海量资源：666java.com

Bellman-Ford & Floyd-Warshall

海量资源：666java.com

这节课的内容

- Bellman-Ford 算法 - 单源最短路径 (含负权重)
- Floyd-Warshall 算法 - *所有结点对* 的最短路径 (含负权重)
- 动态规划

海量资源：666java.com

DIJKSTRA 伪代码

```
DIJKSTRA(G,s):
    Set all vertices to not-sure.
    d[v] = ∞ for all v in V
    d[s] = 0
    while there are not-sure nodes:
        pick the not-sure node u with smallest estimate d[u]
        for v in u.neighbors:
            d[v] ← min(d[v], d[u] + w(u,v))   // update step
        mark u as sure
    now all d[v] indeed equals the true distance d(s,v)
```

海量资源：666java.com

BELLMAN-FORD

单源最短路径 & 负权重边

海量资源：666java.com

DIJKSTRA vs BELLMAN-FORD

Dijkstra 算法

计算权重图中的单源最短路径问题

有时可以处理负权重的边, 有时不能

(你能举出几个例子吗?)

Bellman-Ford 算法

计算权重图中的单源最短路径问题

一定能够处理负权重的边

(如果不存在符合条件的路径则返回无解)

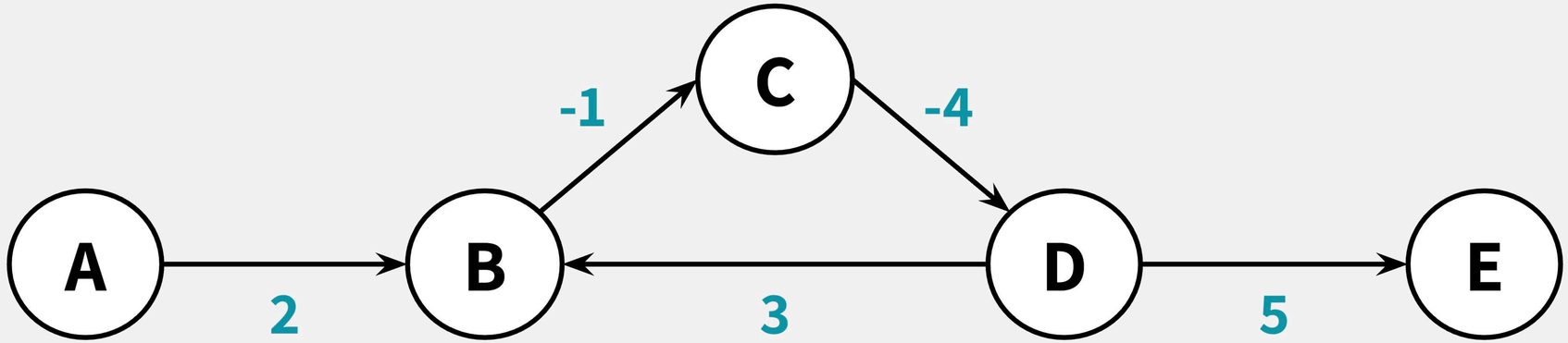
什么时候会需要 negative cycles?

通过某些边会产生 “奖励” (例如一条边代表返现交易)

海量资源：666java.com

NEGATIVE CYCLES

如果图中存在 negative cycles, 则最短路径问题无解. 为什么?



从 A 到 E 的 最短路径?

是: A → B → C → D → E? 代价 = 2 - 1 - 4 + 5 = **2**.

还是: A → B → C → D → B → C → D → B → C → D → B → C → D → B → C → D → E ?

存在 negative cycles 使得最短路径失去了意义

海量资源：666java.com

BELLMAN-FORD

基本思路

Dijkstra 算法中, 如果一个顶点有指向它的邻居中 **u** 的 **d[u]** 最小, 那么该顶点处的路径估值就可能需要更新. 因为通过 **u** 到达当前顶点的路径可能就是最短路径.

但在 Bellman-Ford 算法中, 由于需要考虑负权重的边,

不能单纯考虑拥有最小 d[u] 的顶点, 我们需要考虑所有*指向*当前顶点的邻居!

即使某邻居的路径代价估算很大, 负权重的边也可能大幅减小最终的路径代价, 所以我们无法判断哪些顶点可能位于最短路径上 — 需要考察所有顶点!

海量资源：666java.com

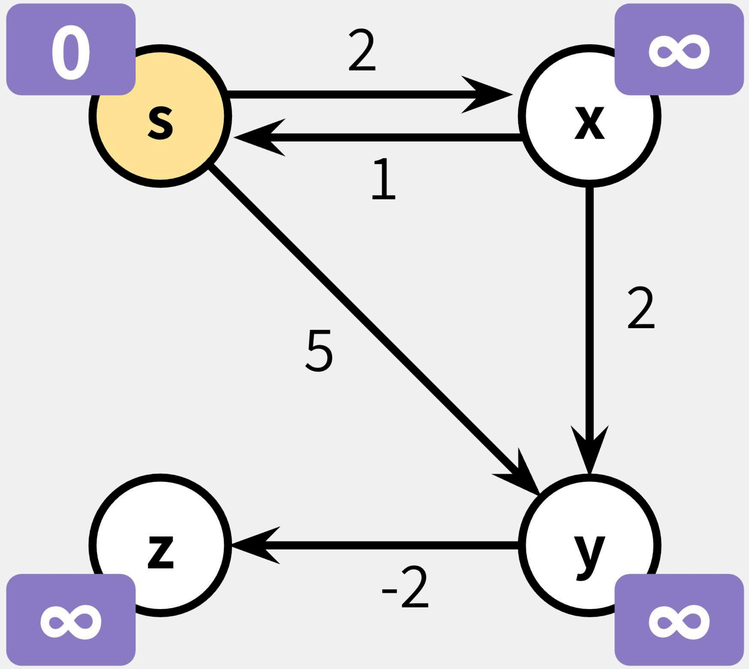
BELLMAN-FORD

维护一个长度为 n 的数组 $\mathbf{d^{(k)}}$, k = 0, 1, …, n–1.

$\mathbf{d^{(k)}[b]}$ = 从 s 到 b 经过*至多 k 条边*的最短路径代价.

显然, 我们知道如何填充 $\mathbf{d^{(0)}}$ -- 经过至多 **k = 0** 条边到达各个顶点的最短路径代价

	s	x	y	z
$\mathbf{d^{(0)}}$	0	∞	∞	∞
$\mathbf{d^{(1)}}$				
$\mathbf{d^{(2)}}$				
$\mathbf{d^{(3)}}$				



海量资源：666java.com

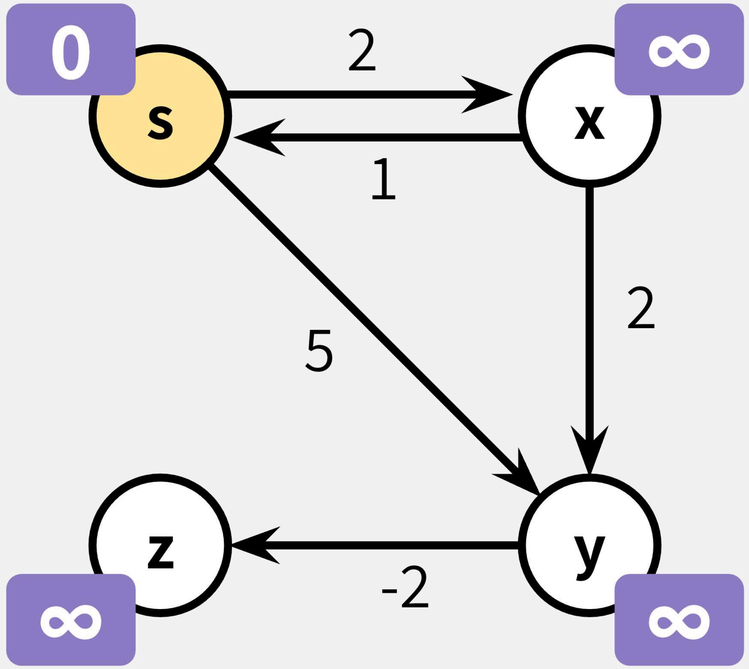
BELLMAN-FORD

维护一个长度为 n 的数组 $\mathbf{d^{(k)}}$, k = 0, 1, …, n–1.

$\mathbf{d^{(k)}[b]}$ = 从 s 到 b 经过*至多 k 条边*的最短路径代价.

通过 $\mathbf{d^{(0)}}$ 中的信息来填充 $\mathbf{d^{(1)}}$. 更一般的, 我们通过 $\mathbf{d^{(k-1)}}$ 中的信息来填充 $\mathbf{d^{(k)}}$.

	s	x	y	z
$\mathbf{d^{(0)}}$	0	∞	∞	∞
$\mathbf{d^{(1)}}$				
$\mathbf{d^{(2)}}$				
$\mathbf{d^{(3)}}$				



海量资源：666java.com

BELLMAN-FORD

维护一个长度为 n 的数组 $\mathbf{d^{(k)}}$, k = 0, 1, …, n–1.

$\mathbf{d^{(k)}[b]}$ = 从 s 到 b 经过*至多 k 条边*的最短路径代价.

如何通过 $\mathbf{d^{(0)}}$ 中的信息更新 $\mathbf{d^{(1)}[b]}$?

Case 1: 从 s 至 b 经过之多 k 条边的最短路径包含至多 k-1 条边! 也就是说, 允许经过 k 条边并没有改变最短路径. 则:

$$\mathbf{d^{(k)}[b] = d^{(k-1)}[b]}$$

Case 2: 从 s 至 b 经过之多 k 条边的最短路径恰好包含 k 条边! 这条长度 k 的最短路径为 [到达某个邻居 a 的长度为 k–1 的最短路径] + w(a,b). 那么 b 的哪个邻居可以提供这样的最短路径呢? 我们需要检查所有指向 b 的邻居:

$$\mathbf{d^{(k)}[b] = \min_{\text{a in b's incoming neighbors}}\{ d^{(k-1)}[a] + w(a,b) \}}$$

海量资源：666java.com

BELLMAN-FORD 伪代码

储存所有 n–1 行使得伪代码更好理解. 实际实现时, 我们只需要保存 2 行即可

检查所有邻居, 选取最小值
这一步耗时 O(deg(b))

```
BELLMAN_FORD(G,s):
    d^(k) = [] for k = 0, ..., n-1
    d^(0)[v] = ∞ for all v in V (except s)
    d^(0)[s] = 0
    for k = 1, ..., n-1:
        for b in V:
            d^(k)[b] ← min{ d^(k-1)[b], min_a{d^(k-1)[a] + w(a,b)} }
    return d^(n-1)
```

CASE 1　　CASE 2

运行时间: O(m·n)

慢于 Dijkstra 的 O(m + n log n)

海量资源：666java.com

BELLMAN-FORD

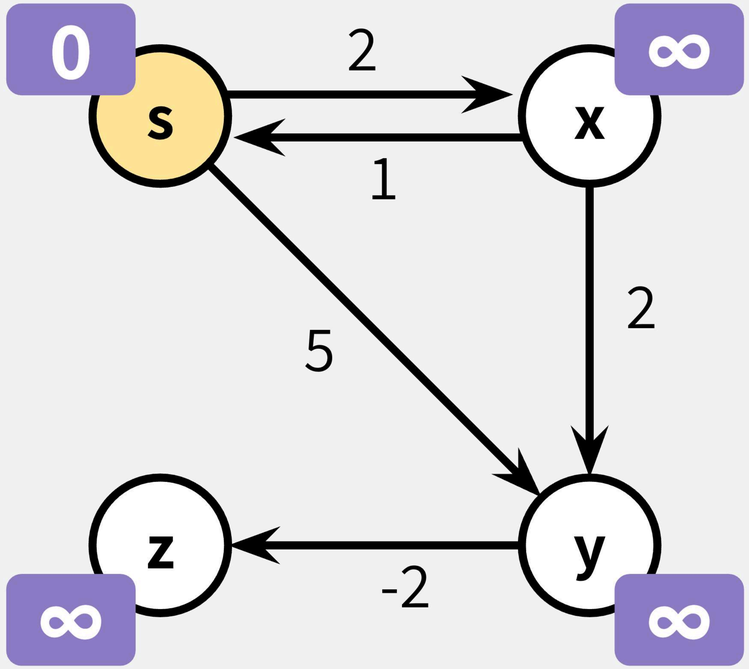
维护一个长度为 n 的数组 $\mathbf{d^{(k)}}$, k = 0, 1, …, n–1.

$\mathbf{d^{(k)}[b]}$ = 从 s 到 b 经过*至多 k 条边*的最短路径代价.

```
for k = 1, ..., n-1:
    for b in V:
        d(k)[b] ← min{d(k-1)[b], min_a{d(k-1)[a] + w(a,b)}}
```

通过 $\mathbf{d^{(0)}}$ 中的信息来填充 $\mathbf{d^{(1)}}$. 更一般的, 我们通过 $\mathbf{d^{(k-1)}}$ 中的信息来填充 $\mathbf{d^{(k)}}$.

	s	x	y	z
$\mathbf{d^{(0)}}$	0	∞	∞	∞
$\mathbf{d^{(1)}}$				
$\mathbf{d^{(2)}}$				
$\mathbf{d^{(3)}}$				



海量资源：666java.com

BELLMAN-FORD

维护一个长度为 n 的数组 $\mathbf{d^{(k)}}$, k = 0, 1, …, n–1.

$\mathbf{d^{(k)}[b]}$ = 从 s 到 b 经过*至多 k 条边*的最短路径代价.

