算法设计与分析

第6讲 动态规划

汪小林

北京大学 信息科学技术学院

主要内容

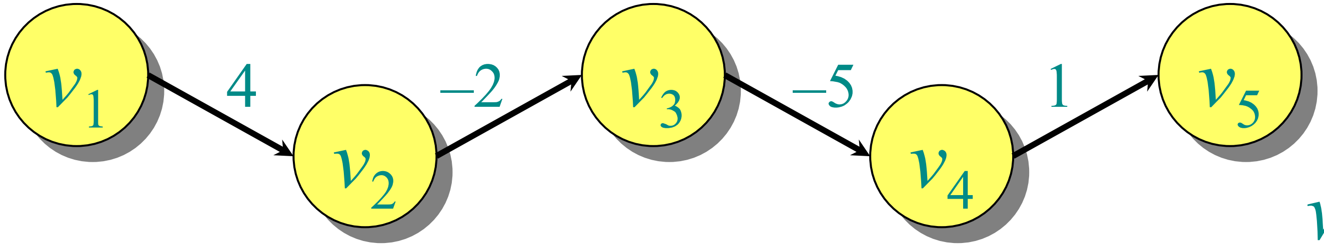
- 带负权边的最短路径问题（Bellman-Ford算法）

最短路径问题

- 路径（Path）：
  对于带权有向图 $G = (V, E)$，边权重函数为 $w: E \rightarrow \mathbf{R}$，路径 $p = v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_k$ 的权重定义为

$$w(p) = \sum_{i=1}^{k-1} w(v_i, v_{i+1})$$

- 例子：



$w(p) = -2$

最短路径

- 从 u 到 v 权重最小的路径称为从 u 到 v 的最短路径。
- 从 u 到 v 的最短路径权重定义为：

$$\boldsymbol{\delta(u,v) = min\{w(p) : p}\text{是一条}\boldsymbol{u}\text{到}\boldsymbol{v}\text{间的路径}\}$$

- 如果 u 到 v 间不存在路径，则记

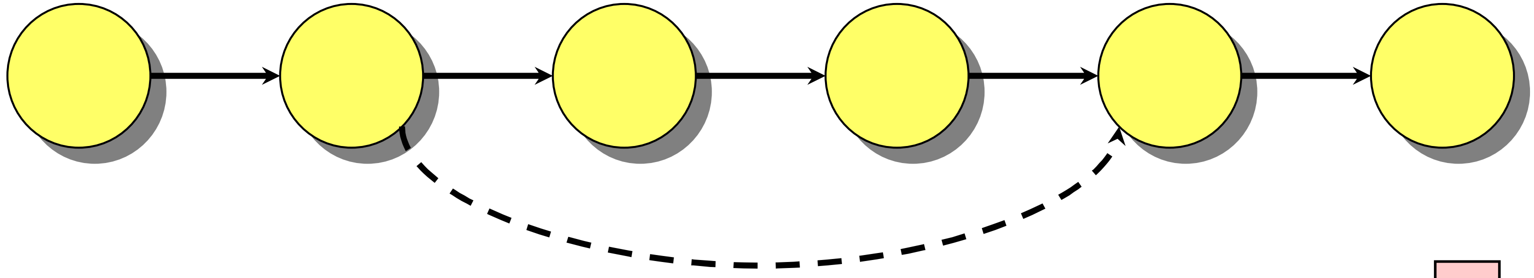
$$\boldsymbol{\delta(u,v)} = \infty$$

最短路径具有最优子结构性质

定理：最短路径的子路径也是最短路径，即：

- 设 $p = v_1 \to v_2 \to \ldots \to v_k$ 是从 v_1 到 v_k 的最短路径，则 p 的子路径 $p_{ij} = v_i \to v_{i+1} \to \ldots \to v_j$ 也是从 v_i 到 v_j 的最短路径，其中 $1 \le i < j \le k$。

