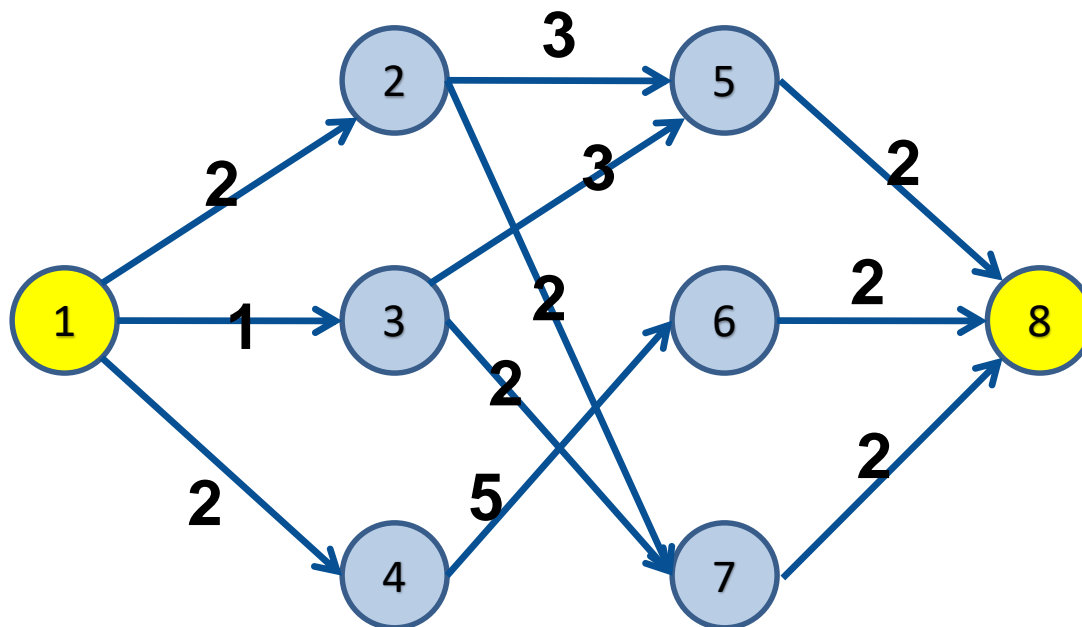


## 5.图的最短路径问题

带权图：边长有“距离”（有时候可能为负值）  
最短路径就是指连接两点的这些路径中最短的一条。  
最短的路径长度称为最短距离。



求1到8的最短距离？


## ➤任意两点间最短路:

–**Floyd**（弗洛伊德）算法：（邻接矩阵）

单源最短路:

