第19章

动态规划

本章学习内容

动态规划方法基本思想

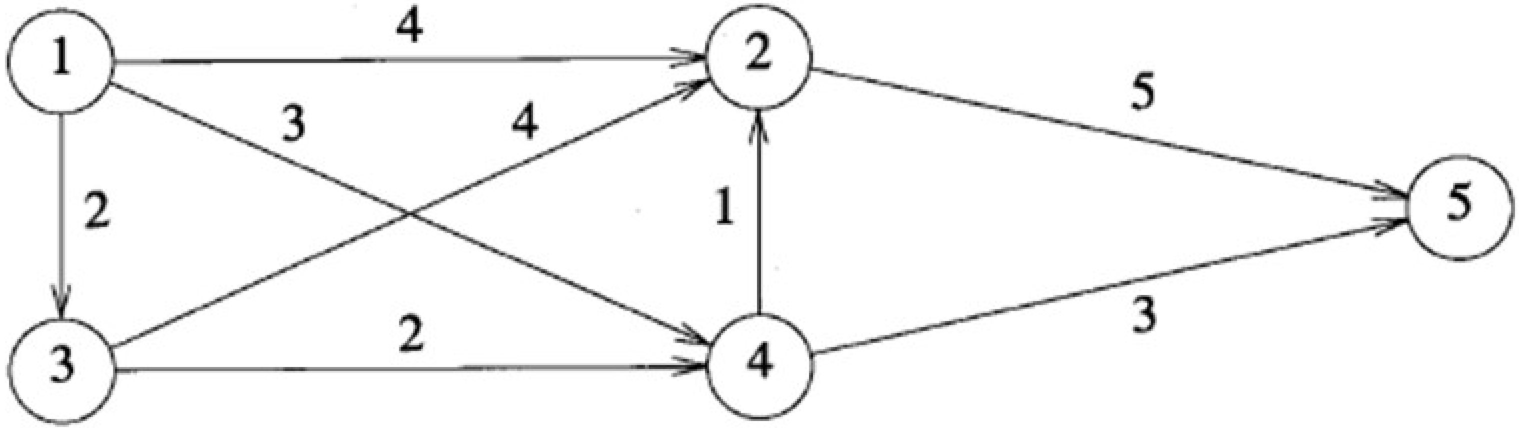
所有顶点对最短路径问题

动态规划方法基本思想

- 将一个问题的解看作是为一系列决策的结果
- 在动态规划中，要确定每个最优决策序列中是否包含最优决策子序列。
  - 用子问题的最优解来构造原问题的最优解

例：最短路径问题

- 找一条从源顶点s 到达目的顶点d 的最短路径



- 求解该问题,需要决策路径中经过的顶点：<u>$s,\ldots,d$</u>
- 首先决策顶点s的下一个顶点v，再决策顶点v 到顶点d 的最短路径
- 顶点s 到顶点d 的最短路径：一定包含顶点v 到顶点d 的最短路径 <u>$v,\ldots,d$</u>

0/1背包问题

- 在0/1背包问题中，需对容量为**c** 的背包进行装载。从**n** 个物品中选取装入背包的物品，每件物品**i** 的重量为w_i，价值为p_i。
- 问题求解：确定x_1，$\cdots x_i \cdots$，x_n的值
  - $x_i = 1$　表示物品i 装入背包中
  - $x_i = 0$　表示物品i 不装入背包

0/1背包问题的动态规划思想

- 假设按$i = 1$，2，…，n 的次序来确定x_i 的值
  - $x_1 = 0$，则问题转变为相对于物品2，3，…，n，容量为c 的背包问题；
  - $x_1 = 1$，问题就变为相对于物品2，3，…，n，容量为$c\text{-}w_1$ 的背包问题

➢ 在第一次决策之后，剩下的问题便是考虑背包容量为r 时的决策

$$r \in \{c,\ c - w_1\}$$

➢ 不管x_1 是0或是1，[x_2，…，x_n] 必须是第一次决策之后的一个最优解

0/1背包问题的动态规划思想

- $f(i, y)$：表示剩余容量为y，剩余物品为i，$i+1$，…，n 时最优解的值，即：

$$f(n,y)=\begin{cases} p_n & y \geqslant w_n \\ 0 & 0 \leqslant y < w_n \end{cases}$$

$$f(i,y)=\begin{cases} \max\{f(i+1,y), f(i+1,y-w_i)+p_i\} & y \geqslant w_i \\ f(i+1,y) & 0 \leqslant y < w_i \end{cases}$$

