7.6 最短路径算法

最短路径(Shortest Path)问题

- 如果图中从一个顶点可以到达另一个顶点，则称这两个顶点间存在一条路径。
- 从一个顶点到另一个顶点间可能存在多条路径，而每条路径上经过的边数并不一定相同。
- 如果图是一个带权图，则路径长度为路径上各边的权值的总和，两个顶点间路径长度最短的那条路径称为两个顶点间的最短路径，其路径长度称为最短路径长度。
- 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？
- 集成电路设计、GPS导航、路由选择、铺设管线等

7.6 最短路径算法

- 问题解法
  - 边上权值非负情形的单源最短路径问题
    - — Dijkstra算法
  - 所有顶点之间的最短路径问题
    - — Floyd算法

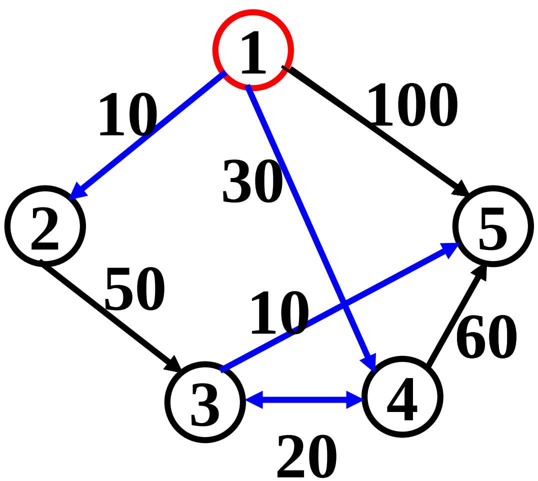
7.6最短路径算法

- 艾兹格·W·迪杰斯特拉 (Edsger Wybe Dijkstra，1930年5月11日~2002年8月6日)荷兰人。计算机科学家，毕业就职于荷兰Leiden大学，早年钻研物理及数学，而后转为计算学。曾在1972年获得过素有计算机科学界的诺贝尔奖之称的图灵奖，之后，他还获得过1974年 AFIPS Harry Goode Memorial Award、1989年ACM SIGCSE计算机科学教育教学杰出贡献奖、以及2002年ACM PODC最具影响力论文奖。

1 提出"goto有害论"-操作系统，虚拟存储
2 提出信号量和PV原语- 操作系统，进程同步
3 "哲学家聚餐"问题 – 操作系统，死锁
4 提出银行家算法-操作系统中资源分配问题
5 最短路径优先算法（ Dijkstra算法 ）的创造者;
6 THE操作系统的设计者和开发者;
7 与D. E. Knuth并称为我们这个时代最伟大的计算机科学家。

7.6 最短路径算法

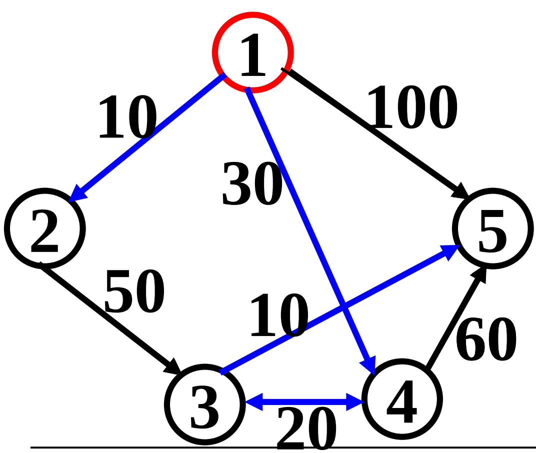
- **Dijkstra算法的基本思想**
  - Dijkstra提出按路径长度的递增次序，逐步产生最短路径的贪心算法----Dijkstra算法（SPF算法）。
  - 首先求出长度最短的一条最短路径,再参照它求出长度次短的一条最短路径，依次类推，直到从顶点*v* 到其它各顶点的最短路径全部求出为止。



