ECNU

算法基础
第14周：最短路径算法

王延昊 副教授

2022.5.23

课程内容

- 最短路径问题
- 单源最短路径算法
  - Bellman–Ford 算法
  - Dijkstra’s 算法

最短路径问题

- 问题引入
  - 假设某同学需要从华师大中山北路校区出发乘坐公共交通前往华师大闵行校区，如何规划耗时最短的线路?
  - 手机导航软件如何自动规划路线?

最短路径问题

- 给定一个带权重的有向图 $G = (V, E, w)$，其中 V 表示顶点的集合，$E \subseteq V \times V$ 表示边的集合，$w: E \to \mathbb{R}$ 给每条边赋予（实数）权重
- 以导航问题为例
  - **顶点**：公交、地铁站
  - **边**：公交/地铁线路
  - **权重**：线路的预计通行时间

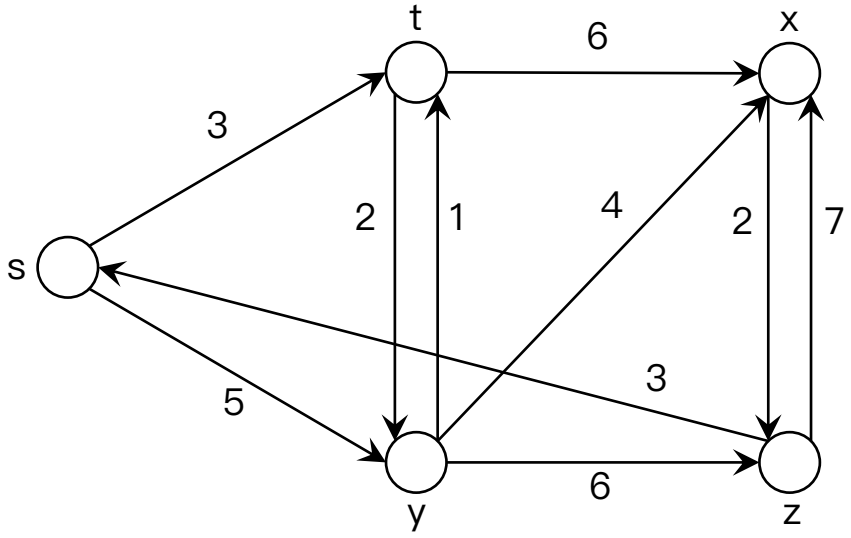
最短路径问题

- 路径 $p = \langle v_1, v_2, \ldots, v_k \rangle$
  - 图 G 中的顶点的序列
  - 任意相邻两个顶点 (v_i, v_{i+1}) 间都有边相连
- 路径的权重
  - 路径上所有边的权重之和 $w(p) = \sum_{i=1}^{k-1} w(v_i, v_{i+1})$
  - 以导航问题为例
    - 通过路径所花费的总时间等于沿路径通过所有边花费的时间之和

最短路径问题

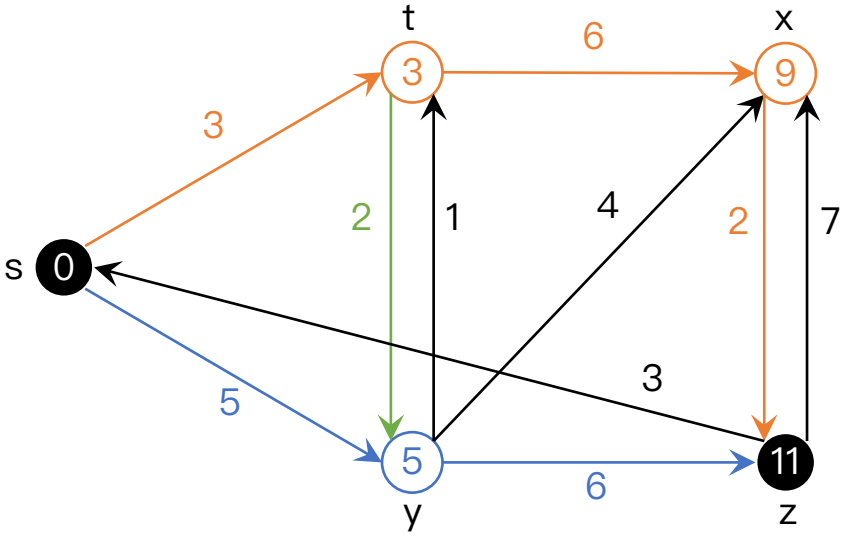
- 给定任意两个顶点 $u, v \in V$，我们定义从 u 到 v 的最短路径为所有以 u 为起点 v 为终点的路径中权重最小的路径（可能不唯一）。
- 定义最短路径权重
  - $$\delta(u,v) = \begin{cases} min\left\{w(p): u \xrightarrow{p} v\right\}, & \text{there is a path from } u \text{ to } v \\ \infty, & \text{there is no path from } u \text{ to } v \end{cases}$$
  - 最短路径问题的等价定义：找到任意一条以 u 为起点，v 为终点且权重等于 $\delta(u,v)$ 的路径

最短路径：例子



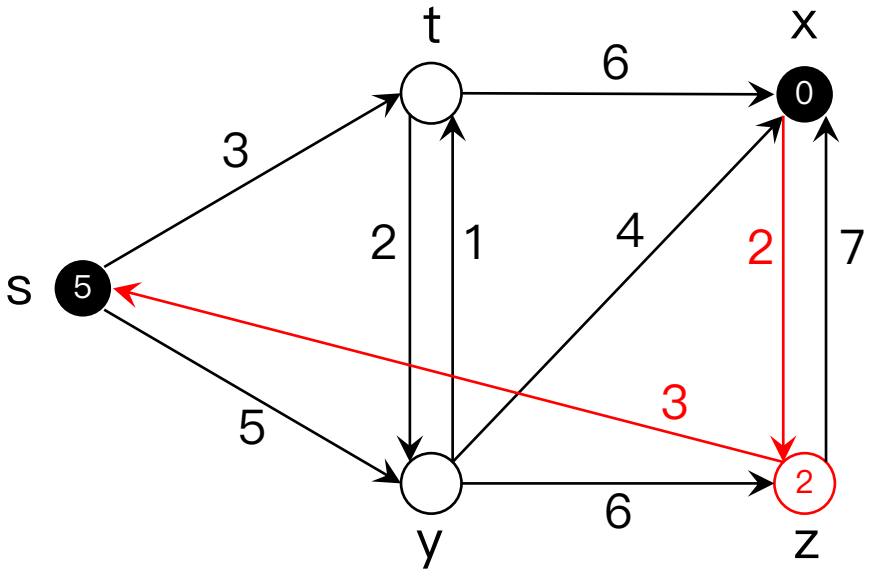
- 在左边的图中，从顶点 s 到顶点 z 的最短路径是?
- 顶点 x 到顶点 s 的最短路径是?

