

数据结构与算法

DATA STRUCTURE

第二十二讲 最短路径

胡浩栋

信息管理与工程学院

2018 - 2019 第一学期

课堂内容

- 最短路径算法

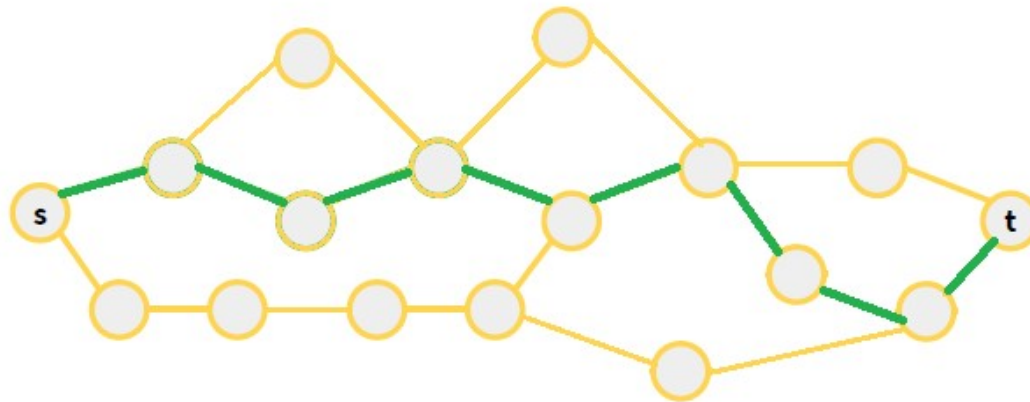
最短路径算法

最短路径

- BFS算法可以找到图上最短路径，如果边没有权重
- 在加权图上，最短路径的算法是
 - **Dijkstra算法**：适用没有负边的图
 - **Bellman-Ford算法**：适用于没有负环的图（可以有负边）
 - 在**DAG**上，最短路径算法是线性的（可以有负边，但无环）
- 在无向加（整数）权图上，有线性的最短路径算法

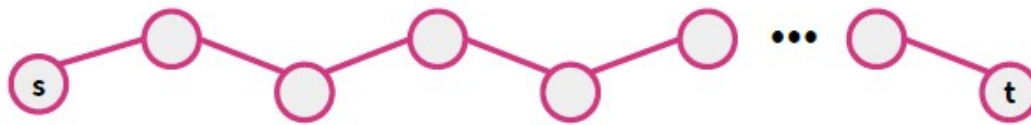
最短路径问题

- 不妨假设这里说的图都是有向图，无向图的边可以看作双向边（双向边的话箭头省略）
- 如果从有向图中某一顶点（称为源点）到达另一顶点（称为终点）的路径可能不止一条，如何找到一条路径使得沿此路径上各边的权值总和达到最小？



直接结论

- 图有**负环**（cycle的边权重和是负数），没有最短路径
 - **Bellman-Ford**算法可以检测到负环的存在
- 如果没负环，最短路径存在，而且一定是**简单路径**
 - 即最短路径上不存在环
- 最短路径的**子路径**一定也是最短路径

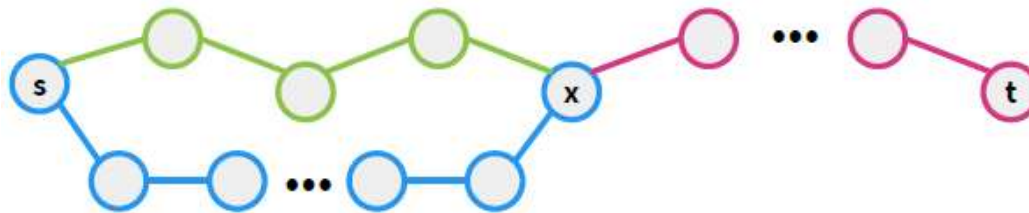


最短路径的子路径也是最短路径

- 假设x是从s到t的最短路径的中间节点，那么s到x的子路径也是最短路径

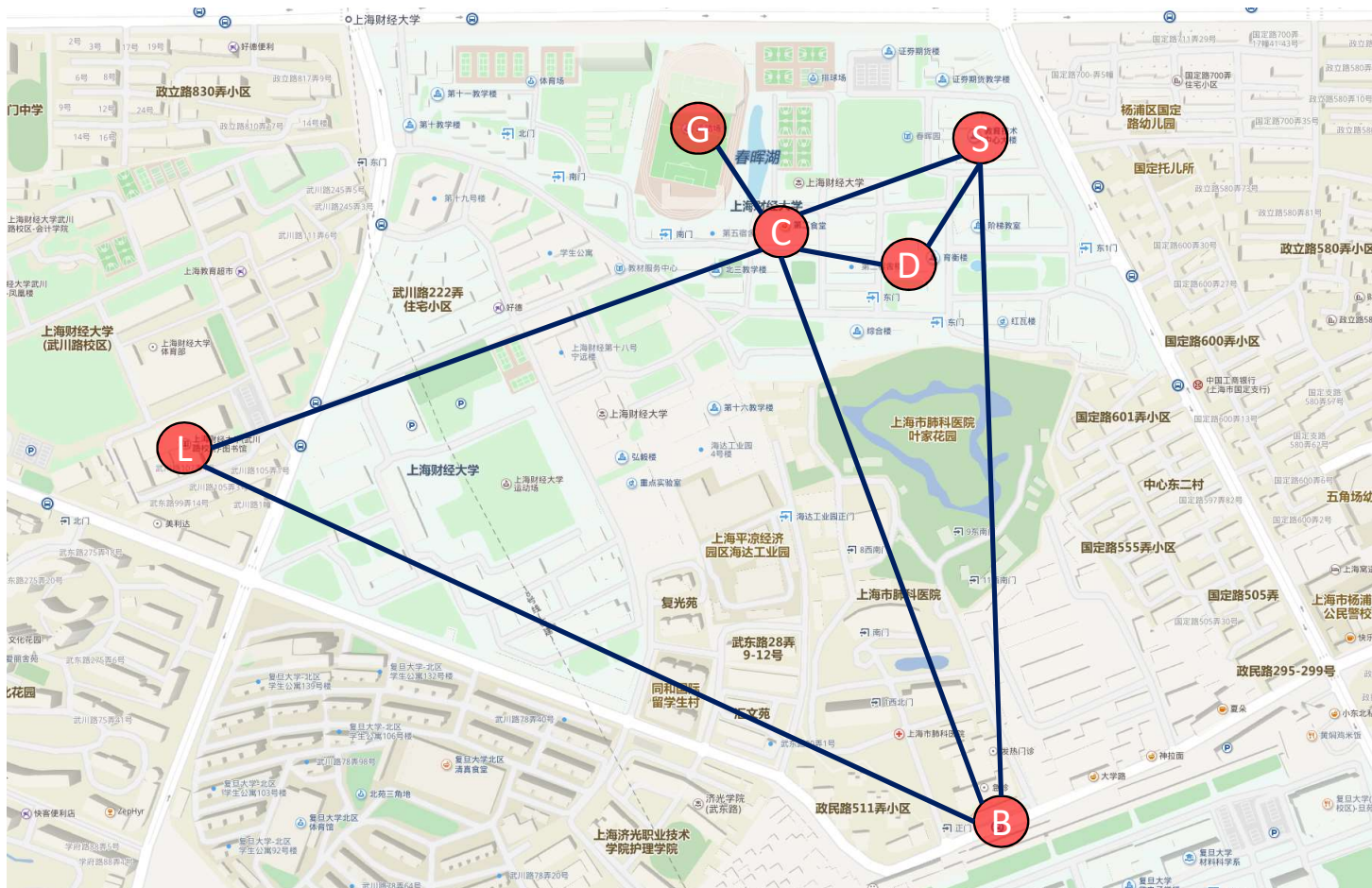


- 反证法：假如s到x的真正最短路径不是绿色路径
那么s到t的最短路径可以更短（蓝色+紫色）



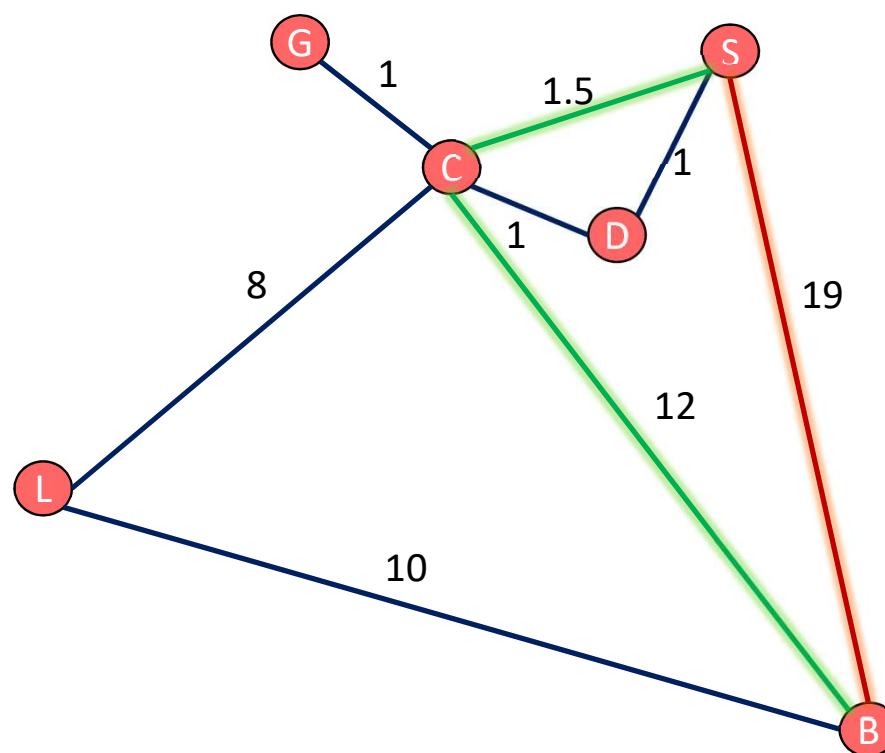
非负边的最短路径算法

如何从上课地点到食堂/宿舍/体育馆/图书馆/车站



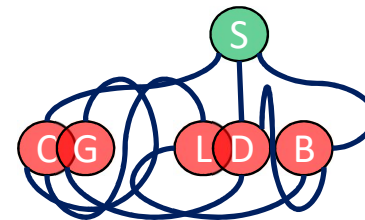
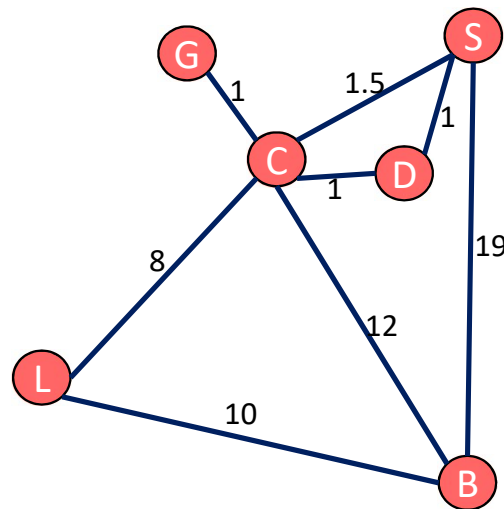
抽象成图，找从s出发的最短路径

