L16：最短路 II

吉林大学计算机学院
谷方明
fmgu2002@sina.com

这是什么算法？

```
for(k=1; k<=n ; k++ )
    for( i =1 ; i <= n ; i ++)
        for( j=1 ; j<=n ;j++ )
            A[i][j] = min( A[i][j], A[i][k]+A[k][j] );
```

学习目标

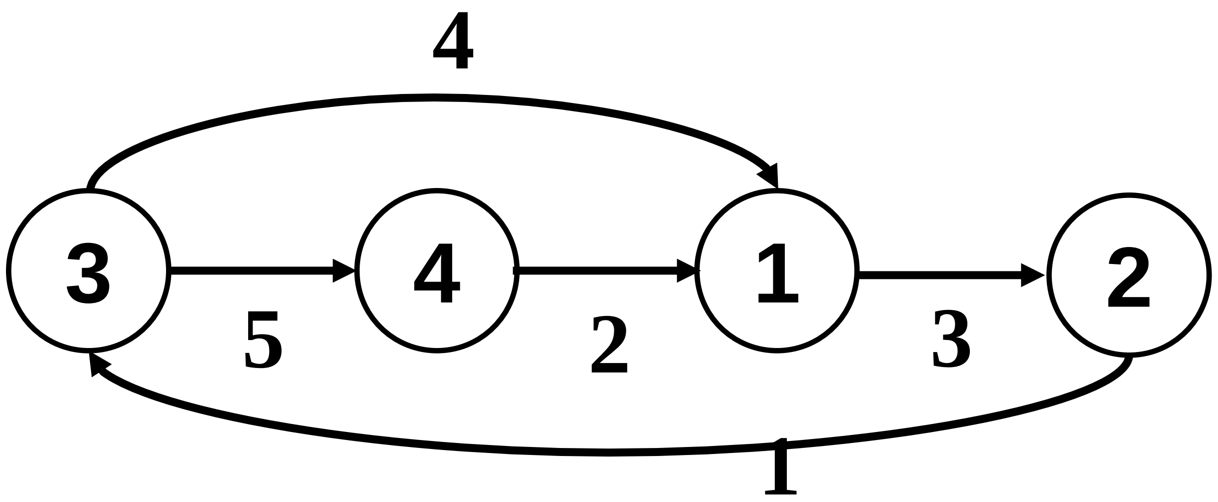
- 掌握Floyd算法
- 掌握图的传递闭包
- 了解Johnson算法
- 了解M最短路

任意两点间的最短路径问题

- 任意两点间最短路问题，也称为多源最短路问题。即对图中的的每一对顶点 $v_i \neq v_j$，求v_i与v_j间的最短路径（最短路径长度）。

- 显然，最容易的想法就是：依次把每个顶点作为源点，执行 n 次单源最短路算法。

枚举法



- 直接枚举所有路径
  - ✓ 计算量大
- 按长度枚举
  - ✓ 处理复杂；路径有重复计算，效率也不高
- 路径拼接
  - ✓ 唯一标识：标号最大点处拼接
  - ✓ 设中间点标号集合： $S_0=\{\ \}, S_1=\{V_1\}, S_2=\{V_1, V_2\},\ldots, S_n=\{V_1, V_2,\ldots, V_n\}$

Floyd算法

设*edge*[n][n]为图的邻接矩阵；

定义初始矩阵$A^{(0)}[i][j] = edge[i][j]$；

1 求$A^{(1)}$,即从v_i到v_j经由顶点可以是$\{v_1\}$的最短路径；

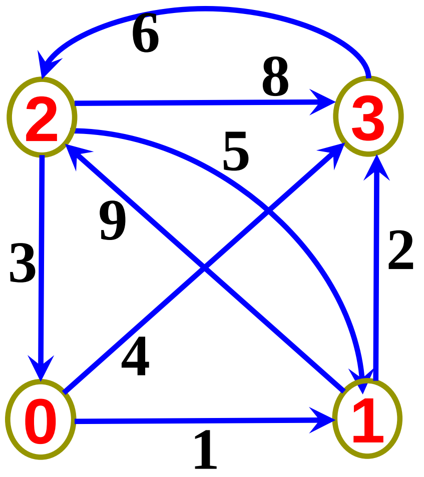
2 求$A^{(2)}$,即从v_i到v_j经由顶点可以是$\{v_1, v_2\}$的最短路径；