源点S	中间结点	终点	路径长度
1		2	10
1		4	30
1	4	3	50
1	4 3	5	60

7.6 最短路径算法

- Dijkstra算法的数据结构
  - 假设带权有向图G=（V，E），其中V={ 1，2，...n }，顶点1为源点。图G的存储结构：采用带权的邻接矩阵C表示。
  - 一维数组D[n]：D[*i*]表示源点1到顶点*i*的当前最短路径长度，初始时，D[*i*]=C[1][*i*]；
  - 一维数组P[n]：P[*i*]表示源点1到顶点*i*的当前最短路径上，最后经过的顶点，初始时，P[*i*]=1（源点）；
  - S[n]：存放源点和已生成的终点，其初态为只有一个源点1



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

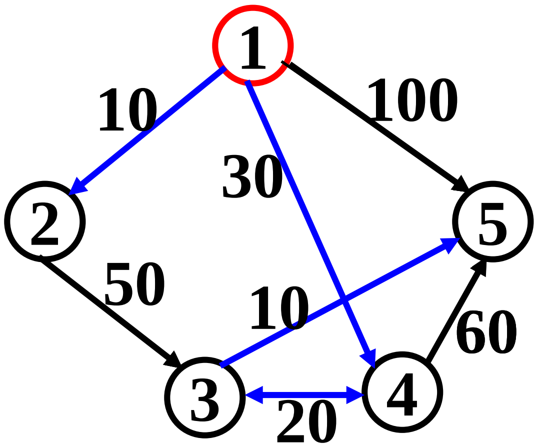
D		10	∞	30	100
P		1	1	1	1
S	T	F	F	F	F

7.6 最短路径算法

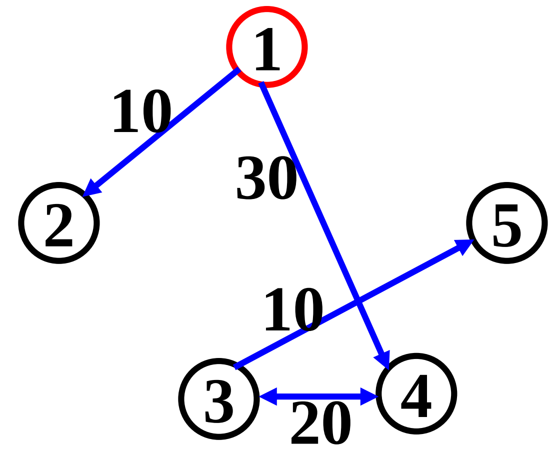
- Dijkstra算法实现步骤：
  - 1. 将Ｖ分为两个集合S（最短路径已经确定的顶点集合）和V-S（最短路径尚未确定的顶点集合。初始时，S={ 1 }，D[i]=C[1][i]（i=2,3,…n），P[i]=1(源点，i≠1) 。
  - 2. 从S之外即V-S中选取一个顶点w，使D[w]最小（即选这样的w，D[w]=min{ D[i]| i∈V-S }），于是从源点到达w只通过S中的顶点，且是一条最短路径（选定路径），并把w加入集合S。
  - 3. 调整D中记录的从源点到V-S中每个顶点的最短距离，即从原来的D[v]和D[w]＋C[w][v]中选择最小值作为D[v]的新值，且P[v]=w。
  - 7. 重复2和3，直到S中包含V的所有顶点为止。结果数组Ｄ就记录了从源到V中各顶点的最短 距离（数组P记录最短路径）。

7.6 最短路径算法

Dijkstra算法示例



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



循环	S	w	D[2]	D[3]	D[4]	D[5]	P[2]	P[3]	P[4]	P[5]
初态	{1}	-	10	∞	30	100	1	1	1	1
1	{1,2}	2	10	60	30	100	1	2	1	1
2	{1,2,4}	4	10	50	30	90	1	4	1	4
3	{1,2,4,3}	3	10	50	30	60	1	4	1	3
4	{1,2,4,3,5}	5	10	50	30	60	1	4	1	3

