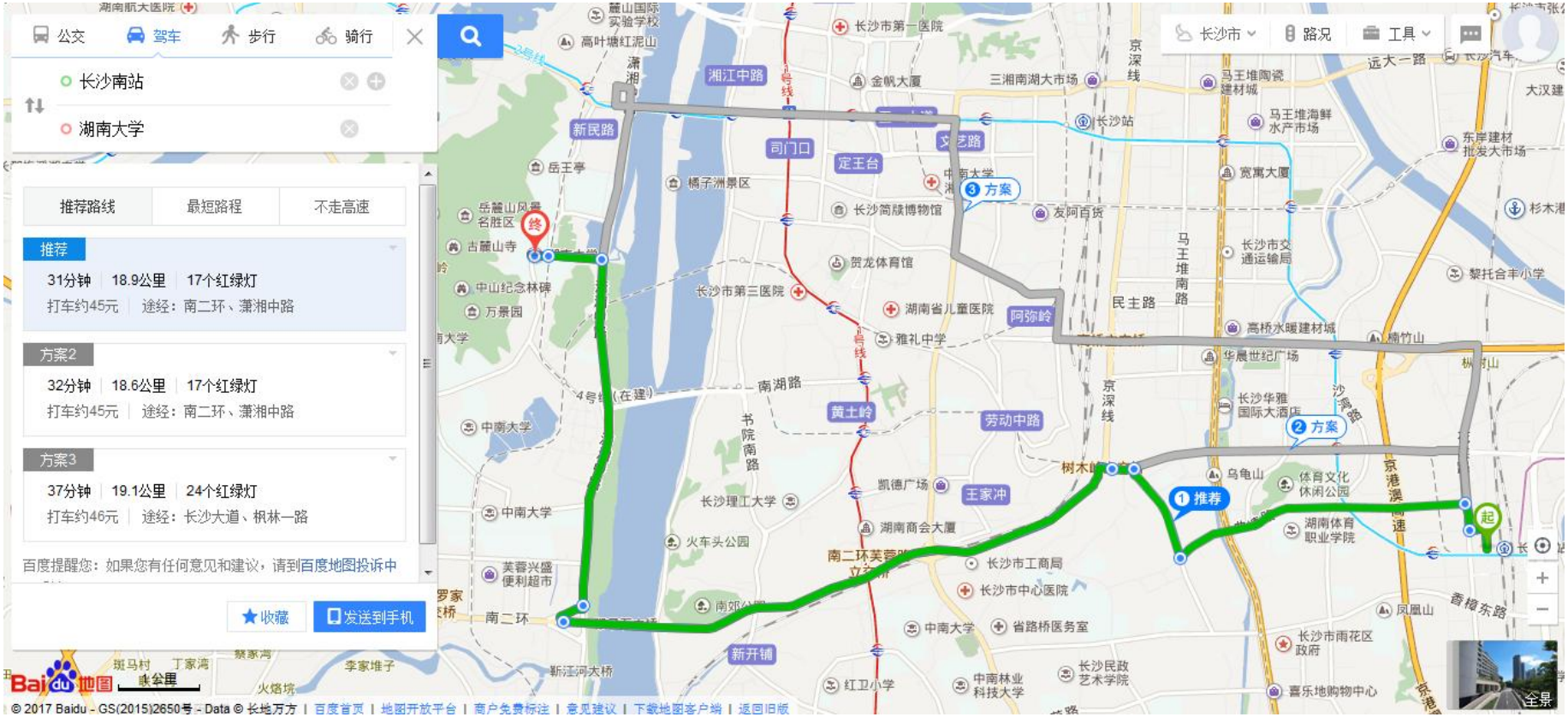
单源最短路径

湖南大学信息科学与工程学院

提 纲

8.1最短路径问题-应用

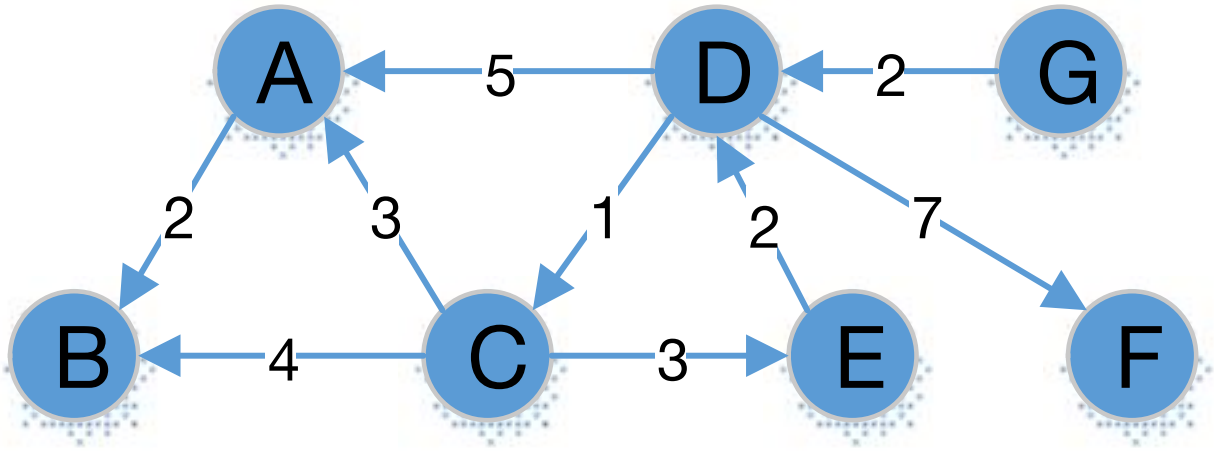

公交
驾车
步行
骑行
长沙南站
湖南大学
推荐路线
最短路程
不走高速
推荐
31分钟 | 18.9公里 | 17个红绿灯
打车约45元 | 途经：南二环、潇湘中路
方案2
32分钟 | 18.6公里 | 17个红绿灯
打车约45元 | 途经：南二环、潇湘中路
方案3
37分钟 | 19.1公里 | 24个红绿灯
打车约46元 | 途经：长沙大道、枫林一路
百度提醒您：如果您有任何意见和建议，请到百度地图投诉中
收藏
发送到手机
长沙市
路况
工具
湘江中路
新民路
司门口
定王台
阿弥岭
劳动中路
南湖路
黄土岭
王家冲
南二环
新开铺
方案
推荐
起
© 2017 Baidu - GS(2015)2650号 - Data © 长地万方 | 百度首页 | 地图开放平台 | 商户免费标注 | 意见建议 | 下载地图客户端 | 返回旧版

8.1最短路径问题

带权图：

带权图可以表示成一个三元组G=<V, E, W>，其中V 是点集合，$E \subseteq V \times V$表示边集合，其中每条边由V 中两个点相连接所构成，

W: $E \rightarrow R$将E中每条边(u,v)被赋上一个权重，记为W (u,v)



8.1最短路径问题

图上的路径：

给定一个带权图$G = (V, E, W)$，G 中顶点与边的交替序列$p = v_0e_1v_1e_2 \ldots e_lv_l$称为点$v_0$ 到点v_l 的**路径**，其中v_{r-1} 和v_r 是e_r 的起点和终点（$1 \le r \le l$）

p 中各条边权重之和$\sum_{k=1}^{k=l} w(e_k)$就是p 的长度，记为w(p)

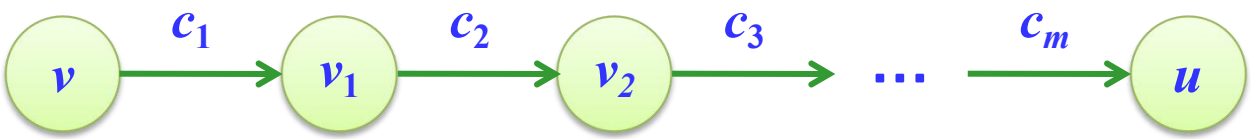
所有v_0 和点v_l 之间的路径中最短的那条路径称为**最短路径**，最短路径的长度被称为v_0 到点v_l 的**距离**，记为$\delta(v_0, v_l)$

如果不存在的v_0 到v_l路径，那么$\delta(v_0, v_l)$记为∞

8.1 最短路径问题

最短路径：

考虑带权有向图，把一条路径（仅仅考虑简单路径）上所经边的权值之和定义为该路径的路径长度或称带权路径长度。



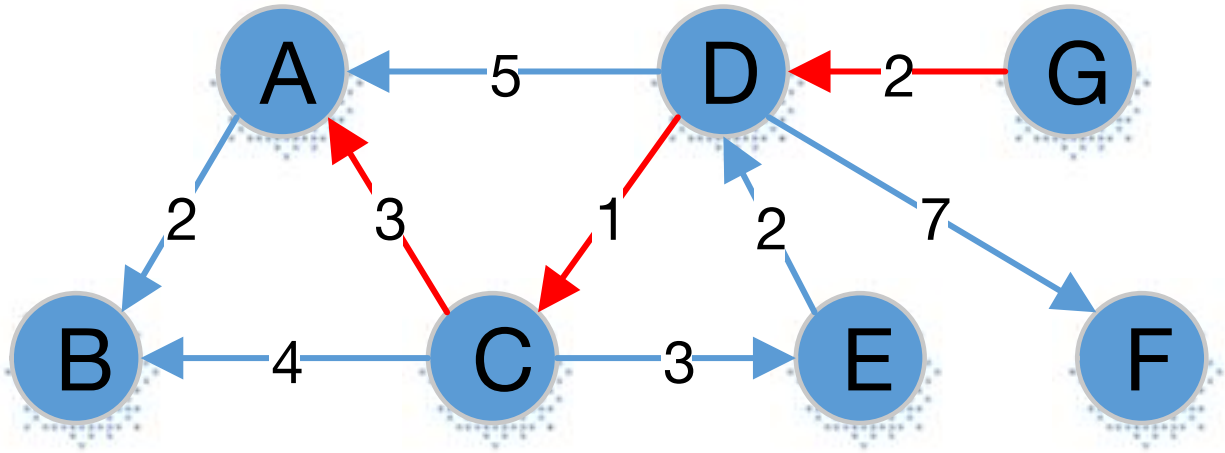
路径长度$=c_1+c_2+\cdots+c_m$

路径：　$(v,\ v_1,\ v_2,\ \cdots,\ u)$

从源点到终点可能不止一条路径，把路径长度最短的那条路径称为最短路径。

8.1最短路径问题

示例：



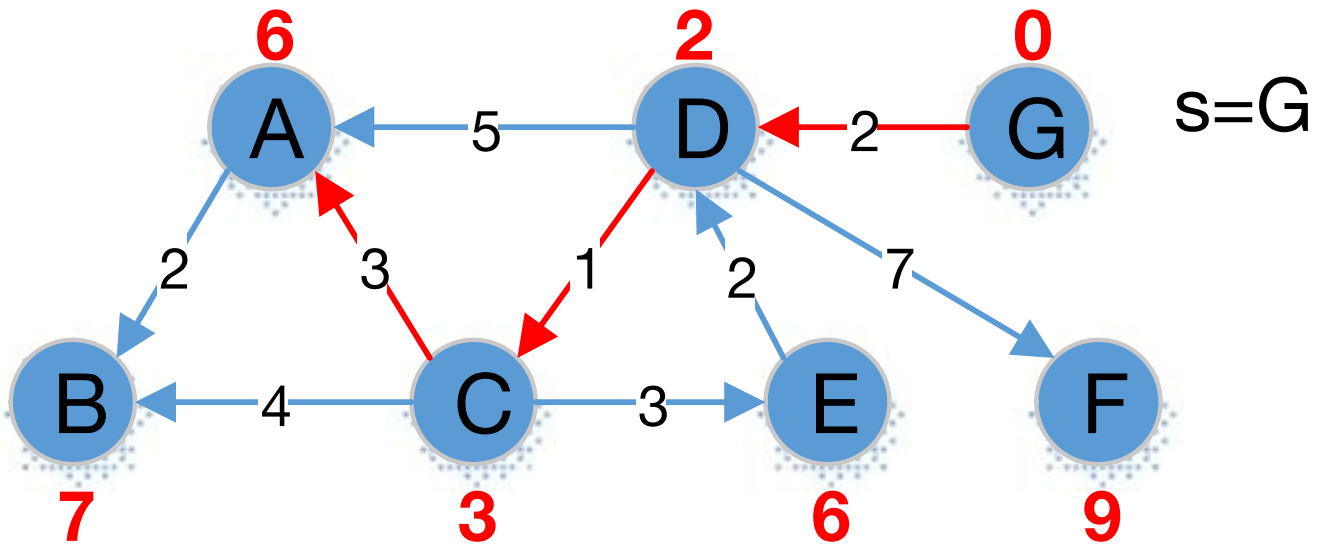
δ (G, A)=W(p)=2+1+3=6

δ (A, G)= ∞

8.1最短路径问题

单源最短路径问题：

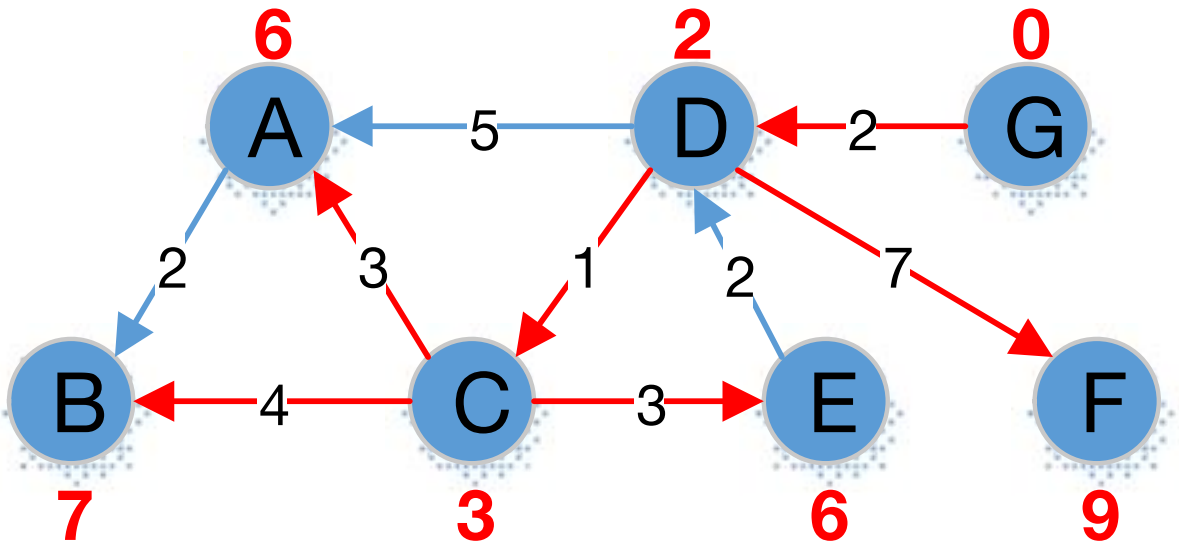
给定一个带权图G=(V, E, W)和一个源点s，计算s到其他V中所有点的距离。



8.1最短路径问题

最短路径树：

由源点s到所有点的最短路径合起来就形成一棵树，称为s的最短路径树。



为了得到最短路径树，计算任一点v在树上的父节点，记为path[v]

提 纲

8.1 最短路径问题

8.2 松弛算法

8.3 有向无环图上的最短路径计算

8.4 Dijkstra算法

8.5 Bellman-Ford算法

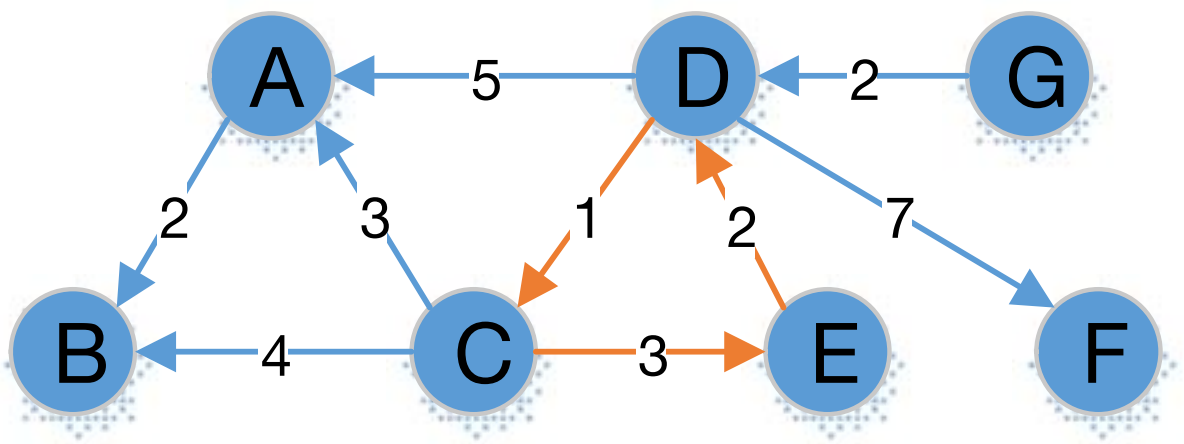
8.6 差分约束和最短路径

8.7 最短路径问题扩展

8.2松弛算法

求解最短路径的蛮力法:

为了计算s到t的最短路径，枚举出s到t的所有路径并计算最小值，如果有环，路径数量是无穷多。



8.2松弛算法

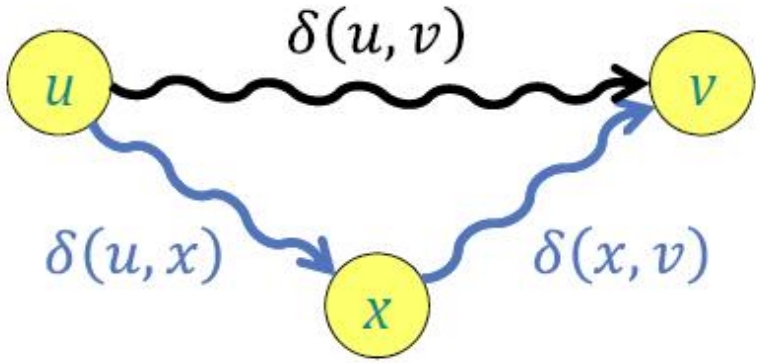
求解最短路径的蛮力法：

为了计算s到t的最短路径，枚举出s到t的所有**简单**路径并计算最小值，路径数量是指数级别的O(|V|!)