…

n 求$A^{(n)}$,即从v_i到v_j经由顶点可以是$\{v_1, v_2, \ldots, v_n\}$的最短路径长度；

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

$A^{(k-1)}[i][k] + A^{(k-1)}[k][j] \}, k = 1,2,\ldots, n$



$$
\begin{array}{c|cccc}
 & 0 & 1 & 2 & 3 \\
\hline
0 & 0 & 1 & \infty & 4 \\
1 & \infty & 0 & 9 & 2 \\
2 & 3 & 5 & 0 & 8 \\
3 & \infty & \infty & 6 & 0
\end{array}
$$

从$A^{(3)}$知，顶点 1 到 0 的最短路径长度为$a[1][0]=11$，其最短路径：
path[1][0]=2，表示顶点2→顶点0；
path[1][2]=3，表示顶点3→顶点2；
path[1][3]=1，表示顶点1→顶点3；从顶点 1 到顶点 0 最短路径为：
<1, 3>，<3, 2>，<2, 0>
（见$Path^{(3)}$中第1行，第0、2、3列）

	$A^{(-1)}$				$A^{(0)}$				$A^{(1)}$				$A^{(2)}$				$A^{(3)}$			
	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
0	0	1	∞	4	0	1	∞	4	0	1	10	3	0	1	10	3	0	1	9	3
1	∞	0	9	2	∞	0	9	2	∞	0	9	2	12	0	9	2	11	0	8	2
2	3	5	0	8	3	4	0	7	3	4	0	6	3	4	0	6	3	4	0	6
3	∞	∞	6	0	∞	∞	6	0	∞	∞	6	0	9	10	6	0	9	10	6	0

	$Path^{(-1)}$				$Path^{(0)}$				$Path^{(1)}$				$Path^{(2)}$				$Path^{(3)}$			
	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
0	−1	0	−1	0	−1	0	−1	0	−1	0	1	1	−1	0	1	1	−1	0	3	1
1	−1	−1	1	1	−1	−1	1	1	−1	−1	1	1	2	−1	1	1	2	−1	3	1
2	2	2	−1	2	2	0	−1	0	2	0	−1	1	2	0	−1	1	2	0	−1	1
3	−1	−1	3	−1	−1	−1	3	−1	−1	−1	3	−1	2	2	3	−1	2	0	3	−1

Floyd算法的正确性

- 最优子结构（最优性原理）：最短路径中的任意一段子路径都是最短的；
- 递推：最短路径能通过最短路径拼接得到，即

$$A^{(k)}[i][j] = \min \{ A^{(k-1)}[i][j], \quad A^{(k-1)}[i][k] + A^{(k-1)}[k][j] \}, k = 1,2,\ldots, n$$

$A^{(0)}[i][j]$ = 邻接矩阵

Folyd算法需要的空间

- 当以k为拼接点时，矩阵的第k行和第k列的数值是不会改变的，相当于坐标；其它行和其它列的值如果改变，都是通过第k行和第k列的数值相加得到。
- 因此，A的计算从前到后都可以使用一个矩阵。

Floyd算法描述

算法 Floyd ( )

/* 求每对顶点间的最短路径，其中edge[n][n]表示有n个顶点的图的邻接矩阵；A[i][j]表示顶点Vi至Vj的最短路径长度；*/

F1[初始化]

```
for( i =1 ; i <= n ; i ++)
  for( j=1 ; j<=n ;j++ )
        A[i][j]  = edge[i][j] ;
```

F2[求图中任意两顶点的最短路径]

```
for(k=1; k<=n ; k++ )
   for( i =1 ; i <= n ; i ++)
        for( j=1 ; j<=n ;j++ )
            if(A[i][k]+A[k][j]<A[i][j]))
                A[i][j] = A[i][k]+A[k][j];
```

path[i][j]

- **path[i][j]表示相应路径上顶点j的前一个顶点的序号；**
  - 初始化： path[i][j] = i <i,j> ∈E ; otherwise -1
  - 递　推： path[i][j] = path[k][j] ;
- **path[i][j]表示相应路径上顶点i的后一个顶点的序号；**
- **path[i][j]表示相应路径上顶点i和顶点j的最近一次中间点编号；**

