算法设计与分析—进阶篇

第二讲 再论动态规划

哈尔滨工业大学
王宏志
wangzh@hit.edu.cn
http://homepage.hit.edu.cn/pages/wang/

本讲内容

2.1 优化子结构的分类
2.2 编号动态规划问题
2.3 划分动态规划问题
2.4 数轴动态规划问题
2.5 前缀动态规划问题

- 动态规划算法的设计步骤
  - 分析优化解的结构
  - 递归地定义最优解的代价
  - 自底向上地计算优化解的代价保存之，并获取构造最优解的信息
  - 根据构造最优解的信息构造优化解

子问题的结构

- **编号动态规划**：输入为$x_1, x_2, \ldots, x_n$, 子问题是$x_1, x_2, \ldots, x_i$，子问题复杂性为O(n)(最大不下降子序列问题)
- **划分动态规划**：输入为$x_1, x_2, \ldots, x_n$, 子问题为$x_i, x_{i+1}, \ldots, x_j$，子问题复杂性是$O(n^2)$ (矩阵链乘问题)
- **数轴动态规划**：输入为$x_1, x_2, \ldots, x_n$和数字C，子问题为$x_1, x_2, \ldots, x_i$, K(K≤C)，子问题复杂性O(nC) (0-1背包问题)
- **前缀动态规划**：输入为$x_1, x_2, \ldots, x_n$和$y_1, y_2, \ldots, y_m$，子问题为$x_1, x_2, \ldots, x_i$和$y_1, y_2, \ldots, y_j$，子问题复杂性是O(mn) (最长公共子序列问题)
- **树形动态规划**：输入是树，其子问题为子树，子问题复杂性是子树的个数。(树中独立集合问题)

本讲内容

2.1 优化子结构的分类

2.2 编号动态规划问题

2.3 划分动态规划问题

2.4 数轴动态规划问题

2.5 前缀动态规划问题

编号动态规划

一般有两种表示状态的方法：

1) 状态i表示前i个元素构成的最优解，可能不包含第i个元素。
2) 状态i表示在必须包含第i个元素的情况下前i个元素构成的最优解。

最长不下降子序列

- 输入：一个数字序列 $a[1..n]$
- 子序列是数字序列的子集合，且和序列中数字顺序相同，即

$$a_{i_1}, a_{i_2}, \ldots, a_{i_k} \quad (1 \le i_1 < i_2 < \cdots < i_k \le n)$$

递增子序列是其中数字严格增大的子序列

- 输入: 具有最大长度的递增子序列.

例子

- Input.　　5; 2; 8; 6; 3; 6; 9; 7
- Output.　2; 3; 6; 9

- 5　2　8　6　3　6　9　7

算法分析

- **优化子结构：** 假设最长递增子序列中包含元素a_k，那么一定存在一组最优解，它包含了$a_1, a_2, \ldots, a_{k-1}$这个序列的最长递增子序列。正确性？
- **重叠子问题**：a_k和a_{k+1}?
- **子问题的表示：** 令dp[i]表示以第i个元素结尾的前i个元素构成序列的最长递增子序列的长度。
- **最优解递归表达式：**

$$dp[i] = \max \{ dp[j] \mid 0 < j < i; a_j < a_i \} + 1$$

最长递增子序列算法代价计算伪代码

输入： 数组a[1..n]

输出： a的最长递增子序列长度

```
FOR  i =  TO  n
    dp[i] ← 0;
    FOR  j  =1  TO  i – 1
        IF  a[j] <= a[i] AND dp[j] > dp[i]
            THEN  dp[i] ← dp[j];
    dp[i] ← dp[i] + 1;
RETURN dp[n]
```

最长递增子序列算法的优化

- 上面的算法的时间复杂度为$O(n^2)$，是否可以优化呢？
- **分析**：设$A_i = \min \{ a_j \mid dp[j] == i\}$，那么如果$i > j$，一定可以推出$A_i \geq A_j$。
- **递推函数**：

