A: adj_list;e,n:integer);
BEGIN  FOR  i←1  TO  n  DO 【A[i].vertex←i;   A[i].link←nil】 ;
       FOR  i←1  TO  e  DO
          【read(u,v);   {读入无向边(u,v)}
           new(P);                  p↑.adjvex←v;     } U链中增加
           p↑.next←A[u].link;       A[u].link←p;     } 一个结点。
           new(P);                  p↑.adjvex←u;     } V链中增加
           p↑.next←A[v].link;       A[v].link←p 】   } 一个结点。
END;
```

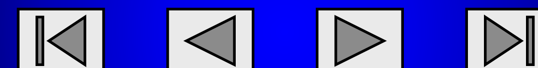
三. 邻接多重表

在图的有些问题处理中需要以边为单位进行，这时需要同时关注边的两个顶点。

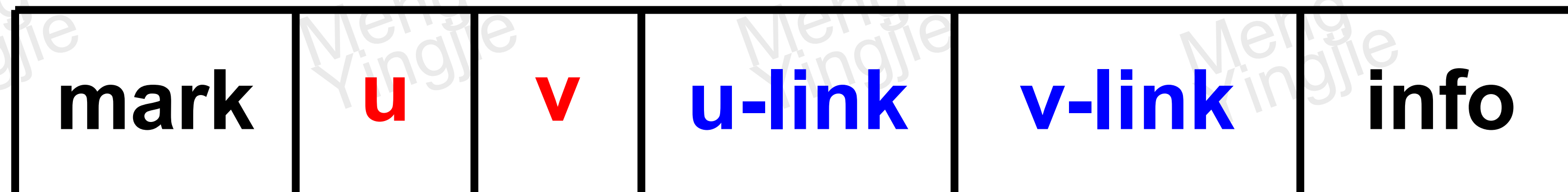
采用邻接矩阵往往无法反映顶点信息；

采用邻接表，对于无向图一条边关联的两个顶点分布在两条链表中；对于有向图，一条边分布在邻接表和逆邻接表中，所以给处理带来不便。

为解决该问题将多重表引入图的邻接表中，可将邻接表改造为邻接多重表(含有多个指针域)。



邻接多重表结点的设计(以边为单位):



mark: 标志域，标记该边是否被处理，也可存放权值等信息；

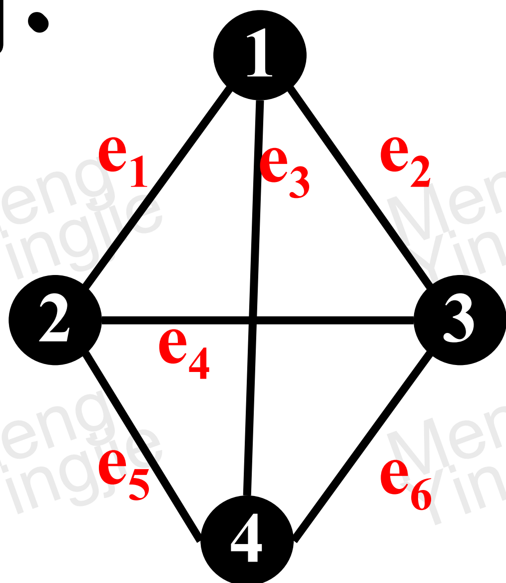
u,v: 关联边的两个顶点域；

u-link: 下一条依附于u的边的地址（指针）；

v-link: 下一条依附于v的边的地址（指针）；

info: 该边的其他相关信息。

举例:

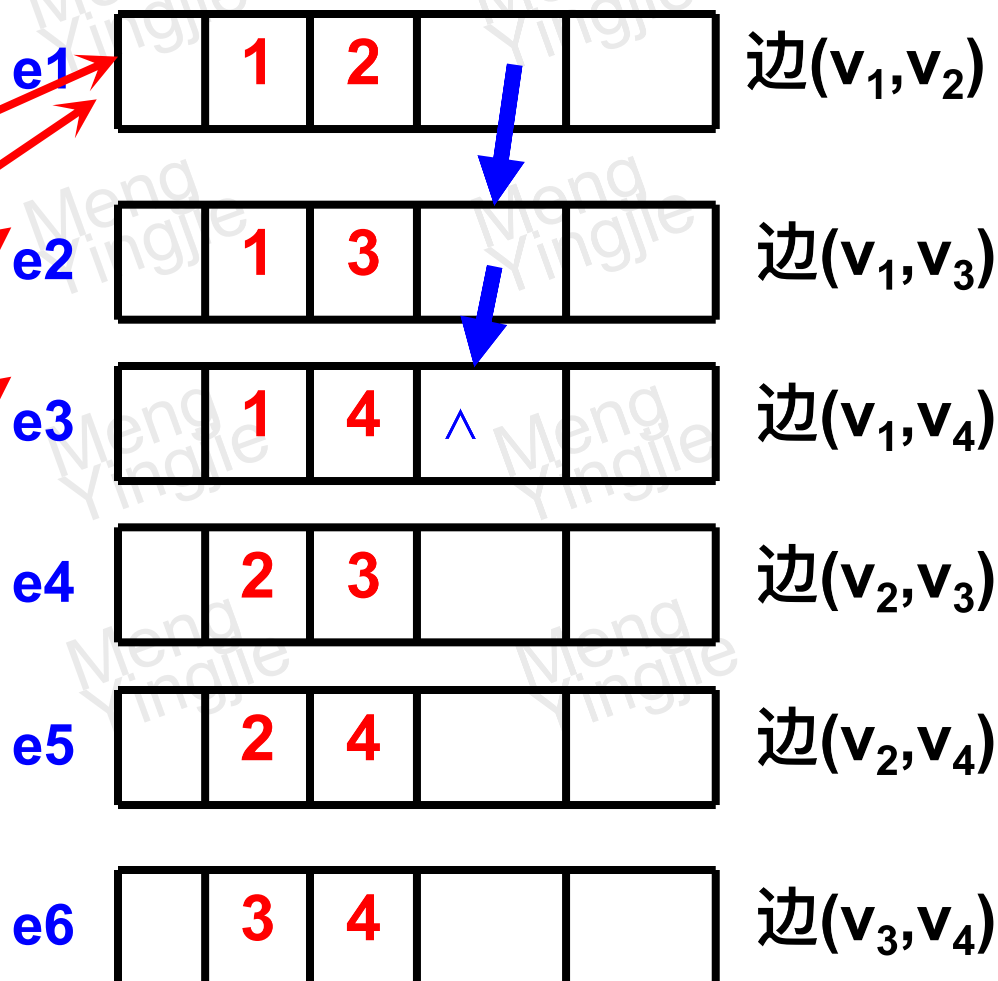


顶点 v_1: e1→e2→e3

v_2: e1→e4→e5

v_3: e2→e4→e6

v_4: e3→e5→e6



本节结束

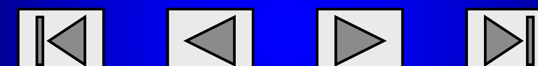
Meng Yingjie

一.引言

与树形结构的遍历类似,同样能够讨论遍历操作,只不过要比树形结构复杂的多,例如:

- 图中没有象树形结构中根那样的顶点;
- 图中顶点没有层次之分,结点间关系的随意性;
- 图中每一个结点都可以与其它结点邻接，故访问某结点后,可能沿着某条边又回到访问过的结点;

因此,为保证访问的顺利进行就需借助一些辅助结构。



定义:

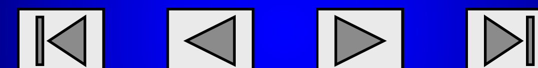
给出图G和其中的任意一个顶点v_0，从v_0出发系统地访问G中所有的顶点,且每个顶点仅被访问一次,这一过程称为图的遍历(graph traversal),或遍历图。

辅助结构:

设立辅助数组visited[1..n]

$$\text{visited}[v]=\begin{cases}1, & v\text{已经被访问}\\0, & v\text{未被访问}\end{cases}$$

Meng Yingjie



搜索策略:

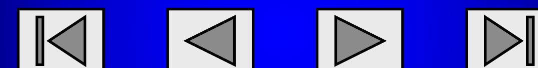
◆深度优先搜索(DFS,depth-first search)

搜索中，结点扩展的次序向某一个分支纵深推进，到底后再回溯。

◆广度优先搜索(BFS,breadth-first search)

搜索中，对所在层次的所有结点逐个依次进行扩展后，再推进到下一个层次进行扩展。

Meng Yingjie



二.图的深度优先搜索

基本思想:

与二叉树的遍历类似，是一个递归过程。

访问出发顶点v_0，

然后选择一个v_0邻接到的未被访问过的结点u，

再从u开始进行深度搜索。

如果从任一顶点出发再也无法抵达一个未曾访问过的顶点，则本次搜索就算结束。

回退策略：每当抵达一个其所有相邻顶点均已被访问过的顶点x时，就回退到最后访问过的那个顶点——该顶点有一个与其邻接的未曾访问过的顶点y，并从y再开始进行深度遍历。

Meng Yingjie

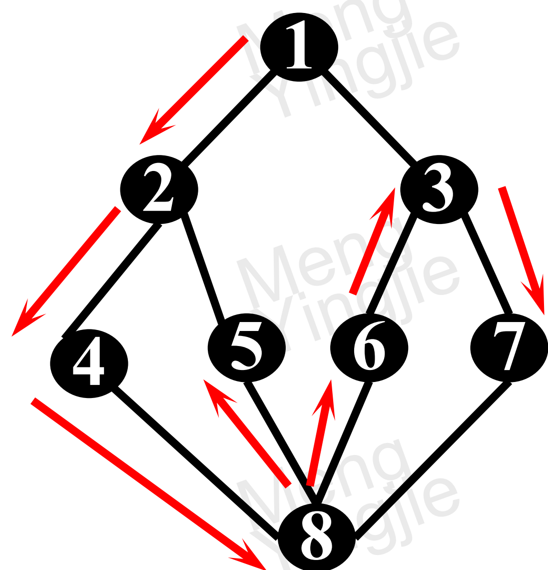
图的深度搜索过程：

```
PROC  Graph_DFS(G,v0);
BEGIN
   visited[v0]←1;  write(v0);
   REPEAT
      CALL  adjvex(v0,u);  {寻找的邻接顶点v0}
      IF visited[u]=0 THEN CALL  Graph_DFS(G,u)
   UNTIL (all adjacent vertexes of v0 have been visited)
END;
```

该过程没有涉及存储结构。

Meng Yingjie

例：



访问标志	顶点	邻接表
1	v_1	2 → 3 ^
1	v_2	1 → 4 → 5 ^
1	v_3	1 → 6 → 7 ^
1	v_4	2 → 8 ^
1	v_5	2 → 8 ^
1	v_6	3 → 8 ^
1	v_7	3 → 8 ^
1	v_8	4 → 5 → 6 → 7 ^

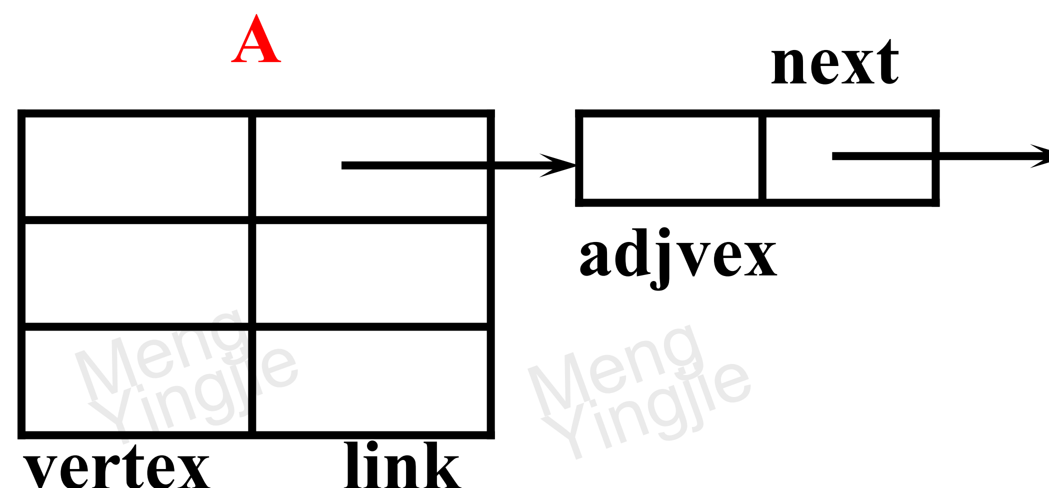
访问结果：

$v_1, v_2, v_4, v_8,$

v_5, v_6, v_3, v_7

遍历的结果依赖于存储状况——结果的不唯一性。

基于邻接表的深度优先搜索过程：




```
PROC Graph_DFS(VAR A: adj_list; v0:integer);
BEGIN
    visited[v0]←1;  write(A[v0].vertex);
    P←A[v0].link;
    WHILE P≠nil DO
      【IF visited[p↑.adjvex]=0 THEN Graph_DFS(A, p↑. adjvex);
        P←P↑.next 】 ;
END;
```

算法时间复杂度为O(e),若采用邻接矩阵则需要$O(n^2)$

图的深度遍历过程：

```
PROC  Graph_Traversal_DFS(VAR A: adj_list);
BEGIN
    FOR  i←1  TO  n  DO
