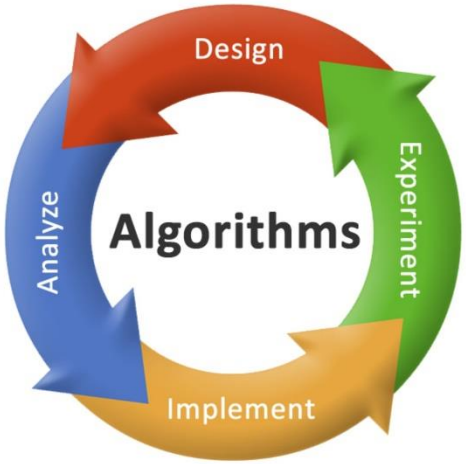


华中科技大学
HUAZHONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Algorithm Design and Analysis

算法设计与分析

Chapter 5: Dynamic Programming ▌张乾坤

课程提要

- 加权区间调度（Weighted interval scheduling）
- 分段最小二乘（Segmented least squares）
- 背包问题（Knapsack problem）
- 字符串对齐（Sequence alignment）
- RNA二级结构（RNA secondary structure）

算法范式

贪心：按照一定的顺序处理输入的信息，目光短浅地做出不可逆转的决定。

分治：将一个问题分解成独立的子问题。解决每个子问题；将子问题的解决方案结合起来，形成对原问题的解决方案。

动态规划：将一个问题分解成一系列重叠的子问题；将较小的子问题的解决方案结合起来，形成大的子问题的解决方案。

动态规划历史

Bellman：在20世纪50年代开创了对动态规划的系统研究。

Etymology(词源)：

- 动态规划：随着时间的推移进行规划。
- 当时的国防部长对数学研究有病态的恐惧。
- Bellman寻求一个 “dynamic "的形容词以避免冲突。

THE THEORY OF DYNAMIC PROGRAMMING

RICHARD BELLMAN

1. **Introduction.** Before turning to a discussion of some representative problems which will permit us to exhibit various mathematical features of the theory, let us present a brief survey of the fundamental concepts, hopes, and aspirations of dynamic programming.

To begin with, the theory was created to treat the mathematical problems arising from the study of various multi-stage decision processes, which may roughly be described in the following way: We have a physical system whose state at any time t is determined by a set of quantities which we call state parameters, or state variables. At certain times, which may be prescribed in advance, or which may be determined by the process itself, we are called upon to make decisions which will affect the state of the system. These decisions are equivalent to transformations of the state variables, the choice of a decision being identical with the choice of a transformation. The outcome of the preceding decisions is to be used to guide the choice of future ones, with the purpose of the whole process that of maximizing some function of the parameters describing the final state.

Examples of processes fitting this loose description are furnished by virtually every phase of modern life, from the planning of industrial production lines to the scheduling of patients at a medical clinic; from the determination of long-term investment programs for universities to the determination of a replacement policy for machinery in factories; from the programming of training policies for skilled and unskilled labor to the choice of optimal purchasing and inventory policies for department stores and military establishments.

动态规划应用

应用范围:

- 计算机科学: AI, compilers, systems, graphics, theory.....
- Operations research(运筹学)
- Information theory(信息论)
- Control theory(控制理论)
- Bioinformatics(生物信息学)

一些著名的动态规划算法:

- Avidan-Shamir用于接缝雕刻
- 用于比较两个文件的Unix diff
- 隐马尔科夫模型的Viterbi
- 评估花键曲线的De Boor
- 最短路径的Bellman-Ford-Moore
- Cocke-Kasami-Younger用于解析无上下文语法
- Needleman-Wunsch/Smith-Waterman用于序列对齐

动态规划书籍

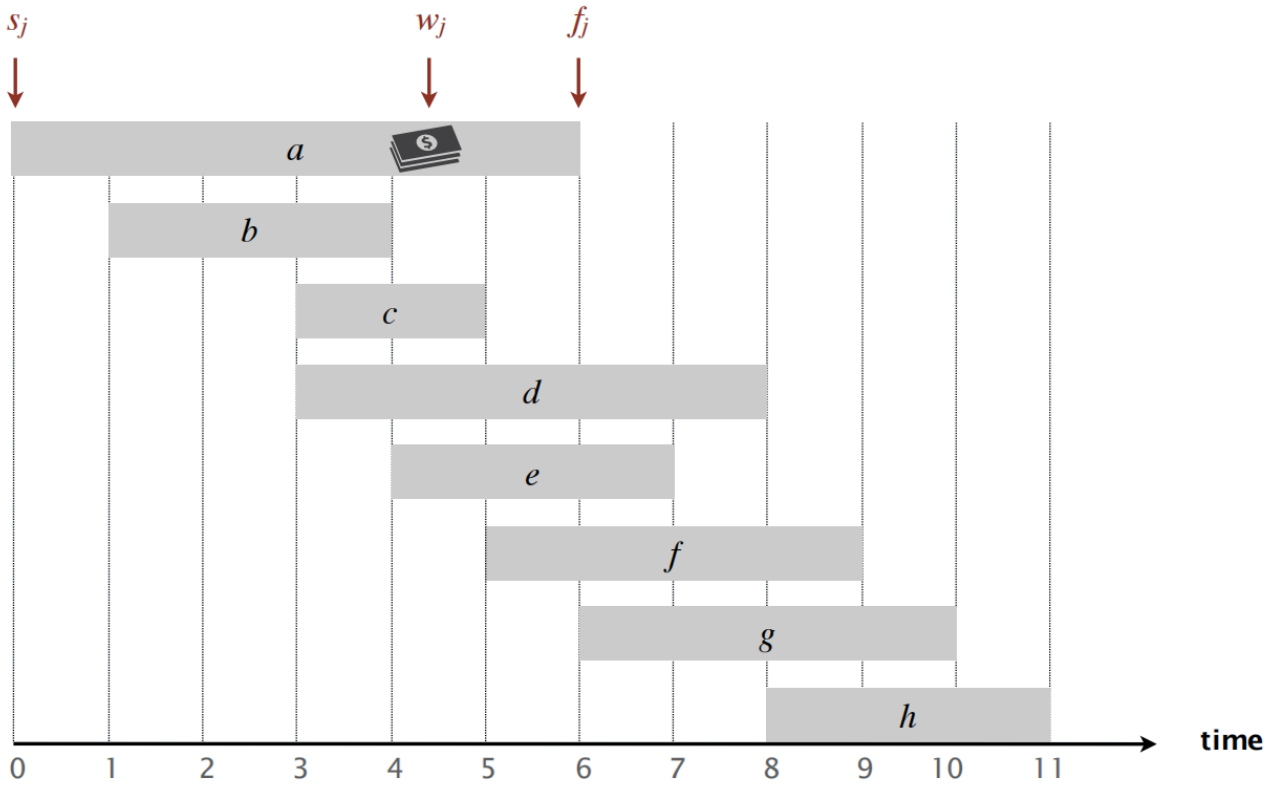

Richard Bellman
Dynamic Programming
Adaptive Dynamic Programming for Control
Algorithms and Stability
Dynamic Programming: Sequential Decision Making
Williams, Kenneth
Markov Decision Processes
Discrete Stochastic Dynamic Programming
MARTIN L. PUTERMAN
Dynamic Programming
Foundations and Principles
Second Edition
Moshe Sniedovich
Approximate Dynamic Programming
Warren B. Powell
DYNAMIC PROGRAMMING FOR CODING INTERVIEWS
MEENAKSHI & KAMAL RAWAT
Applied Dynamic Programming for Optimization of Dynamical Systems
Introduction to Dynamic Programming
Leon Cooper and Mary W Cooper
Approximate Dynamic Programming
Dynamic Programming with Management Applications (Operational Research)
Hastings, N.A.J.
HANDBOOK OF LEARNING AND APPROXIMATE DYNAMIC PROGRAMMING
Dynamic Programming and Optimal Control
The Art and Theory of Dynamic Programming
DYNAMIC PROGRAMMING
Models and Applications
Eric V. Denardo
Approximate Dynamic Programming for Dynamic Vehicle Routing
Stochastic Optimization in Insurance
A Dynamic Programming Approach
Dynamic Programming and Markov Processes
Howard
ITERATIVE DYNAMIC PROGRAMMING
REN LUUS
NEURO-DYNAMIC PROGRAMMING
DIMITRI P. BERTSEKAS
JOHN N. TSITSIKLIS

课程提要

- 加权区间调度（Weighted interval scheduling）
- 分段最小二乘（Segmented least squares）
- 背包问题（Knapsack problem）
- 字符串对齐（Sequence alignment）
- RNA二级结构（RNA secondary structure）