Floyd算法小结

- Floyd算法的时间复杂度为$O(n^3)$。
- Floyd算法的空间复杂度为$O(n^2)$。
- Floyd算法是直接求任意两点间最短路的方法。
- 实现简单，主体是三重循环。
- Floyd算法可应用于负权边的情况。

Floyd算法 VS n次单源最短路

- N次BFS，时间效率n*(n+e)，较好，适用无权
- n次Dijkstra，时间效率$n*n^2$，效率相当，适用非负权
- n次Bellman-Ford，时间效率n*ne，效率低
- n次SPFA ，时间效率n*ne，平均较好

Johnson算法的提出

- Floyd算法优雅
- Floyd效率较低，时间复杂度O(n^3)

- N次dijkstra+堆优化较优，使用Fibonacci堆优化能做到n^2logn，但不能处理负权.

- Johnson算法思想：将图转换成无负权的图，再使用dijkstra.

如何转换成无负权的图？

- 直接加最大负权的绝对值（减最大负权值）？

- 不正确！请构造反例

重赋权(rewrite)

- 引入势能函数h[]，对每一条边(u,v)，其权值重写为 W(u,v) = w(u,v) + h(u) – h(v)

- 定理：重赋权不改变最短路径，而且，使用w时不包含负权回路，当且仅当使用W时也不包含负权回路。

- 证明：设$p=(v_0,v_1,\ldots, v_k)$为从v_0到v_k的任意一条路径。则

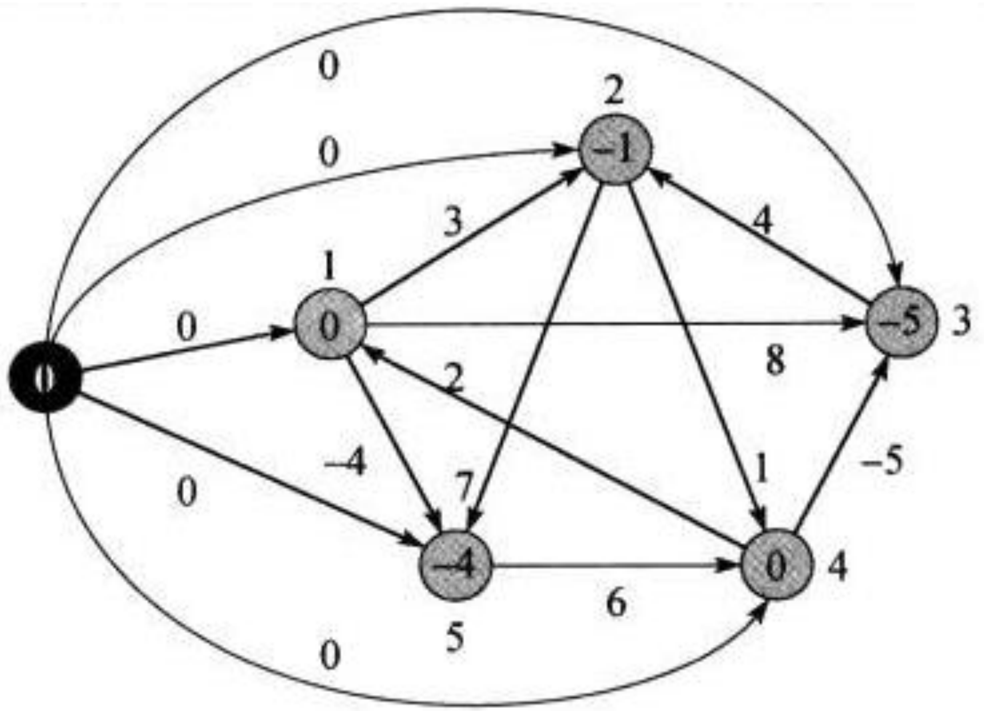
$$W(p) = \sum W(v_{i-1},v_i)$$
$$= \sum \{ w(v_{i-1},v_i) + h(v_{i-1}) - h(v_i) \}$$
$$= \sum w(v_{i-1},v_i) + h(v_0) - h(v_k)$$

因为$h(v_0)$，$h(v_k)$ 不依赖任何具体的路径，因此,$w(p) = \delta(v_0,v_k)$当且仅当$W(p) = \delta(v_0,v_k)$