最短路径：例子



- 顶点 s 到顶点 z 的最短路径
  - $p_1 = \langle s, y, z \rangle$
  - $p_2 = \langle s, t, x, z \rangle$
  - $p_3 = \langle s, t, y, z \rangle$
  - $w(p_1) = w(p_2) = w(p_3) = \delta(s, z) = 11$

最短路径：例子



- 顶点 x 到顶点 s 的最短路径
  - $p = \langle x, z, s \rangle$
  - $w(p) = \delta(x, s) = 5$

最短路径的性质（1）

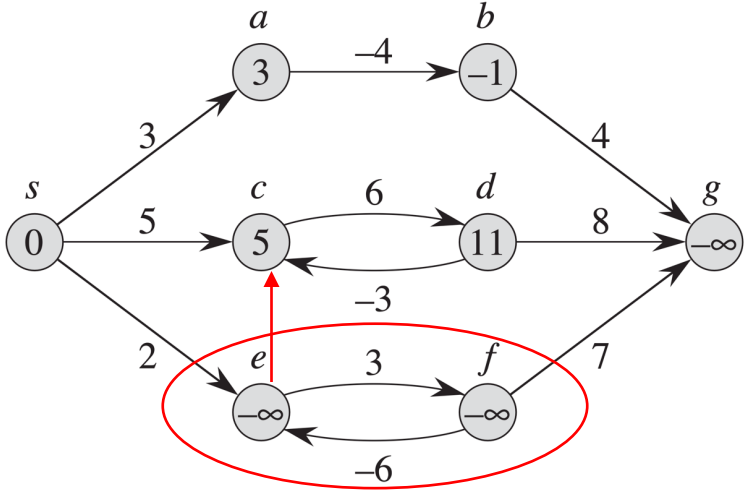
- 如果 $p = \langle v_0, \ldots, v_k \rangle$ 是 v_0 到 v_k 的一条最短路径，那么其任意子路径 $p_{ij} = \langle v_i, \ldots, v_j \rangle$ $(0 \le i < j \le k)$ 是 v_i 到 v_j 的一条最短路径。

- **证明（反证法）**：如果 p_{ij} 不是 v_i 到 v_j 的一条最短路径，我们假设存在另一条 v_i 到 v_j 的路径 p'_{ij} 满足 $w(p'_{ij}) < w(p_{ij})$ 。由此，

  我们可以构造一条 v_0 到 v_k 的新路径 $p' = v_0 \overset{p_{0i}}{\rightarrow} v_i \overset{p'_{ij}}{\rightarrow} v_j \overset{p_{jk}}{\longrightarrow} v_k$ 使得 $w(p') < w(p)$，与 $p = \langle v_0, \ldots, v_k \rangle$ 是 v_0 到 v_k 的一条最短路径这一事实相矛盾。

最短路径的性质（2）

- （假设存在负权重）如果图中有负权重的环路，则从某顶点出发至通过该环路中的顶点可达的任意顶点的最短路径权重为负无穷。
- 例如下图中顶点 s, e, f, g 的情形



最短路径的性质（3）

- 假设排除存在负权重环路的情形，则任意两顶点间的最短路径必不包含任何环路。
  - **正权重环路：**（反证法）设 $p = \langle v_0, \ldots, v_k \rangle$ 为 v_0 到 v_k 的一条最短路径，如果该路径中存在环路 $c = \langle v_i, \ldots, v_j \rangle$ $(v_i = v_j)$ 且 $w(c) > 0$，那么将 c 从 p 中去除得到的新路径 $p' = \langle v_0, \ldots, v_i, v_{j+1}, \ldots, v_k \rangle$ 也是 v_0 到 v_k 的一条路径且 $w(p') = w(p) - w(c) < w(p)$，与 p 为 v_0 到 v_k 的一条最短路径相矛盾。
  - **0 权重环路：**将 0 权重环路去除后，路径的权重保持不变，因此去除 0 权重环路对最短路径的权重将不产生任何影响
  - 接下来，我们将只考虑最短路径为简单路径即不包含任何重复顶点的路径的情形。任何包含 k 个顶点的简单路径必包含 $k - 1$ 条边。

最短路径问题的基本形式

- 单源最短路径问题 (*)
  - 给定起点 s，计算 s 到图中其余所有顶点的最短路径
- 单对顶点最短路径问题
  - 给定起点 s 和终点 t，计算从 s 到 t 的最短路径
  - 可以使用单源最短路径算法构建查询表来解决
- 全部顶点最短路径问题
  - 计算图中所有顶点对间的最短路径
  - 简单的解决方案是对图中每个顶点执行单源最短路径算法
  - 更高效的解决方案：课本第25章

课程内容

- 最短路径问题
- 单源最短路径算法
  - Bellman–Ford 算法
  - Dijkstra’s 算法

单源最短路径算法

- Bellman–Ford 算法
- Dijkstra’s 算法

Bellman–Ford 算法

- **输入**：图 $G = (V, E, w)$，源点 $s \in V$（权重 w 可以为负）

- **输出**：(1) 如果存在 s 可达的负环路，则返回 FALSE；(2) 否则返回 TRUE，并输出 s 到其余顶点 $v \in V \setminus \{s\}$ 的一条最短路径及其权重。

Bellman–Ford 算法

Bellman–Ford(G, s)

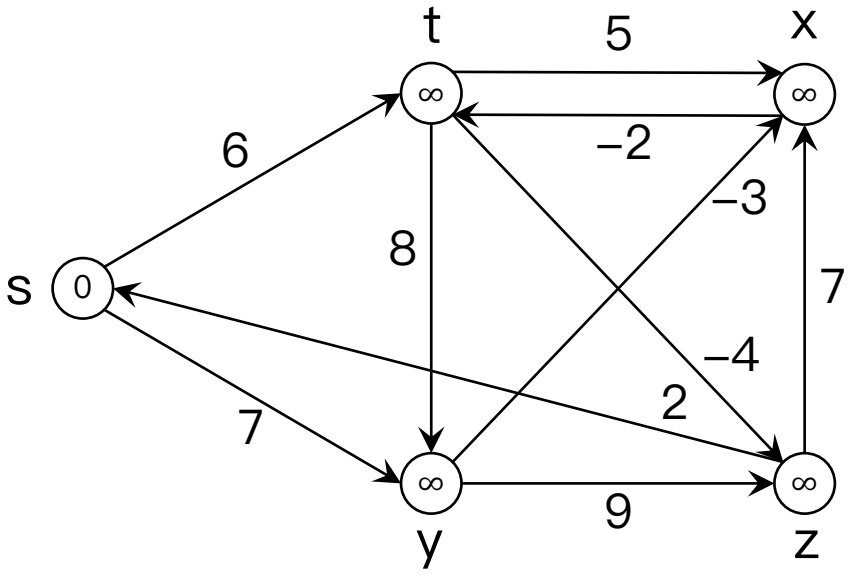
1: **For each** $v \in V$ **do:**
2: $\quad v.d = \infty,\ v.\pi = \text{NIL}$
3: $s.d = 0$
4: **For** $i = 1$ **to** $|V| - 1$ **do:**
5: $\quad$ **For each** $(u, v) \in E$ **do:**
6: $\quad\quad$ **If** $v.d > u.d + w(u, v)$ **then:**
7: $\quad\quad\quad v.d = u.d + w(u, v),\ v.\pi = u$
8: **For each** $(u, v) \in E$ **do:**
9: $\quad$ **If** $v.d > u.d + w(u, v)$ **then:**
10: $\quad\quad$ **Return** FALSE
11: **Return** TRUE