        visited[i]←0;
    FOR  i←1  TO  n  DO
        IF  visited[i]=0  THEN  Graph_DFS(A, i)
END;
```

三.图的广度优先搜索

基本思想:

与二叉树的层次遍历类似，是一个逐层推进过程。

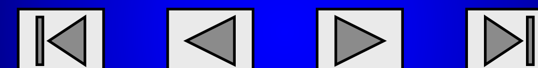
访问出发顶点v_0；

访问v_0邻接到的所有未被访问过的结点$v_1 , v_2 , \ldots , v_t$,

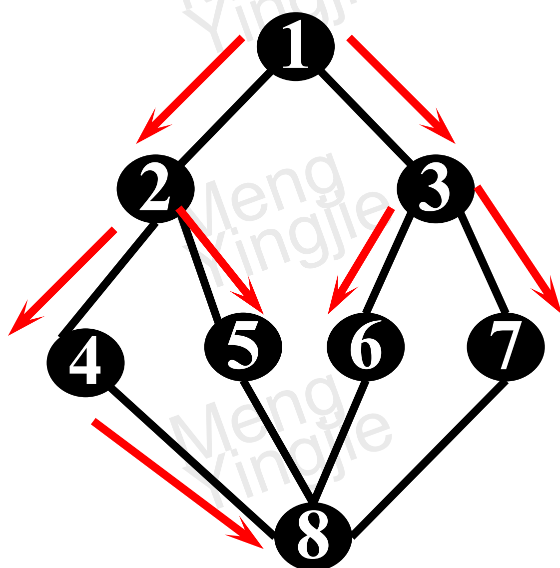
再依次访问$v_1, v_2, \ldots, v_t$ 邻接到的所有未被访问的结点,

如此进行下去，直到无法找到未被访问的结点时，则

本次搜索就算结束。



例2：



访问结果：

$v_1, v_2, v_3, v_4, v_5,$

v_6, v_7, v_8

visited	vertex	adjacency list
1	v_1	2 → 3 ^
1	v_2	1 → 4 → 5 ^
1	v_3	1 → 6 → 7 ^
1	v_4	2 → 8 ^
1	v_5	2 → 8 ^
1	v_6	3 → 8 ^
1	v_7	3 → 8 ^
1	v_8	4 → 5 → 6 → 7 ^

Meng Yingjie

广度搜索过程:

与树的层次遍历类似，须使用队列，可有两种控制策略:

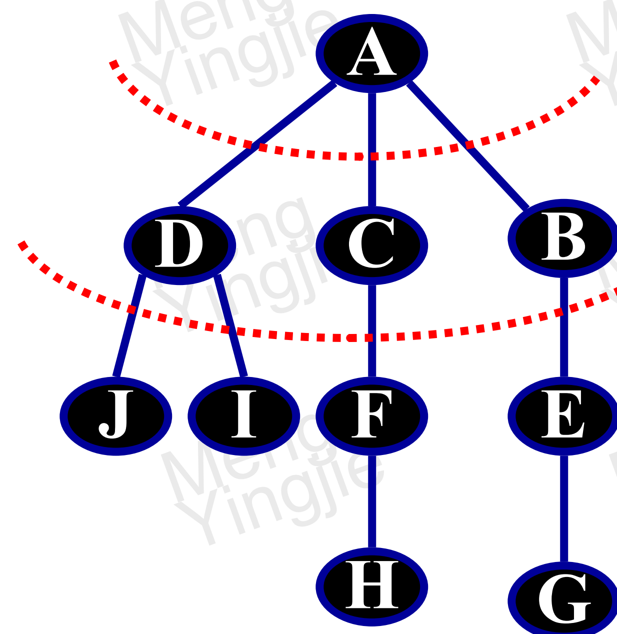
- 先访问，再入队;
- 先入队，再访问。

1.访问出发点；出发点进队;

2.队列非空时做如下工作:

①出队，元素为P;

②对P未访问的全部邻接点进行访问并令其进队;



CBJI…

出队元素:D

Meng Yingjie

图的广度搜索算法：

```
PROC  BFS(G,v0);
BEGIN Q←0;        {置队列Q为空}
      visited[v0]←1;  write(v0);
      CALL ADDQ(Q,v0);  {v0入队Q,采用先访问再进队方式}
      while (non-QueueEmpty) DO
          【CALL  DELQ(Q,u);  {u元素出队}
           REPEAT
               CALL  adjvex(u,w);  {寻找的邻接顶点w}
               IF visited[w]=0  THEN 【visited[w]←1; write(w);
                                      CALL ADDQ(Q,w)】
           UNTIL (all adjacent vertexes of u have been visited)】
END;
```

同样，该过程没有涉及存储结构。

本节结束

Meng Yingjie

一.连通分量和生成树

与树形结构类似,同样能够利用遍历来解决一些图的问题。

1.求图的连通分量(connected component)。

```
PROC  Graph_CC(G);
BEGIN  FOR  i←1  TO  n  DO  visited[i]←0;
       FOR  i←1  TO  n  DO
          IF  visited[i]=0  THEN  【CALL  Graph_DFS(A, p↑. adjvex);
                                   CALL OutPut {输出本次DFS中的所有顶点及边} 】
END;
```

每执行一次DFS(或BFS)都会得到一个连通分量。

例子：

Meng Yingjie

2.生成树(spanned tree):

在图G的遍历过程中，把访问的顶点和通过的边所构成的图我们称为G的生成子图。

根据图的遍历我们知道，对于以下情况：

- 连通图，出发顶点任意
- 强连通图，出发顶点任意
- 有根图，出发顶点为图根

可系统访问所有顶点，图的所有顶点，再加上遍历过程中经历的边所构成的子图实际上是一棵树，我们称做生成树。

对于不连通的无向图和不是强连通的有向图，从任一结点出发只能得到生成森林。

例1： 深度优先搜索——深度优先生成树。

搜索 →

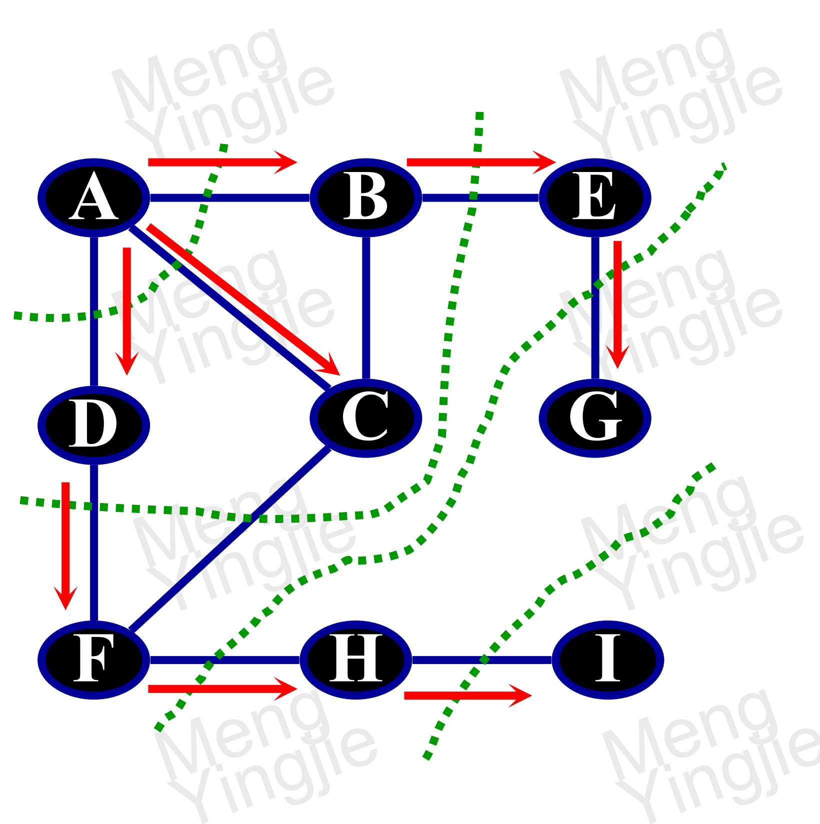
回退 ⋯⋯→

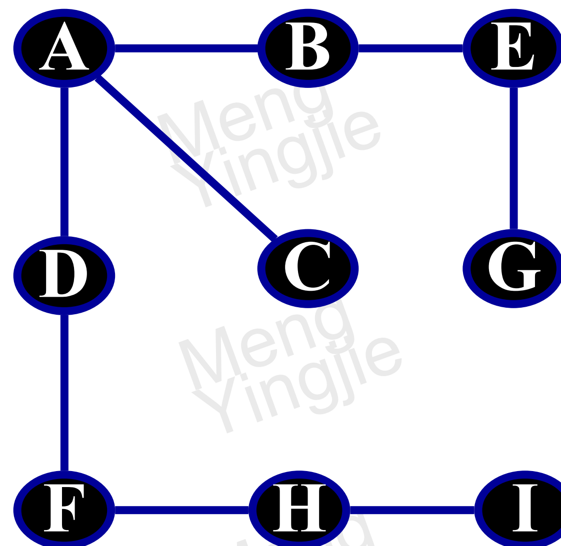
注意：访问的顶点及经历的边

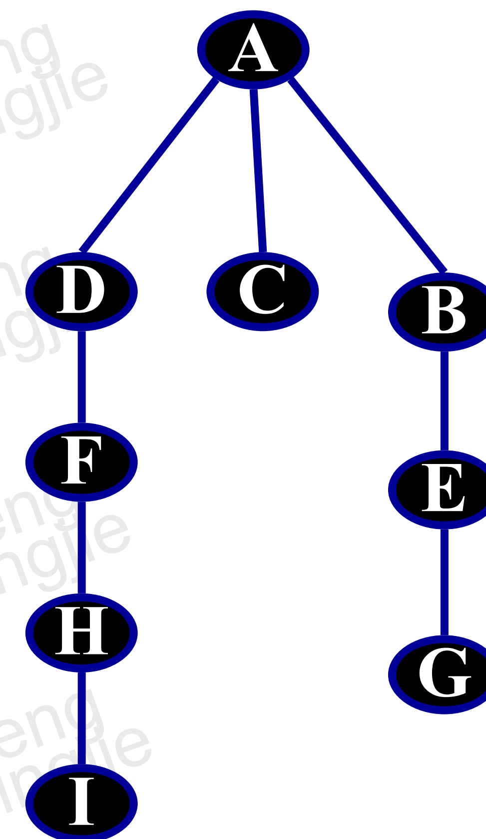
A, B, E, G, C, F, D, H, I

例2：广度优先搜索——广度优先生成树。

搜索
层次

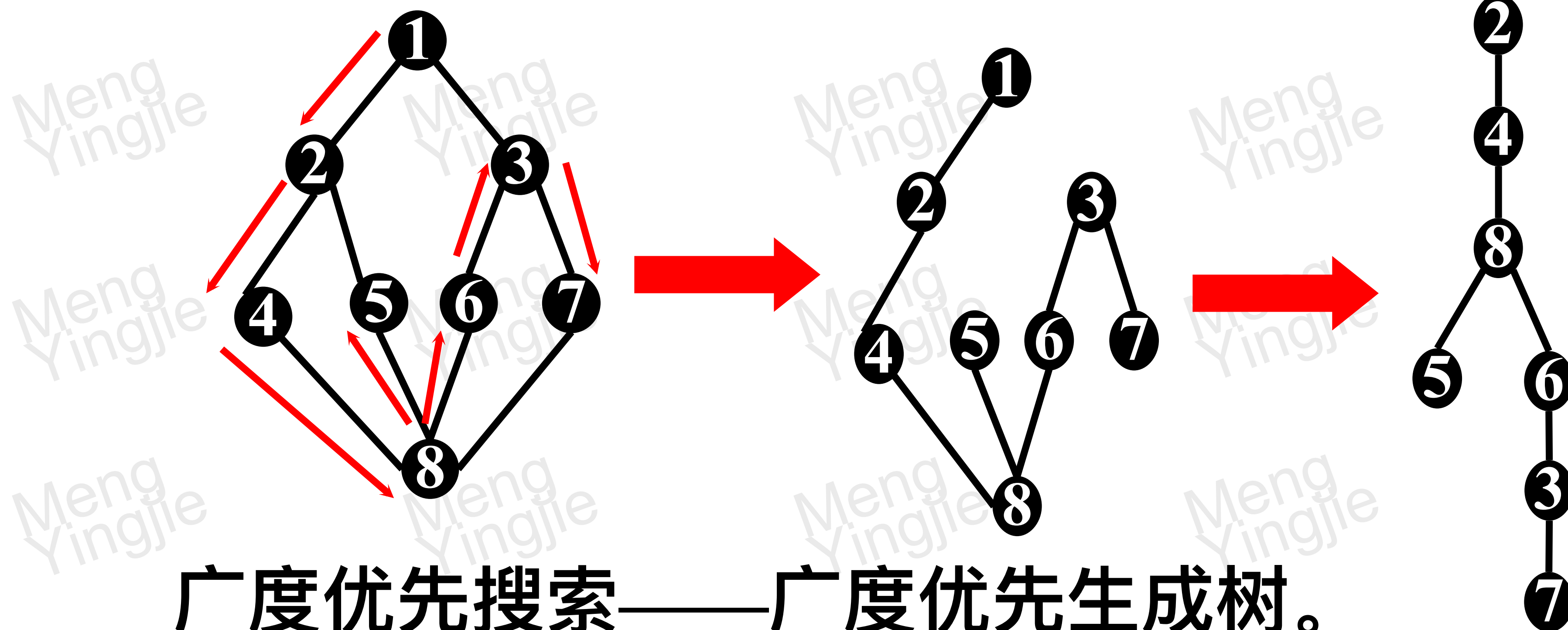




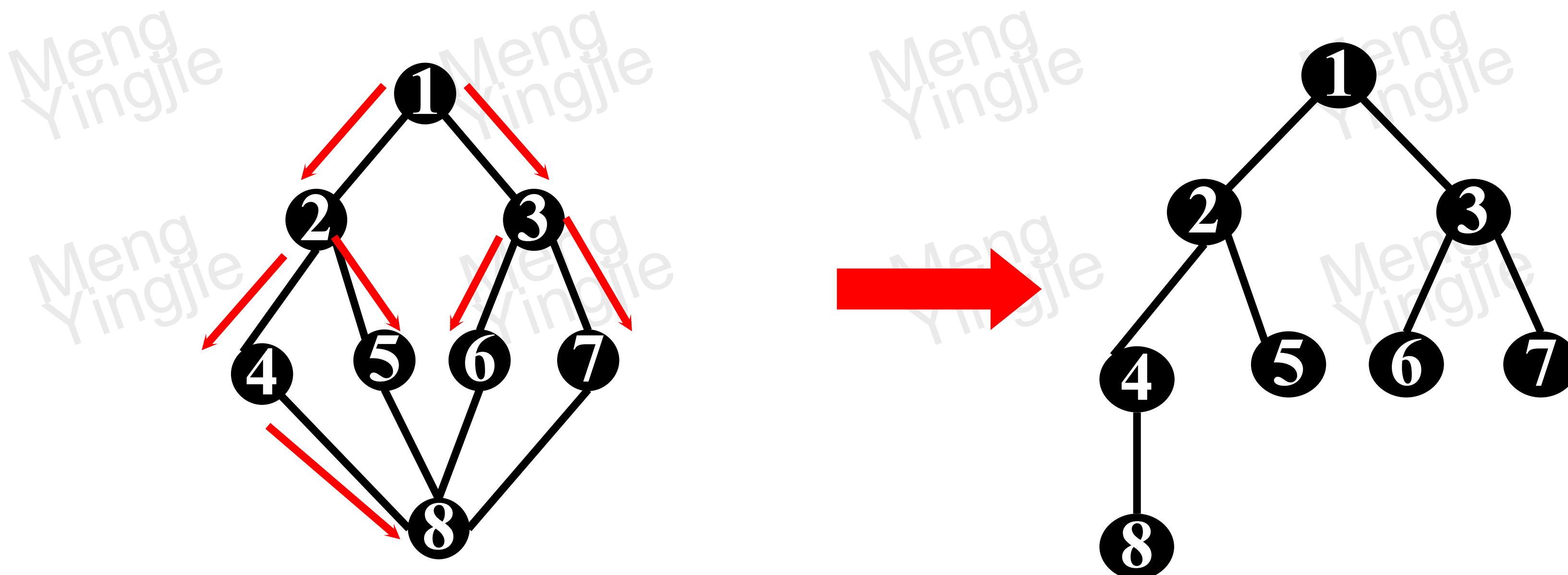


A B C D E F G H I

例3： 深度优先搜索——深度优先生成树。



广度优先搜索——广度优先生成树。



总结:

基于遍历方法显然我们可以构造极连通小图。

对于有n个顶点的连通图，其生成树具有n-1条边，因此是图的极小连通子图。

根据图的遍历我们知道出发顶点不同，得到的遍历顺序是不同的，故生成树是不唯一的。

二.最小生成树

我们把生成树的概念扩充到网络,将很有现实意义,例如6个城市通过公路连接,至少需要修5条路,如何保证总造价最小。

也就是说如何构造网络的最小生成树(边的权值之和最小,minimum-cost spanning tree)? 显然通过图的遍历很难做到,这里主要介绍Prim和Kruskal算法。

Meng Yingjie

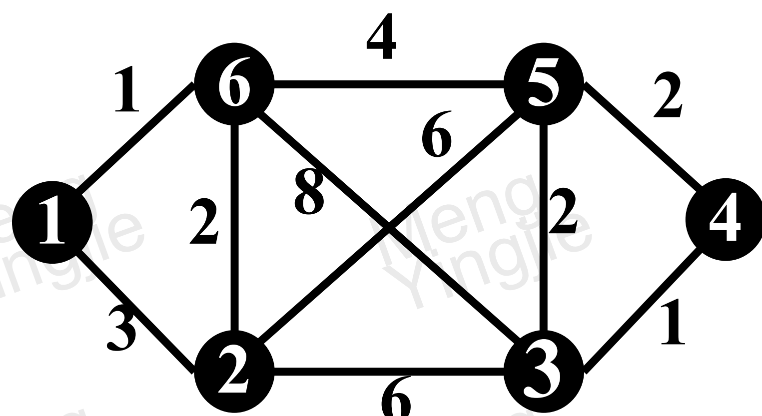
1.普里姆(Prim)算法

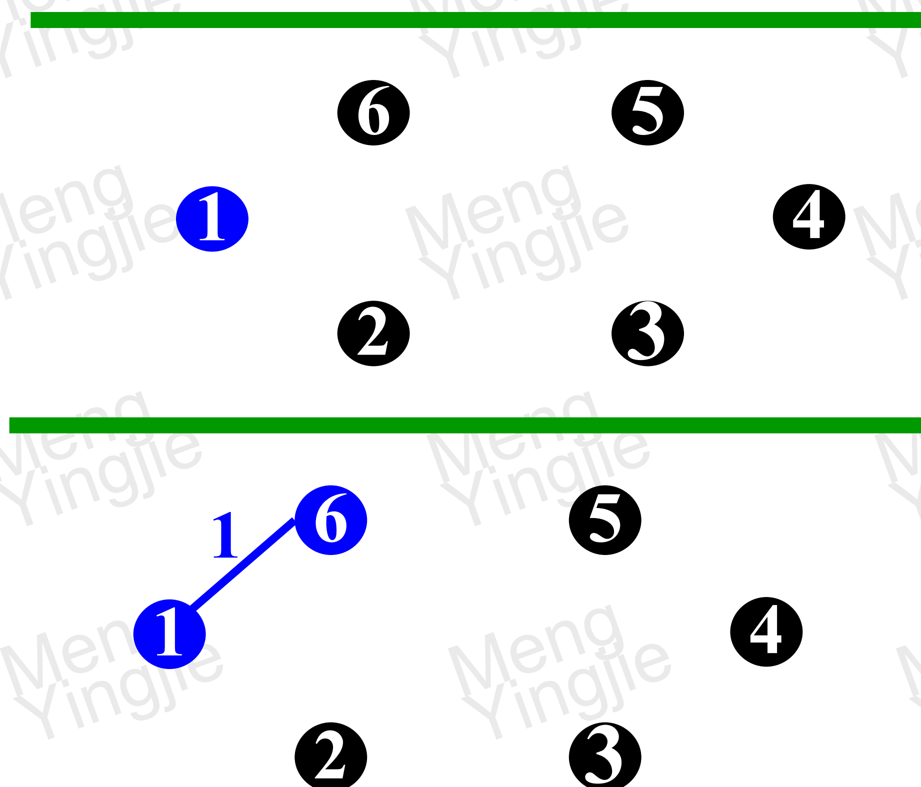
基本思想:

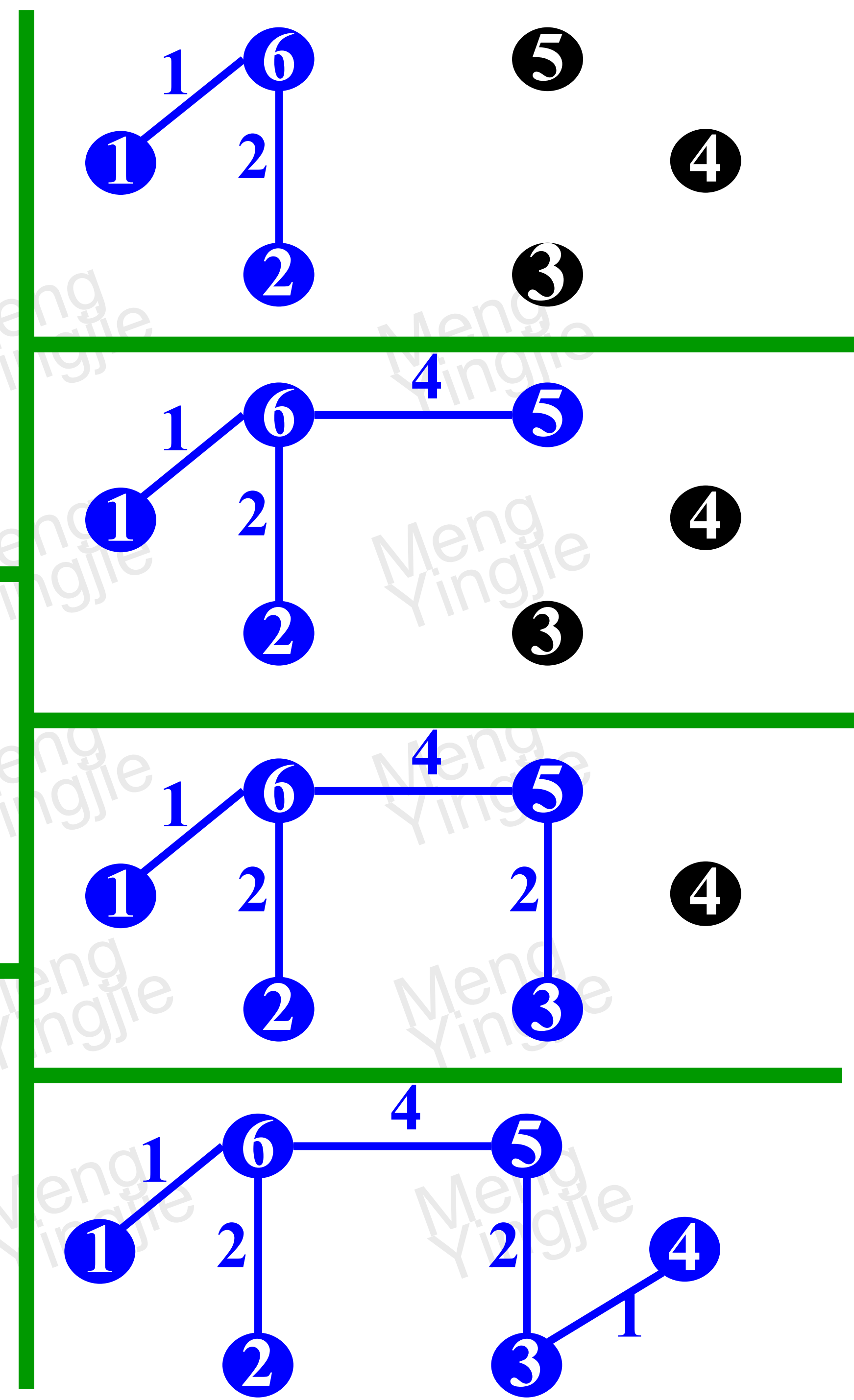
设V={1,2,…,n}是图G的顶点集合，PRIM算法由一个初值为{v_0}的集合U开始，它每次生成一条边，逐渐长成一棵具有最小代价的生成树。

算法在每一步中都找出一个最小权的边(u,v),其中$u \in$ U, $v \in$ V-U.