7.6最短路径算法

Dijkstra算法的实现

```
void  Dijkstra(GRAPH  C, costtype D[n+1]，  int P[n+1]，  bool S[n+1])
{   for ( i=2 ; i<=n; i++ )
    {     D[i]=C[1][i] ;  S[i]=False ;P[i]=1;}
   S [1]= True ;
   for( i=1; i<=n-1; i++)
   {  w=MinCost ( D, S ) ;
      S[w]=True ;
      for ( v=2 ; v<= n ; n++ )
         if ( S[v]!=True )
         {    sum=D[w] + C[w][v] ;
              if (sum < D[v] ){D[v] = sum ; P[v]=w;}}
   }
}//   时间复杂度：O（n²）
```

```
costtype MinCost (D, S)
{
temp = INFINITY ;
 w = 2 ;
 for ( i=2 ; i<=n ; i++ )
   if (!S[i]&&D[i]<temp)
    {  temp = D[i] ;
       w = i ;
    }
return  w ;
}
```

普里姆（Prim）算法的实现

```
void Prim(Costtype C[n+1][n+1] )
{ costtype LOWCOST[n+1]; int CLOSEST[n+1]; int
    i,j,k; costtype min;
   for( i=2; i<=n; i++ )
   {   LOWCOST[i] = C[1][i];    CLOSEST[i] = 1;   }
   for( i = 2; i <= n; i++ )
   {    min = LOWCOST[i];
        k = i;
        for( j = 2;  j <= n;  j++ )
             if ( LOWCOST[j] < min )
              { min = LOWCOST[j] ;   k=j; }
        cout << "(" << k << "," << CLOSEST[k] << ")"
    << end1;
        LOWCOST[k] = 0 ;
        for ( j = 2;  j <= n;  j++ )
 if ( C[k][j] < LOWCOST[j])
      {    LOWCOST[j]=C[k][j];  CLOSEST[j]=k;   }
   }
} /* 时间复杂度：O( |V|² )
```

Dijkstra算法的实现

```
void Dijkstra(GRAPH C, costtype D[n+1]，int P[n+1]，bool
      S[n+1])
{  for ( i=2 ; i<=n; i++ )
  {    D[i]=C[1][i] ; S[i]=False ;P[i]=1;}
  S [1]= True ;
  for( i=1; i<=n-1; i++)
  { w=MinCost ( D, S ) ;
     S[w]=True ;
     for ( v=2 ; v<= n ; n++ )
        if ( S[v]!=True )
        {   sum=D[w] + C[w][v] ;
             if (sum < D[v] ){D[v] = sum ; P[v]=w;}}
  }
}//  时间复杂度：O（n²）
```

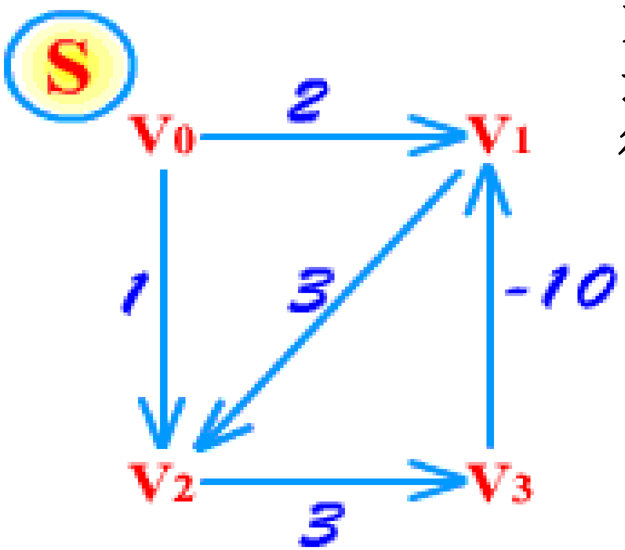
```
costtype MinCost (D, S)
{
temp = INFINITY ;
 w = 2 ;
 for ( i=2 ; i<=n ; i++ )
   if (!S[i]&&D[i]<temp)
    { temp = D[i] ;
        w = i ;
    }
return  w ;
}
```

Prim 与 Dijkstra 算法对比：

区别	Prim算法构造最小生成树	Dijkstra算法构造单源最短路径
起始点	任选一个结点	有一个确定的起点（源点），其余为终点
连通顶点	连通所有顶点，且总造价最低	源点到各终点的两顶点间的最短路径
加入集合外顶点的修改方式	if (C[k][j] < LowCost[j]) { LowCost[j]=C[k][j]; CloseST[j]=k; }	sum=D[w] + G[w][v] ; if (sum < D[v]) { D[v] = sum ; p[v]=w; }
构成结果图	所构造的连通网的权值之和最小	一定是源点到终点的两路径权值最短