- 0/1背包问题：求解$f(1,c)$
- 最优决策序列中包含一个最优子序列

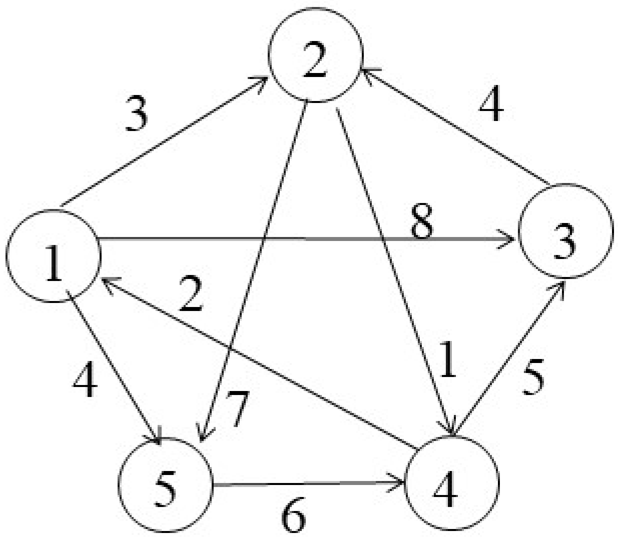
⇒ 建立动态规划递归方程

19.2.3 所有顶点对最短路径问题

- 问题描述：
  - 在n 个顶点的有向图G中，寻找每一对顶点之间的最短路径，即对于每对顶点 (i, j)，需要寻找从 i 到 j 的最短路径及从 j 到 i 的最短路径，对于无向图，这两条路径是一条。
- 对一个n 个顶点的有向图，需寻找$p = n(n-1)$ 条最短路径。

所有顶点对最短路径问题

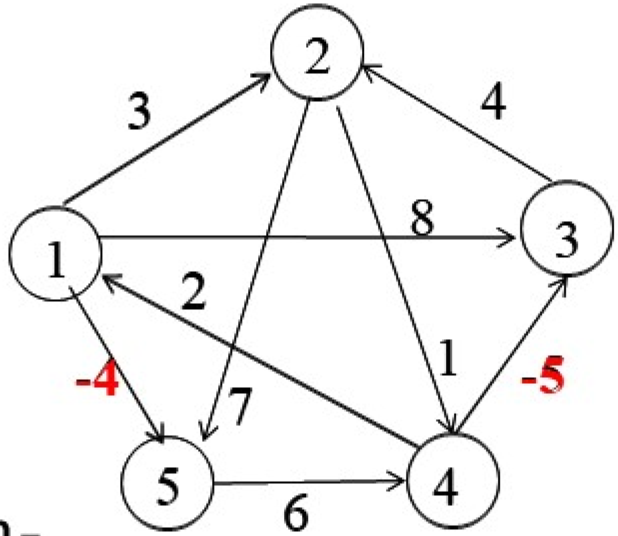
- 思考：这一问题用能否上一章的贪婪算法求解?
- 单源点最短路径Dijkstra算法
  - 边上的权值>=0



- 如何求所有顶点对之间的最短路径?
- 使用Dijkstra算法 n 次, 每次用1个顶点作为源点
- 时间复杂性： $O(n^3)$.

