



上海师范大学  
Shanghai Normal University

# 《算法设计与分析》

## 7-最短路径 (Shortest Path)

杨启哲

上海师范大学信机学院计算机系

2025 年 10 月 27 日



- › 最短路径问题介绍
- › 单源最短路径
- › 所有节点对的最短路径问题

## 最短路径问题介绍

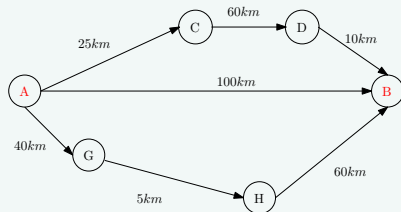
在上节课中，我们介绍了 BFS 可以求得图中的最短路径。但是，这是基于图中所有边的**权重都是相同的**而得出的。

现实生活中的图不同的边可能是有不同的距离。

### 地图上的距离

如右图所示，假设你现在要从 A 地去往 B 地，各个路段的距离如图所示。

哪条路径是最短的呢？



我们引入**带权重的图 (weighted graph)**这一概念。

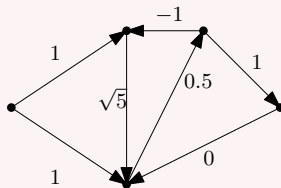
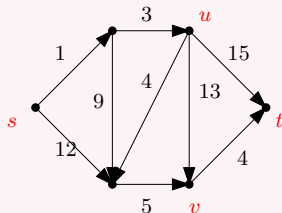
## 定义 1

## [带权重的图].

一个带权重的有向图可以表示为  $G = (V, E, \omega)$ , 其中  $(V, E)$  是一个有向图,  $\omega : E \rightarrow \mathbb{R}$  是一个边上的权重函数, 其给每条边赋予了一个权重。

## 例 2.

下列给出了几个带权重的有向图的例子, 其中边上的标注即是这条边的权重。



现在我们定义上面两个点的最短路径。

## 定义 3

## [最短路径].

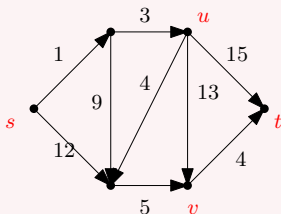
给定一张带权重的图  $G = (V, E)$  和上面的两个点  $u, v$ , 图上的一条路径  $\pi$  可以由其经过的节点序列所表示, 即  $\pi = \langle u_1, \dots, u_n \rangle$ , 其权重  $\omega(\pi)$  定义为构成该路径所有边之和, 则我们可以定义  $u$  到  $v$  的最短路径权重  $\omega(u, v)$  为:

$$\omega(u, v) = \begin{cases} \infty & \text{如果不存在 } u \text{ 到 } v \text{ 的路径} \\ \min \omega(\pi) : u \xrightarrow{\pi} v & \text{如果存在 } u \text{ 到 } v \text{ 的路径} \end{cases}$$

## 例 4.

在上述例子的左图当中, 我们有:

- $\omega(s, u) = 4, \omega(s, v) = 13.$
- $\omega(s, t) = 17.$



最短路径问题存在若干变种：

- 单源最短路径问题。
- 单目的地最短路径问题
- 单节点对最短路径问题。
- 所有节点对的最短路径问题。

最短路径的一个核心性质便是，最短路径的任何子路径也是最短路径。

## 定理 5.

给定带权重的有向图  $G = (V, E, \omega)$ ，设  $\pi = \langle v_0, \dots, v_k \rangle$  是一条从  $v_0$  到  $v_k$  的最短路径，并定义  $\pi_{ij} = \langle v_i, \dots, v_j \rangle$ 。则  $\pi_{ij}$  是  $v_i$  到  $v_j$  的最短路径。

**证明** 假设存在  $i, j$  使得  $\pi_{ij}$  不是  $i$  到  $j$  的最短路径，则令其最短路径为  $\pi'_{ij} = \langle v_i, u_1, \dots, u_m, v_j \rangle$ 。由定义我们有  $\omega(\pi'_{ij}) < \omega(\pi_{ij})$ 。考察如下的  $u$  到  $v$  的一条路径  $\pi'$ ：

$$\pi' = \langle v_0, \dots, v_{i-1}, v_i, u_1, \dots, u_m, v_j, \dots, v_k \rangle$$