8.2松弛算法

三角不等式：

对于图上任意三个点u、v和x而言，$\delta(u,v) < \delta(u,x)+\delta(x,v)$



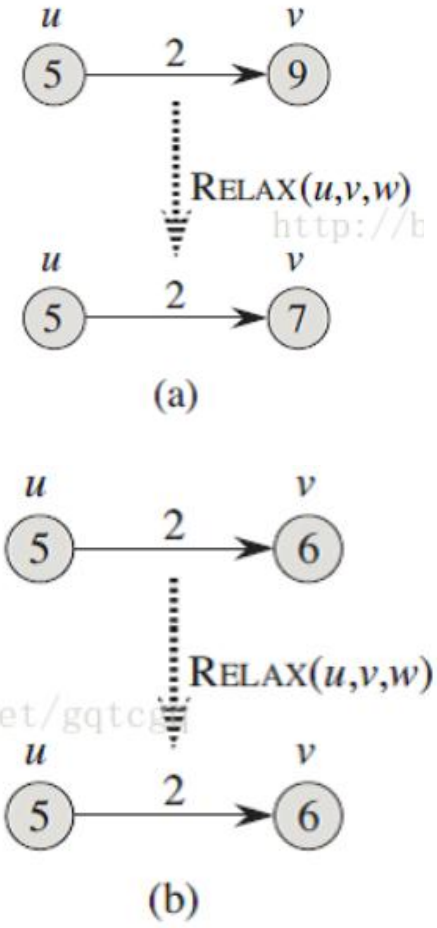
推论：对于图上任意三个点u、v和x且存在边(x,v)而言，

$\delta(u,v) \leq \delta(u,x)+w(x,v)$

8.2松弛算法

对每个点v而言，维护一个属性dist[v]，记录源点s到点d的距离上界，不断地降低对dist[v]的估计，直到dist[v]等于$\delta(s, v)$。

```
Relax(G, s)
   for v in V:
       dist[v]= ∞;
       path[v]=NULL;
   dist[s]=0;
   while some edge (u, v) has
dist[v]>dist[u]+w(u,v):
       pick such an edge (u, v)
       dist[v]=dist[u]+w(u,v);
       path[v]=u;
```



8.2松弛算法

路径松弛性质：

给定带权图G=(V, E, W)，设从源点s到点v_k的最短路径为$se_1v_1e_2 \ldots e_kv_k$，如果对边$e_1, e_2, \ldots, e_k$按次序进行松弛操作之后，$dist[v_k]=\delta(s, v_k)$，且该性质的成立与其他边的松弛操作以及次序无关。

证明：数学归纳法，对于路径边数进行归纳

基础步：边数为0的最短路径就是源点，初始化阶段就有$dist[s]=\delta(s, s)=0$；

归纳步：边数为n的最短路径都成立，边数为n+1最短路径$se_1v_1e_2 \ldots e_nv_n e_{n+1}v_{l+1}$有两部分组成：边数为n的最短路径和只有一条边$e_{n+1}$的一条最短路径。

边数为n的最短路径按归纳假设有$dist[v_n]=\delta(s, v_n)$，于是按照松弛操作有

$dist[v_{n+1}]=dist[v_n]+w(e_{n+1})=\delta(s, v_{n+1})$

8.2松弛算法

松弛算法正确性（上界性质）：

定理：在松弛算法中，对于V中任意的v，dist[v]一直大于等于$\delta(s,v)$（最短路径），且一旦dist[v]到达$\delta(s,v)$之后将不再发生变化。

证明：数学归纳法，针对调用松弛操作的次数

基础步：当松弛操作次数为0的时候，即初始化阶段，$dist[s] = 0 \geq \delta(s,s) = 0$，而其它点v的dist[v]都是∞都是大于等于$\delta(s,v)$。

归纳步：当松弛操作次数小于等于n时候，设第n+1次松弛操作加入了边(u,v)。

由于dist[u]是在前n中操作中得到的值，所以$\delta(s,u) \leq dist[u]$，于是，由三角不等式，$\delta(s,v) \leq \delta(s,u)+w(u,v) \leq dist[u]+w(u,v)$，而$dist[u]+w(u,v)$= dist[v]就是第n+1次松弛操作的结果。

8.2松弛算法

收敛性质：

对于某些节点u, v ∈V，如果s ~> u-> v是图G中的一条最短路径，并且对边(u, v)进行松弛前的任意时间有dist[u] = δ(s, u)，则在对边(u, v)松弛之后的所有时间dist[v] =δ(s, v)。

8.2松弛算法

前驱子图性质：

对于所有的节点v∈ V，一旦dist[v] =δ(s, v)，则前驱子图是一颗根节点为s的最短路径树。

提 纲

8.1 最短路径问题

8.2 松弛算法

8.3 有向无环图上的最短路径计算

8.4 Dijkstra算法

8.5 Bellman-Ford算法

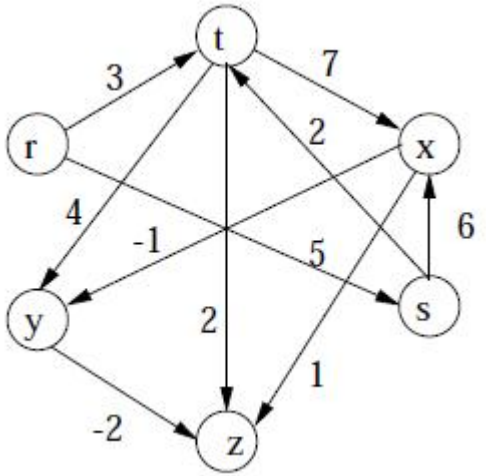
8.6 差分约束和最短路径

8.7 最短路径问题扩展

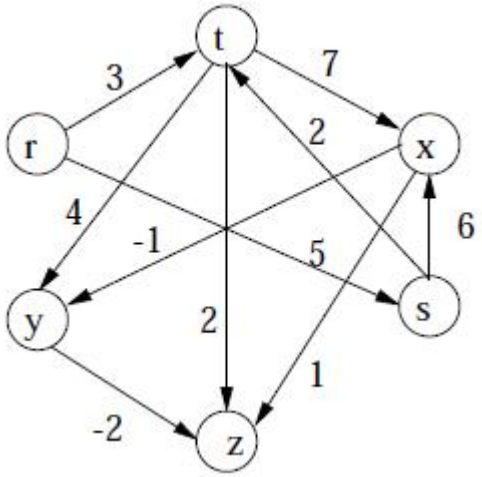
8.3有向无环图上的最短路径计算

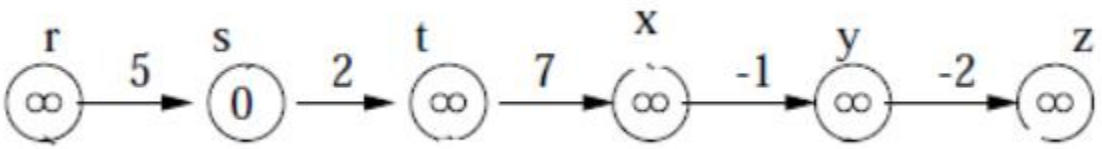
- 首先，对有向无环图中所有点进行拓扑排序；
- 然后，按照拓扑排序的顺序对所有边进行操作松弛。
- 时间复杂度O(|V|+|E|)。

算法导论， P381-383

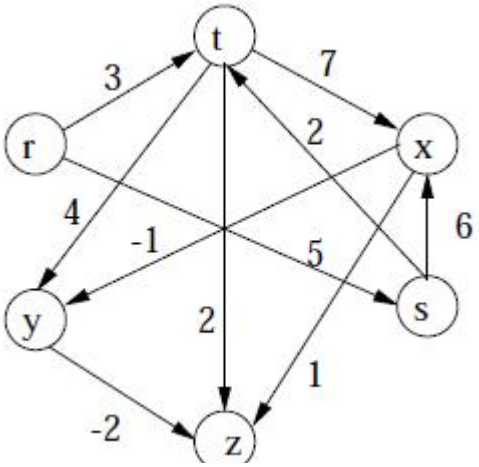


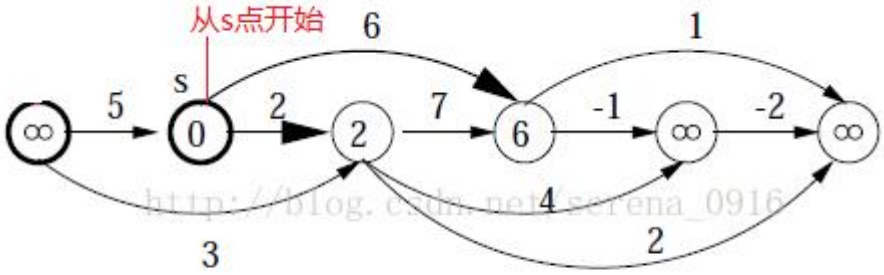
8.3有向无环图上的最短路径计算





8.3有向无环图上的最短路径计算

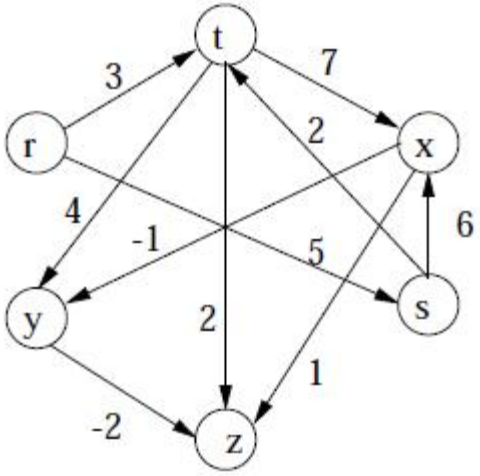


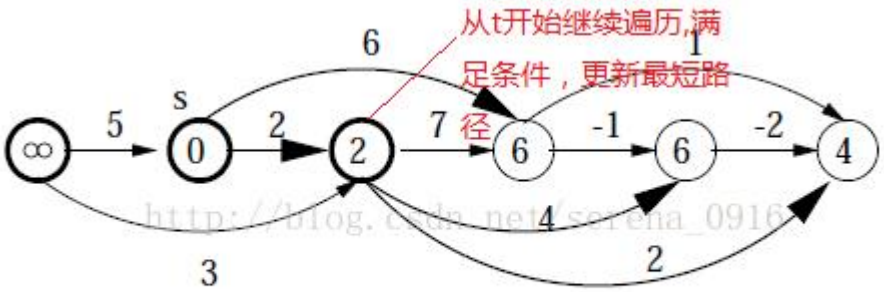


1 if(d[v]>d[u]+w(u,v))
2 then d[v]←d[u]+w(u,v)
3 π[v]←u　/*解释松弛操作*/

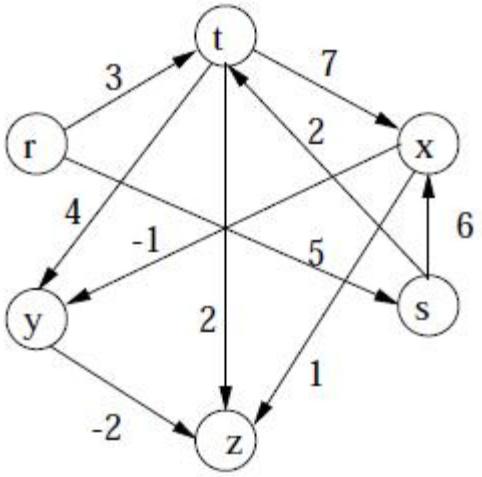
s到t的最短路径长度为2,因此修改d[t]=2,s到x的目前最短路径为6，因此修改d[x]=6,一次松弛操作完成，结果如上图所示。

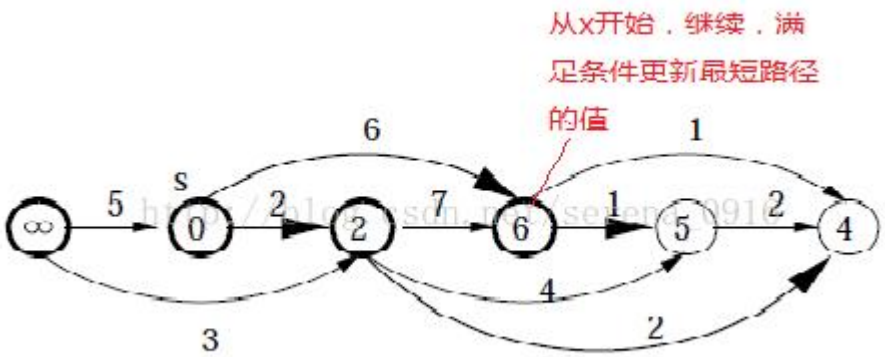
8.3有向无环图上的最短路径计算



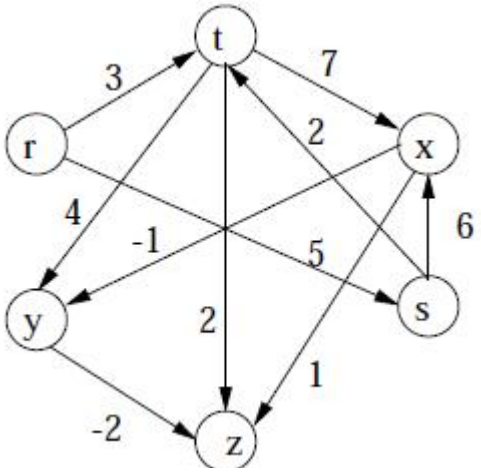


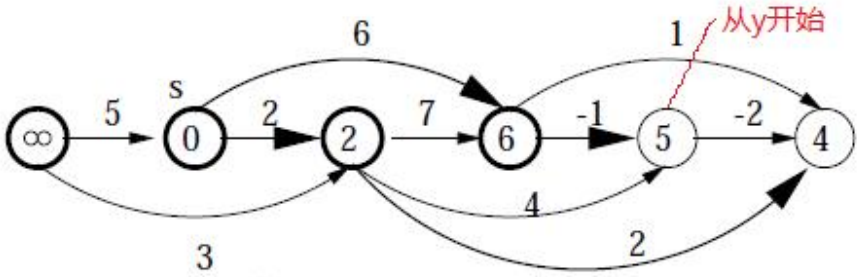
8.3有向无环图上的最短路径计算

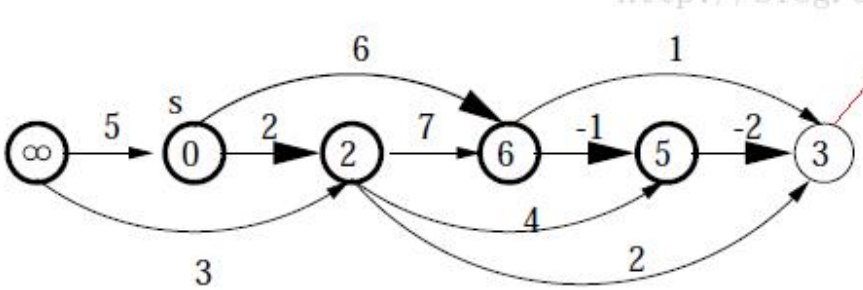


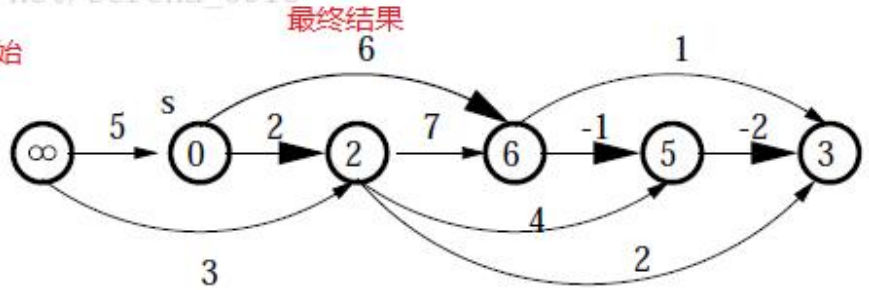


8.3有向无环图上的最短路径计算









提 纲

8.1 最短路径问题

8.2 松弛算法

8.3 有向无环图上的最短路径计算

8.4 Dijkstra算法

8.5 Bellman-Ford算法

8.6 差分约束和最短路径

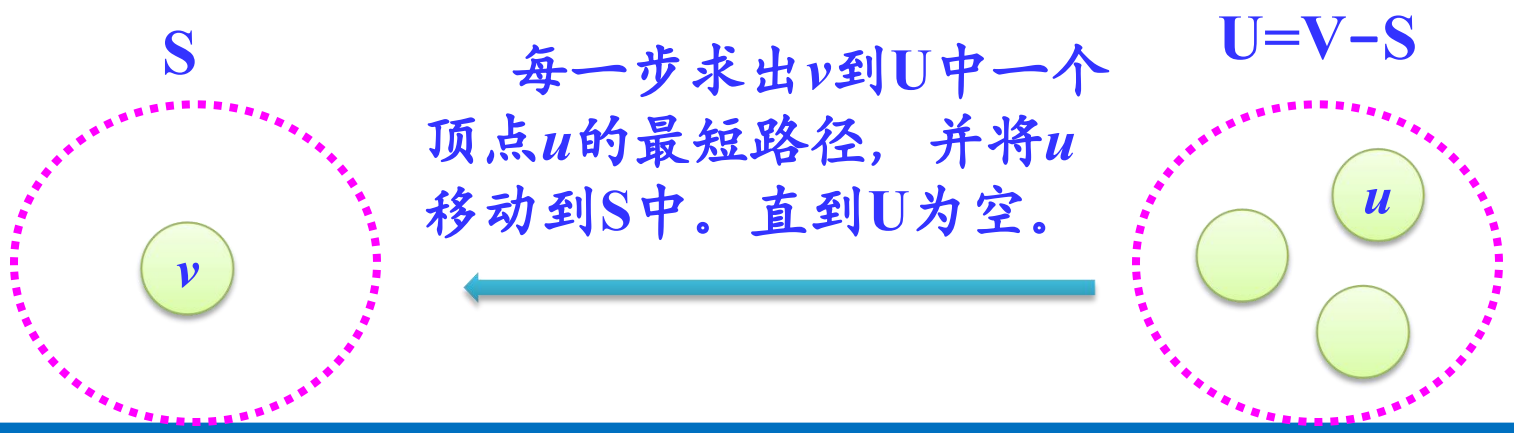
8.7 最短路径问题扩展

8.4 Dijkstra算法

狄克斯特拉（Dijkstra）求解思路

设G=(V，E)是一个带权有向图，把图中顶点集合V分成两组：

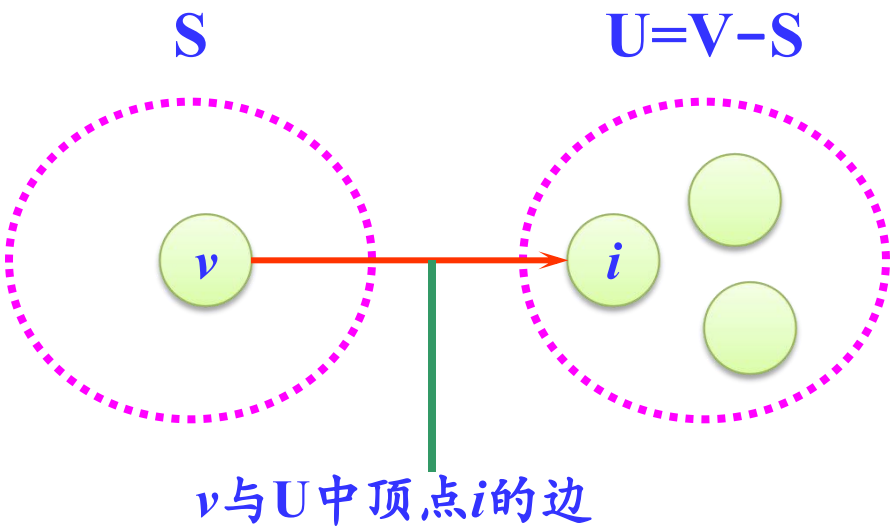
- 第1组为已求出最短路径的顶点集合（用S表示，初始时S中只有一个源点，以后每求得一条最短路径v，…，u，就将u加入到集合S中，直到全部顶点都加入到S中，算法就结束了）。
- 第2组为其余未求出最短路径的顶点集合（用U表示）。