- 如果按BFS算法，从S到B直接去最短
- 实际加权图上应该是S→C→B最短
- 这里边权重看作距离，都是大于0的



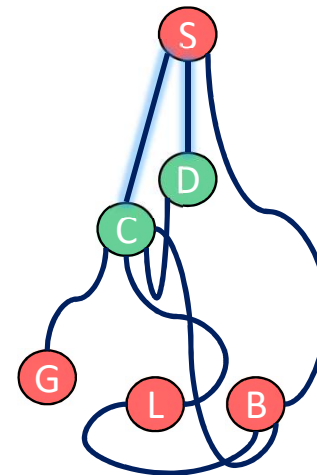
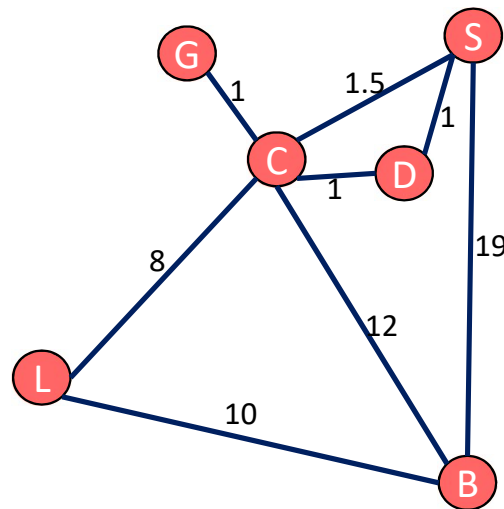
一个直观思路

- 把每个顶点看作带重力的球
- 边看作细绳连接，长度是权重
- 找从S出发的最短路径，可以看作把球S提起来，直到目的球被拉起来的过程



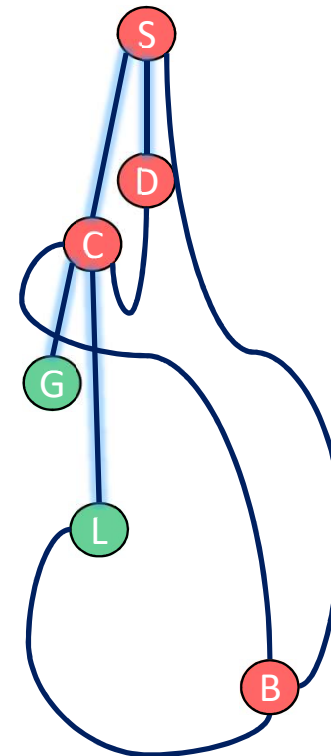
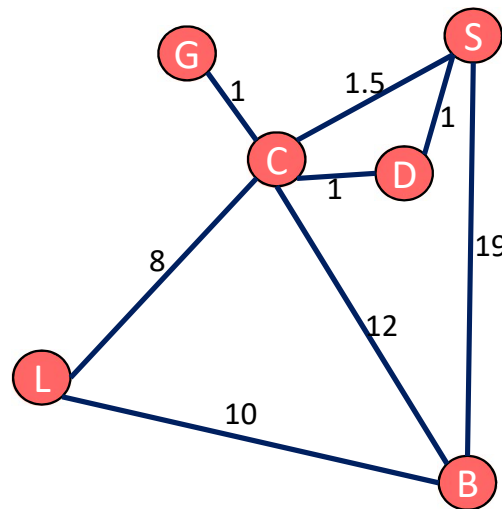
一个直观思路

- 第一个被细绳提起来的球D，是从S出发细绳长最短的
- 然后被提起来的是球C



一个直观思路

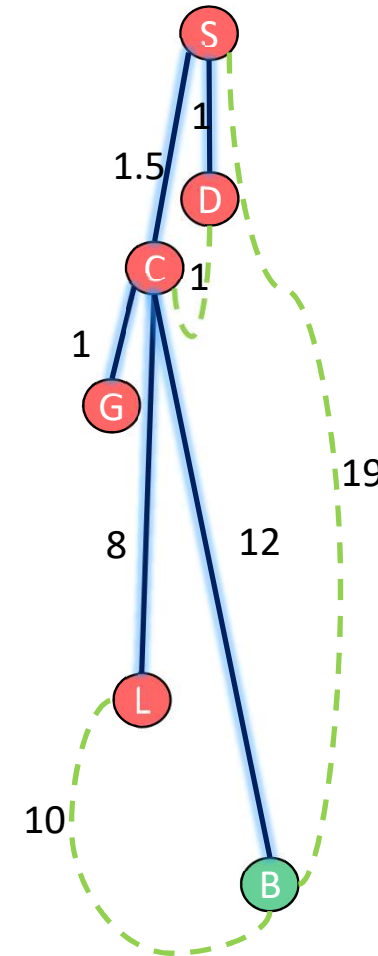
- 接着是CG细绳和CL细绳被拉紧，对应的球G和L被提起来
- 这里G和L通过球C被提起来，球C一定也已经被提起来了
(最短路径的子路径要**更短**，非负边)



一个直观思路

- 最后是被提起来的是球B
- 那么这些被拉紧的细绳就是从S出发到各个顶点的最短路径
- 这也形成了一颗最短路径树
- 只有沿着树走才是最短路径

问题：如果不利用重力和细绳，怎么找单源出发的最短路径？



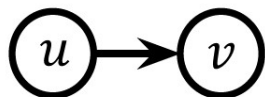
Dijkstra算法

单源最短路径算法

术语

- **Dijkstra**算法:在权重为非负的图上, 从单源出发到别的顶点的最短路径

- 假设 u 和 v 的边的权重是 $w(u, v)$



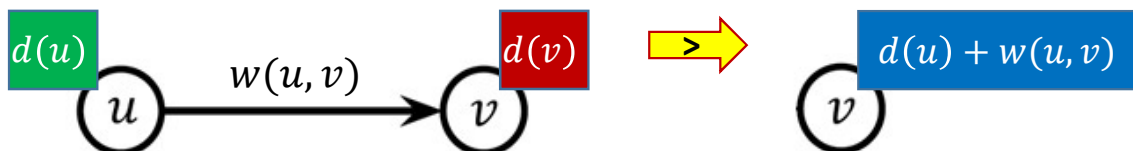
- 假设 s 到 t 的实际最短路径是 $\delta(s, t)$



- 假设顶点 u 相对于源点 s 有个距离预估值 $d(u)$
- 算法的目的是把预估值 $d(u)$ 逐步收敛到最短路径 $\delta(s, t)$

单步收敛

- 对于顶点 u 的任意邻接顶点 v ，即 (u, v) 是边，收敛操作是
$$d(v) = \min\{\textcolor{red}{d(v)}, \textcolor{green}{d(u)} + \textcolor{blue}{w(u, v)}\}$$



- 算法的过程是不断进行单步收敛操作把 $d(u)$ 逐步收敛到最短路径 $\delta(s, t)$
- 把细绳逐个收紧的过程

Dijkstra算法

- 每个顶点 u 都有个距离预估值 $d(u)$ ，除了起始点 s 的值是0，别的顶点初始值都是 ∞
- 每个顶点有两个状态：**未完成**和**完成**。初始时，所有顶点都是**未完成**
 - 1) 取一个“**未完成**”的顶点 u ，其预估值 $d(u)$ 是**最小**的
 - 2) 对于顶点 u 的所有“**未完成**”邻接顶点 v ，进行单步收敛操作
$$d(v) \leftarrow \min\{d(v), d(u) + w(u, v)\}$$
 - 3) 然后把顶点 u 的状态设置成“**完成**”
- 因为每次能把一个顶点归为“**完成**”，最多进行 $|V|$ 次后，算法结束

伪代码

- 初始化
 - 未完成 ●
 - 完成 ●
- 最短距离迭代计算过程

For each vertex u

$d(u) = \infty$; $u.status = \text{●}$
 $d(s) = 0$

For $i = 1, \dots, n$:

Find u with status ●, such that $d(u)$ is min

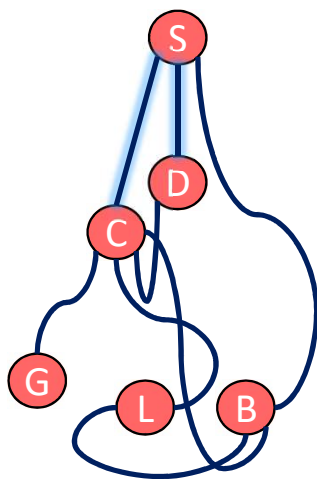
For each neighbor v with status ●:

$d(v) \leftarrow \min\{\textcolor{red}{d(v)}, \textcolor{green}{d(u)} + \textcolor{blue}{w(u, v)}\}$



$u.status = \text{●}$

类比重力+细绳做法

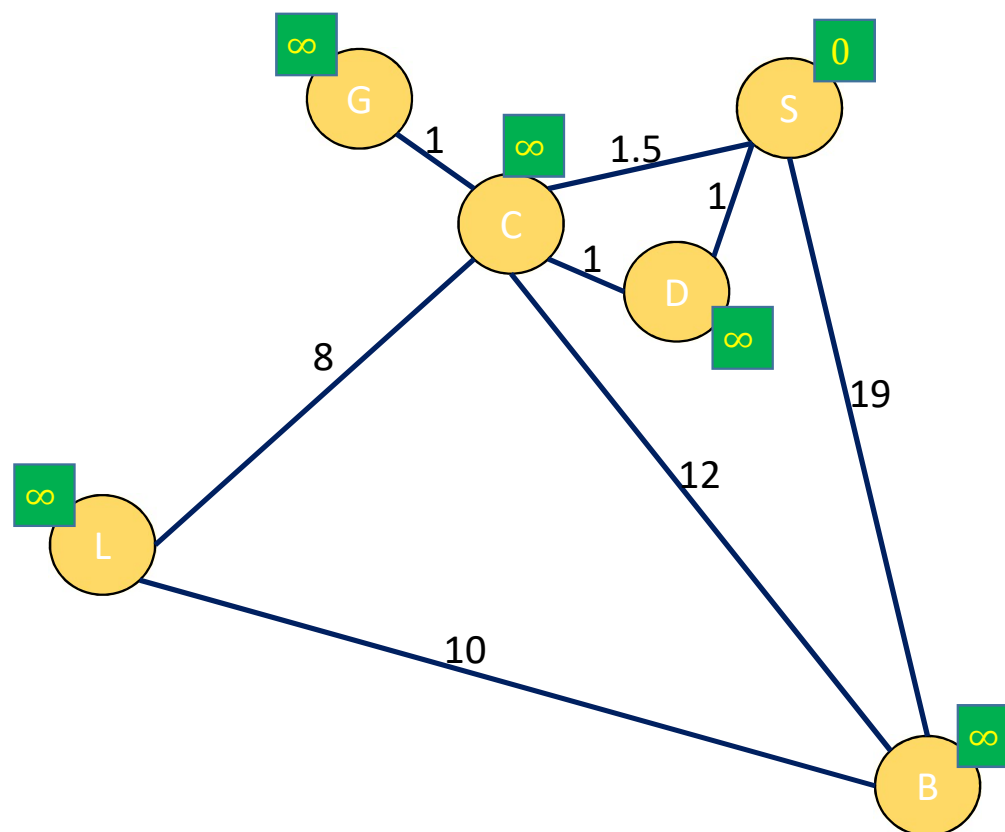
- 每次都提起一个球（提起的球变成完成状态）
- 用预估值 $d(u)$ 计算下一个会被提起的球
 - 即除了已经被提起的球外，下一个距离最短的球