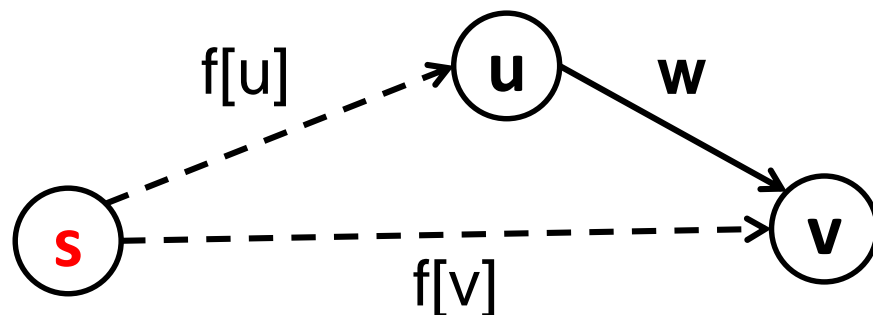
–**Dijkstra**（迪杰斯特拉）算法（邻接矩阵）

–**SPFA**算法（最短路快速算法）（邻接表）

- 
- **1 各种算法的特点及其使用范围**
  - **2 时间复杂度**
  - **3 各种最短路径算法的实现**

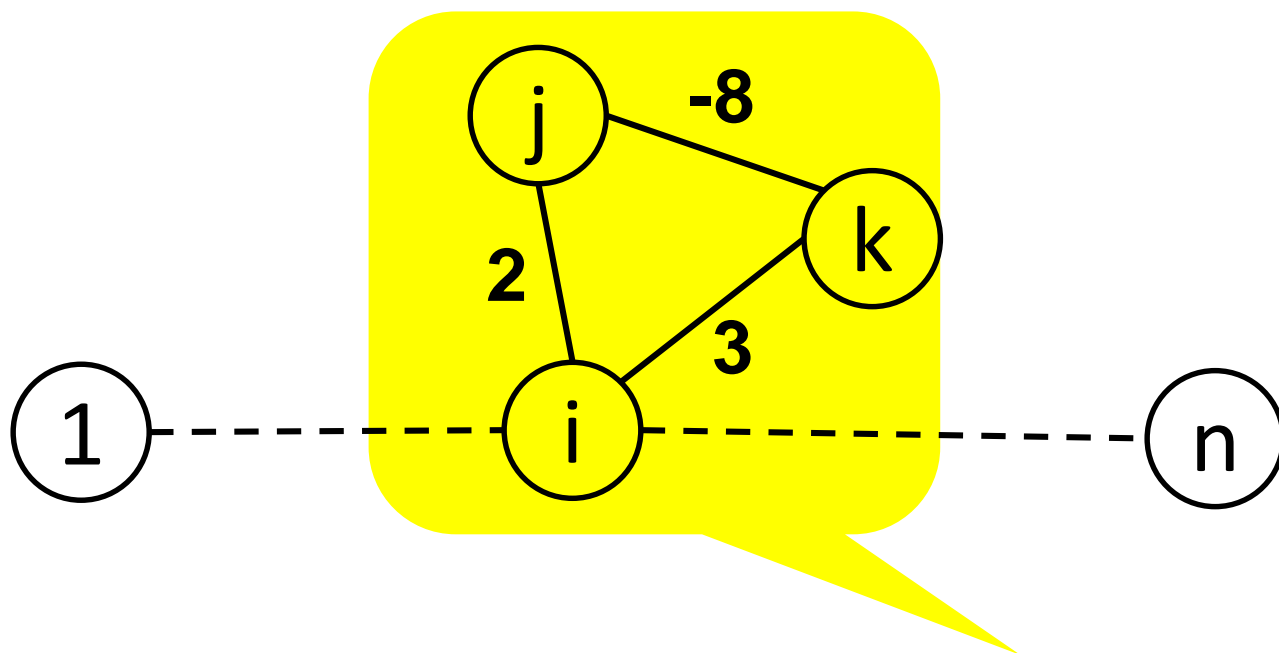
# 松弛技术（三角不等式）

$f[i]$ : 以  $s$  为起点,  $i$  为终点的最短距离



if  $(f[u] + w < f[v])$   $f[v] = f[u] + w;$

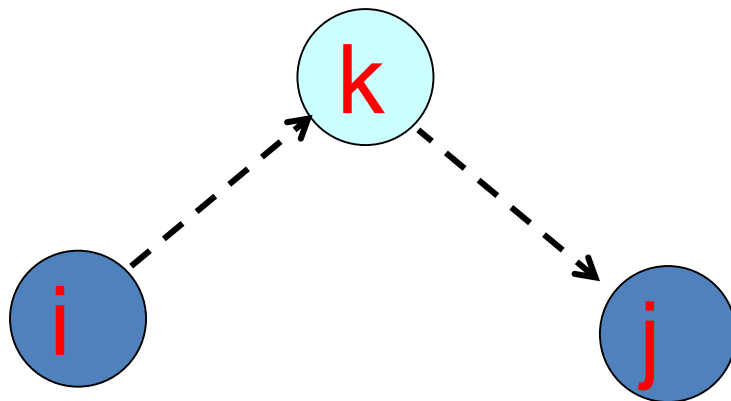
# 不考虑有**负权回路**的图的最短路



长度为-3的负权回路

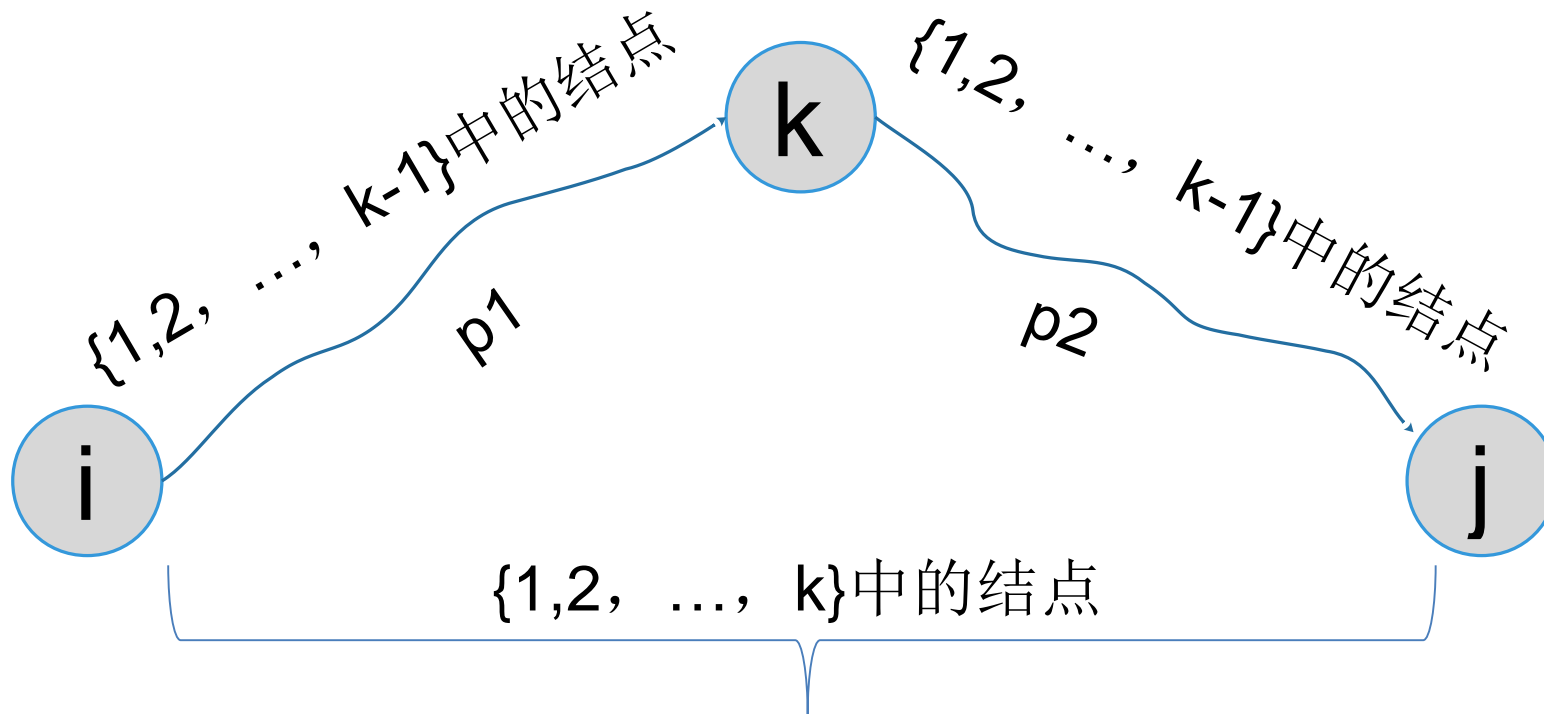
# 一 . floyd算法 ( 计算每一对顶点间的最短路径 )