则我们有：

$$\omega(\pi') = \omega(\pi_{1i}) + \omega(\pi'_{ij}) + \omega(\pi_{j,k}) < \omega(\pi)$$

即  $\pi'$  是一条更短的路径，与假设矛盾。





一个最短路径是否会存在一个圈？

---

不可能。

- 如果存在一个权重大于 0 的圈，则删掉这个圈会得到一个更短的路径。
- 如果存在一个权重等于 0 的圈，则这个圈存在与否不会影响最短路径的权重。
- 如果存在一个权重小于 0 的圈，则最短路径的定义没有意义，因为每走一次负圈，路径权重就会减小。

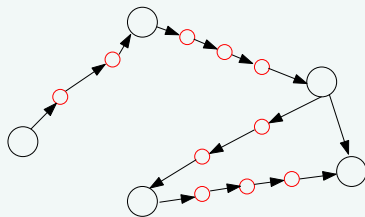
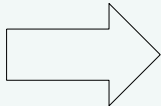
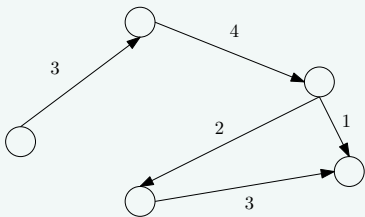
## 推论 6.

给定一个  $n$  个点的带权重的有向图，任何最短路径最多经过  $n - 1$  条不同的边。

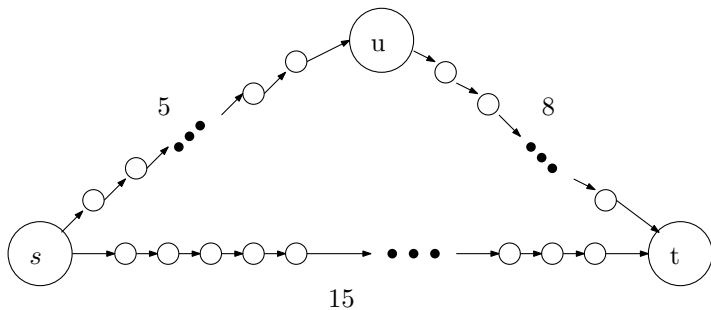
## 单源最短路径

我们说过，BFS 可以求得权重一致的图中的最短路径。那有没有办法使得一个带权重的图变成权重一致的情况那？考虑权重全是**正整数**的情况。

### 增加虚拟节点

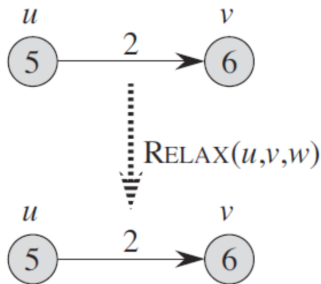
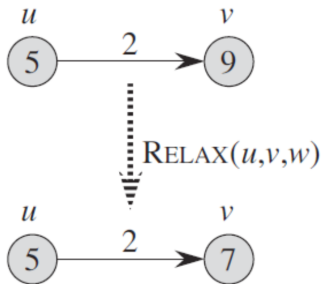


再来考虑以下虚拟节点和真正节点的作用区别。



- 当访问到真正的节点的时候路径长度才起到作用。
- $(u, t)$  这条边会对  $s$  到  $t$  的路径产生优化作用。

假设  $s$  出发到其余各个点已经存在了一条路径，则我们称一条边  $(u, v)$  的**松弛操作**是指，如果存在一条从  $s$  到  $u$  的路径，使得其权重加上  $(u, v)$  的权重小于从  $s$  到  $v$  的路径，则我们可以用这条路径来更新  $s$  到  $v$  的路径。



$$\lambda[v] = \min\{\lambda[v], \lambda[u] + \omega((u, v))\}$$

当权重都为**正数**的时候，考察  $s$  到  $t$  的任何一条最短路径：

$$s \rightarrow u_1 \rightarrow u_2 \rightarrow \dots \rightarrow u_k \rightarrow t$$