$$dp[i] = \max \{ j \mid A_j > a_i\} + 1;$$

（$\max\{j \mid Aj \geq ai\}$可以通过二分查找求出）

本讲内容

2.1 优化子结构的分类
2.2 编号动态规划问题
2.3 划分动态规划问题
2.4 数轴动态规划问题
2.5 前缀动态规划问题

问题的定义

- **多边形**

  多边形表示为顶点坐标集$P=(v_0,v_1,\ldots v_n)$或顶点序列$v_0v_1,\ldots v_{n-1}v_n$

- **简单多边形**

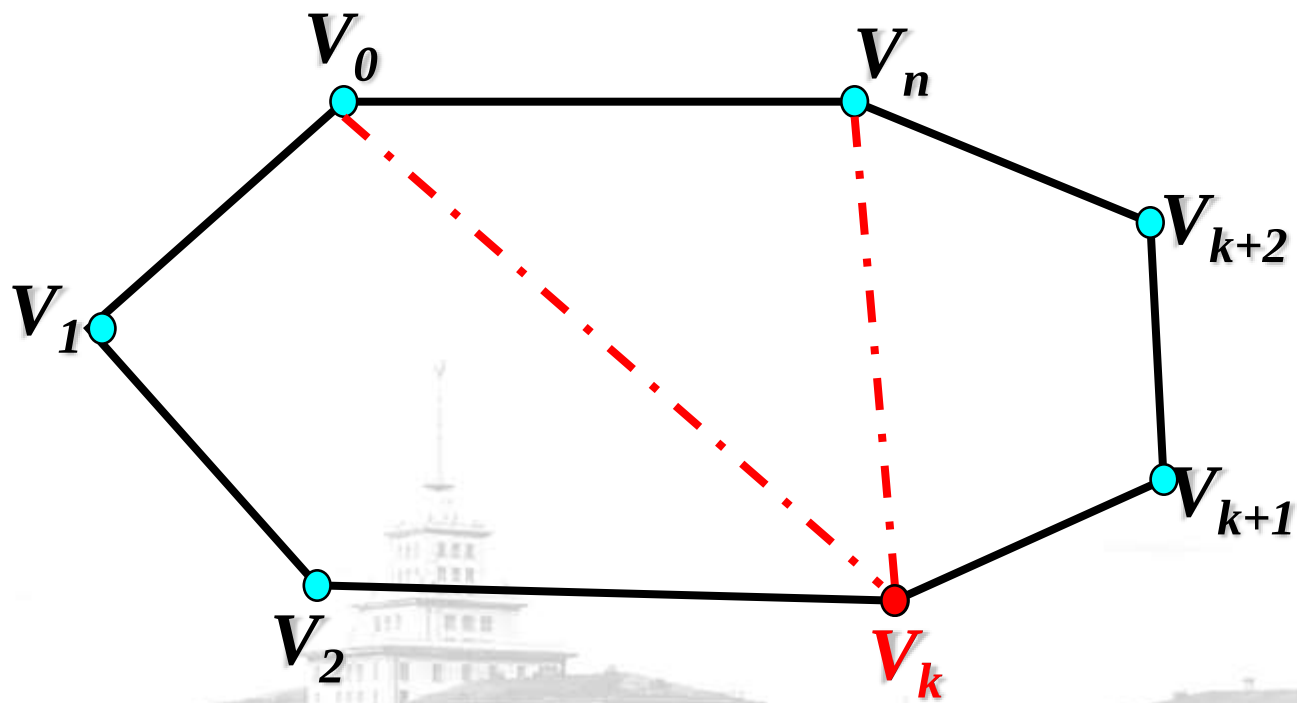
  除了顶点以外没有任何边交叉点的多边形

- **多边形的内部、边界与外部**
  - 平面上由多边形封闭的点集合称为多边形**内部**，
  - 多边形上的点集合称为多边形的**边界**
  - 平面上除多边形内部和边界以外的点集合称为多边形的**外部**

- 弦
  - 多边形P上的任意两个不相邻结点v_i、v_{i+1}所对应的线段v_iv_{i+1}称为弦
- 三角剖分
  - 一个多边形P的三角剖分是将P划分为不相交三角形的弦的集合
- 优化三角剖分问题
  - 输入：多边形P和代价函数W
  - 输出：求P的三角剖分T，使得代价$\sum_{s\in S_T} W(s)$最小，其中S_T是T所对应的三角形集合