初始化步骤，其中 $v.d$ 表示 s 到 v 的最短路径权重的上界，$v.\pi$ 表示当前 s 到 v 最短路径中 v 的前序顶点

在 $|V| - 1$ 轮迭代中，逐渐降低对 s 到 v 的最短路径权重的上界 $v.d$ 的估计，直至其收敛到 $\delta(s, v)$，并实时更新其前序顶点 $v.\pi$

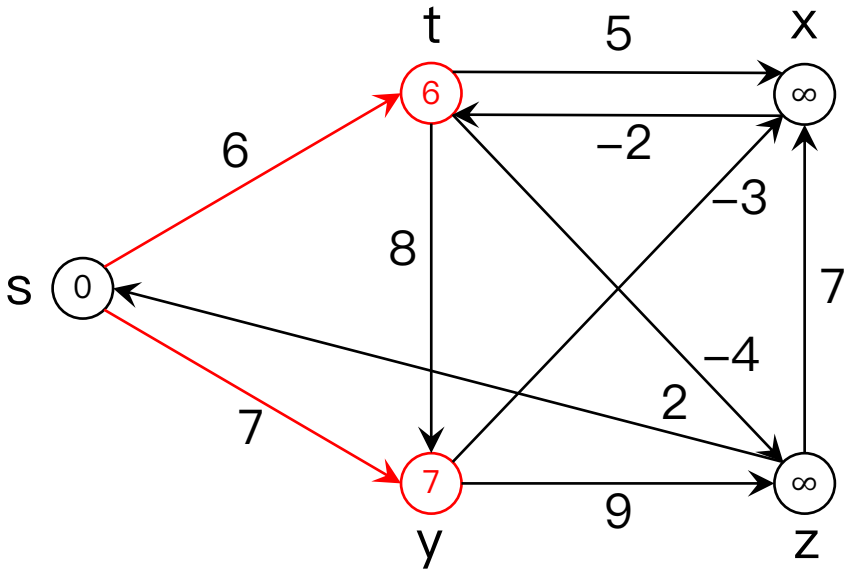
在 $|V| - 1$ 轮迭代结束以后，如果仍然有未收敛的顶点，则说明存在负权重的环路；否则，全部顶点的权重已经收敛，返回最短路径

Bellman–Ford 算法：初始化



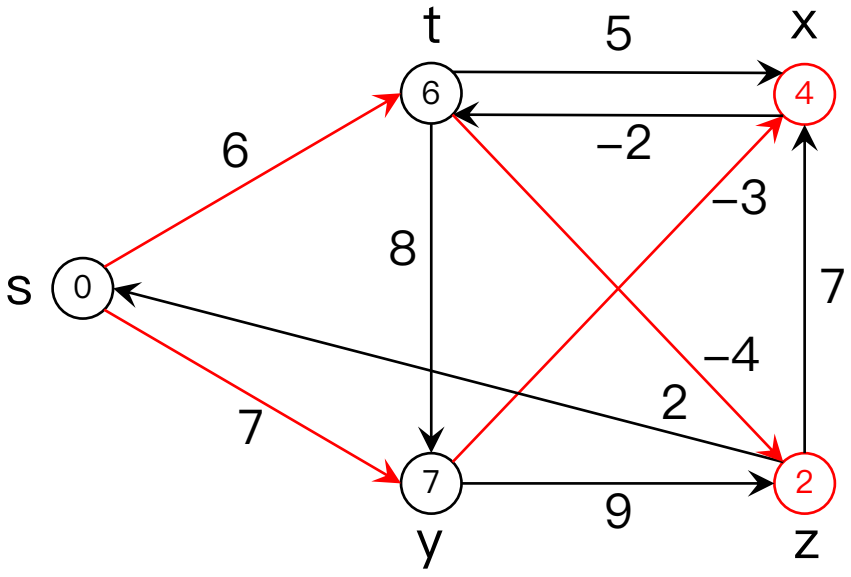
Vertex	$v.d$	$v.\pi$
s	0	NIL
t	∞	NIL
x	∞	NIL
y	∞	NIL
z	∞	NIL

Bellman–Ford 算法：第 1 轮



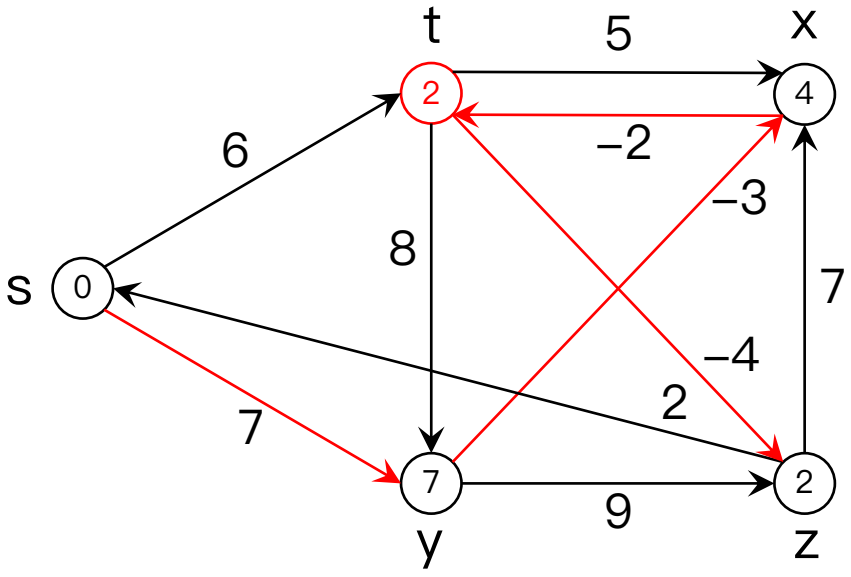
Vertex	$v.d$	$v.\pi$
s	0	NIL
t	6	s
x	∞	NIL
y	7	s
z	∞	NIL

Bellman–Ford 算法：第 2 轮



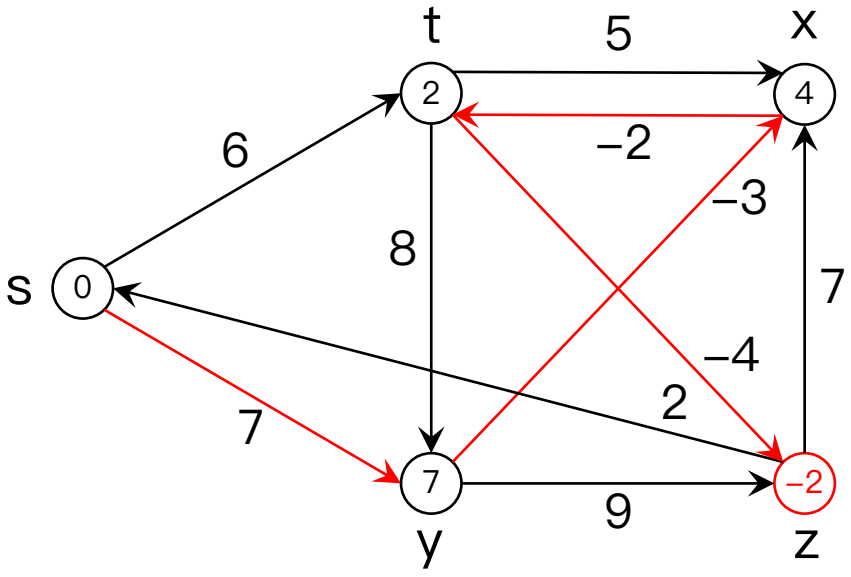
Vertex	$v.d$	$v.\pi$
s	0	NIL
t	6	s
x	4	y
y	7	s
z	2	t