所有顶点对最短路径问题

- 对于下图，能否使用Dijkstra算法，求所有顶点对之间的最短路径？



- 含有权值为负值的边？
- 不含有长度为负数的环路

- 单源最短路径问题——Bellman-Ford(贝尔曼-福特)算法

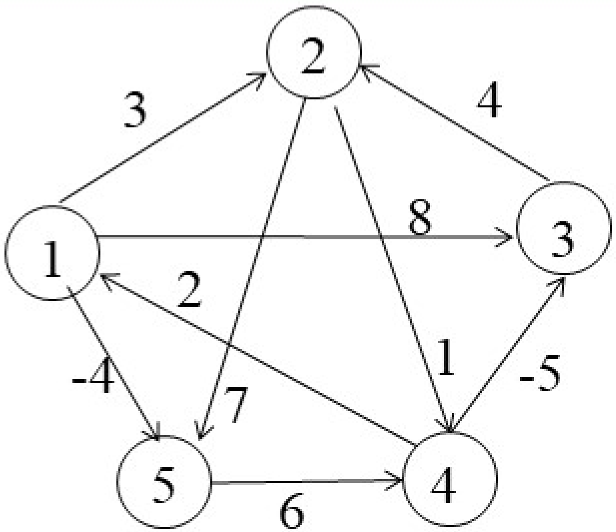
- **Floyd(弗洛伊德)算法**——所有顶点对之间的最短路径

Floyd最短路径算法

- 设图G中n 个顶点的编号为1到n。
- 图G使用邻接矩阵存储， a 是图G的耗费邻接矩阵
- **c(*i,j,k*)**或**c$^{(k)}$ (*i,j*)**: 从顶点i 到顶点j ，允许经过的中间顶点都不大于k的路径中，最短路径的长度。

中间顶点取自集合$\{1,2,\ldots, k\}$

c(*i*, *j*, *k*)示例

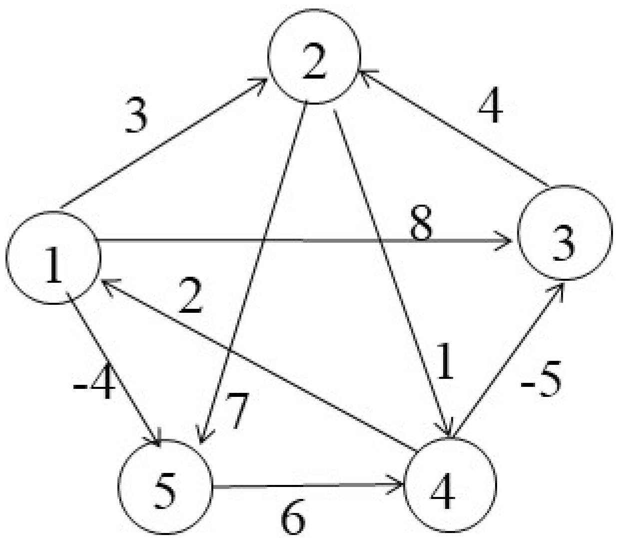


- $c(1,2,0)=3$ $c(1,2,3)=3$ $c(1,2,4)=3$
- $c(3,4,0)=\infty$ $c(3,4,1)=\infty$ $c(3,4,2)=5$

$c^{(0)}$

- $c(i,j,0)=\begin{cases} \text{边 } (i,j)\text{的长度} & (i,j) \in E \\ 0 & i=j \\ \infty \quad (\text{noEdge}) & \text{其它} \end{cases}$

- $c(i,j,0) = a[i][j]$ a 是耗费邻接矩阵



$$c^{(0)} = \begin{bmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix}$$

$c(*,*,0)$

所有顶点对之间的最短路径的长度？

- 顶点i 到顶点j 最短路径的长度？
  - 路径中允许经过的中间顶点$\le n$

- 顶点i 到顶点j 的最短路径的长度: **c(*i*,*j*,*n*)**
- 所有顶点对之间的最短路径长度：**c**$^{(n)}$ 或 **c(*,*,*n*)**

$$
\begin{array}{c|ccc|}
 & \cdots & j & \cdots & n \\
\vdots & & & & \\
i & & \mathbf{c}(i,j,n) & & \\
\vdots & & & & \\
n & & & &
\end{array}
$$

所有顶点对最短路径的长度？

$$\begin{array}{c|c} & \cdots \quad j \quad \cdots \quad n \\ \hline \begin{array}{c}\vdots\\ i\\ \vdots\\ n\end{array} & \begin{bmatrix} & & & \\ & \mathbf{c}(\boldsymbol{i},\boldsymbol{j},\mathbf{0}) & & \\ & & & \\ & & & \end{bmatrix} \end{array} \Rightarrow \begin{array}{c|c} & \cdots \quad j \quad \cdots \quad n \\ \hline \begin{array}{c}\vdots\\ i\\ \vdots\\ n\end{array} & \begin{bmatrix} & & & \\ & \mathbf{c}(\boldsymbol{i},\boldsymbol{j},\boldsymbol{n}) & & \\ & & & \\ & & & \end{bmatrix} \end{array}$$