最简单的最短路径算法，可以计算图中任意两点间*i*到*j*的最短路径 $d[i][j]$ 。  
Floyed的时间复杂度是 $O(N^3)$ ，适用于出现负边权的情况（无负权回路）。



if  $(d[i][k] + d[k][j] < d[i][j])$   $d[i][j] = d[i][k] + d[k][j]$

或写成:  $d[i][j] = \min(d[i][j], d[i][k] + d[k][j])$



考虑  $i$  到  $j$  的路径  $p$  的中间结点均取自于  $\{1, 2, \dots, k\}$  :

如果  $k$  不是路径  $p$  的中间结点, 则  $p$  的所有中间结点都在集合  $\{1, 2, \dots, k-1\}$

如果  $k$  是路径  $p$  的中间结点, 则  $p$  分解为:  $i \rightarrow k \rightarrow j$

$$d[i][j][k] = a[i][j] \quad (k=0)$$

$$d[i][j][k] = \min(d[i][j][k-1], d[i][k][k-1] + d[k][j][k-1]) \quad (k \geq 1)$$

去掉一维  $k$  直接更新:  $d[i][j] = \min(d[i][j], d[i][k] + d[k][j])$



# 弗洛伊德算法：牢记下列4行代码：

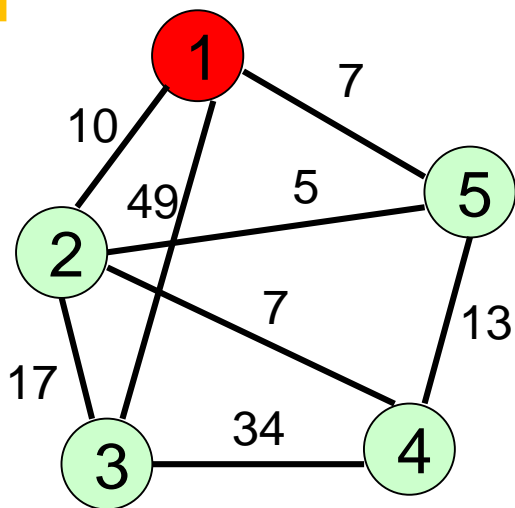
邻接矩阵初始化：对角线为0；无边的无穷大  
使用之前初始化：

$d[i][i]=0$

$d[i][j]=a[i][j]$   $i$ 到 $j$ 有边

=无穷大 $INF=1000000000$ ；不能是 $2000000000$ ；  
= $0x3f3f3f3f$

1. `for(int  $k=1$ ;  $k \leq n$ ;  $k++$ )`
2.     `for(int  $i=1$ ;  $i \leq n$ ;  $i++$ )`
3.         `for(int  $j=1$ ;  $j \leq n$ ;  $j++$ )`
4.              `$d[i][j]=\min(d[i][j], d[i][k]+d[k][j])$ ;`



## $d[i][j]$ 的变化

$k=0$

0	10	49	INF	7
10	0	17	7	5
49	17	0	34	INF
INF	7	34	0	13
7	5	INF	13	0

$k=1$

0	10	49	INF	7
10	0	17	7	5
49	17	0	34	56
INF	7	34	0	13
7	5	56	13	0

$k=2$

0	10	27	17	7
10	0	17	7	5
27	17	0	24	22
17	7	24	0	12
7	5	22	12	0

$k=3$

0	10	27	17	7
10	0	17	7	5
27	17	0	24	22
17	7	24	0	12
7	5	22	12	0

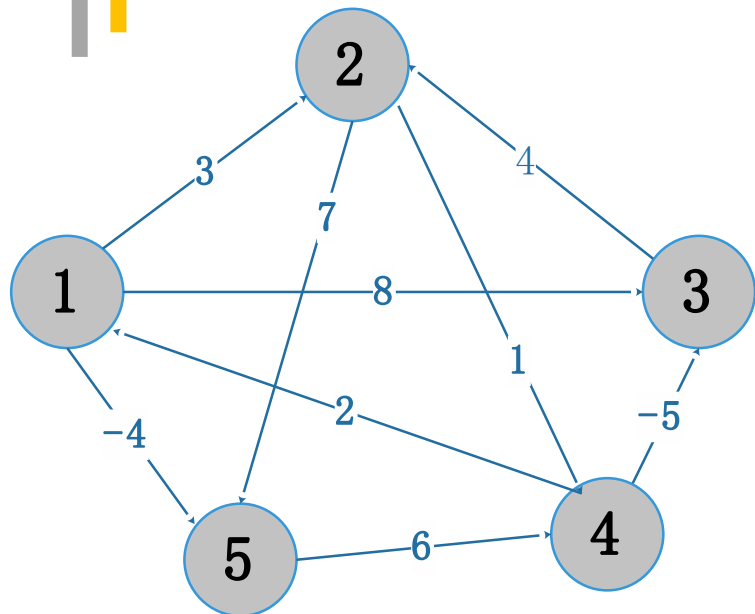
$k=4$

0	10	27	17	7
10	0	17	7	5
27	17	0	24	22
17	7	24	0	12
7	5	22	12	0

$k=5$

0	10	27	17	7
10	0	17	7	5
27	17	0	24	22
17	7	24	0	12
7	5	22	12	0

5 8  
 1 2 10  
 1 3 49  
 1 5 7  
 2 3 17  
 2 4 7  
 2 5 5  
 3 4 34  
 4 5 13



## $d[i][j]$ 的变化

5 9	0	3	8	$\infty$	-4
1 2 3	$\infty$	0	$\infty$	1	7
1 3 8	$\infty$	4	0	$\infty$	$\infty$
1 5 -4	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
2 4 1	2	$\infty$	-5	0	$\infty$
2 5 7	$\infty$	$\infty$	$\infty$	6	0
3 2 4					
4 1 2					
4 3 -5					
5 4 6					