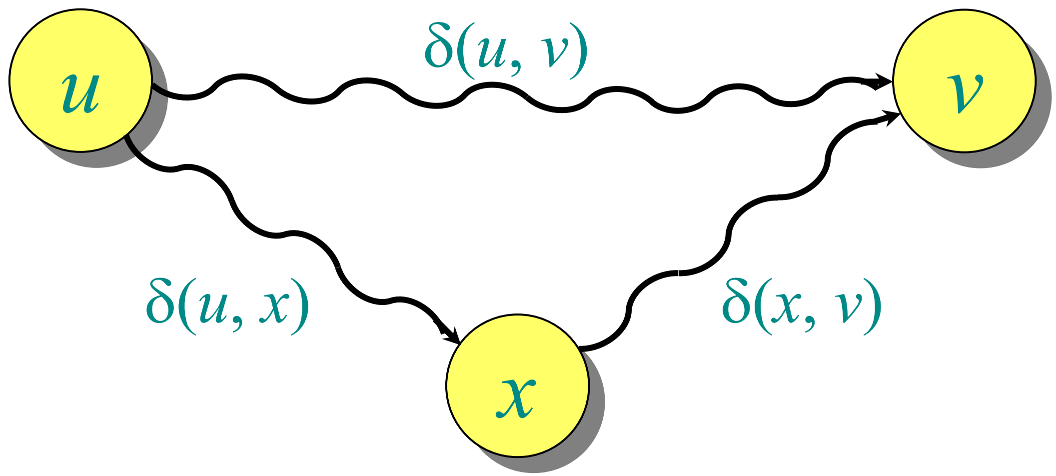
- 证明：反证法，如果不是，通过交换子路径可以获得从 v_1 到 v_k 的更短的路径，与 p 是最短路径矛盾。