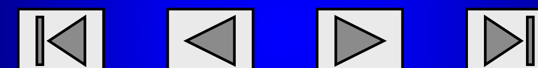
例子：







不唯一性。



PRIM算法处理过程：

```
PROC   PRIM(G,E,U);
BEGIN  E←[  ];  U←[1];
        WHILE U≠V  DO
           【{寻找最小权值的边e=(x,y),x∈U,y∈V-U}
             e←minside(x,y) ;
             E←E+e;
             U←U+[y] 】
END;
```

该过程没有涉及存储结构，还需做许多处理。多用于稠密网。

2.克鲁斯卡尔(Kruskal)算法

基本思想:

对于图G=(V,E)该算法初始化最小生成树T=(V,∅),此时生成树由n个孤立顶点组成n个连通分量——目标使得n个连通分量逐渐成为一个连通分量，具体做法：

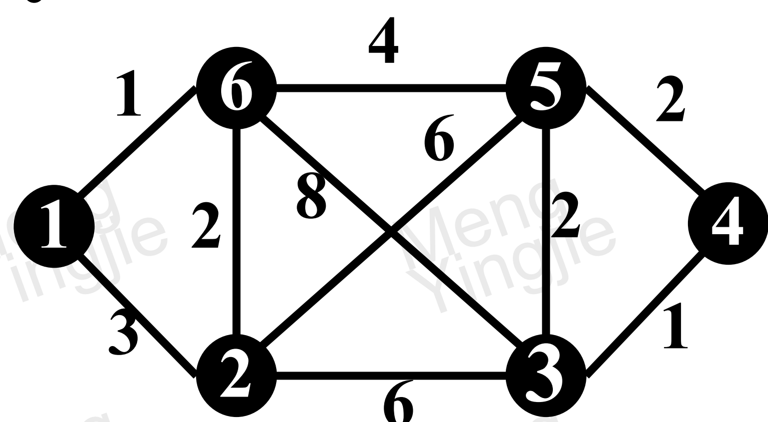
按照权值递增的顺序逐个考虑E中的每条边：

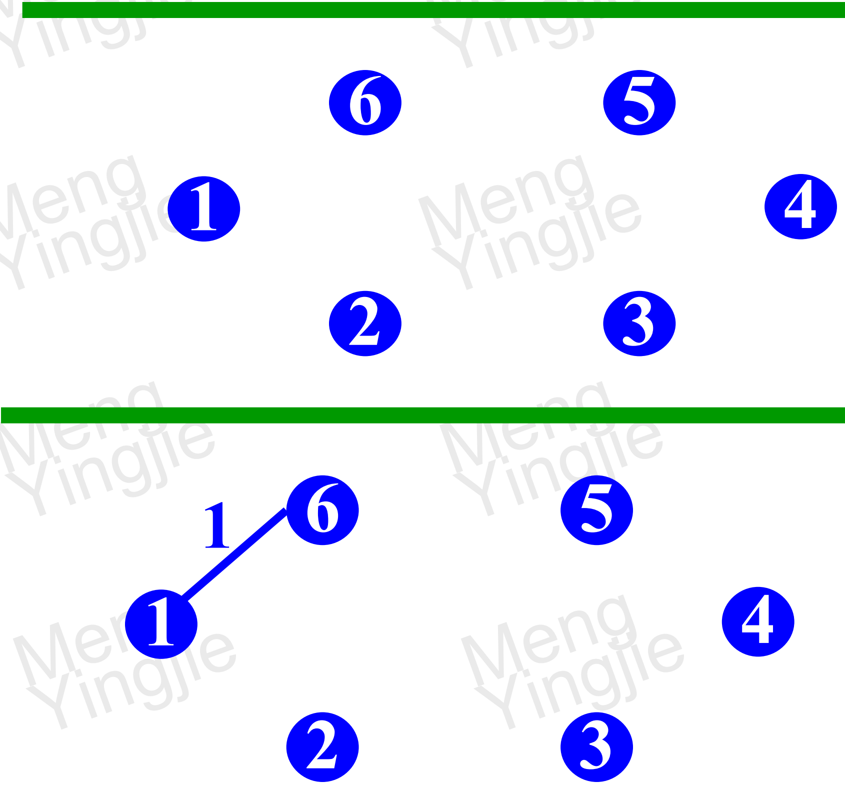
(1)若该边连通了在两个不同连通分量中的顶点，则将该边填加到T中.

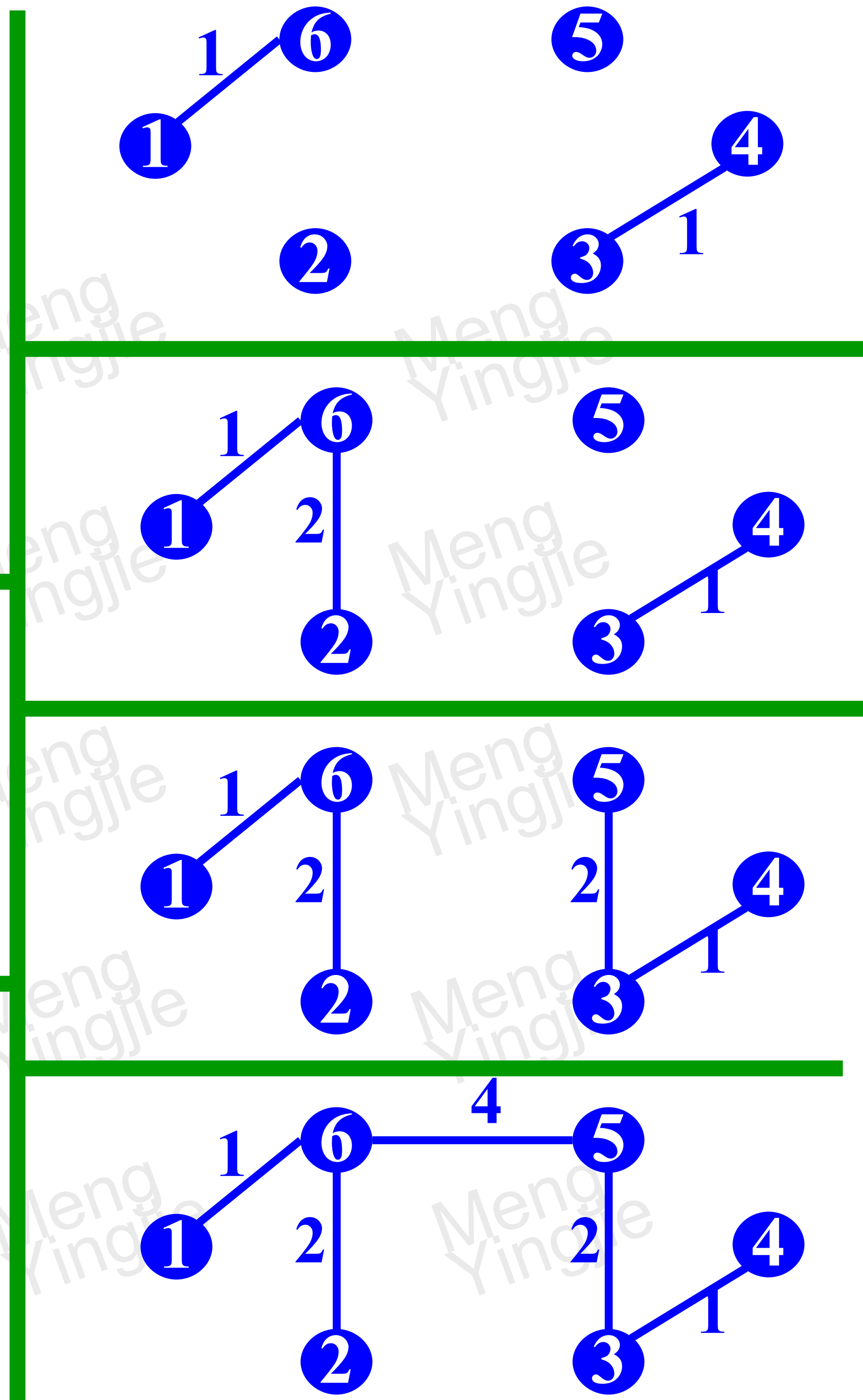
(2)重复(1)，一旦T中包含了n-1条边，则终止运算。

Meng Yingjie

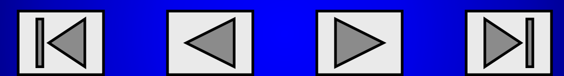
例子：







不唯一性。

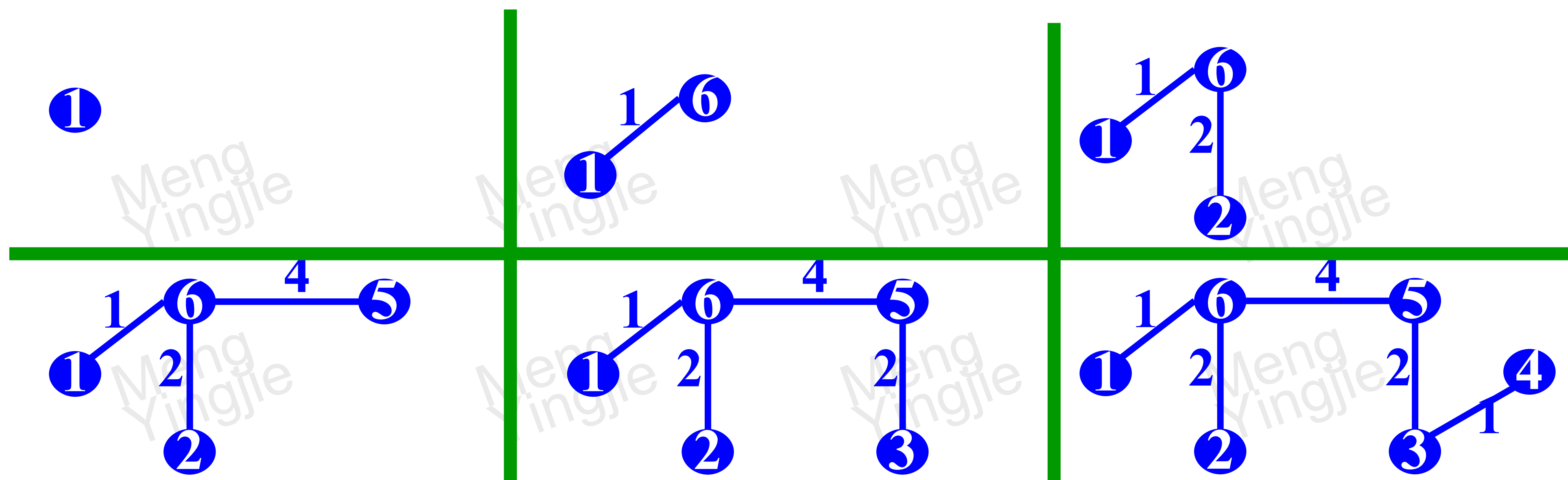


Kruskal算法处理过程：

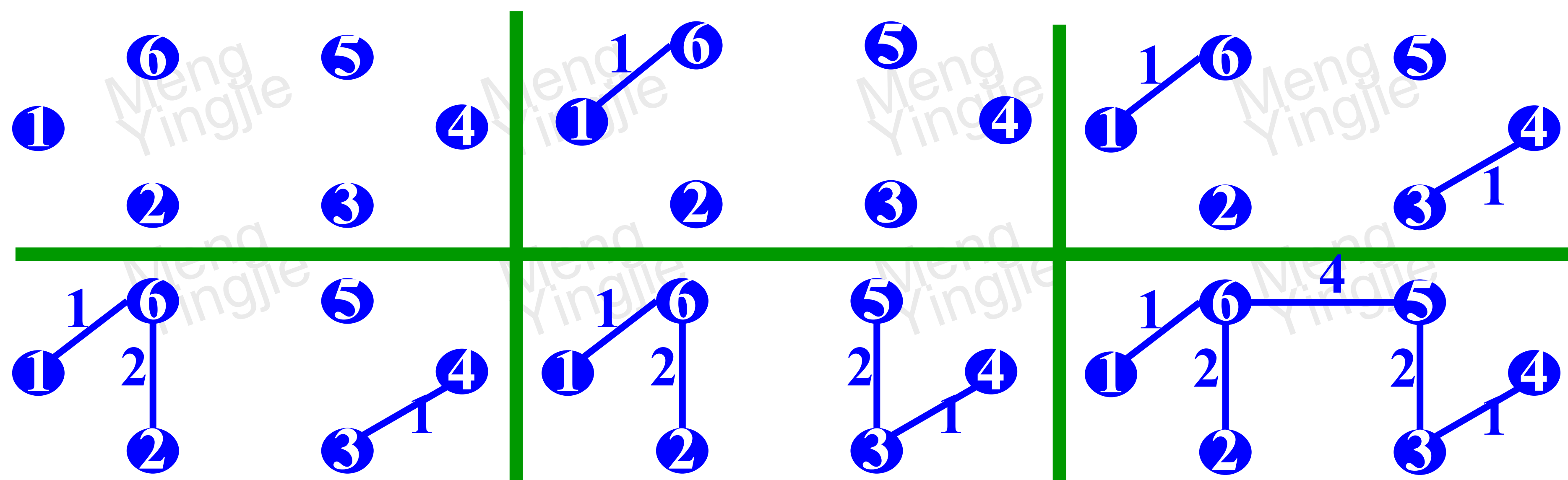
```
PROC   Kruskal(G,T,V,E);
BEGIN  T←(V, ∅);
       对E中的边按照权值递增的顺序排序；
       WHILE T含有少于n-1条边 DO
          【从E中取最小边(x,y);
            E← E-(x,y);
            IF (x,y)在T中不产生回路
            THEN T←T+(x,y) 】
END;
```

该过程没有涉及存储结构，还需作许多处理。多用于稀疏网情况。由于涉及到边的排序，考虑到排序代价故该方法多用于稀疏图。

Prim算法：



Kruskal算法：

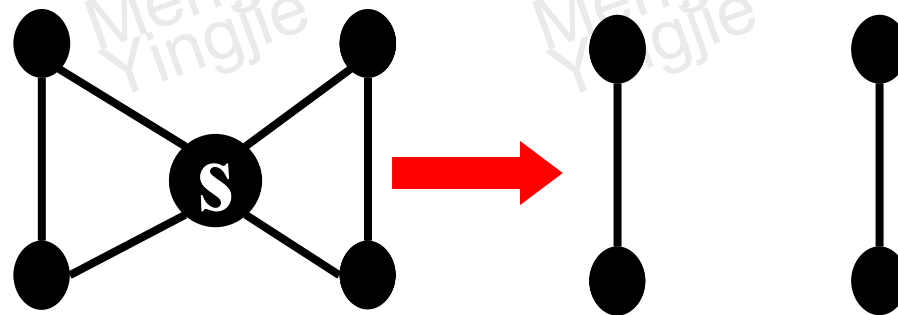


* 三.割点与双边连通图

可以利用遍历来寻找割点并判定图G是否为双边连通图。

割点集定义：设无向图G=(V,E)为连通图，若有结点集$V_1 \subset V$,使图删除了V_1的所有结点(含依附的边)后，所得到的子图是不连通图，而删除了V_1的任何真子集后，所得到的子图仍是连通图，则V1称作是G的一个割点集。

若某一个结点构成一个割点集,则该结点称为割点(或关节点,cut node).如下图中s为割点：



没有割点的连通图称为双边连通图，可以利用遍历来寻找割点并判定图G是否为双边连通图(或重连通图,或多边连通图，biconnected graph)。

双边连通图对于建立实用系统非常有用,其任意两个顶点之间有多于1条的路径(可认为是备用),因此系统的稳定性越高。在有割点的系统中,割点的失效将会导致整个系统的瘫痪。

推论：一个连同无向图G中的顶点v是割点的充分必要条件是存在两个顶点u,w，使得顶点u和w的每一条路径都通过v.