k=0

0	3	8	INF	-4
INF	0	INF	1	7
INF	4	0	INF	INF
2	INF	-5	0	INF
INF	INF	INF	6	0

k=1

0	3	8	INF	-4
INF	0	INF	1	7
INF	4	0	INF	INF
2	5	-5	0	-2
INF	INF	INF	6	0

k=2

0	3	8	4	-4
INF	0	INF	1	7
INF	4	0	5	11
2	5	-5	0	-2
INF	INF	INF	6	0

k=3

0	3	8	4	-4
INF	0	INF	1	7
INF	4	0	5	11
2	-1	-5	0	-2
INF	INF	INF	6	0

k=4

0	3	-1	4	-4
3	0	-4	1	-1
7	4	0	5	3
2	-1	-5	0	-2
8	5	1	6	0

k=5

0	1	-3	2	-4
3	0	-4	1	-1
7	4	0	5	3
2	-1	-5	0	-2
8	5	1	6	0

说明：

1. 记住循环变量的顺序，外层是k
2. 如果边长a后面没用处，直接使用a(或开始就把边存到d中)即可，无需另外的 $d \leftarrow a$

时间复杂度： $O(n^3)$

邻接矩阵存储

可以负权，但不能有负权回路

## 例1：主席的居住城市

### 【问题描述】

OI国共有 $n$ 个城市，任意两个城市都直接或间接连通，每两个直接相连的城市之间都有一双向公路。OI国的CHEN主席要选择其中一个城市居住，由于他到每个城市考察的概率是相等的，所以他选择的城市到其它所有城市的最短距离的和应该最小。

现在，给出城市间的道路，请帮助CHEN主席确定居住的城市。

### 【输入】

第一行， $n$ ，表示城市的个数。

以下是 $n*n$ 的矩阵，描述城市之间的距离，-1表示对应城市间没有道路。

### 【输出】

第一行：CHEN居住城市的编号（如果有多个城市同时最小，输出编号最小的城市）。

第二行：最小距离和。

已知： $n \leq 100$ ；城市间的距离不超过100。

dist.in	dist.out
5	2
-1 31 -1 72 -1	234
31 -1 30 -1 70	
-1 30 -1 76 -1	
72 -1 76 -1 40	
-1 70 -1 40 -1	

## 训练题目：

- 1: 1342 最短路径问题
- 2: 1343 牛的旅行 （课后）
- 3: 1378 **最短路径(shoph)** （课后）
- 4: 1381 **城市路(Dijkstra)**

## 二 . Dijkstra ( 迪杰斯特拉 ) 算法

计算某一顶点到其它所有顶点的最短路径  
(单源最短路径问题)

开始点 (源点) :  $s$

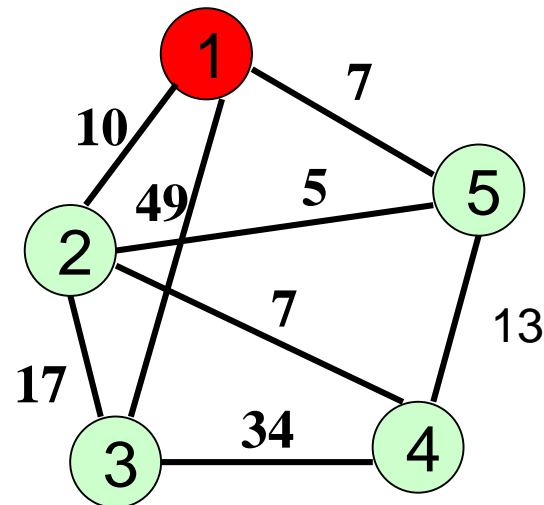
$d[i]$ : 顶点  $i$  到  $s$  的最短距离。

邻接矩阵  $a[][]$  存储边长

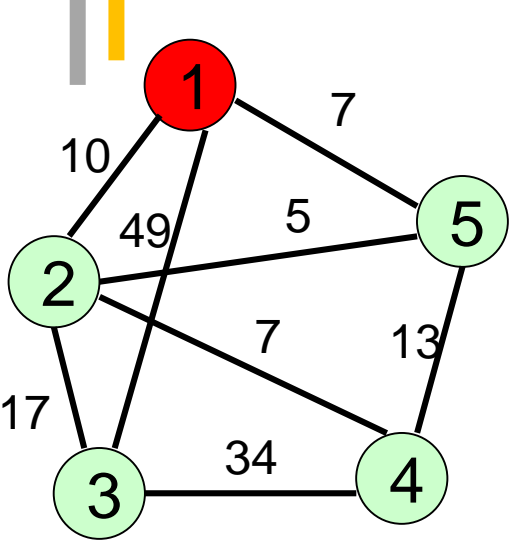
初始:

$d[i]$  = 无边无穷大 1000000000

$d[s] = 0$ ;