8.4 Dijkstra算法

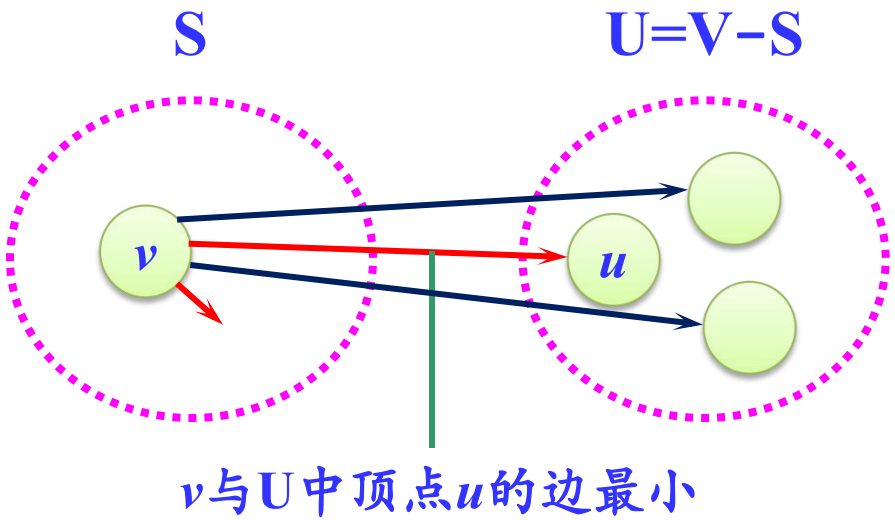
Dijkstra算法的过程

（1）初始化：，S只包含源点即S={v}，v的最短路径为0。U包含除v外的其他顶点，U中顶点i距离为边上的权值（若v与i有边<v, i>）或∞（若i不是v的出边邻接点）。



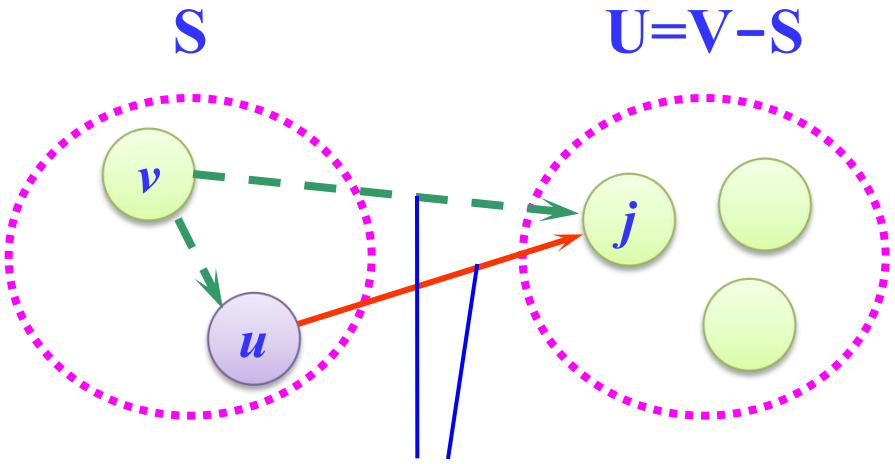
8.4 Dijkstra算法

（2）从U中选取一个距离v最小的顶点u，把u加入S中（该选定的距离就是v ⇨ u的最短路径长度）。



8.4 Dijkstra算法

（3）以u为新考虑的中间点，修改U中各顶点j的最短路径长度：若从源点v到顶点j（$j \in U$）的最短路径长度（经过顶点u）比原来最短路径长度（不经过顶点u）短，则修改顶点j的最短路径长度。

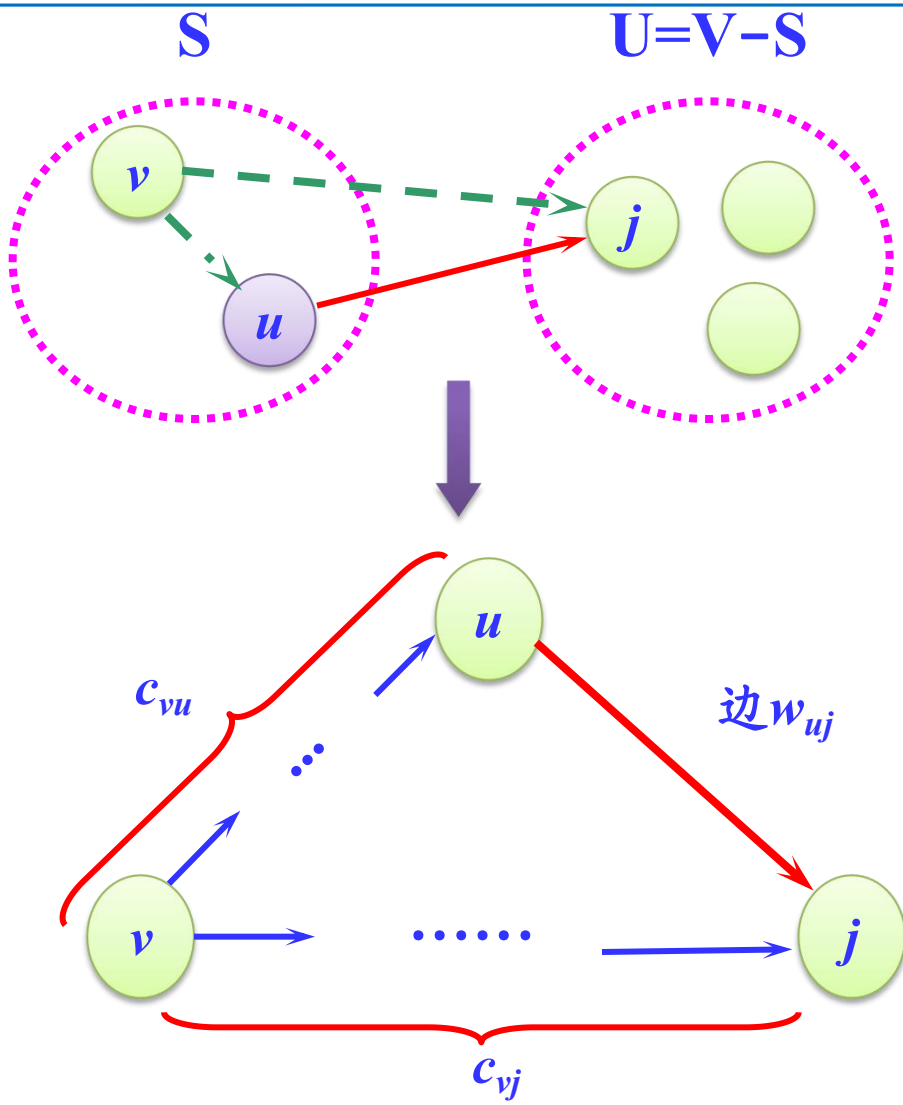


两条路径进行比较：

若经过u的最短路径长度更短，则修正

8.4 Dijkstra算法

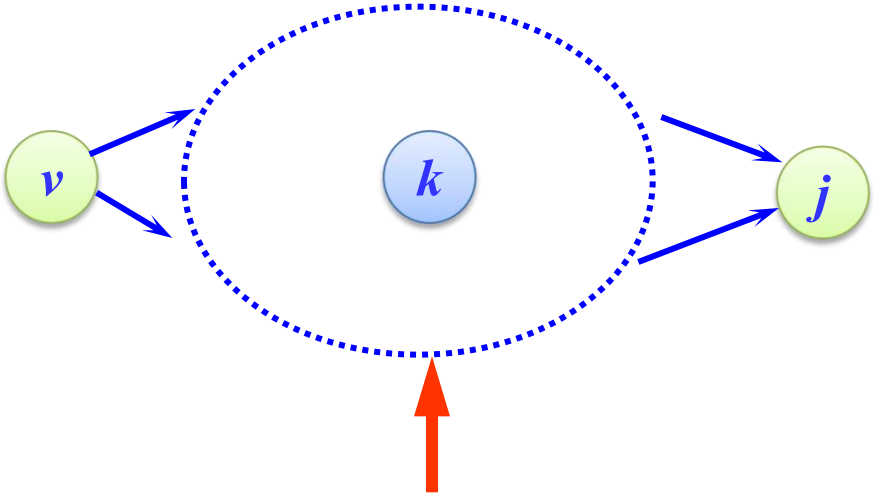
修改方式



顶点v ⇨ j的最短路径长度 = MIN($c_{vk}+w_{kj}$, c_{vj})

8.4 Dijkstra算法

（4）重复步骤（2）和（3）直到所有顶点都包含在S中。



考虑中间其他所有顶点k，通过
比较得到v ⇨ j的最短路径

8.4 Dijkstra算法

算法设计（解决2个问题）

- 如何存放最短路径长度：

  用一维数组dist[*j*]存储！

  源点*v*默认， dist[*j*]表示源点 ⇨ 顶点*j*的最短路径长度。如dist[2]=12表示源点 ⇨ 顶点2的最短路径长度为12。

- 如何存放最短路径：

  从源点到其他顶点的最短路径有*n*-1条，一条最短路径用一个一维数组表示，如从顶点0 ⇨ 5的最短路径为0、2、3、5，表示为

  path[5]={0,2,3,5}。

  所有*n*-1条最短路径可以用二维数组path[][]存储。

8.4 Dijkstra算法

改进的方法是采用一维数组path来保存：

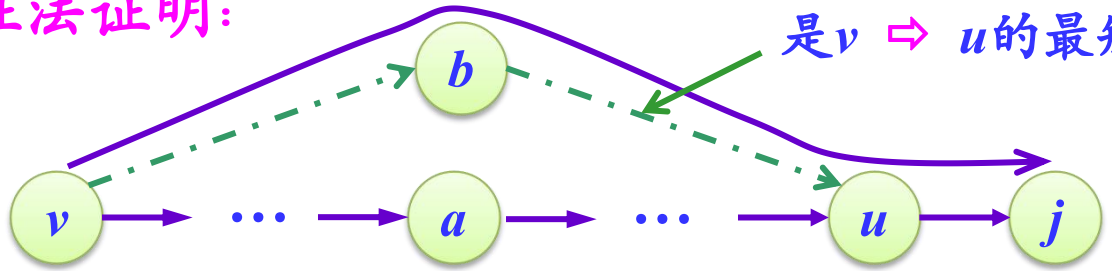
若从源点v ⇨ j的最短路径如下：

v ⇨ j最短路径中j的前一个顶点

则

一定是从源点v ⇨ u的最短路径

反证法证明：

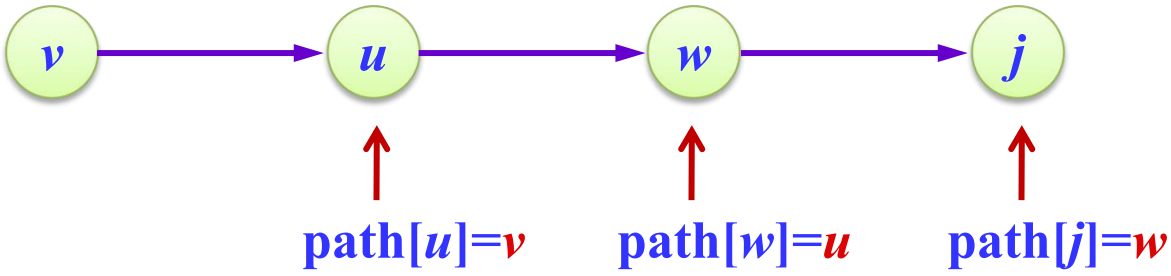


而通过b的路径更短，则$v \rightarrow \cdots a \rightarrow \cdots u \rightarrow j$不是最短路径

与假设矛盾，问题得到证明。

8.4 Dijkstra算法

v ⇨ j的最短路径:

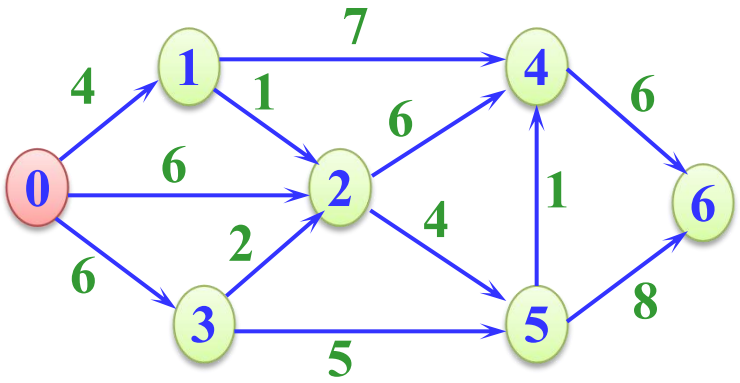


从path[j]推出的逆路径: j, w, u, v

对应的最短路径为: $v \rightarrow u \rightarrow w \rightarrow j$

8.4 Dijkstra算法

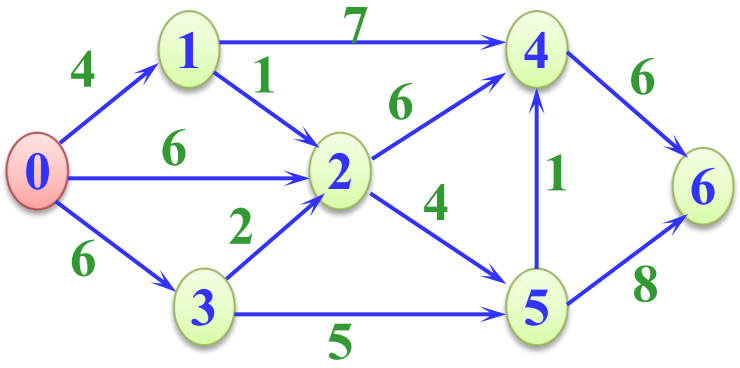
Dijkstra算法示例演示



S	U	dist[]	path[]
		0 1 2 3 4 5 6	0 1 2 3 4 5 6
{0}	{1,2,3,4,5,6}	{0, 4, 6, 6, ∞, ∞, ∞}	{0, 0, 0, 0, -1, -1, -1}
		最小的顶点：1	
{0,1}	{2,3,4,5,6}	{0, 4, 5, 6, 11, ∞, ∞}	{0, 0, 1, 0, 1, -1, -1}
		最小的顶点：2	
{0,1,2}	{3,4,5,6}	{0, 4, 5, 6, 11, 9, ∞}	{0, 0, 1, 0, 1, 2, -1}

8.4 Dijkstra算法

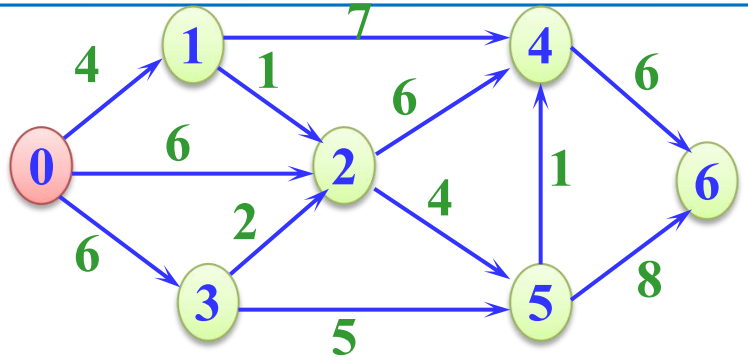
Dijkstra算法示例演示



S	U	dist[]	path[]
		0 1 2 3 4 5 6	0 1 2 3 4 5 6
{0,1,2}	{3,4,5,6}	{0, 4, 5, 6, 11, 9, ∞}	{0, 0, 1, 0, 1, 2, -1}
		最小的顶点：3	
{0,1,2,3}	{4,5,6}	{0, 4, 5, 6, 11, 9, ∞}	{0, 0, 1, 0, 1, 2, -1}
		最小的顶点：5	
{0,1,2,3,5}	{4,6}	{0, 4, 5, 6, 10, 9, 17}	{0, 0, 1, 0, 5, 2, 5}

8.4 Dijkstra算法

Dijkstra算法示例演示



S	U	dist[]	path[]
		0 1 2 3 4 5 6	0 1 2 3 4 5 6
{0,1,2,3,5}	{4,6}	{0, 4, 5, 6, 10, 9, 17}	{0, 0, 1, 0, 5, 2, 5}
		最小的顶点：4	
{0,1,2,3,5,4}	{6}	{0, 4, 5, 6, 10, 9, 16}	{0, 0, 1, 0, 5, 2, 4}
		最小的顶点：6	
{0,1,2,3,5,4,6}	{}	{0, 4, 5, 6, 10, 9, 16}	{0, 0, 1, 0, 5, 2, 4}

最终结果

8.4 Dijkstra算法

利用dist和path求最短路径长度和最短路径

❶ 求0 ⇨ 6的最短路径长度：

0 1 2 3 4 5 6

dist={0, 4, 5, 6, 10, 9, 16}

从顶点0 ⇨6的最短路径长度为16

C++: Graph_Dijkstra

❷ 求0 ⇨ 6的最短路径：

0 1 2 3 4 5 6

path={0, 0, 1, 0, 5, 2, 4}

path[6]=4

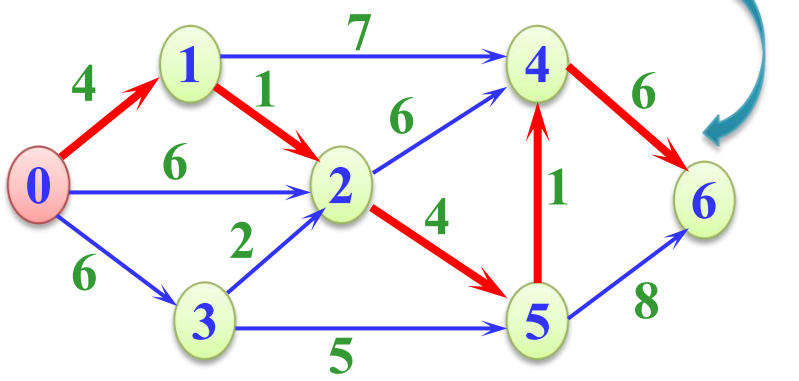
path[4]=5

path[5]=2

path[2]=1

path[1]=0到源点

最短路径为：0→1 →2 →5 →4



8.4 Dijkstra算法

算法导论：P363 伪代码

```
Dijkstra(G, Adj, s)
  dist[s]=0;
  for each v ∈ V – {s}
      do dist[v] = ∞
  S = ∅; Q = V
  while Q ≠ ∅:
      u = EXTRACT-MIN(Q); S = S ∪ {u};
      for each v ∈ Adj[u]:
          if dist[v] > dist[u] + w(u, v)
            then dist[v] =dist[u] + w(u, v)
```

O(Adj(u))

O(|V|)

DECREASE-KEY

8.4 Dijkstra算法

算法正确性：

给定带权图G=(V, E, W)，算法终止的时候dist[v]=δ(s, v)

证明：反证法

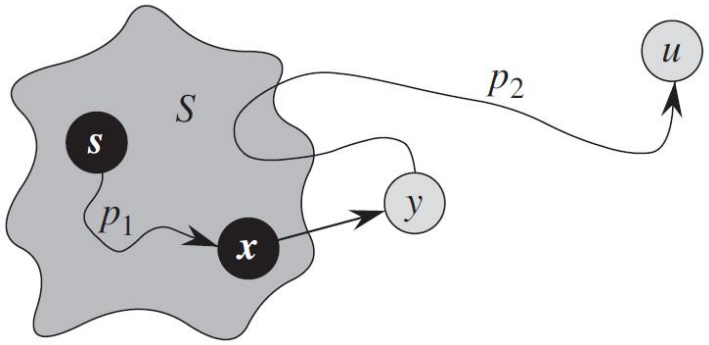
假设u是第一个出优先队列时$dist[u] \neq \delta(s,u)$的点，

必然存在一条最短路径p是s到u，

x是p中u出优先队列时最后一个进S的点，

y是x在p中的后继，显然dits[y]小于等于dist[_]，

所以y肯定要在u之前出队列，与假设矛盾



8.4 Dijkstra算法

时间复杂度：

$$\text{Time} = \Theta(V)\cdot T_{\text{EXTRACT-MIN}} + \Theta(E)\cdot T_{\text{DECREASE-KEY}}$$

Q	$T_{\text{EXTRACT-MIN}}$	$T_{\text{DECREASE-KEY}}$	Total
array	$O(V)$	$O(1)$	$O(V^2)$
binary heap	$O(\lg V)$	$O(\lg V)$	$O(E \lg V)$
Fibonacci heap	$O(\lg V)$ amortized	$O(1)$ amortized	$O(E + V \lg V)$ worst case

8.4 Dijkstra算法

思考题：

Dijkstra算法为什么不适合负权值的情况?