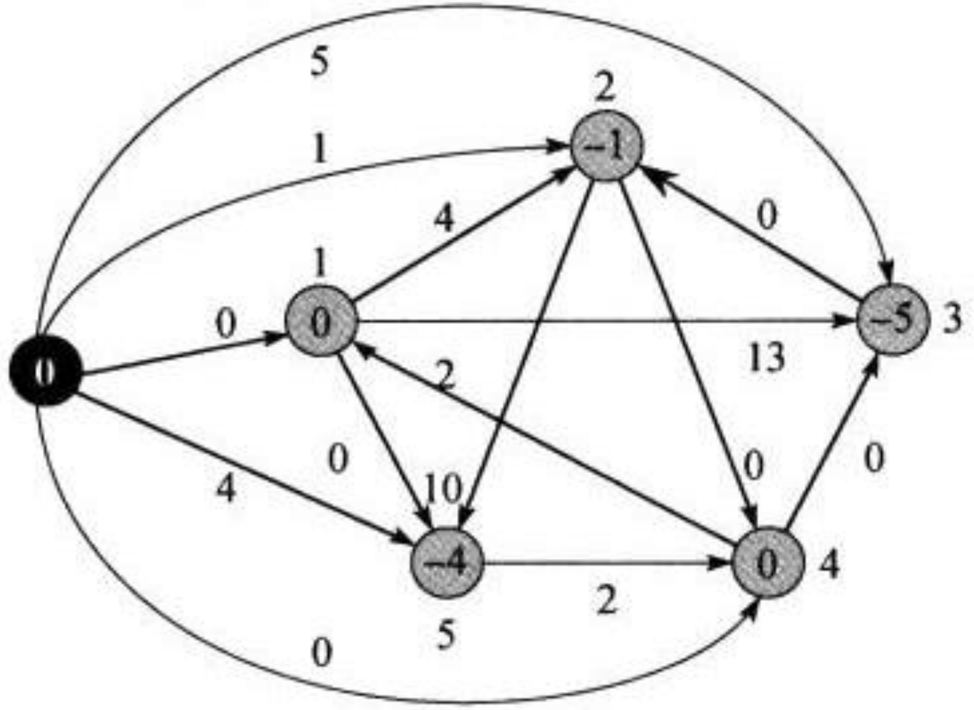
考虑任意环路$c=(v_0,v_1,\ldots, v_k)$, 有$v_0= v_k$, 从而,$W(c) = w(c)$.因此，环路c在w下为负当且仅当在W下为负

如何确定h?

- 引入虚源点 s
- 令 $h(v) = \delta(s,v)$
- 从而，对于所有的边(u,v)，W(u,v)为非负



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



5
2
1
−1
4
0
1
0
0
−5
3
13
2
0
0
4
0
10
0
−4
2
0
4
0
5

Johnson算法描述

1. 对于给定图 G=(V,E)，新增一顶点 S，对 S 到图中所有点都建一条边，得到新图 G'
2. 对图 G' 中点 S 使用 Bellman-Ford 算法计算单源最短路，得到势能函数 h[]
3. 对原图 G 中所有的边进行重赋值：对于每条边 (u,v)，其新的权值为 dis(u,v)+h[u]-h(v)
4. 对原图 G 的每个顶点运行 Dijkstra，求得全源最短路径

Johnson算法描述

- 算法Johnson(G,w)

1. [新图G’] G’.V = G.V + {s},G’.E=G.E+{(s,v)},w(s,v)=0;
2. if ! Bellman-Ford(G’,w,s) { print(“n-cycle”);return; }
3. for each v∈ G’.V { h[v] = δ(s,v);}
4. for each (u,v) ∈ G’.E, {W(u,v)=w(u,v)+h[u]－h[v]}
5. D = new (d_{uv})
6. for each u∈ G.V {
7. Dijkstra(G,W,u);
8. for each v ∈ G.V {d_{uv} = d_{uv} + h[v] - h[u];}
9. }