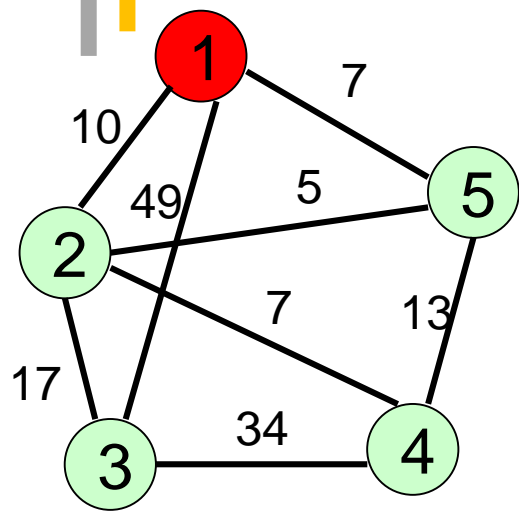


5	8	
1		
1	2	10
1	3	49
1	5	7
2	3	17
2	4	7
2	5	5
3	4	34
4	5	13

顶点	1	2	3	4	5
vis[i]	1	0	0	0	0
D[i]	0	10	49	$\infty$	7
Path[i]	1	1,2	1,3		1,5

顶点	1	2	3	4	5
vis[i]	1	0	0	0	1
D[i]	0	10	49	20	7
Path[i]	1	1,2	1,3	1,5,4	1,5

顶点	1	2	3	4	5
vis[i]	1	1	0	0	1
D[i]	0	10	27	17	7
Path[i]	1	1,2	1,2,3	1, 2,4	1,5

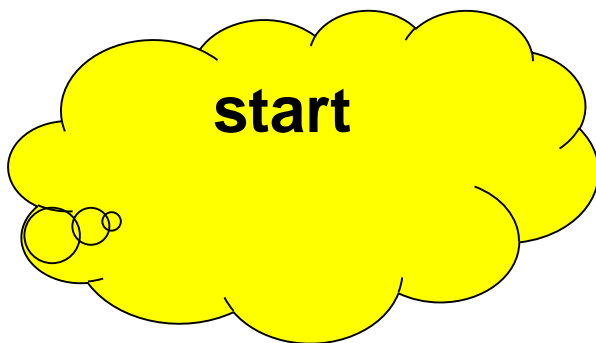


5 8  
 1  
 1 2 10  
 1 3 49  
 1 5 7  
 2 3 17  
 2 4 7  
 2 5 5  
 3 4 34  
 4 5 13

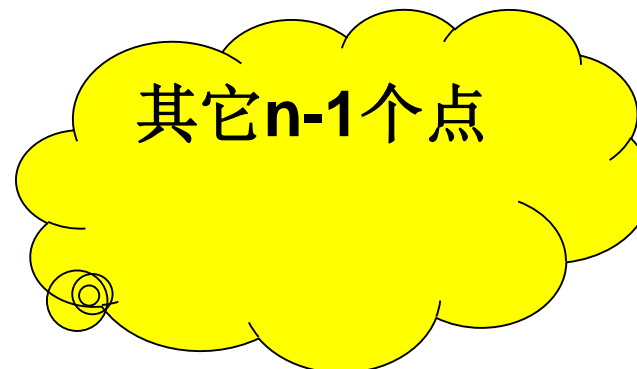
顶点	1	2	3	4	5
vis[i]	1	1	0	1	1
D[i]	0	10	27	17	7
Path[i]	1	1,2	1,2,3	1, 2,4	1,5

顶点	1	2	3	4	5
vis[i]	1	1	1	1	1
D[i]	0	10	27	17	7
Path[i]	1	1,2	1,2,3	1, 2,4	1,5

集合1：已求点



集合2：未求点




- 1、在集合2中找一个到start距离最近的顶点k :  $\min\{d[k]\}$
- 2、把顶点k加到集合1中，同时修改集合2 中的剩余顶点j的 $d[j]$ 是否经过k后变短。如果变短修改 $d[j]$   
$$d[j] = \min(d[j], d[k] + a[k][j])$$
- 3、重复1，直至集合2空为止。

## 方法1: 四行

```
void dijkstra(int s){
    memset(vis,0,sizeof(vis));
    for(int i=1;i<=n;i++) d[i]=a[s][i];
    d[s]=0;
    vis[s]=1;
    for(int i=1;i<n;i++){
        int x=-1,mn=INF;
        for(int j=1;j<=n;j++) if(!vis[j]&& d[j]<mn) mn=d[x=j];
        vis[x]=1;
        for(int j=1;j<=n;j++) if(!vis[j]) d[j]=min(d[j],d[x]+a[x][j]);
    }
}
```

## 方法2:

```
void dijkstra(int s){
    memset(vis,0,sizeof(vis));
    for(int i=1;i<=n;i++) d[i]=INF;
    d[s]=0;
    for(int i=1;i<=n;i++){
        int x=-1,mn=INF;
        for(int j=1;j<=n;j++) if(!vis[j]&& d[j]<mn) mn=d[x=j];
        vis[x]=1; //第一次是起点s
        for(int j=1;j<=n;j++) if(!vis[j]) d[j]=min(d[j],d[x]+a[x][j]);
    }
}
```