$c^{(0)}$ $\qquad$ $c^{(n)}$

- $c^{(0)} \Rightarrow c^{(n)}$ ？
- $c^{(k\text{-}1)} \Rightarrow c^{(k)}, \quad k > 0$ ？

$c^{(k-1)} \Rightarrow c^{(k)}, \quad k>0$ ？

$$\begin{array}{cc} & \begin{matrix} \cdots & j & \cdots & n \end{matrix} \\ \begin{matrix} \vdots \\ i \\ \vdots \\ n \end{matrix} & \begin{bmatrix} & & & \\ & c(i,j,k\text{-}1) & & \\ & & & \\ & & & \end{bmatrix} \end{array} \Longrightarrow \begin{array}{cc} & \begin{matrix} 1\ldots & j & \cdots & n \end{matrix} \\ \begin{matrix} \vdots \\ i \\ \vdots \\ n \end{matrix} & \begin{bmatrix} & & & \\ & c(i,j,k) & & \\ & & & \\ & & & \end{bmatrix} \end{array}$$

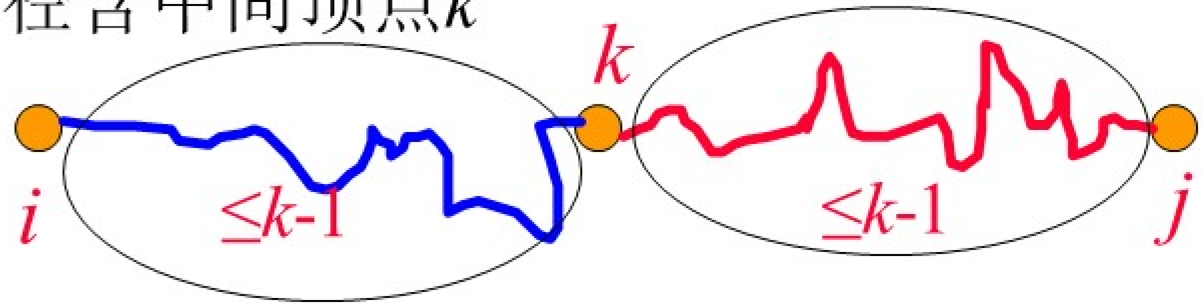
$c^{(k-1)}$ $\qquad\qquad$ $c^{(k)}$

- $c(i,j,k)$: 从i到j, 中间顶点取自集合$\{1,2,\ldots, k\}$,最短路径的长度;
- $c(i,j,k-1)$: 从i到j, 中间顶点取自集合$\{1,2,\ldots, k-1\}$, 最短路径的长度;

$c^{(k-1)} \Rightarrow c^{(k)}, \quad k>0$

- $c(i,j,k)$ 有两种可能：
  - 1.该路径不含中间顶点k， 该路径长度为$c(i, j, k-1)$

  - 2. 该路径含中间顶点k



路径长度为$c(i, k, k-1) + c(k, j, k-1)$

$c^{(k-1)} \Rightarrow c^{(k)}, \quad k>0$

- 结合以上两种情况，$c(i, j, k)$ 取两者中的最小值

$$\Rightarrow c(i,j,k) = \min\{\ c(i,j,k-1),\ c(i,k,k-1) + c(k,j,k-1)\}$$

- 动态规划递归公式

$$
\begin{array}{c|c}
 & \cdots \quad j \quad \cdots \quad n \\
\hline
\begin{array}{c} \vdots \\ i \\ \vdots \\ n \end{array} & \left[\begin{array}{c} \\ c(i,j,k-1) \\ \\ \\ \end{array}\right]
\end{array}
\Longrightarrow
\begin{array}{c|c}
 & \cdots \quad j \quad \cdots \quad n \\
\hline
\begin{array}{c} \vdots \\ \mathrm{i} \\ \vdots \\ \mathrm{n} \end{array} & \left[\begin{array}{c} \\ c(i,j,k) \\ \\ \\ \end{array}\right]
\end{array}
$$

$c^{(k-1)}$ $\qquad\qquad$ $c^{(k)}$

$c^{(0)} \Rightarrow c^{(n)}$?