加权区间调度

- 任务 j 从 s_j开始，在 f_j 结束，并且权重 $w_j > 0$
- 不重叠的两个任务是兼容的
- 目标：找到相互兼容的任务的最大权重子集



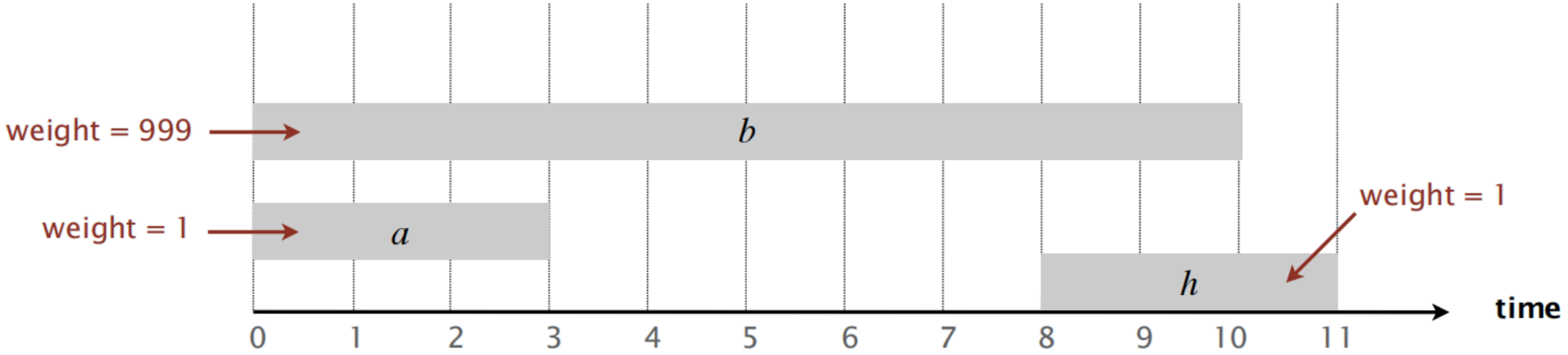
最早完成时间算法

最早完成时间：

- 按完成时间的升序考虑任务。
- 如果与先前选择的任务兼容，则将任务添加到子集中。

回顾：如果所有权重都为 1，则贪心算法是正确的。

观察：加权版本的贪心算法十分失败



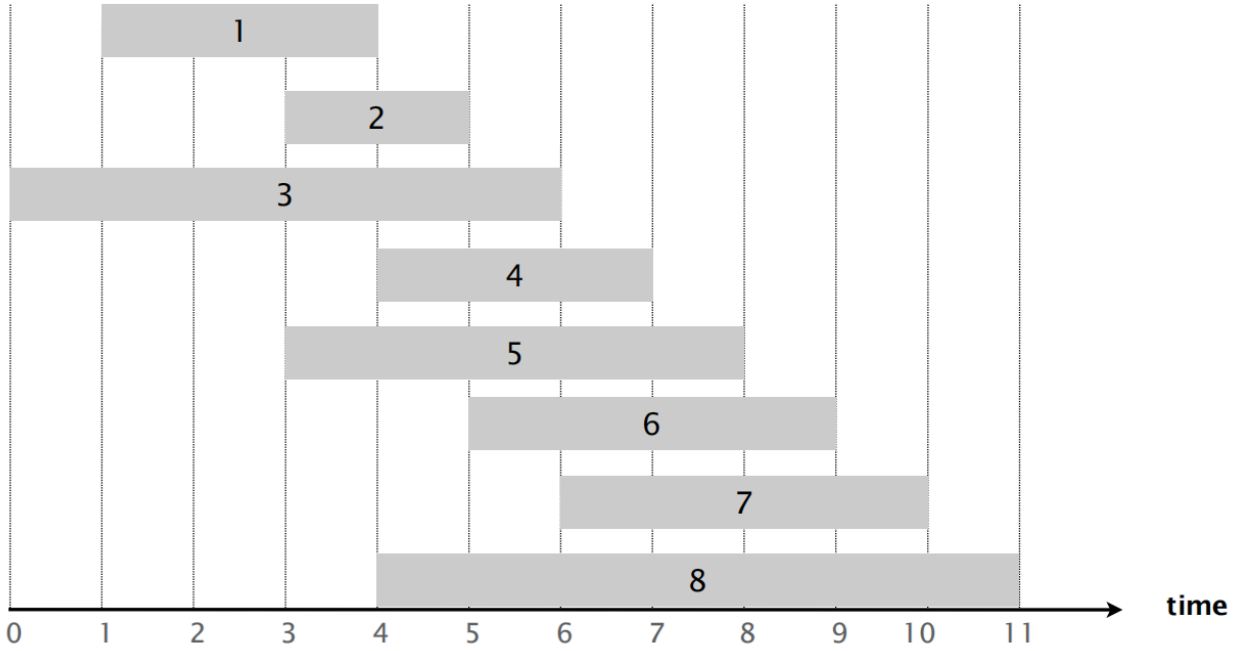
加权区间调度

约定：任务按完成时间升序排列：$f_1 \leq f_2 \leq \ldots \leq f_n$

定义：$p(j)$= 最大索引 $i < j$ 使得 job i 和 j 兼容

Ex：$p(8) = 1, p(7) = 3, p(2) = 0$

i 是在 j 开始之前的最右边的区间



动态规划：二元选择

定义：$OPT(j)$ = 由任务 $1,2,\ldots,j$ 组成的子任务的最大权重（前j个任务的最优解）。

目标：$OPT(n)$ = 任何相互兼容的任务的子集的最大权重。

情况 1：$OPT(j)$ 不选择任务 j 。

- 对于由剩余的任务 $1,2,\ldots,j-1$ 组成的问题来说，必须是一个最佳的解决方案

情况 2：$OPT(j)$ 选择任务 j 。

- 获得收益w_j。
- 不能使用不兼容的任务$\{p(j)+1, p(j)+2, \ldots, j-1\}$.
- 对于由剩余的任务 $1,2,\ldots,p(j)$ 组成的问题来说，必须包括最佳的解决方案

Bellman 等式：$$OPT(j) = \begin{cases} 0 & if\ j = 0 \\ max\{OPT(j-1), w_j + OPT(p(j))\} & if\ j > 0 \end{cases}$$

动态规划：暴力

BRUTE-FORCE $(n, s_1, \ldots, s_n, f_1, \ldots, f_n, w_1, \ldots, w_n)$

Sort jobs by finish time and renumber so that $f_1 \leq f_2 \leq \ldots \leq f_n$.

Compute $p[1], p[2], \ldots, p[n]$ via binary search.

RETURN COMPUTE-OPT(n).

COMPUTE-OPT(j)

IF $(j = 0)$

  RETURN 0.

ELSE

  RETURN max {COMPUTE-OPT$(j-1)$, w_j + COMPUTE-OPT$(p[j])$ }.

动态规划：问题 1

COMPUTE-OPT(n)最差情况的时间复杂度是多少?

A. $\Theta(n \log n)$

B. $\Theta(n^2)$

C. $\Theta(1.618^n)$

D. $\Theta(2^n)$

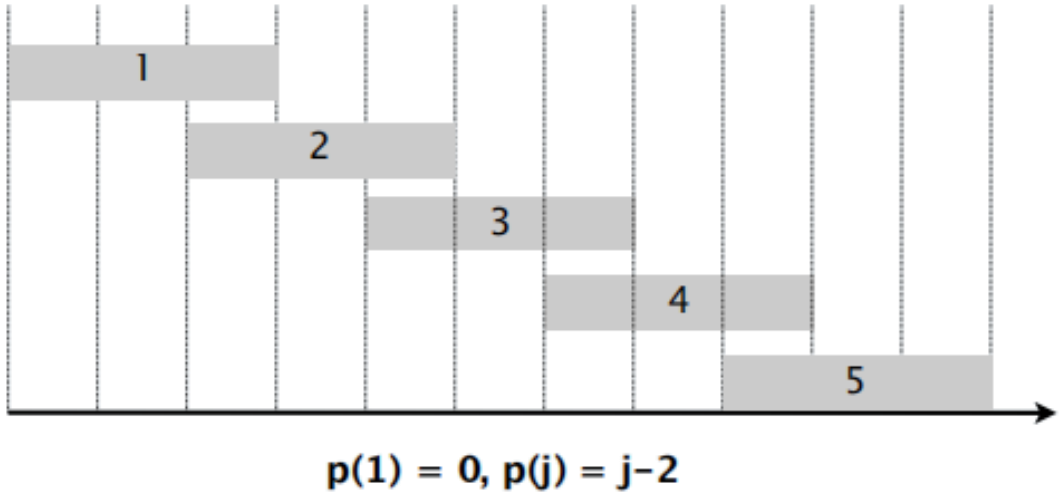
```
COMPUTE-OPT(j)
---------------------------------------------------
IF (j = 0)
   RETURN 0.
ELSE
   RETURN max {COMPUTE-OPT(j – 1), w_j + COMPUTE-OPT(p[j]) }.
```

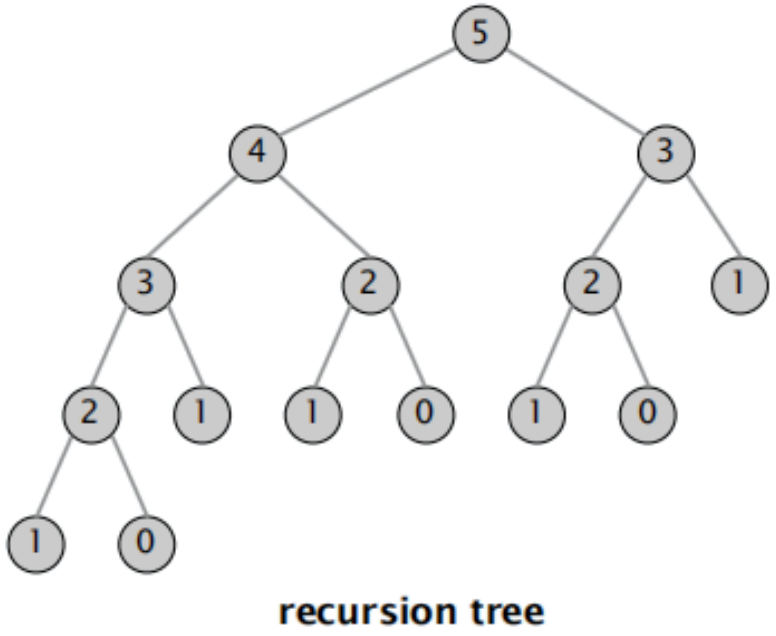
加权区间调度：暴力

观察：递归算法的速度非常慢，因为：

重叠子问题⇒指数时间算法。

Ex：“分层”实例族的递归调用的数量增长像斐波纳契序列一样。





加权区间调度：记忆法

自顶而下动态规划（备忘录）：

- 暂存子问题j的结果在$M[j]$中。
- 使用$M[j]$以避免需要多次解决子问题j。

TOP-DOWN($n, s_1, \ldots, s_n, f_1, \ldots, f_n, w_1, \ldots, w_n$)

Sort jobs by finish time and renumber so that $f_1 \le f_2 \le \ldots \le f_n$.

Compute $p[1], p[2], \ldots, p[n]$ via binary search.

$M[0] \leftarrow 0$. ← global array

RETURN M-COMPUTE-OPT(n).

M-COMPUTE-OPT(j)

IF ($M[j]$ is uninitialized)

$M[j] \leftarrow \max \{$ M-COMPUTE-OPT $(j-1)$, w_j + M-COMPUTE-OPT($p[j]$) $\}$.

RETURN $M[j]$.

加权区间调度：运行时间

结论：算法的记忆版本时间复杂度为 $O(n\ log\ n)$ 。

证明：

- 排序的完成时间：使用归并排序， $O(n\ log\ n)$ 。
- Compute p[j] for each j：使用二分搜索， $O(n\ log\ n)$ 。
- M-COMPUTE-OPT(n) 的总体运行时间是 $O(n)$。

Those who cannot remember the past are condemned to repeat it.

-Dynamic Programming

加权区间调度：寻找解决方案

Q：DP算法可以计算出最优值。如何找到最优解法？

A：通过调用FIND-SOLUTION(n)进行第二遍计算。

FIND-SOLUTION(j)

IF ($j = 0$)

  RETURN $\varnothing$.

ELSE IF ($w_j + M[p[j]] > M[j-1]$)

  RETURN $\{ j \} \cup$ FIND-SOLUTION($p[j]$).

ELSE

  RETURN FIND-SOLUTION($j-1$).

$$M[j] = \max \{ M[j-1],\ w_j + M[p[j]] \}.$$

分析：递归调用次数$\leq n \Rightarrow O(n)$.

加权区间调度：自底而上动态规划

自底而上动态规划：解除递归

BOTTOM-UP($n, s_1, \ldots, s_n, f_1, \ldots, f_n, w_1, \ldots, w_n$)

Sort jobs by finish time and renumber so that $f_1 \leq f_2 \leq \ldots \leq f_n$.

Compute $p[1], p[2], \ldots, p[n]$.

$M[0] \leftarrow 0$.

previously computed values

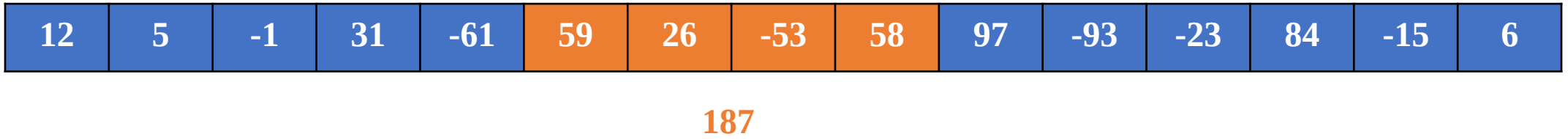
FOR $j = 1$ TO n

  $M[j] \leftarrow \max \{ M[j-1],\ w_j + M[p[j]] \}$.