时间复杂度:  $O(n^2)$   
不能有负权的边



dijkstra训练题目：

- 1: **1381城市路(Dijkstra)**
- 2: 1379热浪(heatwv)
- 3: 1376信使(msner)
- 4: 1344 **【例4-4】最小花费**
- 5: 1377: 乘车路线



floyed  
dijkstra  
邻接矩阵存储



## ◆ 优先队列（堆）优化的dijkstra算法

1. `#include<queue>`
2. `struct Heapnode{`
3.     `int d,u;`
4.     `bool operator < (const Heapnode &x)const{return d>x.d;}`
5. `};`小顶堆
6. `priority_queue<Heapnode>q;`
7. `q.push((Heapnode){0,s});`
8. `Heapnode x=q.top();q.pop();`

```

void dijkstra(int s) {
    memset(vis,0,sizeof(vis));
    priority_queue<Heapnode>q;
    for(int i=1;i<=n;i++) d[i]=INF;
    d[s]=0;
    q.push((Heapnode){0,s});
    while(!q.empty()) {
        Heapnode x=q.top();q.pop();
        int u=x.u;
        if(vis[u]) continue;
        vis[u]=1;
        for(int i=g[u];i>0;i=e[i].next) {
            int v=e[i].v;
            if(d[u]+e[i].w<d[v]) {
                d[v]=d[u]+e[i].w;
                q.push((Heapnode){d[v],v});
            }
        }
    }
}

```

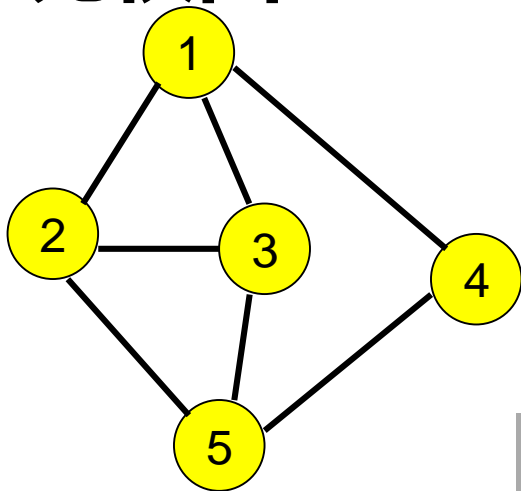
## 补充：图的邻接表存储

- 邻接表是图的一种链式存储结构，在邻接表中，对于图中的每一个顶点 $u$ 都建立一个单向链表，链表中的每一结点用来描述顶点 $u$ 的邻接点。

邻接表的实现有两种方法：**vector**和**数组模拟**。

## Vector实现邻接表

无权图：



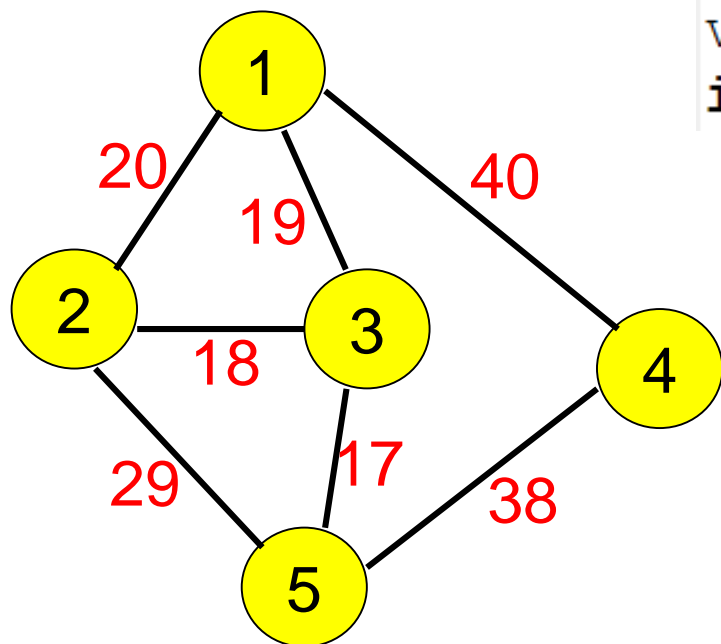
5 7  
1 2  
1 3  
1 4  
2 3  
2 5  
3 5  
4 5

```
1:2 3 4
2:1 3 5
3:1 2 5
4:1 5
5:2 3 4
```

```
vector<int>g[maxn];  
int n,m;  
int main() {  
    cin>>n;  
    cin>>m;  
    for(int i=0;i<m;i++){  
        int u,v;  
        cin>>u>>v;  
        g[u].push_back(v);  
        g[v].push_back(u);  
    }  
    for(int i=1;i<=n;i++){  
        int k=g[i].size();  
        cout<<i<<": ";  
        for(int j=0;j<k;j++){  
            cout<<g[i][j]<<" ";  
        }  
        cout<<endl;  
    }  
    return 0;  
}
```

# 带权图：

```
const int maxn=10010;//最大顶点数量
struct Edge{int from;int to;int w;};//边记录
vector<Edge>e;           //所有的边
vector<int>g[maxn];      //每个顶点的邻接点的边的编号
int n,m;
```



```
1: 0(1 2 20)->2(1 3 19)->4(1 4 40)
2: 1(2 1 20)->6(2 3 18)->8(2 5 29)
3: 3(3 1 19)->7(3 2 18)->10(3 5 17)
4: 5(4 1 40)->12(4 5 38)
5: 9(5 2 29)->11(5 3 17)->13(5 4 38)
```

g[i]

```
1: 0->2->4
2: 1->6->8
3: 3->7->10
4: 5->12
5: 9->11->13
```

e[i]

```
0: 1 2 20
1: 2 1 20
2: 1 3 19
3: 3 1 19
4: 1 4 40
5: 4 1 40
6: 2 3 18
7: 3 2 18
8: 2 5 29
9: 5 2 29
10: 3 5 17
11: 5 3 17
12: 4 5 38
13: 5 4 38
```

```
5 7
1 2 20
1 3 19
1 4 40
2 3 18
2 5 29
3 5 17
4 5 38
```