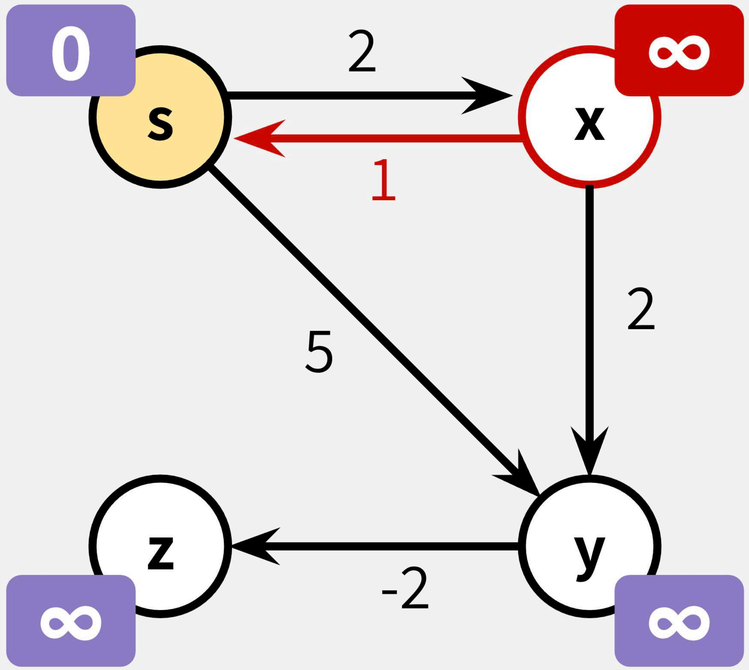
```
for k = 1, ..., n-1:
    for b in V:
        d(k)[b] ← min{d(k-1)[b], min_a{d(k-1)[a] + w(a,b)}}
```

	s	x	y	z
$\mathbf{d^{(0)}}$	0	∞	∞	∞
$\mathbf{d^{(1)}}$	0			
$\mathbf{d^{(2)}}$				
$\mathbf{d^{(3)}}$				

This is min of:

$d^{(0)}[s]$

$d^{(0)}[\mathbf{x}] + w(\mathbf{x}, s)$



海量资源：666java.com

BELLMAN-FORD

维护一个长度为 n 的数组 $\mathbf{d^{(k)}}$, k = 0, 1, …, n–1.

$\mathbf{d^{(k)}[b]}$ = 从 s 到 b 经过*至多 k 条边*的最短路径代价.

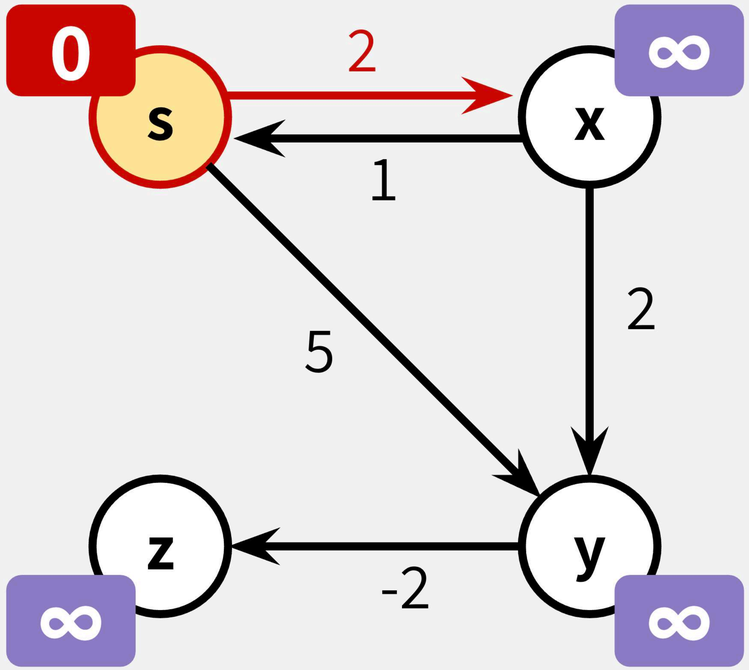
```
for k = 1, ..., n-1:
    for b in V:
        d(k)[b] ← min{d(k-1)[b], min_a{d(k-1)[a] + w(a,b)}}
```

	s	x	y	z
$\mathbf{d^{(0)}}$	0	∞	∞	∞
$\mathbf{d^{(1)}}$	0	2		
$\mathbf{d^{(2)}}$				
$\mathbf{d^{(3)}}$				

This is min of:

$d^{(0)}[x]$

$d^{(0)}[\mathbf{s}] + w(\mathbf{s}, x)$



海量资源：666java.com

BELLMAN-FORD

维护一个长度为 n 的数组 $\mathbf{d^{(k)}}$, k = 0, 1, …, n–1.

$\mathbf{d^{(k)}[b]}$ = 从 s 到 b 经过*至多 k 条边*的最短路径代价.

```
for k = 1, ..., n-1:
    for b in V:
        d(k)[b] ← min{d(k-1)[b], min_a{d(k-1)[a] + w(a,b)}}
```

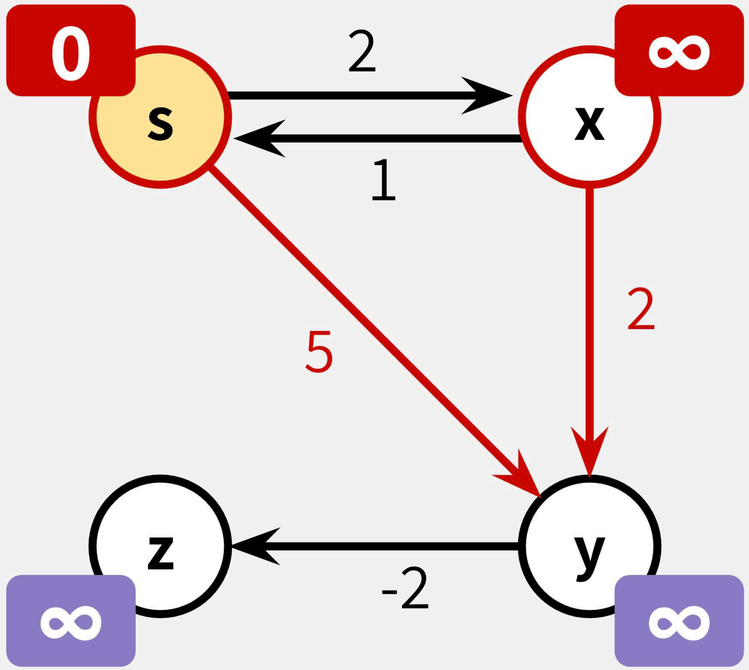
	s	x	y	z
$\mathbf{d^{(0)}}$	0	∞	∞	∞
$\mathbf{d^{(1)}}$	0	2	5	
$\mathbf{d^{(2)}}$				
$\mathbf{d^{(3)}}$				

This is min of:

$d^{(0)}[y]$

$d^{(0)}[\mathbf{s}] + w(\mathbf{s}, y)$

$d^{(0)}[\mathbf{x}] + w(\mathbf{x}, y)$



海量资源：666java.com

BELLMAN-FORD

维护一个长度为 n 的数组 $\mathbf{d^{(k)}}$, k = 0, 1, …, n–1.

$\mathbf{d^{(k)}[b]}$ = 从 s 到 b 经过*至多 k 条边*的最短路径代价.

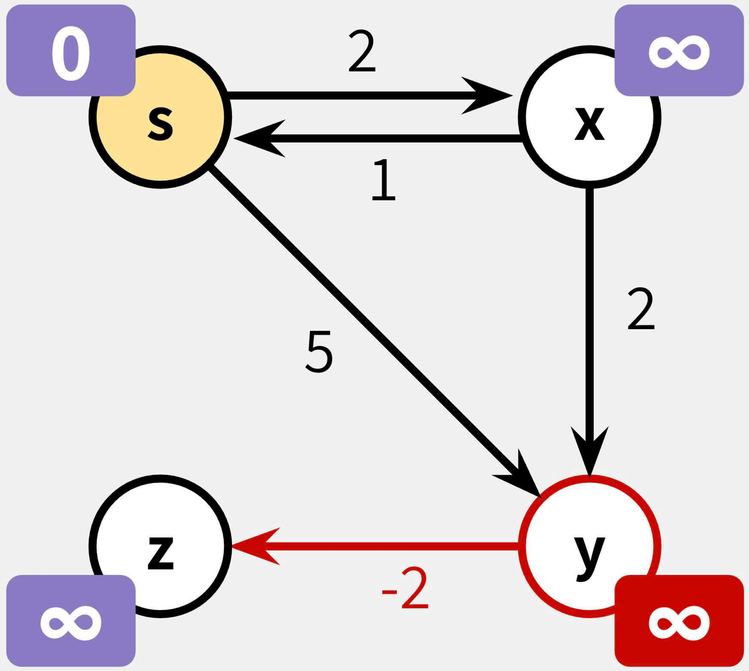
```
for k = 1, ..., n-1:
    for b in V:
        d(k)[b] ← min{d(k-1)[b], min_a{d(k-1)[a] + w(a,b)}}
```

	s	x	y	z
$\mathbf{d^{(0)}}$	0	∞	∞	∞
$\mathbf{d^{(1)}}$	0	2	5	∞
$\mathbf{d^{(2)}}$				
$\mathbf{d^{(3)}}$				

This is min of:

$d^{(0)}[z]$

$d^{(0)}[\mathbf{y}] + w(\mathbf{y}, z)$



海量资源：666java.com

BELLMAN-FORD

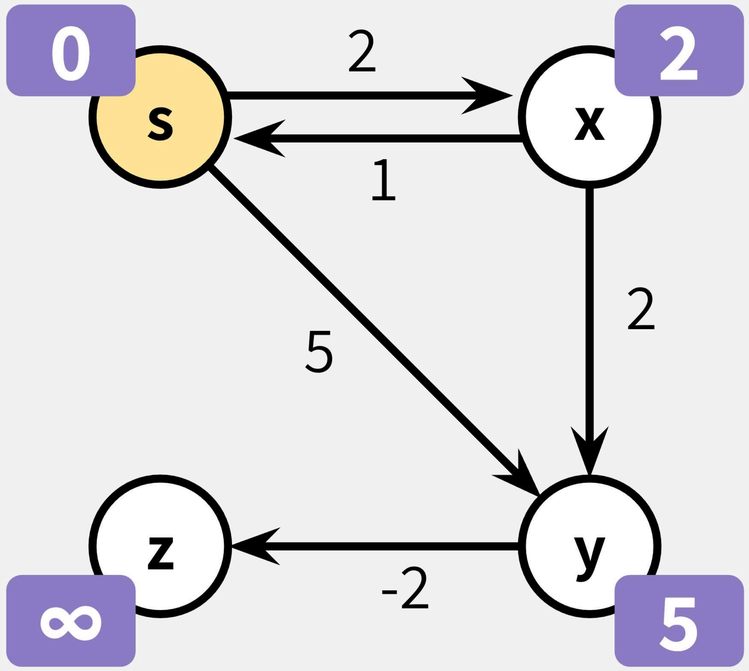
维护一个长度为 n 的数组 $\mathbf{d^{(k)}}$, k = 0, 1, …, n–1.

$\mathbf{d^{(k)}[b]}$ = 从 s 到 b 经过*至多 k 条边*的最短路径代价.

```
for k = 1, ..., n-1:
    for b in V:
        d(k)[b] ← min{d(k-1)[b], min_a{d(k-1)[a] + w(a,b)}}
```

$\mathbf{d^{(1)}}$ 完成

	s	x	y	z
$\mathbf{d^{(0)}}$	0	∞	∞	∞
$\mathbf{d^{(1)}}$	0	2	5	∞
$\mathbf{d^{(2)}}$				
$\mathbf{d^{(3)}}$				



海量资源：666java.com

BELLMAN-FORD

维护一个长度为 n 的数组 $\mathbf{d^{(k)}}$, k = 0, 1, …, n–1.

$\mathbf{d^{(k)}[b]}$ = 从 s 到 b 经过*至多 k 条边*的最短路径代价.

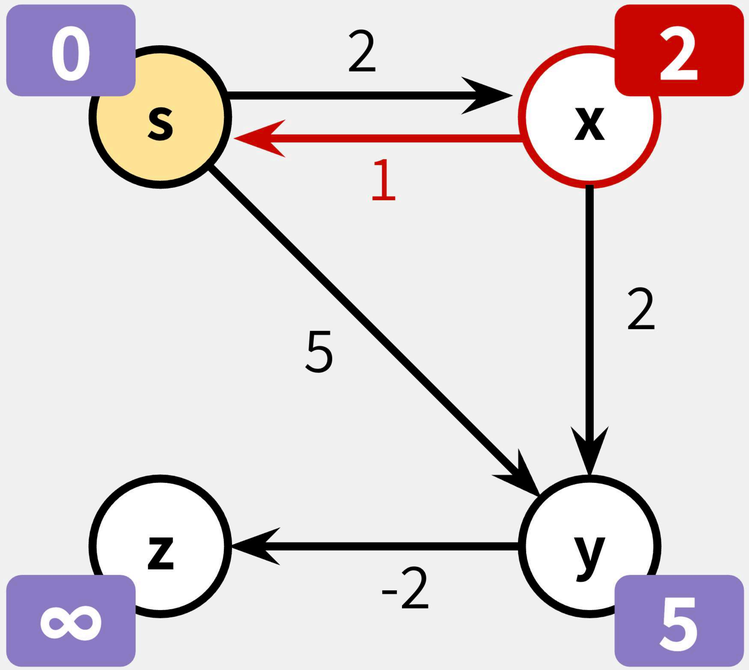
```
for k = 1, ..., n-1:
    for b in V:
        d(k)[b] ← min{d(k-1)[b], min_a{d(k-1)[a] + w(a,b)}}
```

	s	x	y	z
$\mathbf{d^{(0)}}$	0	∞	∞	∞
$\mathbf{d^{(1)}}$	0	2	5	∞
$\mathbf{d^{(2)}}$	0			
$\mathbf{d^{(3)}}$				

This is min of:

$d^{(1)}[s]$

$d^{(1)}[\mathbf{x}] + w(\mathbf{x}, s)$



海量资源：666java.com

BELLMAN-FORD

维护一个长度为 n 的数组 $\mathbf{d^{(k)}}$, k = 0, 1, …, n–1.