记  $t = u_{k+1}$ ，我们有如下观察事实：

## 事实 7.

对于任意的  $i \in [1, k]$ ，我们有  $\omega(s, u_i) \leq \omega(s, u_{i+1})$ ，也就是说  $u_i$  对于  $u_{i+1}$  说一定是距离  $s$  更近的点。

---

因此我们有了个简单的想法：

- 每次找到还**未考虑过**的目前距离  $s$  最近的点，并用该点连出去的边去考虑**松弛操作**。

这就是**Dijkstra 算法**的思想。

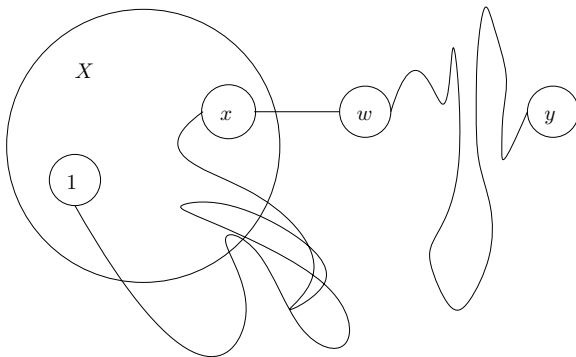
## 算法: Dijkstra

输入: 含权有向图  $G = (V, E)$ ,  $V = \{1, 2, \dots, n\}$

输出:  $G$  中顶点 1 到其余各个顶点的最短路径长度

```
1:  $X = \{1\}$ ,  $Y \leftarrow V - \{1\}$ ,  $\lambda[1] \leftarrow 0$ 
2: for  $y \leftarrow 2$  to  $n$  do
3:   if  $(1, y) \in E$  then
4:      $\lambda[y] \leftarrow \omega(1, y)$ 
5:   else
6:      $\lambda[y] \leftarrow \infty$ 
7: while  $Y \neq \emptyset$  do
8:   从  $Y$  中选取一个点  $u$ , 使得  $\lambda[u] = \min_{y \in Y} \lambda[y]$ 
9:    $X \leftarrow X \cup \{u\}$ ,  $Y \leftarrow Y - \{u\}$ 
10:  for  $y \in Y$  do
11:    if  $(u, y) \in E$  then
12:       $\lambda[y] \leftarrow \min\{\lambda[y], \lambda[u] + \omega(u, y)\}$ 
```

我们要证明，每次第 8 步有顶点  $y$  选中时，这个时候的  $\lambda[y]$  就是  $\omega(1, y)$  的值。



---

证明的**核心**在于，如果到  $y$  的最短路径上有别的  $X$  外的点，则该点对应的值一定是  $\omega(1, x)$ 。



## 定理 8.

在上述算法 Dijkstra 的过程中, 当第 8 步的点  $y$  被选中时, 我们有:  $\lambda[y] = \omega(1, y)$ .

我们对顶点离开集合  $Y$  的顺序进行归纳。第一个离开的点是 1, 从而  $\lambda[1] = \omega(1, 1)$

假设  $u_k$  是第  $k$  个离开的点, 定理对前  $k$  个点都有  $\lambda[u_k] = \omega(1, u_k)$ 。

考虑第  $k+1$  个离开的点  $y$ , 令

$$\pi = \langle 1, \dots, x, w, \dots, y \rangle$$

是 1 到  $y$  的最短路径, 其中  $x$  是在  $y$  前最迟离开  $Y$  的顶点。

$$\lambda[y] \leq \lambda[w] \leq \lambda[x] + \omega(x, w) = \omega(1, x) + \omega(x, w) = \omega(1, w) \leq \omega(1, y)$$

从而结论成立。



我们先从边的数目  $|E| = m$  和顶点的数目  $|V| = n$  进行对算法的分析。

- 2-6 行要遍历所有的顶点一次，因此时间复杂性是  $O(n)$ 。
- 算法第 8 步每次要找到最小的  $\lambda[y]$ ，因此总共需要的执行时间是  $O(n^2)$ 。
- 算法 10-12 步的循环恰好对每条边都执行了一次，因此总共需要的执行时间是  $O(m)$ 。