负权最短路径问题

Dijkstra算法要求边的权值非负。事实上，一旦某些边的权值为负，那么Dijkstra算法就无法得出正确的最小路径长度。因为根据选取规则选取V访问时，得到Dv，但此时的路径可能没有经过顶点u，而weight(<u,v>)<0，如果Du+weight(<u,v>)<Dv，那么Dv就不是v的最小路径长度。



如图，D0=0,D0-2=1,D0-1=2,D0-2-3=4，但从S到V1还有其它路径：路径V0-V2-V3-V1的长度为-6；路径V0-V2-V3-V1-V2-V3-V1的长度为-10，如此下去，V1的最小路径长度是-∞。

出现这种情况的原因是图中出现了包括负权边的回路，我们称这种回路为负开销回路。

7.6 最短路径算法

- 其它最短路径问题及解法
  - 单目标最短路径问题：
    找出图中每个顶点*v*到某个指定结点*c*最短路径
    - 每边取反
  - 单结点对间最短路径问题：
    对于某对顶点*u*和*v*,找出*u*到*v*的一条最短路径
    - 以*u*为源点
  - 所有顶点间的最短路径问题：
    对图中每对顶点*u*和*v*，找出*u*到*v*的最短路径
    - 以每个顶点为源点
    - 直接用**Floyd**算法

7.6 最短路径算法

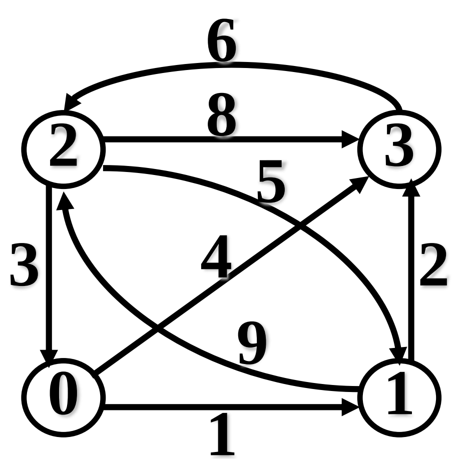
任意两个顶点之间的最短路径

- 问题描述：已知一个带权的有向图G=（V，E），对每一对顶点v_i，$v_j \in$ V，（$i \neq j$），要求：求出v_i与v_j之间的最短路径和最短路径长度。
- 限制条件：不允许有负长度的环路。
- Floyd算法的基本想法：动态规划算法
  - 如果v_i与v_j之间有有向边，则v_i与v_j之间有一条路径，但不一定是最短的；也许经过某些中间点会使路径长度更短。
  - 经过哪些中间点会使路径长度缩短呢?经过哪些中间点会使路径长度最短呢?
    - 只需尝试在原路径中间加入其它顶点作为中间顶点。
  - 如何尝试?
    - 系统地在原路径中间加入每个顶点，然后不断地调整当前路径(和路径长度)即可。

7.6 最短路径算法

- 示例：
  - <2,3>8
  - 考虑加入v_0： <2,0,3>7 <2,3>8 a[2][3]=a[2][0]+a[0][3] 调整
  - 考虑继续加入v_1： <2,0><0,1><1,3>=<2,0,1,3> a[2][3]=6 调整
  - 注意：①考虑v_0做中间点可能还会改变其它顶点间的距离：

    ②有时加入中间顶点后的路径比原路径长 保持



$$A=\begin{array}{cc} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} & \\ \begin{pmatrix} \infty & 1 & \infty & 4 \\ \infty & \infty & 9 & 2 \\ 3 & 5 & \infty & 8 \\ \infty & \infty & 6 & \infty \end{pmatrix} & \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \end{array} \end{array}$$