$\square$

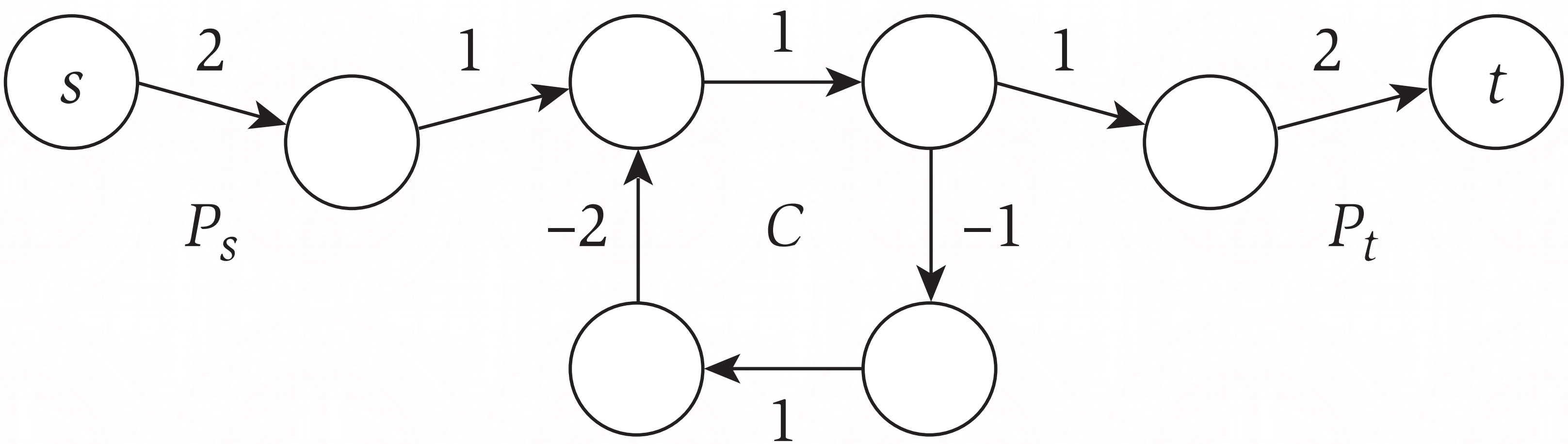
最短路径满足三角不等式

定理：对于任意的 $u, v, x \in V$，有 $\delta(u, v) \leq \delta(u, x) + \delta(x, v)$



最短路径存在的必要条件

- 如果图 G 含有负权环，则某些顶点之间不存在最短路径。



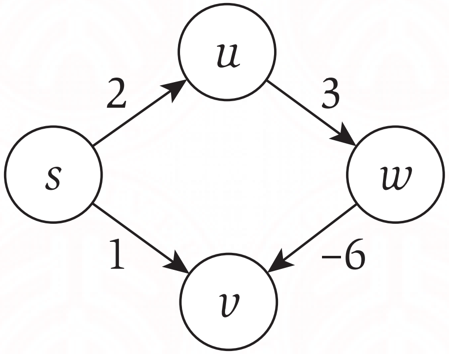
单源最短路径问题

- 问题：给定图 G 中的顶点 s，求它到所有顶点 $v \in V$ 的最短路径权重 $\delta(s, v)$。
- 当所有边的权重 $\delta(u, v)$ 非负时，该问题可解。
- 可以贪心法求解（Dijkstra算法）：
  1. 维护所有到 s 最短距离已知的顶点集合 S；
  2. 将到 s 估算距离最短的顶点 $v \in V\text{-}S$ 加入集合 S；
  3. 更新所有与 v 相连的顶点到 s 的估算距离；
  4. 重复步骤 2、3，直到 $V\text{-}S=\Phi$。

考虑图中有负权边

- 如何判断图中是否有负权环？
- 如果不存在负权环，如何求最短路径？

- 此时，**Dijkstra**算法不再可用。



无负权环的图中存在简单最短路径

- **命题：**如果图G（无向连通带权图）中无负权环，则在顶点s和t之间存在一条最短路径，并且该路径是一条简单路径（无重复顶点），因此该路径的边数至多为$n-1$条。
- **证明：**因为每个环均为非负，故边数最少的s-t最短路径P中无重复顶点，否则可以通过消除环的方式得到长度不会更长，但边数更少的路径。
- 由于s-t简单路径上最多有n-2个中间顶点，路径数量有限，故最短路径必然存在。

最短路径权值上的递推方程

- 定义 $M(i, v)$ 为边数不超过 i 条的 v-t 路径的最小权值（最短路径长度）。
- 求 s 到 t 的最短距离为：$M(n\text{-}1, s)$

- **递推方程（6.23）**

$$M(0, v) = 0$$

$$M(i, v) = min\left\{M[i-1, v], \min_{x \in V}[M(i-1, x) + w(v, x)]\right\}, i \geq 1$$

一个动态规划算法

Shortest-Path(G, s, t)

n= number of nodes in G

Array $M[0...n-1,V]$

Define $M[0,t]=0$ and $M[0,v]=\infty$ for all other $v\in V$

For $i=1,...,n-1$

For $v\in V$ in any order

Compute $M[i, v]$ using the recurrence (6.23)

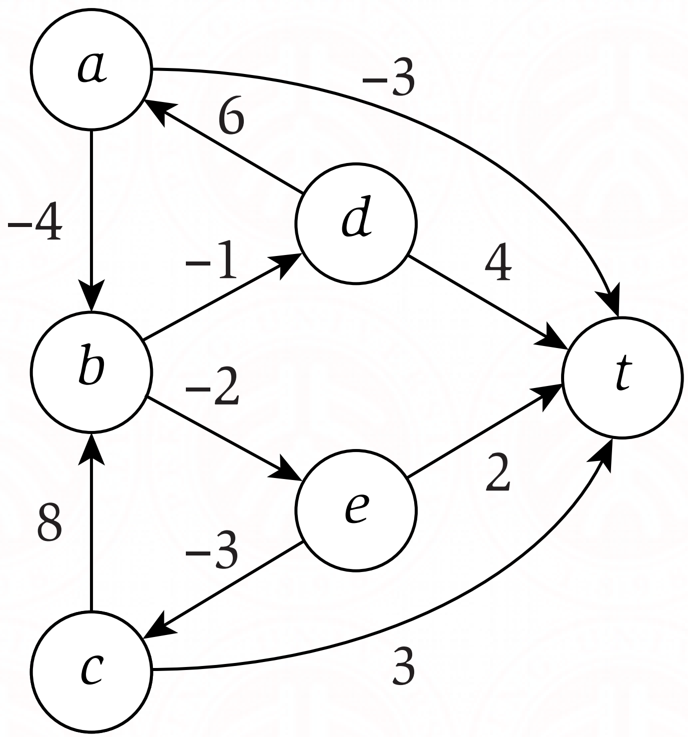
Endfor

Endfor

Return $M[n\text{-}1, s]$

算法的时间效率和空间效率？

算法运行实例



	0	1	2	3	4	5
t	0	0	0	0	0	0
a	∞	−3	−3	−4	−6	−6
b	∞	∞	0	−2	−2	−2
c	∞	3	3	3	3	3
d	∞	4	3	3	2	0
e	∞	2	0	0	0	0