$\mathbf{d^{(k)}[b]}$ = 从 s 到 b 经过*至多 k 条边*的最短路径代价.

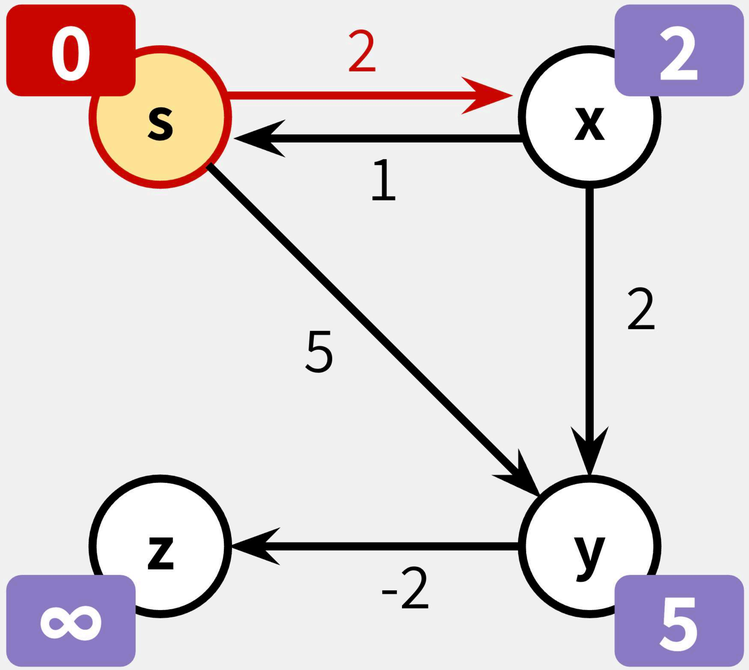
```
for k = 1, ..., n-1:
    for b in V:
        d(k)[b] ← min{d(k-1)[b], min_a{d(k-1)[a] + w(a,b)}}
```

This is min of:

$d^{(1)}[x]$

$d^{(1)}[\mathbf{s}] + w(\mathbf{s}, x)$

	s	x	y	z
$\mathbf{d^{(0)}}$	0	∞	∞	∞
$\mathbf{d^{(1)}}$	0	2	5	∞
$\mathbf{d^{(2)}}$	0	2		
$\mathbf{d^{(3)}}$				



海量资源：666java.com

BELLMAN-FORD

维护一个长度为 n 的数组 $\mathbf{d^{(k)}}$, k = 0, 1, …, n–1.

$\mathbf{d^{(k)}[b]}$ = 从 s 到 b 经过*至多 k 条边*的最短路径代价.

```
for k = 1, ..., n-1:
    for b in V:
        d(k)[b] ← min{d(k-1)[b], min_a{d(k-1)[a] + w(a,b)}}
```

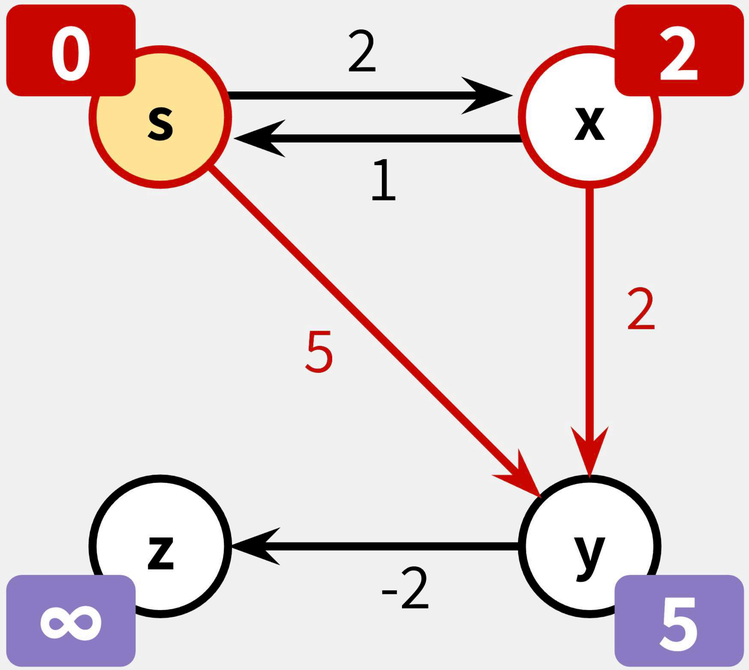
	s	x	y	z
$\mathbf{d^{(0)}}$	0	∞	∞	∞
$\mathbf{d^{(1)}}$	0	2	5	∞
$\mathbf{d^{(2)}}$	0	2	4	
$\mathbf{d^{(3)}}$				

This is min of:

$d^{(1)}[y]$

$d^{(1)}[\mathbf{s}] + w(\mathbf{s}, y)$

$d^{(1)}[\mathbf{x}] + w(\mathbf{x}, y)$



海量资源：666java.com

BELLMAN-FORD

维护一个长度为 n 的数组 $\mathbf{d^{(k)}}$, k = 0, 1, …, n–1.

$\mathbf{d^{(k)}[b]}$ = 从 s 到 b 经过*至多 k 条边*的最短路径代价.

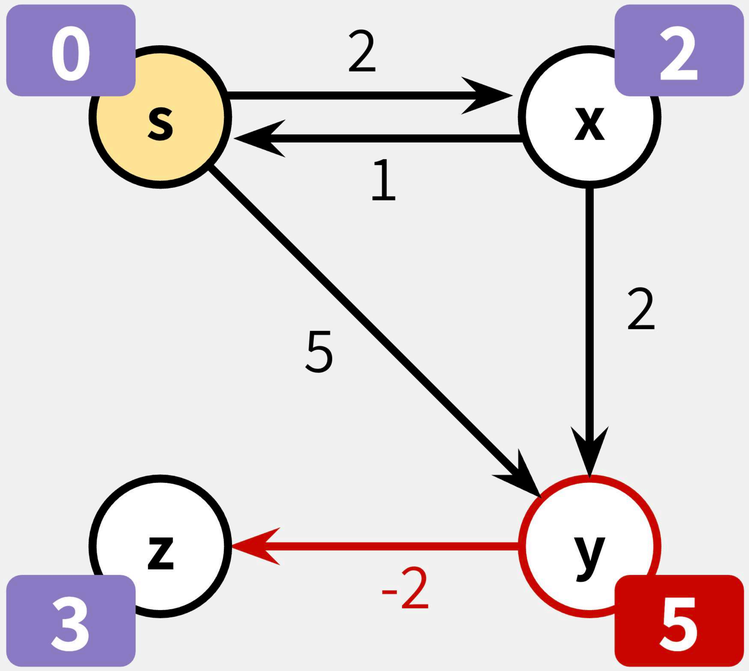
```
for k = 1, ..., n-1:
   for b in V:
      d(k)[b] ← min{d(k-1)[b], min_a{d(k-1)[a] + w(a,b)}}
```

This is min of:

$d^{(1)}[z]$

$d^{(1)}[\mathbf{y}] + w(\mathbf{y}, z)$

	s	x	y	z
$\mathbf{d^{(0)}}$	0	∞	∞	∞
$\mathbf{d^{(1)}}$	0	2	5	∞
$\mathbf{d^{(2)}}$	0	2	4	3
$\mathbf{d^{(3)}}$				



海量资源：666java.com

BELLMAN-FORD

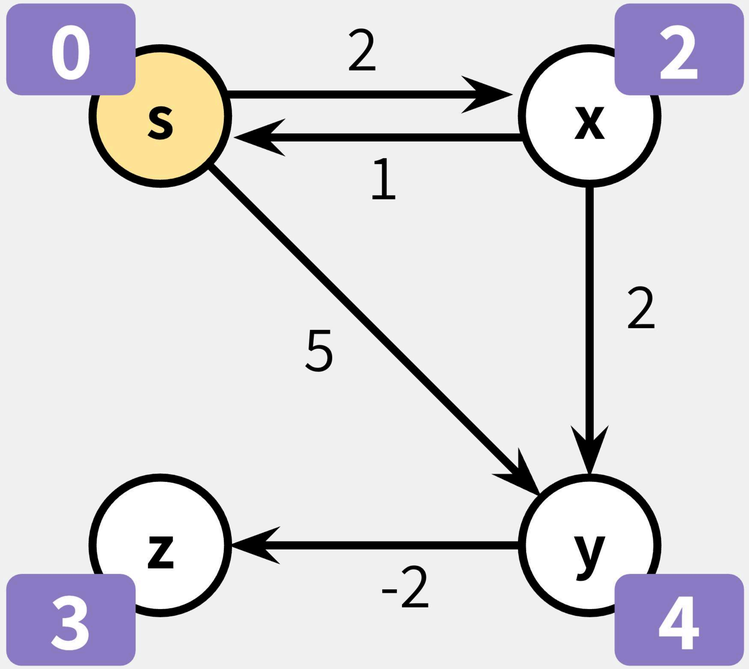
维护一个长度为 n 的数组 $\mathbf{d^{(k)}}$, k = 0, 1, …, n–1.

$\mathbf{d^{(k)}[b]}$ = 从 s 到 b 经过*至多 k 条边*的最短路径代价.

```
for k = 1, ..., n-1:
    for b in V:
        d(k)[b] ← min{d(k-1)[b], min_a{d(k-1)[a] + w(a,b)}}
```

$\mathbf{d^{(2)}}$ 完成

	s	x	y	z
$\mathbf{d^{(0)}}$	0	∞	∞	∞
$\mathbf{d^{(1)}}$	0	2	5	∞
$\mathbf{d^{(2)}}$	0	2	4	3
$\mathbf{d^{(3)}}$				



海量资源：666java.com

BELLMAN-FORD

维护一个长度为 n 的数组 $\mathbf{d^{(k)}}$, k = 0, 1, …, n–1.

$\mathbf{d^{(k)}[b]}$ = 从 s 到 b 经过*至多 k 条边*的最短路径代价.

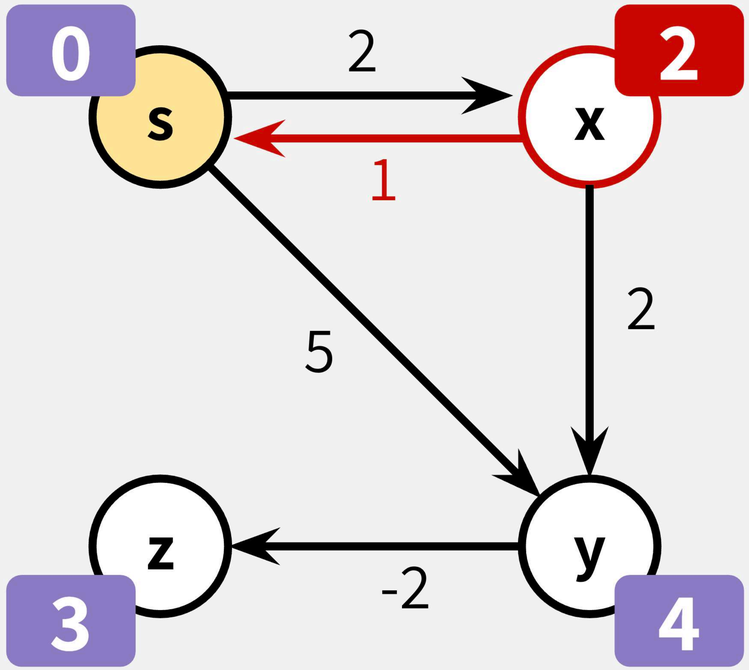
```
for k = 1, ..., n-1:
    for b in V:
        d(k)[b] ← min{d(k-1)[b], min_a{d(k-1)[a] + w(a,b)}}
```

	s	x	y	z
$d^{(0)}$	0	∞	∞	∞
$d^{(1)}$	0	2	5	∞
$d^{(2)}$	0	2	4	3
$d^{(3)}$	0			

This is min of:

$d^{(2)}[s]$

$d^{(2)}[\mathbf{x}] + w(\mathbf{x}, s)$



海量资源：666java.com

BELLMAN-FORD

维护一个长度为 n 的数组 $\mathbf{d^{(k)}}$, k = 0, 1, …, n–1.

$\mathbf{d^{(k)}[b]}$ = 从 s 到 b 经过*至多 k 条边*的最短路径代价.