优化解结构的分析

- 设
  - $P=(v_0,v_1,\ldots v_n)$是$n+1$个顶点的多边形
  - T_P是P的优化三角剖分，包含三角形$v_0v_k v_n$



- $T_P=T(v_0, \ldots, v_k) \cup T(v_k, \ldots, v_n) \cup \{v_0v_k, v_kv_n\}$

优化三角剖分的代价函数

- 设 $t[i,j] = <v_{i-1}, v_i, \ldots, v_j>$的优化三角剖分代价

$$t[i, i] = t[j, j] = 0$$

$$t[i, j] = \min \{t[i, k] + t[k+1, j] + w(A$$

注意：

$t[i, k] = <v_{i-1}, v_i, \ldots, v_k>$的优化三角剖分代价

$t[k+1, j] = <v_k, v_i, \ldots, v_j>$的优化三角剖分代价

优化三角剖分动态编程算法

与矩阵链乘法问题一致，把算法

Matrix-chain-Order

Print-Optimal-Parens

略加修改即可计算*t[i,j]*并构造优化三角剖分解

本讲内容

2.1 优化子结构的分类
2.2 编号动态规划问题
2.3 划分动态规划问题
2.4 数轴动态规划问题
2.5 前缀动态规划问题

问题的定义

给定n种物品和一个背包，物品i的重量是w_i，价值v_i，背包容量为C，问如何选择装入背包的物品，使装入背包中的物品的总价值最大？

对于每种物品只能选择完全装入或不装入，一个物品至多装入一次。

- 输入：$C>0, w_i>0, v_i>0, 1\le i\le n$
- 输出：$(x_1, x_2, \ldots, x_n), x_i\in\{0, 1\}$, 满足

$\Sigma_{1\le i\le n} w_i x_i \le C$, $\Sigma_{1\le i\le n} v_i x_i$ 最大

等价的整数规划问题

$$\max \Sigma_{1\le i\le n} v_i x_i$$

$$\Sigma_{1\le i\le n} w_i x_i \le C$$

$$x_i\in\{0, 1\},\ 1\le i\le n$$

优化解结构的分析

定理 (优化子结构) 如果$(y_1, y_2, \ldots, y_n)$是0-1背包问题的优化解，则$(y_2, \ldots, y_n)$是如下子问题的优化解：

$$max \sum_{2\le i\le n} v_i x_i$$
$$\sum_{2\le i\le n} w_i x_i \le C - w_1 y_1$$
$$x_i \in \{0, 1\},\ \ 2\le i \le n$$

证明： 如果$(y_2, \ldots, y_n)$不是子问题优化解, 则存在$(z_2, \ldots, z_n)$是子问题更优的解。于是, $(y_1, z_2, \ldots, z_n)$是原问题比$(y_1, y_2, \ldots, y_n)$更优解，矛盾。

建立优化解代价的递归方程

- 设子问题

$$\max \sum_{i \le k \le n} v_k x_k$$
$$\sum_{i \le k \le n} w_k x_k \le j$$
$$x_k \in \{0, 1\}, \ i \le k \le n$$

的最优解代价为$m(i, j)$.

即$m(i, j)$是背包容量为j, 可选物品为i, $i+1, \ldots, n$ 时问题最优解的代价.

递归方程

$$m(i, j) = m(i+1, j) \qquad 0 \le j < w_i$$
$$m(i, j) = \max\{m(i+1, j), m(i+1, j-w_i)+v_i\} \qquad j \ge w_i$$

$$m(n, j) = 0 \qquad 0 \le j < w_n$$
$$m(n, j) = v_n \qquad j \ge w_n$$

自底向上计算优化解的代价

计算$m(i, j)$需要 $m(i+1, j-w_i)$和$m(i+1, j)$

令w_i=整数, $n=4$

$m(1,C)$

$m(2, 1)$ ••• ••• ••• $m(2,C-w_1)$ ••• $m(2,C)$

$m(3, 1)$ ••• $m(3, C-w_1-w_2)$ ••• $m(3,C-w_2)$ ••• $m(3,C-w_1)$ ••• $m(3,C)$

$m(4, 1)$ ••• $m(4, C-w_1-w_2)$ •• $m(4,C-w_2)$ ••• $m(4,C-w_1)$ ••• $m(4, 6)$

- 算法

0 **To** $\min(w_n-1, C)$ **Do**

  $m[n, j] = 0;$

For $j=w_n$ **To** C **Do**

  $m[n, j] = v_n;$

For $i=n-1$ **To** 2 **Do**

  For $j=0$ **To** $\min(w_i-1, C)$ **Do**

    $m[i, j] = m[i+1, j];$

  For $j=w_i$ **To** C **Do**

    $m[i, j]=\max\{m[i+1, j], m[i+1, j-w_i]+v_i\};$

If $C<w_1$

Then $m[1, C]=m[2, C];$

Else $m[1, C]=\max\{m[2, C], m[2, C-w_1]+v_1\};$

构造优化解

1. $m(1, C)$是最优解代价值，相应解计算如下：

   If $m(1, C)=m(2, C)$

   Then $x_1=0$

   Else $x_1=1$;

2. 如果$x_1=0$，由$m(2, C)$继续构造最优解；

3. 如果$x_1=1$，由$m(2, C-w_1)$继续构造最优解.