## 定理 9.

给定一个含有  $n$  个顶点和  $m$  条边的带权重的有向图，Dijkstra 算法可以在  $O(n^2 + m) = O(n^2)$  的时间内计算出从顶点 1 到其余各个顶点的最短路径长度。

我们可以发现，导致算法是  $O(n^2)$  的原因是每次第 8 步要找出最小的  $\lambda[y]$ 。

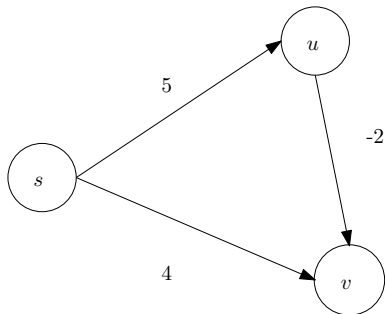
有什么办法可以使得这一步速度变快？

使用堆！

- 
- 当使用二分堆时，每次可以通过  $O(1)$  的时间获取当前的最小值，而对于堆的修改操作至多进行  $m + n + 1$  次，而每个堆运算都需要  $O(\log n)$  的时间，因此总的时间复杂度是  $O(m \log n)$ 。
  - 如果使用二分堆的推广-d 堆，如果图时稠密的，即  $m \geq n^{1+\epsilon}$ ，通过合适的 d 的选取，我们可以将结果提升至  $O(\frac{m}{\epsilon})$ 。

Dijkstra 算法的局限性是不能存在有负权重的边。

- 从  $s$  出发第一次会将  $v$  放进去。
- 从  $s$  出发第二次会将  $u$  放进去。
- $\omega[v]$  的值会在第一次确定。



Dijkstra 算法一个重要的性质是，从起始点到任意点的最短路径一定会经过比  $v$  距离更近的顶点。但这一性质在有负权重的时候是失效的。

---

- 在上述例子中， $s$  到  $v$  的最短路径首先要去一个距离更远的顶点  $u$ 。
- 在证明中， $\omega(1, w) \leq \omega(1, y)$  不再成立。



回顾一下 Relax 操作:

$$\lambda[v] = \min\{\lambda[v], \lambda[u] + \omega((u, v))\}$$

- 
- 首先这一方式说明, 从  $s$  到  $v$  的最短路径不可能超过  $s$  到  $u$  的距离加上  $(u, v)$  的权重。
  - 其次, 当  $u$  是  $s$  到  $v$  的最短路径上的倒数第二个顶点并且  $\lambda[u]$  被正确设定时, 我们可以求得  $s$  到  $v$  的最短路径。

假设  $s$  到  $t$  的最短路径如下:

$$s \rightarrow u_1 \rightarrow \dots \rightarrow u_k \rightarrow t$$

- 这意味着如果我们按  $(s, u_1), (s, u_2), \dots, (u_k, t)$  的方式去松弛, 我们便可以求得  $s$  到  $t$  的最短路径。这个序列的长度显然最长是  $|V| - 1$ 。

### 引理 10.

假设  $s$  到  $t$  的最短路径为  $\pi = \langle s = u_0, u_1, \dots, u_{k+1} = t \rangle$ , 则如果按照边  $(s, u_1), (s, u_2), \dots, (u_k, t)$  的次序去松弛, 最终获得  $\lambda[t] = \omega(\pi)$ , 且其他边的松弛操作不会对其产生影响。

但问题是我们预先并不知道这样的序列, 怎么办?

---

那就**全部尝试一遍**! 这就是**BellmanFord 算法**!

## 算法: BellmanFord

**输入:** 含权有向图  $G = (V, E)$ ,  $V = \{1, 2, \dots, n\}$

**输出:**  $G$  中顶点 1 到其余各个顶点的最短路径长度

```
1: for  $i = 2$  to  $n$  do
2:   if  $(1, i) \in E$  then
3:      $\lambda[i] \leftarrow \omega(1, i)$ 
4:   else
5:      $\lambda[i] \leftarrow \infty$ 
6: for  $i = 1$  to  $n - 1$  do
7:   for  $(u, v) \in E$  do
8:      $\lambda[v] \leftarrow \min\{\lambda[v], \lambda[u] + \omega((u, v))\}$ 
9: for  $(u, v) \in E$  do
10:  if  $\lambda[v] > \lambda[u] + \omega((u, v))$  then
11:    return "存在负圈"
```

时间复杂性:  $O(nm)$ !