运行时间：自底而上版本具有 $O(n \log n)$ 时间复杂度。

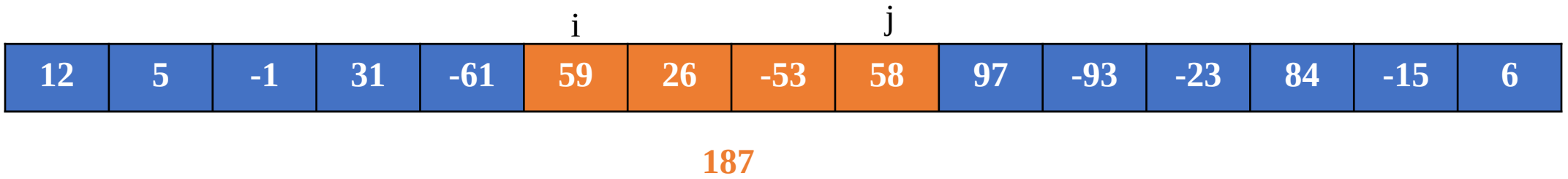
最大子数组问题

目标：给定含有n个整数(正或负)的数组x，找出和最大的邻接子数组。



最大子数组问题

Goal. 给定含有n个整数(正或负)的数组x，找出和最大的邻接子数组。



暴力算法:

- 对于每个 i 和 j : 计算 $a[i] + a[i+1] + \ldots + a[j]$.
- 时间复杂度为 $\Theta(n^3)$.

使用“累加和”技巧

- 提前计算累加和: $S[i] = a[0] + a[1] + \ldots + a[i]$.
- $a[i] + a[i+1] + \ldots + a[j] = S[j] - S[i-1]$.
- 时间复杂度优化为 $\Theta(n^2)$.

KADANE 算法

定义：$OPT(i)$ = 任意最右侧下标为 i 的子数组的最大和。

目标：$\max_i OPT(i)$

Bellman 等式：$$OPT(j) = \begin{cases} 0 & if\ j = 0 \\ max\{ x_i, x_i + OPT(i-1) \}\ if\ j > 0 \end{cases}$$

运行时间：$O(n)$.

只取元素i

取元素 i 和以索引 $i-1$ 结束的最佳子数组

课程提要

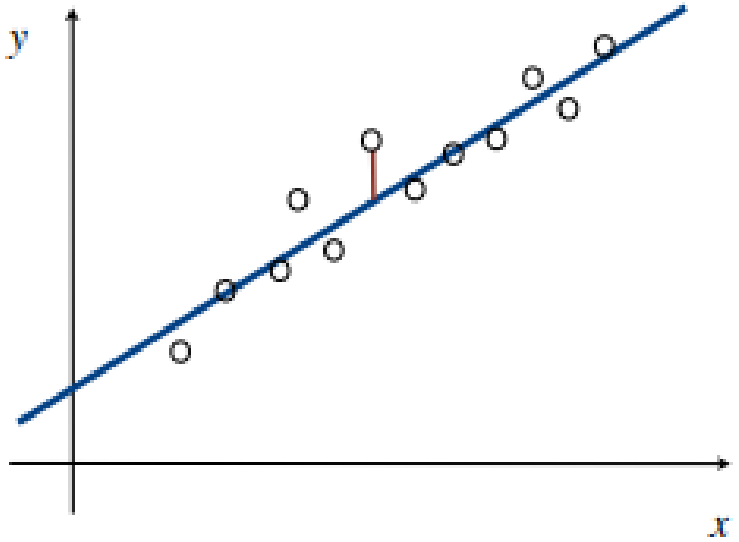
- 加权区间调度（Weighted interval scheduling）
- 分段最小二乘（Segmented least squares）
- 背包问题（Knapsack problem）
- RNA二级结构（RNA secondary structure）

最小二乘

最小二乘：　统计学中的基础问题

- 给定平面上的 n 个点：$(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$。
- 求一条线 $y = ax + b$，使平方误差之和最小：

$$SSE = \sum_{i=1}^{n} (y_i - ax_i - b)^2$$



解决方案：　当满足下式时达到最小误差：

$$a = \frac{n\sum_i x_i y_i - (\sum_i x_i)(\sum_i y_i)}{n\sum_i x_i^2 - (\sum_i x_i)^2}, \quad b = \frac{\sum_i y_i - a\sum_i x_i}{n}$$

分段最小二乘

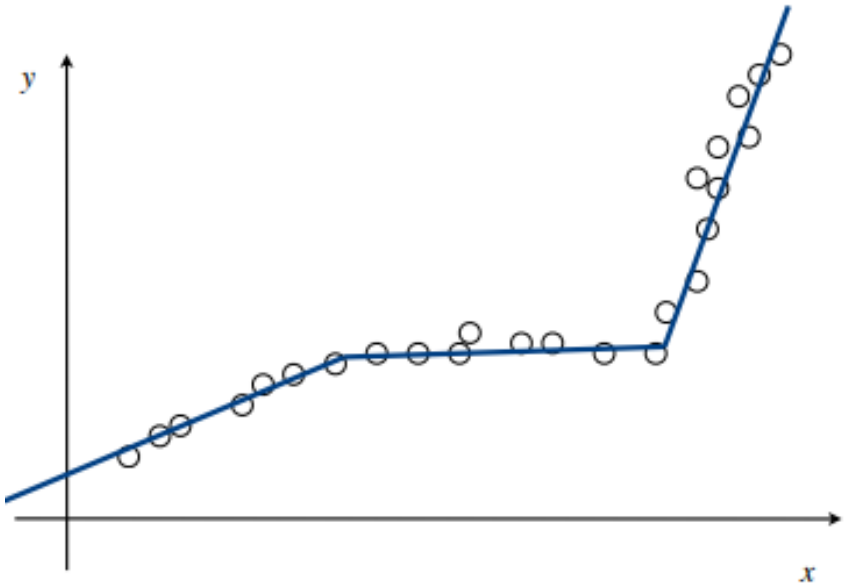
分段最小二乘:

- 点大致位于几条线段的序列上。
- 给定平面上的 n 个点: $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$且 $x_1 < x_2 < \ldots < x_n$, 找到使 $f(x)$ 最小化的线的序列。

Q.: 为了平衡accuracy(准确性)和parsimony(简约性), $f(x)$的合理选择是什么?

拟合的精度

线段数量



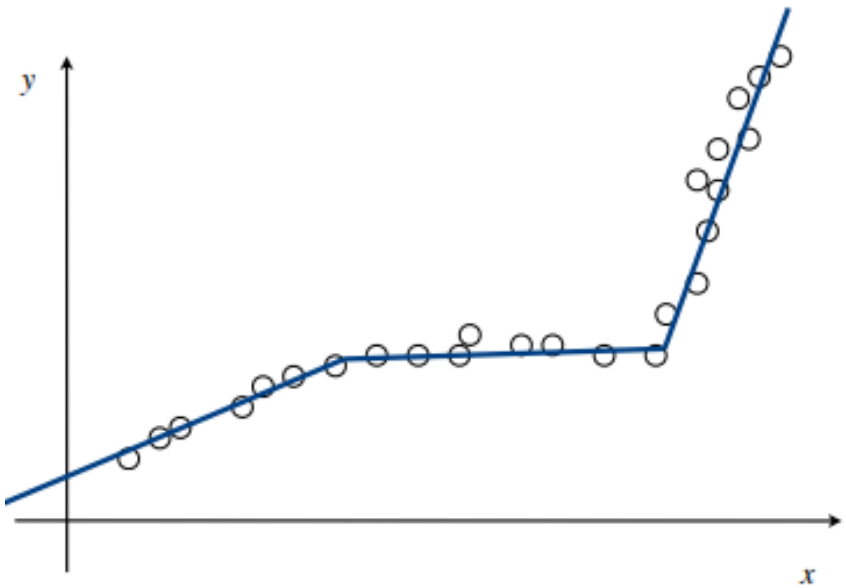
分段最小二乘

分段最小二乘:

- 点大致位于几条线段的序列上。
- 给定平面上的 n 个点: $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$且 $x_1 < x_2 < \ldots < x_n$，找到使 $f(x)$ 最小化的线的序列。

目标: 对于某个常数 $c > 0$，最小化 $f(x) = E + cL$，其中

- E = 每个段的SSE的总和。
- L = 线的数量。



动态规划：多路选择

记号：

- $OPT(j)$ = 点 $p_1, p_2, \ldots, p_j$ 的最小代价
- e_{ij} = 点 $p_i, p_{i+1}, \ldots, p_j$ 的SSE

计算OPT(j):

- 对于某些 $i \leq j$，假设最后一段使用 $p_i, p_{i+1}, \ldots, p_j$ 。
- Cost = $e_{ij} + c + OPT(i-1)$ 。 ← 最优子结构性质

Bellman 等式：

$$OPT(j) = \begin{cases} 0 & if\ j = 0 \\ \min\limits_{1 \leq i \leq j} \{e_{ij} + c + OPT(i-1)\} & if\ j > 0 \end{cases}$$

分段最小二乘

SEGMENTED-LEAST-SQUARES($n, p_1, \ldots, p_n, c$)

FOR j = 1 TO n

  FOR i = 1 TO j

    Compute the SSE e_{ij} for the points $p_i, p_{i+1}, \ldots, p_j$.

$M[0] \leftarrow 0$.

FOR $j = 1$ TO n

  $M[j] \leftarrow \min_{1 \le i \le j} \{ e_{ij} + c + M[i-1] \}$.

previously computed value

RETURN $M[n]$.

分段最小二乘分析

定理：[Bellman 1961] DP 算法求解分段最小二乘问题需要$O(n^3)$时间。

证明：

- 时间瓶颈: 计算每个 i 和 j 的SSE e_{ij} 。

$$a_{ij} = \frac{n\sum_k x_k y_k - (\sum_k x_k)(\sum_k y_k)}{n\sum_k x_k^2 - (\sum_k x_k)^2}, \quad b_{ij} = \frac{\sum_k y_k - a_{ij}\sum_k x_k}{n}$$

- 计算单个 e_{ij} 的时间复杂度为$O(n)$ 。

Q：还可以改进吗？

课程提要

- 加权区间调度（Weighted interval scheduling）
- 分段最小二乘（Segmented least squares）
- 背包问题（Knapsack problem）
- 字符串对齐（Sequence alignment）
- RNA二级结构（RNA secondary structure）

背包问题

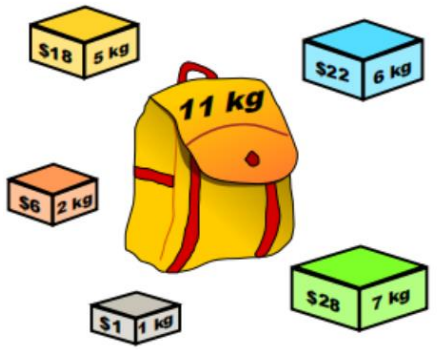
目标：背包问题的目的是最大化所带物品的总价格。

- 有 n 个物品：物品 i 的价格 $v_i > 0$，重量 $w_i > 0$。
- 一个子集的价格=子集内所有单个物品价格的总和。
- 背包有W的重量限制。

Ex：子集 $\{1, 2, 5\}$ 的价格为 35 美元（重量为 10kg）。

Ex：子集 $\{3, 4\}$ 的价格为 40 美元（重量为 11kg）。

假设：所有的价格和重量都是整数。



i	v_i	w_i
1	$1	1 kg
2	$6	2 kg
3	$18	5 kg
4	$22	6 kg
5	$28	7 kg

weights and values can be arbitrary positive integers

knapsack instance
(weight limit W = 11)

动态规划：问题 2

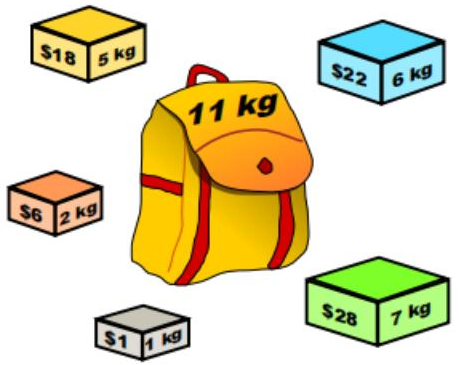
哪种算法可以解决背包问题?