使用$c(i,j,k) = \min\{ c(i,j,k\text{-}1),$
$c(i,k,k\text{-}1) + c(k,j,k\text{-}1)\}$

- 按 $k = 1, 2, 3, \ldots, n$的顺序计算$c^{(k)}$

$$c^{(0)} \Rightarrow c^{(1)} \Rightarrow c^{(2)} \quad \ldots\ldots \Rightarrow c^{(n)}$$

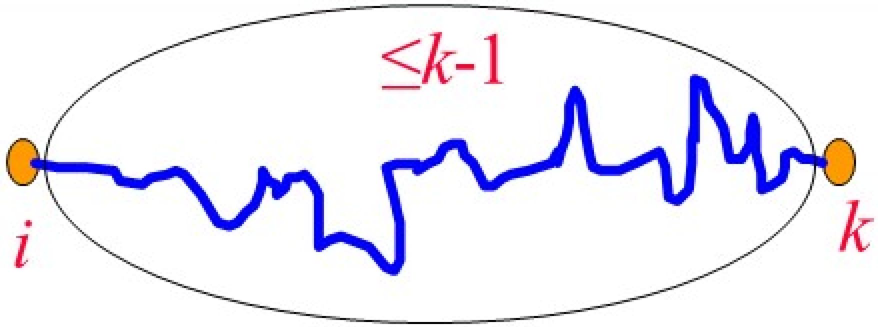
$$\begin{pmatrix} \\ c(i,j,0) \\ \\ \end{pmatrix} \rightarrow \begin{pmatrix} \\ c(i,j,1) \\ \\ \end{pmatrix} \rightarrow \begin{pmatrix} \\ c(i,j,2) \\ \\ \end{pmatrix} \rightarrow \cdots\cdots \rightarrow \begin{pmatrix} \\ c(i,j,n) \\ \\ \end{pmatrix}$$

如果用递归方法求解上式，则计算最终结果的复杂性：$\Theta(n^2 2^n)$

Floyd算法的伪代码

```
//寻找最短路径的长度
//初始化c(i, j, 0)
for(int i=1;i<=n;i++)
for (int j=1;j<=n;j++)
    c(i,j,0)=a(i,j);//a是耗费邻接矩阵
//计算c(i,j,k)(1≤k≤n)
for (int k = 1; k <= n; k++)
  for (int i = 1; i <= n; i++)
    for (int j = 1; j <= n; j++)
      if (c(i,k,k-1) + c(k,j,k-1)< c(i,j,k-1) )
                c(i,j,k) = c(i,k,k-1) + c(k,j,k-1)
       else     c(i,j,k) = c(i,j,k-1)
```

- $1 \le i \le n$
  - ✓ $c(i,k,k\text{-}1)=c(i,k,k)$
  - ✓ $c(k,i,k\text{-}1)=c(k,i,k)$



- $c(i,k,k\text{-}1)+c(k,j,k\text{-}1)<c(i,j,k\text{-}1)$:

➡ $c(i,j,k) = c(i,k,k\text{-}1)+c(k,j,k\text{-}1)$

$= c(i,k,k)+c(k,j,k)$

- $c(i,k,k\text{-}1)+c(k,j,k\text{-}1) \ge c(i,j,k\text{-}1)$:

➡ $c(i,j,k) = c(i,j,k\text{-}1)$;

- 用$c(i,j)$ 代替$c(i,j,k)$,最后所得的$c(i,j)$ 之值等于$c(i,j,n)$ 值

细化的Floyd算法伪代码

```
//寻找最短路径的长度
//初始化c(i,j) = c(i， j， 0)
for(int i=1; i<=n; i++)
for (int j=1; j<=n;j++)
   c(i,j)=a(i,j);//a是耗费邻接矩阵
//计算c(i,j)=c(i,j,k)(1≤k≤n)
for (int k = 1; k <= n; k++)
   for (int i = 1; i <= n; i++)
      for (int j = 1; j <= n; j++)
         if (c(i,k) + c(k,j)< c(i,j) )
                  c(i,j) = c(i,k) + c(k,j)
         ~~else  c(i,j) = c(i,j)~~
```

如何表示最短路径?