Johnson算法分析

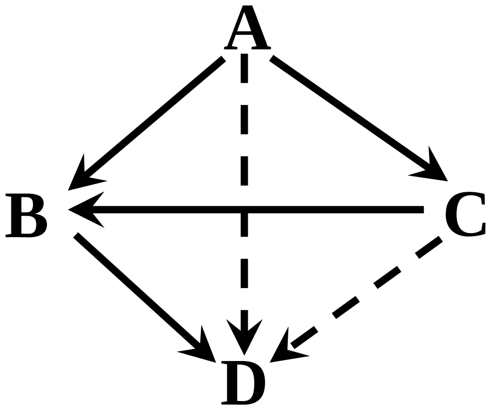
- 时间复杂度：$O(n^3+n*e)$

- 优化措施：对于无负环的边稠密图
  - ✓ 使用SPFA代替Bellman-Ford
  - ✓ 使用Fibonacci堆优化的Dijkstra
  - ✓ 优化后可达到$O(n^2 logn)$

可及性

- 图G中的两个顶点v_i和v_j，若从v_i到v_j存在一条有向路径，则称v_i到v_j可及。
- 用n×n阶可及矩阵R来描述顶点之间的可及关系，可及矩阵R的定义如下：

  若v_i到v_j可及，则$R_{ij}=1$，否则$R_{ij}=0$.

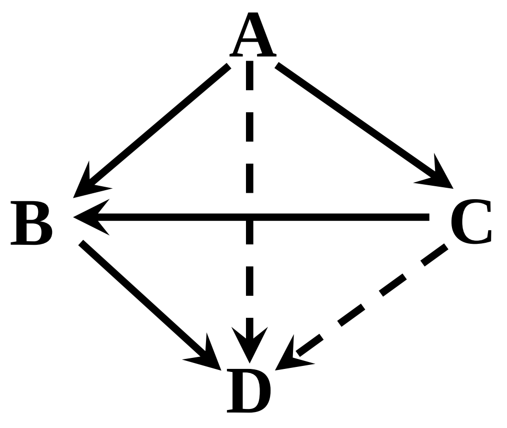


$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

可及矩阵

图的传递闭包

- 可及的传递性：若v_i到v_j可及， v_j到v_k可及, 则v_i到v_k可及。
- 由图G的顶点集V、边集E，以及新添加的虚边（表示顶点可及）构成原图G的扩展图，也称图G的传递闭包。



Warshall算法

- $WSM^{(k)}[i][j] =$

  $WSM^{(k-1)}[i][j]$ OR

  $WSM^{(k-1)}[i][k]$ AND $WSM^{(k-1)}[k][j]$，$1 \le k \le n$

  其中，$WSM^0=A$，$WSM^{(k)}[i][j]$表示顶点 i 只经过顶点1, 2, …, k 到达 j 的可及性. A为有向图G的邻接矩阵，称WSM为沃肖尔矩阵.

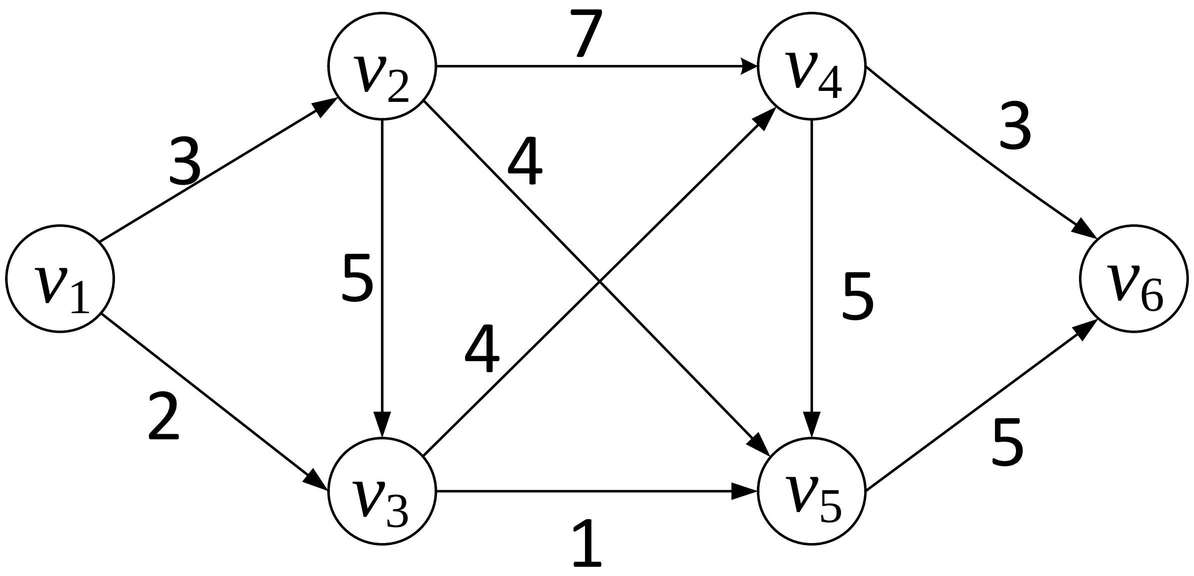
- 沃肖尔(Warshall) 算法与弗洛伊德(Floyd)算法类似, 是一个动态规划过程