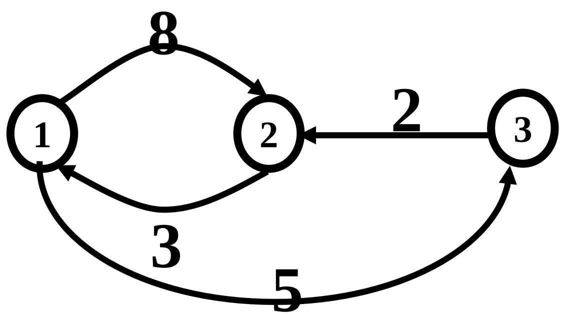
7.6 最短路径算法

- Floyd算法的基本思想：
  - 假设求顶点v_i到顶点v_j的最短路径。如果从 v_i 到 v_j 存在一条长度为C[*i*][*j*]的路径，该路径不一定是最短路径，尚需进行 *n* 次试探。
  - 首先考虑路径（v_i， v_0， v_j）是否存在。如果存在，则比较（v_i， v_j）和（v_i， v_0， v_j）的路径长度取长度较短者为从 v_i 到 v_j 的中间顶点的序号不大于0的最短路径。
  - 假设在路径上再增加一个顶点 v_1，也就是说，如果（v_i ,…， v_1)和（v_1,…， v_j）分别是当前找到的中间顶点的序号不大于0的最短路径,那么（v_i,…， v_1 ,…， v_j)就是有可能是从v_i到v_j的中间顶点的序号不大于1的最短路径。将它与已经得到的从v_i到v_j中间顶点序号不大于0的最短路径相比较，从中选出中间顶点的序号不大于1的最短路径，再增加一个顶点v_2，继续进行试探。
  - 一般情况下，若（v_i ,…， v_k）和（v_k， …， v_j）分别是从 v_i到v_k和从v_k到v_j的中间顶点序号不大于 k-1 的最短路径，则将（v_i ,…， v_k ,…， v_j)和已经得到的从 v_i到 v_j且中间顶点序号不大于k-1的最短路径相比较，其长度较短者便是从v_i到 v_j 的中间顶点的序号不大于 *k* 的最短路径。

- 假设已求出$A_{(k-1)}(i,\ j)$ $(1\le i,\ j\le n)$，怎样求出 $A_{(k)}(i, j)$？

  - 如果从顶点i到顶点j的最短路径不经过顶点k，则由$A_{(k)}(i,\ j)$的定义可知，从i到j的中间顶点序号不大于k的最短路径长度就是$A_{(k-1)}(i,\ j)$，即$A_{(k)}(i, j)=A_{(k-1)}(i, j)$。
  - 如果从顶点i到顶点j的最短路径经过顶点k，则这样的一条路径是由i到k和由k到j的两条路径所组成。由于$A(_{k-1})(i, k)+A(_{k-1})(k, j)<A(_{k-1})(i, j)$，则$A(_{k})(i, j)=A(_{k-1})(i, k)+A(_{k-1})(k, j)$。

对于n个顶点的图G，求任意一对顶点 Vi —> Vj 之间的最短路径可分为如下几个阶段：



#初始：不允许在其他顶点中转，最短路径是？
#1：若允许在 V_1 中转，最短路径是？
#2：若允许在 V_1、V_2 中转，最短路径是？
#3：若允许在 V_1、V_2、V_3 中转，最短路径是？ …
#n：若允许V_1、V_2……V_n 中转，最短路径是？