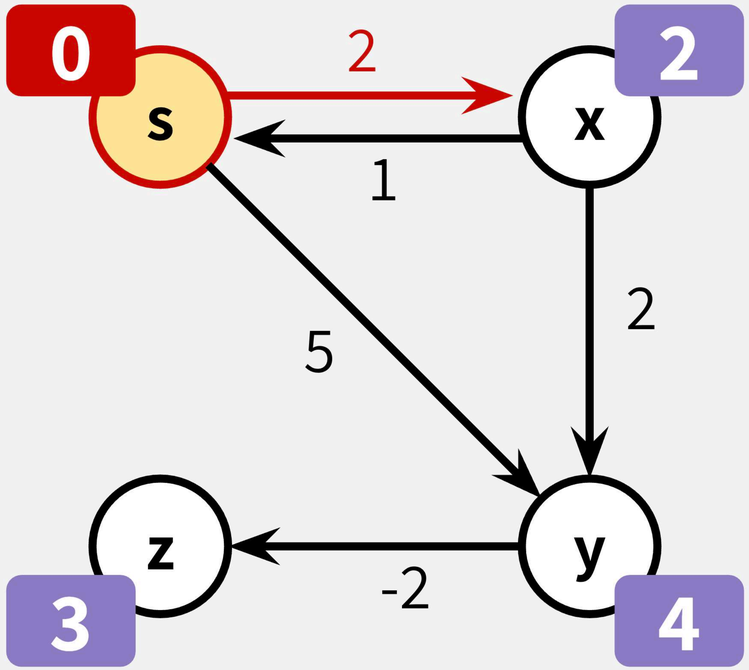
```
for k = 1, ..., n-1:
    for b in V:
        d(k)[b] ← min{d(k-1)[b], min_a{d(k-1)[a] + w(a,b)}}
```

	s	x	y	z
$d^{(0)}$	0	∞	∞	∞
$d^{(1)}$	0	2	5	∞
$d^{(2)}$	0	2	4	3
$d^{(3)}$	0	2		

This is min of:

$d^{(2)}[x]$

$d^{(2)}[\mathbf{s}] + w(\mathbf{s}, x)$



海量资源：666java.com

BELLMAN-FORD

维护一个长度为 n 的数组 $\mathbf{d^{(k)}}$, k = 0, 1, …, n–1.

$\mathbf{d^{(k)}[b]}$ = 从 s 到 b 经过*至多 k 条边*的最短路径代价.

```
for k = 1, ..., n-1:
    for b in V:
        d(k)[b] ← min{d(k-1)[b], min_a{d(k-1)[a] + w(a,b)}}
```

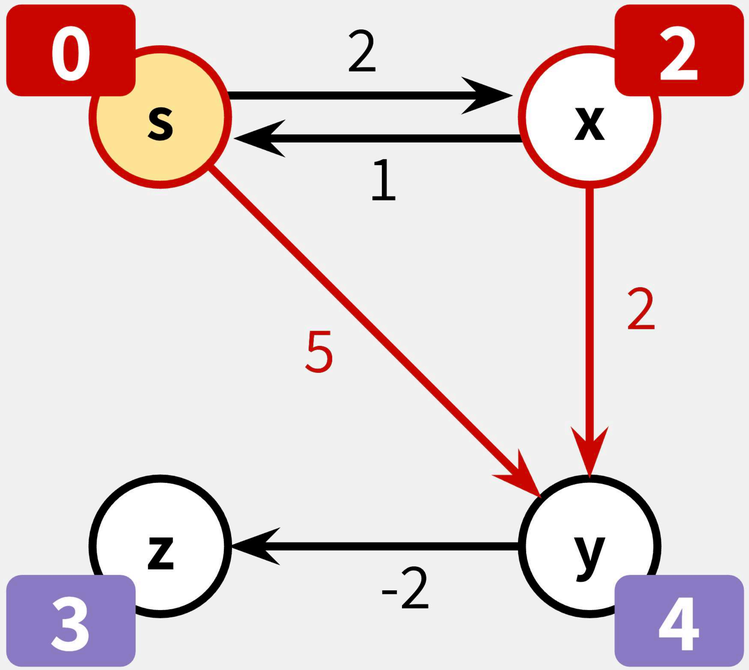
	s	x	y	z
$\mathbf{d^{(0)}}$	0	∞	∞	∞
$\mathbf{d^{(1)}}$	0	2	5	∞
$\mathbf{d^{(2)}}$	0	2	4	3
$\mathbf{d^{(3)}}$	0	2	4	

This is min of:

$d^{(2)}[y]$

$d^{(2)}[\mathbf{s}] + w(\mathbf{s}, y)$

$d^{(2)}[\mathbf{x}] + w(\mathbf{x}, y)$



海量资源：666java.com

BELLMAN-FORD

维护一个长度为 n 的数组 $\mathbf{d^{(k)}}$, k = 0, 1, …, n–1.

$\mathbf{d^{(k)}[b]}$ = 从 s 到 b 经过*至多 k 条边*的最短路径代价.

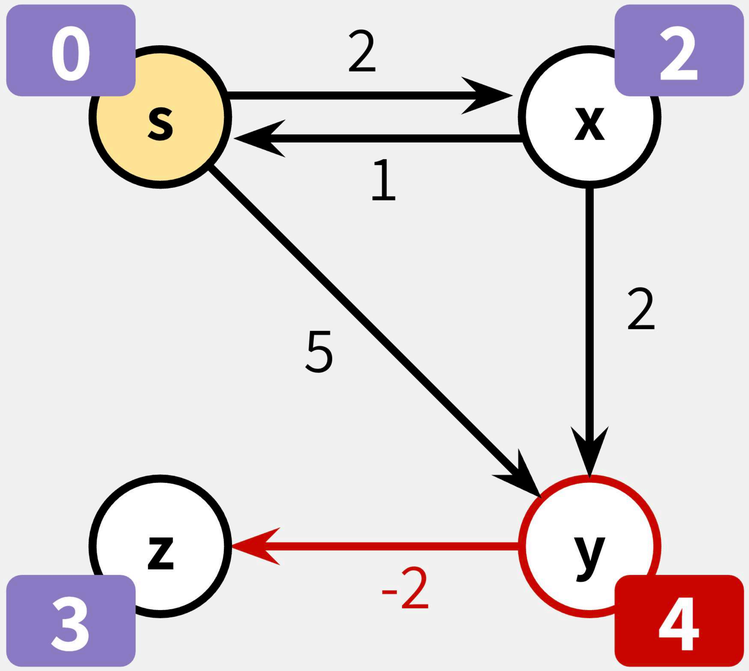
```
for k = 1, ..., n-1:
    for b in V:
        d(k)[b] ← min{d(k-1)[b], min_a{d(k-1)[a] + w(a,b)}}
```

This is min of:

$d^{(2)}[z]$

$d^{(2)}[\mathbf{y}] + w(\mathbf{y}, z)$

	s	x	y	z
$\mathbf{d^{(0)}}$	0	∞	∞	∞
$\mathbf{d^{(1)}}$	0	2	5	∞
$\mathbf{d^{(2)}}$	0	2	4	3
$\mathbf{d^{(3)}}$	0	2	4	2



海量资源：666java.com

BELLMAN-FORD

维护一个长度为 n 的数组 $\mathbf{d^{(k)}}$, k = 0, 1, …, n–1.

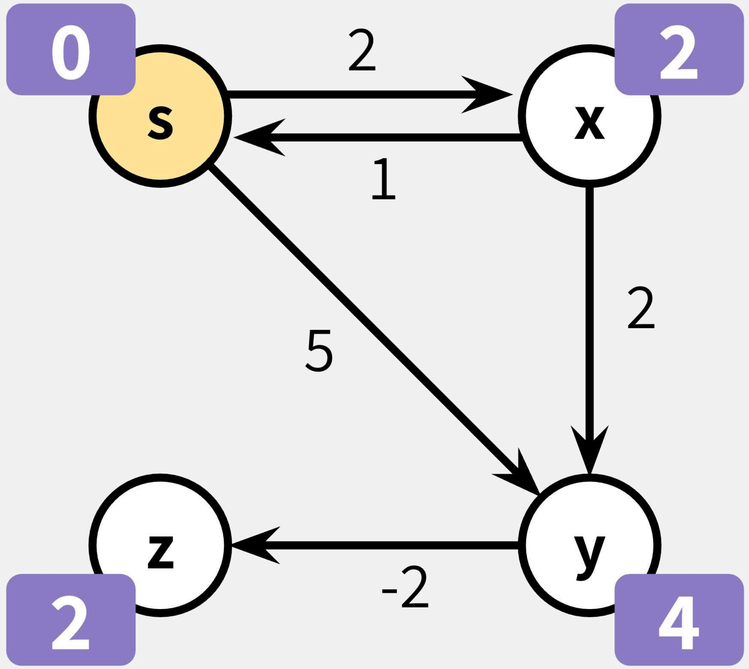
$\mathbf{d^{(k)}[b]}$ = 从 s 到 b 经过*至多 k 条边*的最短路径代价.

```
for k = 1, ..., n-1:
    for b in V:
        d(k)[b] ← min{d(k-1)[b], min_a{d(k-1)[a] + w(a,b)}}
```

$d^{(3)}$ 完成!

检查顶点 **z** 的每个 $d^{(i)}$:

	s	x	y	z
$d^{(0)}$	0	∞	∞	∞
$d^{(1)}$	0	2	5	∞
$d^{(2)}$	0	2	4	3
$d^{(3)}$	0	2	4	2



海量资源：666java.com

BELLMAN-FORD

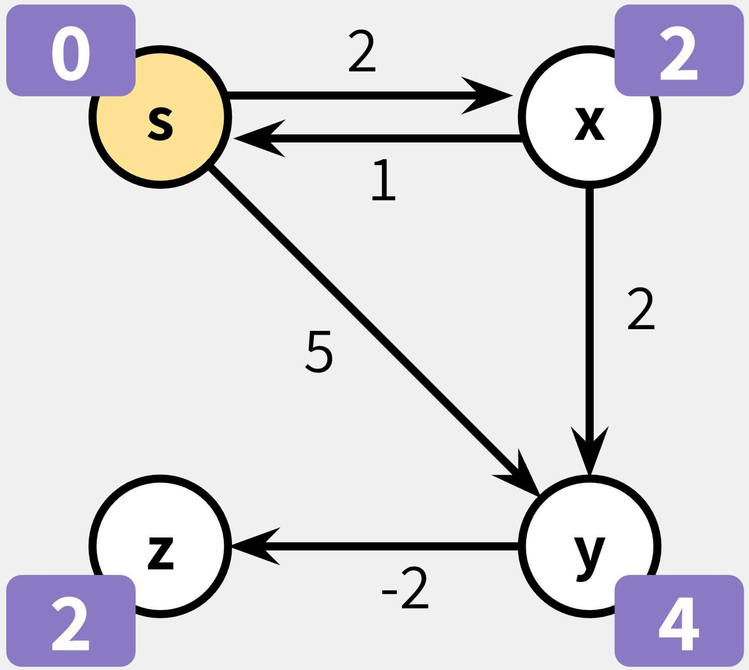
维护一个长度为 n 的数组 $\mathbf{d^{(k)}}$, k = 0, 1, …, n–1.

$\mathbf{d^{(k)}[b]}$ = 从 s 到 b 经过*至多 k 条边*的最短路径代价.

```
for k = 1, ..., n-1:
    for b in V:
        d(k)[b] ← min{d(k-1)[b], min_a{d(k-1)[a] + w(a,b)}}
```

从 s 到 z 经过 **1** 条边的最短路径代价 = **∞**

	s	x	y	z
$\mathbf{d^{(0)}}$	0	∞	∞	∞
$\mathbf{d^{(1)}}$	0	2	5	∞
$\mathbf{d^{(2)}}$	0	2	4	3
$\mathbf{d^{(3)}}$	0	2	4	2



海量资源：666java.com

BELLMAN-FORD

维护一个长度为 n 的数组 $\mathbf{d^{(k)}}$, k = 0, 1, …, n–1.

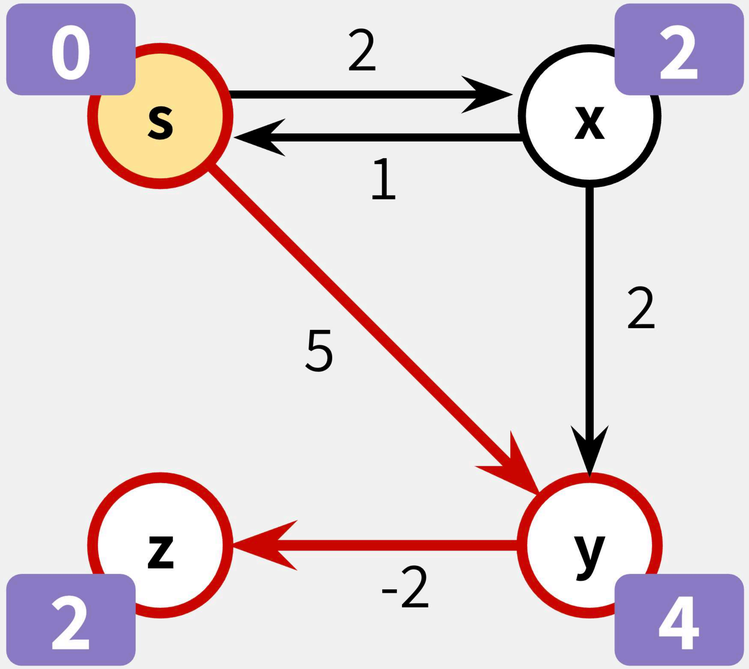
$\mathbf{d^{(k)}[b]}$ = 从 s 到 b 经过*至多 k 条边*的最短路径代价.

```
for k = 1, ..., n-1:
   for b in V:
      d(k)[b] ← min{d(k-1)[b], min_a{d(k-1)[a] + w(a,b)}}
```

从 s 到 z 经过 **1** 条边的最短路径代价 = **∞**

从 s 到 z 经过 **2** 条边的最短路径代价 = **3**

	s	x	y	z
$d^{(0)}$	0	∞	∞	∞
$d^{(1)}$	0	2	5	∞
$d^{(2)}$	0	2	4	3
$d^{(3)}$	0	2	4	2



海量资源：666java.com

BELLMAN-FORD

维护一个长度为 n 的数组 $\mathbf{d^{(k)}}$, k = 0, 1, …, n–1.

$\mathbf{d^{(k)}[b]}$ = 从 s 到 b 经过*至多 k 条边*的最短路径代价.