例子

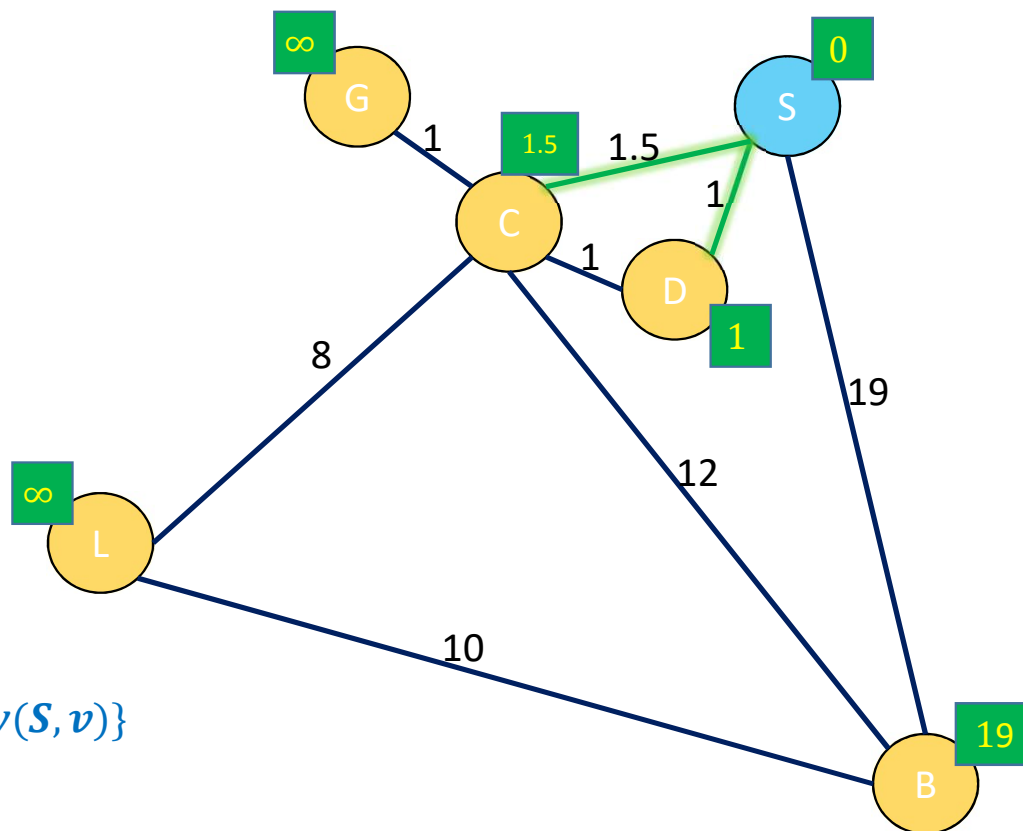
- 未完成 
- 完成 

- 取预估值最小的顶点S





例子

- 未完成 ●
- 完成 ●

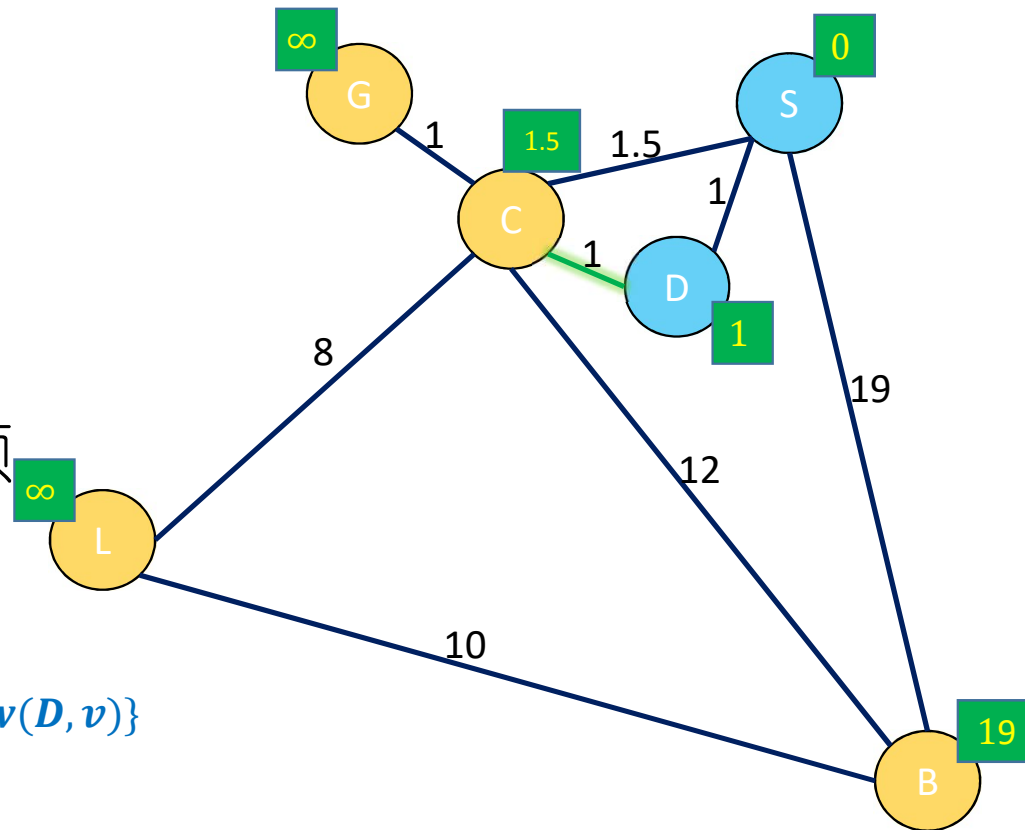


- 对于S邻接顶点 v ,
 $d(v) \leftarrow \min\{d(v), d(S) + w(S, v)\}$
- 把S标成完成

例子

- 未完成 
- 完成 

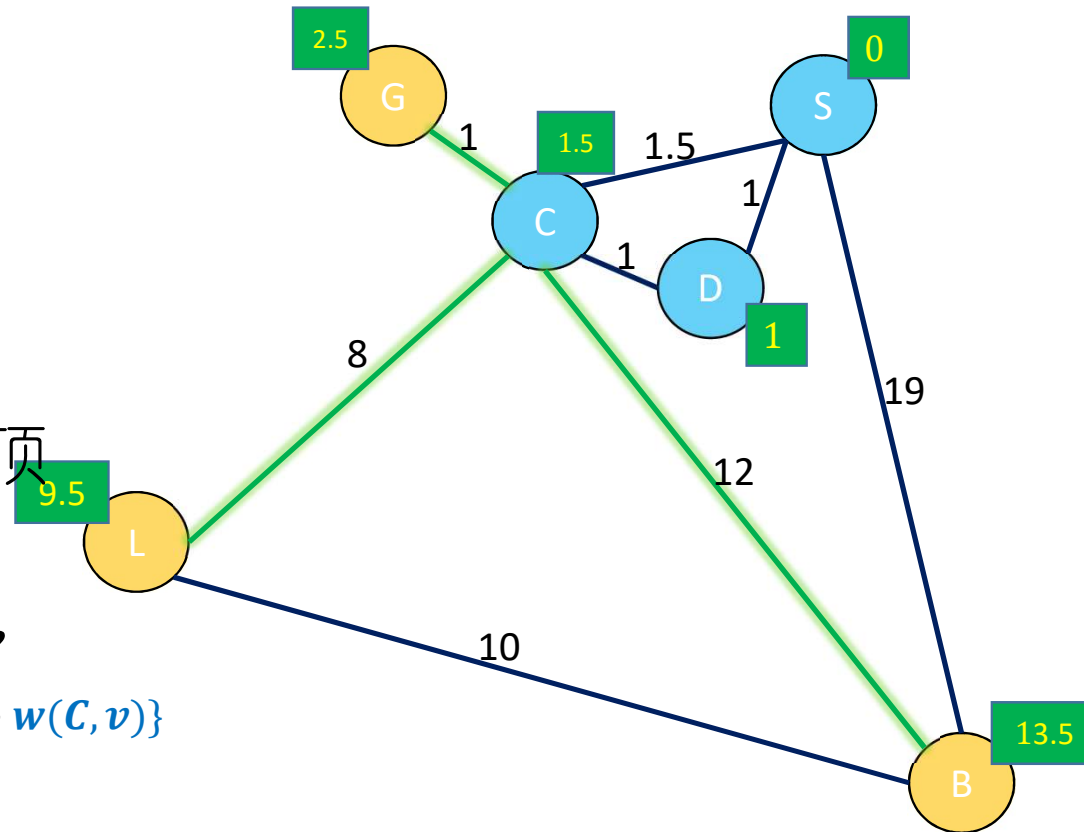
- 取预估值最小的顶点D
- 对于D邻接顶点 v ,
$$d(v) \leftarrow \min\{d(v), d(D) + w(D, v)\}$$
- 把D标成完成



例子

- 未完成 ●
- 完成 ●

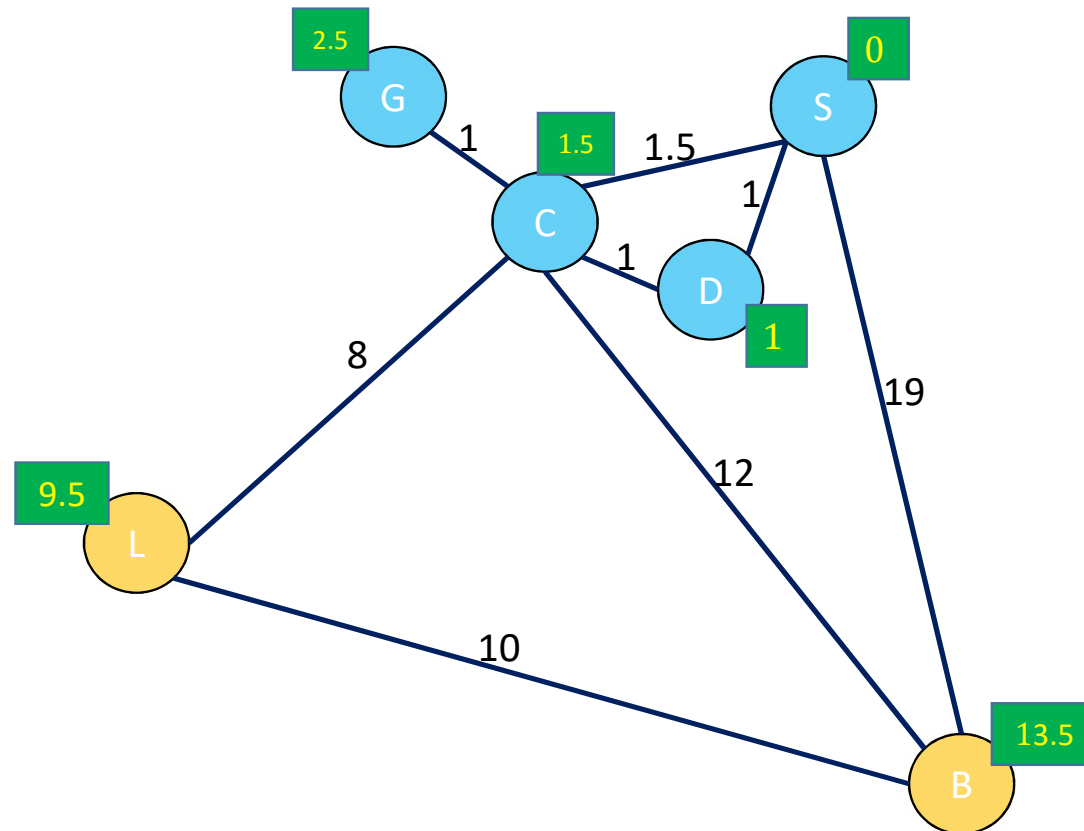
- 取预估值最小的顶点C
- 对于C邻接顶点 v ,
$$d(v) \leftarrow \min\{d(v), d(C) + w(C, v)\}$$
- 把C标成完成