Bellman–Ford 算法：第 3 轮



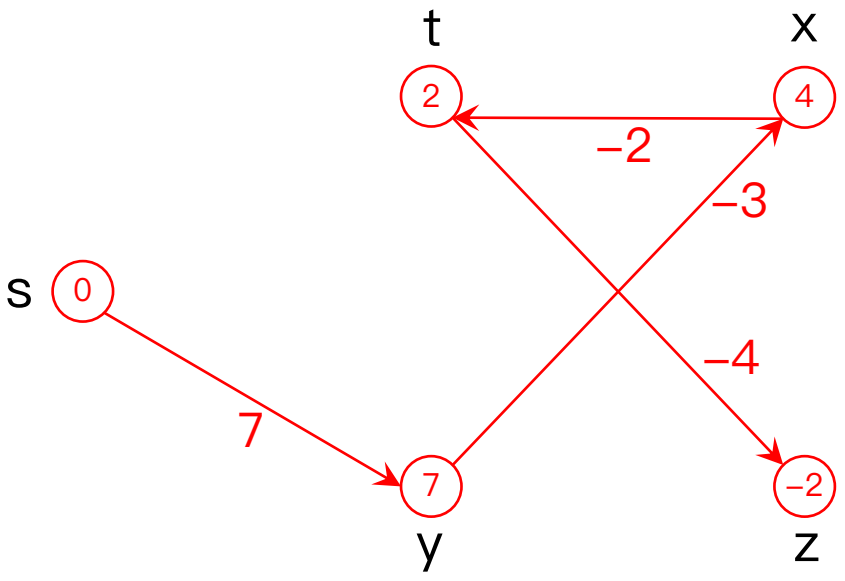
Vertex	$v.d$	$v.\pi$
s	0	NIL
t	2	x
x	4	y
y	7	s
z	2	t

Bellman–Ford 算法：第 4 轮



Vertex	$v.d$	$v.\pi$
s	0	NIL
t	2	x
x	4	y
y	7	s
z	−2	t

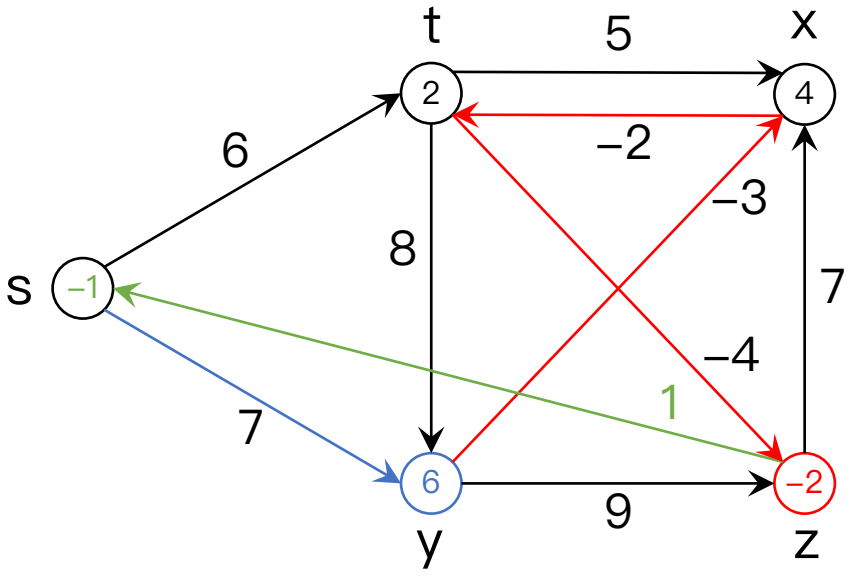
Bellman–Ford 算法：结果



Vertex	$v.d$	$v.\pi$
s	0	NIL
t	2	x
x	4	y
y	7	s
z	−2	t

在本例子中，B–F算法在第 4 轮后收敛，返回 TRUE

Bellman–Ford 算法：存在负环路



Vertex	$v.d$	$v.\pi$
s	−1	z
t	2	x
x	4	y
y	6	s
z	−2	t

负权重环路会导致B–F算法永远无法收敛！！！
因此，$|V|-1$ 轮后如果未收敛，则返回 FALSE

Bellman–Ford 算法：时间复杂度

Bellman–Ford(G, s)

1: **For each** $v \in V$ **do:**
2: $\quad v.d = \infty,\ v.\pi = \text{NIL}$
3: $s.d = 0$
4: **For** $i = 1$ **to** $|V| - 1$ **do:**
5: $\quad$ **For each** $(u, v) \in E$ **do:**
6: $\quad\quad$ **If** $v.d > u.d + w(u, v)$ **then:**
7: $\quad\quad\quad v.d = u.d + w(u, v),\ v.\pi = u$
8: **For each** $(u, v) \in E$ **do:**
9: $\quad$ **If** $v.d > u.d + w(u, v)$ **then:**
10: $\quad\quad$ **Return** FALSE
11: **Return** TRUE

第1–3行的时间复杂度 $\Theta(n)$，其中 $n = |V|$

第5–7行的时间复杂度为 $\Theta(m)$，其中 $m = |E|$；第 4 行的迭代最多运行 $n - 1$ 轮，最少运行 2 轮，因此第4–7行的时间复杂度为 $O(nm)$

第8–10行的时间复杂度为 $O(m)$

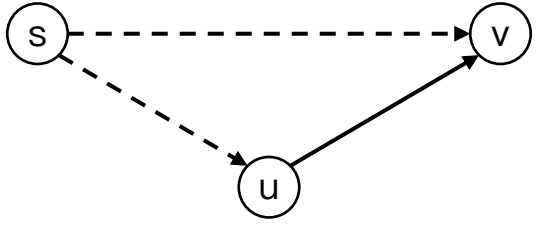
因此，B–F算法的时间复杂度为 $\Theta(n) + O(nm) + O(m) = O(nm)$

Bellman–Ford 算法：正确性

- **引理 1 (三角不等式)**：图 G 中的任意源点 $s \in V$ 和边 $(u, v) \in E$ 都满足如下性质：$\delta(s, v) \leq \delta(s, u) + w(u, v)$
- **引理 2 (上界性)**：B–F 算法中 $v.d \geq \delta(s, v)$ 永远成立且当达到 $v.d = \delta(s, v)$ 后 $v.d$ 不再改变
- **引理 3 (收敛性)**：假设 $p = \langle s \to u, v \rangle$ 为 s 到 v 的一条最短路径。B–F 算法中，如果在第 $i' < i$ 轮中 $u.d = \delta(s, u)$ 成立，那么第 i 轮执行 (u, v) 和 $v.d$ 的路径松弛后，我们可以得到 $v.d = \delta(s, v)$ 成立
- **引理 4 (路径松弛)**：假设路径 $p = \langle v_0, v_1, \ldots, v_k \rangle$ 为 $s = v_0$ 到 v_k 的最短路径。B–F 算法中先后执行 $(v_0, v_1), \ldots, (v_{k-1}, v_k)$ 的路径松弛操作后，我们可以得到 $v_k.d = \delta(s, v_k)$