可以通过深度优先生成树来判定是否是否有割点。例如，生成树的树根的子树个数≥2则根必为割点；对于生成树中非叶结点v其某棵子树的根和子树的其它结点没有与v的祖先结点连接的边，则v为割点。

本节结束

Meng Yingjie

引言：

一个无环的有向图，我们称为有向无环图(DAG, directed acyclic graph)。

DAG图是一类较有向树更一般的有向图，在实际工程等问题中有广泛的应用。

这里主要讨论：

- 拓扑排序；
- 关键路径；

Meng Yingjie

一.拓扑排序(toplogical sorting)

是有向图的一种重要运算(实际是一种偏序运算)。

有向图可以用来表示工程的施工图，产品生产的流程图，学生的课程安排图等。

图中一个顶点可以对应于一个子工程、一门课程等，图中的边表示子工程完成的顺序（一种优先条件或者先决条件）。

下面以课程安排为例进行说明：

Meng Yingjie

例子：

Meng Yingjie

1、相关定义：

(1) AOV网：

若有向图G中,顶点表示活动或任务,有向边表示活动或任务之间的优先关系,则此有向图称为顶点表示活动的网络。

(**AOV-网**, Activity On Vertex Network)

Meng Yingjie

一个AOV-网要能够表达一个可执行(或运行)的工程则其优先关系应当是非自反的（因这里不研究自身环）。

显然AOV-网中由边规定的优先关系是具有传递性的,在AOV-网中不应当出现回路。

若存在有向回路，则说明某项活动的能否进行是要以自身任务的完成作为先决条件。显然这种工程是不能完成的，对一个程序来说相当于出现了死循环。

因此给定一个AOV-网,当要检测该网所表示的工程是否可以实现时,就可以通过检查其是否有回路来完成。

一种有效的方法就是拓扑排序(是否能找到偏序序列)。

(2)拓扑序列:

对于有向图G=(V,E),V中的顶点的线性序列(v_{i1}, v_{i2}, … , v_{in})，称作一个拓扑序列，若此结点序列满足如下条件：在G中从顶点u到顶点v有一条路径，则在序列中u必在v之前。

(注：拓扑序列不唯一)

任何无环的有向图，其顶点都可以排在一个拓扑序列中。

寻找拓扑序列的有效手段，就是进行拓扑排序。

Meng Yingjie

2.拓扑排序基本思想:

①从图中选择一个入度为零的顶点；

②输出该顶点，从图中删除此顶点及其所有的出边；

反复执行以上两步，直到所有顶点都输出,此时拓扑排序完成；

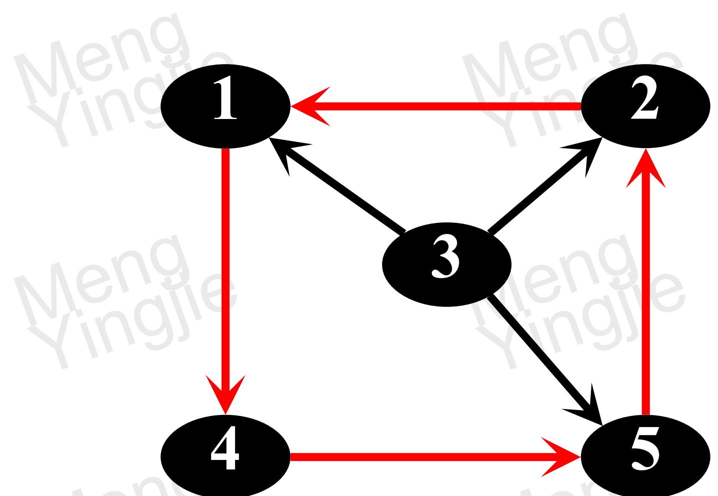
或者直到剩下的图中再无入度为零的顶点,此时说明图G是有环的图，拓扑排序无法完成。

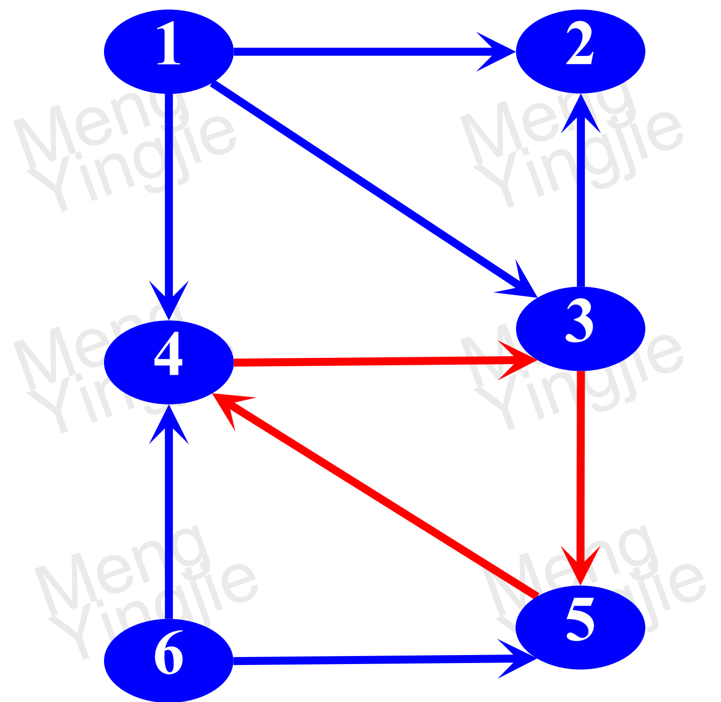
例1： $C_1 \rightarrow C_2 \rightarrow C_4 \rightarrow C_3 \rightarrow C_5 \rightarrow C_6$

不唯一性。

Meng Yingjie

例2：死锁示例





Meng Yingjie

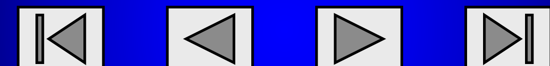
3.拓扑排序实现——基于邻接矩阵

基于基本思想中的关键两步，进行具体细化：

第一步：选择入度为零的结点u

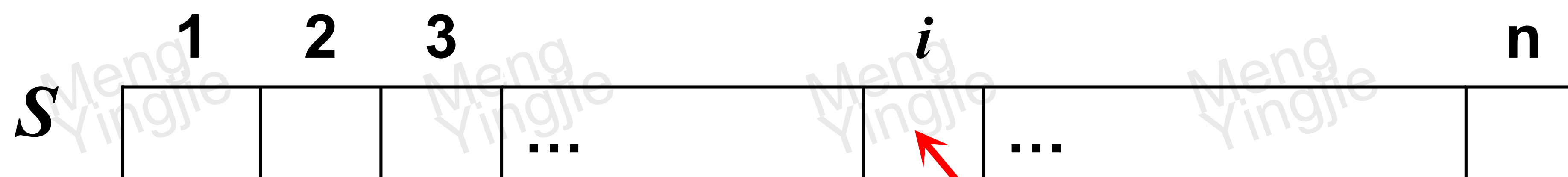
——选择全零的列u。

Meng Yingjie



第二步：输出顶点，删除出边——两个运算再细化：

（1）顶点的输出：假设拓扑序列需要保存，如何保存?



V_i 的输出顺序号

策略：拓扑序列可通过辅助辅助数组S[1..n]保存，具体取值：

$$S[i]=\begin{cases} 0, & \text{该节点未获输出} \\ m, & m\text{为第}v_i\text{个结点在拓扑序列中的输出次序} \end{cases}$$

Meng Yingjie

第二步：输出顶点，删除出边——两个运算再细化：

（2）删除出边，删除u所有出边

——邻接矩阵中第u行置零。

Meng Yingjie

算法步骤：

(1).S置零，输出次序计数器置初值(即：设定顶点的输出的顺序编号的起步值)

(2).寻找没有输出号的全零的列u，如果没有，则算法终止。此时S中所有元素都有输出号，则拓扑排序完成；否则有环。

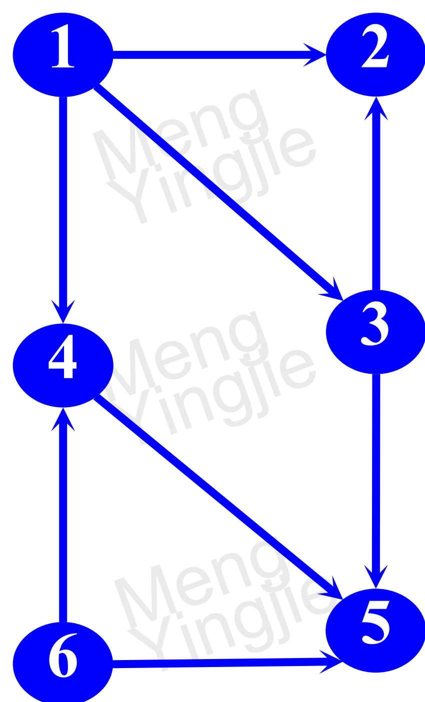
(3).将输出次序号赋给S[u]。

(4).把第u行置成全零。

(5).输出次序号加1，回到(2)。

算法复杂度O(n^3)

例：



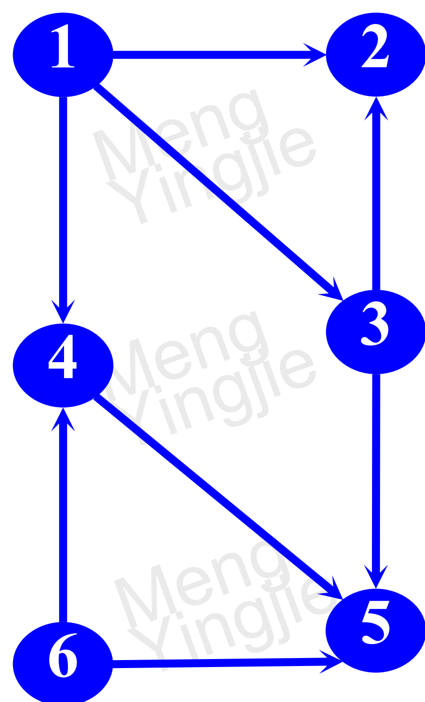
S[1..6]						
列 行	1	2	3	4	5	6
1	0	1	1	1	0	0
2	0	0	0	0	0	0
3	0	1	0	0	1	0
4	0	0	0	0	1	0
5	0	0	0	0	0	0
6	0	0	0	1	1	0

初始输出顺序号m=1

选择的全零列→1

Meng Yingjie

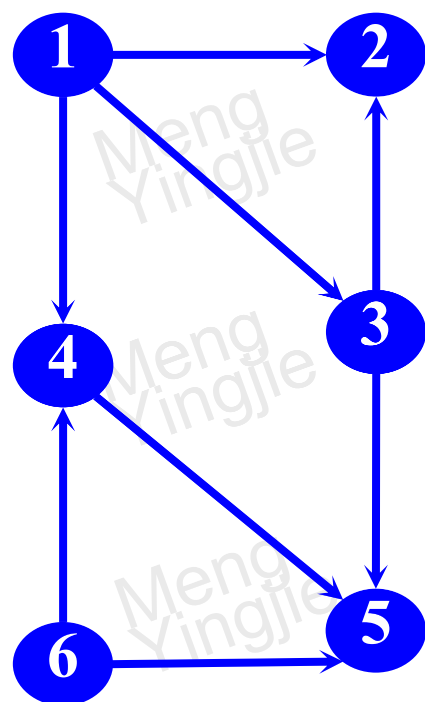
例：



S[1..6]	1					
行 \ 列	1	2	3	4	5	6
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	1	0	0	1	0
4	0	0	0	0	1	0
5	0	0	0	0	0	0
6	0	0	0	1	1	0

下次输出顺序号m=2　　　选择的全零列→3

例：

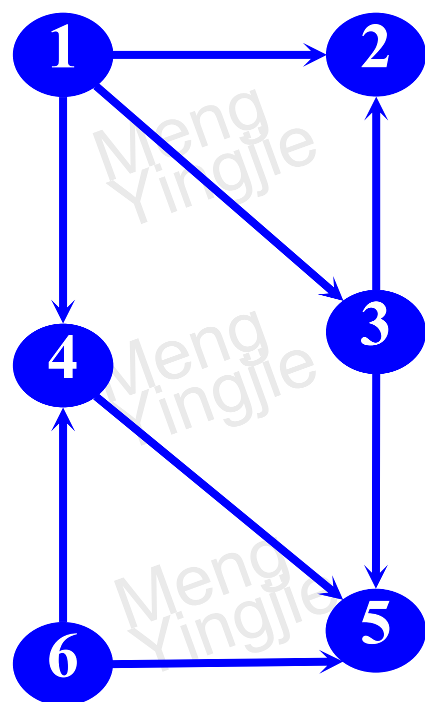


S[1..6]	1		2			
行＼列	1	2	3	4	5	6
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	1	0
5	0	0	0	0	0	0
6	0	0	0	1	1	0

下次输出顺序号m=3　　选择的全零列→2

Meng Yingjie

例：

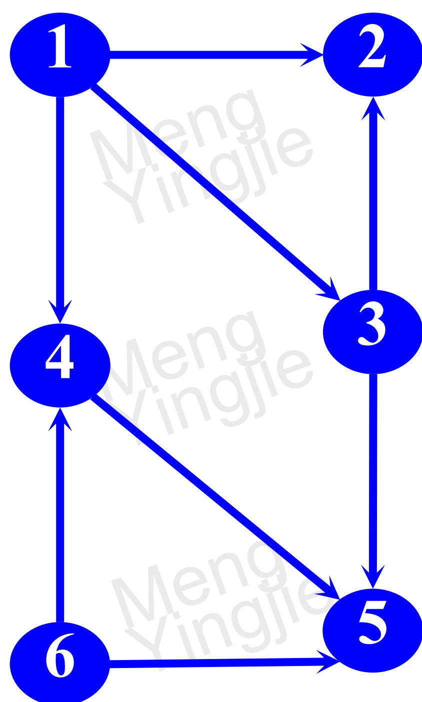


S[1..6]	1	3	2			
行 \ 列	1	2	3	4	5	6
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	1	0
5	0	0	0	0	0	0
6	0	0	0	1	1	0

下次输出顺序号m=4　　选择的全零列→6

Meng Yingjie

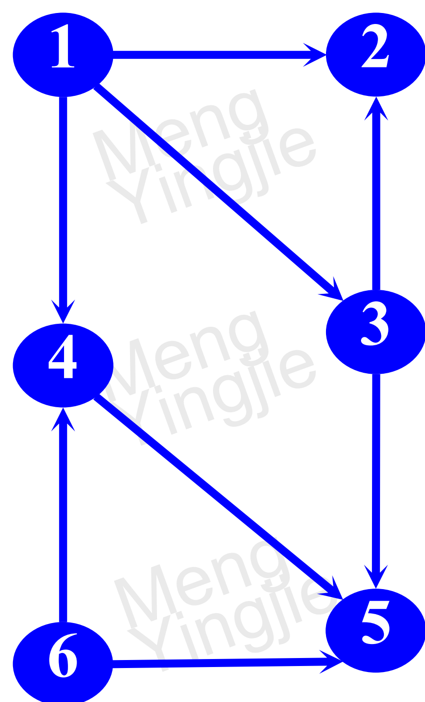
例：



S[1..6]	1	3	2			4
行 \ 列	1	2	3	4	5	6
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	1	0
5	0	0	0	0	0	0
6	0	0	0	0	0	0

下次输出顺序号m=5　　选择的全零列→4

例：

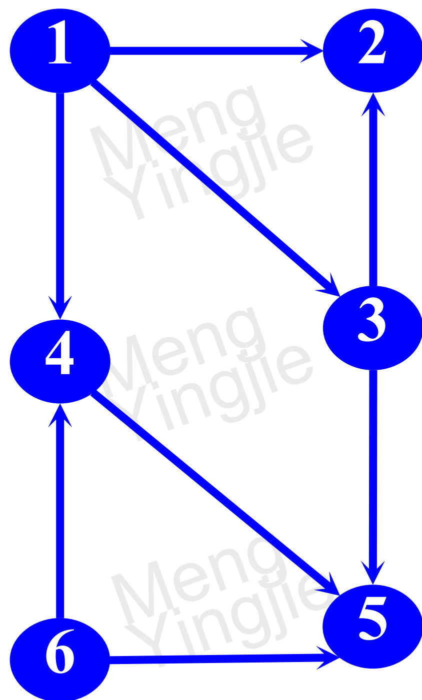


S[1..6]	1	3	2	5		4
行 \ 列	1	2	3	4	5	6
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0