例子

- 未完成 ●
- 完成 ●

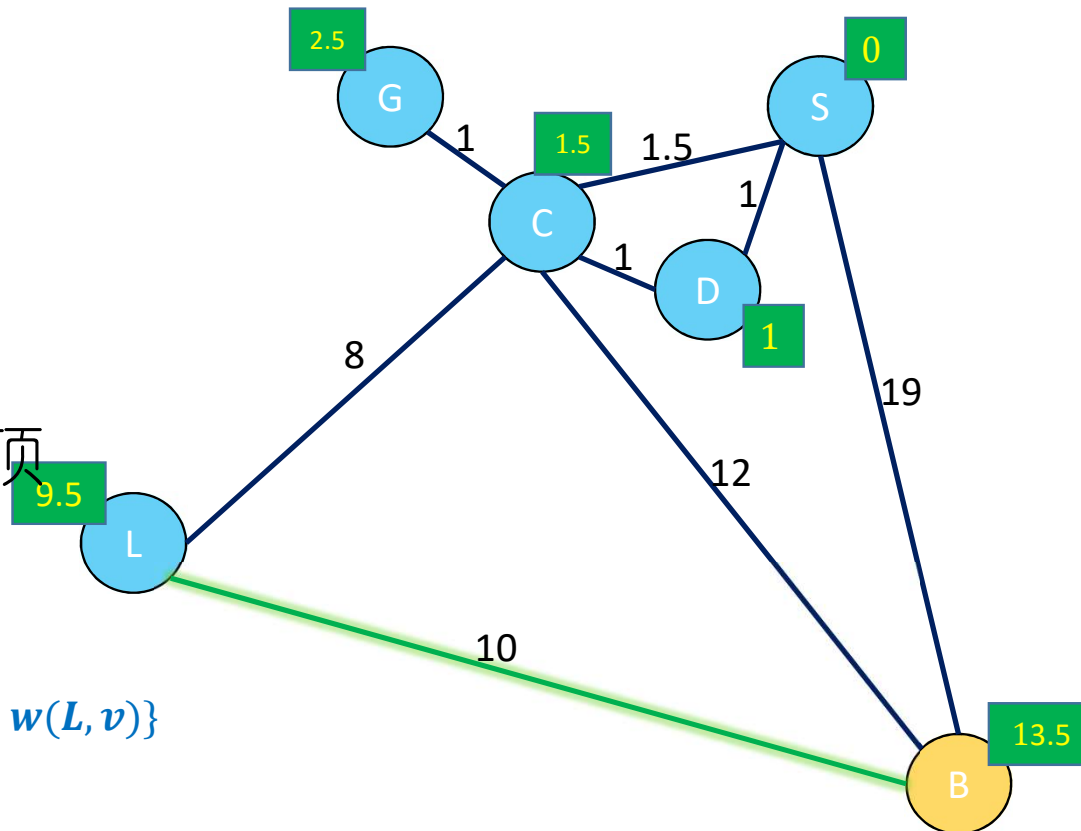
- 取预估值最小的顶点**G**
- 没有邻接顶点
- 把**G**标成完成





例子

- 未完成 
- 完成 

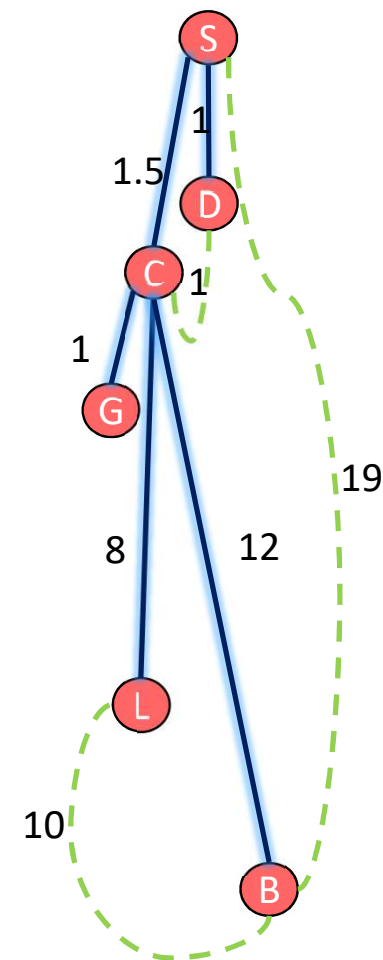
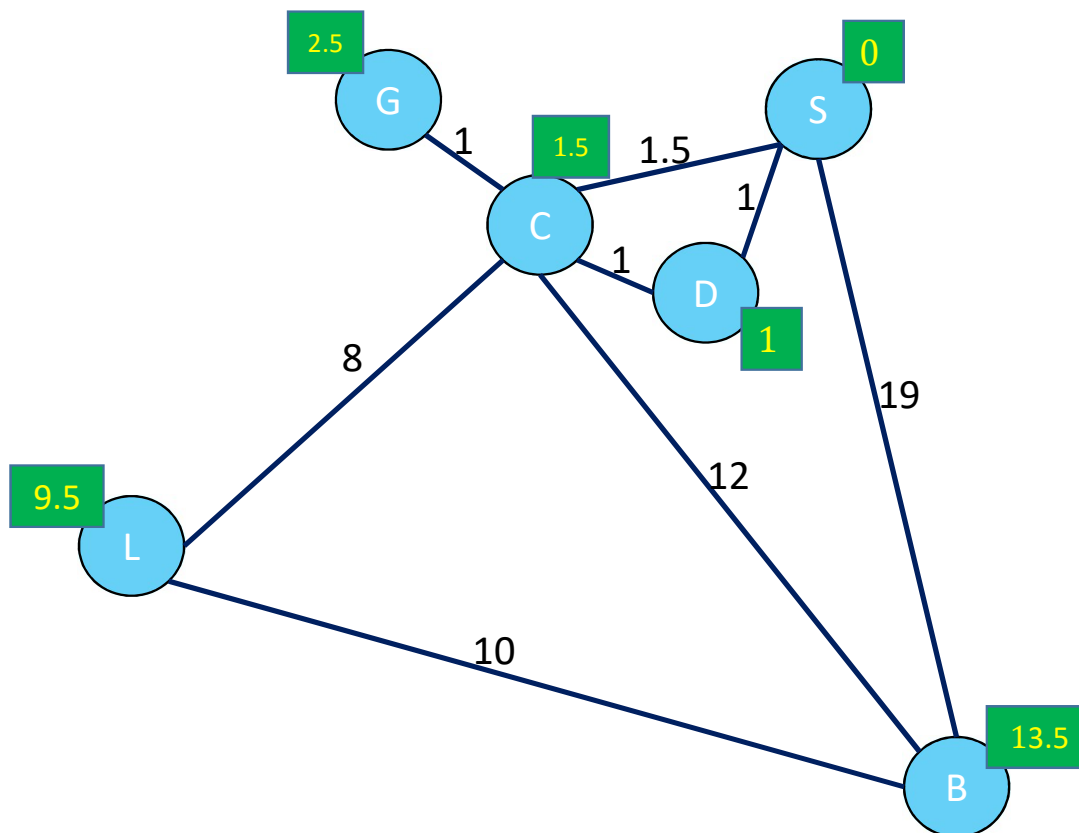
- 取预估值最小的顶点L
- 对于L邻接顶点 v ,
$$d(v) \leftarrow \min\{d(v), d(L) + w(L, v)\}$$
- 把L标成完成



例子

- 未完成 
- 完成 

- 取预估值最小的顶点B
- 没有未完成邻接顶点
- 把B标成完成



为什么正确

定理：**dijkstra**算法结束后，所有顶点的预估值 $d(v)$ 都是实际最短距离 $\delta(s, v)$

证明：

假设下面两条成立

- 1) 对任意顶点 v ，算法过程中 $d(v) \geq \delta(s, v)$ ，且递减
- 2) 当一个预估值最小顶点 v 设置为完成状态，满足 $d(v) = \delta(s, v)$


因为收敛操作只减不增，根据1)， $d(v)$ 从一个方向逼近 $\delta(s, v)$
顶点 v 变成完成状态时，正好其预估值是 $\delta(s, v)$ ，之后就不再变动
最后每个顶点都处于完成状态，所以最后 $d(v) = \delta(s, v)$

证明 (一)

第一条： 对任意顶点 v , $d(v) \geq \delta(s, v)$

- 数学归纳法：假设第 i 次迭代后，我们有 $d(v) \geq \delta(s, v)$ ，我们要证明第 $i + 1$ 次迭代后， $d(v) \geq \delta(s, v)$ 依然成立
- Base Case：起始状态 $d(s) = 0 = \delta(s, s)$, $d(v) = \infty \geq \delta(s, v)$
- 在第 $i + 1$ 次迭代，我们找到预估值 $d(u)$ 最小的未完成顶点 u ，对于它的所有未完成邻接顶点 v ，执行单步收敛操作

$$d(v) = \min\{d(v), d(u) + w(u, v)\}$$


$$\delta(s, v) \leq \underline{d(v)} \qquad \delta(s, v) \leq \delta(s, u) + w(u, v) \\ \qquad \qquad \qquad \leq \underline{d(u)} + w(u, v)$$

证明（二）

第二条： 当一个顶点 v 设置为完成状态，那么 $d(v) = \delta(s, v)$

- 数学归纳法：假设第 i 次迭代后，所有完成状态的顶点 v 都有 $d(v) = \delta(s, v)$ ，我们要证明下一个预估值最小的未完成顶点 x ，已经满足 $d(x) = \delta(s, x)$
- Base Case：起始状态 $d(s) = 0 = \delta(s, s)$ ，已经满足
- 归纳步骤证明：在第 $i + 1$ 次迭代，我们找到预估值最小的顶点 x ，然后对其邻接顶点执行收敛操作，同时把 x 设为完成状态
- 我们用反证法证明对于顶点 x ，已经有 $d(x) = \delta(s, x)$

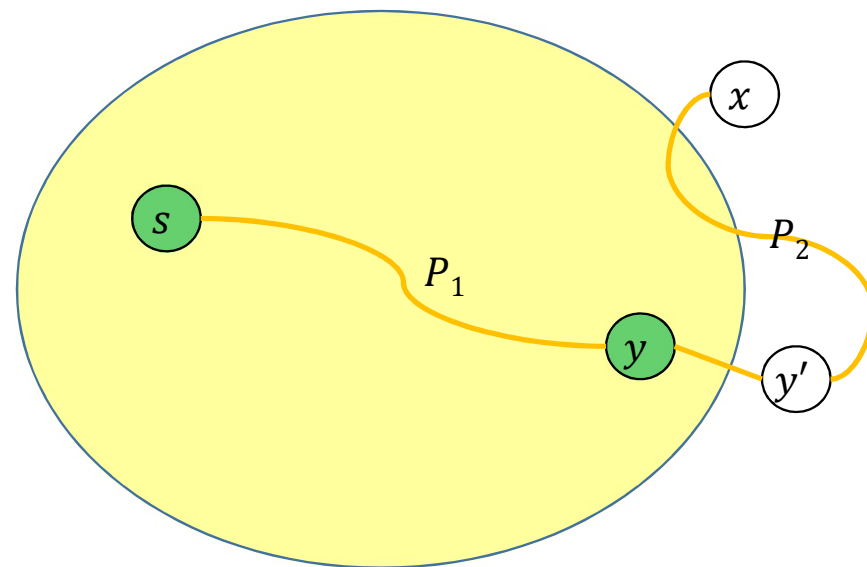
证明 (二)

- 反证法：假设下一个预估值最小的未完成顶点 x ， $d(x) > \delta(s, x)$

假设 s 到 x 的真正最短路径是

$$s \xrightarrow{P_1} y \rightarrow y' \xrightarrow{P_2} x$$

- y' 是路径上第一个未完成顶点（注意 y' 可能和 x 重合）
- y 是 y' 前一个的顶点，那么 y 是完成状态（注意 y 可能和 s 重合）
- 路径 P_2 可能进入黄圈，也可能不进入
- y' 是个特殊点、分界点，有下面性质**