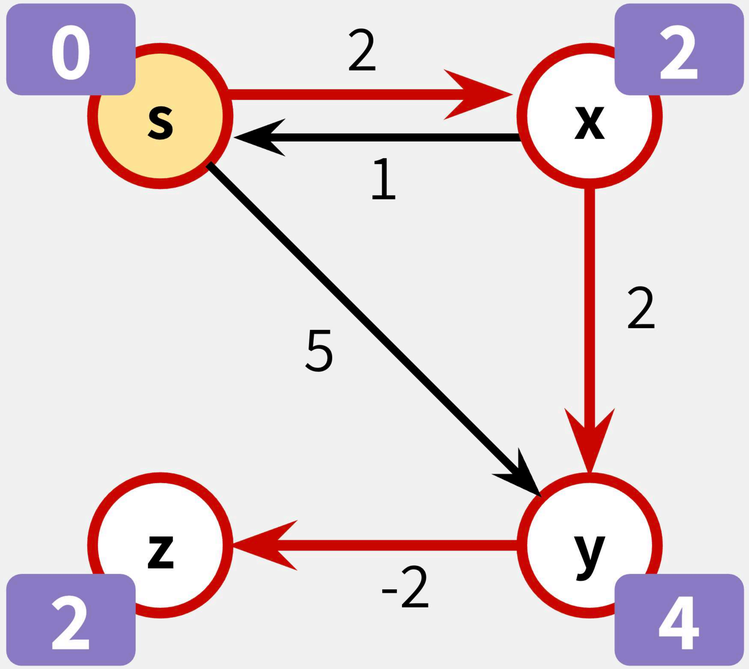
```
for k = 1, ..., n-1:
    for b in V:
        d(k)[b] ← min{d(k-1)[b], min_a{d(k-1)[a] + w(a,b)}}
```

	s	x	y	z
$\mathbf{d^{(0)}}$	0	∞	∞	∞
$\mathbf{d^{(1)}}$	0	2	5	∞
$\mathbf{d^{(2)}}$	0	2	4	3
$\mathbf{d^{(3)}}$	0	2	4	2

从 s 到 z 经过 **1** 条边的最短路径代价 = **∞**

从 s 到 z 经过 **2** 条边的最短路径代价 = **3**

从 s 到 z 经过 **3** 条边的最短路径代价 = **2**



海量资源：666java.com

BELLMAN-FORD 正确性

定理: 只要图中不存在 negative cycles, 则 Bellman-ford 正确.

证明: 通过证明以下两点来证明:

- **1:** $d^{(n-1)}[v]$ 是从 s 至 v 经过至多 n-1 条边的最短路径代价.
- **2:** 若存在 s 至 v 的最短路径, 则经过至多 n-1 条边的最短路径就是 s 至 v 的最短路径(允许经过任意个数的顶点).

1 & **2** 表明 $\mathbf{d^{(n-1)}[v]}$ 即 s 至 v 的最短路径代价

海量资源：666java.com

BELLMAN-FORD 正确性

1: $d^{(n-1)}[v]$ 是从 s 至 v 经过至多 n-1 条边的最短路径代价.

证明: 对 k 进行归纳, k 为算法迭代次数.

- **归纳假设**: k 次迭代后, $d^{(k)}[v]$ = s 至 v 经过至多 k 条边的最短路径代价.
- **基本情况**: 0 次迭代后, 从 s 至 s 经过 0 条边的最短路径含有 0 条边 (代价为 0), 所以 $d^{(0)}[s] = 0$! 由于从 s 至其他顶点的最短路径需要经过至少 1 条边, 所以不存在 s 至 v 经过 0 条边的最短路径, 所以 $d^{(0)}[v]$ 初始化为 ∞. 所以, $d^{(0)}$ 正确.
- **归纳**: 假设第 k 次迭代开始时, $d^{(k-1)}[b]$ 是由 s 至 b 经过至多 k-1 条边的最短路径代价. 考虑两种情况:
  - **Case 1: $d^{(k-1)}[b] < \min_a\{d^{(k-1)}[a] + w(a, b)\}$**. 此时最短路径包含少于 k 条边. 更新 $d^{(k)}[b] = d^{(k-1)}[b]$.
  - **Case 2: $d^{(k-1)}[b] \geq \min_a\{d^{(k-1)}[a] + w(a, b)\}$**. 此时最短路径恰好包含 k 条边. 更新 $d^{(k)}[b] = \min_a\{d^{(k-1)}[a] + w(a, b)\}$, 大小为经过至多 k-1 条边到达邻居 a 的最短路径代价与 a 至 b 的路径代价和的最小值.
- **结论**: 第 n–1 次迭代完成后, $d^{(n-1)}[v]$ 为 s 至 v 经过至多 n-1 条边的最短路径代价.

海量资源：666java.com

BELLMAN-FORD 正确性

2: 若存在 s 至 v 的最短路径, 则经过至多 n-1 条边的最短路径就是 s 至 v 的最短路径(允许经过任意个数的顶点).

证明: 直接证明.

- 假设图中没有 negative cycles, 则 s 至 v 若存在最短路径, 该最短路径要么是一条简单路径(无环), 要么包含 positive cycles.
- 最短路径不能包含 positive cycles, 因为可以通过剔除环上的边而得到一个更短的路径. (代价为 0 的环也可以被剔除 & 不会影响路径的代价).
- 而简单路径包含至多 n–1 条边, 由此我们知道若存在 s 至 v 的最短路径, 则经过至多 n-1 条边的最短路径就是 s 至 v 的最短路径(允许经过任意个数的顶点).

海量资源：666java.com

BELLMAN-FORD 正确性

定理: 只要图中不存在 negative cycles, 则 Bellman-ford 正确.

证明: 通过证明以下两点来证明:

- **1:** $d^{(n-1)}[v]$ 是从 s 至 v 经过至多 n-1 条边的最短路径代价.
- **2:** 若存在 s 至 v 的最短路径, 则经过至多 n-1 条边的最短路径就是 s 至 v 的最短路径.

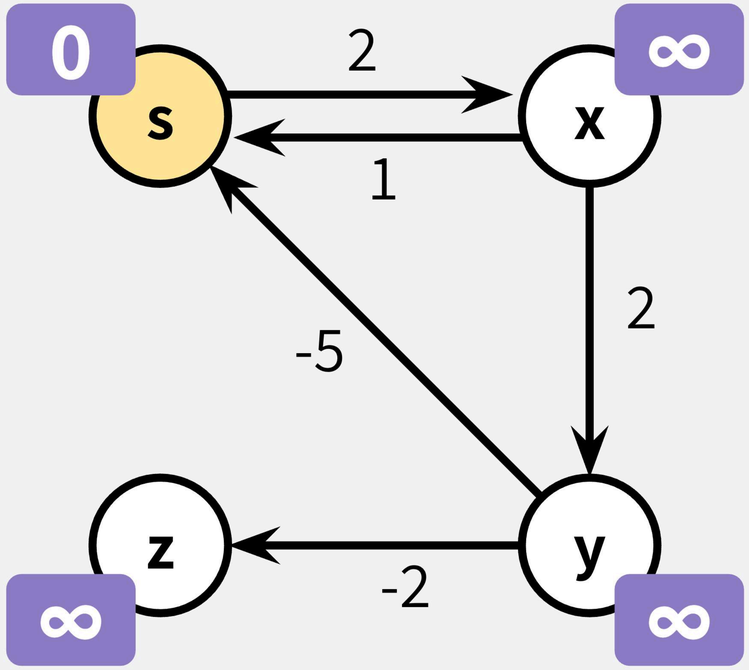
证明完成

海量资源：666java.com

存在 NEGATIVE CYCLES 时会发生什么?

考虑含有 negative cycle 的图. 填充 d 表:

	s	x	y	z
$d^{(0)}$	0	∞	∞	∞
$d^{(1)}$				
$d^{(2)}$				
$d^{(3)}$				

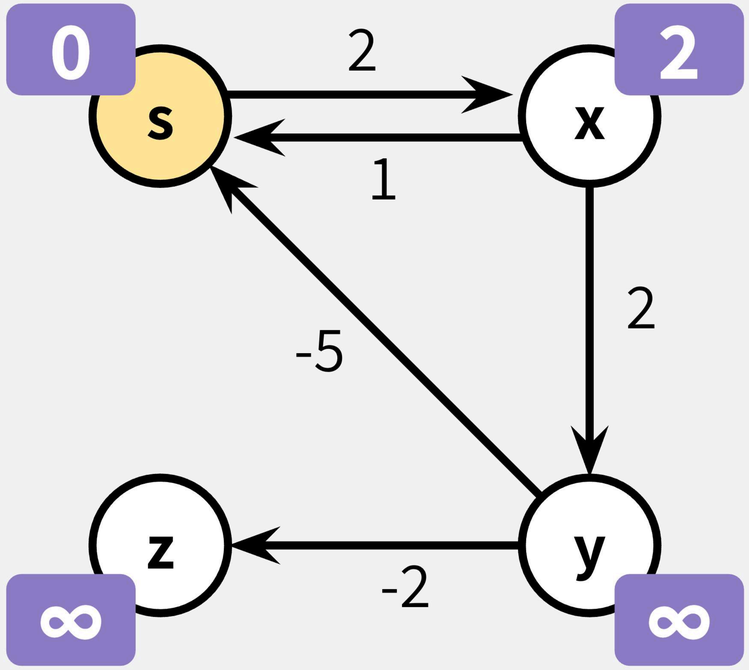


海量资源：666java.com

存在 NEGATIVE CYCLES 时会发生什么?

考虑含有 negative cycle 的图. 填充 d 表:

	s	x	y	z
$d^{(0)}$	0	∞	∞	∞
$d^{(1)}$	0	2	∞	∞
$d^{(2)}$				
$d^{(3)}$				

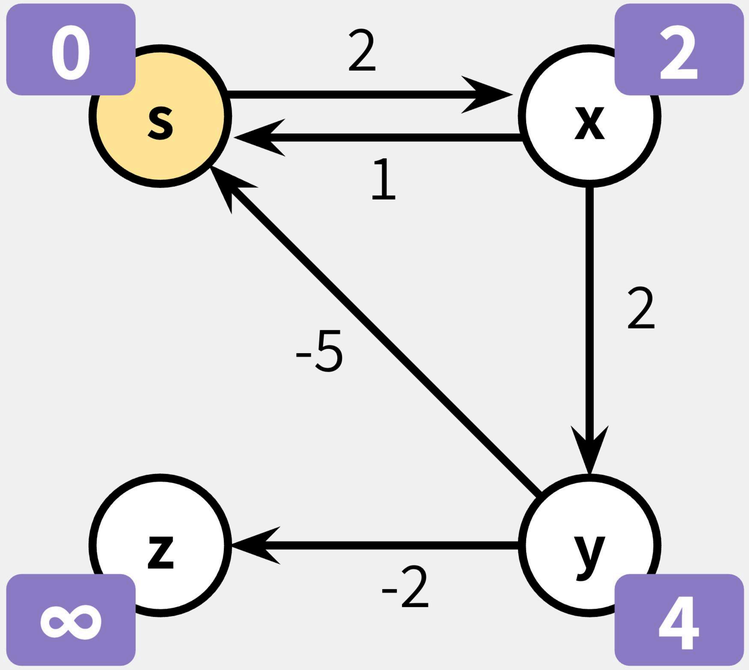


海量资源：666java.com

存在 NEGATIVE CYCLES 时会发生什么?

考虑含有 negative cycle 的图. 填充 d 表:

	s	x	y	z
$d^{(0)}$	0	∞	∞	∞
$d^{(1)}$	0	2	∞	∞
$d^{(2)}$	0	2	4	∞
$d^{(3)}$				



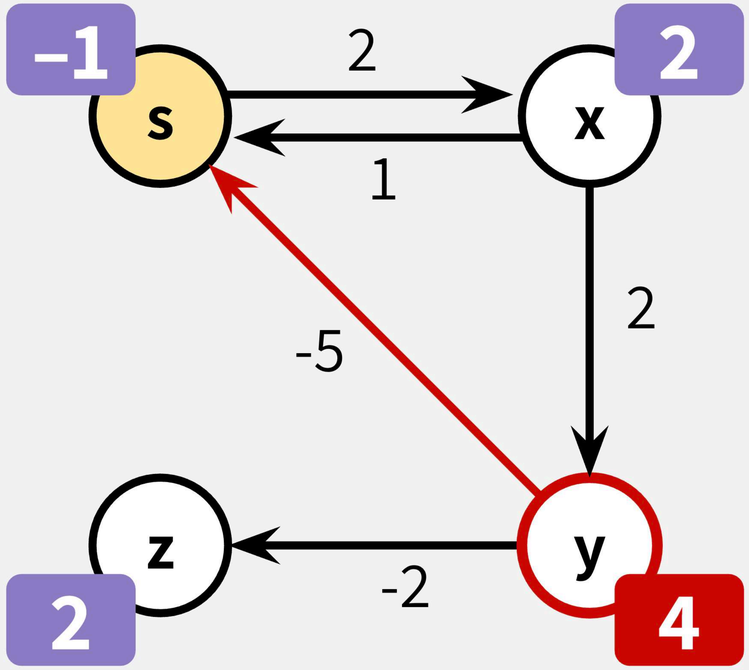
海量资源：666java.com

存在 NEGATIVE CYCLES 时会发生什么?

考虑含有 negative cycle 的图. 填充 d 表:

	s	x	y	z
$d^{(0)}$	0	∞	∞	∞
$d^{(1)}$	0	2	∞	∞
$d^{(2)}$	0	2	4	∞
$d^{(3)}$	-1	2	4	2

出现问题! 我们可以绕着环而得到一个更小的代价...