本讲内容

2.1 优化子结构的分类
2.2 编号动态规划问题
2.3 划分动态规划问题
2.4 数轴动态规划问题
2.5 前缀动态规划问题

字符串相似性

- 两个字符串有多像?
  - ocurrance
  - occurrence

o	c	u	r	r	a	n	c	e	-
o	c	c	u	r	r	e	n	c	e

o	c	-	u	r	r	a	n	c	e
o	c	c	u	r	r	e	n	c	e

o	c	-	u	r	r	-	a	n	c	e
o	c	c	u	r	r	e	-	n	c	e

编辑距离

输入: 两个字符串$A = a_1a_2 .... a_m$ 和$B = b_1b_2 ... b_n$

输出： 将A变成B的编辑操作序列的最小代价 $D(A,B)$.

编辑操作:

1. 用B中的一个字符替换A中的一个字符
2. 删除A中的一个字符
3. 插入一个 B中的字符

编辑距离的计算

三种可能:

1. a_m 被b_n 替代:

$$D_{m,n} = D_{m\text{-}1,n\text{-}1} + 1$$

2.删除a_m : $D_{m,n} = D_{m\text{-}1,n} + 1$

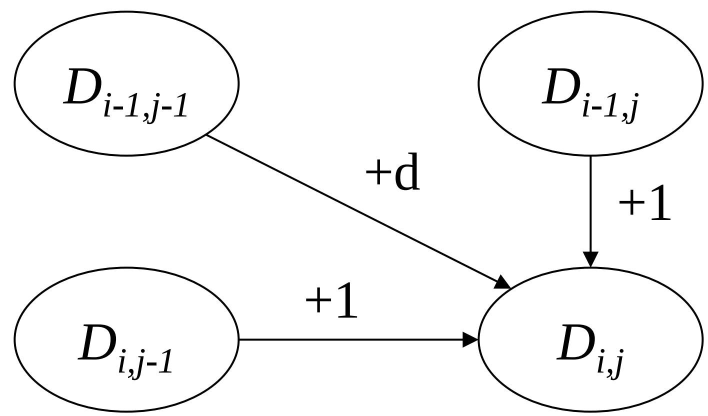
3. 插入b_n : $D_{m,n} = D_{m,n\text{-}1} + 1$

编辑距离的计算

递推关系, $m,n \geq 1$:

$$D_{m,n} = \min \begin{Bmatrix} D_{m-1,n-1} & + & c(a_m, b_n), \\ D_{m-1,n} & + & 1, \\ D_{m,n-1} & + & 1 \end{Bmatrix}$$

$$c(a,b) = \begin{cases} 1 & \text{if } a \neq b \\ 0 & \text{if } a = b \end{cases}$$



→ 需要计算所有$D_{i,j}$, $0 \leq i \leq m$, $0 \leq j \leq n$.

编辑距离的递推方程

基本情况:

$$D_{0,0} = D(\varepsilon, \varepsilon) = 0$$

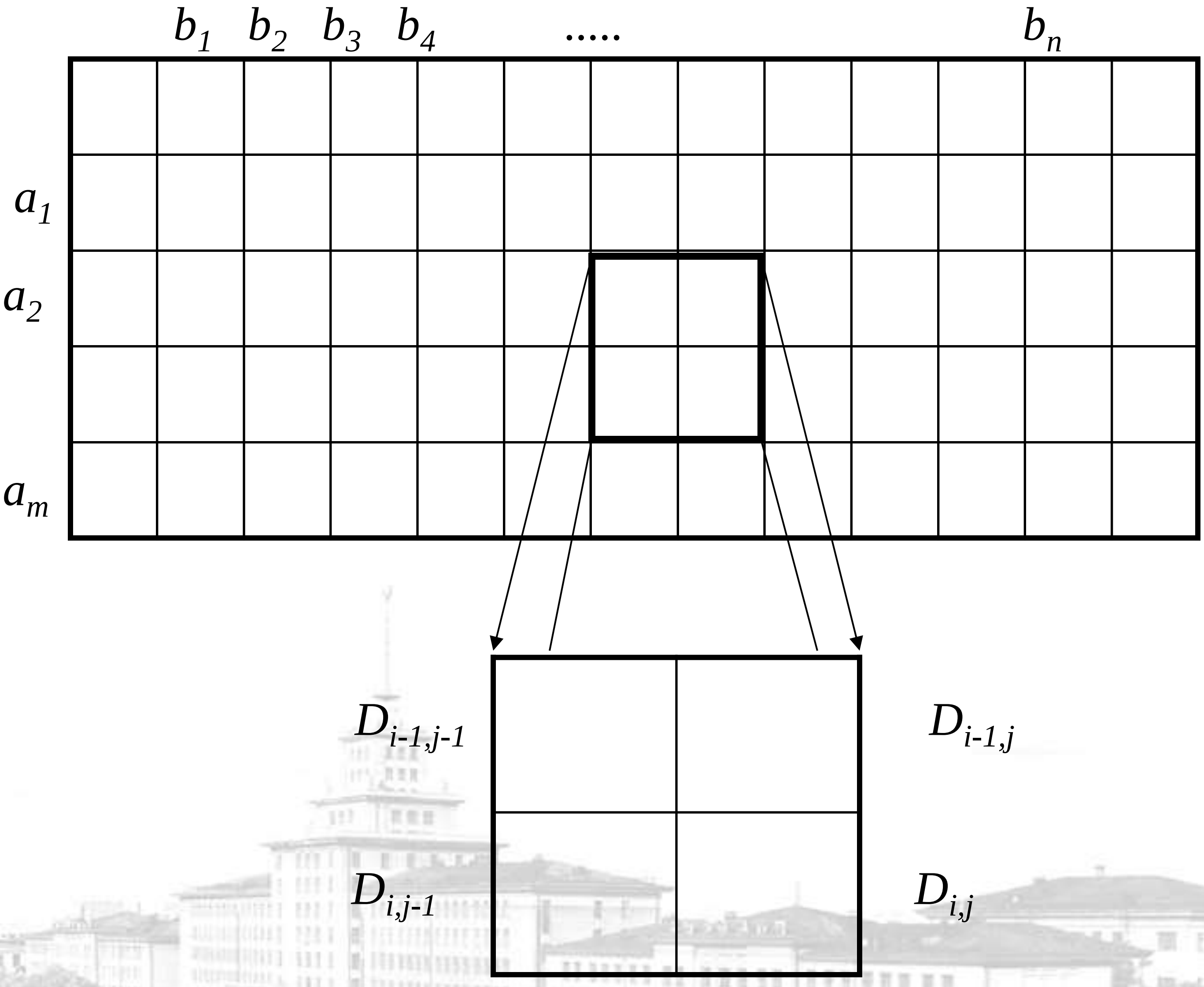
$$D_{0,j} = D(\varepsilon, B_j) = j$$

$$D_{i,0} = D(A_i, \varepsilon) = i$$

递推方程

$$D_{i,j} = \min\begin{Bmatrix} D_{i-1,j-1} & + & c(a_i, b_j) \\ D_{i-1,j} & + & 1, \\ D_{i,j-1} & + & 1 \end{Bmatrix}$$