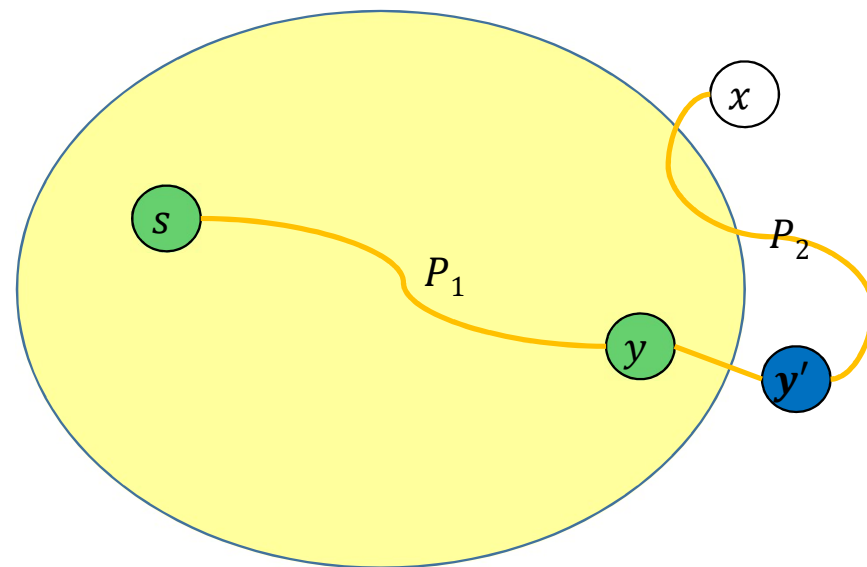
证明 (二)

- 我们首先证明 $d(y') = \delta(s, y')$, 即 y' 已经收敛
- 根据第一条: $d(y') \geq \delta(s, y')$
- 因为最短路径的子路径是最短路径, 所以 $\delta(s, y') = \delta(s, y) + w(y, y')$

顶点 y 是完成状态,
归纳结果 $d(y) = \delta(s, y)$

$$\begin{aligned} &= d(y) + w(y, y') \\ &\geq d(y') \end{aligned}$$

在设置顶点 y 完成状态时, 已经更新
 $d(y') = \min\{d(y'), d(y) + w(y, y')\}$



证明 (二)

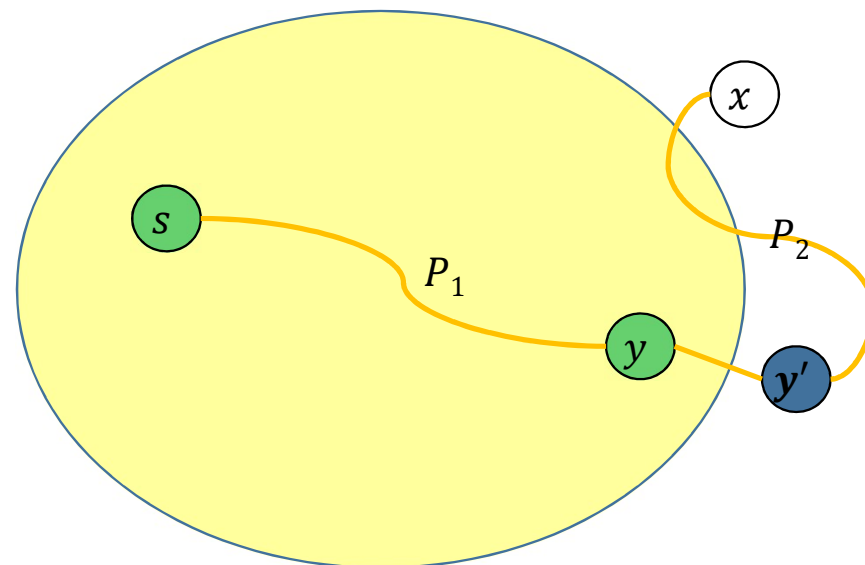
- 根据我们的选法,

$$\begin{aligned} d(x) &\leq d(y') \\ &= \delta(s, y') \\ &\leq \delta(s, x) \end{aligned}$$

上一页证明结果

$\delta(s, y')$ 是无负边最短路径的子路径

- 从而 $d(x) \leq \delta(s, x)$, 导出矛盾



为什么正确

定理：dijkstra算法结束后，所有顶点的预估值 $d(v)$ 都是实际最短距离 $\delta(s, v)$

证明：假设下面两条成立

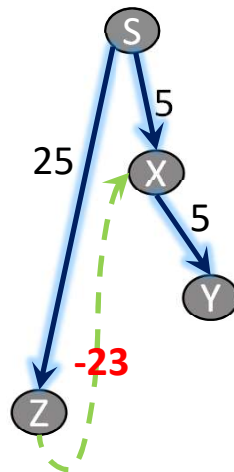
- 1) 对任意顶点 v ，算法过程中 $d(v) \geq \delta(s, v)$ ，且递减
- 2) 当一个预估值最小的顶点 v 设置为完成状态，满足 $d(v) = \delta(s, v)$

因为收敛操作只减不增，根据1)， $d(v)$ 从一个方向逼近 $\delta(s, v)$

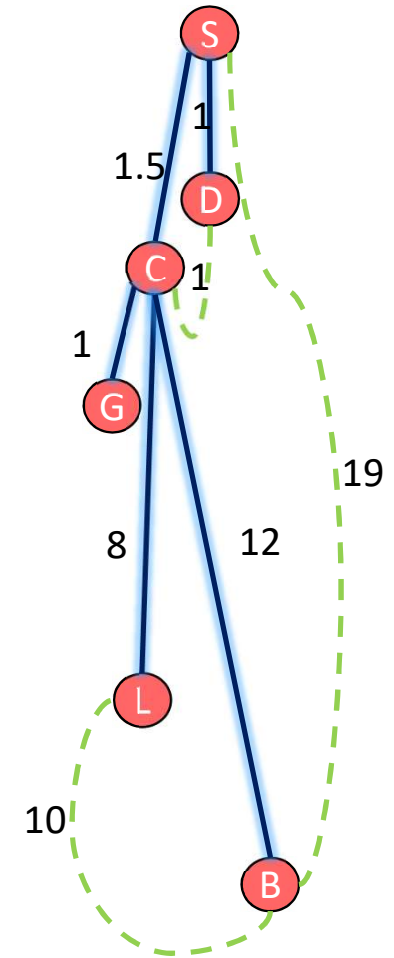
顶点 v 变成完成状态时，正好其预估值是 $\delta(s, v)$ ，之后就不再变动
最后每个顶点都处于完成状态，所以最后 $d(v) = \delta(s, v)$

小结

- Dijkstra算法可以找到单源到其余顶点的最短距离
 - 图的边权重**非负**



- 同时也建立了一颗最短距离树，如果在收敛操作的时候把父节点记录下来



时间复杂度

- 对于每个顶点，都执行右边的一次迭代操作

- 关键是如何实现一次迭代操作，设计管理“未完成”顶点的数据结构：优先队列

1) 找预估值 $d(u)$ 最小的顶点 u

$u = \text{FindMin}()$



1) 取一个“未完成”的顶点 u ，其预估值 $d(u)$ 是最小的

2) 更新邻接顶点的预估值

$\text{UpdateKey}(v, d)$



2) 对于顶点 u 的所有“未完成”邻接顶点 v ，进行单步收敛操作

3) 移去预估值最小顶点 u ，

$\text{ExtractMin}(u)$



$d(v) \leftarrow \min\{d(v), d(u) + w(u, v)\}$

3) 然后把顶点 u 的状态设置成“完成”

算法复杂度是

$$|V| \times \{T(\text{FindMin}) + T(\text{ExtractMin})\} + |E| \times T(\text{UpdateKey})$$

方案一：数组

如果 $|V| = n$, $|E| = m$,

- $T(\textit{FindMin}) = O(n)$
- $T(\textit{ExtractMin}) = O(n)$
- $T(\textit{UpdateKey}) = O(1)$
- Dijkstra算法的复杂度是
$$|V| \times \{T(\textit{FindMin}) + T(\textit{ExtractMin})\} + |E| \times T(\textit{UpdateKey})$$
$$= n \times O(n) + O(m) = O(n^2)$$

方案二：平衡二叉查找树

如果 $|V| = n$, $|E| = m$,

- $T(\textit{FindMin}) = O(\log(n))$
- $T(\textit{ExtractMin}) = O(\log(n))$
- $T(\textit{UpdateKey}) = O(\log(n))$
- Dijkstra算法的复杂度是
$$|V| \times \{T(\textit{FindMin}) + T(\textit{ExtractMin})\} + |E| \times T(\textit{UpdateKey})$$
$$= n \times O(\log(n)) + O(m \log(n)) = O((n + m) \log(n))$$

方案三：hash table

如果 $|V| = n$, $|E| = m$,

- $T(\textit{FindMin}) = O(n)$
- $T(\textit{ExtractMin}) = O(1)$
- $T(\textit{UpdateKey}) = O(1)$
- Dijkstra算法的复杂度是
$$|V| \times \{T(\textit{FindMin}) + T(\textit{ExtractMin})\} + |E| \times T(\textit{UpdateKey})$$
$$= n \times O(n) + O(m) = O(n^2)$$

方案四：堆

如果 $|V| = n$, $|E| = m$,

- $T(\textit{FindMin}) = O(1)$
- $T(\textit{ExtractMin}) = O(\log(n))$
- $T(\textit{UpdateKey}) = O(\log(n))$
- Dijkstra算法的复杂度是
$$|V| \times \{T(\textit{FindMin}) + T(\textit{ExtractMin})\} + |E| \times T(\textit{UpdateKey})$$
$$= n \times O(\log(n)) + O(m \log(n)) = O((n + m) \log(n))$$

方案五：Fibonacci堆

如果 $|V| = n$, $|E| = m$,

- $T(\textit{FindMin}) = O(1)$
- $T(\textit{ExtractMin}) = O(\log(n))$
- $T(\textit{UpdateKey}) = O(1)$
- Dijkstra算法的复杂度是
$$|V| \times \{T(\textit{FindMin}) + T(\textit{ExtractMin})\} + |E| \times T(\textit{UpdateKey})$$
$$= n \times O(\log(n)) + O(m) = O(n \log(n) + m)$$

小结

- **Dijkstra**算法虽然很快
- 只适用于权重非负的图
- 如果图的权重变了，需要重新计算最短路径

负边的最短路径算法

Bellman-Ford算法

- 比dijkstra算法要慢
- 不过可以处理负边
 - 如果不光考虑距离，额外考虑别的因素
- 检测负环

带负环的图

- 这种情况没有最短路径
- 因为每走一次负环，路径权重和降低

Bellman-Ford算法

- 每个顶点 u 都有个预估值 $d(u)$ ，除了起始点 s 的值是0，别的顶点初始值都是 ∞
- 对下面的步骤执行 $n - 1$ 次：
 - 对于图中所有边 (u, v) ，进行单步收敛操作
$$d(v) \leftarrow \min\{d(v), d(u) + w(u, v)\}$$
- 最后
 - 如果存在某条边 (u, v) ： $d(v) > d(u) + w(u, v)$ ，那么返回存在负环
 - 否则预估值 $d(v)$ 就是从起始点出发到 v 的最短距离

伪代码

- 初始化

For each vertex u

$$d(u) = \infty$$

$$d(s) = 0$$

- 最短距离迭代计算过程

For $i = 1, \dots, n - 1$:

For each edge (u, v) in E :

$$d(v) \leftarrow \min\{d(v), d(u) + w(u, v)\}$$

- 检测负环

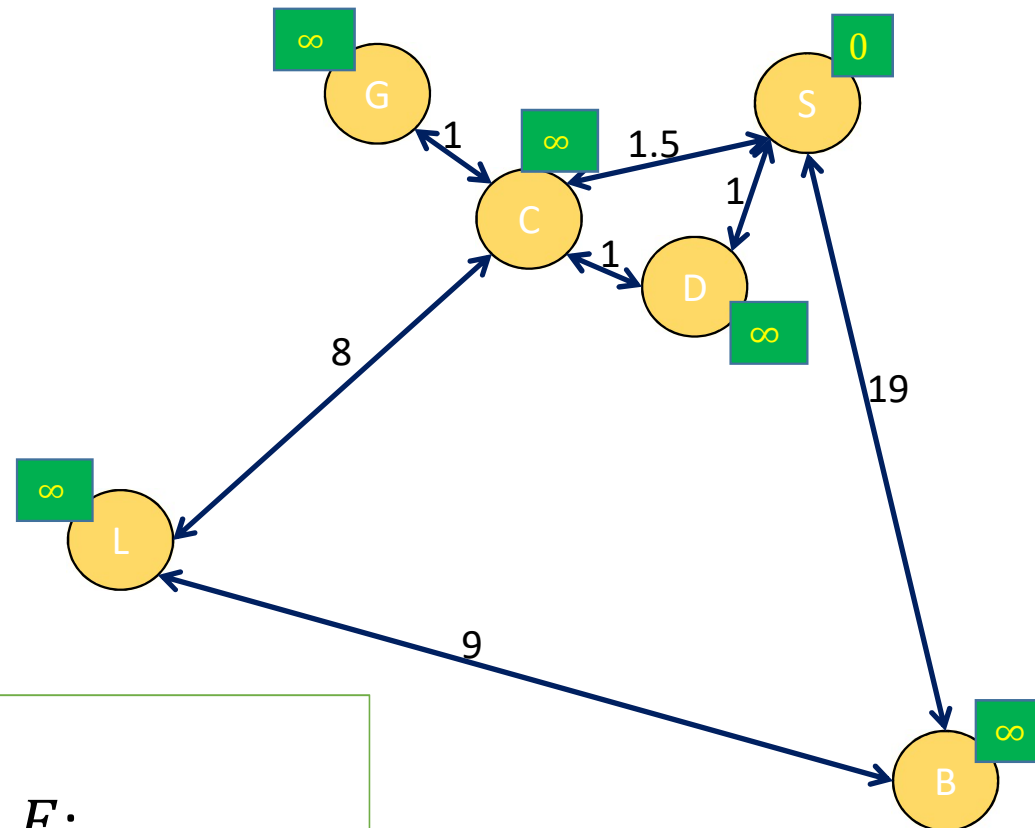
For each edge (u, v) in E :

If $d(v) > d(u) + w(u, v)$

return “negative cycle”

例子

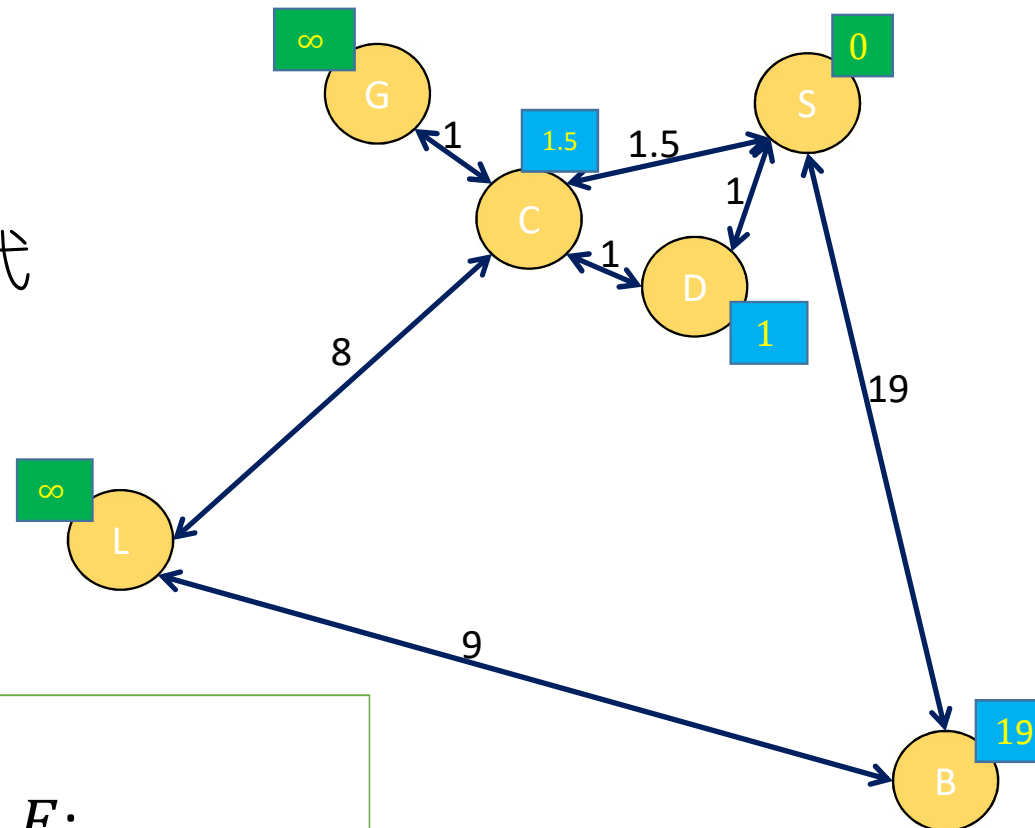
- 设置初始状态



For $i = 1, \dots, n - 1$:
For each edge (u, v) in E :
 $d(v) \leftarrow \min\{\textcolor{red}{d(v)}, \textcolor{green}{d(u)} + \textcolor{blue}{w(u, v)}\}$

例子

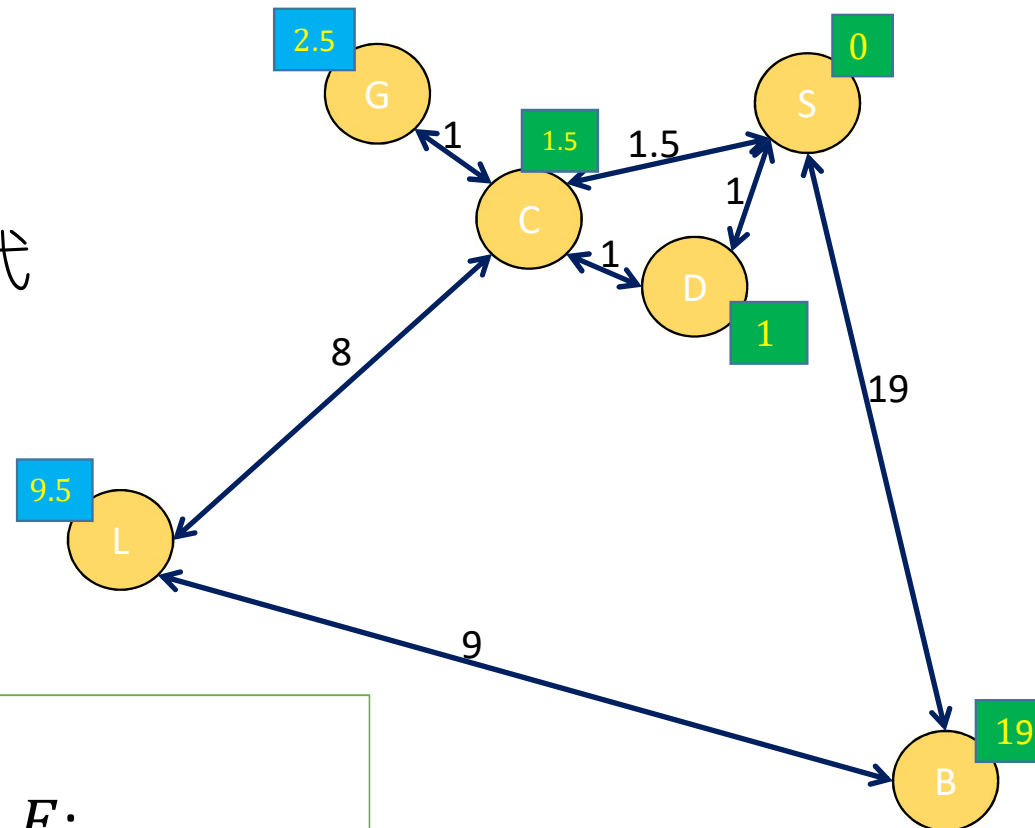
- 执行一遍距离迭代



For $i = 1, \dots, n - 1$:
For each edge (u, v) in E :
 $d(v) \leftarrow \min\{\textcolor{red}{d}(v), \textcolor{green}{d}(u) + \textcolor{blue}{w}(u, v)\}$

例子

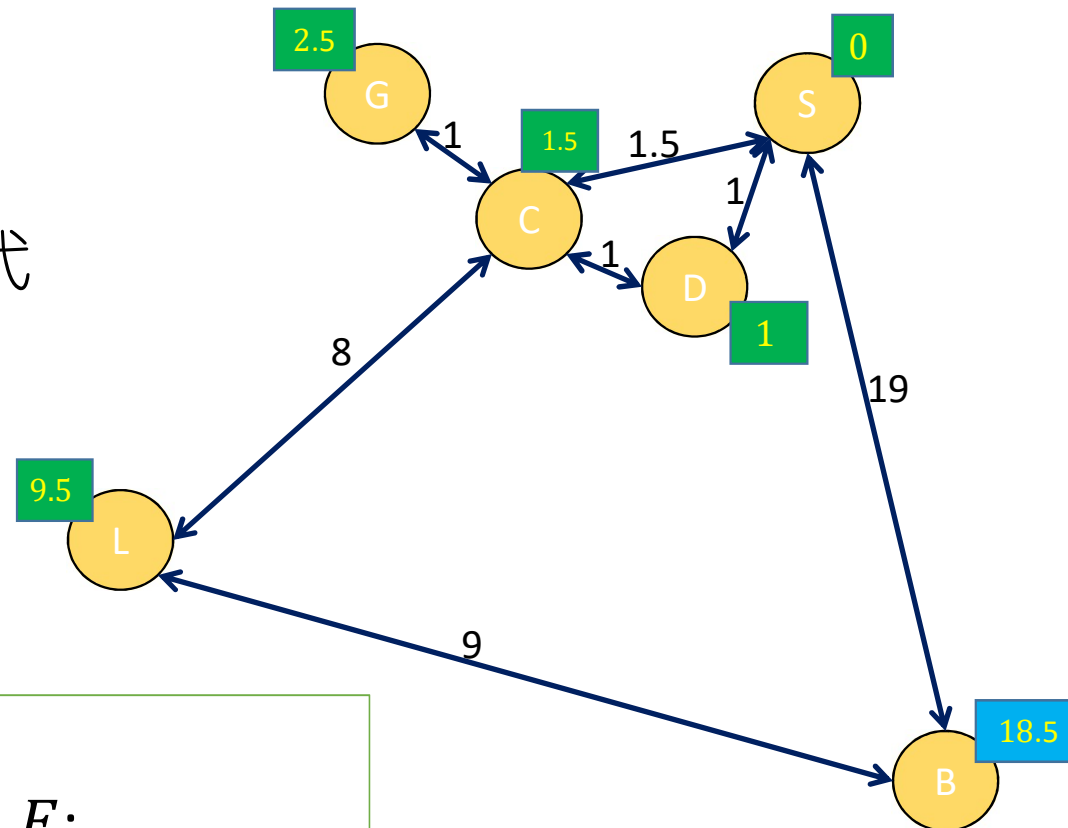
- 执行二遍距离迭代



For $i = 1, \dots, n - 1$:
For each edge (u, v) in E :
 $d(v) \leftarrow \min\{\textcolor{red}{d}(v), \textcolor{green}{d}(u) + \textcolor{blue}{w}(u, v)\}$

例子

- 执行三遍距离迭代



For $i = 1, \dots, n - 1$:
For each edge (u, v) in E :
 $d(v) \leftarrow \min\{\textcolor{red}{d}(v), \textcolor{green}{d}(u) + \textcolor{blue}{w}(u, v)\}$

时间复杂度

如果 $|V| = n$, $|E| = m$,

- Bellman-Ford算法复杂度是 $O(mn)$
- 比dijkstra算法要慢
- 好处是
 - 可以处理负边
 - 实现思路更简单

For $i = 1, \dots, n - 1$:

For each edge (u, v) in E :