Bellman-Ford 算法

Shortest-Path(*G*, *s*, *t*)

n = number of nodes in G

Array $M[V]$

Define $M[t]$=0, $f[t]$=t, and $M[v]$=∞, $f[v]$=nil for all other $v \in V$

For i=1,...,$n-1$

For $v \in V$ in any order

Compute $M[v]$ and update $f[v]$ using the recurrence

Endfor

$$M(v) = min\left\{M[v], \min_{x \in V}[M[x] + w(v,x)]\right\}$$

Endfor

$$f(v) = x \text{ iif } M(v) \text{ decreases with node } x$$

Return $M[s]$

Bellman-Ford 算法的正确性

- **(6.26)** *Throughout the algorithm* $M[v]$ *is the length of some path from* v *to* t, *and after* i *rounds of updates the value* $M[v]$ *is no larger than the length of the shortest path from* v *to* t *using at most* i *edges.*
- **(6.27)** *If the pointer graph* P (*of* $(v, f[v])$) *contains a cycle* C, *then this cycle must have negative cost.*
- **(6.28)** *Suppose* G *has no negative cycles, and consider the pointer graph* P *at the termination of the algorithm. For each node* v, *the path in* P *from* v *to* t *is a shortest* v-t *path in* G.

Bellman-Ford 算法优化

Push-Based-Shortest-Path(G, s, t)

n= number of nodes in G

Array $M[V]$

Initialize $M[t]$=0 and $M[v]=\infty$ for all other $v \in V$

For i=1,...,$n-1$

For $x \in V$ in any orde

If $M[x]$ has been updated in the previous iteration then

For all edges (v, x) in any order

$M[v]= \min(M[v], w(v, x) + M[x])$

If this changes the value of $M[v]$, then *first*[v]=x

Endfor

Endfor

If no value changed in this iteration, then end the algorithm Endfor

Return $M[s]$

分布式的优化：用于网络路由的算法

Asynchronous-Shortest-Path(*G*, *s*, *t*)