Bellman–Ford 算法：正确性

- **引理 1 (三角不等式)：**图 G 中的任意源点 $s \in V$ 和边 $(u, v) \in E$ 都满足如下性质：$\delta(s, v) \leq \delta(s, u) + w(u, v)$
- **证明：**假设 p 为 s 到 v 的一条最短路径，即不存在任何 s 到 v 的路径权重小于 $w(p)$。而 $\delta(s, u) + w(u, v)$ 等于 s 到 u 的一条最短路径加上边 (u, v) 这一条 s 到 v 的路径的权重，所以其不可能低于 $\delta(s, v)$。



Bellman–Ford 算法：正确性

- **引理 2 (上界性)**：B–F 算法中 $v.d \geq \delta(s,v)$ 永远成立且当达到 $v.d = \delta(s,v)$ 后 $v.d$ 不再改变。
- **证明**：（数学归纳法）
  1. 在初始状态下，所有 $v \in V \setminus \{s\}$ 都有 $v.d = \infty$，因此， $v.d \geq \delta(s,v)$ 显然成立。而对 s 自身，由于不存在负环路，$s.d = \delta(s,s) = 0$ 也成立。
  2. 假设执行路径松弛操作（即 B–F 算法的 5–7 行）前 $x.d \geq \delta(s,x)$ 对所有 $x \in V$ 都成立。此时，如果 $v.d$ 被边 (u,v) 更新，我们有
  $$v.d = u.d + w(u,v) \geq \delta(s,u) + w(u,v) \geq \delta(s,v)$$
  - 因此，$v.d \geq \delta(s,v)$ 永远成立，同时该结论也自然引出 $v.d = \delta(s,v)$ 后 $v.d$ 不再改变这一推论，因为 $v.d$ 在算法执行过程中只能减小或不变。

Bellman–Ford 算法：正确性

- **引理 3 (收敛性)**：假设 $p = \langle s \to u, v \rangle$ 为 s 到 v 的一条最短路径。 B–F 算法中，如果在第 $i' < i$ 轮中 $u.d = \delta(s,u)$ 成立，那么第 i 轮执行 (u,v) 和 $v.d$ 的路径松弛后，我们可以得到 $v.d = \delta(s,v)$
- **证明**：根据路径松弛的规则，当执行边 (u,v) 和 $v.d$ 的路径松弛后，$v.d \leq u.d + w(u,v) = \delta(s,u) + w(u,v) = \delta(s,v)$；另外根据上界性可以得到 $v.d \geq \delta(s,v)$。因此，$v.d = \delta(s,v)$

Bellman–Ford 算法：正确性

- **引理 4 (路径松弛)**：假设路径 $p = \langle v_0, v_1, \ldots, v_k \rangle$ 为 $s = v_0$ 到 v_k 的最短路径。B–F 算法中先后执行 $(v_0, v_1), \ldots, (v_{k-1}, v_k)$ 的路径松弛操作后，我们可以得到 $v_k.d = \delta(s, v_k)$
- **证明**：（数学归纳法）
  1. 在初始状态下，$v_0.d = s.d = 0 = \delta(s, s)$ 成立。
  2. 根据收敛性规则，假设 $v_{i-1}.d = \delta(s, v_{i-1})$ 成立，那么执行 (v_{i-1}, v_i) 和 $v_i.d$ 的路径松弛操作后， $v_i.d = \delta(s, v_i)$ 成立。
  - 因此，我们得到了上述结论。

Bellman–Ford 算法：正确性

- **引理 5**：假设 G 中不存在 s 可达的负环路，那么 G 中每个 s 可达的顶点 v 经过 $|V|-1$ 轮路径松弛迭代后都满足 $v.d = \delta(s,v)$。
- **证明**：
  - 在不存在负环路的情况下，所有最短路径均为简单路径，G 中的简单路径最多只能包含 $|V|-1$ 条边。
  - 经 $|V|-1$ 轮迭代后，$s = v_0$ 到 $v_k = v$ 的最短路径 $p = \langle v_0, v_1, \ldots, v_k \rangle$ 必然先后执行过 $(v_0, v_1), \ldots, (v_{k-1}, v_k)$ 的路径松弛，满足 $v.d = \delta(s,v)$。

Bellman–Ford 算法：正确性

- **主定理：**在图 $G = (V, E, w)$ 和源点 $s \in V$ 上运行 Bellman–Ford 算法后，如果 G 中存在 s 可达的负环路，则返回 FALSE；否则返回 TRUE 并输出 s 到每个顶点 $v \in V \setminus \{s\}$ 的一条最短路径。
- **证明：**
  1. 如果 v 是 s 不可达的，则 $v.d = \delta(s, v) = \infty$；
  2. 若不存在 s 可达的负环路，根据引理 5 经过 $|V| - 1$ 轮迭代后 $v.d = \delta(s, v)$。此时 $v.d = \delta(s, v) \leq \delta(s, u) + w(u, v) = u.d + w(u, v)$，算法返回 TRUE；
  3. 如果存在 s 可达的负环路，假设环路为 $c = (v_0, \ldots, v_k)$，其中 $v_0 = v_k$ 且 $\sum_{i=1}^{k} w(v_{i-1}, v_i) < 0$。此时我们假设 B–F 算法输出 TRUE，则对 $i = 1, \ldots, k$ 都满足 $v_i.d \leq v_{i-1}.d + w(v_{i-1}, v_i)$。

Bellman–Ford 算法：正确性

- **主定理：**在图 $G = (V, E, w)$ 和源点 $s \in V$ 上运行 Bellman–Ford 算法后，如果 G 中存在 s 可达的负环路，则返回 FALSE；否则返回 TRUE 并输出 s 到每个顶点 $v \in V \setminus \{s\}$ 的一条最短路径。
- **证明（续）：**

  累加前面的不等式，可以得到

$$\sum_{i=1}^{k} v_i.d \leq \sum_{i=1}^{k} \left(v_{i-1}.d + w(v_{i-1}, v_i)\right) < \sum_{i=1}^{k} v_{i-1}.d$$

  由于 $v_0 = v_k$，我们有 $\sum_{i=1}^{k} v_i.d = \sum_{i=1}^{k} v_{i-1}.d$，与上述不等式相矛盾，因此 B–F 算法输出只能为 FALSE。

单源最短路径算法

- Bellman–Ford 算法
- Dijkstra’s 算法

Dijkstra's 算法

- Bellman–Ford 算法的时间复杂度是 $O(nm)$

- 如果我们限制图中的边权重是**非负的**（在实际情况中非常普遍，例如距离、时间、计数等），有没有更为高效的单源最短路径算法?