M最短路问题

- 按照路径长度的递增次序枚举出给定的两个结点间所有可能的简单路径
- 假设$G=(V,E)$是一个包含n个节点的有向图。集合P包含G中从v_1到v_n的所有简单路径，设$p_1=r[0], r[1], \ldots, r[k]$是从$v_1$到$v_n$的一条最短路径，即，$p_1$始于$v_{r[0]}=v_1$，到$v_{r[1]}$，然后到$v_{r[2]}, \ldots, v_{r[k]}=v_n$.

集合$P-\{p_1\}$可划分成k种情况

- (k) 不包含边$< v_{r[k-1]},v_{r[k]}>$。
- (k-1) 包含边$< v_{r[k-1]},v_{r[k]}>$，但不包含$< v_{r[k-2]},v_{r[k-1]}>$
- ……
- (2) 包含边$<v_{r[2]}, v_{r[3]}>, \ldots, <v_{r[k-1]}, v_{r[k]}>$, 但是不包含边$<v_{r[1]}, v_{r[2]}>$;
- (1) 包含边$<v_{r[1]}, v_{r[2]}>, \ldots, <v_{r[k-1]}, v_{r[k]}>$, 但是不包含边$<v_{r[0]}, v_{r[1]}>$;



最短路径	路径长度	包含的边	不包含的边	新的路径
v1v3v5v6	8	无	无	
		无	<v5,v6>	v1v3v4v6=9
		<v5,v6>	<v3,v5>	v1v2v5v6=12
		<v3,v5> ,<v5,v6>	<v1,v3>	v1v2v3v5v6= 14

求有向图前M条最短路算法 MshortestPath(*G*, M)

MSP1[求从v_1到v_n的最短路径]

$Q \leftarrow$ {(从v_1到v_n的最短路径，Ø)}.

MSP2[求M条最短路径] //依次产生前M条最短路径

FOR *i*=1 TO M DO (

(1)令(*p*,C)是Q所有元素中*p*长度最小的二元组.

(2)输出路径*p*, 并从Q中删除*p*. //*p*是第*i*短的最短路径

(3)在已有约束C和划分产生的新约束下找出若干条G中的最短路径.

(4)将这些最短路径及其约束构成的二元组添加到Q中). ▌

M最短路算法分析

- 时间复杂度： $O(Mn^3)$
  - FOR循环的第3行需要求出$n-1$条最短路径，在每轮迭代中求最短路径的时间复杂度为$O(n^2)$，因此执行完FOR循环后这一行需要的总时间为$O(Mn^3)$。
  - For循环的第一行和第四行，将Q看作堆结构，则计算的时间复杂度会小于$O(Mn^3)$。
- 空间复杂度：$O(Mn^2)$
  - FOR循环的第三行每次迭代的时候至多会生成n-1条最短路径，最多有$O(Mn)$个元组存入Q中。每条路径最多有$O(n)$条边。因此，该算法的空间复杂度是$O(Mn^2)$

最短路小结

- 单源最短路径
  - ✓ 无权最短路径(BFS)
  - ✓ 正权最短路径(Dijkstra)
  - ✓ 负权最短路径:Bellman-Ford,SPFA
- 每对顶点之间的最短路径问题
  - ✓ Floyd算法
  - ✓ 图的传递闭包
- 拓展：M最短路、Johnson算法
- 求解最短路径的算法，有向图无向图均可。

思考

- 最长路问题
- 应用**Floyd**算法判负权环