## 读入数据建立邻接表

```
cin>>n;
cin>>m;
for(int i=0;i<m;i++){
    int u,v,w;
    cin>>u>>v>>w;
    e.push_back((Edge){u,v,w});
    e.push_back((Edge){v,u,w});
    g[u].push_back(2*i);
    g[v].push_back(2*i+1);
}
```

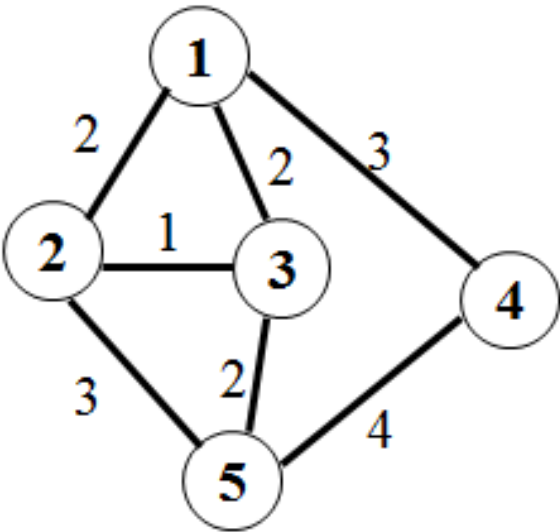
- 访问顶点 $u$ 的邻接点方法:
- `int k=g[u].size();`
- `for(int j=0;j<k;j++){`
- `int p=g[u][j];;`  
      `int v=e[p].to;//u的邻接点`  
      `int w=e[p].w; //<u,v>边长`
- `}`

## ◆ 数组模拟链表

```
➤ struct Edge{  
    int u,v,w,next;  
➤ };  
➤ Edge e[maxe];
```



顶点i	g[i]
1	13
2	10
3	8
4	14
5	12



x	y	w
2	5	3
1	3	2
3	5	2
2	3	1
1	2	2
4	5	4
1	4	3

边	i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
起点	e[i].u	2	5	1	3	3	5	2	3	1	2	4	5	1	4
终点	e[i].v	5	2	3	1	5	3	3	2	2	1	5	4	4	1
边长	e[i].w	3	3	2	2	2	2	1	1	2	2	4	4	3	3
下一条边	e[i].next	0	0	0	0	4	2	1	5	3	7	0	6	9	11

```
1. for(int i=1;i<=m;i++) {  
2.     int u, v, w;  
3.     scanf("%d%d%d", &u, &v, &w) ;  
4.     e[i]=(Edge) {u, v, w, g[u]} ;g[u]=i;  
5.     e[i+m]=(Edge) {v, u, w, g[v]} ;g[v]=i+m;  
6. }
```

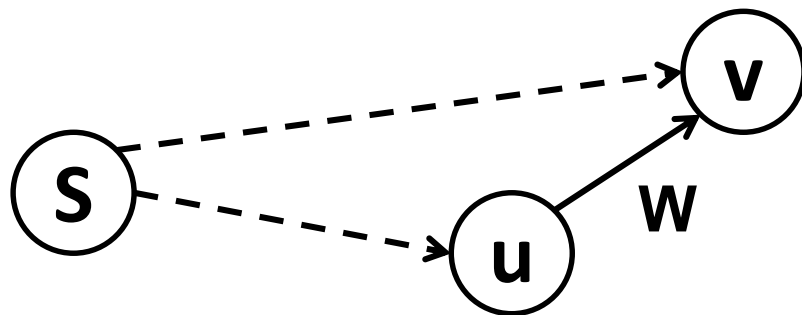
## ◆访问结点 u 的邻接点:

```
1. for(int i=g[u];i>0;i=e[i].next) {  
2.     int v=e[i].v;  
3.     ...  
4. }
```

## ◆附. Bellman-Ford算法

- 同Dijkstra算法相比，Bellman-Ford算法能够在图中**存在负权边**的情况下解决单源最短路问题。
- 其基本思路为：
- 如果两点之间有最短路，那么最多经过图中所有顶点各一次（如果经过某个顶点两次，那么我们走出了一个环。如果环的权值为非负，显然不划算；如果权值为负，则最短路不存在），也就是说，这条**路最多有 $n-1$ 条边**。根据最短路的最优子结构性质，最多包含 $k$ 条边的最短路由最多包含 $k-1$ 条边的最短路“加一条边”来求得，因此通过反复松弛每条边 **$n-1$ 遍**，即可求得源点到所有点的最短路。

## ➤ 边表存图 ( $u[i], v[i], w[i]$ )



```
for(int i=1;i<=n;i++) d[i]=INF;
d[s]=0
for(int i=1;i<=n-1;i++)
    for(int j=1;j<=m;j++) {
        x=u[i]; y=v[i];
        if d[x]<INF d[y]=min(d[y], d[x]+w[i]);
    }
```