A. Greedy-by-value：重复添加具有最大 v_i 的物品。

B. Greedy-by-weight：重复添加具有最小 w_i 的物品。

C. Greedy-by-ratio：重复添加具有最大比值 $\frac{v_i}{w_i}$的物品。

D. 以上都不是。



i	v_i	w_i
1	$1	1 kg
2	$6	2 kg
3	$18	5 kg
4	$22	6 kg
5	$28	7 kg

knapsack instance
(weight limit W = 11)

动态规划：问题 3

选择哪些子问题？

A. $OPT(w)$ = 重量限制为 w 的背包问题的最优值。

B. $OPT(i)$ = 具有物品 $1,\ldots,i$ 的背包问题的最优值。

C. $OPT(i,w)$ = 具有物品 $1,\ldots,i$，重量限制为 w 的背包问题的最优值。

D. 以上任何一项。

动态规划：两个变量

定义：$OPT(i,w)$ = 具有物品 $1,\ldots,i$，重量限制为 w 的背包问题的最优值。

目标：$OPT(n,W)$.

情况 1：$OPT(i,w)$ 不选择物品 i。（有几种可能？）

- 重量限制为 w，$OPT(i,w)$ 选择 $\{1,2,\ldots,i-1\}$ 中最优的物品。

情况 2：$OPT(i,w)$ 选择物品 i。

- 收集价格v_i。
- 新的重量限制 = $w - w_i$。
- 受新的重量限制，$OPT(i,w)$ 选择 $\{1,2,\ldots,i-1\}$ 中最优的物品。

$$OPT(i,w) = \begin{cases} 0 & \text{if } i = 0 \\ OPT(i-1,w) & \text{if } w_i > w \\ \max\{\, OPT(i-1,w),\ v_i + OPT(i-1,w-w_i)\,\} & \text{otherwise} \end{cases}$$

背包问题：自底向上动态规划

KNAPSACK$(n, W, w_1, \ldots, w_n, v_1, \ldots, v_n)$

FOR $w = 0$ TO W

  $M[0, w] \leftarrow 0$.

FOR $i = 1$ TO n

  FOR $w = 0$ TO W

    IF $(w_i > w)$ $M[i, w] \leftarrow M[i-1, w]$.

    ELSE $M[i, w] \leftarrow \max \{ M[i-1, w],\ v_i + M[i-1, w-w_i] \}$.

RETURN $M[n, W]$.

---

previously computed values

$$OPT(i, w) = \begin{cases} 0 & \text{if } i = 0 \\ OPT(i-1, w) & \text{if } w_i > w \\ \max\{ OPT(i-1, w),\ v_i + OPT(i-1, w-w_i) \} & \text{otherwise} \end{cases}$$

背包问题：自底向上动态规划演示

i	v_i	w_i
1	\$1	1 kg
2	\$6	2 kg
3	\$18	5 kg
4	\$22	6 kg
5	\$28	7 kg

$$OPT(i,w) = \begin{cases} 0 & \text{if } i = 0 \\ OPT(i-1,w) & \text{if } w_i > w \\ \max\{OPT(i-1,w),\ v_i + OPT(i-1,w-w_i)\} & \text{otherwise} \end{cases}$$

weight limit w

subset of items 1, ..., i	0	1	2	3	4	5	6	7	8	9	10	11
{ }	0	0	0	0	0	0	0	0	0	0	0	0
{ 1 }	0	1	1	1	1	1	1	1	1	1	1	1
{ 1, 2 }	0	1	6	7	7	7	7	7	7	7	7	7
{ 1, 2, 3 }	0	1	6	7	7	18	19	24	25	25	25	25
{ 1, 2, 3, 4 }	0	1	6	7	7	18	22	24	28	29	29	40
{ 1, 2, 3, 4, 5 }	0	1	6	7	7	18	22	28	29	34	35	40

OPT(i, w) = optimal value of knapsack problem with items 1, ..., i, subject to weight limit w

背包问题：运行时间

定理：

DP算法解决n个物品的背包问题需要 $\Theta(n\,W)$ 时间，其中W是最大重量。

证明：

- 每个表格条目花费 $O(1)$ 时间。
- 有$\Theta(n\,W)$ 个表条目。
- 计算出最优值后，可以追溯找到解决方案：

$OPT(i,w)$ 接受项目 i 当且仅当 $M\,[i,w]\;>\;M\,[i-1,w]$

注意：

- 算法严重依赖于重量是整数这个假设。
- 价格是整数的假设没有被使用。

动态规划：问题 4

背包问题是否存在多项式时间算法？

A. 是的，因为 DP 算法需要 $\Theta(n\,W)$ 时间。

B. 不，因为 $\Theta(n\,W)$ 不是一个关于输入尺寸的多项式函数。

C. 不，因为问题是 **NP**-hard的。

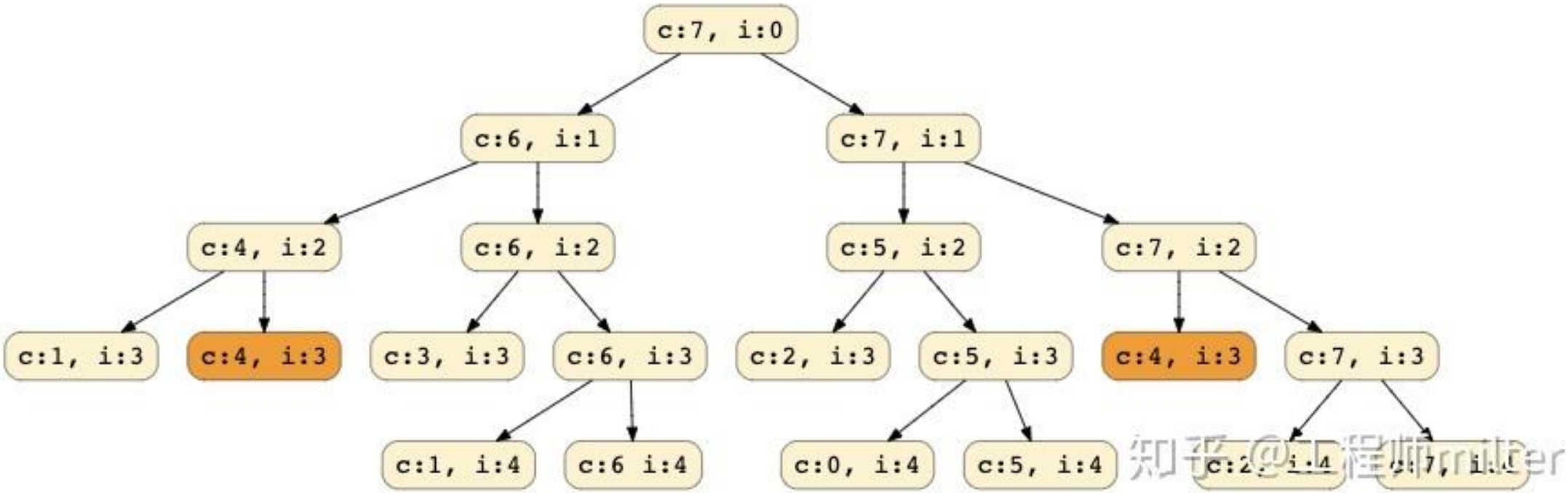
D. 未知。

小练习

- 背包问题的状态转移方程是什么?

包大小：7	A	B	C	D
价值	**1**	**6**	**10**	**16**
重量	**1**	**2**	**3**	**5**

包大小：7	A	B	C	D
价值	1	6	10	16
重量	1	2	3	5



包大小：7	A	B	C	D
价值	1	6	10	16
重量	1	2	3	5

capacity -->

profit []	weight []	index	0	1	2	3	4	5	6	7
1	1	0	0	1	1	1	1	1	1	1
6	2	1	0	1	6	7	7	7	7	7
10	3	2	0	1	6	10	11	16	17	17
16	5	3	0	1	6	10	11	16	17	22

课程提要

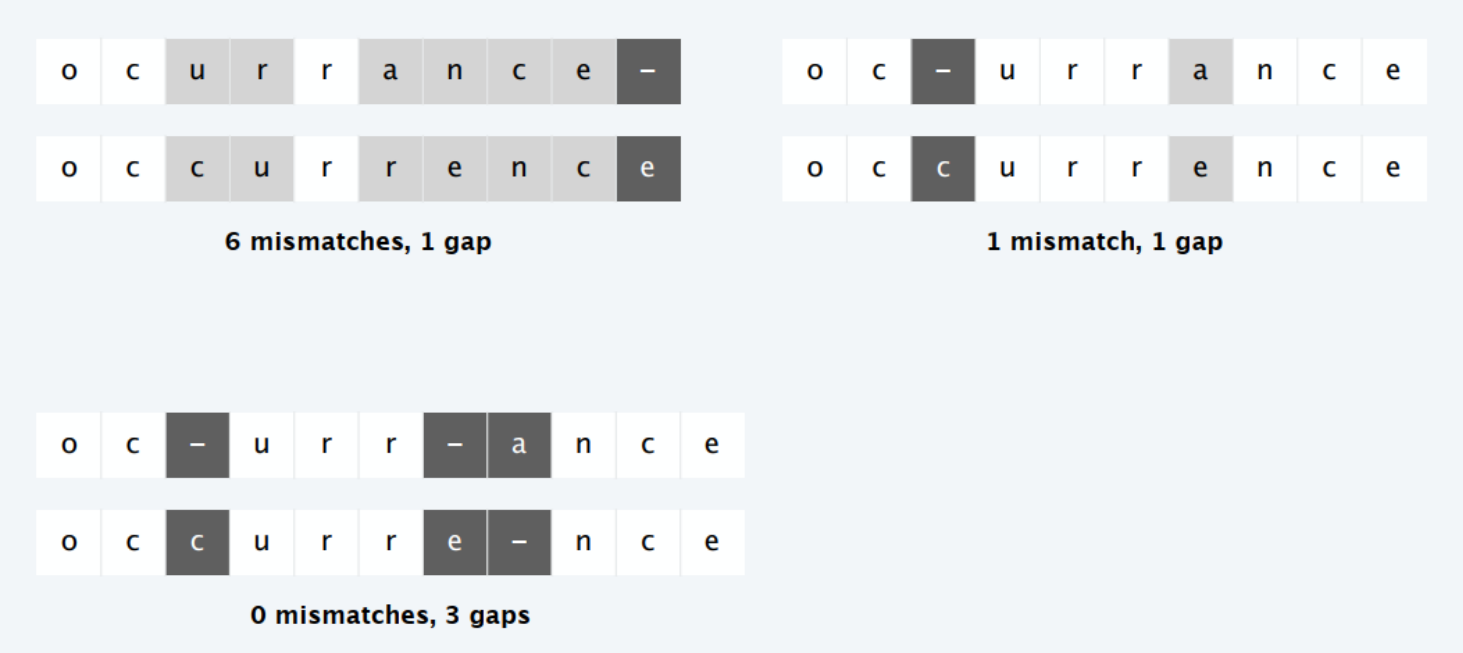
- 加权区间调度（Weighted interval scheduling）
- 分段最小二乘（Segmented least squares）
- 背包问题（Knapsack problem）
- 字符串对齐（Sequence alignment）
- RNA二级结构（RNA secondary structure）