思考题：

Dijkstra算法可以用于带权无向图求最短路径吗?

思考题：

Dijkstra算法是否一定可以停止吗?

提 纲

8.1 最短路径问题

8.2 松弛算法

8.3 有向无环图上的最短路径计算

8.4 Dijkstra算法

8.5 Bellman-Ford算法

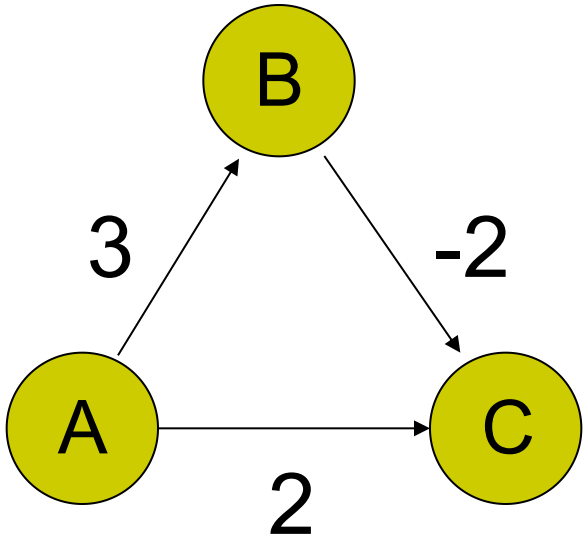
8.6 差分约束和最短路径

8.7 最短路径问题扩展

8.5 Bellman-Ford算法

Dijkstra算法的局限性:

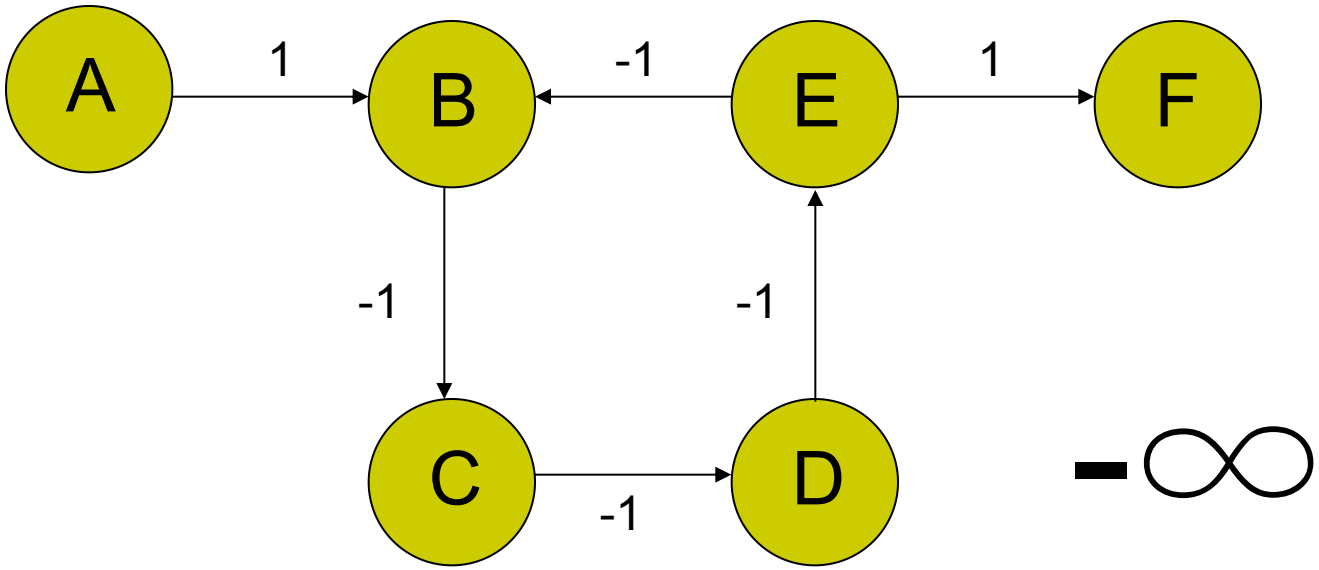
- 如果边权为负值，Dijkstra算法还正确吗?
- 求解右图A 至其他点的最短距离
- 算法步骤:
  1）标记点A
  2）dist[C]=2最小，标记点C
  3）dist[B]=3最小，标记点B
  结束
- 但是δ(A, C) =1



8.5 Bellman-Ford算法

Dijkstra算法的局限性：

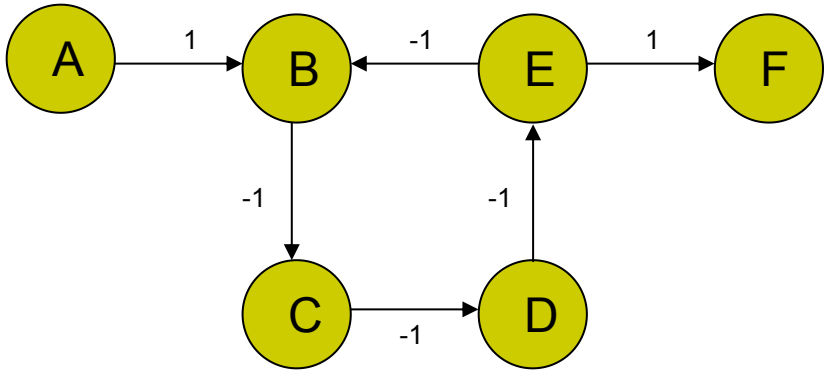
下图中，A至F的最短路径长度是多少？



8.5 Bellman-Ford算法

Dijkstra算法的局限性：

- 如果利用Dijkstra算法求解，结果为……
- 标记点A，dist[B]=1，标记点B
- dist[C]=0，标记点C
- dist[D]=-1，标记点D
- dist[E]=-2，标记点E
- dist[F]=-1，标记点F
- 所求得的距离
  并不是最短的



8.5 Bellman-Ford算法

Dijkstra算法的局限性：错误结果的原因

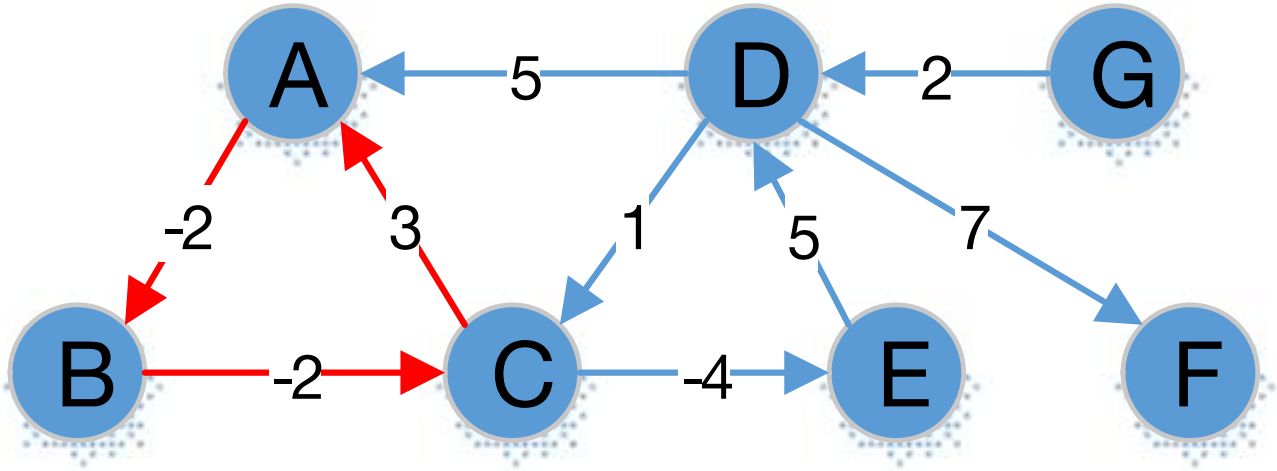
- Dijkstra的缺陷就在于它不能处理负权回路：Dijkstra对于标记过的点就不再进行更新了，所以即使有负权导致最短距离的改变也不会重新计算已经计算过的结果。
- 我们需要新的算法——Bellman-Ford

8.5 Bellman-Ford算法

负权环路:

给定一条路径p = $v_0e_1v_1e_2 \ldots e_lv_l$， 如果若$v_0$ 和点v_l 是同一个点，则p 被称为一条环路。

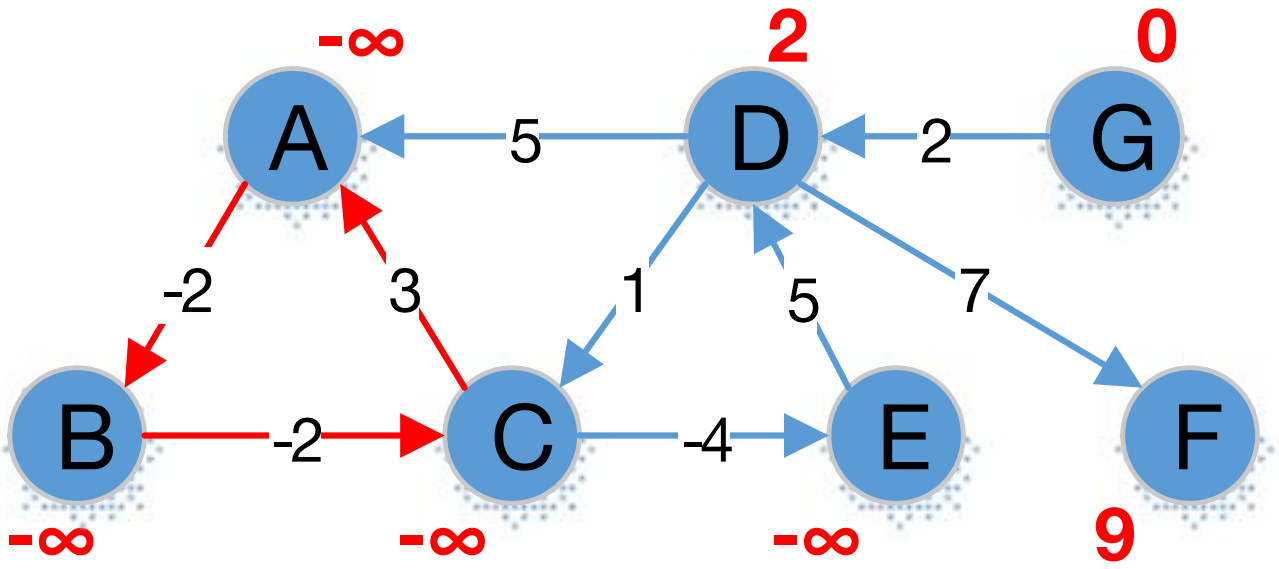
如果p 中各条边权重之和$\sum_{k=1}^{k=l} w(e_k)$是负数，那么p称为负权环路。



8.5 Bellman-Ford算法

基于负权环路的距离定义：

当图中有负权环路的时候，部分点的最短路径就未定义了，记为- ∞



8.5 Bellman-Ford算法

算法伪代码：

```
Bellman-Ford(G,w,s)
  for v in V:
      dist[v]= ∞; path[v]=NULL;
      dist[s]=0;
  for i from 1 to |V|-1 :
     for each edge (u, v) ∈ E:
         Relax(u,v,w)
  for each edge (u, v) ∈ E:
     if dist[v] > dist[u] + w(u, v)
         return FALSE
   return True
```

Bellman – Ford算法可以大致分为三个部分：

第一，初始化所有点。每一个点保存一个值，表示从原点到达这个点的距离，将原点的值设为0，其它的点的值设为无穷大（表示不可达）。

第二，进行循环，循环下标为从1到n – 1（n等于图中点的个数）。在循环内部，遍历所有的边，进行松弛计算。

第三，遍历途中所有的边（edge（u，v）），判断是否存在这样情况：

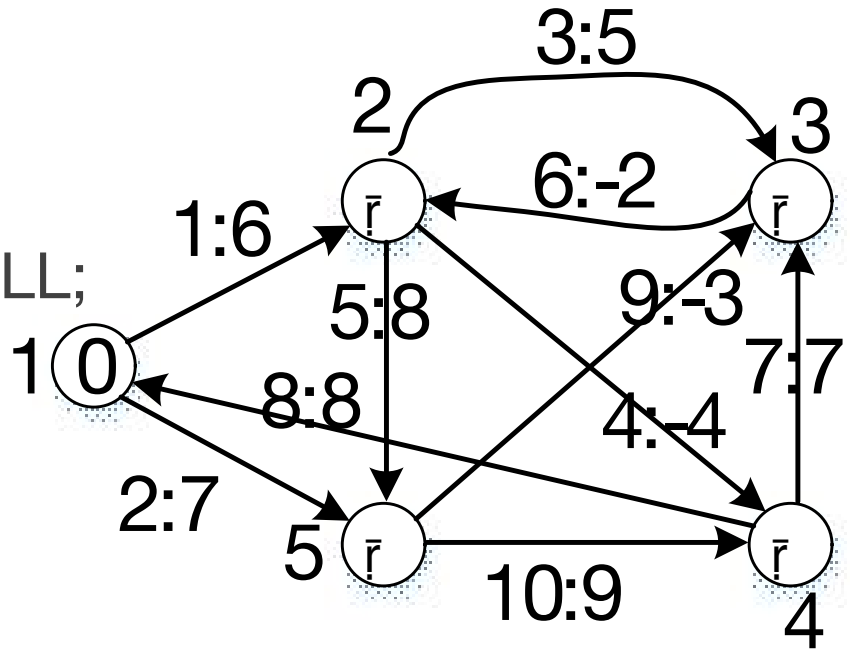
dist[v] > dist[u] + w(u,v)

则返回false，表示途中存在从源点可达的权为负的回路。

8.5 Bellman-Ford算法

算法伪代码：

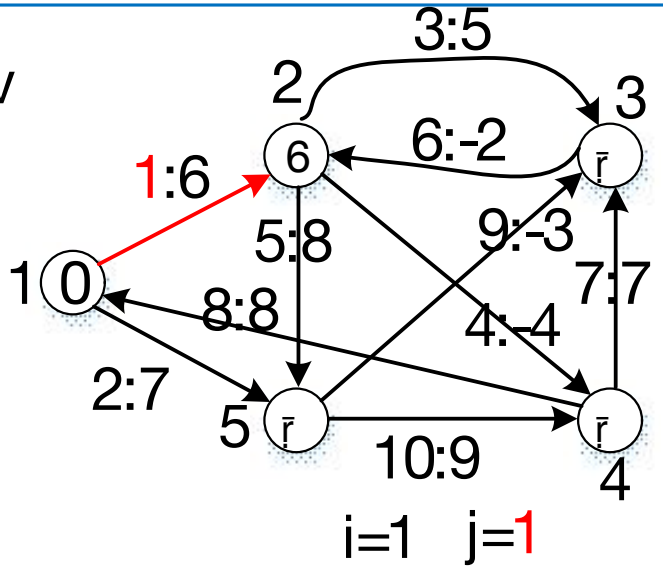
```
Bellman-Ford(G,w,s)
  for v in V:
      dist[v]= ∞; path[v]=NULL;
      dist[s]=0;
  for i from 1 to |V|-1 :
     for each edge (u, v) ∈ E:
         Relax(u,v,w)
  for each edge (u, v) ∈ E:
     if dist[v] > dist[u] + w(u, v)
         return FALSE
   return True
```

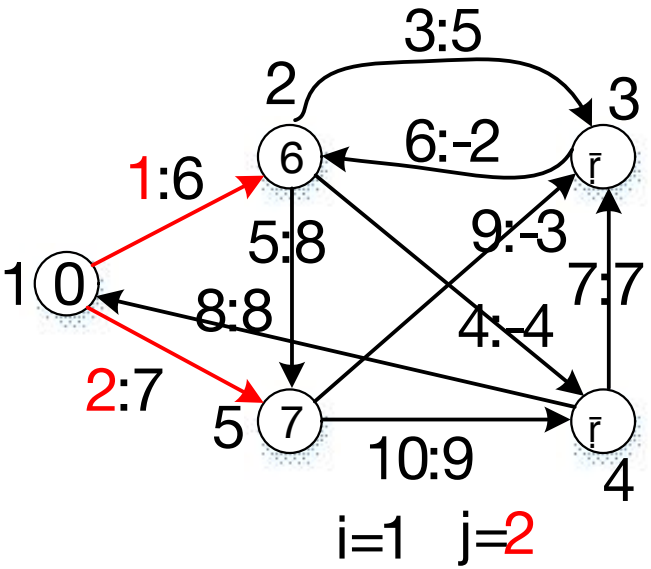


8.5 Bellman-Ford算法

算法伪代码：


```
Bellman-Ford(G,w,s)
  for v in V:
      dist[v]= ∞; path[v]=NULL;
      dist[s]=0;
  for i from 1 to |V|-1 :
     for each edge (u, v) ∈ E:
         Relax(u,v,w)
  for each edge (u, v) ∈ E:
     if dist[v] > dist[u] + w(u, v)
         return FALSE
   return True
```