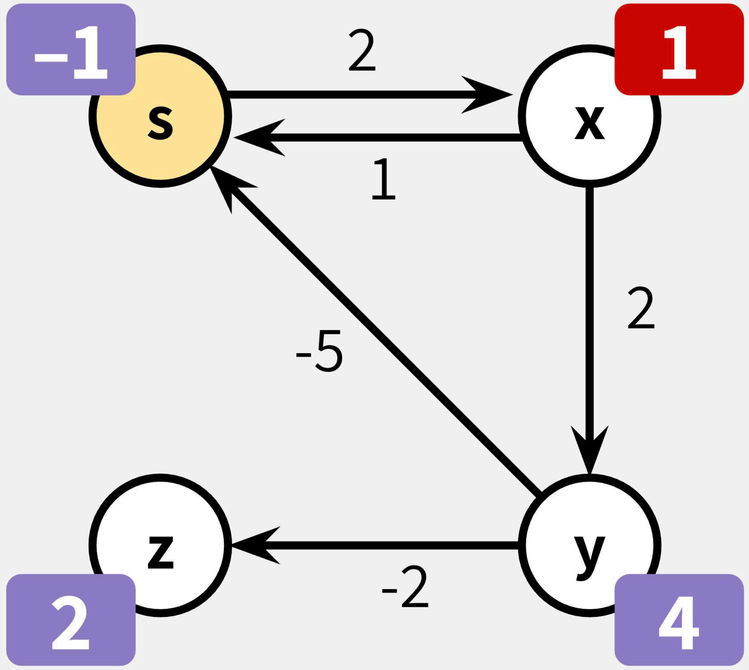
海量资源：666java.com

存在 NEGATIVE CYCLES 时会发生什么?

考虑含有 negative cycle 的图. 填充 d 表:

	s	x	y	z
$d^{(0)}$	0	∞	∞	∞
$d^{(1)}$	0	2	∞	∞
$d^{(2)}$	0	2	4	∞
$d^{(3)}$	-1	2	4	2
$d^{(4)}$	-1	1	4	2

x 的代价进一步减小!



海量资源：666java.com

存在 NEGATIVE CYCLES 时会发生什么?

考虑含有 negative cycle 的图. 填充 d 表:

在 Bellman-Ford 过程中我们可以通过一次额外的迭代来检测 negative cycles.

如果最短路径的代价发生了变化,
则表明图中存在 negative cycle!

这表明对某个顶点 v, 存在一条经过 n 条边的路径其代价小于经过至多 n-1 条边的最短路径的代价. 那么这条路径中一定包含环 (否则长度不会大于 n–1), 而且由于代价更小, 说明环的代价为负值 (如果不为负, 则路径的代价不会减小)

海量资源：666java.com

BELLMAN-FORD (+ CYCLE DETECTION)

```
BELLMAN_FORD(G,s):
    d^(k) = [] for k = 0, ..., n-1
    d^(0)[v] = ∞ for all v in V (except s)
    d^(0)[s] = 0
    for k = 1, ..., n:
        for b in V:
            d^(k)[b] ← min{ d^(k-1)[b], min_a{d^(k-1)[a] + w(a,b)} }
    if d^(n-1) != d^(n):
        return “NEGATIVE CYCLE!”
    return d^(n-1)
```

海量资源：666java.com

BELLMAN-FORD 总结

寻找包含负权重边的图中的最短路径. 运行时间 O(mn).

如果图中不包含 negative cycles: BF 得到的最短路径为 $d^{(n-1)}[v] = d(s,v)$

如果图中包含 negative cycles: BF 返回 **`negative cycle`**

Bellman-Ford 的一个优点是非常适用于分布式计算. 每个结点处的更新只依赖于上一次迭代的代价估计和邻居结点出发的边权重. 这使得算法非常适用于求解网络上的问题! 每个结点处不需要保存整个网络的信息, 而只需要当前结点的信息即可.
(e.g. Routing Information Protocol, or RIP)

海量资源：666java.com

动态规划

另一种算法设计范例!

海量资源：666java.com

动态规划

Bellman-Ford 是一种 **动态规划** 算法!

动态规划 Dynamic programming (DP) 是另一种算法设计范例.
常用于求解最优化问题 (e.g. *最短*路径).

稍后会介绍另一种 DP 算法 (Floyd-Warshall), 下节课将深入介绍更多动态规划问题.

简要介绍一下动态规划的思路

海量资源：666java.com

动态规划

动态规划的要点:

最优子结构: 一个问题的最优解包含其子问题的最优解.

e.g. $\mathbf{d^{(k)}[b] = \min\{ d^{(k-1)}[b], \min_a\{d^{(k-1)}[a] + w(a, b)\} \}}$

重叠子问题: 子问题大量重叠!

可以通过缓存已求解子问题的解来减少算法运行时间.

(也被称为 “记忆化”)

e.g. **$d^{(k)}$ 中很多项的计算都需要知道 $d^{(k-1)}[v]$**

海量资源：666java.com

动态规划

2 种 设计/实现 动态规划算法的方式

Bottom-up: 按照问题大小进行迭代, 先求解小的问题 (先求解基本情况 & 在此基础上不断扩大求解问题的范围).

e.g. **Bellman-Ford 先计算 $d^{(0)}$, 再计算 $d^{(1)}$, 再计算 $d^{(2)}$, ...**

Top-down: 通过递归调用来解决更小的子问题, 同时通过记忆化/缓存来保存已计算过的问题的解

下面是 **Bellman-Ford** 自顶向下的实现.

海量资源：666java.com

TOP-DOWN BELLMAN-FORD

```
RECURSIVE_BELLMAN_FORD(G,s):
    d^(k) = [None] * n for k = 0, ..., n-1
    d^(0)[v] = ∞ for all v in V (except s)
    d^(0)[s] = 0
    for b in V:
        d^(n-1)[b] ← RECURSIVE_BF_HELPER(G, b, n-1)

RECURSIVE_BF_HELPER(G, b, k):
    A = {a such that (a,b) in E} U {b}    // b's in-neighbors
    for a in A:
        if d^(k-1)[a] is None:                    // not yet solved
            d^(k-1)[a] ← RECURSIVE_BF_HELPTER(G, a, k-1)
    return min{ d^(k-1)[b], min_a{d^(k-1)[a] + w(a,b)} }
```

用来储存已求解子问题解的表.

如果该子问题未被求解过, 我们在计算完成后立即将解进行储存, 之后的计算中不需要在此计算该子问题.

运行时间: O(m·n)

海量资源：666java.com

DYNAMIC PROGRAMMING

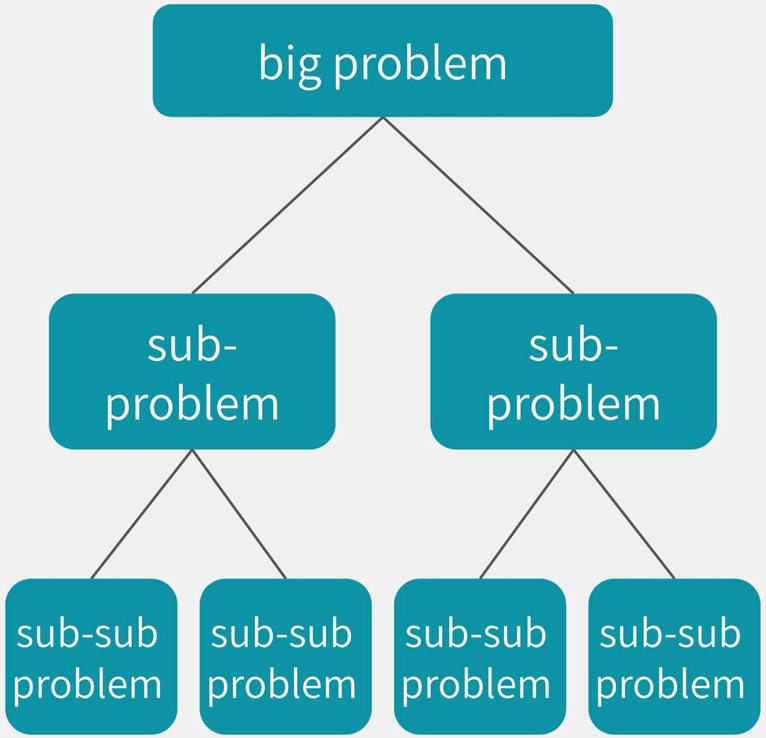
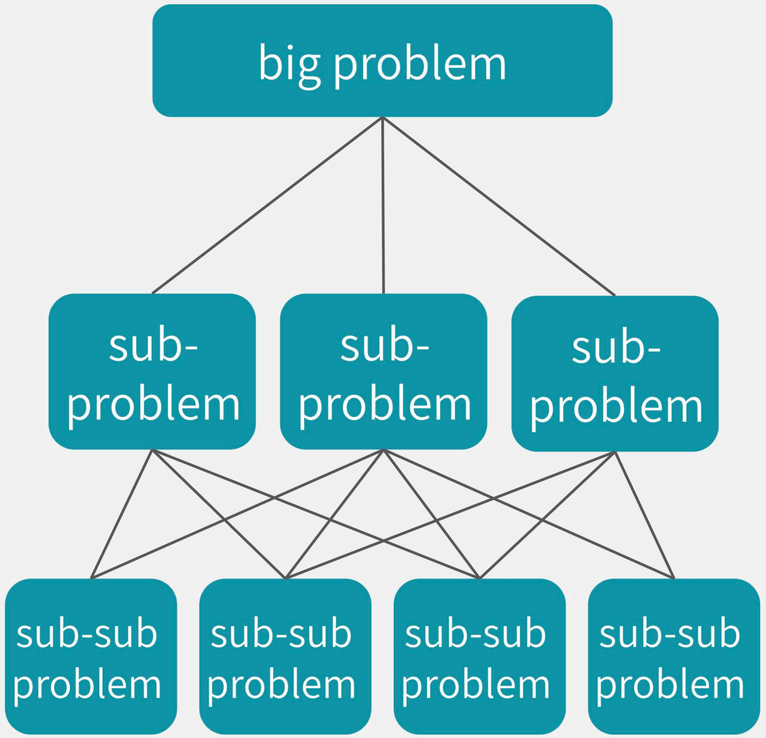
Why “dynamic programming”?

1950年的秋季学期，Bellman 在兰德公司第一件事就是为多阶段决策过程命名。20世纪50年代对数学研究而言并非是有利的时机。当时的国防部长 Wilson 极其厌恶“research”这个词。如果有人当着他的面使用“research”这个词，他的脸色开始变红，然后会变得很暴躁。那么你可以想象一下他听到“mathematical”数学这个词时会有何反应。当时兰德公司被空军雇佣，而空军基本上是受控于 Wilson。因此 Bellman 并不能让 Wilson 和空军知道他在德兰公司里做着与数学相关的研究。他希望能够给当前的研究内容——多阶段决定过程，确定一个合适的名字以掩人耳目。起初，Bellman 对 planning, decision making, thinking 这三个词感兴趣。但出于种种原因，planning 并不合适，反复斟酌后，他决定使用 programming 这个词。接下来，他进一步考虑哪个词可以体现该项研究的动态性，多阶段性和时变性。为此，他想到了一个一举两得的方法。dynamic 这个词不但能准确地表达动态这个物理含义，而且具有形容词的词性，没有贬义的意思。Bellman 想到一些词语组合后可能会有带有贬义的意思，但动态规划 (dynamic programming) 这个名字是他认为连国会会员都不会反对的好名字。

“It’s impossible to use the word dynamic in a pejorative sense… I thought dynamic programming was a good name. It was something not even a Congressman could object to.”

海量资源：666java.com

DIVIDE & CONQUER vs DP

DIVIDE-AND-CONQUER	DYNAMIC PROGRAMMING
	

海量资源：666java.com

FLOYD-WARSHALL

计算所有结点对的最短路径

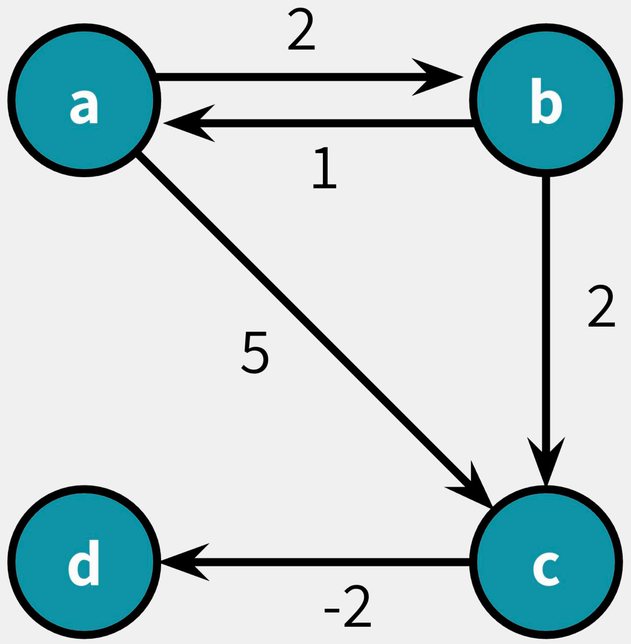
海量资源：666java.com

所有结点对间的最短路径

寻找图中所有结点对 **v**, **w** 间的最短路径
ALL-PAIRS SHORTEST PATHS (APSP)
(而不是某单一源结点 **s**)

Destination

Source	a	b	c	d
a	0	2	4	2
b	1	0	2	0
c	∞	∞	0	-2
d	∞	∞	∞	0



不考虑效率, 如何实现?