编辑距离计算顺序



编辑距离算法

算法 edit_distance

输入: 字符串 $A = a_1 \ldots. a_m$ 和 $B = b_1 \ldots b_n$

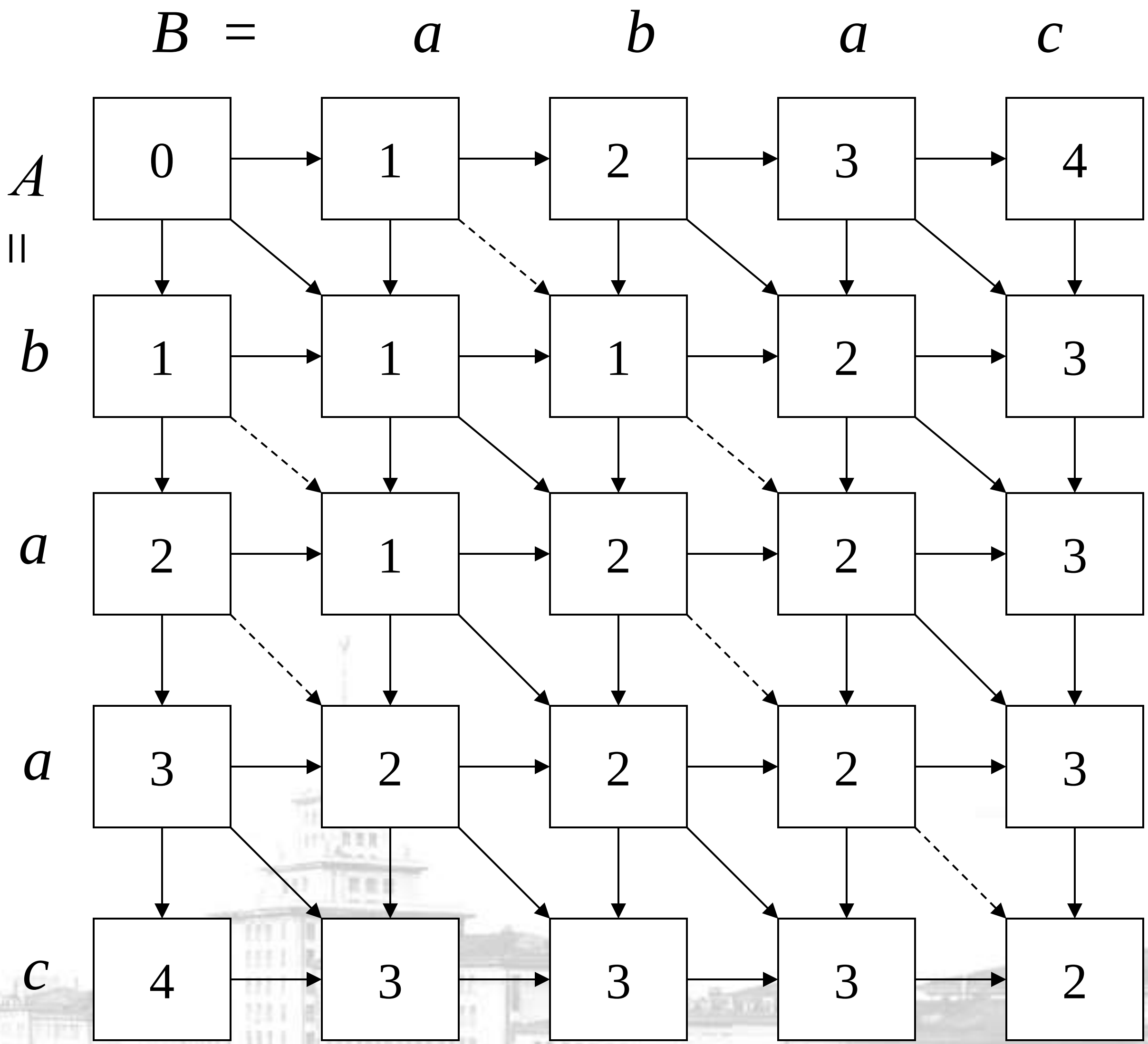
输出: 矩阵 $D = (D_{ij})$

```
D[0,0] := 0
for i := 1 to m do D[i,0] = i
for j := 1 to n do D[0,j] = j
for i := 1 to m do
    for j := 1 to n do
        D[i,j] := min( D[i - 1,j] + 1,
                       D[i,j - 1] + 1,
                       D[i –1, j – 1] + c(a_i,b_j))
```

例子

		a	b	a	c
	0	1	2	3	4
b	1				
a	2				
a	3				
c	4				

恢复编辑操作图



编辑操作的计算

算法 edit_operations (*i*,*j*)

输入: 矩阵 *D*

```
if i = 0 and j = 0 then return
if i ≠ 0 and D[i,j] = D[i – 1 , j] + 1
  then  "delete a[i]"
          edit_operations (i – 1, j)
else if j ≠ 0 and D[i,j] = D[i, j – 1] + 1
   then "insert b[j]"
          edit_operations (i, j – 1)
else
  /* D[i,j] = D[i – 1, j – 1 ] + c(a[i], b[j])  */
        "replace a[i] by b[j] "
        edit_operations (i – 1, j – 1)
```


Initial call: edit_operations(*m*,*n*)