字符串相似程度问题

Q：如何判断两个字符串的相似程度

Ex：ocurrance 和 occurrence

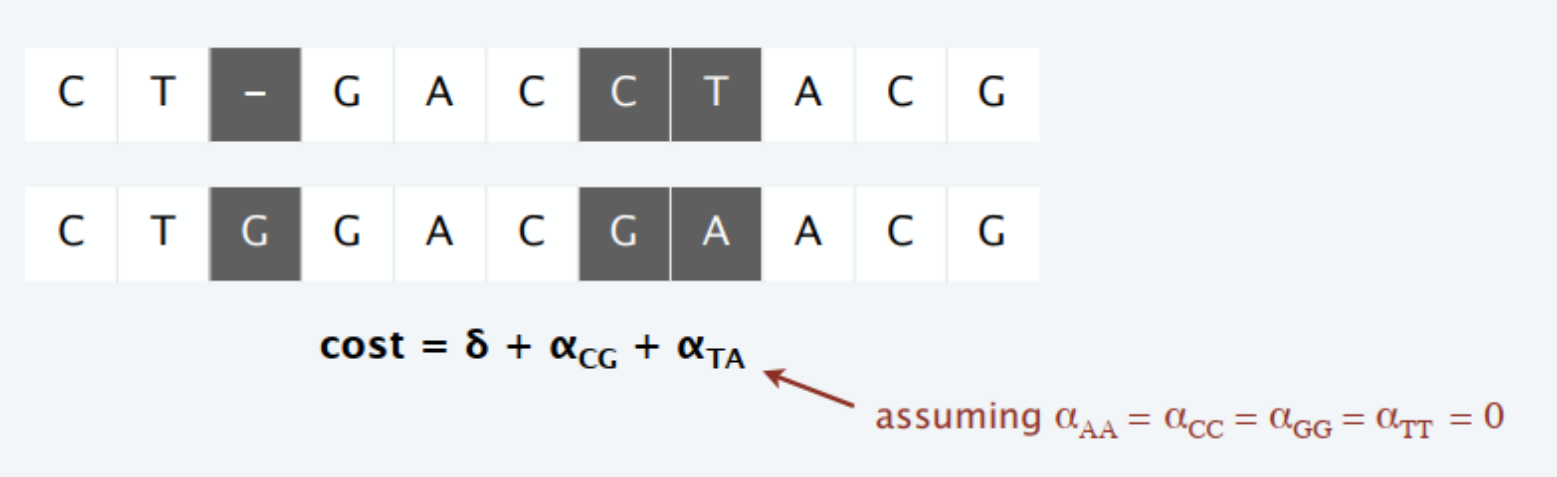
匹配方式：字符一对一，结果可能为配对(match)/失配(mismatch)/跳过(gap)



编辑距离(Edit distance)

Edit distance. [Levenshtein 1966, Needleman-Wunsch 1970]

- 跳过惩罚 δ；失配惩罚 α_{pq}
- 代价(cost) = 所有跳过惩罚和失配惩罚之和
- 编辑距离：代价的最小值



编辑距离(Edit distance)

- 应用:
  - 生物信息学
  - 拼写矫正
  - 机器翻译
  - 语音识别
  - 信息提取
  - ……

```
Spokesperson confirms      senior government adviser was found
Spokesperson said      the senior             adviser was found
```

动态规划：问题5

假设：跳过惩罚=2，失配惩罚=1

以下两个字符串的编辑距离是多少

PALETTE

PALATE

A. 1　B. 2　C. 3　D. 4　E. 5

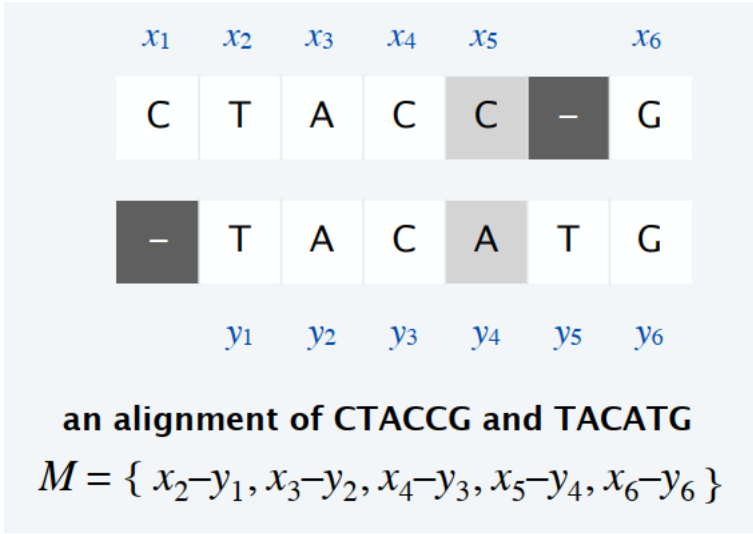
序列对齐(Sequence alignment)

目标：两个字符串 $x_1x_2 \dots x_m$ 和 $y_1y_2 \dots y_n$找到最小代价对齐方案。

定义：一个对齐方案 M 是有序配对 $x_i - y_j$ 的集合，集合保证每个字符至多出现在一个配对中且没有交叉。

（$x_i - y_j$ 和 $x_{i'} - y_{j'}$ 在 $i < i', j > j'$ 时交叉）

序列对齐(Sequence alignment)



定义：一个对齐方案 M 代价(cost) 为：

$$cost(M) = \underbrace{\sum_{(x_i,y_j)\in M} \alpha_{x_i y_j}}_{\text{mismatch}} + \underbrace{\sum_{i:x_i \text{ unmatched}} \delta + \sum_{j:y_j \text{ unmatched}} \delta}_{\text{gap}}$$

序列对齐(Sequence alignment)-问题解构

定义：$OPT(i,j)=$ 两个字符串的前缀 $x_1x_2\dots x_i$ 和 $y_1y_2\dots y_j$的**最小**对齐代价

目标：求 $OPT(m,n)$ （也即两个字符串的编辑距离/**最小**对齐代价）

情况1：$OPT(i,j)$ 配对了x_i-y_j；此时代价=x_i-y_j失配惩罚+$x_1x_2\dots x_{i-1}$ 和 $y_1y_2\dots y_{j-1}$的最小对齐代价

情况2a：$OPT(i,j)$ 留下 x_i 不匹配；此时代价=x_i 的跳过惩罚+$x_1x_2\dots x_{i-1}$ 和 $y_1y_2\dots y_j$的最小对齐代价

情况2b：$OPT(i,j)$ 留下 y_j 不匹配；此时代价=y_j 的跳过惩罚+$x_1x_2\dots x_i$ 和 $y_1y_2\dots y_{j-1}$的最小对齐代价

序列对齐(Sequence alignment)-问题解构

定义：$OPT(i,j)$ = 两个字符串的前缀 $x_1x_2 \dots x_i$ 和 $y_1y_2 \dots y_j$的**最小**对齐代价

目标：求 $OPT(m,n)$ （也即两个字符串的编辑距离/**最小**对齐代价）

Bellman equation.

$$OPT(i,j) = \begin{cases} j\delta & \text{if } i = 0 \\ i\delta & \text{if } j = 0 \\ \min \begin{cases} \alpha_{x_i y_j} + OPT(i-1, j-1) \\ \delta + OPT(i-1, j) \\ \delta + OPT(i, j-1) \end{cases} & \text{otherwise} \end{cases}$$

序列对齐(Sequence alignment)-自下而上算法

SEQUENCE-ALIGNMENT$(m, n, x_1, \ldots, x_m, y_1, \ldots, y_n, \delta, \alpha)$

FOR $i = 0$ TO m

  $M[i, 0] \leftarrow i\,\delta$.

FOR $j = 0$ TO n

  $M[0, j] \leftarrow j\,\delta$.

FOR i = 1 TO m

  FOR j = 1 TO n

    $M[i, j] \leftarrow \min \{ \alpha_{x_i y_j} + M[i-1, j-1],$

    $\delta + M[i-1, j],$

    $\delta + M[i, j-1] \}$.

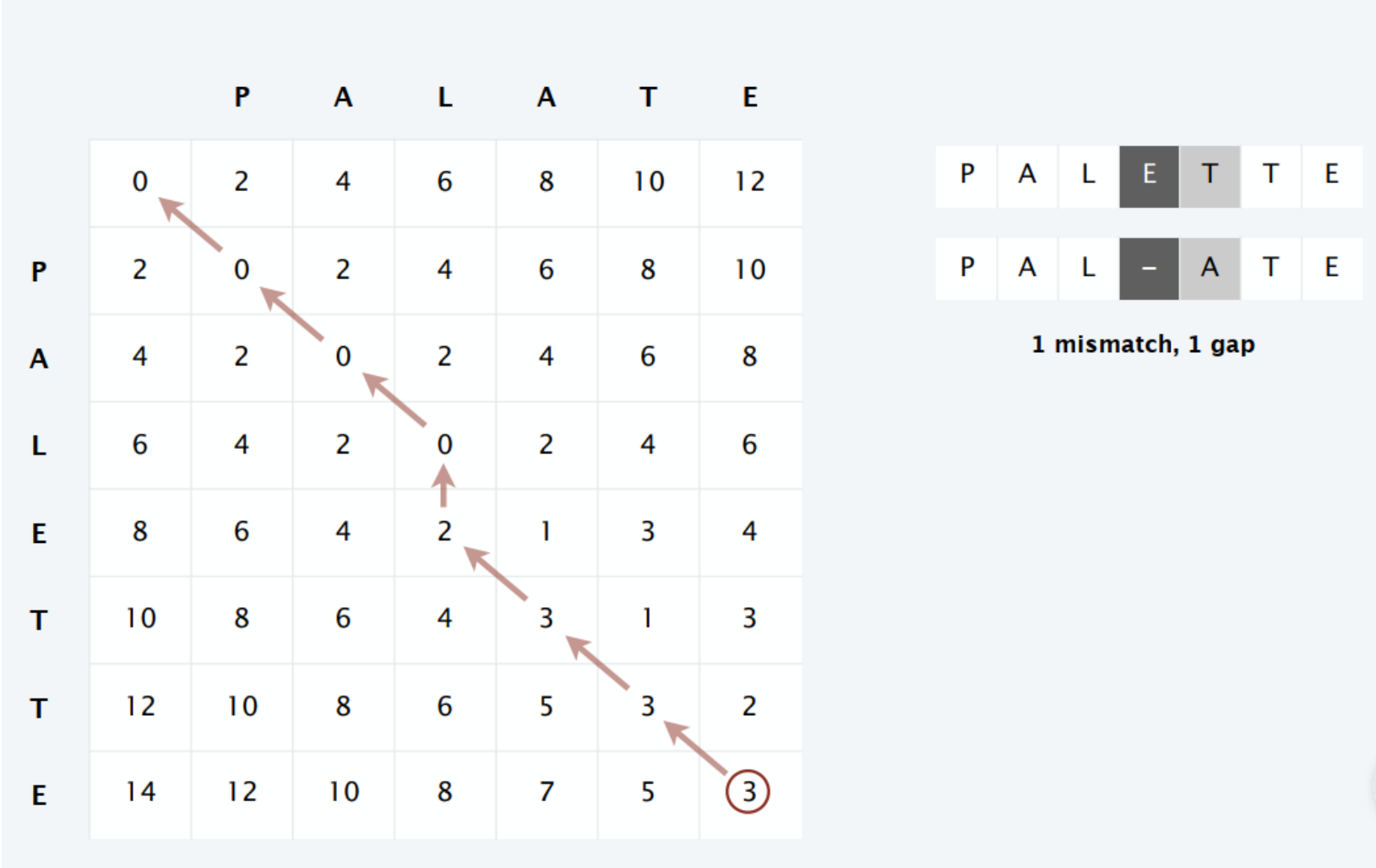
already computed

RETURN $M[m, n]$.