海量资源：666java.com

所有结点对间的最短路径

寻找图中所有结点对 **v**, **w** 间的最短路径

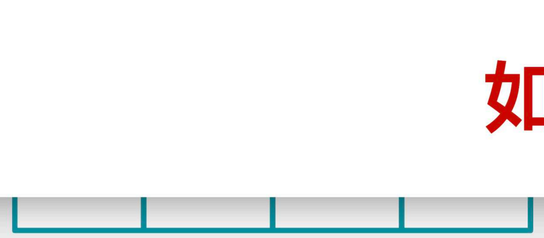
ALL-PAIRS SHORTEST PATHS (APSP)

暴力做法 (支持负权重的边):

```
For all s in G:
      Run Bellman-Ford on G starting at s
```

运行时间: $O(n \cdot mn) = \mathbf{O(mn^2)}$... 当 $m = n^2$ 时将达到 n^4

如何优化?



不考虑效率, 如何实现?

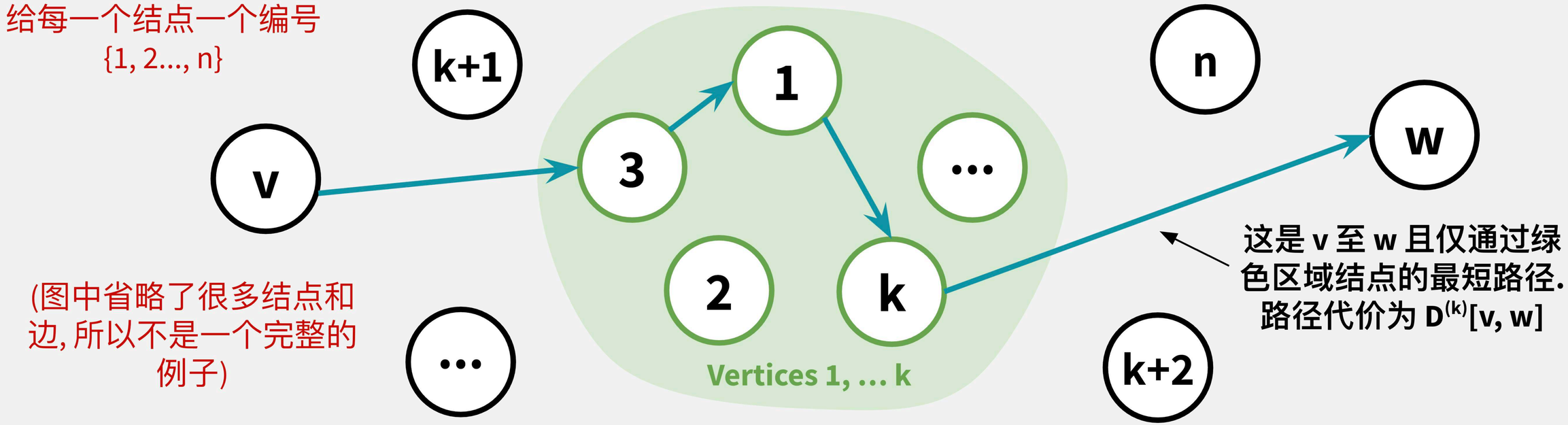
海量资源：666java.com

FLOYD-WARSHALL: DP 实现

我们需要定义最优子结构: 如何构造子问题, 以及如何通过子问题构造原问题的解.

Subproblem(k): 对所有结点对 **v**, **w**, 寻找从 **v** 到 **w** 的最短路径是的路径上的所有结点属于 {1, …, k}

令 $D^{(k)}[v, w]$ 为 Subproblem(k) 的解



海量资源：666java.com

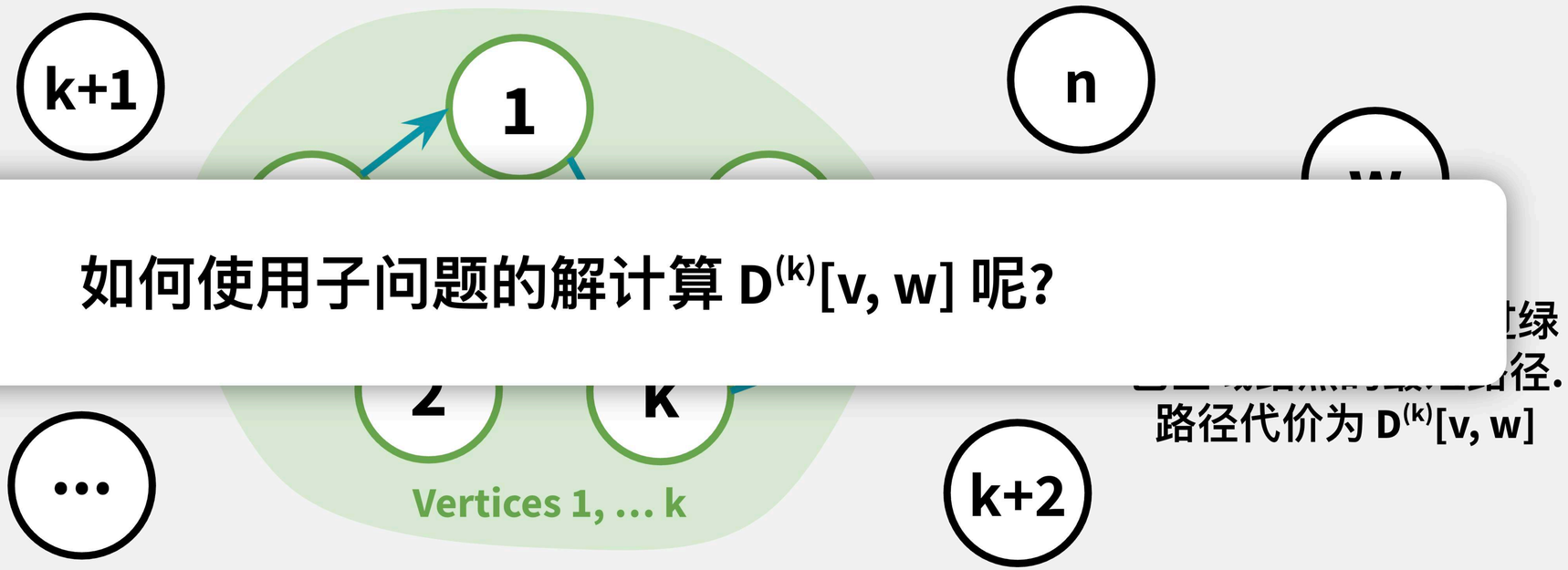
FLOYD-WARSHALL: DP 实现

我们需要定义最优子结构: 如何构造子问题, 以及如何通过子问题构造原问题的解.

Subproblem(k): 对所有结点对 **v**, **w**, 寻找从 **v** 到 **w** 的最短路径是的路径上的所有结点属于 {1, …, k}

令 $D^{(k)}[v, w]$ 为 Subproblem(k) 的解

给每一个结点一个编号 {1, 2..., n}



如何使用子问题的解计算 $D^{(k)}[v, w]$ 呢?

(图中省略了很多结点和边, 所以不是一个完整的例子)

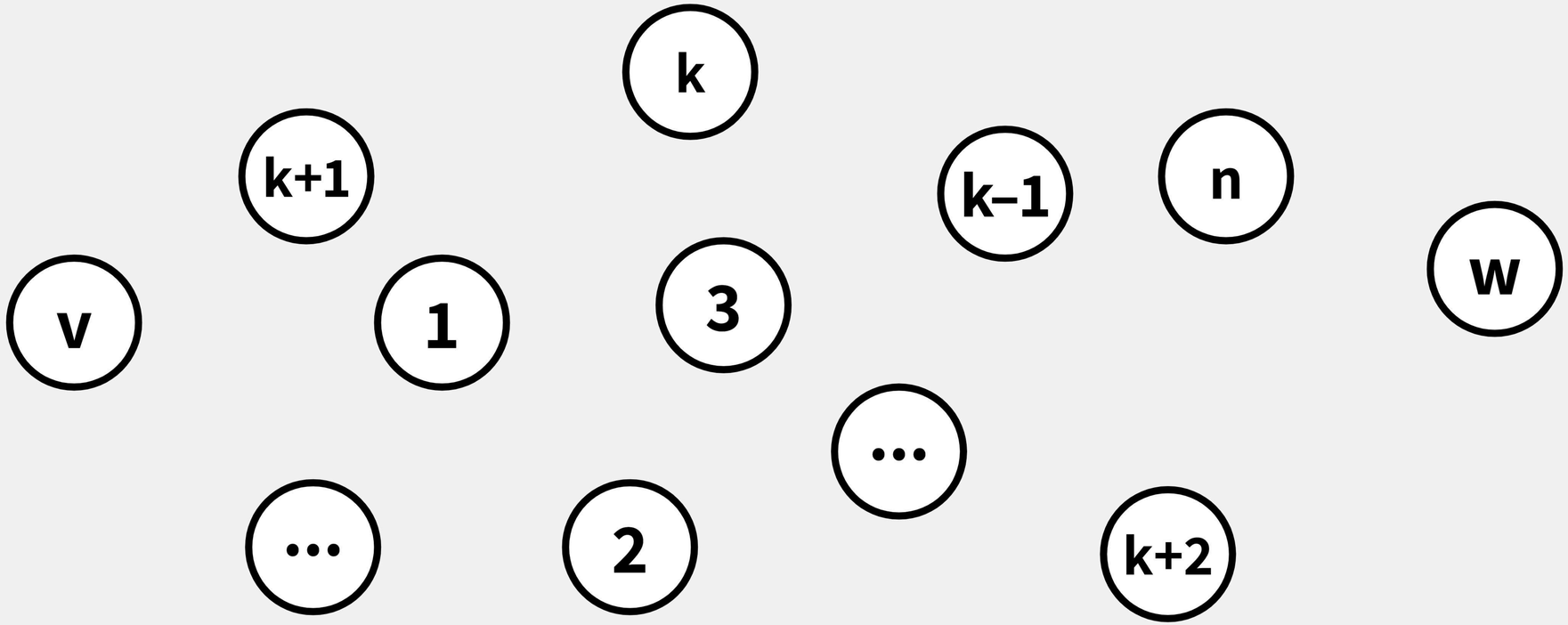
路径代价为 $D^{(k)}[v, w]$

海量资源：666java.com

FLOYD-WARSHALL: DP 实现

$D^{(k)}[v, w]$ 是 **v** 到 **w** 的最短路径, 且路径上的所有结点属于集合 {1, . . ., k}.

考虑两种情况: 路径*不包括*结点 k, 或*包括*结点 k.



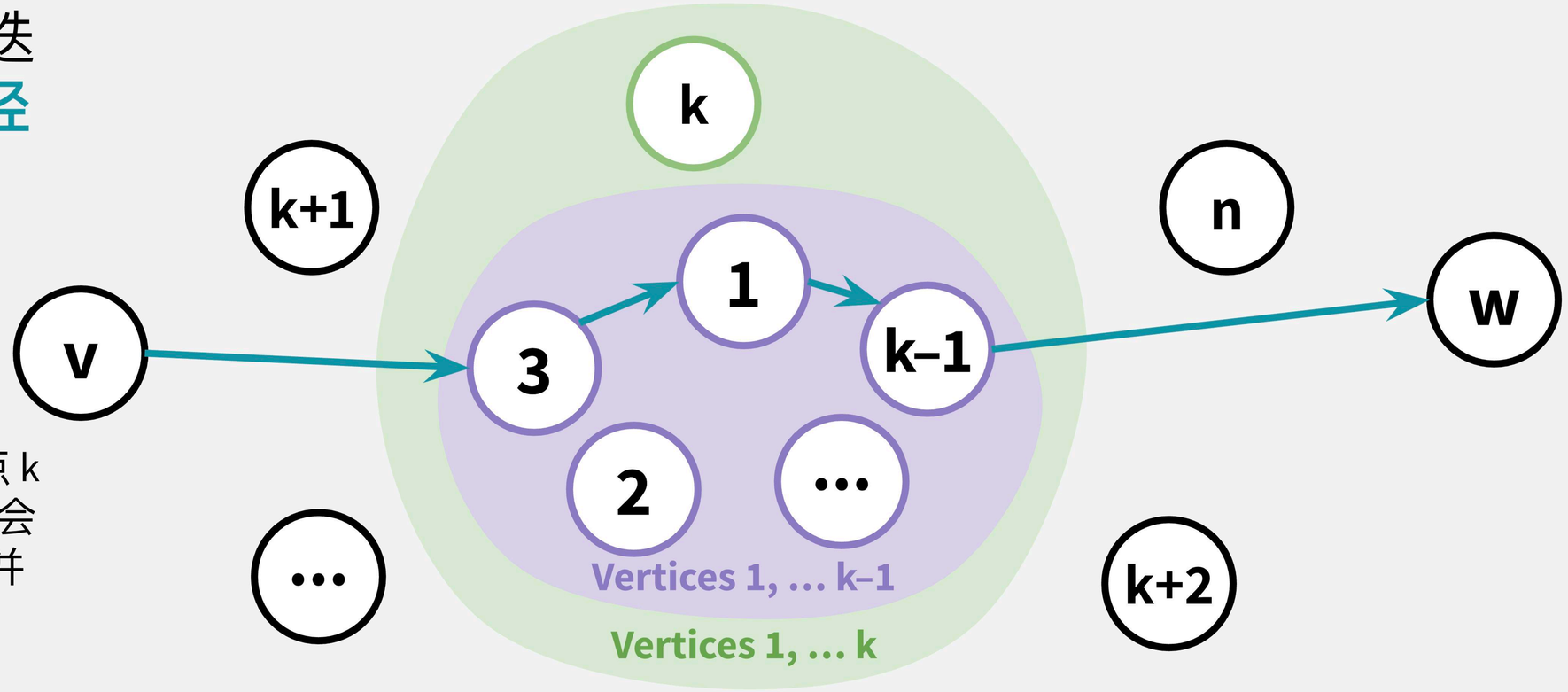
海量资源：666java.com

FLOYD-WARSHALL: DP 实现

$D^{(k)}[v, w]$ 是 v 到 w 的最短路径, 且路径上的所有结点属于集合 {1, …, k}.