$$A_k[i][j] = \min(\ A_{k-1}[i][j]\ ,\ A_{k-1}[i][k] + A_{k-1}[k][j]\ )$$

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

路径矩阵P

A[2][3]= ∞,k=1
A[2][1]+A[1][3]=8

A[3][1]= ∞,k=2
A[3][2]+A[2][1]=5

A[1][2]= 8,k=3
A[1][3]+A[3][2]=7

	1	2	3
1	0	8	5
2	3	0	∞
3	∞	2	0

$A_0[i][j]$

	1	2	3
1	0	8	5
2	3	0	8
3	∞	2	0

$A_1[i][j]$

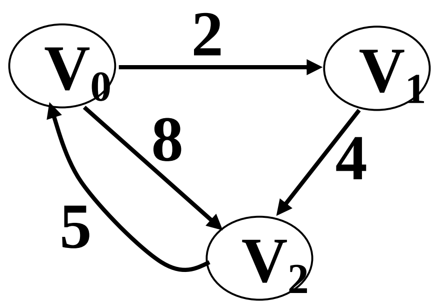
	1	2	3
1	0	8	5
2	3	0	8
3	5	2	0

$A_2[i][j]$

	1	2	3
1	0	7	5
2	3	0	8
3	5	2	0

$A_3[i][j]$

例：带权有向图及其邻接矩阵



$$\begin{pmatrix} 0 & 2 & 8 \\ \infty & 0 & 4 \\ 5 & \infty & 0 \end{pmatrix}$$

用Floyd算法求任意一对顶点间最短路径

步骤	初态	K=0	K=1	K=2
A	$\begin{pmatrix} 0 & 2 & 8 \\ \infty & 0 & 4 \\ 5 & \infty & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 2 & 8 \\ \infty & 0 & 4 \\ 5 & 7 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 2 & 6 \\ \infty & 0 & 4 \\ 5 & 7 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 2 & 6 \\ 9 & 0 & 4 \\ 5 & 7 & 0 \end{pmatrix}$
Path	$\begin{pmatrix} -1 & -1 & -1 \\ -1 & -1 & -1 \\ -1 & -1 & -1 \end{pmatrix}$	$\begin{pmatrix} -1 & -1 & -1 \\ -1 & -1 & -1 \\ -1 & 0 & -1 \end{pmatrix}$	$\begin{pmatrix} -1 & -1 & 1 \\ -1 & -1 & -1 \\ -1 & 0 & -1 \end{pmatrix}$	$\begin{pmatrix} -1 & -1 & 1 \\ 2 & -1 & -1 \\ -1 & 0 & -1 \end{pmatrix}$
S	{ }	{ 0 }	{ 0, 1 }	{ 0, 1, 2 }

用Floyd算法求任意一对顶点间最短路径

步骤	初态	K=0	K=1	K=2
A	$\begin{pmatrix} 0 & 2 & 8 \\ \infty & 0 & 4 \\ 5 & \infty & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 2 & 8 \\ \infty & 0 & 4 \\ 5 & 7 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 2 & 6 \\ \infty & 0 & 4 \\ 5 & 7 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 2 & 6 \\ 9 & 0 & 4 \\ 5 & 7 & 0 \end{pmatrix}$
Path	$\begin{pmatrix} -1 & -1 & -1 \\ -1 & -1 & -1 \\ -1 & -1 & -1 \end{pmatrix}$	$\begin{pmatrix} -1 & -1 & -1 \\ -1 & -1 & -1 \\ -1 & 0 & -1 \end{pmatrix}$	$\begin{pmatrix} -1 & -1 & 1 \\ -1 & -1 & -1 \\ -1 & 0 & -1 \end{pmatrix}$	$\begin{pmatrix} -1 & -1 & 1 \\ 2 & -1 & -1 \\ -1 & 0 & -1 \end{pmatrix}$
S	{ }	{ 0 }	{ 0, 1 }	{ 0, 1, 2 }