6	0	0	0	0	0	0

下次输出顺序号m=6　　　　选择的全零列→5

Meng Yingjie

例:

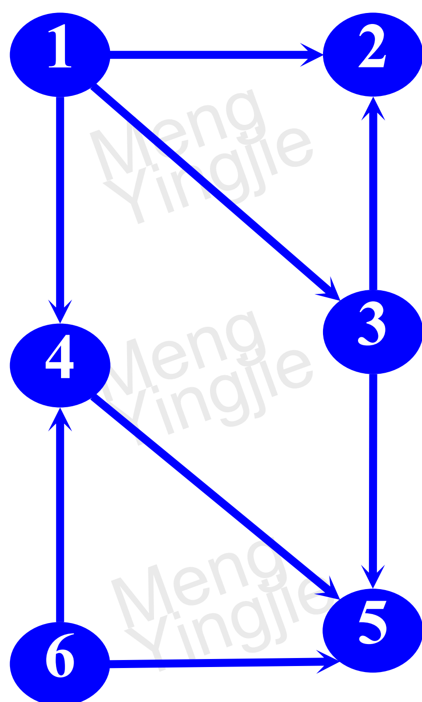


S[1..6]	1	3	2	5	6	4
行 \ 列	1	2	3	4	5	6
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0
6	0	0	0	0	0	0

下次输出顺序号m=7 选择的全零列→5

Meng Yingjie

例：



S[1..6]	1	3	2	5	6	4
列 / 行	1	2	3	4	5	6
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0
6	0	0	0	0	0	0

所有的列都已得到顺序(编)号，拓扑排序完成。

4.拓扑排序实现——基于邻接表

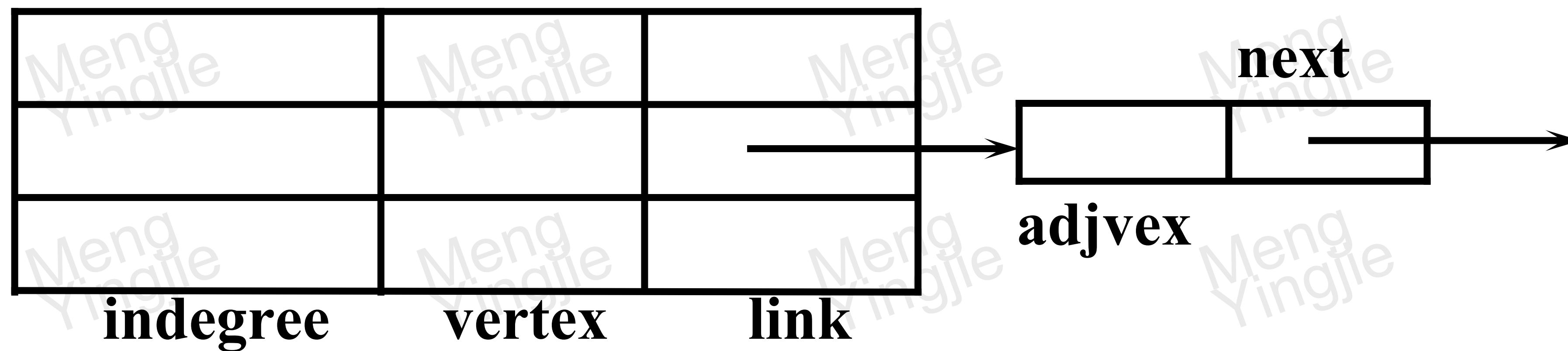
①选择入度为零的结点

- 需要事先计算各顶点的入度；
- 计算出的入度需要地方保存；
- 当出现多个入度为零的顶点时，因每次只能选择1个，所以为防止重选、漏选就需要一个结构(S)保存其它入度为零的顶点，以备选用——S可以采用栈。

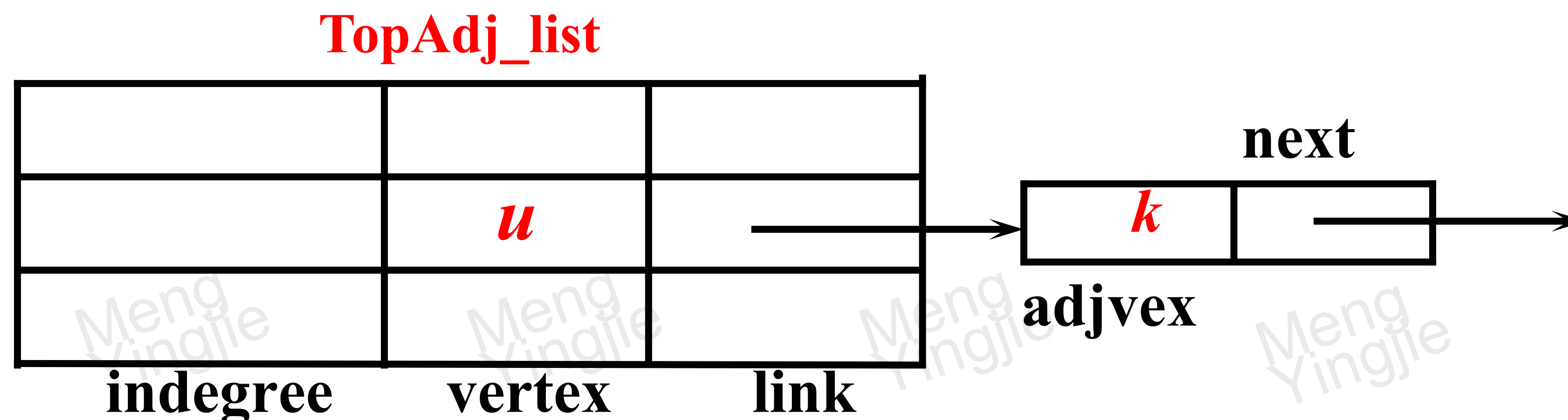
②删除入度为零的顶点的出边。

- 扫描删除顶点对应的链表,扫描到的顶点的入度减1；
- 对于入度减至零的顶点，将其保存到备选结构S中。

邻接表结构设计：



在前面定义过的邻接表中可以增加入度域，其值可以在输入边建立邻接表过程的同时产生。



算法步骤：

(1).搜索邻接表中入度为零的顶点，并令其进栈。

(2).当栈非空时，进行拓扑排序：

①退栈，输出栈顶元素u;

②(循环)在邻接表中扫描u的直接后继顶点k,

- 将k的入度减1,
- 若此时k的入度为零,则令k入栈。

(3).若栈空，输出元素不足n则有环，否则拓扑排序完成。

Meng Yingjie

栈结构的优化设计：

为了保存待输出的入度为零的结点(可保存在栈S中)，但这需要额外的空间。

邻接表中的入度域是要参与运算的(减法)，因此在拓扑排序后，该域将失去实际意义；

另外入度为0的顶点的入度域是不会参与运算的,因此,S可借用这些单元作为栈的存储空间；

	indegree		
1			→
2	6	b	→
3			
4			→
5	2	a	
6	0	c	

这样可以将入度为零的顶点的入度域组织成一个**静态的链栈**。此时indegree域相当于next域的作用。

例如，如图若栈顶指针top=5,则栈中的元素依次为a,b,c.

Meng Yingjie

算法过程：


```
PROC  TopSort(VAR A: Topadj_list);
BEGIN   top←0;   {栈初始化}   m←0; {输出顶点计数器}
  FOR  i←1  TO  n  DO
        IF A[i].indegree=0  THEN 【A[i].indegree←top;  top←i】 ;
  while  top≠0  DO
     【u=top;  top←A[top].indegree ; {退栈}
      write(A[u].vertex);  m←m+1;   p←A[u].link;
      while  p≠nil   DO  {删除u的所有出边<u,k>}
        【k←p↑.adjvex;      A[k].indegree←A[k].indegree-1;
          IF A[k].indegree=0 THEN 【A[k].indegree←top;  top←k】 ;
         p←p↑.next】 】 ;
  IF  m<n   THEN   write('The network has a cycle ')
END;
```

```
PROC   TopSort(VAR A: Topadj_list);
BEGIN    top←0;   m←0;
  FOR i←1 TO n DO IF A[i].indegree=0 THEN 【A[i].indegree←top;  top←i】;
  while  top≠0  DO
     【u=top;  top←A[top].indegree ;
       write(A[u].vertex);  m←m+1;   p←A[u].link;
       while  p≠nil  DO
         【k←p↑.adjvex;      A[k].indegree←A[k].indegree-1;
          IF A[k].indegree=0 THEN 【A[k].indegree←top;  top←k】  ;
          p←p↑.next】  】 ;
   IF  m<n  THEN   write('The network has a cycle ')
END;
```

算法简单分析：
- 正常情况下，n个顶点入栈n次出n次；
- 入度减1进行e次；
- 所以总时间代价O(n+e) .

二.关键路径

1、AOE-网及关注的问题:

与AOV-网对应的是AOE网。AOE-网在企业管理、工程计划等当中有广泛的应用。

若在带权有向图中顶点表示事件，有向边表示活动，权表示活动持续的时间，则此有向图称为边表示活动的网络(AOE-网，Activity On Edge Network).

表示实际工程(或计划)的AOE-网，应该是无环的，且存在唯一入度为零的起始顶点(始点)，以及唯一的出度为零的完成顶点(终点)。

Meng Yingjie

关注点：

利用事件AOE-网可以进行工程安排估算，研究完成整个工程至少需要多少时间；为缩短整个工程的完成时间，应该加快哪些活动的速度等问题。

在AOE-网要解决这些问题，可以采用多种技术：

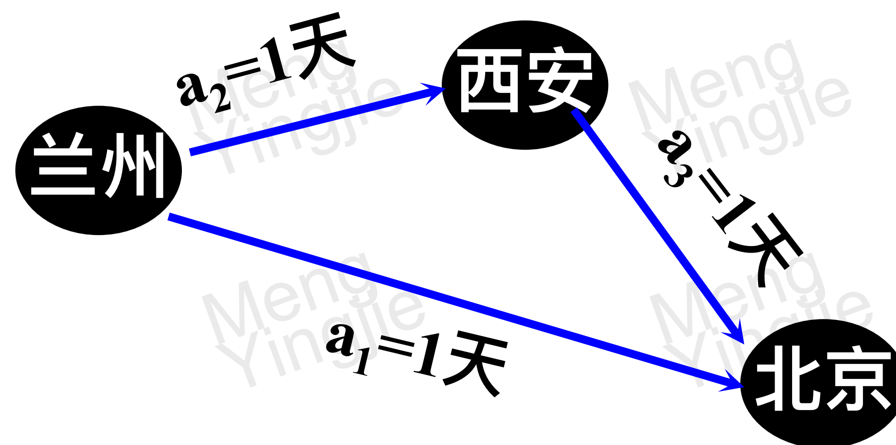
- PERT(程序评价和审定技术);
  (program evaluation and review technique)
- CPM(关键路径法);
  (Critical Path Method)
- RAMPS(资源分配和多项目调度)
  (resources allocation and multi-project scheduling)

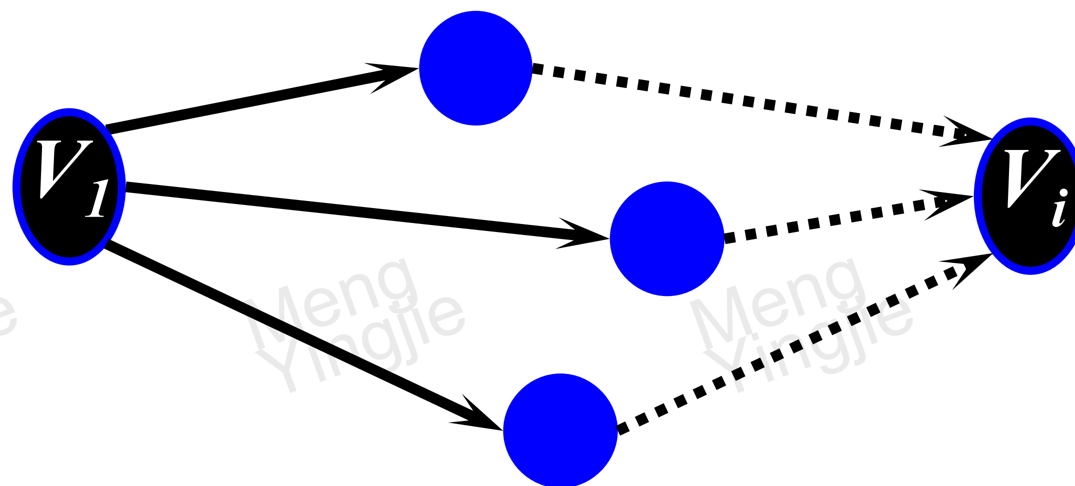
Meng Yingjie

2、定义及计算原理:

关键路径:任务计划作业图上的**需要时间最长的路径(可有多条)**。它决定完成总任务的时间。

工程的最小时间:从开始顶点到结束顶点的最长路径的长(该路径上各活动的时间总和)。



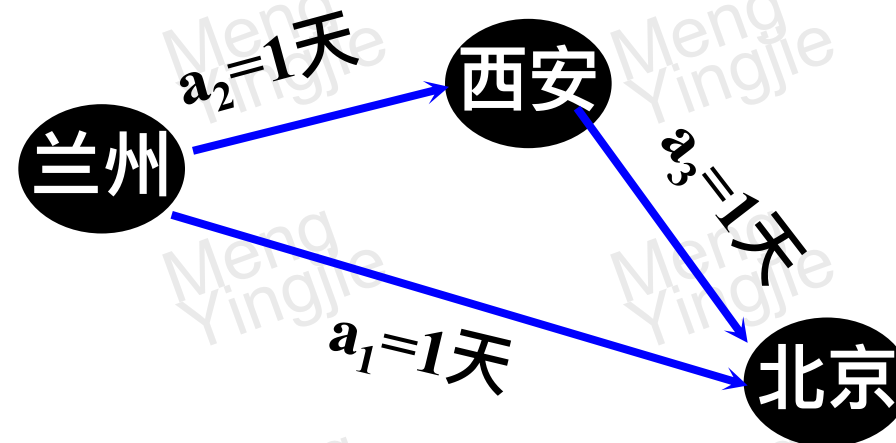


事件v_i能够发生的最早时间:从始点v_1到v_i的最长路径的长，它决定了表示该事件的这个顶点发出去的所有边表示的活动的最早开始时间；

用e(i)表示活动 a_i的最早开始时间；

在不推迟整个工程完成的前提下，一个活动可以最晚开始的时间定义为该活动的最迟开始时间，用l(i)表示.