CASE 1: 我们不需要用到 k! 此时, $D^{(k)}[v, w] = D^{(k-1)}[v, w]$

这种情况下, 第 k 次迭代依然采纳 **这条路径**



也就是说, 允许路径包括结点 k (在 1, …, k–1 的基础上) 并不会改变最短路径的代价, 因为并不会用到结点 k.

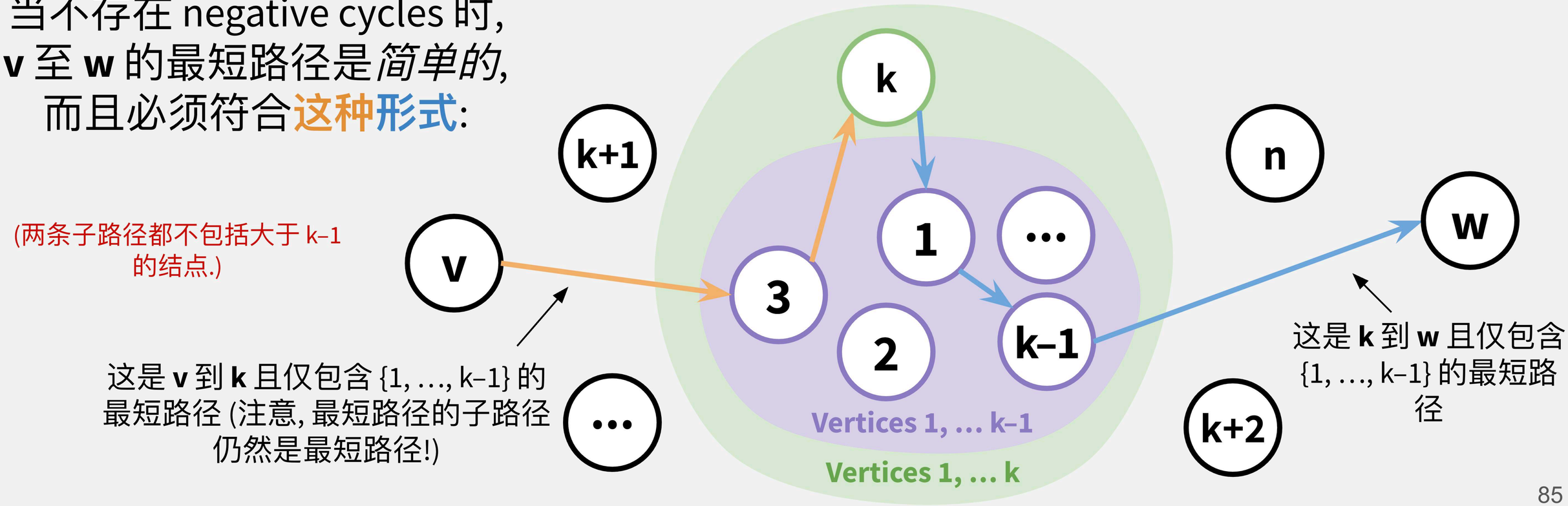
海量资源：666java.com

FLOYD-WARSHALL: DP 实现

$D^{(k)}[v, w]$ 是 v 到 w 的最短路径, 且路径上的所有结点属于集合 {1, …, k}.

CASE 2: 我们需要用到 k! So, $D^{(k)}[v, w] = D^{(k-1)}[v, k] + D^{(k-1)}[k, w]$

当不存在 negative cycles 时,
v 至 w 的最短路径是*简单的*,
而且必须符合这种形式:

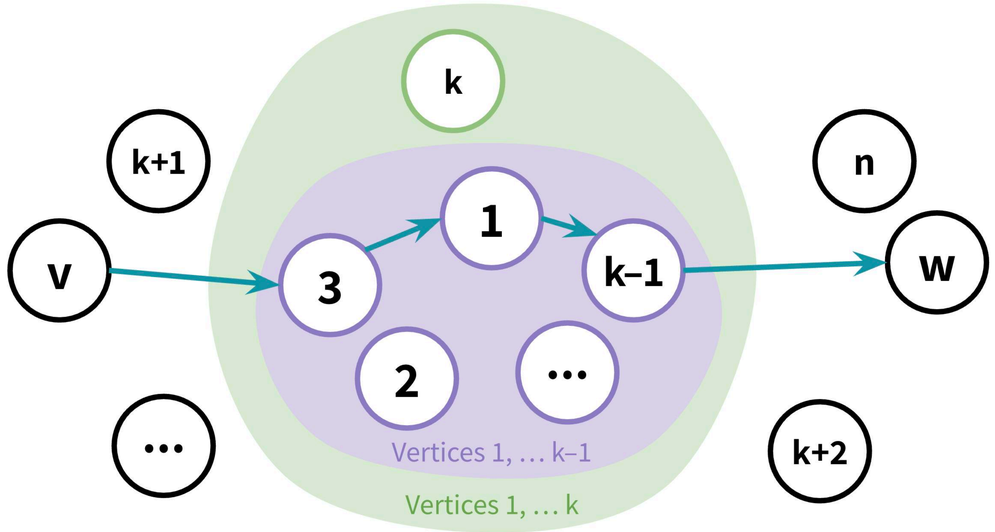


海量资源：666java.com

FLOYD-WARSHALL: DP 实现

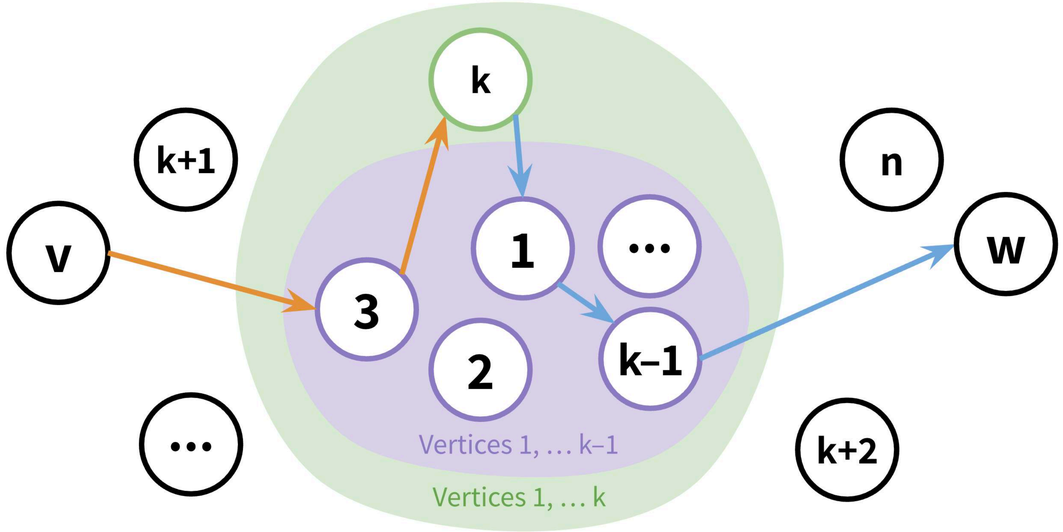
如何通过 $D^{(k-1)}$ 得到 $D^{(k)}[v, w]$？选择下面 2 种情况中的较小值:

CASE 1: We don't need vertex **k**



$$D^{(k)}[v, w] = D^{(k-1)}[v, w]$$

CASE 2: We need vertex **k**



$$D^{(k)}[v, w] = D^{(k-1)}[v, k] + D^{(k-1)}[k, w]$$

海量资源：666java.com

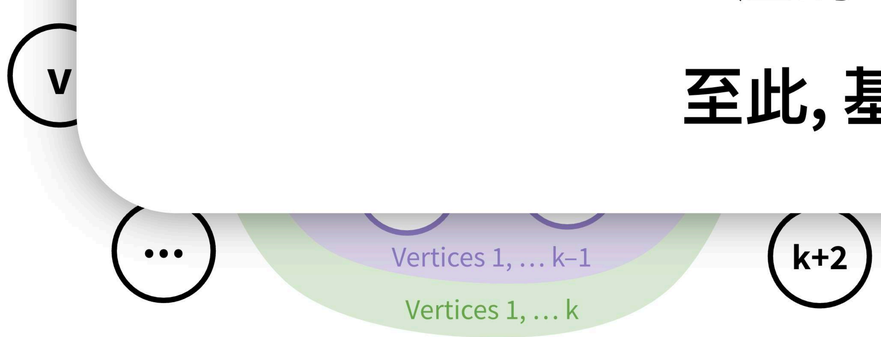
FLOYD-WARSHALL: DP 实现

如何通过 $D^{(k-1)}$ 得到 $D^{(k)}[v, w]$？选择下面 2 种情况中的较小值:

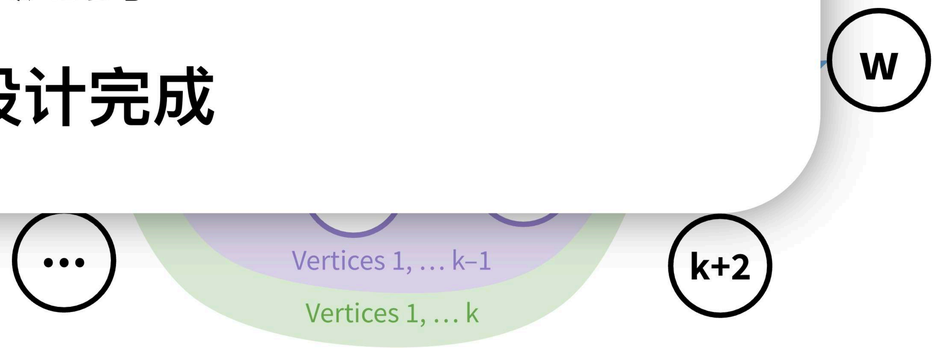
这就是最优子结构: 我们定义了子问题 (寻找特定集合上的最短路径), 并且通过子问题的解构造了最优解 (两种情况中的较小值).

子问题互相重叠: 需要使用记忆化/缓存! 例如, $D^{(k-1)}[k, w]$ 在计算不同 v 对应的 $D^{(k)}[v, w]$ 时都会被用到!

至此, 基于 DP 的算法设计完成



$$D^{(k)}[v, w] = D^{(k-1)}[v, w]$$



$$D^{(k)}[v, w] = D^{(k-1)}[v, k] + D^{(k-1)}[k, w]$$

海量资源：666java.com

FLOYD-WARSHALL: DP 实现

```
FLOYD_WARSHALL(G):
    Initialize n x n arrays D^(k) for k = 0,...,n
        D^(k)[v,v] = 0 for all v, for all k
        D^(k)[v,w] = ∞ for all v ≠ w, for all k
        D^(k)[v,w] = weight(v,w) for all (v,w) in E
    for k = 1,...,n:
        for pairs v,w in V^2:
            D^(k)[v,w] = min{ D^(k-1)[v,w], D^(k-1)[v,k] + D^(k-1)[k,w] }
    return D^(n)
```

储存完整的 n x n 数组会浪费大量空间. 实际上, 我们只需要储存2 行!

两种情况中取较小值

运行时间: $O(n^3)$

(优于运行 n 遍 Bellman-Ford)

海量资源：666java.com

存在 NEGATIVE CYCLES 时会发生什么?

Negative cycle 表示对于某些 **v**
存在一条 **v** 到 **v** 且代价小于 0 的路径

```
FLOYD_WARSHALL(G):
   Initialize n x n arrays D(k) for k = 0,...,n
      D(k)[v,v] = 0 for all v, for all k
      D(k)[v,w] = ∞ for all v ≠ w, for all k
      D(k)[v,w] = weight(v,w) for all (v,w) in E
   for k = 1,...,n:
      for pairs v,w in V2:
         D(k)[v,w] = min{ D(k-1)[v,w], D(k-1)[v,k] + D(k-1)[k,w] }
   for v in V:
      if D(n)[v,v] < 0:
         return “NEGATIVE CYCLE!”
   return D(n)
```

海量资源：666java.com

最短路径算法

$n = |V|$
$m = |E|$

BFS	DFS	DIJKSTRA	BELLMAN-FORD	FLOYD-WARSHALL
$O(m+n)$	$O(m+n)$	$O(m+n\log n)$*	$O(mn)$	$O(n^3)$
无权重 (或问题不关心权重)	无权重 (或问题不关心权重)	含权重 (权重必须为非负)	含权重 (可以处理负权重)	含权重 (可以处理负权重)
单源最短路径 判断二分图 寻找连通分量	寻找路径 (s,t) 拓扑排序 (DAG!!) 寻找 SCC's 寻找连通分量	***单源最短路径:*** 计算从源结点 s 至所有结点的最短路径	***单源最短路径:*** 计算从源结点 s 至所有结点的最短路径 检测 negative cycles	***所有结点对间的最短路径:*** 计算所有结点对 (v,w) 间的最短路径

海量资源：666java.com

总结

- 介绍了使用 Bellman-Ford 求解单源最短路径问题, 同时支持负权重边 (Dijkstra 无法处理的情况).
    - 第一个动态规划算法!
- 介绍了使用 Floyd-Warshall 求解所有结点对间的最短路径问题, 同样支持负权重边.
    - Floyd-Warshall 也是动态规划算法!
- 粗略介绍了什么是动态规划. 感觉有些难? - BF 和 F-W 是两种非常巧妙的动态规划算法 (实际遇到的问题要远比此容易).

海量资源：666java.com

接下来

- 更多动态规划问题!