Dijkstra's 算法

- **输入**：图 $G = (V, E, w)$，源点 $s \in V$（权重 w 非负）

- **输出**：源点 s 到所有顶点 $v \in V \setminus \{s\}$ 的一条最短路径及其权重。

Dijkstra's 算法

Dijkstra(G, s)

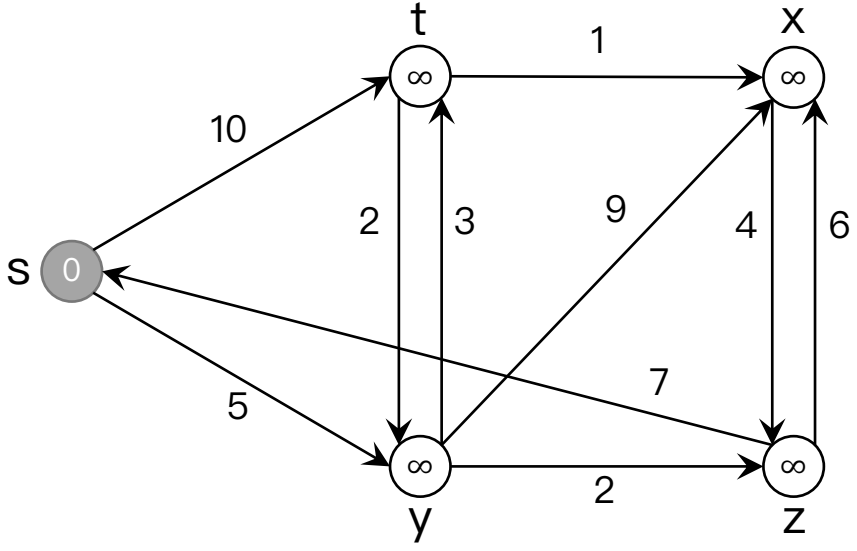
1: **For each** $v \in V$ **do:**
2: $v.d = \infty,\ v.\pi = \text{NIL}$
3: $s.d = 0$
4: $S = \emptyset,\ Q = V$
5: **While** $Q \neq \emptyset$ **do:**
6: $u = \text{Extract–Min}(Q)$
7: $S = S \cup \{u\}$
8: **For each** $v \in G.Adj[u]$ **do:**
9: **If** $v.d > u.d + w(u, v)$ **then:**
10: $v.d = u.d + w(u, v),\ v.\pi = u$