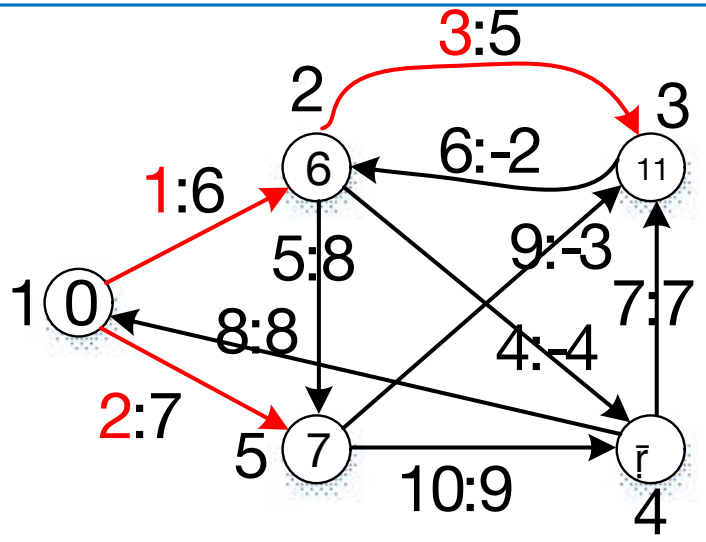


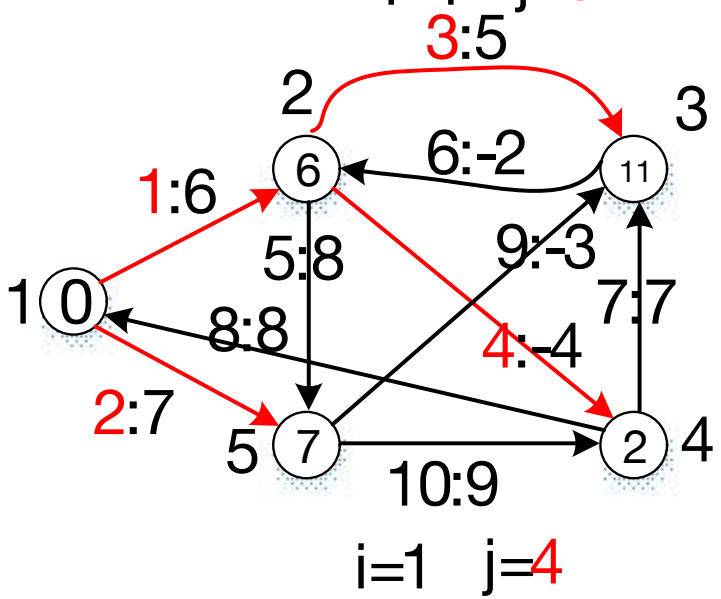


8.5 Bellman-Ford算法

算法伪代码：


```
Bellman-Ford(G,w,s)
  for v in V:
      dist[v]= ∞;path[v]=NULL;
      dist[s]=0;
  for i from 1 to |V|-1 :
     for each edge (u, v) ∈ E:
         Relax(u,v,w)
  for each edge (u, v) ∈ E:
     if dist[v] > dist[u] + w(u, v)
         return FALSE
   return True
```

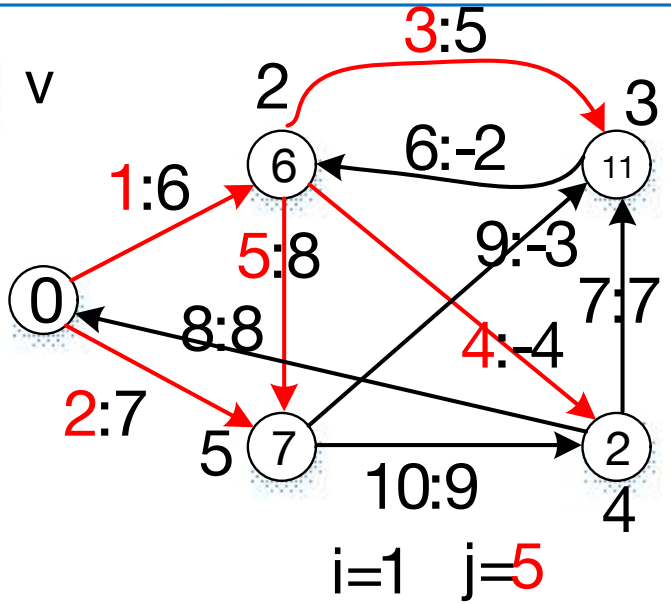


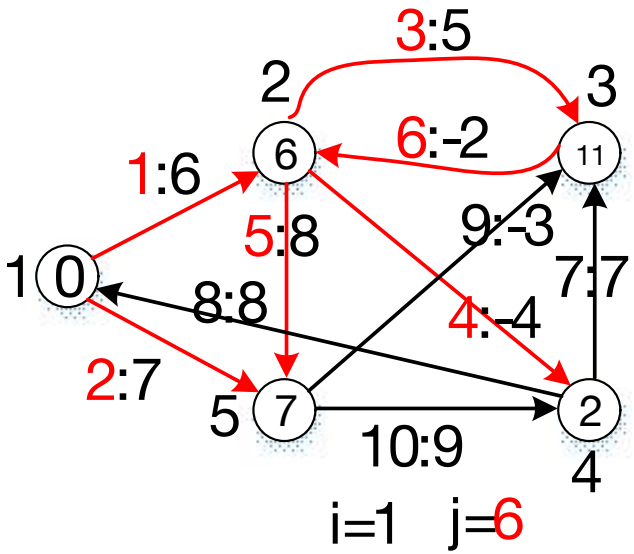


8.5 Bellman-Ford算法

算法伪代码：


```
Bellman-Ford(G,w,s)
  for v in V:
      dist[v]= ∞; path[v]=NULL;
      dist[s]=0;
  for i from 1 to |V|-1 :
     for each edge (u, v) ∈ E:
         Relax(u,v,w)
  for each edge (u, v) ∈ E:
     if dist[v] > dist[u] + w(u, v)
         return FALSE
  return True
```

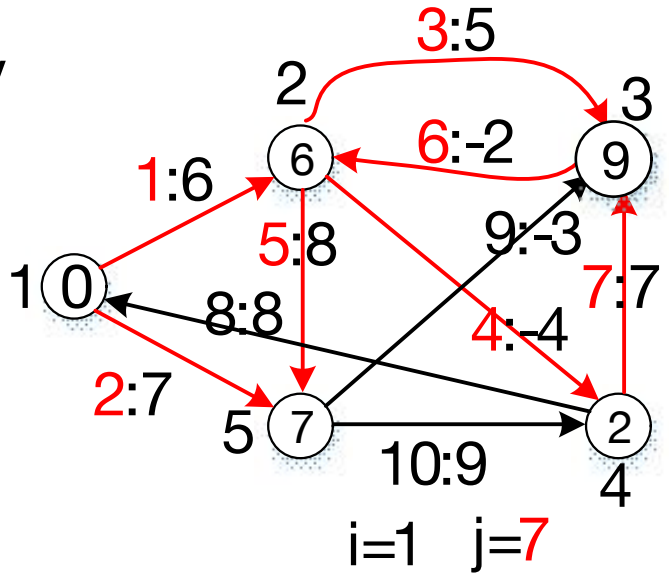




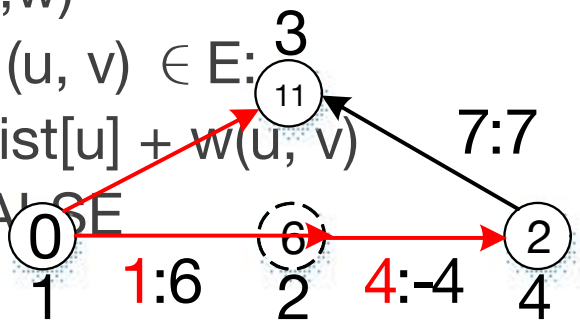
8.5 Bellman-Ford算法

算法伪代码：

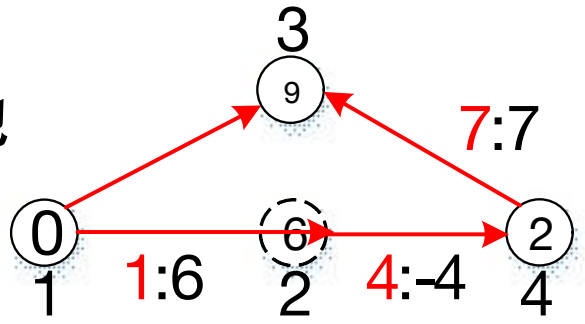

```
Bellman-Ford(G,w,s)
   for v in V:
        dist[v]= ∞;
path[v]=NULL;
        dist[s]=0;
   for i from 1 to |V|-1 :
      for each edge (u, v) ∈ E:
          Relax(u,v,w)
   for each edge (u, v) ∈ E:
      if dist[v] > dist[u] + w(u, v)
          return FALSE
    return True
```



1->3目前最短路径：11



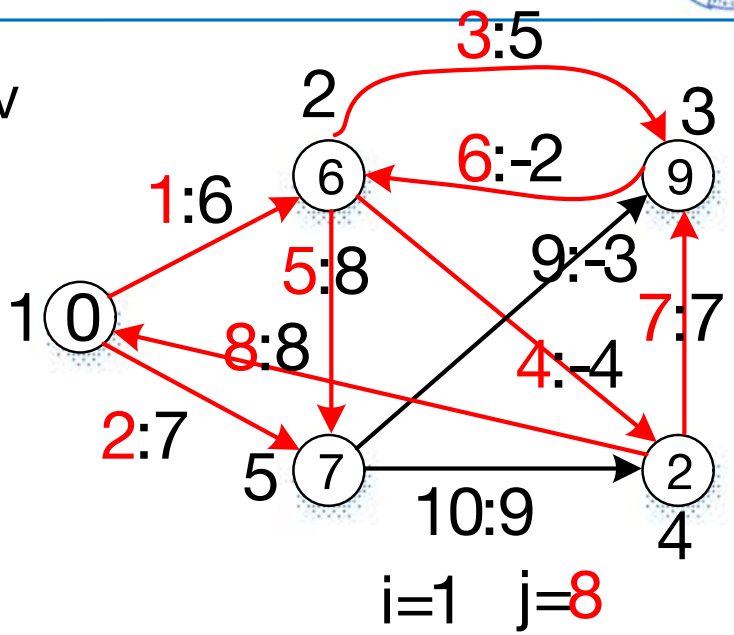
松弛

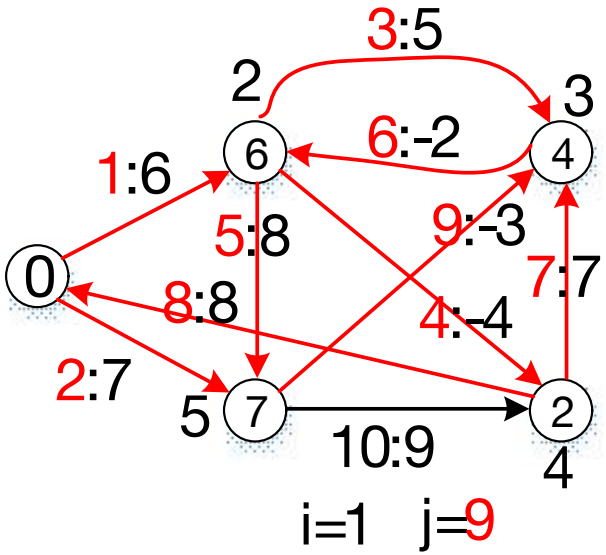


8.5 Bellman-Ford算法

算法伪代码：


```
Bellman-Ford(G,w,s)
  for v in V:
      dist[v]= ∞; path[v]=NULL;
      dist[s]=0;
  for i from 1 to |V|-1 :
     for each edge (u, v) ∈ E:
         Relax(u,v,w)
  for each edge (u, v) ∈ E:
     if dist[v] > dist[u] + w(u, v)
         return FALSE
   return True
```

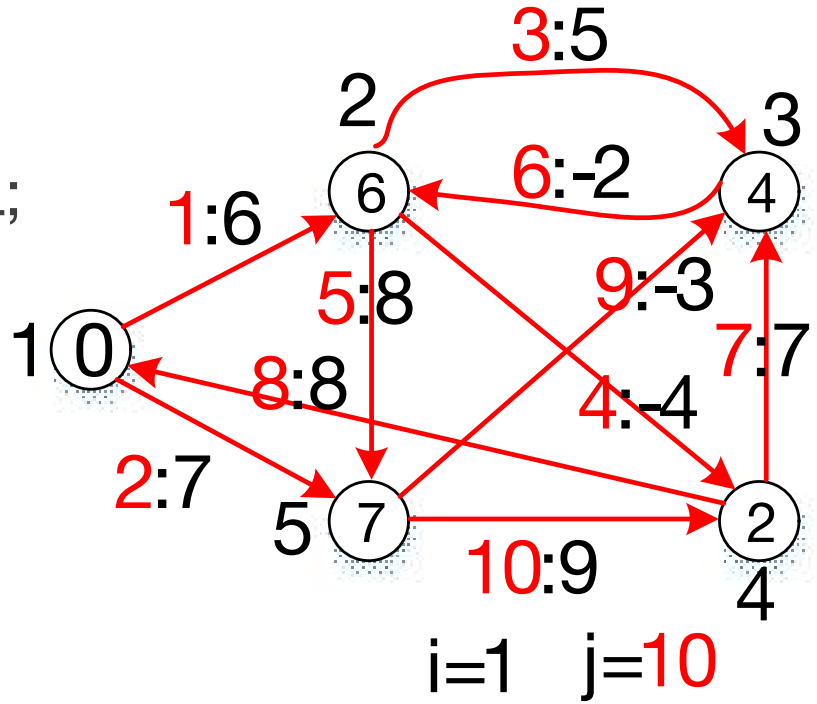




8.5 Bellman-Ford算法

算法伪代码：

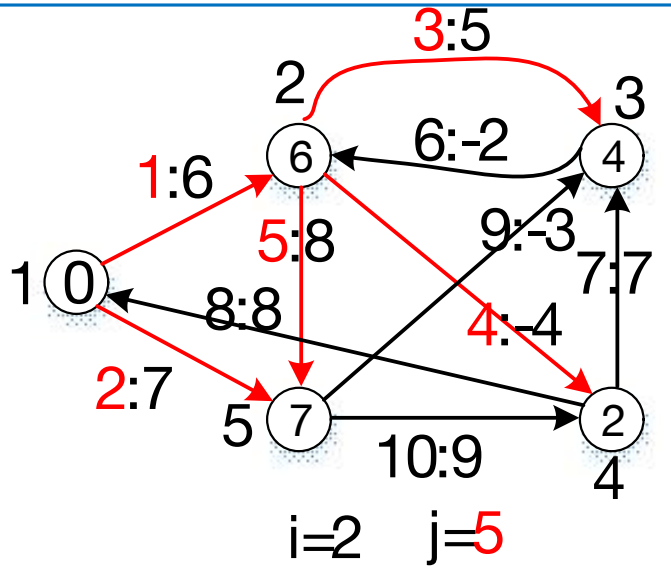

```
Bellman-Ford(G,w,s)
  for v in V:
      dist[v]= ∞; path[v]=NULL;
      dist[s]=0;
  for i from 1 to |V|-1 :
     for each edge (u, v) ∈ E:
         Relax(u,v,w)
  for each edge (u, v) ∈ E:
     if dist[v] > dist[u] + w(u, v)
         return FALSE
   return True
```

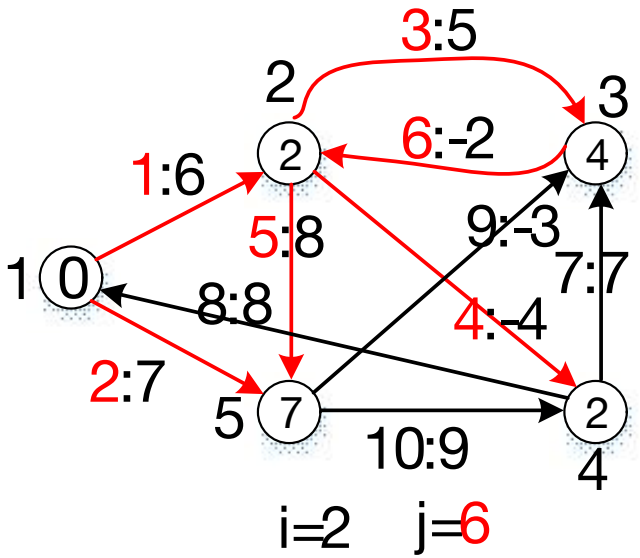


8.5 Bellman-Ford算法

算法伪代码：


```
Bellman-Ford(G,w,s)
  for v in V:
      dist[v]= ∞; path[v]=NULL;
      dist[s]=0;
  for i from 1 to |V|-1 :
     for each edge (u, v) ∈ E:
         Relax(u,v,w)
  for each edge (u, v) ∈ E:
     if dist[v] > dist[u] + w(u, v)
         return FALSE
   return True
```

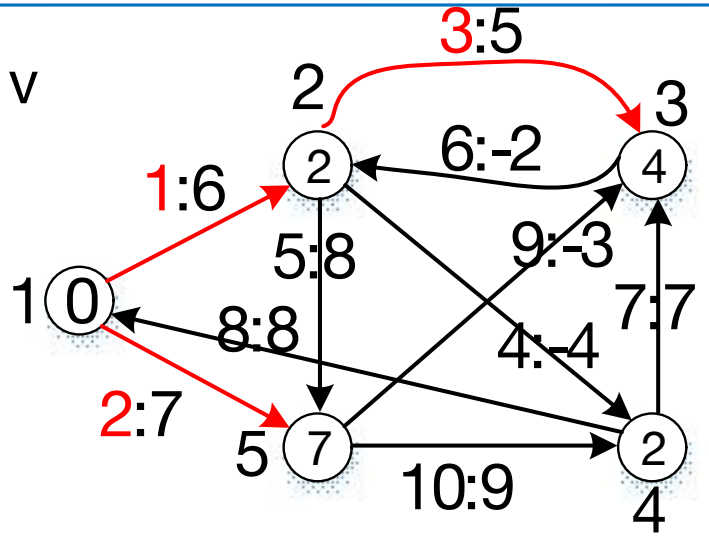


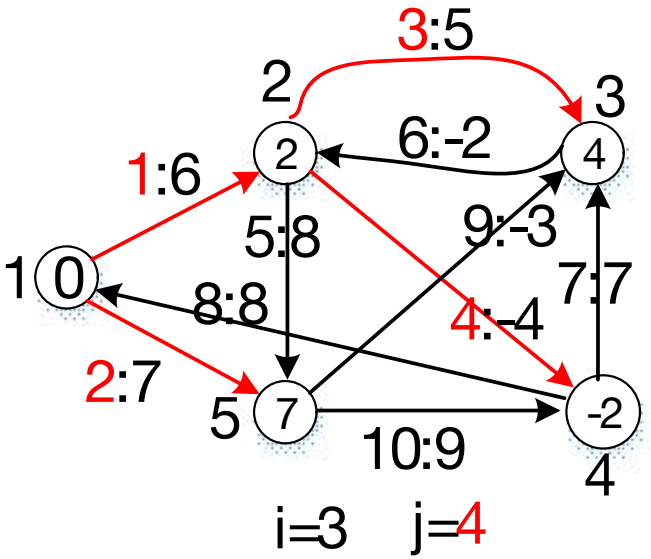


8.5 Bellman-Ford算法

算法伪代码：


```
Bellman-Ford(G,w,s)
  for v in V:
      dist[v]= ∞; path[v]=NULL;
      dist[s]=0;
  for i from 1 to |V|-1 :
     for each edge (u, v) ∈ E:
         Relax(u,v,w)
  for each edge (u, v) ∈ E:
     if dist[v] > dist[u] + w(u, v)
         return FALSE
   return True
```