Bellman equation.

$$OPT(i,j) = \begin{cases} j\delta & \text{if } i = 0 \\ i\delta & \text{if } j = 0 \\ \min \begin{cases} \alpha_{x_i y_j} + OPT(i-1, j-1) \\ \delta + OPT(i-1, j) \\ \delta + OPT(i, j-1) \end{cases} & \text{otherwise} \end{cases}$$

序列对齐(Sequence alignment)-追溯路径 找到方案



		P	A	L	A	T	E
	0	2	4	6	8	10	12
P	2	0	2	4	6	8	10
A	4	2	0	2	4	6	8
L	6	4	2	0	2	4	6
E	8	6	4	2	1	3	4
T	10	8	6	4	3	1	3
T	12	10	8	6	5	3	2
E	14	12	10	8	7	5	3

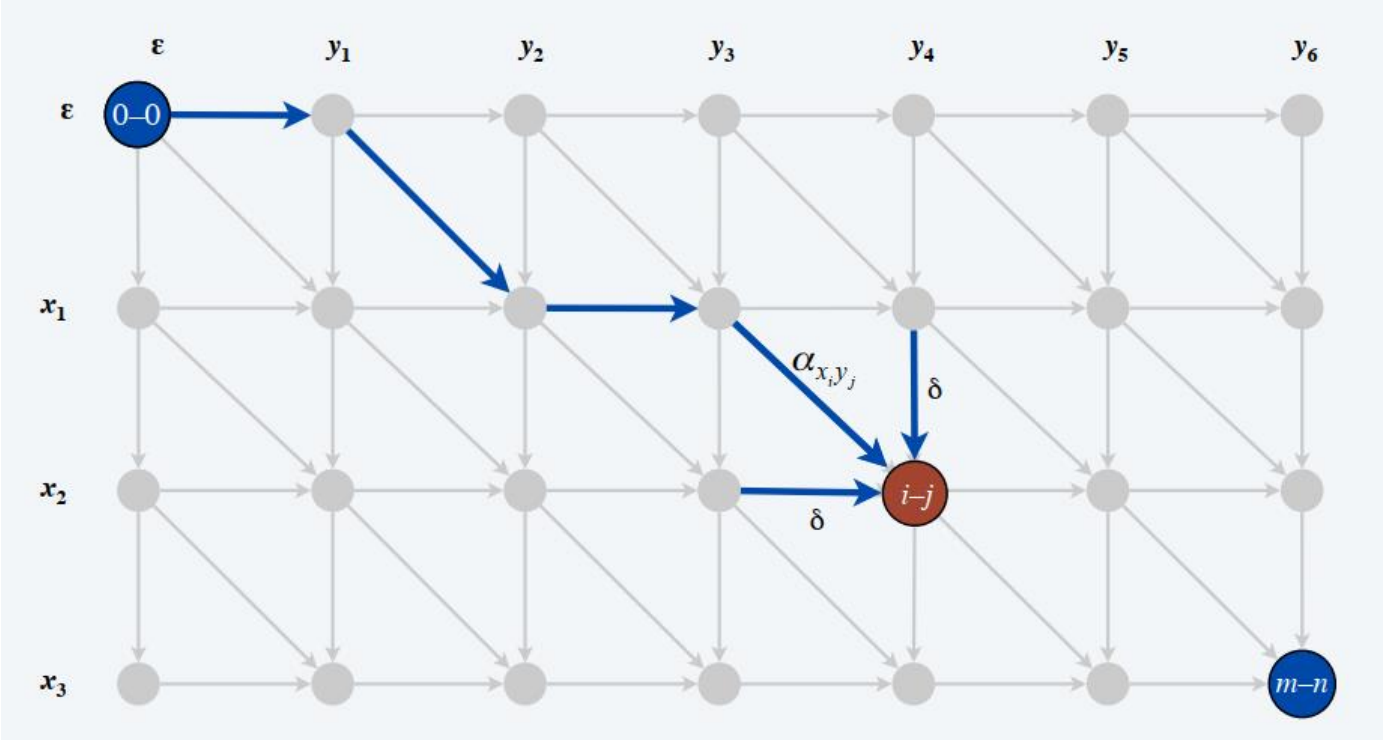
序列对齐(Sequence alignment)-分析

- 定理:
  - DP算法能够利用 $\Theta(mn)$ 的**时间**和**空间**计算两个长度分别为 m 和 n 的字符串的编辑距离和最优对齐方式

- 证明:
  - DP算法能够计算编辑距离
  - DP算法可以通过追溯（trace back）得到对应的最优对齐方式

一种额外的理解

- 将编辑距离的求解过程抽象成图
  - 令 $f(i,j)$ 表示从 $(0,0)$ 到 (i,j) 的最短路径长度
  - 引理：对于任意 i,j，有 $f(i,j) = OPT(i,j)$



一种额外的理解

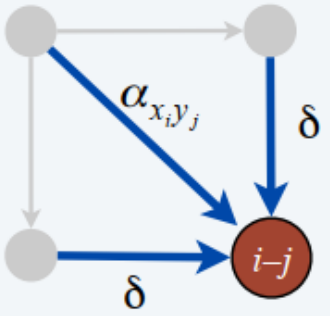
引理证明（数学归纳法）

- 基于：$f(0,0) = OPT(0,0) = 0$
- 归纳假设：假设引理对于满足 $i' + j' < i + j$ 的所有 (i', j') 成立
- 注意到：到 (i, j) 的最短路径的最后一条边一定来自于 $(i-1, j-1), (i-1, j)$或$(i, j-1)$
- 因此：

$$
\begin{aligned}
f(i,j) &= \min\{\alpha_{x_i y_j} + f(i-1,j-1),\ \delta + f(i-1,j),\ \delta + f(i,j-1)\} \\
&= \min\{\alpha_{x_i y_j} + OPT(i-1,j-1),\ \delta + OPT(i-1,j),\ \delta + OPT(i,j-1)\} \\
&= OPT(i,j) \ \blacksquare
\end{aligned}
$$

inductive hypothesis

Bellman equation



练习：最长公共子序列 (Longest Common Subsequence, LCS)

- 问题：给两个字符串 $x_1x_2 \dots x_m$ 和 $y_1y_2 \dots y_n$ 找到尽可能长的公共子序列
- 理解：删掉字符串 x 中的一些字符，删掉字符串 y 中的一些字符，同时保持剩下字符的相对位置不变，如果得到了两个相同的字符串，则这个相同的字符串是公共子序列
- 例如：LCS(GGCACCACG, ACGGCGGATACG) = GGCAACG.

练习：最长公共子序列 (Longest Common Subsequence, LCS)

- 解法1 动态规划
- 定义 $OPT(i,j)$ = 字符串前缀 $x_1x_2 \dots x_i$ 和 $y_1y_2 \dots y_j$ 的LCS长度
- 目标 $OPT(m,n)$
- 情况1 $x_i = y_i$
  - $x_1x_2 \dots x_{i-1}$ 和 $y_1y_2 \dots y_{j-1}$的LCS长度+1
- 情况2 $x_i \neq y_j$
  - 删掉 x_i： $x_1x_2 \dots x_{i-1}$ 和 $y_1y_2 \dots y_j$的LCS长度
  - 删掉 y_j： $x_1x_2 \dots x_i$ 和 $y_1y_2 \dots y_{j-1}$的LCS长度

Bellman equation:

$$OPT(i,j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ 1 + OPT(i-1, j-1) & \text{if } x_i = y_j \\ \max\{OPT(i-1, j), OPT(i, j-1)\} & \text{if } x_i \neq y_j \end{cases}$$

最长公共子序列(Longest Common Subsequence)

- 解法2 归纳为找到字符串 x 和 y 的最小代价对齐，其中
  - 跳过惩罚(gap penalty) $\delta = 1$
  - 失配惩罚(mismatch penalty)

$$\alpha_{pq} = \begin{cases} 0 & \text{if } p = q \\ \infty & \text{if } p \neq q \end{cases}$$

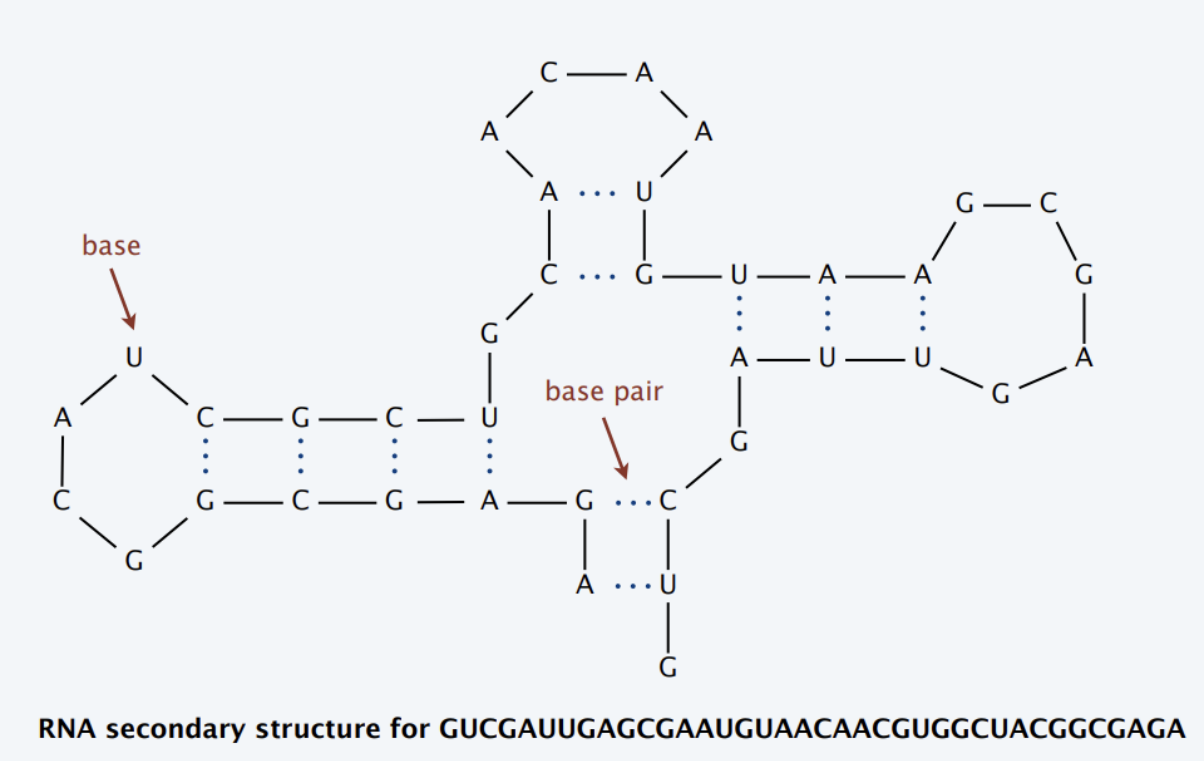
  - 编辑距离= 跳过的字符个数=字符串 x 和 y 中删去的字符个数
  - 求得的LCS长度=$(m + n - \text{edit distance})/2$

课程提要

- 加权区间调度（Weighted interval scheduling）
- 分段最小二乘（Segmented least squares）
- 背包问题（Knapsack problem）
- 字符串对齐（Sequence alignment）
- RNA二级结构（RNA secondary structure）