初始化步骤，其中 $v.d$ 和 $v.\pi$ 的初始化与B–F算法相同。我们用 S 表示当前最短路径已确定的顶点，Q 表示顶点以 $v.d$ 为键的优先队列

每一轮迭代从 Q 中取出当前 d 值最小的顶点 u （贪心策略）并将其加入 S，表示其最短路径已经确定，随后对 u 的所有出边执行和B–F算法相同路径松弛操作

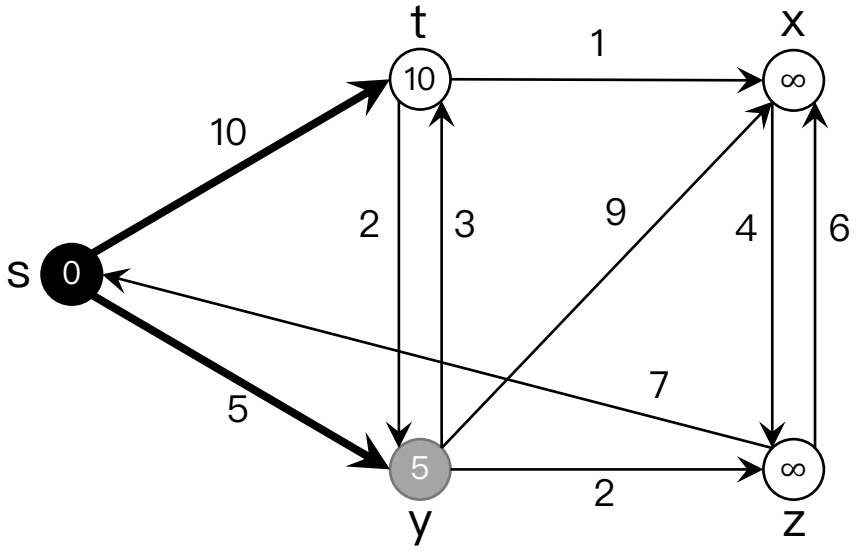
如此往复，直到所有顶点最短路径都确定为止

Dijkstra's 算法：例子



Vertex	$v.d$	$v.\pi$
s	0	NIL
t	∞	NIL
x	∞	NIL
y	∞	NIL
z	∞	NIL

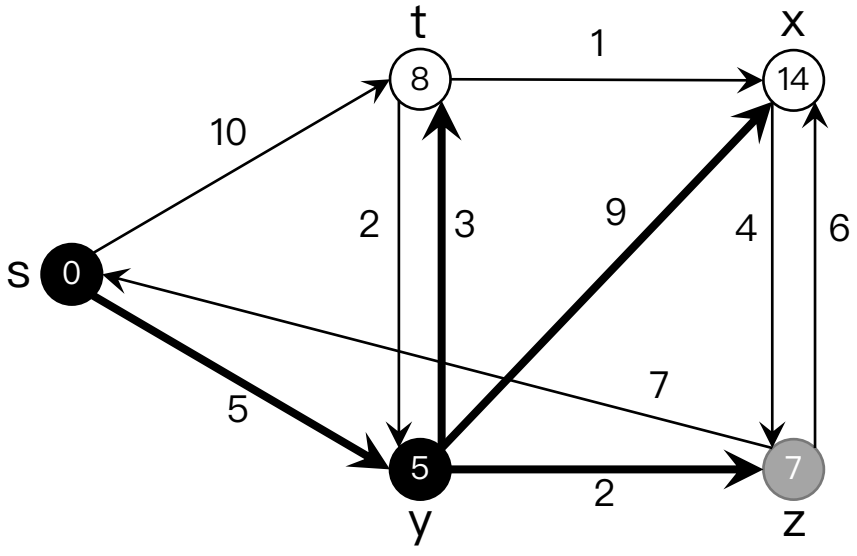
Dijkstra's 算法：例子



Vertex	$v.d$	$v.\pi$
t	10	s
x	∞	NIL
y	5	s
z	∞	NIL

Vertex	$v.d$	$v.\pi$
s	0	NIL

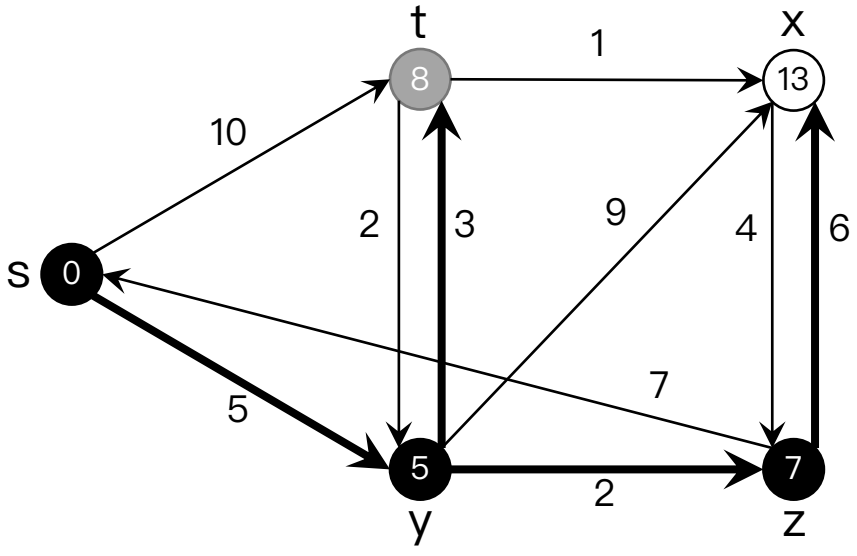
Dijkstra's 算法：例子



Vertex	$v.d$	$v.\pi$
t	8	y
x	14	y
z	7	y

Vertex	$v.d$	$v.\pi$
s	0	NIL
y	5	s

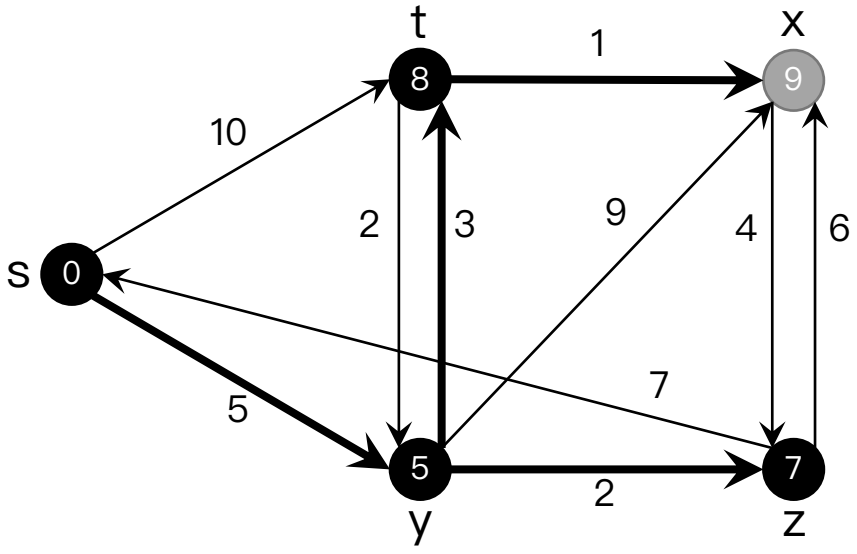
Dijkstra's 算法：例子



Vertex	$v.d$	$v.\pi$
t	8	y
x	13	z

Vertex	$v.d$	$v.\pi$
s	0	NIL
y	5	s
z	7	y

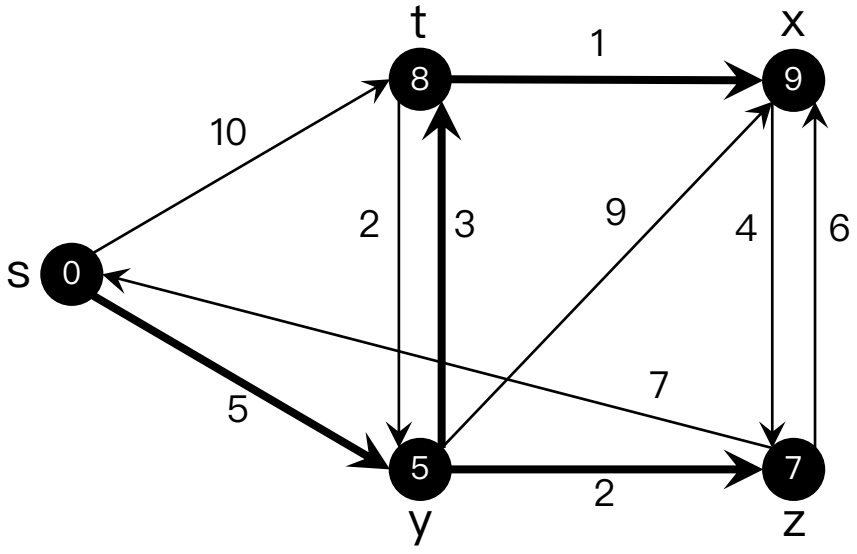
Dijkstra's 算法：例子



Vertex	$v.d$	$v.\pi$
x	9	t

Vertex	$v.d$	$v.\pi$
s	0	NIL
y	5	s
z	7	y
t	8	y

Dijkstra's 算法：例子



Vertex	$v.d$	$v.\pi$
s	0	NIL
y	5	s
z	7	y
t	8	y
x	9	t

Dijkstra's 算法：正确性（1）

- **定理**：在图 $G = (V, E, w)$，其中所有边权重 $w(e) \geq 0$，和源点 $s \in V$ 上运行 Dijkstra's 算法，结束时每个顶点 $u \in V$ 必然满足 $u.d = \delta(s, u)$。
- **证明**：
  - 为了证明上述定理，我们需要证明在第 5–10 行的 While 循环开始时，S 中的每个顶点 u 都满足 $u.d = \delta(s, u)$
  - 得到上述结论只需证明在 u 加入 S 时已满足 $u.d = \delta(s, u)$，根据 B–F 算法已经证明过的上界性（引理 2），$u.d = \delta(s, u)$ 在之后的运行过程中会永远成立。
  - **引理 2 (上界性)**：B–F 算法中 $v.d \geq \delta(s, v)$ 永远成立且当达到 $v.d = \delta(s, v)$ 后 $v.d$ 不再改变。

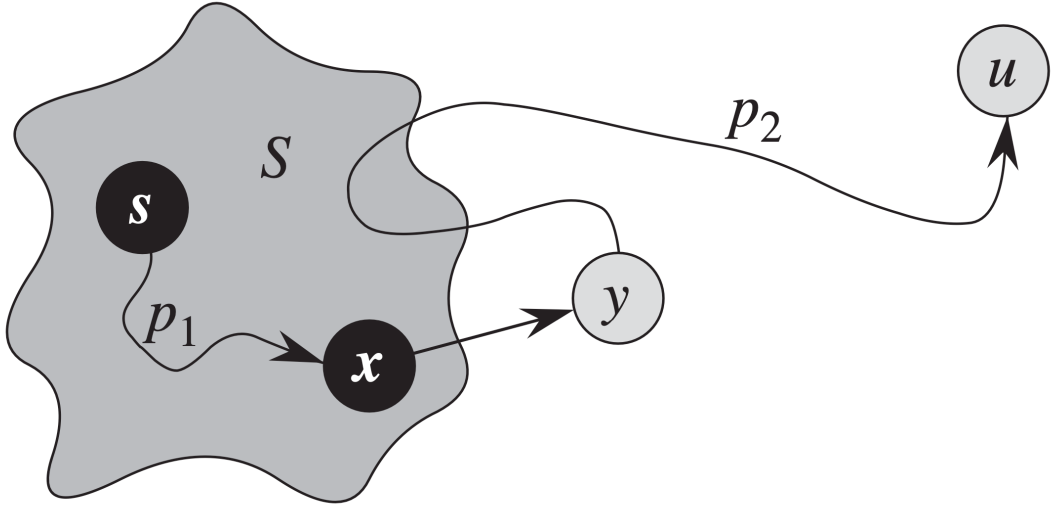
Dijkstra's 算法：正确性（2）

- **定理：**在图 $G = (V, E, w)$，其中所有边权重 $w(e) \geq 0$，和源点 $s \in V$ 上运行 Dijkstra's 算法，结束时每个顶点 $u \in V$ 必然满足 $u.d = \delta(s, u)$。
- **证明：**
  - **初始化：**Dijkstra's 算法开始运行时 $S = \emptyset$ 结论成立
  - **更新维护：**我们希望使用反证法证明在每个 While 循环被加入 S 顶点 u 都满足 $u.d = \delta(s, u)$。假设 u 是第一个被加入 S 时 $u.d > \delta(s, u)$ 的顶点，我们考虑 u 被加入 S 的那一轮 While 循环以得出 $u.d = \delta(s, u)$ 的结论，从而得出与假设相矛盾以完成证明。