用e(i)- *l*(i)表示活动 a_i 的**时间余量**，即 a_i 在不影响总工期的前提下，可以延缓的时间。

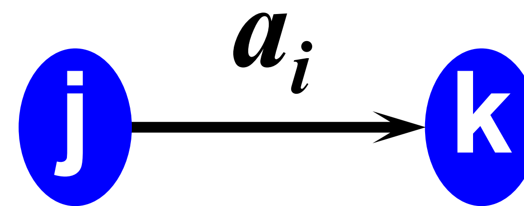


e(i)=*l*(i)的表示活动 a_i 称**关键活动**，显然关键路径上的所有活动都是关键活动，要加快工期只有提速关键活动的速度。

计算出e(i)与l(i)后就可得全部关键活动,删除非关键活动后，再由开始点到终止点之间的全部路径都是关键路径。

在实际应用中，关键活动时间调整后,可能导致关键路径的改变,因此,在完成工程的各阶段,必须要随时进行关键路径的计算,以便使各活动时间估计量更合乎实际。

e(i)与l(i)的计算：



设活动 a_i 用弧<j,k>表示，dut(<j,k>)表示权值。

e(i)：活动 a_i 的最早开始时间；

l(i)：活动 a_i 的最迟开始时间；

Ve(j)：事件 j 的最早发生时间；

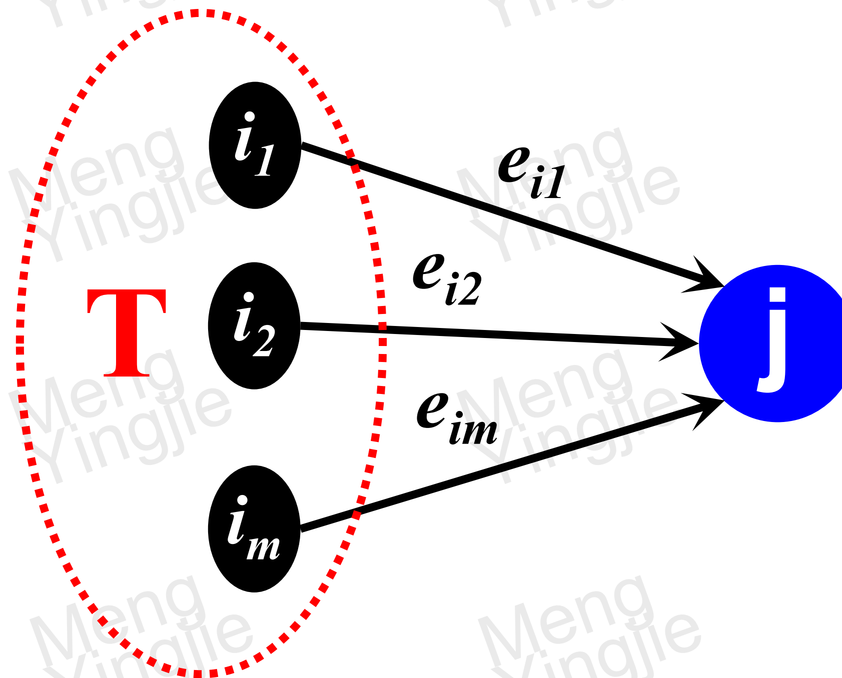
Vl(j)：事件 j 的最迟发生时间；

则：e(i)=Ve(j) →演变为计算Ve

l(i)=Vl(k)-dut(<j , k>) →演变为计算求Vl

(1).从Ve(1)=0开始向前递推求Ve(j)

可以用拓扑排序方法按照递推顺序求出。

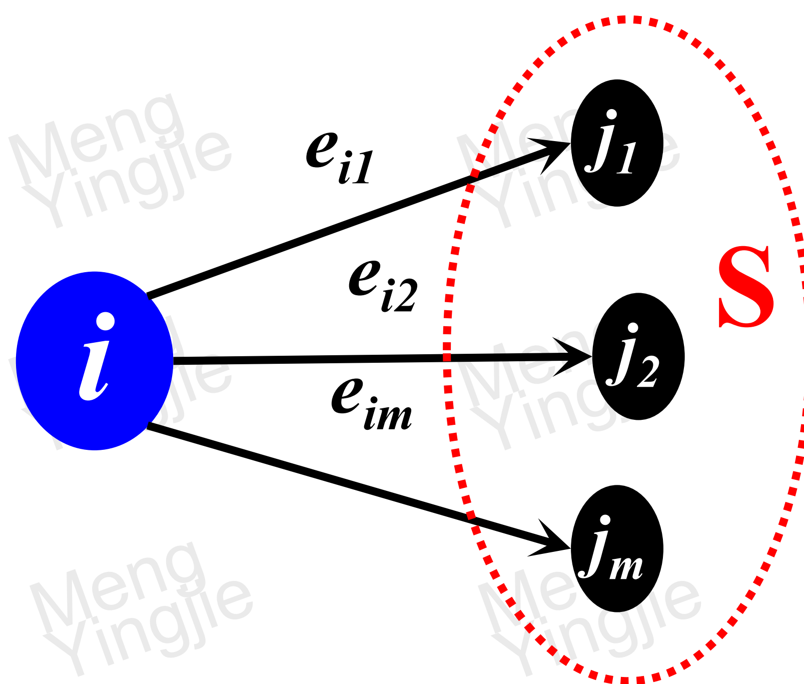


$$Ve(j)=\max_{\langle i,j\rangle\in T}\{Ve(i)+dut(\langle i,j\rangle)\},\quad 2\le j\le n$$

由于是求Max，因此Ve[1..n]初始均可置为0

(2).从Vl(n)=Ve(n)起向后递推求Vl(i)

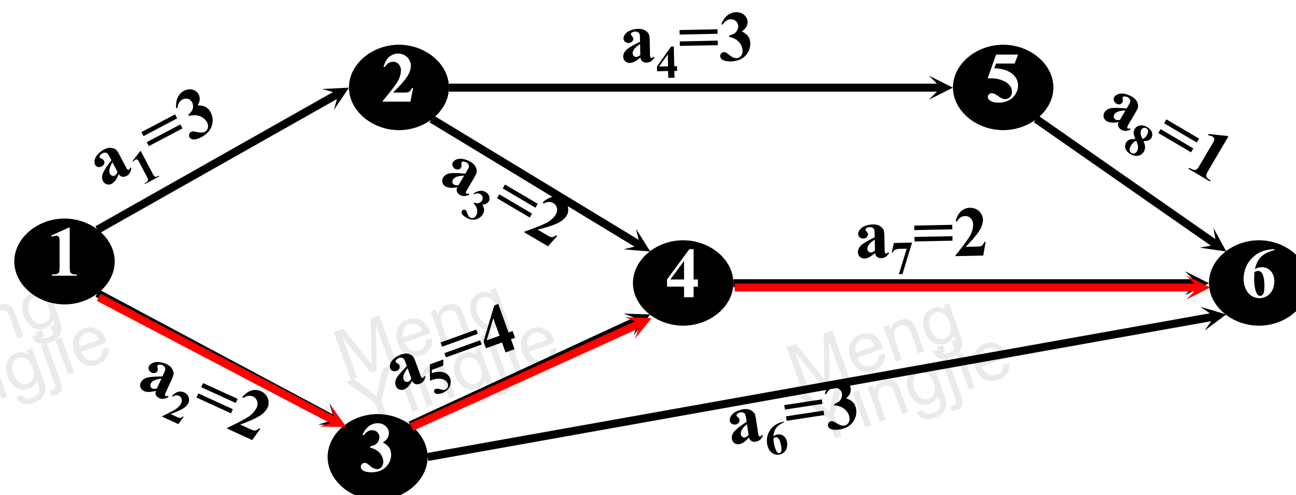
若将前一过程得到的拓扑序列存储的话，可以按照反向策略，用逆拓扑序列后退求出。



$$Vl(i)=\min_{<i,j>\in S}\{Vl(j)-dut(<i,j>)\}, \quad 1\le i\le n-1$$

由于是求Min，因此Vl[1..n]初始均可置为最大值Ve(n)

3.计算举例

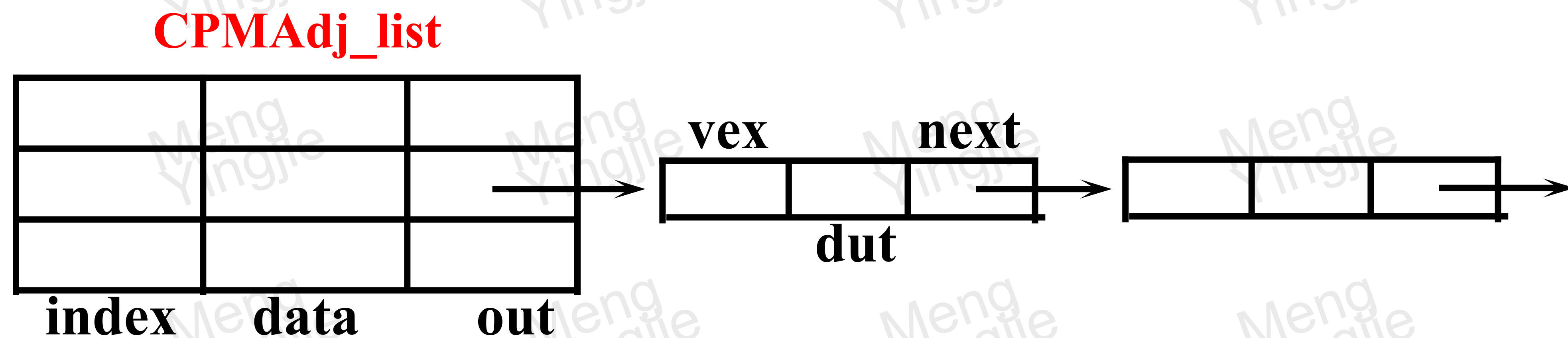


顶点	Ve	Vl
v_1	0	0
v_2	3	4
v_3	2	2
v_4	6	6
v_5	6	7
v_6	8	8

活动	e(i)=ve(j)	l(i)=Vl(k)- dut(a_i)	l(i)-e(i)
a_1	0	1	1
a_2	0	0	0
a_3	3	4	1
a_4	3	4	1
a_5	2	2	0
a_6	2	5	3
a_7	6	6	0
a_8	6	7	1

4、关键路径算法的处理过程：

(1) 存储结构设计：



index：初始放入度，运行过程中可被两个栈占用，拓扑栈和逆拓扑栈；

拓扑出栈时，逆拓扑进栈。

(2)、较为完整的算法处理主要步骤：

①. 计算Ve(i),i=1,2,…,n

(利用拓扑排序过程)

②. 计算Vl(i),i=1,2,…,n

(利用①得到的拓扑序列的逆序)

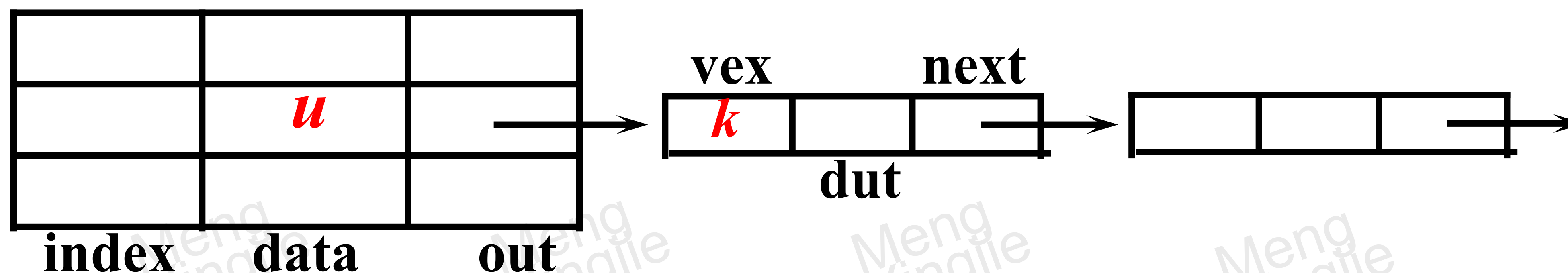
③. 计算e(i)及l(i)

(利用①、②得到Ve,Vl)

④. 计算e(i)–l(i), 输出差值为零的活动

现在对以上①、②进行细化：

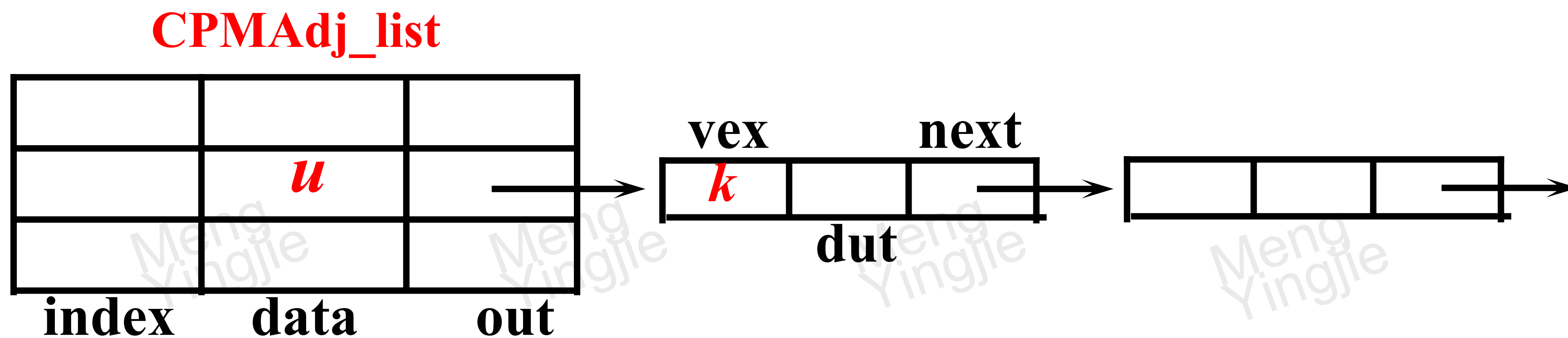
Meng Yingjie



求Ve的算法步骤：

(1).初始化： Ve[1..n] ←0 ;{由于求的是最大值，故所有Ve置零}
(2).搜索邻接表中入度为零的顶点，并令其进栈。
(3).当栈非空时，进行拓扑排序：
①退栈，输出栈顶元素u;
②(循环)在邻接表中扫描u的直接后继顶点k,
- 将k的入度减1,
- 若此时k的入度为零,则令k入栈。
- 经由u若能增大Ve(k),则增大
- 保存u到逆拓扑栈;

(4).若栈空，输出元素不足n则有环，否则拓扑排序完成。



求Vl的算法步骤：

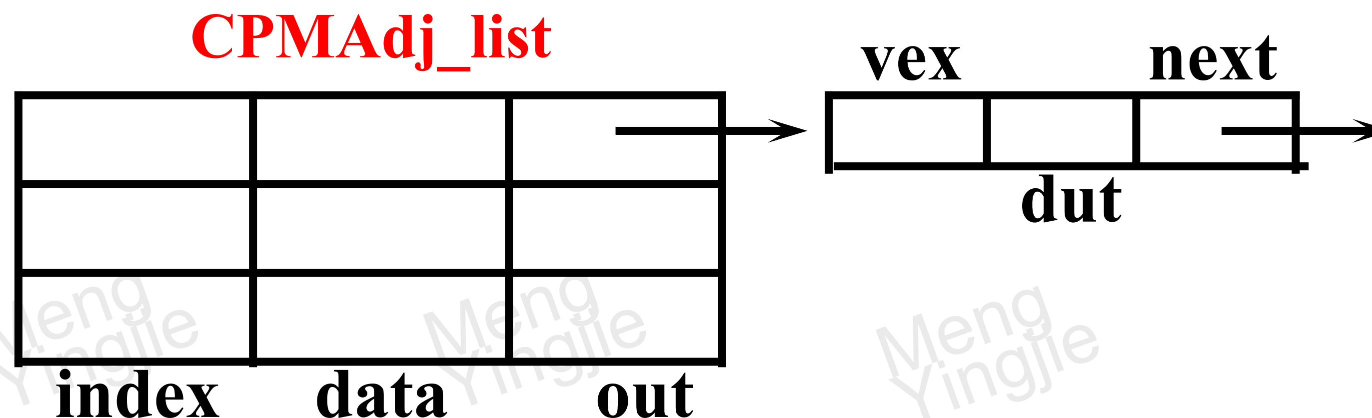
(1).初始化:若n为工程的终点，不影响工期则：Vl[n]=Ve[n],故：
Vl[1..n] ←Ve[n]；{由于求的是最小值，故所有Vl置最大值}

(2).当逆拓扑栈非空时，进行逆拓扑排序：

①退栈，输出栈顶元素u;

②(循环)在邻接表中扫描u的直接后继顶点k,
经由k若能缩小Vl(u),则缩小；

算法过程:




```
PROC   CPM(VAR A: CPMadj_list);
BEGIN    btop←0;   {逆拓扑栈初始化}   ftop←0; {拓扑栈初始化}
   FOR  i←1  TO  n  DO
       IF A[i].index=0  THEN 【A[i].index←ftop;  ftop←i;  Ve[i]←0】 ;
   m←0; {拓扑排序输出顶点计数器}
   while  ftop≠0  DO
     【 j=ftop;  ftop←A[ftop].index ; {拓扑退栈}
       A[j].index←btop;  btop←j; {逆拓扑入栈}
       m←m+1;
       p←A[j].out;
```

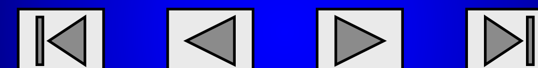
算法过程(续1):

```
while  p≠nil  DO    {修改拓扑出栈后相关顶点入度}
    【k←p↑.vex;    A[k].index←A[k].index-1;
      IF A[k].index=0 THEN 【A[k].index←ftop;  ftop←k】 ;
      IF Ve[k]<Ve[j] +p↑.dut THEN  Ve[k]←Ve[j]+p↑.dut ;
      p←p↑.next】 】 ;
IF  m<n  THEN 【write('The network has a cycle '); exit 】 ;
endnode←btop;    { endnode 为终点元素编号 }
Vl[endnode]←Ve[endnode];     {  Vl(n) ←Ve(n)  }
 while  btop≠0  DO    {依次求Vl}
 【j=btop;  btop←A[btop].index ;
   Vl[j]←Ve[endnode]; {Vl[j]一开始取最大值Ve(n)}
   p←A[j].out;
```

Meng Yingjie

算法过程(续2):

```
while  p≠nil  DO
      【k←p↑.vex;
        IF Vl[j]>Vl[k]－p↑.dut THEN  Vl[j]←Vl[k]－p↑.dut ;
        p←p↑.next】 】 ;
 writeln('ACT    E    L  ');
 FOR  k←1  TO n  DO { 输出结果 }
    【p←A[k].out;
      while  p≠nil  DO
        【write('<', k,p↑.vex, '>',  Ve[k], Vl[p↑.vex]- p↑.dut );
         IF Ve[k]=Vl[p↑.vex]- p↑.dut THEN  writeln('*')
                                      ELSE  writeln;
         p←p↑.next】 】 ;
END;
```

本节结束

Meng Yingjie

引言:

在交通网中常常提出这样的问题:

- 两地之间是否有路径可通?
- 在有多条通路的情况下，哪一条(代价)最短?