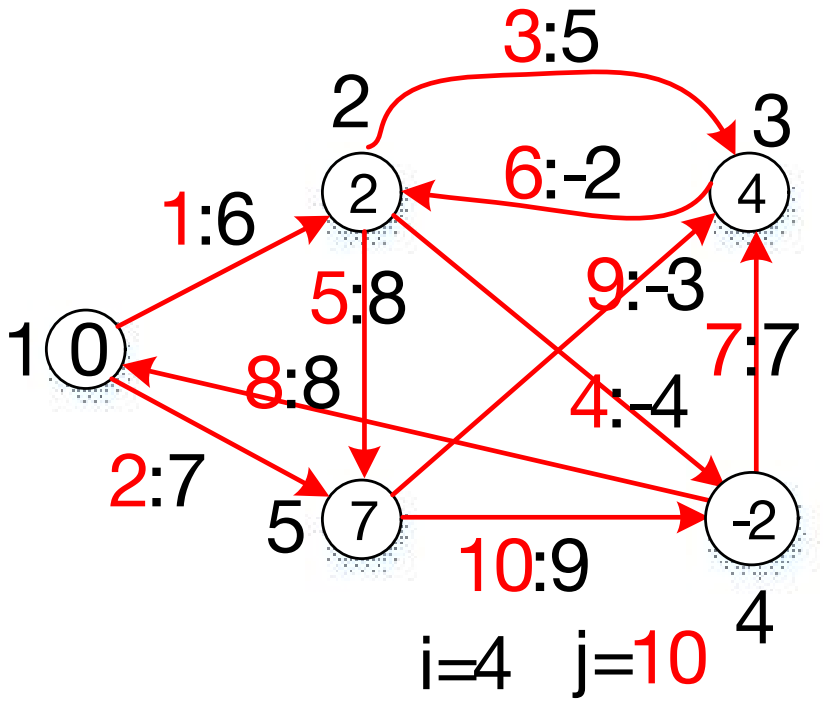




8.5 Bellman-Ford算法

算法伪代码：

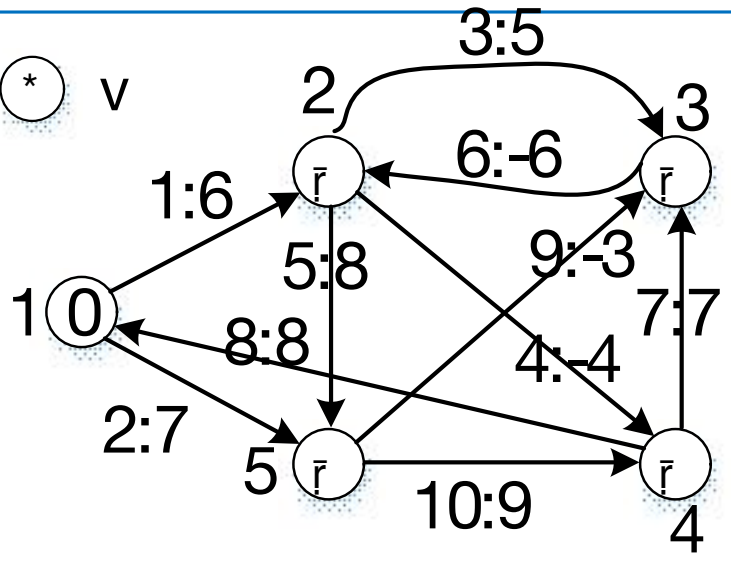

```
Bellman-Ford(G,w,s)
   for v in V:
        dist[v]= ∞;
path[v]=NULL;
        dist[s]=0;
   for i from 1 to |V|-1 :
       for each edge (u, v) ∈ E:
            Relax(u,v,w)
   for each edge (u, v) ∈ E:
       if dist[v] > dist[u] + w(u, v)
            return FALSE
    return True
```

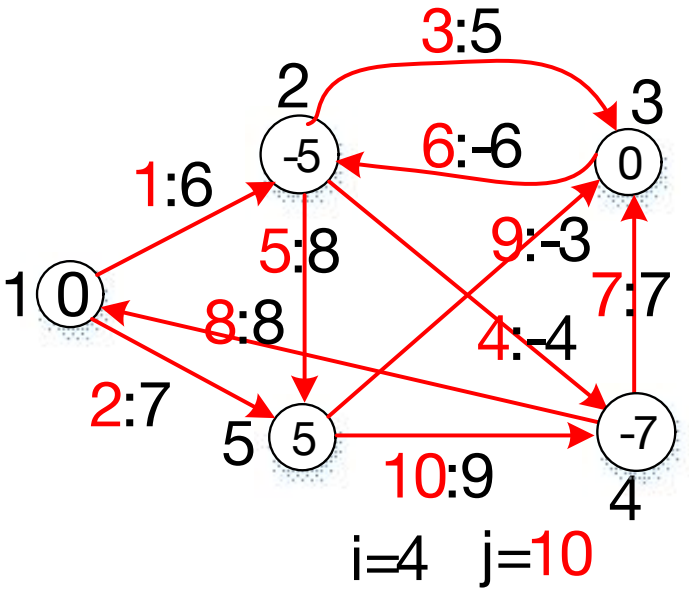


8.5 Bellman-Ford算法

算法伪代码：


```
Bellman-Ford(G,w,s)
  for v in V:
      dist[v]= ∞; path[v]=NULL;
      dist[s]=0;
  for i from 1 to |V|-1 :
     for each edge (u, v) ∈ E:
         Relax(u,v,w)
  for each edge (u, v) ∈ E:
     if dist[v] > dist[u] + w(u, v)
         return FALSE
  return True
```





8.5 Bellman-Ford算法 C++:Graph_Bellman-Ford

算法实现：

```
typedef struct Edge //边
{
    int u, v;
    int cost;
}Edge;

Edge edge[N];
int dis[N], pre[N];

bool Bellman_Ford()
{ }

bool Bellman_Ford()
{}
```

```
/*
测试数据:
算法导论
P379
a.txt
1 2 6
1 5 7
2 3 5
2 4 -4
2 5 8
3 2 -2/-6
4 3 7
4 1 8
5 3 -3
5 4 9
*/
```

```
int main()
{
    scanf("%d%d%d", &nodenum, &edgenum,
&original);
    pre[original] = original;
    for(int i = 1; i <= edgenum; ++i)
    {
        scanf("%d%d%d", &edge[i].u, &edge[i].v,
&edge[i].cost);
    }
    if(Bellman_Ford())
        for(int i = 1; i <= nodenum; ++i) //每个点最短路
        {
            printf("%d\n", dis[i]);
            printf("Path:");
            print_path(i);
        }
    else
        printf("have negative circle\n");
    return 0;
}
```

```
/*
测试数据:
b.txt
1 2 20
1 3 5
4 1 -200
2 4 4
4 2 4
3 4 2
*/
```

8.5 Bellman-Ford算法

算法时间复杂度分析：

```
Bellman-Ford(G,w,s)
  for v in V:
      dist[v]= ∞; path[v]=NULL;
  dist[s]=0;
  for i from 1 to |V|-1 :
      for each edge (u, v) ∈ E:
          if dist[v] > dist[u] + w(u, v)
              then dist[v] =dist[u] + w(u, v)
  for each edge (u, v) ∈ E:
          if dist[v] > dist[u] + w(u, v)
              report negative weight cycle
```

O(|E|)

O(|E|)

O(|V|*|E|)

8.6Bellman-Ford算法 正确性证明：

给定带权图G=(V, E, W)，如果图G中不包含从源点s可达的负权环路，那么上述算法在|V|-1次迭代之后对所有s可达的点v都有dist[v]=δ(*s*, *v*)。

证明：数学归纳法，对于路径长度进行归纳，证明长度为n的最短路径在第n次循环就已经得到

基础步：长度为0的最短路径在第0次循环就已经得到，由于没有负权环路，初始化阶段就有dist[s]=δ(*s*, *s*)=0；

归纳步：长度小于n的最短路径在第n次循环就已经得到，长度为n+1最短路径有两部分组成：长度为n的最短路径和只有一条边e_{n+1}的一条最短路径。

长度为n的最短路径按归纳假设有$v_n.d=\delta(s, v_n)$，于是按照松弛操作有

$v_{n+1}.d=v_n.d+w(e_{n+1})=\delta(s, v_{n+1})$

8.6Bellman-Ford算法 正确性证明：

如果dist[v] 在Bellman-Ford算法的|V|-1次迭代之后还能继续松弛，那么图中有负权环路。

证明：如果在|V|-1次循环之后点v还能继续松弛，那么从s到v的最短路径比|V|-1还要长，那么其中必须有个环路。

这个环路必然权重是负数，否则不会松弛

提 纲

8.1 最短路径问题

8.2 松弛算法

8.3 有向无环图上的最短路径计算

8.4 Dijkstra算法

8.5 Bellman-Ford算法

8.6 差分约束和最短路径

8.7 最短路径问题扩展

8.6差分约束和最短路径

线性规划问题：

研究线性约束条件下线性目标函数的极值问题的数学理论和方法。

定义：给定一个m*n维矩阵A、n维向量c和m维向量b，求一个n维变量矩阵，在$Ax \leq b$的情况下，最大化$\sum_{i=0}^{n} c_i x_i$

8.6差分约束和最短路径

差分约束系统:

对于线性规划问题而言，如果矩阵A中每一行只有一个1和-1且其它值为0。因此，由$Ax \le b$ 所给出的约束条件变为m个涉及n个变量的差额限制条件，其中每个约束条件是如下所示的简单线性不等式:

$$x_i - x_j \le b_k$$

这里$1 \le i \quad j \le n \quad 1 \le k \le m$

8.6差分约束和最短路径

差分约束系统:

For example, consider the problem of finding a 5-vector $x = (x_i)$

$$\begin{pmatrix} 1 & -1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 & -1 \\ -1 & 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} \leq \begin{pmatrix} 0 \\ -1 \\ 1 \\ 5 \\ 4 \\ -1 \\ -3 \\ -3 \end{pmatrix}.$$

8.6差分约束和最短路径

差分约束系统：

$$x_1 - x_2 \le 0$$

$$x_1 - x_5 \le -1$$

$$x_2 - x_5 \le 1$$

$$x_3 - x_1 \le 5$$

$$x_4 - x_1 \le 4$$

$$x_4 - x_3 \le -1$$

$$x_5 - x_3 \le -3$$

$$x_5 - x_4 \le 3$$

8.6差分约束和最短路径

差分约束系统:

$$x_1 - x_2 \le 0$$

$$x_1 - x_5 \le -1$$

$$x_2 - x_5 \le 1$$

$$x_3 - x_1 \le 5$$

$$x_4 - x_1 \le 4$$

$$x_4 - x_3 \le -1$$

$$x_5 - x_3 \le -3$$

$$x_5 - x_4 \le 3$$

可能解:

x=(-5,-3,0,-1,-4)

x'=(0,2,5,4,1)

8.6差分约束和最短路径

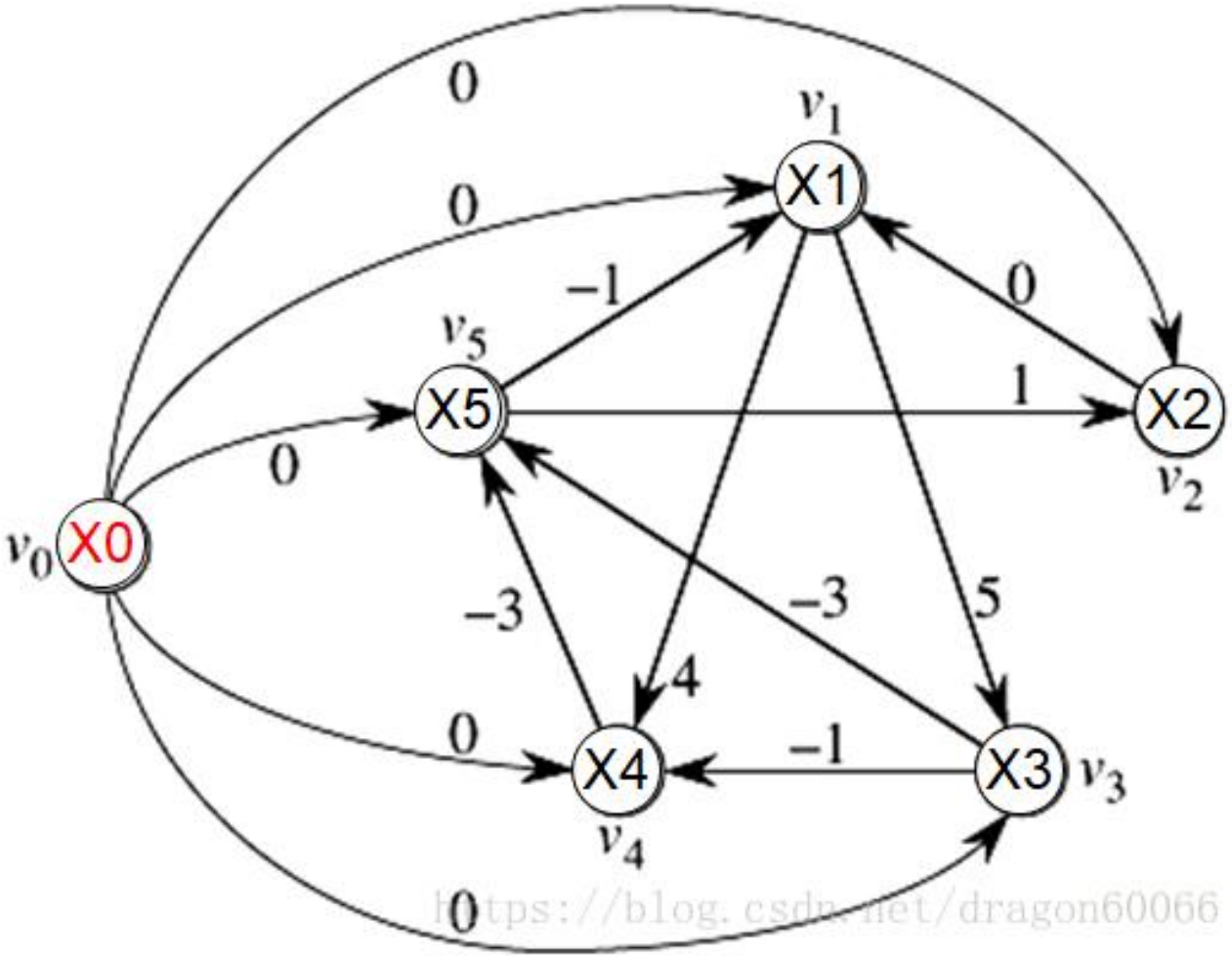
引理1：

如果存在一组解 $\{x_1,x_2,..,x_n\}$的话，那么对于任何一个常数 d 有 $\{x_1+d,x_2+d,..,x_n+d\}$也肯定是一组解，因为任何两个数加上一个数以后，它们之间的关系（差）是不变的，这个差分约束系统中的所有不等式都不会被破坏。

证明见算法导论P388

8.6差分约束和最短路径

约束图：



X1 - X0 <= 0
X2 - X0 <= 0
X3 - X0 <= 0
X4 - X0 <= 0
X5 - X0 <= 0

8.6差分约束和最短路径

定理（算法导论P389）：

如果限制图中没有负权环路，那么上述算法求出的最短路径距离就是一个差分约束系统的解。

证明：对于任意一条限制图上的边(x_i, x_j)，根据三角不等式，我们有$\delta(s, x_i) + w_{ij} \geq \delta(s, x_j)$，于是$\delta(s, x_i) - \delta(s, x_j) \leq w_{ij}$，即差分约束

8.6差分约束和最短路径

定理（算法导论P389）：

如果限制图G中有负权环路，那么这个差分约束系统没有结果

证明： 假设存在一条负权环路 $v_0e_1v_1e_2 \ldots e_nv_n\ e_{n+1}v_0$且$x_i$对应$v_i$，那么

$$x_0 - x_1 \le w_{01}$$

$$x_1 - x_2 \le w_{12}$$

……

$$x_n - x_0 \le w_{n0}$$

上面的不等式左右分别求和，就有$0 \le w_{01} + w_{12} + \ldots + w_{n0}$，由

8.6差分约束和最短路径

求解差分约束系统：

基于Bellman-Ford算法的算法。首先，在限制图G的基础上，构造一个图G’，引入一个源点s，s到所有点都有一条边，长度都为0

然后，对G’进行Bellman-Ford算法得到s到所有点的最短路径距离

这些距离就是x_i.d就是一个解

复杂度为O(m*n)