算法的正确性可以通过之前的讨论获得。我们最后再讨论一个问题：

为什么松弛了  $|V| - 1$  轮之后如果还能松弛成功，就说明图中存在负圈？

---

- 因为如果路径长度  $\geq |V|$ ，那路径中一定包含一个圈！

## 引理 11.

BellmanFord 算法可以在  $O(nm)$  的时间内判断是否存在负圈，如果不存在负圈的话则计算出从顶点 1 到其余各个顶点的最短路径长度。

## ► 所有节点对的最短路径问题

# 所有节点对的最短路径问题

前面计算了单源的最短路径问题。那么如果我们想要计算所有节点对的最短路径问题呢？

	s	$v_1$	$\dots$	$v_n$
s	?	?	$\dots$	?
$v_1$	?	?	$\dots$	?
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$v_n$	?	?	$\dots$	?

Versus

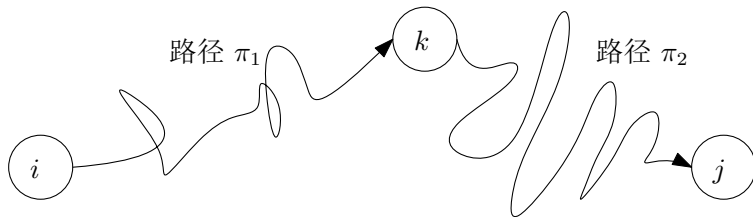
	s	$v_1$	$\dots$	$v_n$
s	?	?	$\dots$	?
$v_1$			$\dots$	
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$v_n$			$\dots$	

当然我们可以执行  $n$  次 BellmanFord 算法。这需要  $O(n^2m)$  的时间。

---

我们下面来介绍一个更快的算法-Floyd 算法。

- 考虑一个从  $i$  到  $j$  的只经过  $1 \sim k$  个顶点的最短路径，它一定是如下的形式：



- 其中  $\pi_1$  是  $i$  到  $k$  的只经过  $1 \sim k-1$  个顶点的最短路径.
- 其中  $\pi_2$  是  $k$  到  $j$  的只经过  $1 \sim k-1$  个顶点的最短路径.

从而令  $d_{i,j}^k$  表示从  $i$  到  $j$  的只经过  $1 \sim k$  个顶点的最短路径长度，则我们有：

$$d_{i,j}^k = \begin{cases} \omega(i, j) & \text{如果 } k = 0 \text{ 且 } (i, j) \in E \\ \infty & \text{如果 } k = 0 \text{ 且 } (i, j) \notin E \\ \min\{d_{i,j}^{k-1}, d_{i,k}^{k-1} + d_{k,j}^{k-1}\} & \text{如果 } k \geq 1 \end{cases}$$

---

根据该方程和  $k$  的值依次更新所有的  $d_{i,j}^k$ ，便可以得到一个求所有节点的最短路径的算法。

## 算法:Floyd

**输入:** 用  $n \times n$  矩阵表示的图  $I$ , 其中  $I_{ij}$  表示边  $(i,j)$  的权重, 不存在的边权重为  $\infty$ 。

**输出:**  $n \times n$  矩阵  $D$ , 其中  $D_{ij}$  表示从  $i$  到  $j$  的最短路径长度。

```
1:  $D \leftarrow I$ 
2: for  $k = 1$  to  $n$  do
3:   for  $i = 1$  to  $n$  do
4:     for  $j = 1$  to  $n$  do
5:        $D_{ij} \leftarrow \min\{D_{ij}, D_{ik} + D_{kj}\}$ 
```

---

显然这是一个 $O(n^3)$ 的算法, 比调用  $n$  次 BellmanFord 算法要快! (为什么?)



## 本节内容

- 单源最短路径
  - Dijkstra 算法
  - BellmanFord 算法
- 所有节点对最短路径
  - Floyd 算法