交通网可以用一个带权的图,顶点表示城镇,边表示城市间的道路,权表示道路的里程长度(或费用、时间等)。

以上问题就是网中求**最短路径**的问题,即求两个顶点间的长度最短的路径。

这里的路径长度是指路径上所带的权值之和，而不是路径上边的数目。

概念简单，算法不复杂，但计算量很大。

Meng Yingjie

一.单源最短路径:

单源最短路径是指从一个顶点到其它各顶点之间的最短路径(single-source shortest path)。

迪杰斯特拉(E.W.Dijkstra)于1959年提出了一个寻找单源最短路径的方法。

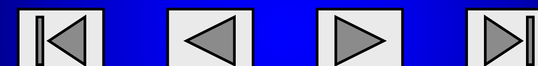
其基本思想是，设置一个顶点集合S，并不断地作贪心选择来扩充这个集合。

(该算法属于算法设计方法中的贪心算法(greedy selector)类——总是作出当前看来最好的选择，通过获取局部最优，最终达到获取整体最优)

基本做法：把图中所有顶点分成两组，第一组包括已经确定最短路径的顶点，按最短路径长度递增的顺序逐个把第二组中的顶点加到第一组中去，直到从v_0出发可以到达的所有顶点都包括到第一组中——即按广义递增顺序进行。

在这个过程中，总保持从v0到第一组各顶点的最短路径长度都不大于从v_0到第二组的任何顶点的最短路径长度。

另外，每个顶点对应一个距离值，第一组的结点对应的距离值就是从v_0到此顶点的最短路径长度，第二组的顶点对应的距离值是从v_0到此顶点的只包括第一组的顶点为中间顶点的最短路径长度。



即：

先求到源点全局第 1 条最短的

再求到源点全局第 2 条最短的

全局第 3 条最短的

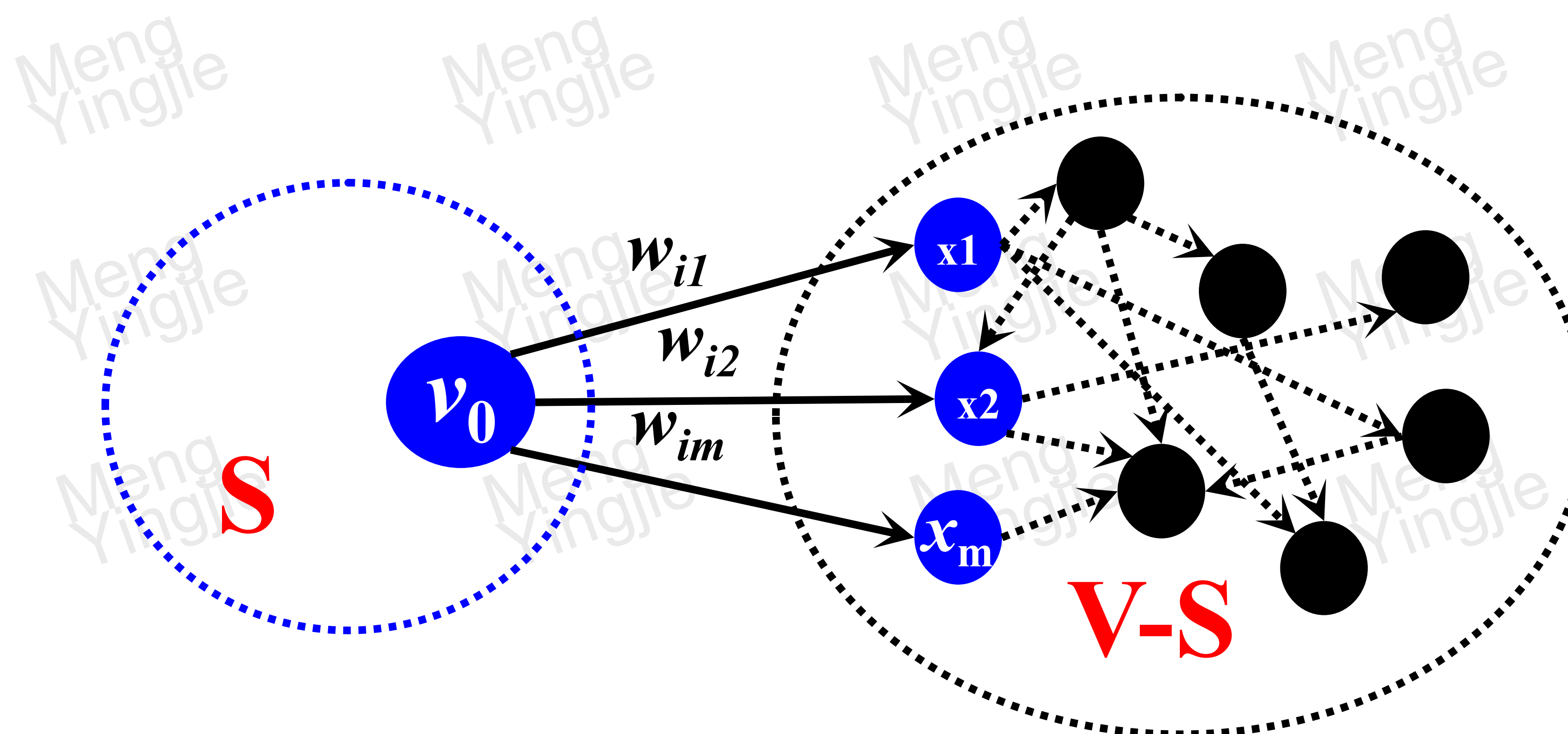
。。。。。。

Meng Yingjie

具体做法:

1.先确定全局第一个离v_0最近的顶点x(全局第一条最短的路径$v_0 \rightarrow x$):

$x=\min(w_{i1},w_{i2},\ldots,w_{im})$ (从v0可直达的边中找)

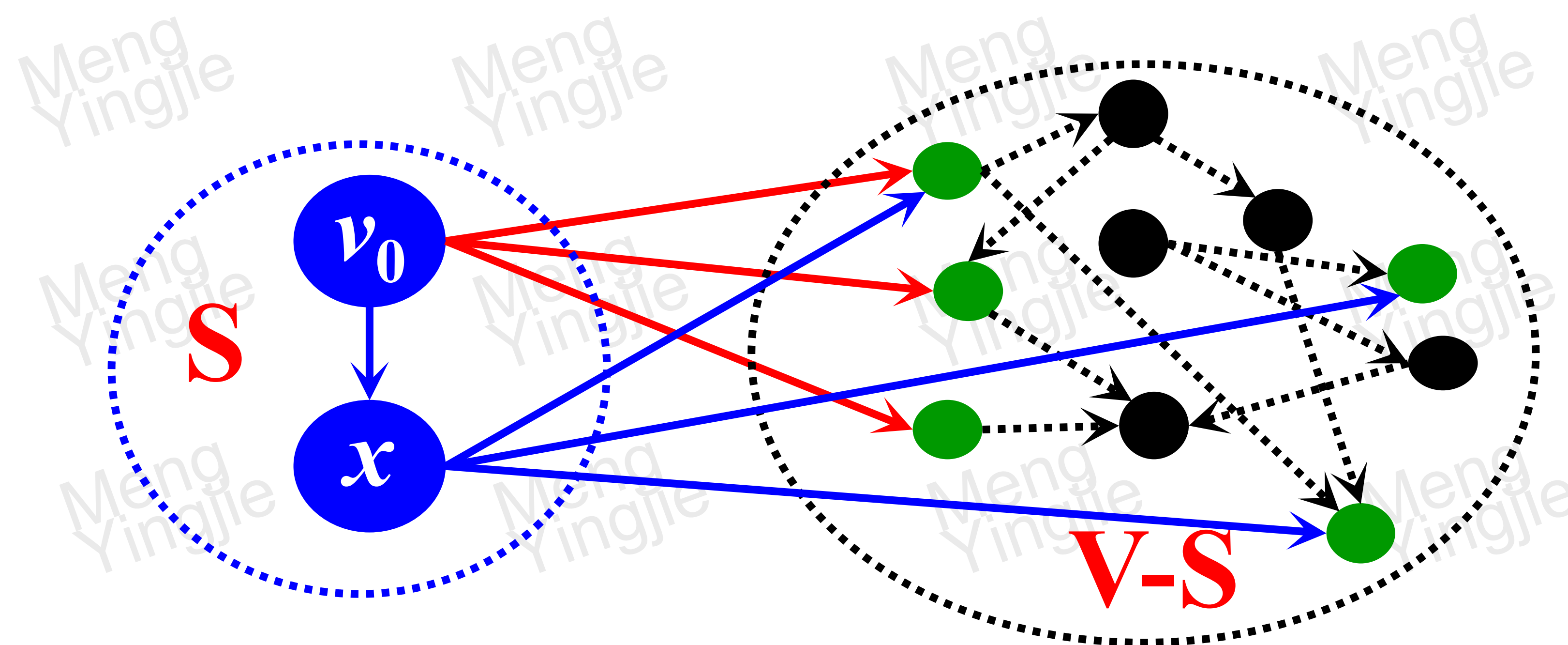


具体做法(续)：

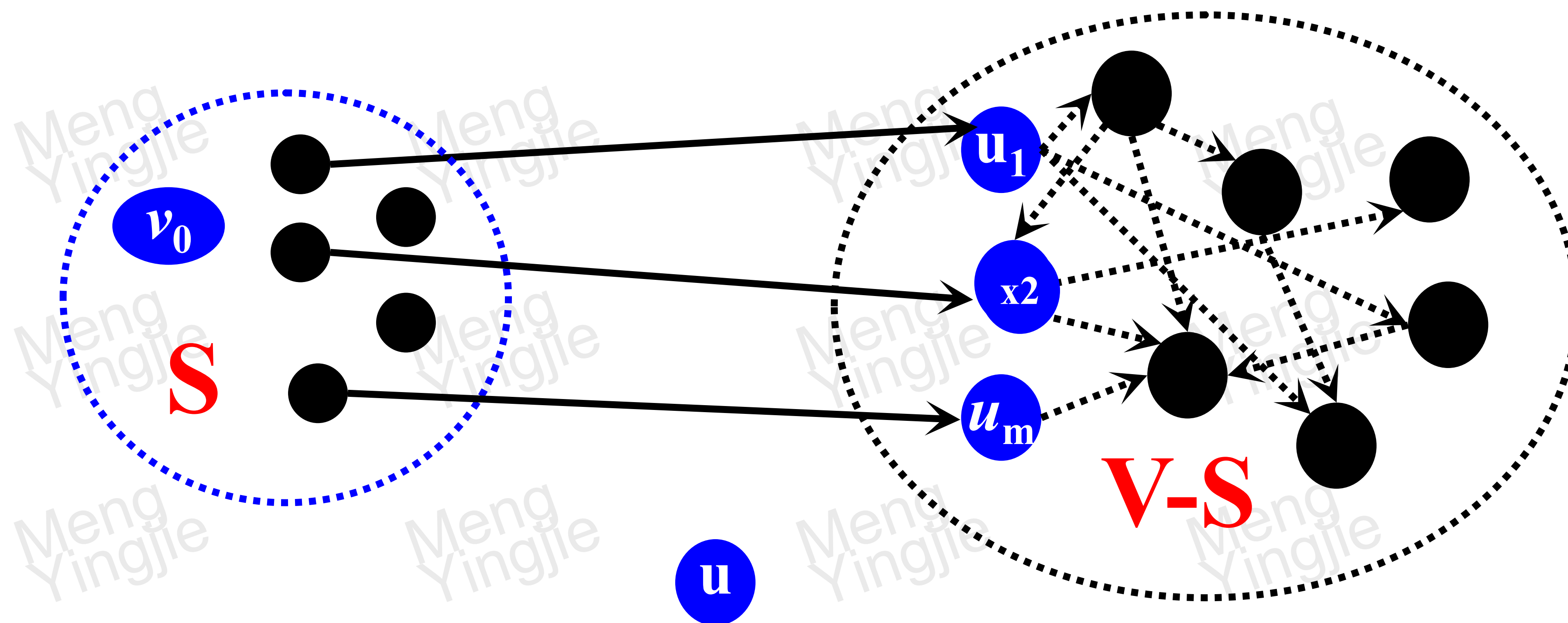
2.确定全局第二个离v_0最近的顶点y(全局第二条最短的路径v_0…y)，必定是下列两种情况之一：

- $v_0 \to y$ (红线连接的路径中的最小者)
- $v_0 \to x \to y$ (兰线连接的路径中的最小者)

取二者最小者，必是全局第二条最短的。



依次类推，可以求得第3条，第4条，…,第n-1条最短路径。



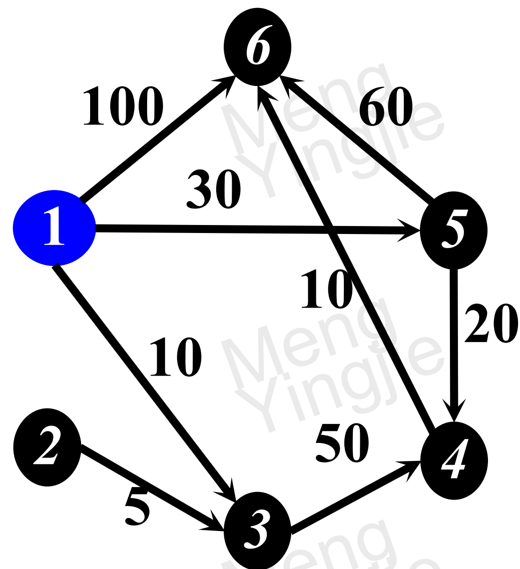
u

每次:

（1）在V-S中找 1 个距离S最近的 u，将u加入S

（2）通过 u 看可否将S 到 V-S中的结点距离拉近（眼前），能拉近就拉近。

例：

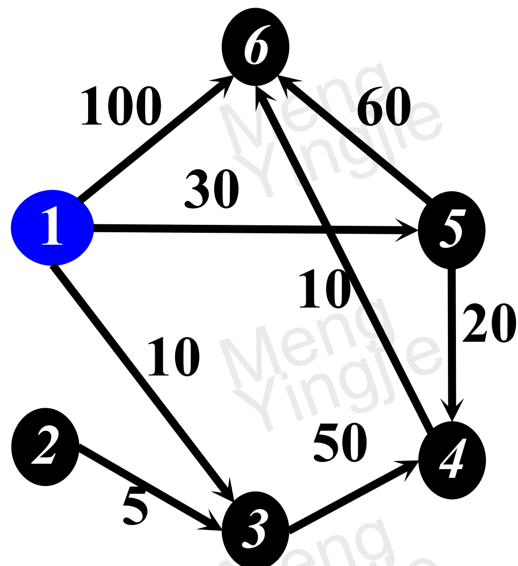


$$\begin{bmatrix} \infty & \infty & 10 & \infty & 30 & 100 \\ \infty & \infty & 5 & \infty & \infty & \infty \\ \infty & \infty & \infty & 50 & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & 10 \\ \infty & \infty & \infty & 20 & \infty & 60 \\ \infty & \infty & \infty & \infty & \infty & \infty \end{bmatrix}$$

终点	从V_1到各终点得最短路径变迁过程	
V_2	∞	
V_3	10 (V_1,V_3)	全局第1条最短路径
V_4	∞	
V_5	30 (V_1,V_5)	
V_6	100 (V_1,V_6)	
距S最近点u	V_3	
新S	$\{V_1,V_3\}$	
需修改点V-S	$\{V_2,V_4,V_5,V_6\}$	

Meng Yingjie

例:



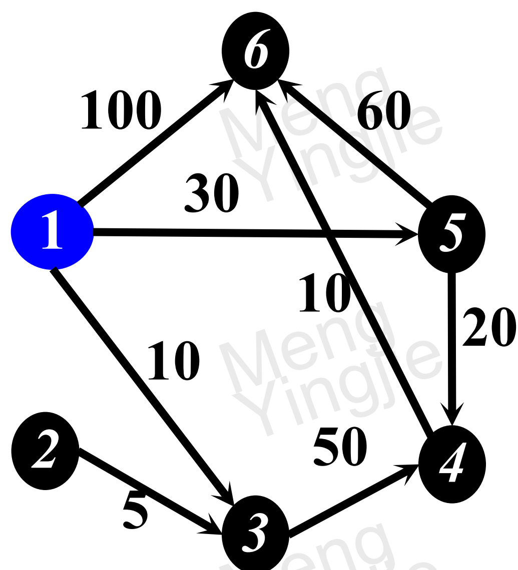
$$\begin{bmatrix} \infty & \infty & 10 & \infty & 30 & 100 \\ \infty & \infty & 5 & \infty & \infty & \infty \\ \infty & \infty & \infty & 50 & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & 10 \\ \infty & \infty & \infty & 20 & \infty & 60 \\ \infty & \infty & \infty & \infty & \infty & \infty \end{bmatrix}$$

终点	从V_1到各终点得最短路径变迁过程		
V_2	∞	∞	
V_3	10 (V_1,V_3)	全局第1条最短路径	
V_4	∞	60 (V_1,V_3,V_4)	
V_5	30 (V_1,V_5)	30 (V_1,V_5)	全局第2条最短路径
V_6	100 (V_1,V_6)	100 (V_1,V_6)	

距S最近点u	V_3	V_5	
新S	$\{V_1,V_3\}$	$\{V_1,V_3,V_5\}$	
需修改点V-S	$\{V_2,V_4,V_5,V_6\}$	$\{V_2,V_4,V_6\}$	

Meng Yingjie

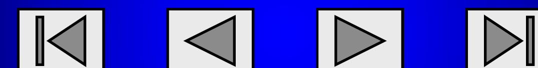
例:



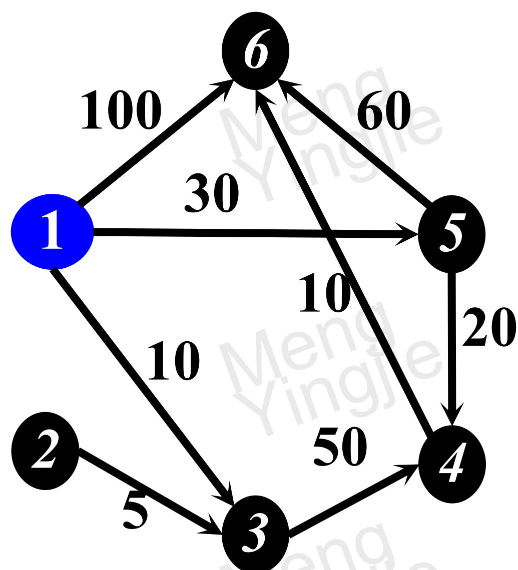
$$\begin{bmatrix} \infty & \infty & 10 & \infty & 30 & 100 \\ \infty & \infty & 5 & \infty & \infty & \infty \\ \infty & \infty & \infty & 50 & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & 10 \\ \infty & \infty & \infty & 20 & \infty & 60 \\ \infty & \infty & \infty & \infty & \infty & \infty \end{bmatrix}$$

终点	从V_1到各终点得最短路径变迁过程			
V_2	∞	∞	∞	
V_3	10 (V_1,V_3)	全局第1条最短路径		
V_4	∞	60 (V_1,V_3,V_4)	50 (V_1,V_5,V_4)	第3条
V_5	30 (V_1,V_5)	30 (V_1,V_5)	全局第2条最短路径	
V_6	100 (V_1,V_6)	100 (V_1,V_6)	90 (V_1,V_5,V_6)	

距S最近点u	V_3	V_5	V_4	
新S	$\{V_1,V_3\}$	$\{V_1,V_3,V_5\}$	$\{V_1,V_3,V_4,V_5\}$	
需修改点V-S	$\{V_2,V_4,V_5,V_6\}$	$\{V_2,V_4,V_6\}$	$\{V_2,V_6\}$	



例:

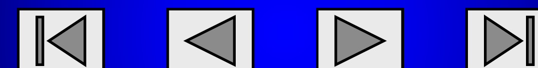


$$\begin{pmatrix} \infty & \infty & 10 & \infty & 30 & 100 \\ \infty & \infty & 5 & \infty & \infty & \infty \\ \infty & \infty & \infty & 50 & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & 10 \\ \infty & \infty & \infty & 20 & \infty & 60 \\ \infty & \infty & \infty & \infty & \infty & \infty \end{pmatrix}$$

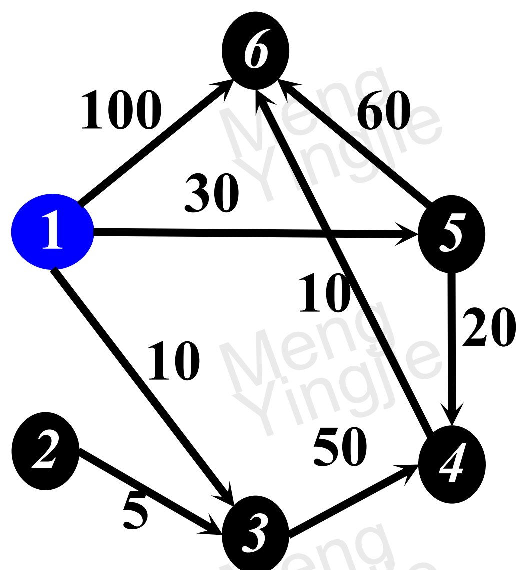
终点	从V_1到各终点得最短路径变迁过程				
V_2	∞	∞	∞	∞	
V_3	10 (V_1,V_3)	全局第1条最短路径			
V_4	∞	60 (V_1,V_3,V_4)	50 (V_1,V_5,V_4)	第3条	
V_5	30 (V_1,V_5)	30 (V_1,V_5)	全局第2条最短路径		
V_6	100 (V_1,V_6)	100 (V_1,V_6)	90 (V_1,V_5,V_6)	60 (V_1,V_5,V_4,V_6)	第4条

距S最近点u	V_3	V_5	V_4	V_6	
新S	$\{V_1,V_3\}$	$\{V_1,V_3,V_5\}$	$\{V_1,V_3,V_4,V_5\}$	$\{V_1,V_3,V_4,V_5,V_6\}$	
需修改点V-S	$\{V_2,V_4,V_5,V_6\}$	$\{V_2,V_4,V_6\}$	$\{V_2,V_6\}$	$\{V_2\}$	

Meng Yingjie



例:



$$\begin{bmatrix} \infty & \infty & 10 & \infty & 30 & 100 \\ \infty & \infty & 5 & \infty & \infty & \infty \\ \infty & \infty & \infty & 50 & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & 10 \\ \infty & \infty & \infty & 20 & \infty & 60 \\ \infty & \infty & \infty & \infty & \infty & \infty \end{bmatrix}$$

终点	从V_1到各终点得最短路径变迁过程				
V_2	∞	∞	∞	∞	∞
V_3	10 (V_1,V_3)	全局第1条最短路径			
V_4	∞	60 (V_1,V_3,V_4)	50 (V_1,V_5,V_4)	第3条	
V_5	30 (V_1,V_5)	30 (V_1,V_5)	全局第2条最短路径		
V_6	100 (V_1,V_6)	100 (V_1,V_6)	90 (V_1,V_5,V_6)	60 (V_1,V_5,V_4,V_6)	第4条

距S最近点u	V_3	V_5	V_4	V_6	V_2
新S	$\{V_1,V_3\}$	$\{V_1,V_3,V_5\}$	$\{V_1,V_4,V_3,V_5\}$	$\{V_1,V_3,V_4,V_5,V_6\}$	$\{V_1,V_2,V_3,V_4,V_5,V_6\}$
需修改点V-S	$\{V_2,V_4,V_5,V_6\}$	$\{V_2,V_4,V_6\}$	$\{V_2,V_6\}$	$\{V_2\}$	

算法设计步骤：

(一).存储结构的组织

图——邻接矩阵(cost)

最短路径的长度值存储辅助向量dist[*i*],

最短路径的顶点序列存储辅助向量path[*i*],

(二).处理步骤：

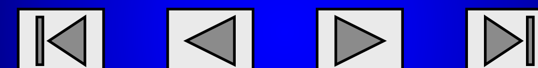
1、初始化

① dist初始状态获取:

$$dist[i]=\left\{\begin{array}{ll} w_i, & <v_0,v_i>\in E(G) \\ \infty, & \text{反之} \end{array}\right\} = cost[v_0,i]$$

用常量Max，相对于问题的一个充分大的数值

Meng Yingjie



② path初始状态获取:

IF $<v_0,v_i> \in E(G)$ THEN Path[i] $\leftarrow \{v_0,v_i\}$,

ELSE Path[i] $\leftarrow \varnothing$

③ S $\leftarrow \{v_0\}$

由于是逐条求解，还需设置计数器m,初始=1

2、逐条求解 (重复)

① 在V–S中挑选最小的dist[u]，即选择距S最近的顶点u

② 将u并入S

③ 通过u尝试缩小V–S中的顶点i的dist[i]，若能缩小则缩小(并同步替换path[i])

算法过程：

PROC Dijkstra(VAR A: cost; v_0:integer);

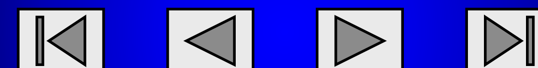
{$A_{n\times n}$网的邻接矩阵,dist[i]为v_0到i的当前最短路径长,path[i]为相应的路径}

BEGIN **FOR** i←1 **TO** n **DO** {产生dist、path初值}

【dist[i]←cost[v_0,i];

IF dist[i]<max **THEN** path[i]←[v_0]+[i] **ELSE** path[i]←∅】 ;

S←[v0]; m←1; {初始化}



```
while  m<n-1  DO
    【wm←max;   u=v0 ;    {u为待选顶点，设置初值}
     FOR  i←1  TO  n  DO     {挑选dist最小的顶点u}
          IF (i∉S)AND(dist[i]<wm) THEN 【u←i;  wm←dist[i]】 ;
     S←S+[u];
     FOR  i←1  TO  n  DO     {修改V-S中的顶点当前dist值}
          IF (i∉S)AND(dist[u]+cost[u,i]<dist[i])
          THEN  【dist[i]←dist[u]+cost[u,i];
                  path[i]←path[u]+[i]】 ;
     m←m+1 】
END;
```

复杂度$O(n^2)$

Meng Yingjie

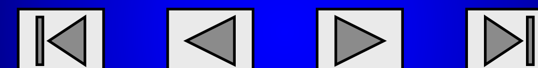
二.每对顶点的最短路径(all-pairs shortest path)

欲求每对顶点之间的最短路径只需运行Dijkstra算法n次即可，算法复杂度为$O(n^3)$.

但也可以采用运算模式更为简单的另一算法弗洛伊德 (Floyed)算法，其时间复杂度仍为$O(n^3)$.

关于Floyed算法的改进及变形这里不再论述。

下面简单介绍Floyed算法的基本思想。



Floyed 算法基本思想：

对图的顶点从1到n进行编号。A和path均为n阶方阵。

通过运算,递推地产生一个矩阵序列$A^{(0)}$, $A^{(1)}$, $A^{(2)}$,…, $A^{(k)}$, …,$A^{(n)}$。

其中:$A^{(0)}[i,j]=cost[i,j]$,表示从顶点V_i到V_j中间经过顶点序号不大于0(即不经过任何顶点)的**最短路径长度**。

一般地$A^{(k)}[i,j]$表示从顶点V_i到V_j中间经过顶点序号不大于k的最短路径长度。

$A^{(n)}[i,j]$表示从顶点V_i到V_j中间经过顶点序号不大于n的最短路径长度。这就是所求的从V_i到V_j的最短路径长度。

Meng Yingjie

Floyed 算法基本思想(续):

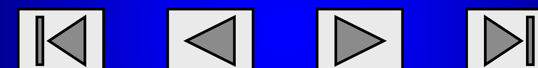
递推产生$A^{(0)}, A^{(1)}, A^{(2)},\ldots, A^{(k)}, \ldots,A^{(n)}$的过程，就是逐步允许越来越多的顶点作为路径的中间顶点，直到所有的顶点都允许作为中间顶点，这时最短路径就求出来了。

设已经求得$A^{(k-1)}$,则

$A^{(k)}[i,j]=\min\{A^{(k-1)}[i,j],\ A^{(k-1)}[i,k]+A^{(k-1)}[k,j]\}, 1\le k\le n$

$path^{(k)}[i,j]$为相应的路径。

算法过程如下，算法执行中A的值被逐渐代之为$A^{(1)}, A^{(2)},\ldots, A^{(k)}, \ldots,A^{(n)}$;在算法每步中path[i,j]是从V_i到V_j中间顶点序号不大于k的最短路径上V_j的前一个顶点的序号；算法结束时，由path[i,j]的值追溯,可以得到从V_i到V_j的最短路径。



算法过程：

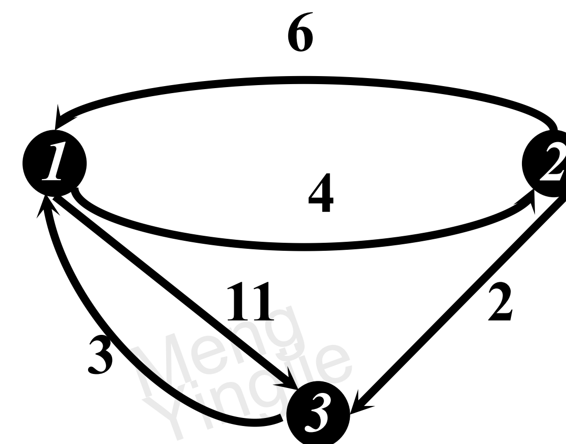
```
PROC  Floyed(cost,A,path);
BEGIN
    FOR  i←1  TO  n  DO
        FOR  j←1  TO  n  DO
            【a[i,j]←cost[i,j];
             IF (a[i,j] ≠max) THEN path[i,j]←i
                              ELSE path[i,j]←0】;
    FOR  k←1  TO  n  DO
        FOR  i←1  TO  n  DO
            FOR  j←1  TO  n  DO
             IF (a[i,k]+a[k,j]<a[i,j])
             THEN 【 a[i,j]←a[i,k]+a[k,j];
                     path[i,j]←path[k,j]】
END;
```

产生$A^{(k)}$（标注第二组循环）

Meng Yingjie

例:

$$cost=\begin{bmatrix}0 & 4 & 11\\ 6 & 0 & 2\\ 3 & \infty & 0\end{bmatrix}$$



$$A^{(0)}=\begin{bmatrix}0 & 4 & 11\\ 6 & 0 & 2\\ 3 & \infty & 0\end{bmatrix}\quad A^{(1)}=\begin{bmatrix}0 & 4 & 11\\ 6 & 0 & 2\\ 3 & 7 & 0\end{bmatrix}\quad A^{(2)}=\begin{bmatrix}0 & 4 & 6\\ 6 & 0 & 2\\ 3 & 7 & 0\end{bmatrix}\quad A^{(3)}=\begin{bmatrix}0 & 4 & 6\\ 5 & 0 & 2\\ 3 & 7 & 0\end{bmatrix}$$

$$path^{(0)}=\begin{bmatrix}1 & 1 & 1\\ 2 & 2 & 2\\ 3 & 0 & 3\end{bmatrix}\quad path^{(1)}=\begin{bmatrix}1 & 1 & 1\\ 2 & 2 & 2\\ 3 & 1 & 3\end{bmatrix}\quad path^{(2)}=\begin{bmatrix}1 & 1 & 2\\ 2 & 2 & 2\\ 3 & 1 & 3\end{bmatrix}\quad path^{(3)}=\begin{bmatrix}1 & 1 & 2\\ 3 & 2 & 2\\ 3 & 1 & 3\end{bmatrix}$$

算法在结束时,由path可推知任何一对顶点的最短路径是什么。

例如，欲知V_1到V_3的最短路径. 由path[1,3]=2，知V_3的前一个顶点是V_2,由path[1,2]=1，知V_2的前一个顶点是V_1,因此最短路径为(V_1,V_2, V_3),其长度A[1,3]=6.

Meng Yingjie

Floyed 算法改进：

上述算法在结束时，最短路径长度可以直接获取，但最短路径顶点序列需要追溯才可获取，若把path的元素类型由整数值类型改为集合类型，在算法结束时就可直接从path获取路径上顶点序列。算法过程如下：

```
PROC  Floyed(cost,A,path);
BEGIN  FOR  i←1  TO  n  DO
          FOR  j←1  TO  n  DO
            【a[i,j]←cost[i,j];
             IF (i≠j) AND (a[i,j]<max) THEN path[i,j]←[i]+[j]
                                       ELSE  path[i,j]←∅】  ;
     FOR  k←1  TO  n  DO
        FOR  i←1  TO  n  DO
           FOR  j←1  TO  n  DO
             IF (a[i,k]+a[k,j]<a[i,j])
             THEN  【 a[i,j]←a[i,k]+a[k,j];
                     path[i,j]←path[i,k]+path[k,j]】
END;
```

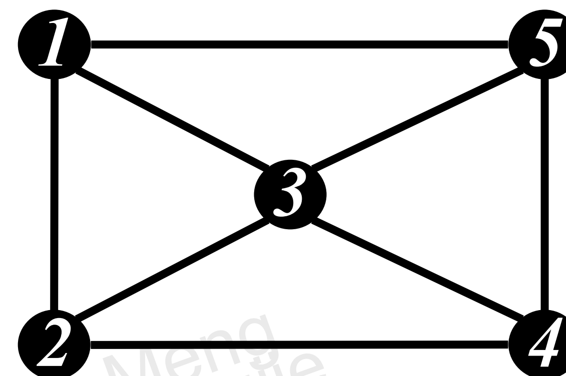
产生A(k)

Meng Yingjie

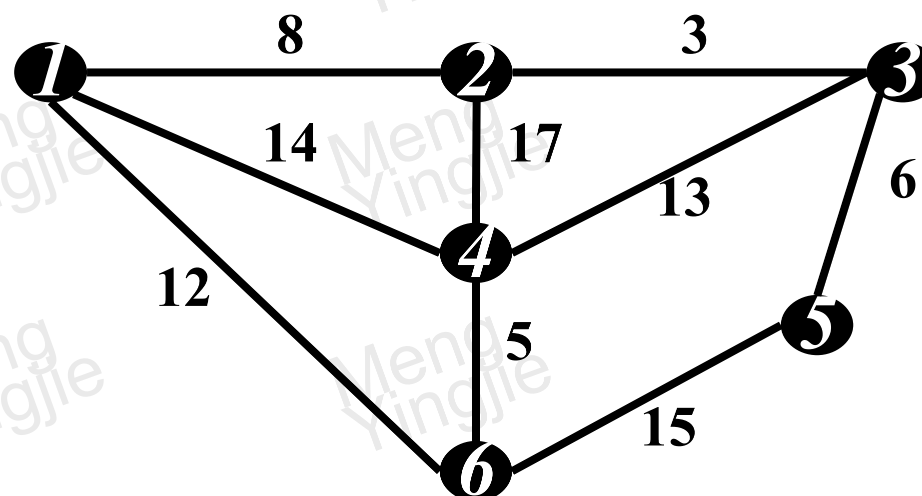
本节结束

Meng Yingjie

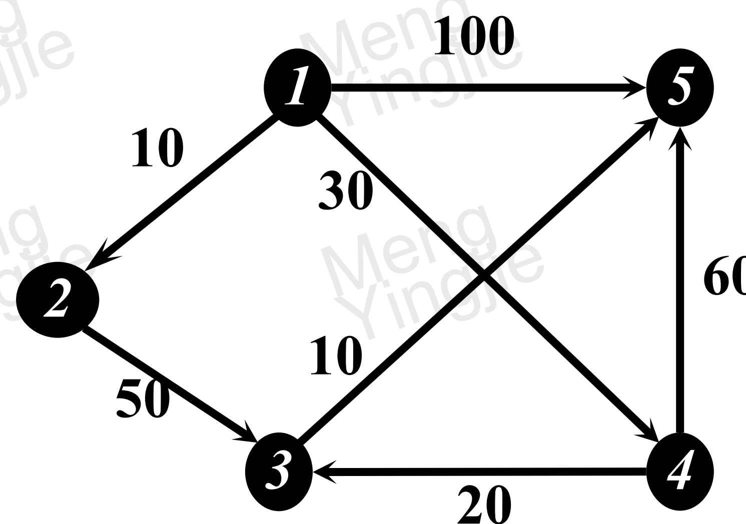
1.对右图给出其邻接表，并给出从3进行深 度和广度遍历的结果。



2.对右图，分别按照PRIM算法和Kruskal构造其最小生成树。



3.对右图，给出其邻接矩阵求从顶点1出发分别到其它各顶点的最短路径。并写出计算过程。



本章结束

Meng Yingjie

拓扑排序、拓扑分类(toplogical sorting)

指把“偏序”嵌入到一个线性次序中去。即把对象排列成线性序列$a_1,a_2,\ldots,a_n$,使得每当$a_j \prec a_k$时，就有$j<k$。

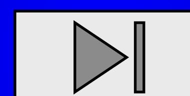
所谓“偏序”是指集合的对象之间的一种(序)关系。

设S是一个集合，如果S上的一个关系R满足自反性、反对称性和传递性，则称R是S上的一个偏序关系。

并把序偶(S,R)称作偏序集。

偏序一般用“$\prec$”表示，读作“前于”或“先于”。

拓扑分类很有用。用在有向图，意味着把平面上的方框图重新排成一行，使得所有的箭头都指向后方。



PERT(程序评价和审定技术)

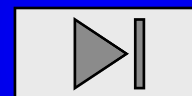
(program evaluation and review technique)

一种经营管理方法。

为了完成预定的计划目标，需要完成许多任务和产生许多事件。计划评估法就是用来定义这些任务和事件，并确定它们之间的相互关系。根据各任务的估计，确定各任务的完成日期和各任务之间的依赖关系，从而建立起一个网络。

网络的最长路径叫关键路径，它决定完成总任务的时间。利用计算机可以很快地完成计划的确定分配、调度、以及成本分析等工作。

Meng Yingjie



附录结束

Meng Yingjie