- kay(i,j) : 从i 到j 的最短路径中最大的k 值(编号最大的中间顶点)。
- 初始, kay(i,j)=0 (最短路径中没有中间顶点).

```
for (int k = 1; k <= n; k++)
   for (int i = 1; i <= n; i++)
      for (int j = 1; j <= n; j++)
         if (c(i,j) > c(i,k) + c(k,j))
          {kay(i,j) = k; c(i,j) = c(i,k) + c(k,j);}
```

AdjacencyWDigraph::Allpairs 1/2

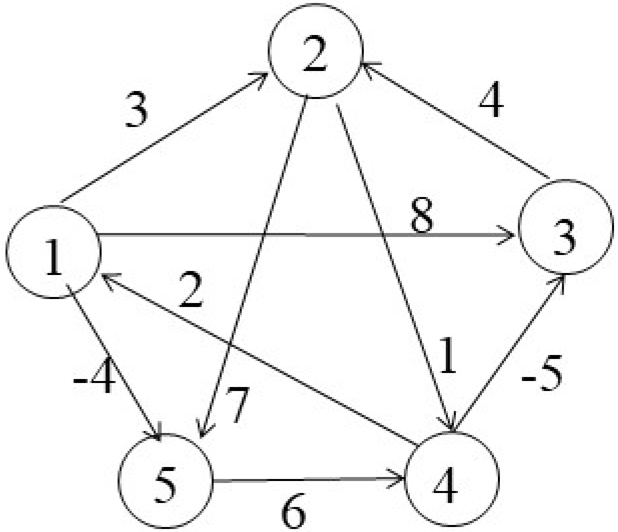
```
template<class T>
void  Allpairs(T **c, int **kay)
{//所有点对的最短路径;对于所有i和j，计算c[i][j]和kay[i][j]
//初始化c[i][j]=c(i，j，0)
for (int i=1;i<=n;i++)
for (int j=1;j<=n;j++){
   c[i][j]=a[i][j];
   kay[i][j]=0;}
for  (i=1;i<=n;i++)
   c[i][i]=0;
```

AdjacencyWDigraph::Allpairs 1/2

```
//计算c[i][j]=c(i,j,k)
for (int k=1;k<=n;k++)
for (int i=1;i<=n;i++)
for (int j=1;j<=n;j++)
    {if  (c[i][k]!=NoEdge && c[k][j]!=NoEdge &&
         (c[i][j]==NoEdge || c[i][j] > c[i][k] + c[k][j]))
         {c[i][j]= c[i][k] + c[k][j];
          kay[i][j]=k;
         }
    }
}
```

- 时间复杂性： $\Theta(n^3)$.

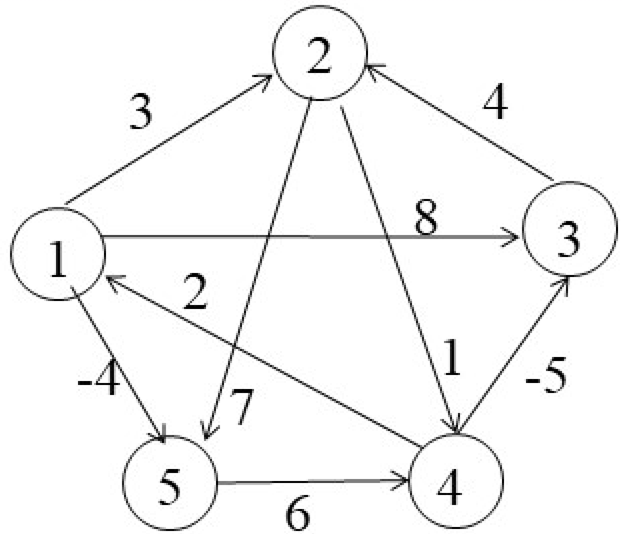
Floyd算法示例



初始：c(*,*,0)或$c^{(0)}$

$$\begin{bmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix}$$

Floyd算法示例



$c^{(n)}$

$$\begin{bmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & \color{red}{3} \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{bmatrix}$$

$kay^{(n)}$

$$\begin{bmatrix} 0 & 5 & 5 & 5 & 0 \\ 4 & 0 & 4 & 0 & 4 \\ 4 & 0 & 0 & 2 & 4 \\ 0 & 3 & 0 & 0 & 1 \\ 4 & 4 & 4 & 0 & 0 \end{bmatrix}$$

kay 矩阵→最短路径

$$\begin{bmatrix} 0 & 5 & 5 & 5 & 0 \\ 4 & 0 & 4 & 0 & 4 \\ 4 & 0 & 0 & 2 & 4 \\ 0 & 3 & 0 & 0 & 1 \\ 4 & 4 & 4 & 0 & 0 \end{bmatrix}$$