$d(v) \leftarrow \min\{d(v), d(u) + w(u, v)\}$

为什么正确

定理：**Bellman-Ford**算法结束后，所有顶点的预估值 $d(v)$ 都是实际最短距离 $\delta(s, v)$

证明：用数学归纳法证明

- 假设第 i 次迭代后，对于顶点 v ，如果其真正最短距离最多 i 条边，那么 $d(v) = \delta(s, v)$
- 首先，base case：第0次迭代后， $d(s) = 0 = \delta(s, s)$

证明

- 要证明再迭代一次后, 对于顶点 v , 如果其真正最短距离最多 $i + 1$ 条边, 那么 $d(v) = \delta(s, v)$

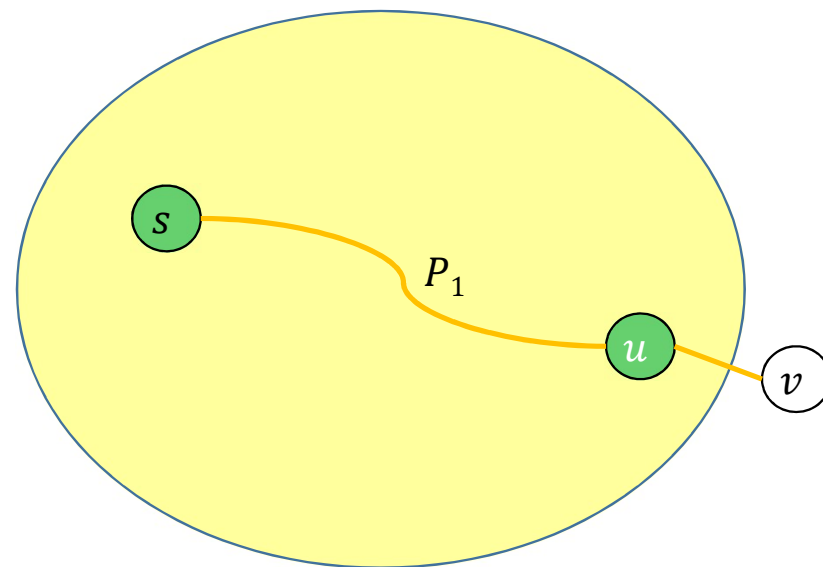
假设 s 到 v 的真正最短路径是

$$s \xrightarrow{P_1} u \rightarrow v$$

- 因为 s 到 v 的最短路径最多 $i + 1$ 条边
- 所以 s 到 u 的最短路径 P_1 最多 i 条边
- 根据归纳假设, $d(u) = \delta(s, u)$
- 那么再迭代一次后,

$$d(v) = \min\{d(v), d(u) + w(u, v)\}$$

$$\delta(s, u) + w(u, v) = \delta(s, v)$$



证明

- Base Case: 第0次迭代后, $d(s) = 0 = \delta(s, s)$ ✓
- 假设第*i*次迭代后, 对于顶点*v*, 如果其真正最短距离最多*i*条边, 那么 $d(v) = \delta(s, v)$
- 再迭代一次后, 对于任意顶点*v*, 如果其真正最短距离最多*i* + 1条边, 那么 $d(v) = \delta(s, v)$ ✓
- 结论: 第*n* - 1次迭代后, 对于任意顶点*v*, 如果其真正最短距离最多*n* - 1条边, 那么 $d(v) = \delta(s, v)$ ✓
- 在没有负环的情况下, 最短距离是简单路径, 即最多*n* - 1条边
- 所以结论对所有顶点都成立 ✓

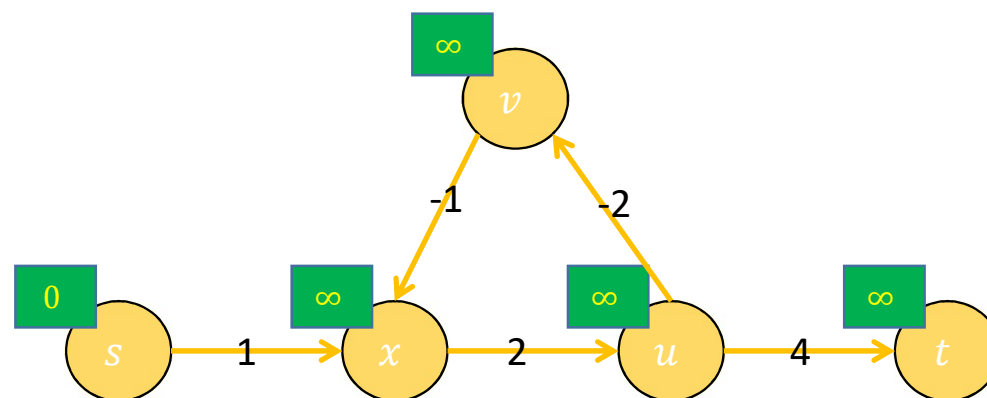
小结

- Bellman-Ford算法复杂度是 $O(mn)$
- 如果没有负环，算法结束时，对于任意顶点 v ， $d(v) = \delta(s, v)$
- 注意负边是允许的
- 如果有负环，那么最短路径不会收敛。可以按以下方法检测

```
For each edge  $(u, v)$  in  $E$ :  
  if  $d(v) > d(u) + w(u, v)$   
    return “negative cycle”
```

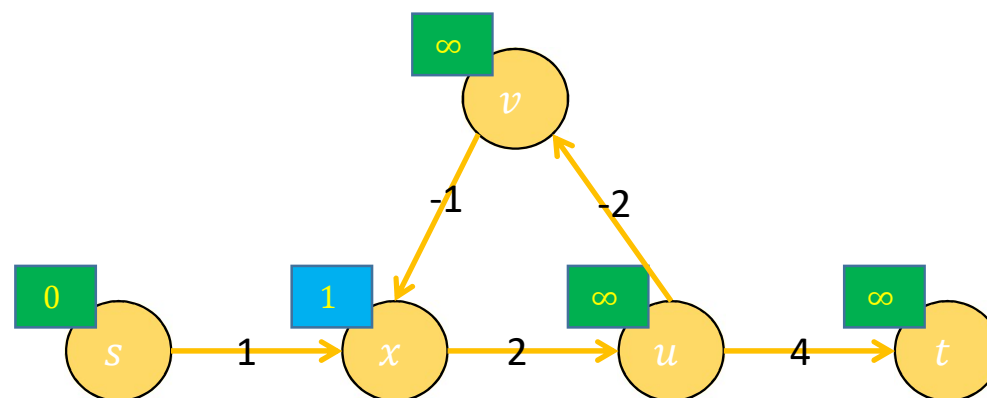

负环图

- 设置初始状态
- 更新顺序
 ut, vx, uv, xu, sx



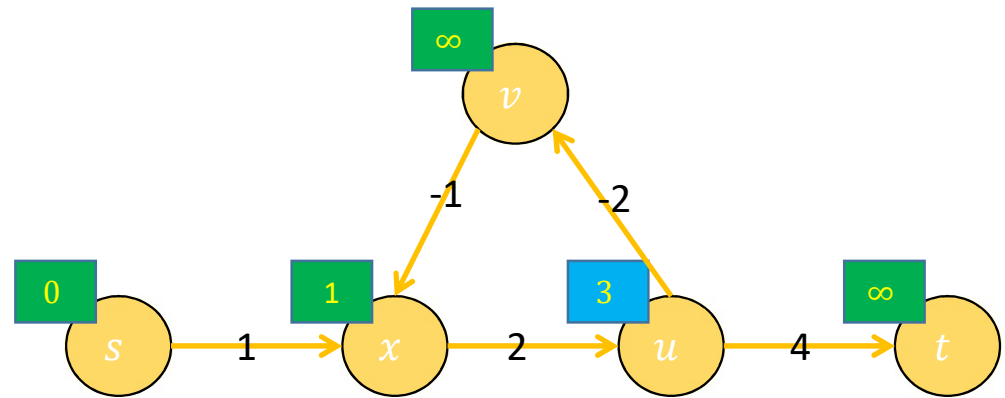
负环图

- 执行一遍距离迭代
- 更新顺序
 ut, vx, uv, xu, sx



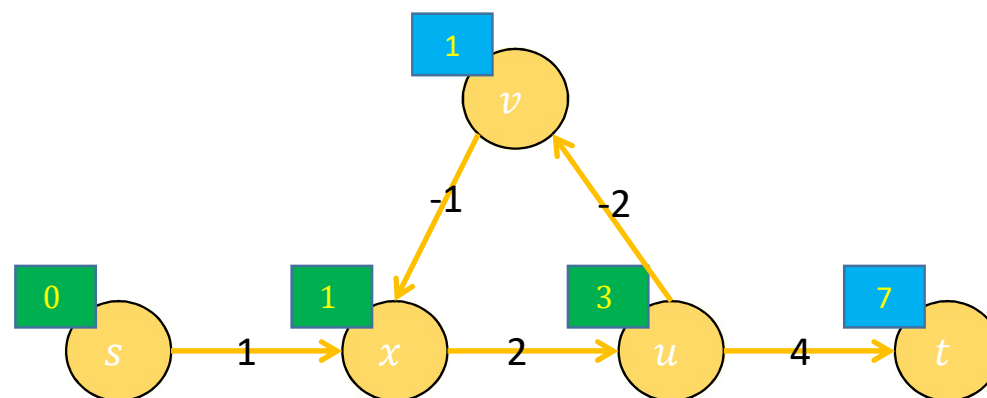
负环图

- 执行二遍距离迭代
- 更新顺序
 ut, vx, uv, xu, sx



负环图

- 执行三遍距离迭代
- 更新顺序
 ut, vx, uv, xu, sx

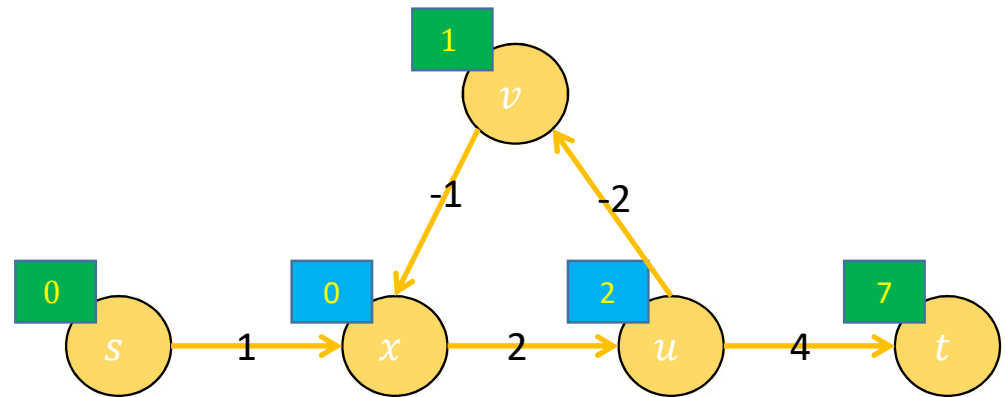


负环图

- 执行四遍距离迭代

- 更新顺序
 ut, vx, uv, xu, sx

- $d(v) = 1$
- $d(u) + w(u, v) = 0$



For each edge (u, v) in E :
 If $d(v) > d(u) + w(u, v)$
 return “negative cycle”

总结

- BFS算最短路径只适合没有权重的图
- 在加权图上，单源最短路径的算法是
 - **Dijkstra**算法效率很高，不过
 - 适用没有负边的图
 - 需要有个优先队列对所有顶点集中管理
 - **Bellman-Ford**算法速度要慢点，不过
 - 可以处理有负边的图
 - 可以分布式计算，每个顶点更新只需要知道邻接顶点的信息，更新的顺序无关紧要