n= number of nodes in G

Array $M[V]$

Initialize $M[t]$=0 and $M[v]=\infty$ for all other $v \in V$

Declare *t* to be active and all other nodes inactive

While there exists an active node

Choose an active node *x*

For all edges (v, x) in any order

$M[v]= \min(M[v], w(v, x) + M[x])$

If this changes the value of $M[v]$, then

$first[v]=x$ and v becomes active

Endfor

x becomes inactive

EndWhile

如何确定是否有负权环？

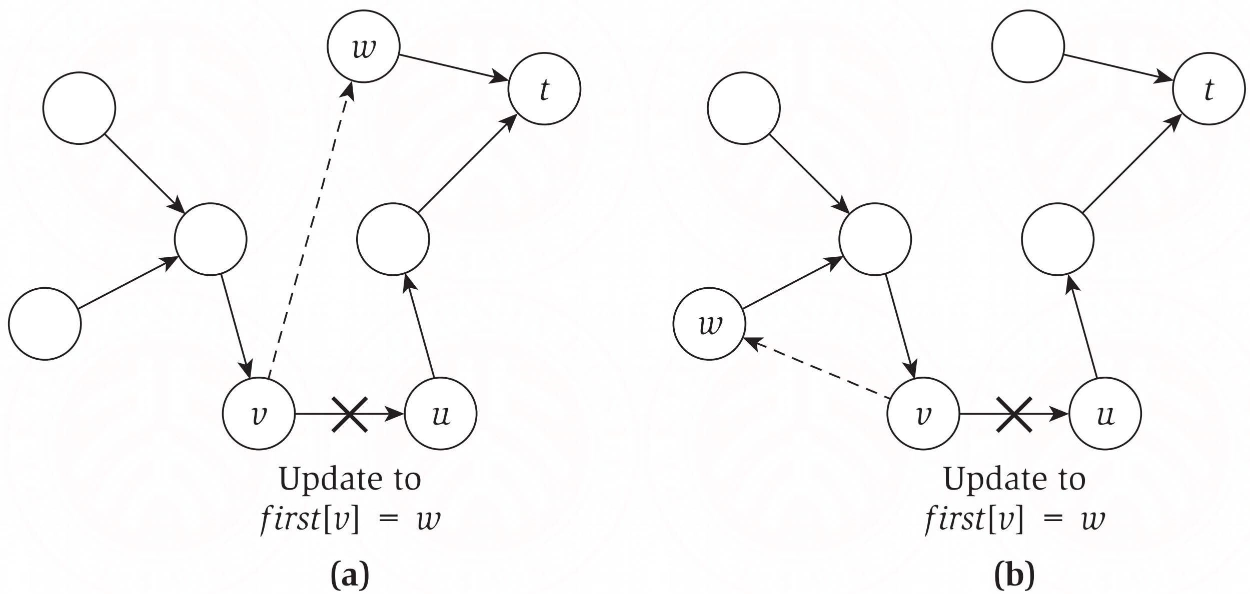
- 定义 **M(i, v)** 为不超过 i 条边的 v-t 路径的最小权值（最短路径长度），其中 i 可以大于等于 n 。
- **(6.30)** *If node v can reach node t and is contained in a negative cycle, then* $M(i, v) = -\infty$ *as* $i \to +\infty$
- **(6.31)** *If there are no negative cycles in G, then* $M(i, v) = M(n-1, v)$ *for all nodes v and all $i \geq n$.*
- **(6.32)** *There is no negative cycle with a path to t if and only if* $\mathrm{M}(n, v) = \mathrm{M}(n-1, v)$ *for all nodes v.*

如何找到负权环？

- Bellman-Ford 算法循环 n 次。
- **(6.33)** *If G has n nodes and $M(n,v) \neq M(n-1,v)$, then a path P from v to t of cost $M(n, v)$ contains a cycle C, and C has negative cost.*

改进的最短路及负权环算法

- 如何更早的判断负权环的存在？
- 考虑在更新 ***M*[v]** 和 ***f*[v]** 时可能构成环。
  - 判断环的方法？



休眠结点

- 两种判定方法
  - 沿着路径的方向前进直到遇到起点 v；
  - 维护所有到达结点 v 的路径树，判断 $w=f[v]$ 是否在树中。
- 都需要 $O(n)$ 的时间
  - 但是，对第二种方法可以对结果加以利用
  - 当更新某级结点时，可以将该结点子树中的所有结点均置为休眠结点：
    - 这些结点的 M 值仍待更新（更新后恢复为一般结点）；
    - 其他结点 M 值的更新不参照休眠结点。
  - 被置为休眠结点的次数不超过结点 M 值更新的次数。

Bellman-Ford-OPT

Shortest-Path(G, s, t)

$M[t]=0$, $f[t]=t$;

$M[v]=\infty$, $f[v]=$nil, for all other $v \in V$.

Enqueue t to the active FIFO queue Q

While Q is not empty, do // implicit iterations of i from 1 to n

x = dequeue(Q), if $f[x]$ is nil then continue

For all edges (v, x) in any order

$M[v]= \min(M[v], w(v, x) + M[x])$

If this changes the value of $M[v]$, then // $f[\ldots f[f[y]]\ldots] = v$

$f[y]=$nil, for all y in the subtree directed toward v

If x in the subtree, then report a negative circle and return

$f[v]=x$, and enqueue v to Q if v is not in Q

Endfor

EndWhile

Return $M[s]$

增加了结点休眠后算法的性质

- **(6.35)**
  Throughout the algorithm $M[v]$ is the length of some simple path from v to t;
  the path has at least i edges if the distance value $M[v]$ is updated in iteration i;
  and after i iterations, the value $M[v]$ is the length of the shortest path for all nodes v where there is a shortest v-t path using at most i edges.

改进的最短路及负权环算法

- **(6.36)** *The improved algorithm outlined above finds a negative cycle in* G *if such a cycle exists. It terminates immediately if the pointer graph* P *of* $first[v]$ *pointers contains a cycle* C, *or if there is an iteration in which no update occurs to any distance value* $M[v]$. *The algorithm uses* $O(n)$ *space, has at most* n *iterations, and runs in* $O(mn)$ *time in the worst case.*