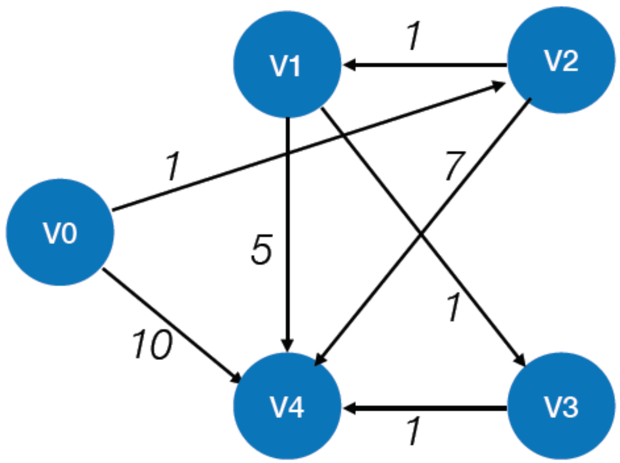
根据上述过程中Path[i][j]数组，得出：

V_0到V_1： 最短路径是{ 0, 1 }， 路径长度是2 ；

V_0到V_2： 最短路径是{ 0, 1, 2 }， 路径长度是6 ；

V_1到V_0： 最短路径是{ 1, 2, 0 }， 路径长度是9 ；

7.6 最短路径算法



path =

	V0	V1	V2	V3	V4
V0	-1	2	-1	2	3
V1	-1	-1	-1	-1	3
V2	-1	-1	-1	1	3
V3	-1	-1	-1	-1	-1
V4	-1	-1	-1	-1	-1

求V0到V4的最短路径？

A =

	V0	V1	V2	V3	V4
V0	0	2	1	3	4
V1	∞	0	∞	1	2
V2	∞	1	0	2	3
V3	∞	∞	∞	0	1
V4	∞	∞	∞	∞	0

7.6 最短路径算法

- **Floyd算法的数据结构**
  - 图的存储结构：
    - 带权的有向图采用邻接矩阵C[n][n]存储
  - 数组A[n][n]：
    - 存放在迭代过程中求得的最短路径长度。迭代公式：

$$\begin{cases} A_0[i][j]=C[i][j] \\ A_k[i][j]=\min\{ A_{k-1}[i][j], A_{k-1}[i][k]+A_{k-1}[k][j] \} \quad 0\le k\le n-1 \end{cases}$$

  - 数组P[n][n]：
    - 存放从 v_i 到 v_j求得的最短路径。初始时, P[i][j]= -1

7.6 最短路径算法

Floyd算法的实现

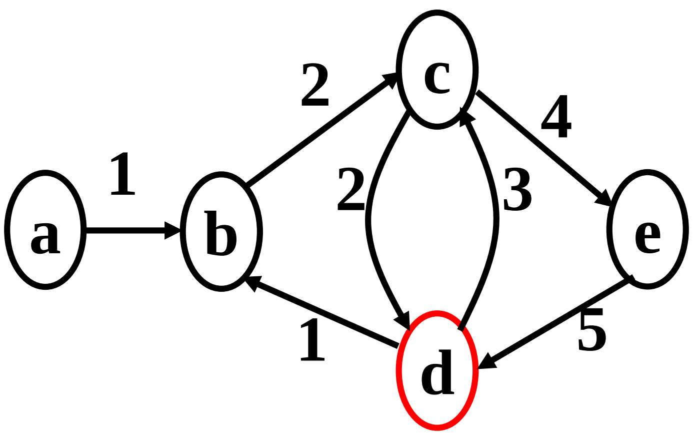
```
void   Floyd( costtype A[][], costtype C[][], int P[][], int n)
{   for ( i = 0; i < n; i++ )
        for ( j = 0; j< n; j++ )
        {   A[i][j] = C[i][j];
            P[i][j] = -1;    }
    for ( k = 0; k < n; k++ )
        for ( i = 0; i < n; i++ )
            for ( j = 0; j < n; j++ )
                if ( A[i][k] + A[k][j] < A[i][j] )
                  {  A[i][j] = A[i][k] + A[k][j] ;
                     P[i][j] = k;}
}
```

/* 时间复杂度：$O(n^3)$ */

7.6 最短路径算法

Floyd算法的应用----求有向图的中心点

- 顶点的偏心度：
  - 设G=(V, E)是一个带权有向图，D[i][j]表示从 i 到 j的最短距离。对任意一个顶点k, E(k) = max{ d[i][k] | i ∈V }称作顶点 k 的偏心度。
- 图G 的中心点：
  - 称具有最小偏心度的顶点为图G的中心点。



最短路径矩阵D

$$\begin{array}{c|c} & \begin{array}{ccccc} a & b & c & d & e \end{array} \\ \begin{array}{c} a \\ b \\ c \\ d \\ e \end{array} & \begin{pmatrix} 0 & 1 & 3 & 5 & 7 \\ \infty & 0 & 2 & 4 & 6 \\ \infty & 3 & 0 & 2 & 4 \\ \infty & 1 & 3 & 0 & 7 \\ \infty & 6 & 8 & 5 & 0 \end{pmatrix} \end{array}$$