RNA二级结构

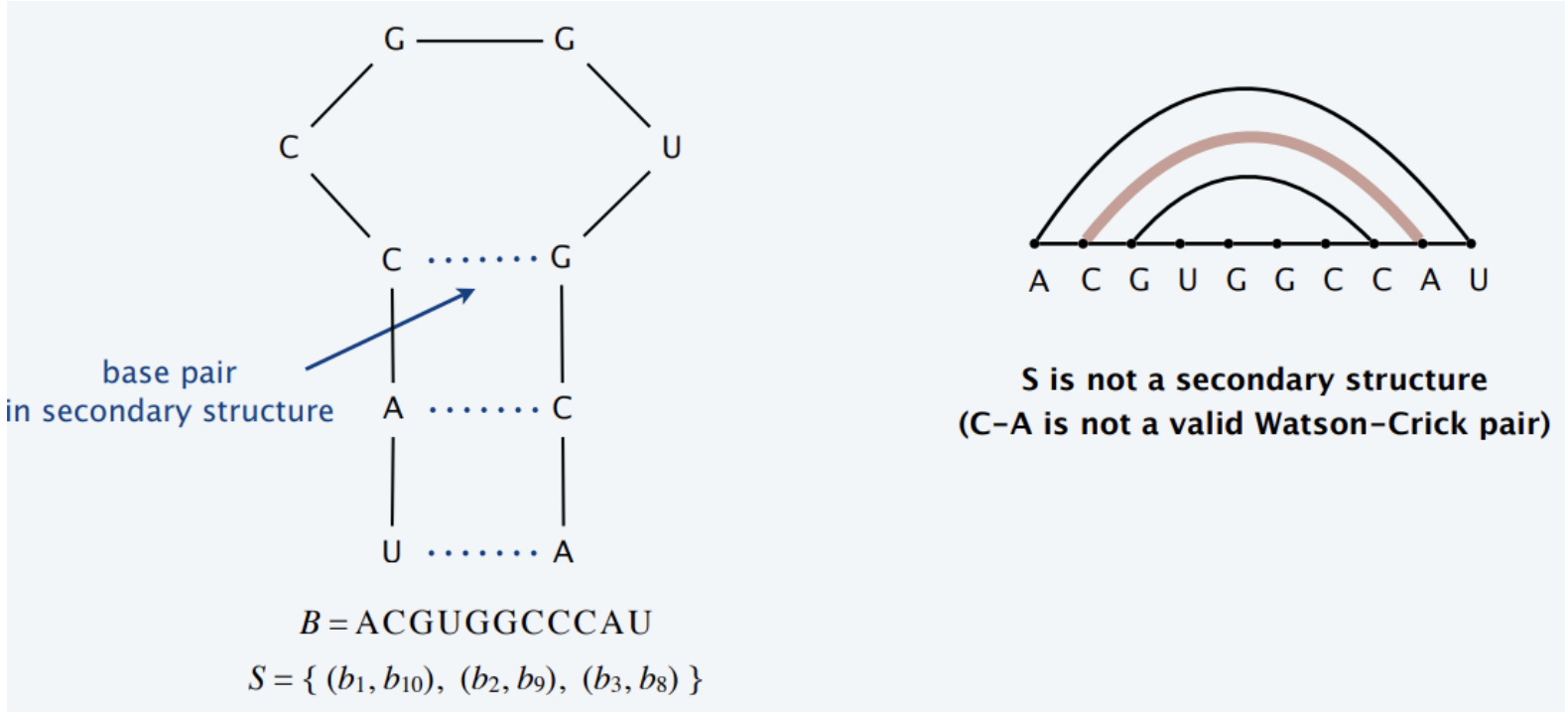
- RNA：字母表{A, C, G, U}上的字符串B=$b_1 b_2 \ldots b_n$
- 二级结构：单链RNA通过“绕回”和自身形成碱基对



RNA secondary structure for GUCGAUUGAGCGAAUGUAACAACGUGGCUACGGCGAGA

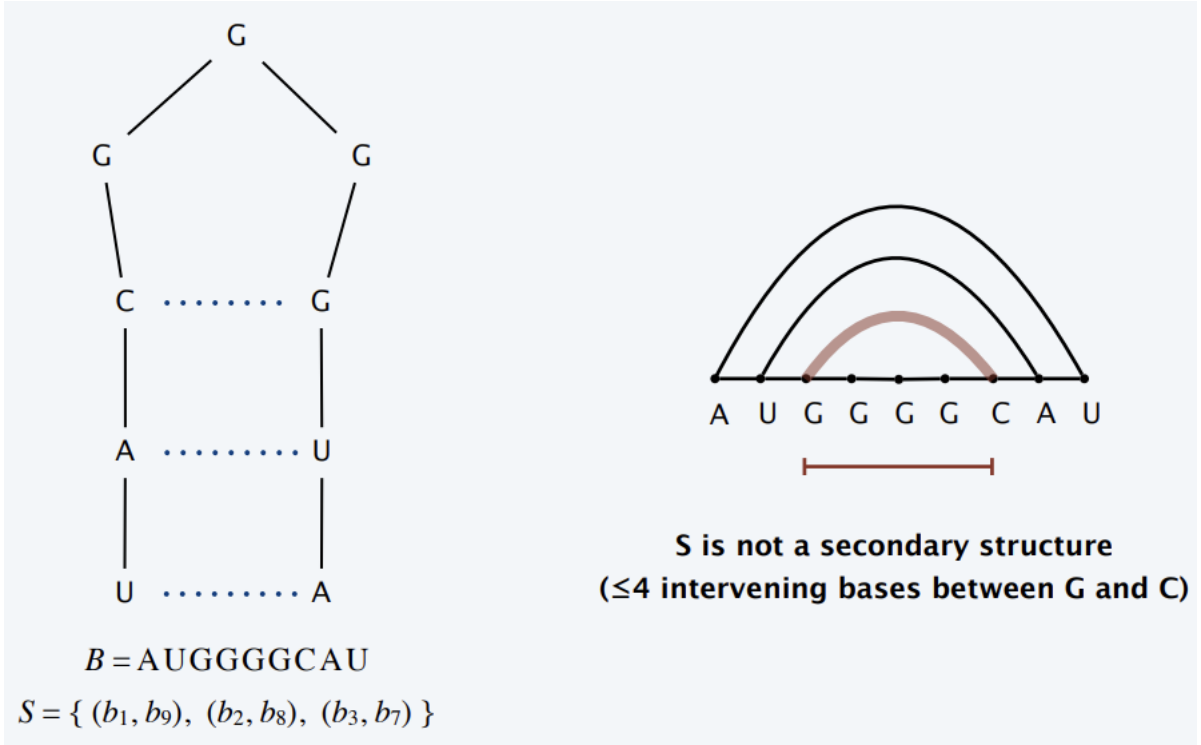
RNA二级结构

- 二级结构：满足以下条件的一组配对$S = \{(b_i, b_j)\}$:
- （1）Watson和Crick互补碱基配对：A-U和C-G



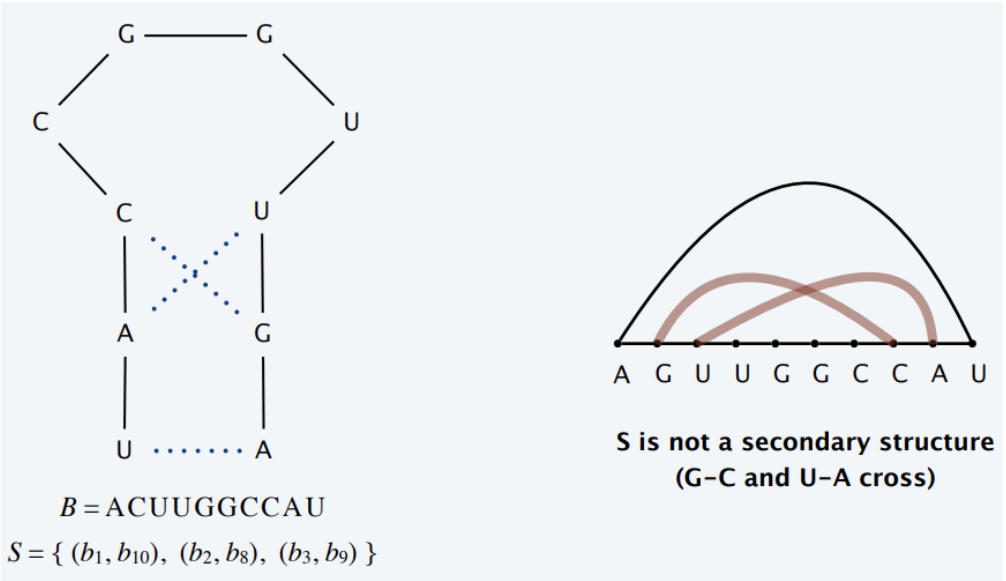
RNA二级结构

- 二级结构：满足以下条件的一组配对X：
- （1）Watson和Crick互补碱基配对：A-U和C-G
- （2）没有急转弯：S中每对至少被4个中间碱基隔开，也就是说，如果$(b_i, b_j) \in S$，则$i < j - 4$。



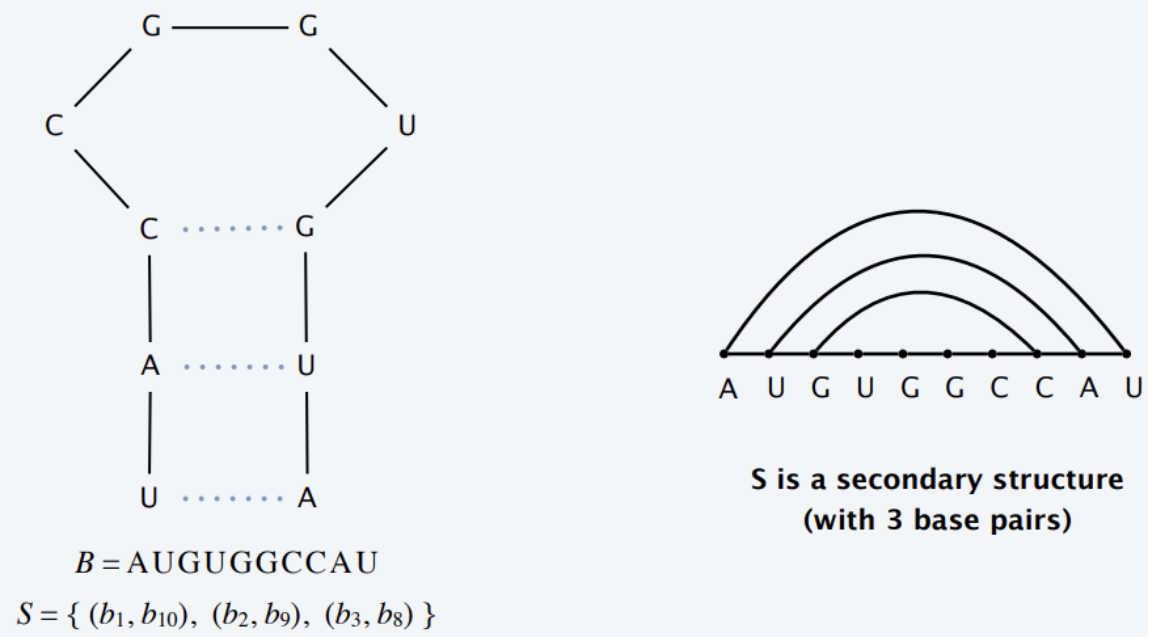
RNA二级结构

- 二级结构：满足以下条件的一组配对X：
- （1）Watson和Crick互补碱基配对：A-U和C-G
- （2）没有急转弯：S中每对至少被4个中间碱基隔开，也就是说，如果$(b_i, b_j) \in S$，则$i < j - 4$。
- （3）没有交叉：如果(b_i, b_j)和(b_k, b_ℓ)是S中的两对，不允许$i < k < j < \ell$