提 纲

8.1 最短路径问题

8.2 松弛算法

8.3 有向无环图上的最短路径计算

8.4 Dijkstra算法

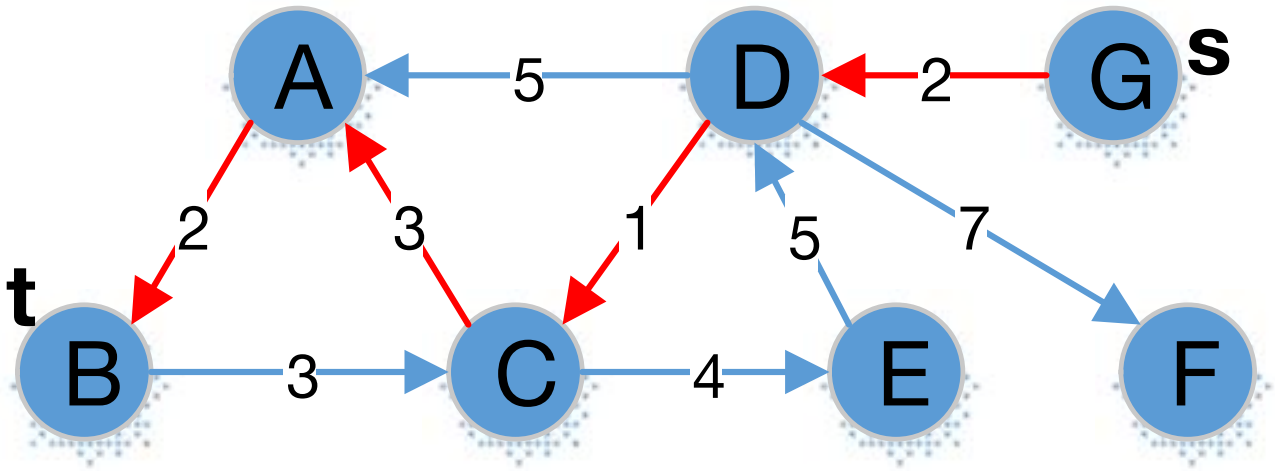
8.5 Bellman-Ford算法

8.6 差分约束和最短路径

8.7 最短路径问题扩展

问题定义

给定一个带权图G=(V, E, W)和两个点s和t， 计算G上从s到t的距离$\delta(s, t)$



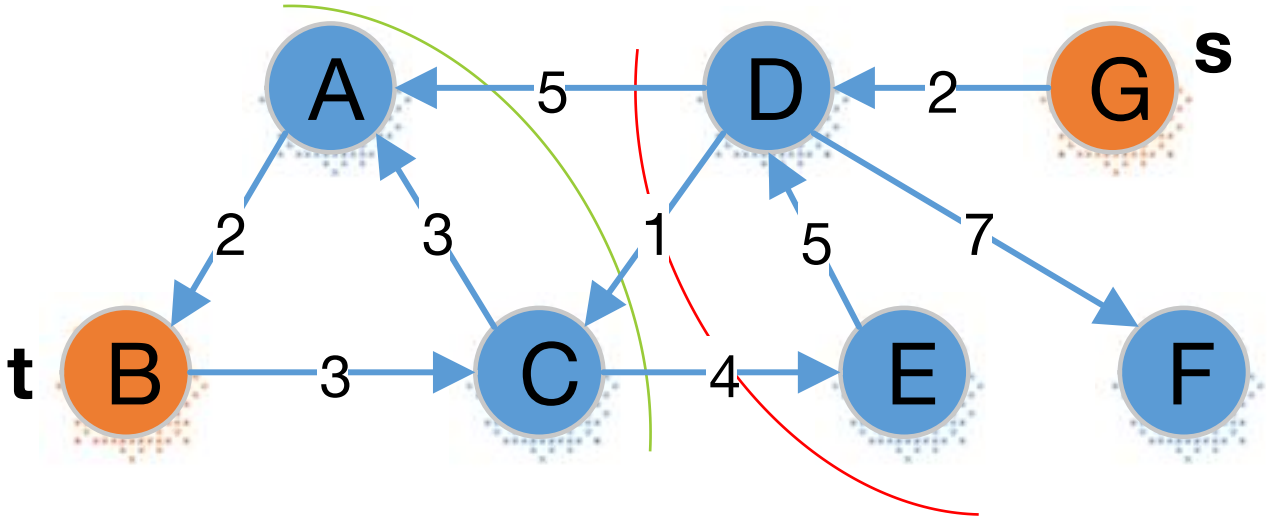
基本算法

```
Dijkstra(G, Adj, s)
  Initialize();
  Q=V;
  while Q ≠ ∅:
    u = EXTRACT-MIN(Q); S = S ∪ {u};
    if u==t
        return δ(s, t)
    for each v ∈ Adj[u]:
      Relax(u, v, G)
```

遇到t的时候直接终止

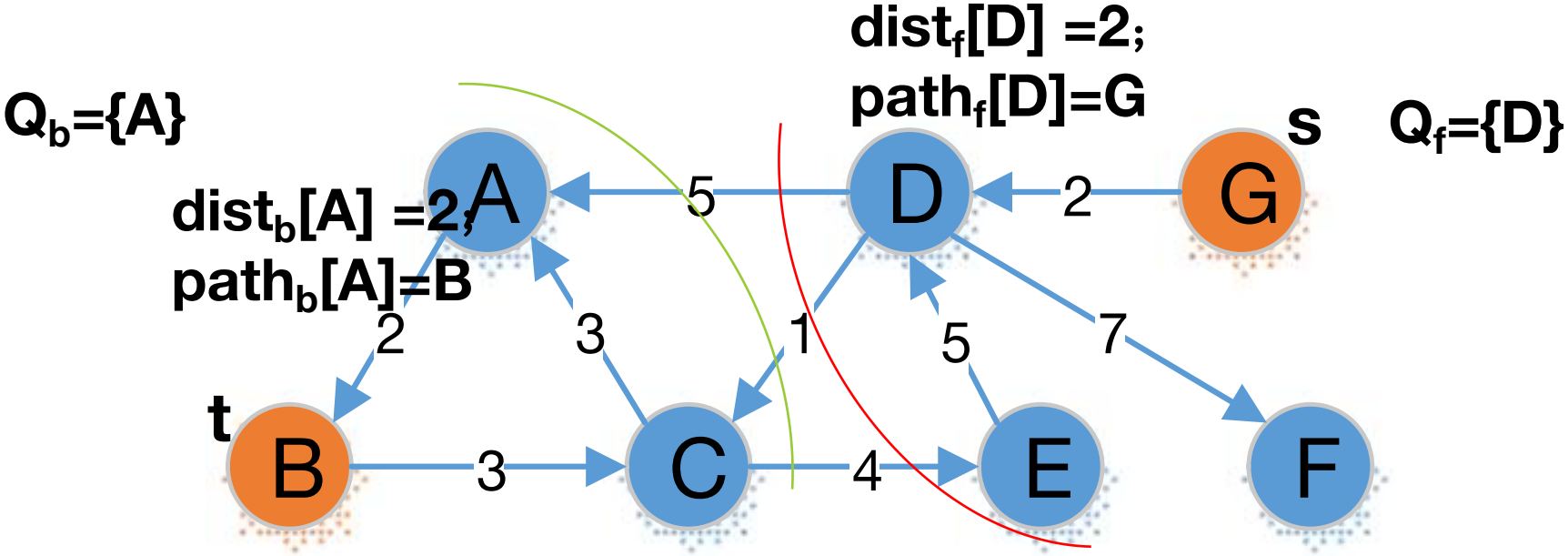
双向Dijkstra算法

给定一个带权图G=(V, E, W)和两个点s和t，从s向前做搜索（Forward Search）的同时交替地进行从t开始的反向搜索（Backward Search）



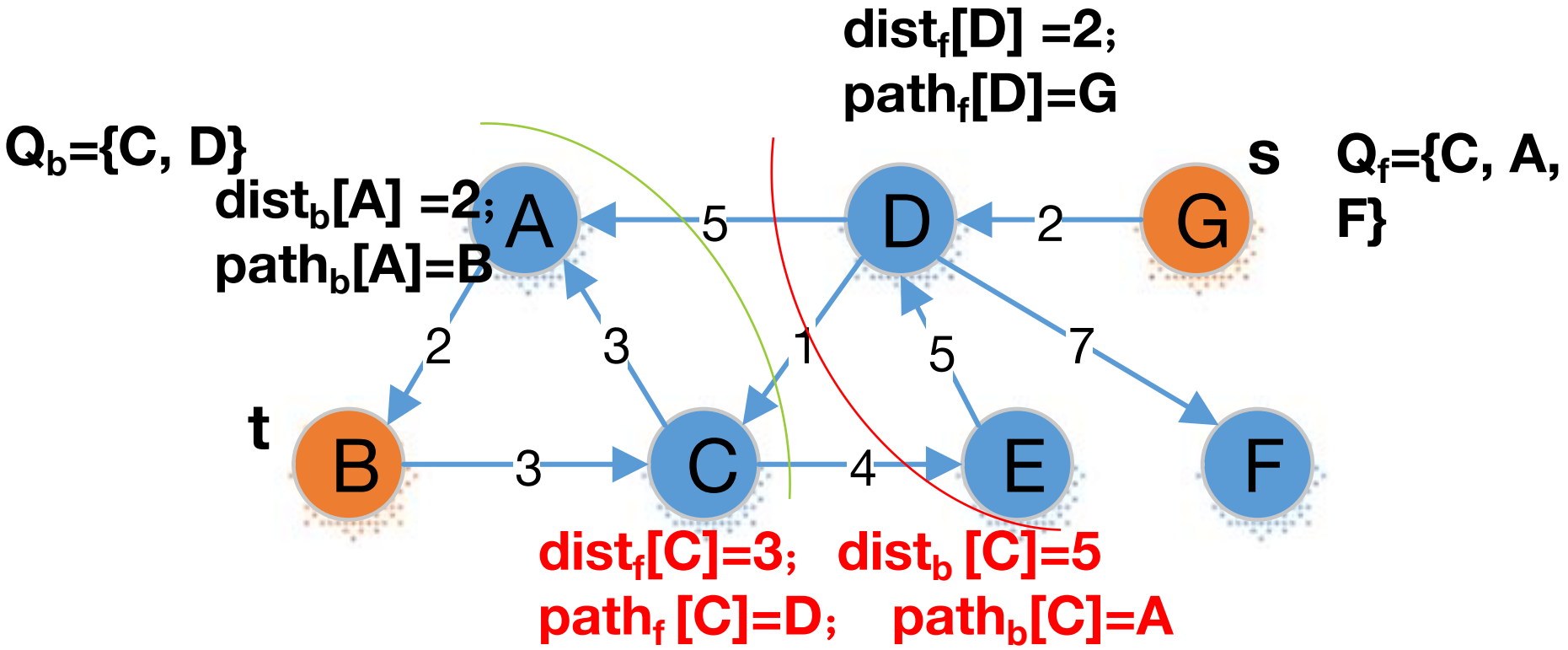
相关定义

给定一个点v，$dist_f[v]$ 和$dist_b[v]$ 定义为前向搜索和后向搜索过程中s到v和v到t的距离；$path_f[v]$和$path_b[v]$定义为前向搜索和后向搜索过程中s到v和v到t的路径上的前驱结点；Q_f和Q_b分别为前向搜索和后向搜索过程中优先队列



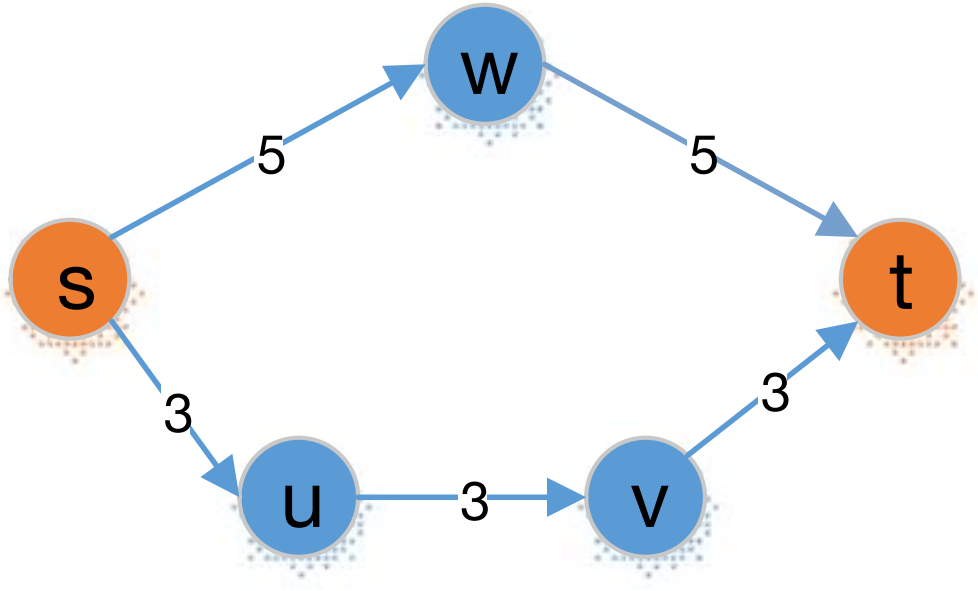
算法中止条件

存在一个点w从Q_f和Q_b中都被删除的时候，算法停止



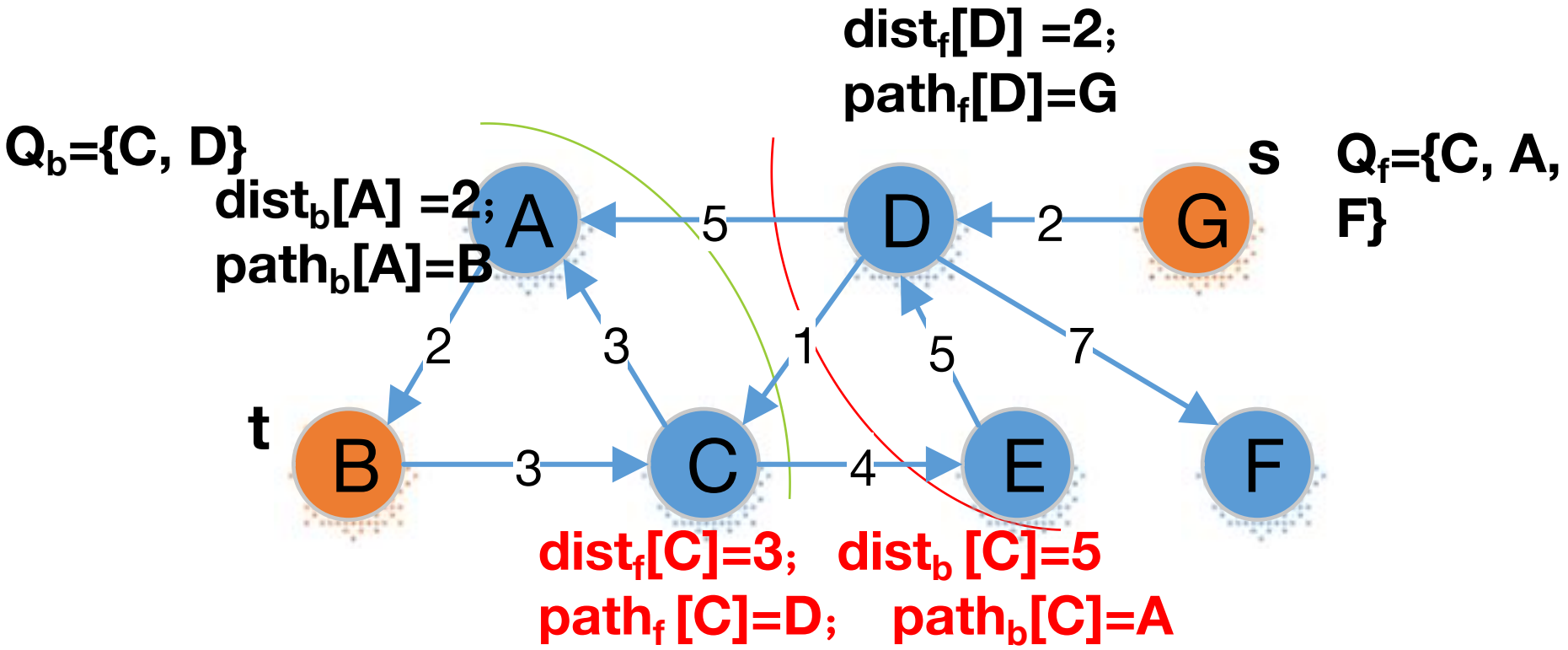
最终解计算

在点w从Q_f和Q_b中都被删除的时候，直接从w出发得到最终解是不行的



正确的最终解

正确的最终解计算方法应该是从Q_f和Q_b中找出连接s和t的最小路径来返回，也就是返回：$\min_{x \in Q_f \cup Q_b} \{dist_f[x] + dist_b[x]\}$



基于Landmark的优化

可以按照之前的边重赋权方法对所有边进行重赋权，即找出一个函数h，对于任意边(u,v)，我们有

$$w_h(u,v) = w(u,v) + h(v) - h(u)$$

于是，对于两点s和t而言，它们之间的路径p有

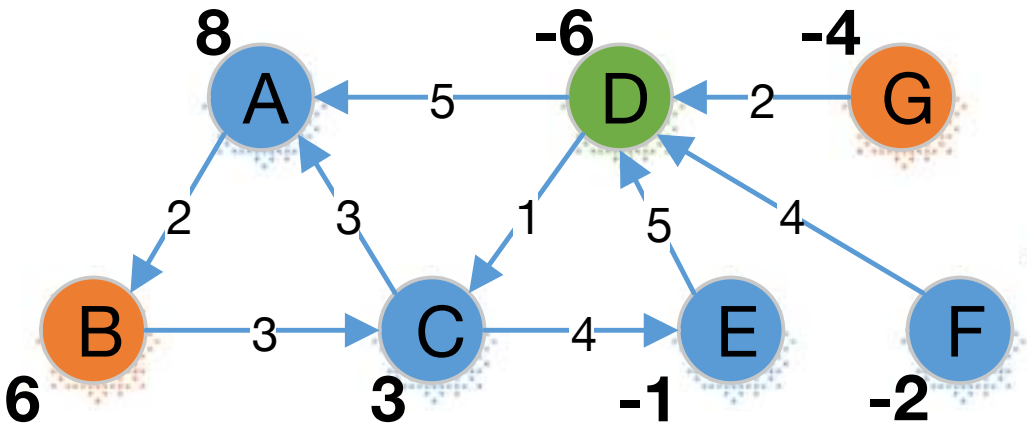
$$w_h(p) = w(p) + h(t) - h(s)$$

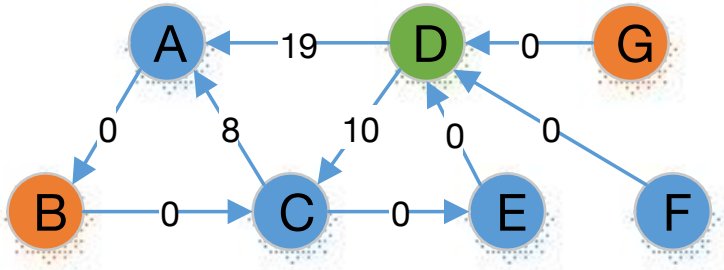
基于Landmark的优化

之后，找出一个landmark l来定义h以加快Dijkstra算法计算过程，即

$$h(u) = \delta(u, l) - \delta(l, t)$$

所有$\delta(u, l)$和$\delta(l, t)$可以在预处理阶段事先算好


8
-6
-4
A
D
G
5
2
2
3
1
5
4
B
C
E
F
3
4
6
3
-1
-2


A
D
G
19
0
0
8
10
0
0
B
C
E
F
0
0

基于Landmark的距离估计

Michalis Potamias, Francesco Bonchi, Carlos Castillo, Aristides Gionis: Fast shortest path distance estimation in large networks. CIKM 2009: 867-876

Andrey Gubichev, Srikanta J. Bedathur, Stephan Seufert, Gerhard Weikum: Fast and accurate estimation of shortest paths in large graphs. CIKM 2010: 499-508

Miao Qiao, Hong Cheng, Lijun Chang, Jeffrey Xu Yu: Approximate Shortest Distance Computing: A Query-Dependent Local Landmark Scheme. ICDE 2012: 462-473

Mingdao Li, Peng Peng, Yang Xu, Hao Xia, Zheng Qin:Distributed Landmark Selection for Lower Bound Estimation of Distances in Large Graphs. APWeb/WAIM (1) 2019: 223-239