顶点	偏心度
a	∞
b	6
c	8
d	5
e	7

本章小结

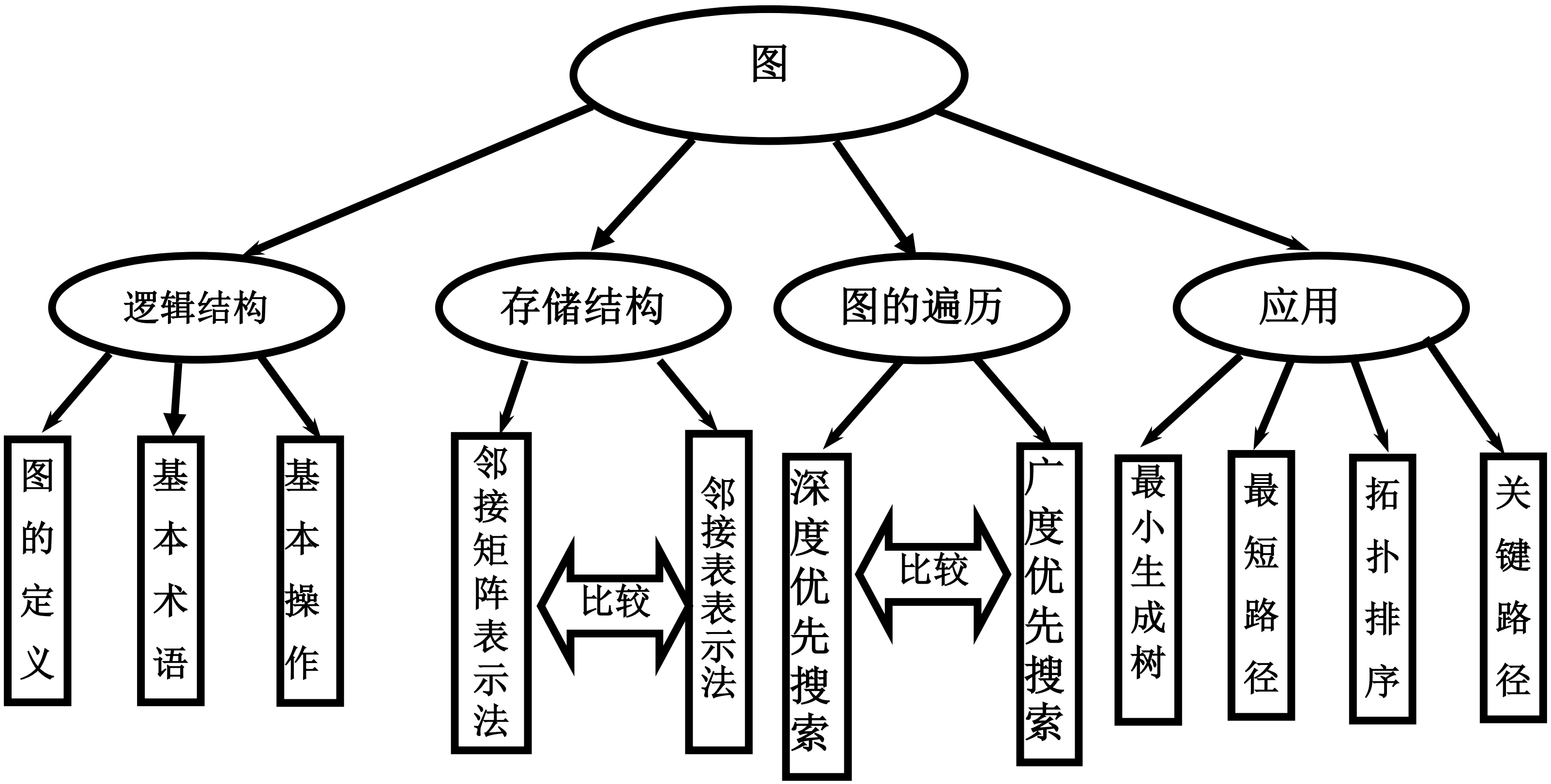


图
逻辑结构
存储结构
图的遍历
应用
图的定义
基本术语
基本操作
邻接矩阵表示法
比较
邻接表表示法
深度优先搜索
比较
广度优先搜索
最小生成树
最短路径
拓扑排序
关键路径