## 有负环的判定：

- 如果还能松弛 则有负环
- `for (int i=1 ;i<=m;i++)`
- `if (d[u[i]]+w[i]<d[v[i]] )` 有负环

# Bellman-Ford算法：

单源最短路

可以有负权

能判断是否有负环

时间复杂度为 $O(nm)$

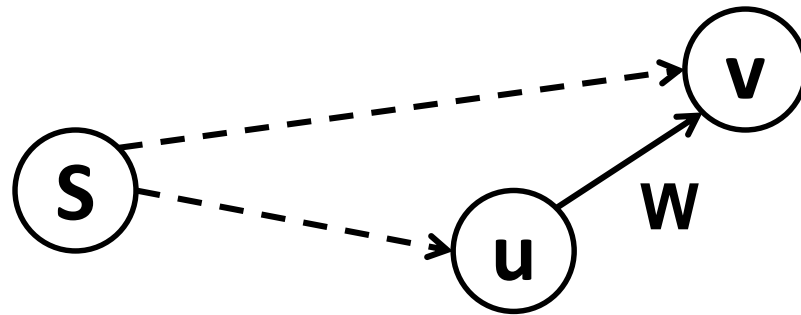
### ◆ 3 . SPFA算法（单源）

- SPFA算法的全称是：Shortest Path Faster Algorithm
- SPFA实际上是Bellman-Ford基础上的优化
- 西南交通大学段凡丁于1994年发表的
- 给定的图存在负权边，这时类似Dijkstra等算法便没有了用武之地；而Bellman-Ford算法的复杂度又过高。
- SPFA算法目前应用最广的最短路径算法。
- 期望的时间复杂度  $O(km)$  ( $k < 2, m$  是边数)
- 邻接表存储边的关系



## 实现方法：

- 建立一个队列，初始时队列里只有起始点。
- $d[i]$ 记录起始点 $s$ 到 $i$ 所有点的最短路径。
- 然后执行松弛操作，用依次用队列里有的点 $u$ 去更新所有后继结点 $v$ 的最短路，如果 $v$ 被更新成功且不在队列中则把该点加入到队列最后。重复执行直到队列为空。结点可能被多次被更新，可以多次进队列。



If  $d[u] + w < d[v]$  then  $d[v] = d[u] + w$

- 1. Spfa算法中，可以有负边。
- 如图中无负环，存在最短路，路径上的顶点个数不会超过 $n$ 。
- 所以每个顶点进入队列的次数不会超过 $N$ 。
- 2. Spfa的时间：
  - 在 SPFA 算法中，如果每个顶点都入队列一遍，则将扫描所有向图中每条边一次且仅一次，没有重复，其时间复杂度为  $O(m)$ ；如果每个顶点平均入队列  $k$  次，则 SPFA 算法的时间复杂度为  $O(km)$ 。通常的情况， $k$  为 2 左右。
- 3. **SPFA** 算法中判断负权值回路的方法
  - 在 SPFA 算法中，如果一个顶点入队列的次数超过  $n$ ，则表示有向网中存在负权值回路。如：昆虫洞(Wormholes, poj3259)

- 用 SPFA 算法判断是否存在负权值回路的原理是：如果存在负权值回路，那么从源点到某个顶点的最短路径可以无限缩短，某些顶点入队列将超过  $N$  次（ $N$  为顶点个数）。因此，只需在 SPFA 算法中统计每个顶点入队列的次数，在取出队列头顶点时，都判断该顶点入队列的次数是否已经超过  $N$  次，如果是，说明存在负权值回路，则 SPFA 算法不需要再进行下去了。另外，如果存在负权值回路，在退出 SPFA 算法时队列可能不为空，所以在进行下一个测试前清空队列。
- `While(!q.empty())q.pop();`

```

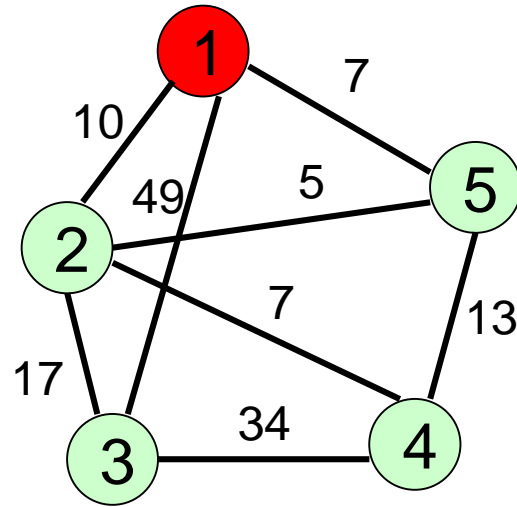
void spfa(int s){
    memset(vis,0,sizeof(vis));
    memset(d,0x3f,sizeof(d));
    d[s]=0;
    vis[s]=1;
    q[0]=s;
    int head=0,tail=1;
    while(head!=tail){
        int u=q[(head++)%maxn];
        vis[u]=0;
        int k=g[u].size();
        for(int i=0;i<k;i++){
            int p=g[u][i];
            int v=e[p].to,w=e[p].w;
            if(d[v]>d[u]+w){
                d[v]=d[u]+w;
                if(vis[v]==0){
                    q[(tail++)%maxn]=v;
                    vis[v]=1;
                }
            }
        }
    }
}

```

数组模拟

```
void spfa(int s){
    queue<int>q;
    memset(inq,0,sizeof(inq));
    for(int i=1;i<=n;i++) d[i]=INF;
    d[s]=0;
    q.push(s);inq[s]=1;
    while(!q.empty()){
        int u=q.front();q.pop();
        inq[u]=0;
        for(int i=g[u];i>0;i=e[i].next){
            int v=e[i].v,w=e[i].w;
            if(d[u]+w<d[v]){
                d[v]=d[u]+w;
                if(!inq[v]){
                    q.push(v);
                    inq[v]=1;
                }
            }
        }
    }
}
```

}



5 8  
1  
1 2 10  
1 3 49  
1 5 7  
2 3 17  
2 4 7  
2 5 5  
3 4 34  
4 5 13




训练题目：

1: 1382 [最短路\(Spfa\)](#)

2: 1380 [分糖果\(candy\)](#)

3: 1345 香甜的黄油



## ◆应用:

- 1. 奶牛派对 (Silver Cow party poj3268)
- 2. 昆虫洞(Wormholes poj3259)