RNA二级结构

- 二级结构：满足以下条件的一组配对X：
- （1）Watson和Crick互补碱基配对：A-U和C-G
- （2）没有急转弯：S中每对至少被4个中间碱基隔开，也就是说，如果$(b_i, b_j) \in S$，则$i < j - 4$。
- （3）没有交叉：如果(b_i, b_j)和(b_k, b_ℓ)是S中的两对，不允许$i < k < j < \ell$



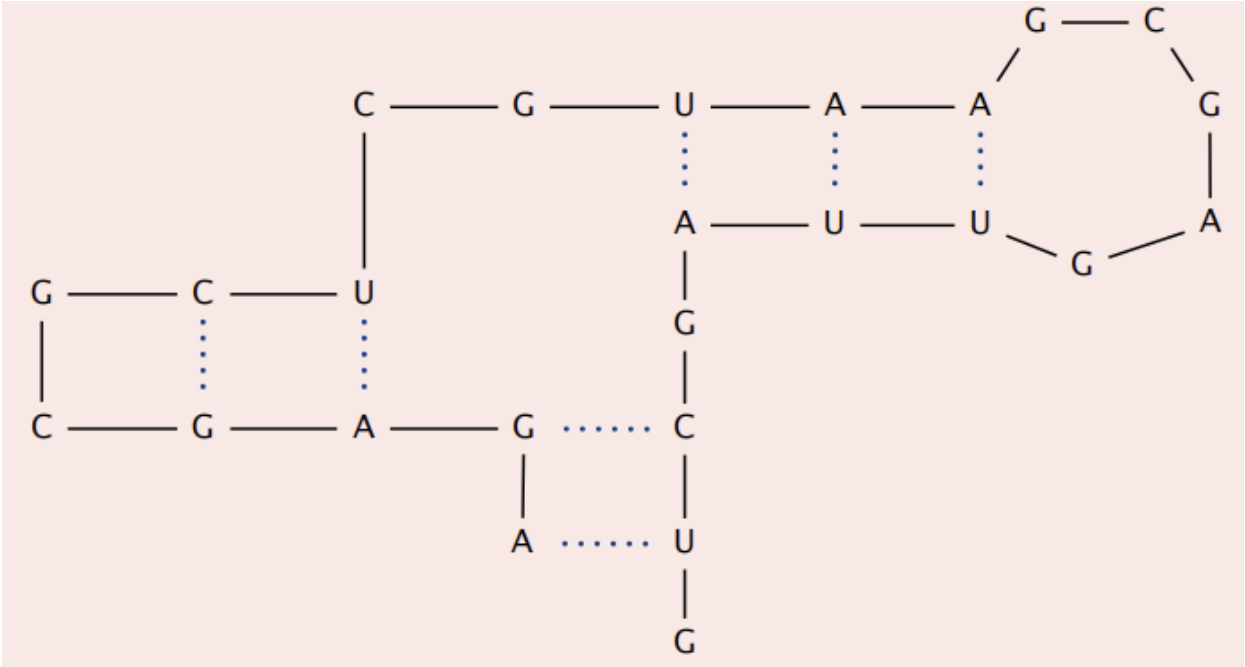
RNA二级结构

- 二级结构：满足以下条件的一组配对X：
- （1）Watson和Crick互补碱基配对：A-U和C-G
- （2）没有急转弯：S中每对至少被4个中间碱基隔开，也就是说，如果$(b_i, b_j) \in S$，则$i < j - 4$。
- （3）没有交叉：如果(b_i, b_j)和(b_k, b_ℓ)是S中的两对，不允许$i < k < j < \ell$

- 目标：给定RNA链B=$b_1 b_2 \ldots b_n$，寻找最大化碱基对数的二级结构S。

RNA二级结构：问题1

- 下图是二级结构吗？
- A. 是的
- B. 不是，违反Watson和Crick互补碱基规则
- C. 不是，违反没有急转弯规则
- D. 不是，违反没有交叉规则

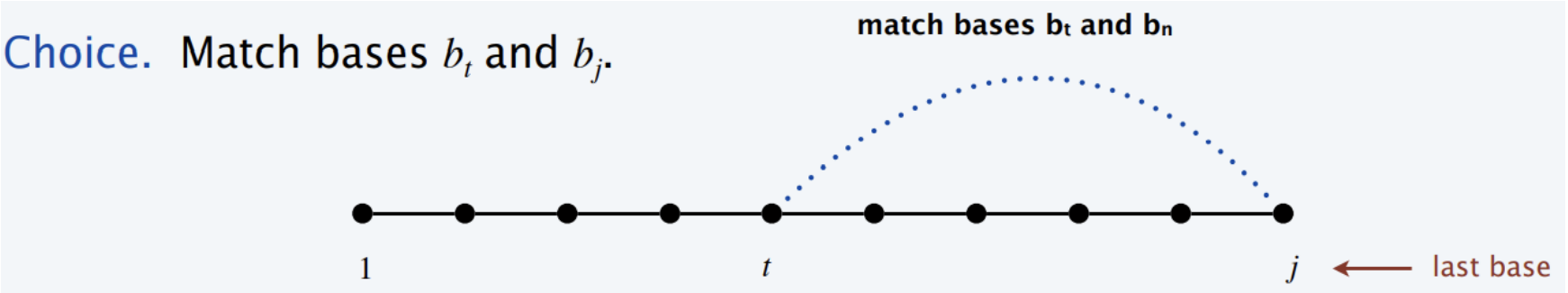


RNA二级结构：问题2

- 如何定义子问题？
- A. OPT(j)=子串$b_1 b_2 \ldots b_j$中的最长匹配碱基个数
- B. OPT(j)=子串$b_j b_2 \ldots b_n$中的最长匹配碱基个数
- C. A和B都对
- D. A和B都不对

RNA二级结构：子问题

- 第一次尝试： OPT(j)=子串$b_1 b_2 \dots b_j$中的最长匹配碱基个数
- 目标：OPT(n)

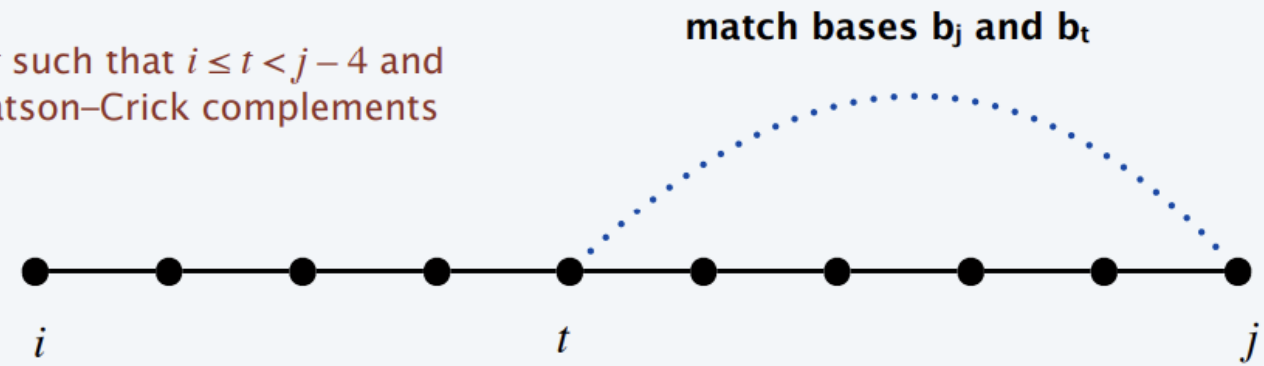


- 问题：形成两个子问题：
- （1）在$b_1 b_2 \dots b_{t-1}$中寻找二级结构：OPT(t-1)
- （2）在$b_{t+1} b_{t+2} \dots b_{j-1}$中寻找二级结构：？

解决：区间DP

- 定义OPT(i, j)=在$b_i b_{i+1} \ldots b_j$中的最多配对碱基的二级结构
- 讨论：
- （1）若$i \geq j-4$，则根据无急转弯规则，OPT(i, j)=0
- （2）若碱基b_j没有成对，则OPT(i, j)= OPT(i, j-1)
- （3）若碱基b_j和b_t配对且$i \leq t < j-4$，根据无交叉规则，形成两个子问题
- $OPT(i,\ j) = 1 + \max_t \{\ OPT(i,\ t-1) + OPT(t+1,\ j-1)\ \}.$

take max over t such that $i \leq t < j-4$ and b_t and b_j are Watson–Crick complements



RNA二级结构：问题3

- 以什么顺序计算OPT(i, j)?
- A. 先增加i，再增加j
- B.先增加j，再增加i
- C. A和B都对
- D. A和B都不对

计算OPT(i,j)的顺序

- 区间长度更小的优先——$|i-j|$增长的方向

RNA-SECONDARY-STRUCTURE$(n, b_1, \ldots, b_n)$

FOR $k = 5$ TO $n-1$

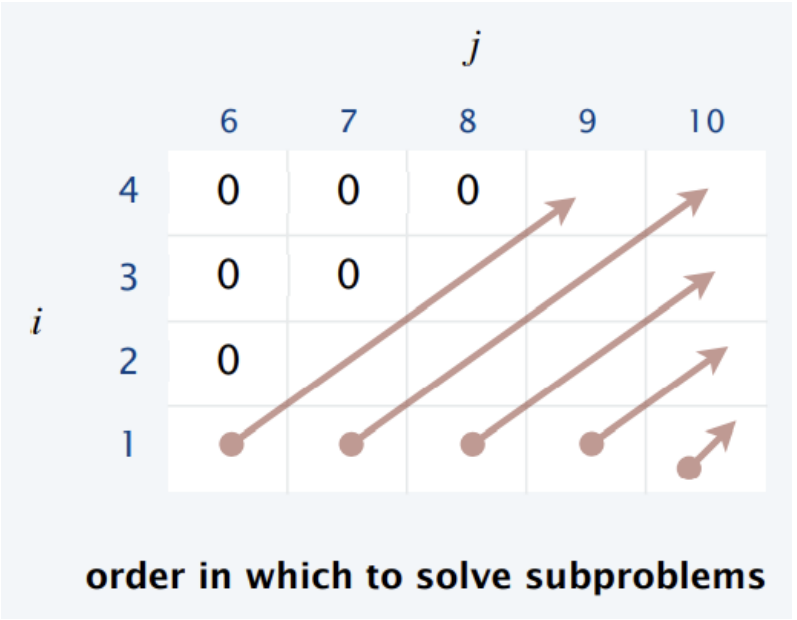
  FOR $i = 1$ TO $n-k$

    $j \leftarrow i+k$.

    Compute $M[i, j]$ using formula.

RETURN $M[1, n]$.

all needed values are already computed



order in which to solve subproblems

- 时间复杂度： $O(n^3)$；空间复杂度： $O(n^2)$

动态规划总结

- 套路：
  - 定义多项式个数的子问题
  - 原问题可在多项式时间通过子问题计算
  - 解决子问题的顺序从小到大
- 一些技巧：
  - 二元选择：加权区间调度
  - 多路选择：分段最小二乘
  - 二维DP：背包问题，字符串对齐，LCS
  - 区间DP：RNA二级结构