- 3到5的最短路径？
- kay(3,5) = 4

$3 \longrightarrow 4 \longrightarrow 5$

- kay(3,4) = 2

$3 \longrightarrow 2 \longrightarrow 4 \longrightarrow 5$

- kay(3,2) = 0

$3 \quad 2 \longrightarrow 4 \longrightarrow 5$

- kay(2,4) = 0

$3 \quad 2 \quad 4 \longrightarrow 5$

- kay(4,5) = 1

$3 \quad 2 \quad 4 \longrightarrow 1 \longrightarrow 5$

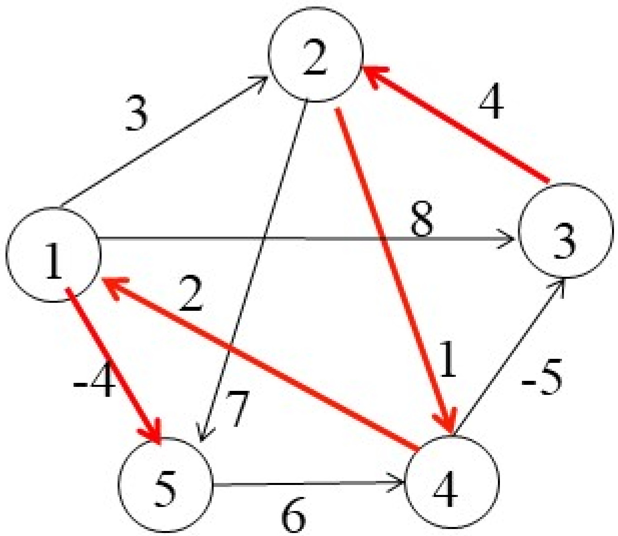
- kay(4,1) = 0

$3 \quad 2 \quad 4 \quad 1 \longrightarrow 5$

- kay(1,5) = 0

$3 \quad 2 \quad 4 \quad 1 \quad 5$

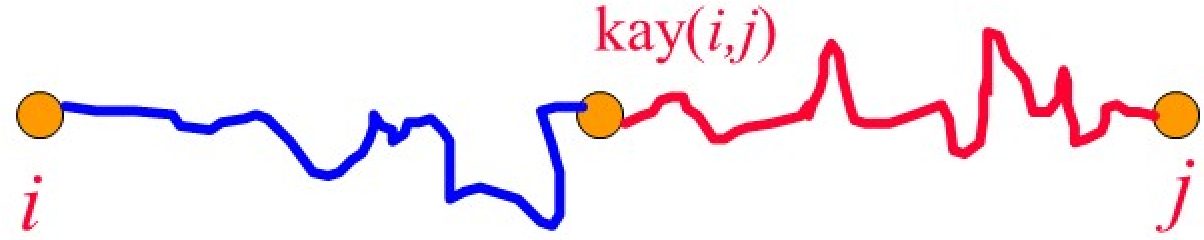
kay 矩阵→最短路径



3到 5的最短路径 : 3 2 4 1 5

路径长度3.

输出 *i* 到 *j* 的最短路径



输出i到j的最短路径中，**i**之后的顶点序列

如果 （ kay(i,j)=0 ） 输出j

否则 输出i到kay(i,j)的最短路径中，i之后的顶点序列

输出kay(i,j)到j的最短路径中， kay(i,j)之后的顶点序列

输出 *i* 到 *j* 的最短路径实现

```
template<class T>
void outputPath(T **c, int **kay, T noEdge, int i, int j)
{// 输出从i 到j的最短路径
    if (c[i][j] = = noEdge) {
        cout << "There is no path from " << i << " to " << j <<
        endl;
        return;}
    cout << "The path is" << endl;
    cout << i << ' ';
    //输出i到j的最短路径中，i之后的顶点序列
    outputPath(kay, i, j);
    cout << endl;
}
```

输出 *i* 到 *j* 的最短路径实现

```
void outputPath(int **kay, int i, int j)
{//输出i 到j 的路径的实际代码
  // 不输出路径上的第一个顶点 (i)
if (i = = j) return;
if (kay[i][j] = = 0) //路径上没有中间顶点
   cout<<j << ' ';
else {// kay[i][j]是路径上的中间顶点
    outputPath(kay,  i,  kay[i][j]);
    outputPath(kay, kay[i][j], j);}
}
```