问题定义

给定一个无向图G=(V, E)，所谓landmark，就是一些特定的点。我们记录并保存所有点到landmark的距离，进而对于每个点v形成一个向量

$$\phi(v) =< \delta(v, l_1), \delta(v, l_2), ..., \delta(v, l_n) >$$

对于两个查询点s和t而言，显然有

$$\delta_{\text{approx}}(s,\ t\ ) = \min_{1 \le k \le n} \{\delta(s, l_k) + \delta(t, l_k)\} \ge \delta(s,\ t\ )$$

$$\delta_{\text{approx}}(s,\ t\ ) = \max_{1 \le k \le n} \{|\delta(s, l_k) - \delta(t, l_k)|\} \le \delta(s,\ t\ )$$

Potamias et.al@CIKM2009

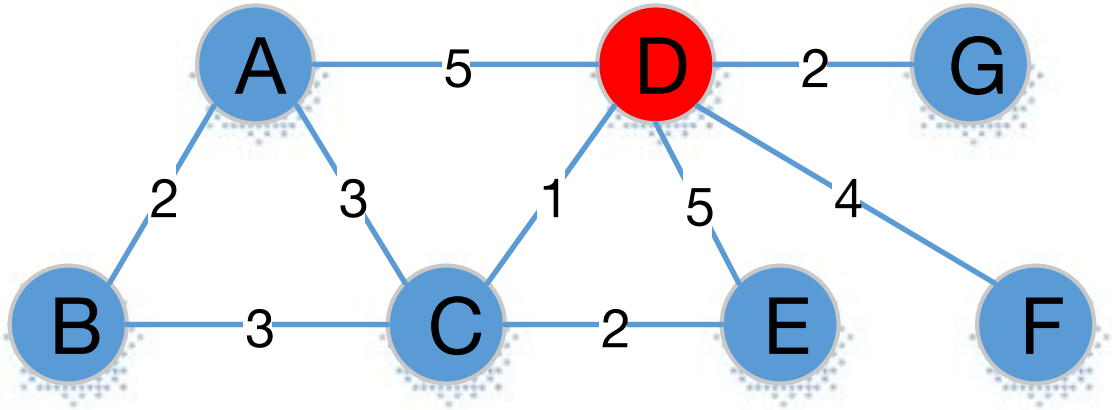
本文主要讨论了lankmark的选取策略问题

本文着重介绍了基于度中心度和介数中心度的两种策略

度中心度

度中心度（Degree Centrality）通过点的邻居数目来计算节点的重要程度

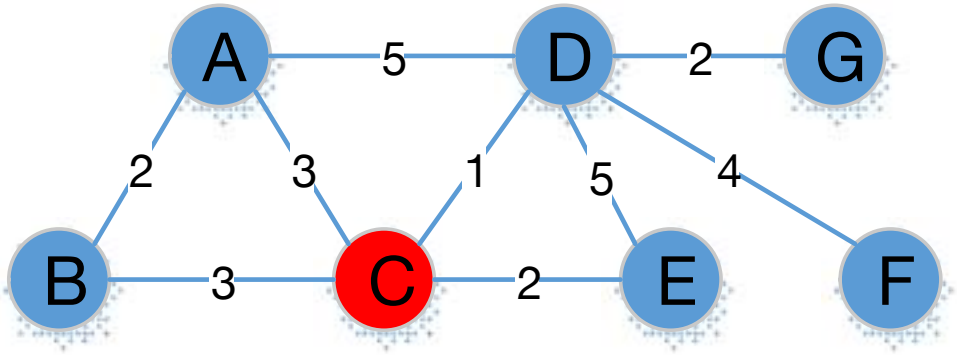
给定图$G = (V,E)$，点v的度中心度计算公式为：$C_D = \deg(v)$



介数中心度

介数中心度（Betweenness Centrality）定义为G 中除了v 以外的点对间最短路径有多少条最短路径经过了点v

介数中心度的计算公式为：$C_B = \sum_{s \neq v \neq t \in V} \frac{\pi_{st}(v)}{\pi_{st}}$，其中$\pi_{st}$表示s到t的最短路径，$\pi_{st}(v)$表示s到t经过v的最短路径

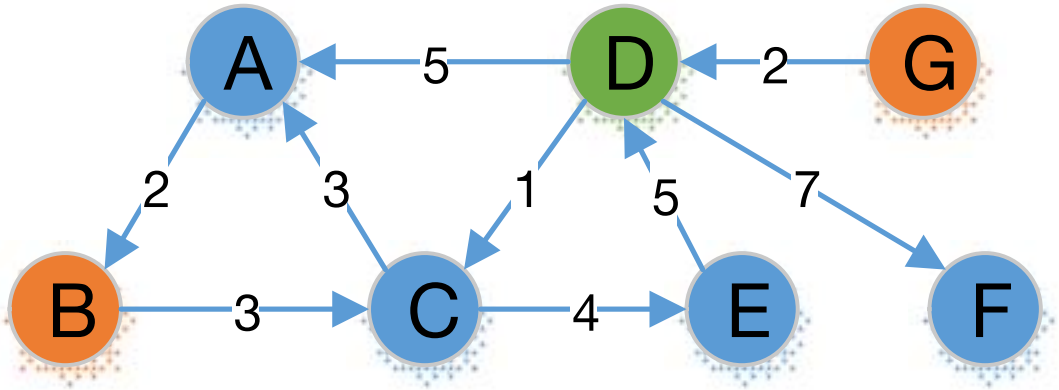


Gubichev et.al@CIKM2010

本文主要讨论了基于landmark进一步提高距离预测准确度的方法

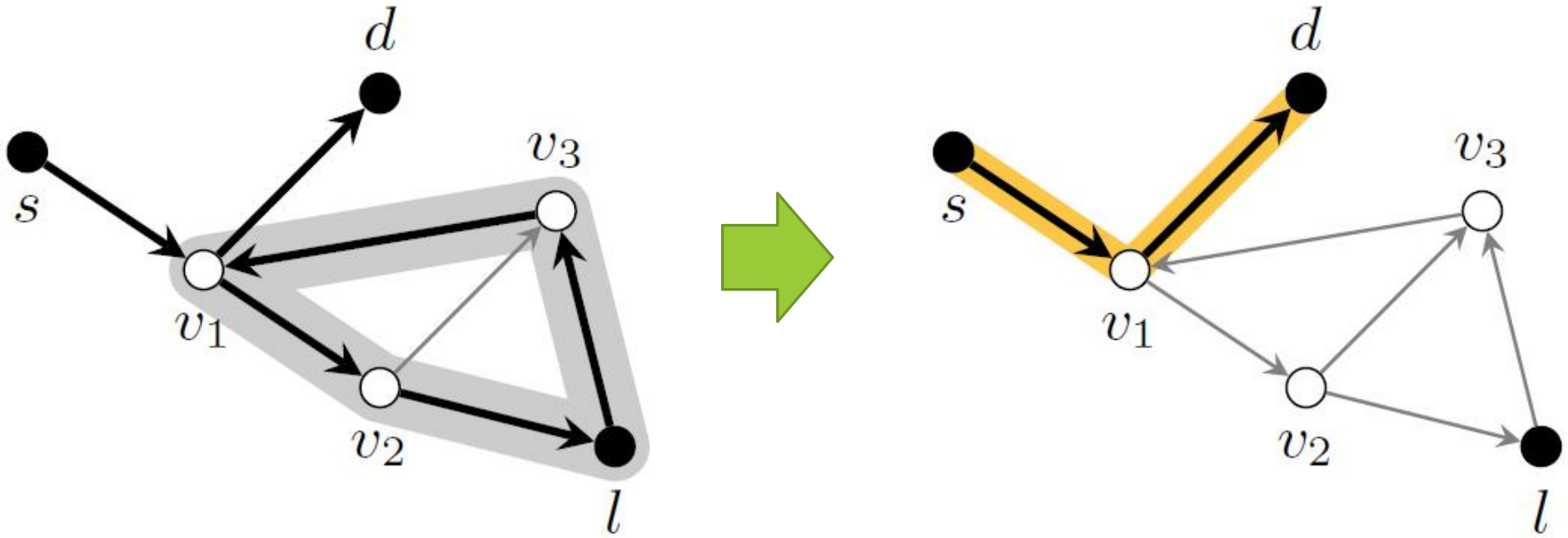
在预处理阶段，预计算出所有Landmark的最短路径树

在查询处理阶段，π_{st}就等于π_{sl}和π_{lt}的拼接



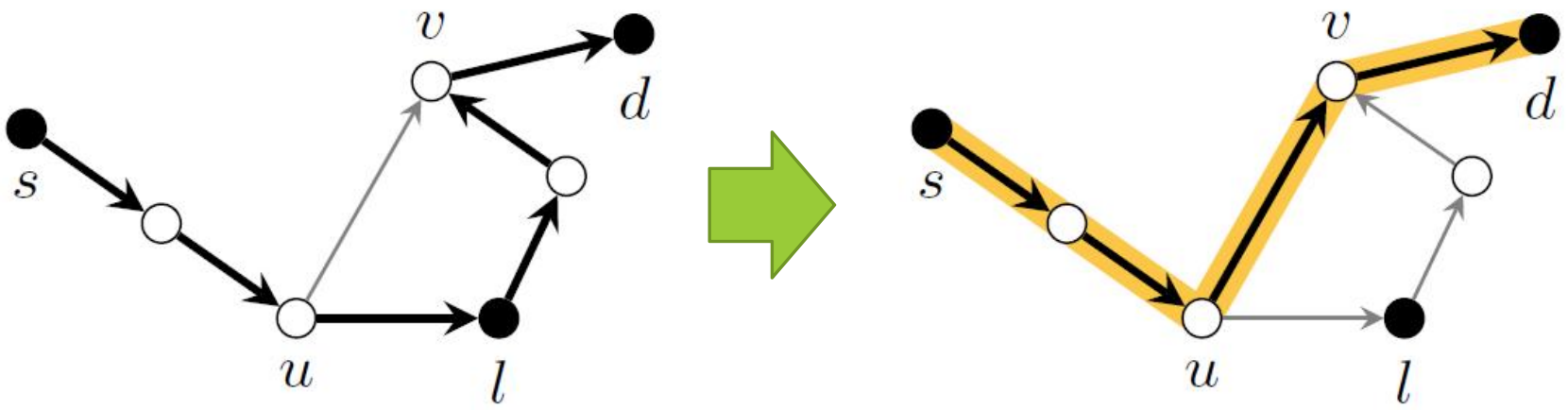
Cycle Elimination

$\pi_{s,l}$和$\pi_{l,t}$的拼接过程中消除期间的环



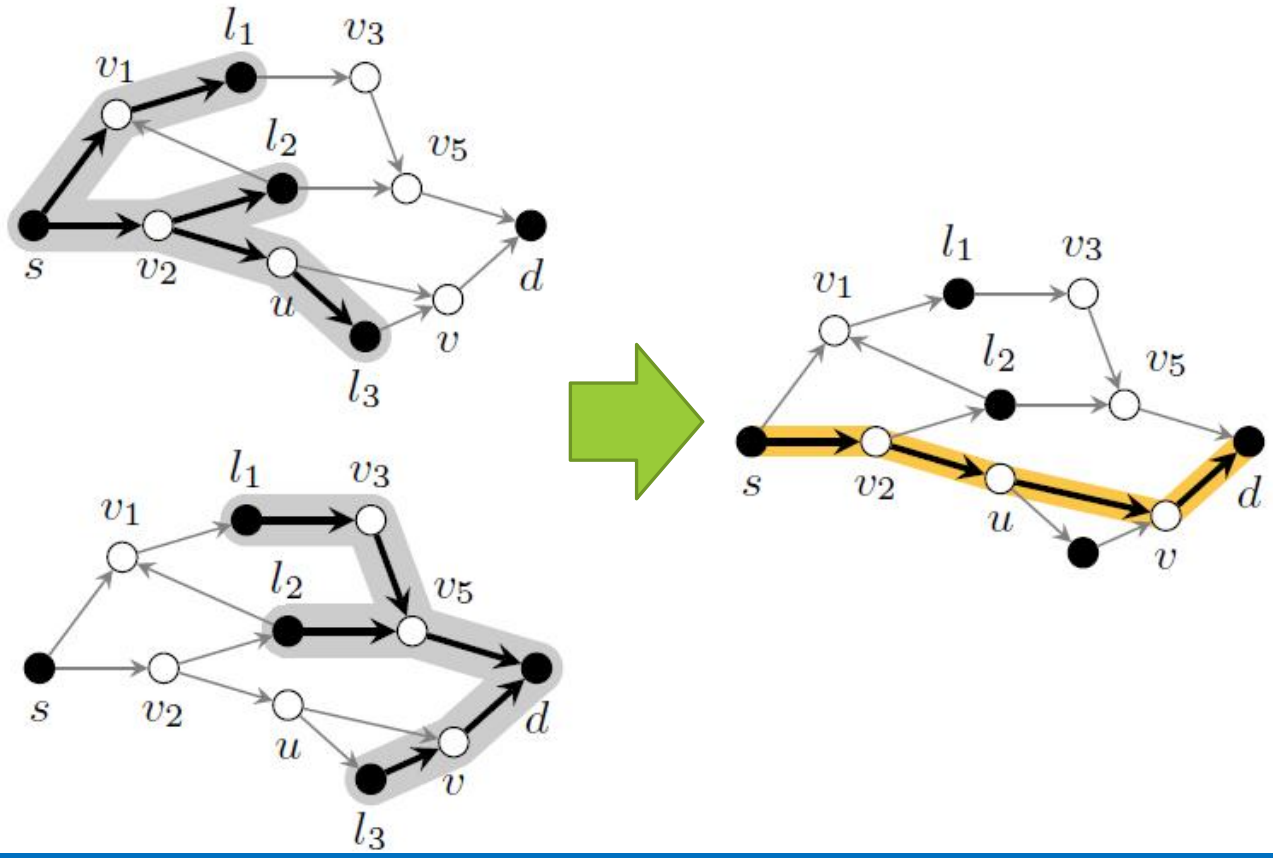
Shortcutting

π_{sl}和π_{lt}的拼接过程中找出期间的捷径



Tree Algorithm

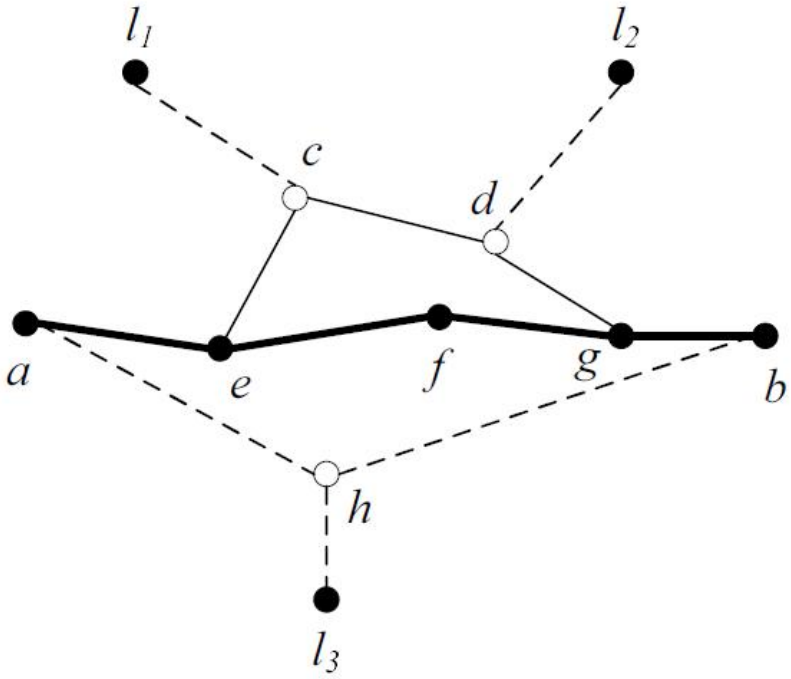
π_{sl}和π_{lt}的拼接过程中从最短路径树上找出跨树的路径



Qiao et.al@ICDE2012

本文再进一步讨论了如何进一步构建索引提高效率

具体而言，本文重点讨论了基于最短路径树上最近公共祖先的优化

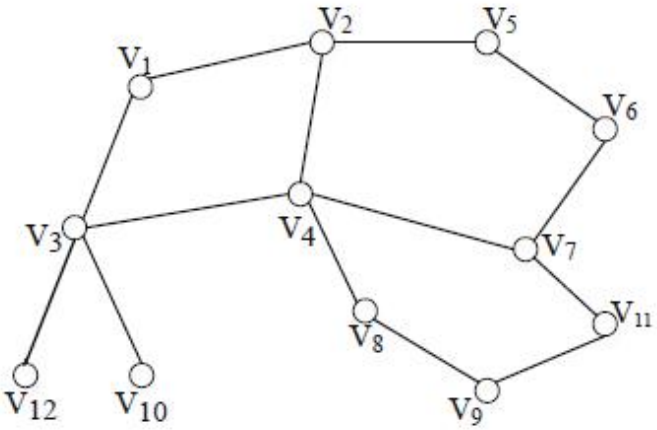


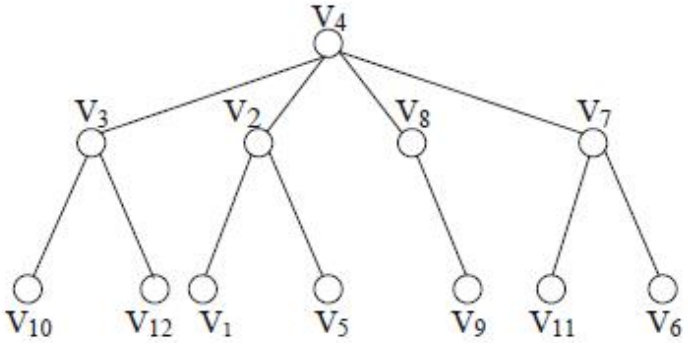
Li et.al@APWeb/WAIM 2019

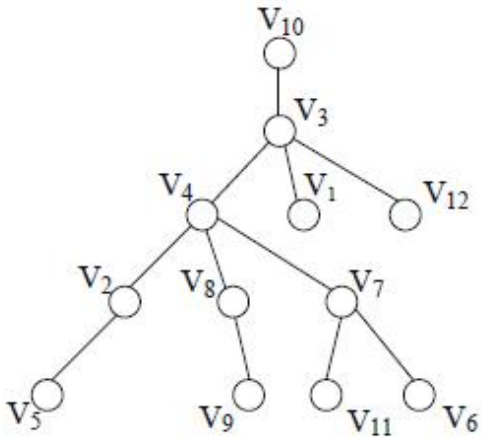
本文讨论了如何挑选Landmark利用三角不等式进行下界估计

$$\delta_{approx}(s,\ t) = \max_{1 \le k \le n} \{|\delta(s, l_k) - \delta(t, l_k)|\} \le \delta(s,\ t)$$

具体而言，我们应该选择尽量在更多最短路径延长线上的点作为landmark







谢谢!

hnu16pp@foxmail.com