DAG最短路径算法

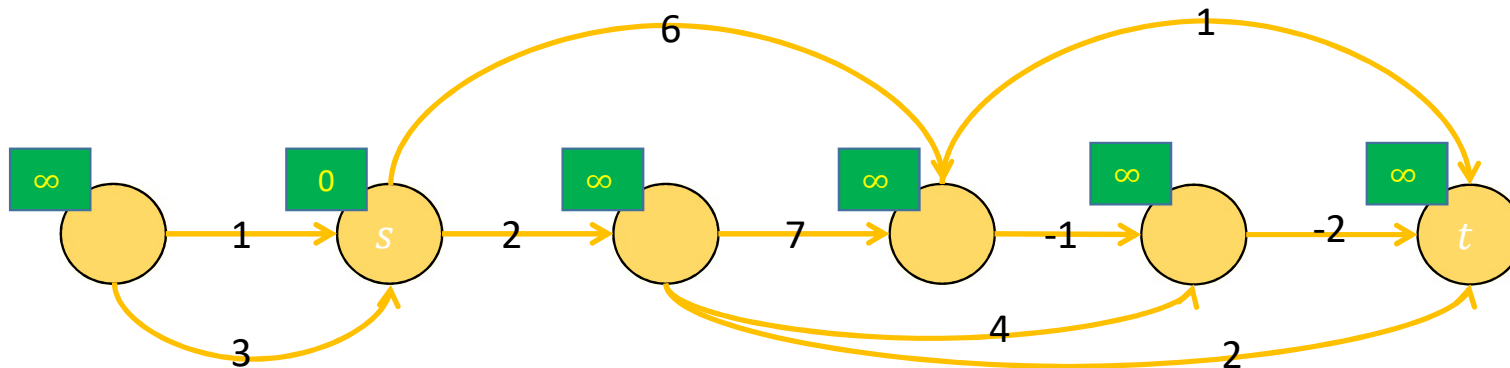
- 时间复杂度是 $O(|V| + |E|)$

TopoSort()

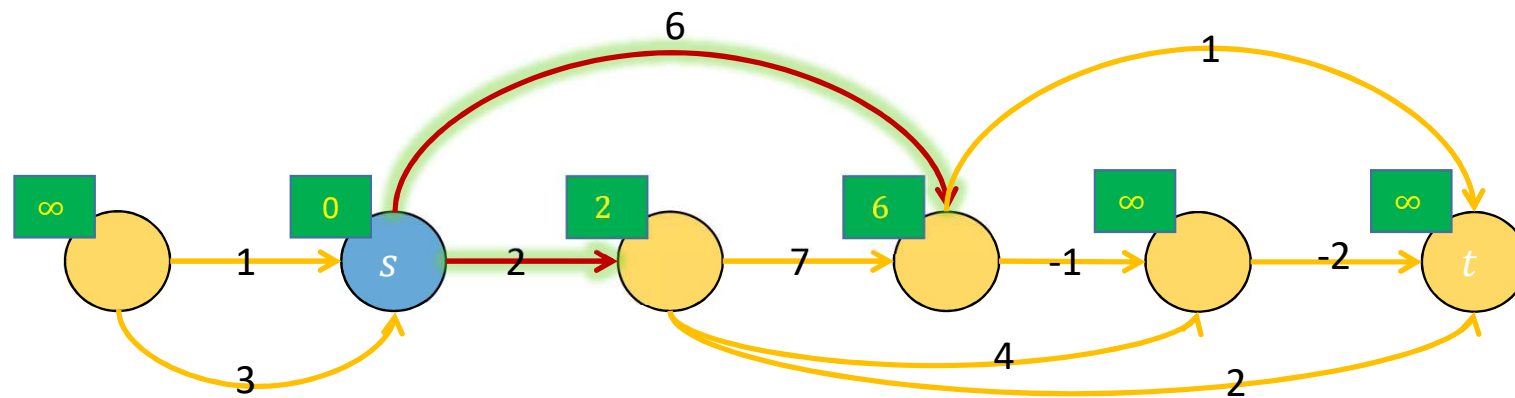
For each vertex u in topology order:

For each neighbor v of u :

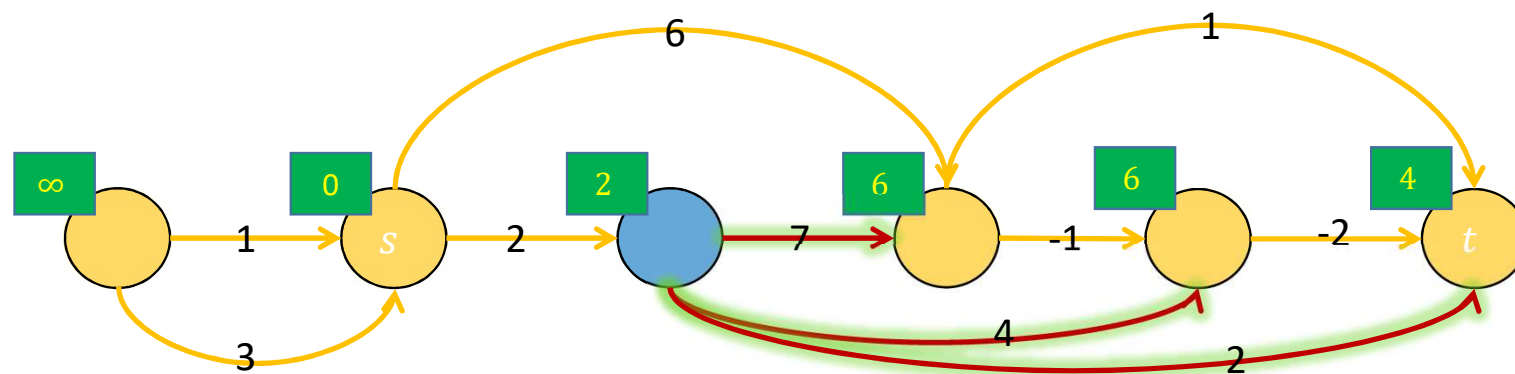
$$d(v) \leftarrow \min\{\mathbf{d}(v), \mathbf{d}(u) + \mathbf{w}(u, v)\}$$



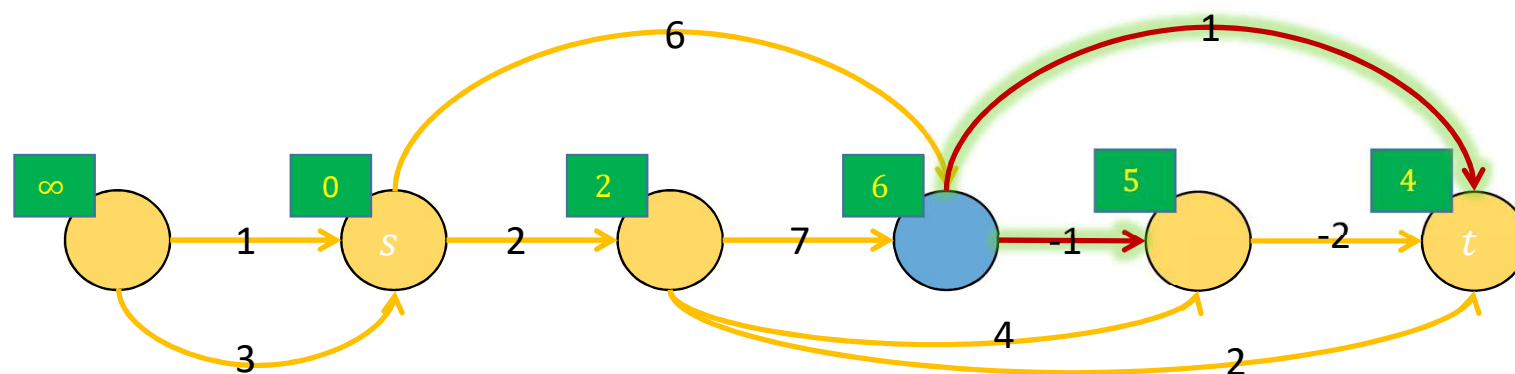
DAG最短路径算法



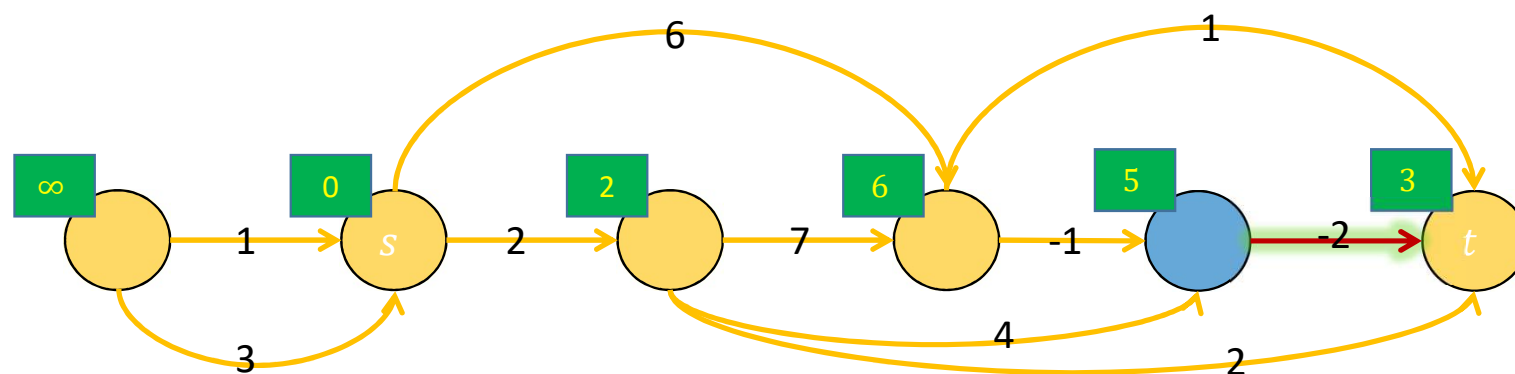
DAG最短路径算法



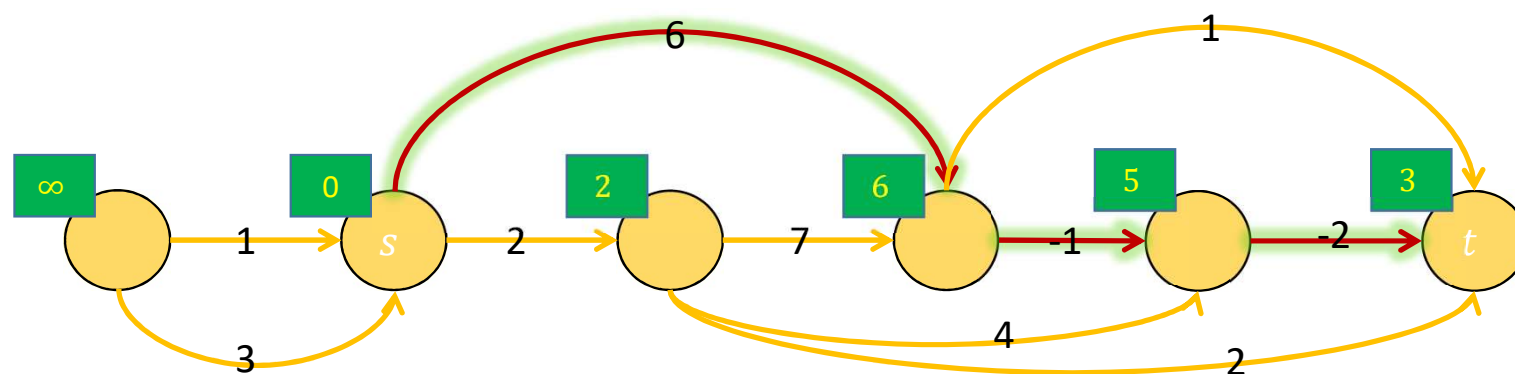
DAG最短路径算法



DAG最短路径算法



DAG最短路径算法



下面.....

- 图中两两之间的最短距离
- 使用动态规划Dynamic Programming编程

Q&A

Thanks!