Dijkstra's 算法：正确性（3）

- **证明：**
  - 首先，当 $u = s$ 时，因为 s 一定是第一个被加入 S 的顶点，且必然满足 $s.d = \delta(s, s) = 0$ 与假设相矛盾
  - 然后我们考虑 $u \neq s$ 的情况，因为 s 已加入 S，我们有 $S \neq \emptyset$。我们同样排除 u 是 s 不可达的，否则 $u.d = \delta(s, u) = \infty$ 也与假设相矛盾。在排除上述情况后，我们假设路径 p 为 s 到 u 的一条最短路径。在 u 加入 S 前，路径 p 连通了 $s \in S$ 和 $u \in V \setminus S$，其中 y 是路径 p 上第一个不在 S 里的顶点，x 是路径 p 上 y 的前序顶点。此时，我们可以将路径 p 分解为 $\left\langle s \xrightarrow{p_1} x, y \xrightarrow{p_2} u \right\rangle$ 的形式。

Dijkstra’s 算法：正确性（图示）



Dijkstra's 算法：正确性（4）

- **证明：**
  - 基于上述对路径 p 的分解，我们证明 $y.d = \delta(s,y)$ 。首先，因为我们假设 u 是第一个被加入 S 时 $u.d > \delta(s,u)$ 的顶点，所以当 x 被加入 S 时 $x.d = \delta(s,x)$。此时根据收敛性（引理 3）我们得到 $y.d = \delta(s,y)$。
  - **引理 3 (收敛性)：**假设 $p = \langle s \to u, v \rangle$ 为 s 到 v 的一条最短路径。 B–F 算法中，如果在第 $i' < i$ 轮中 $u.d = \delta(s,u)$ 成立，那么第 i 轮执行 (u,v) 和 $v.d$ 的路径松弛后，我们可以得到 $v.d = \delta(s,v)$ 成立。

Dijkstra's 算法：正确性（5）

- **证明：**
  - 在 $y.d = \delta(s,y)$ 的基础上，我们可以推出矛盾。因为 y 在最短路径上的次序在 u 之前，且所有边权重非负，我们得到
    $$y.d = \delta(s,y) \leq \delta(s,u) \leq u.d$$
  - 考虑 $u, y \in V \setminus S$ 且在 While 循环中 u 被 Extract–Min 选择，我们又得到 $u.d \leq y.d$
  - 因此，只可能是 $u.d = y.d$ 且 $u.d = \delta(s,u)$，与假设相矛盾。
  - 综上所述，当 u 被加入 S 时，必然满足 $u.d = \delta(s,u)$。
  - **算法结束时：**所有顶点都已经被加入 S 中，因此所有顶点 u 都满足 $u.d = \delta(s,u)$，结论得证。

Dijkstra's 算法

Dijkstra(G, s)

1: **For each** $v \in V$ **do:**
2: $v.d = \infty,\ v.\pi = \text{NIL}$
3: $s.d = 0$
4: $S = \emptyset,\ Q = V$
5: **While** $Q \neq \emptyset$ **do:**
6: $u = \text{Extract–Min}(Q)$
7: $S = S \cup \{u\}$
8: **For each** $v \in G.Adj[u]$ **do:**
9: **If** $v.d > u.d + w(u, v)$ **then:**
10: $v.d = u.d + w(u, v),\ v.\pi = u$

初始化步骤，其中 $v.d$ 和 $v.\pi$ 的初始化与B–F算法相同。我们使用 S 表示当前最短路径已确定的顶点，Q 表示顶点以 $v.d$ 为键的优先队列

每一轮迭代从 Q 中取出当前 d 值最小的顶点 u （贪心策略）并将其加入 S，表示其最短路径已经确定，随后对 u 的所有出边执行和B–F算法相同路径松弛操作

如此往复，直到所有顶点最短路径都确定为止

Dijkstra's 算法：时间复杂度

- **初始化步骤**：与B–F算法相同，时间复杂度为 $\Theta(n)$
- **迭代步骤**
  - 分析的关键在于对优先队列 Q 的操作包括 Insert (第 4 行构建 $Q = V$)，Extract–Min (第 6 行从 Q 中取出键值最小的顶点) 和 Decrease–Key (第 10 行更新顶点键值 $v.d$)
  - 因为每个顶点被加入 S 的次数 1 次，顶点被加入 S 时它的所有出边被遍历 1 次，所以 While 循环执行次数为 n 次，第 8–10 行的 For 循环执行次数为 m 次。
  - 因此，Insert 被执行 n 次，Extract–Min被执行 n 次，Decrease–Key 被执行 m 次。

Dijkstra's 算法：时间复杂度

- **方案 1：**使用线性表实现优先队列
- 将 $v.d$ 按照顶点编号存储在长度为 n 的数组中
- Insert 和 Decrease–Key 的时间复杂度为 $O(1)$，Extract–Min 的时间复杂度为 $O(n)$
- Insert 被执行 n 次，Extract–Min被执行 n 次，Decrease–Key被执行 m 次，总时间复杂度为

$$O(n) + O(n^2) + O(m) = O(n^2 + m) = O(n^2)$$

Dijkstra's 算法：时间复杂度

- **方案 2**：使用最小堆实现优先队列
- 构建最小堆的时间复杂度为 $O(n)$，Extract–Min 和 Decrease–Key 的时间复杂度为 $O(\log n)$
- Insert 被执行 n 次，Extract–Min被执行 n 次，Decrease–Key被执行 m 次，总时间复杂度为
  $O(n) + O(n \log n) + O(m \log n) = O\big((n+m)\log n\big) = O(m \log n)$
- 如果图是稀疏的，假设 $m = o(n^2/\log n)$，方案 2 相对于方案 1 的时间复杂度更低

Dijkstra's 算法：时间复杂度

- 还有时间复杂度更低的方案吗?

- **方案 3**：使用 Fibonacci 堆实现优先队列（参考课本第 19 章，在本课程中不做要求）
- 时间复杂度为 $O(n \log n + m)$

Dijkstra’s 算法：与其他算法的对比

- Dijkstra’s 算法 vs. 广度优先搜索算法
  - 当所有边权重为 1 时，广度优先搜索算法等价于Dijkstra’s算法

- Dijkstra’s 算法 vs Prim’s 最小生成树算法
  - 学过最小生成树算法后，同学们可以比较一下两个算法的异同

课程总结

- 最短路径问题
- 单源最短路径算法
  - Bellman–Ford 算法
  - Dijkstra’s 算法
- 扩展阅读
  - 每对顶点最短路径算法（课本第25章）
  - Fibonacci 堆（课本第19章）优先队列改进 Dijkstra’s 算法