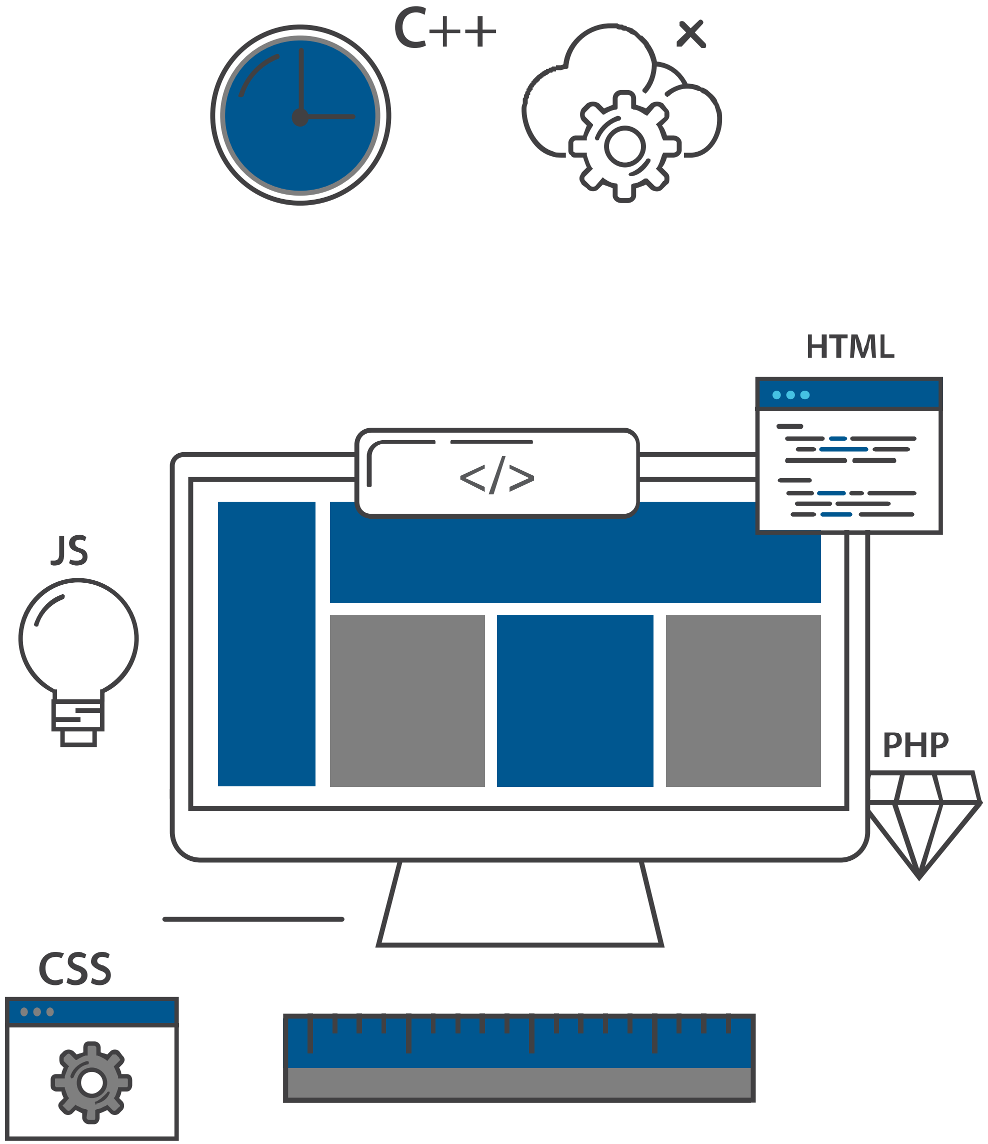
第六章 动态规划

李 翔
https://implus.github.io/



1

算法回顾

- 贪心算法
- 利用局部最优化原理，逐渐建立起完整的最优解
- 构成算法设计最自然的方法；对人们碰到的大多数问题，实际困难不在于确定几个贪心策略中那一个是正确的，而是事实上可能**不存在**有效的贪心算法

算法回顾

- 分治策略

- 把问题递归分解成几个更小规模的子问题；然后通过一个合理的复杂度把子问题的解合并成全局的解
- 分治策略没有强到可以把指数的蛮力搜索减少到多项式时间；倾向于*降低已经存在的多项式阶算法的阶数*

算法介绍

- 动态规划

- 通过把事情**分解为一系列子问题**，然后对越来越大的子问题建立正确的解，从而隐含的探查所有可行解的空间。
- 穿过问题可行解的指数规模的集合，**不必明确检查所有的解**

动态规划简介

- Bellman
- 1950s 为了研究系统控制问题
  而提出了动态规划的方法;

对控制理论界和数学界有深远影响。

贝尔曼, R.

Dynamic programming = planning over time.

动态规划应用

- 应用领域
  - 生物信息学
  - 控制论
  - 信息论
  - 行为学研究.
  - 计算机科学：理论, 图形, 人工智能, 系统, ….
- 一些经典的动态规划算法
  - Unix 比较两个文件的不同算法
  - Smith-Waterman序列比对算法.
  - Bellman-Ford 网络中的最短路径路由算法
  - Cocke-Kasami-Younger 自然语言处理算法

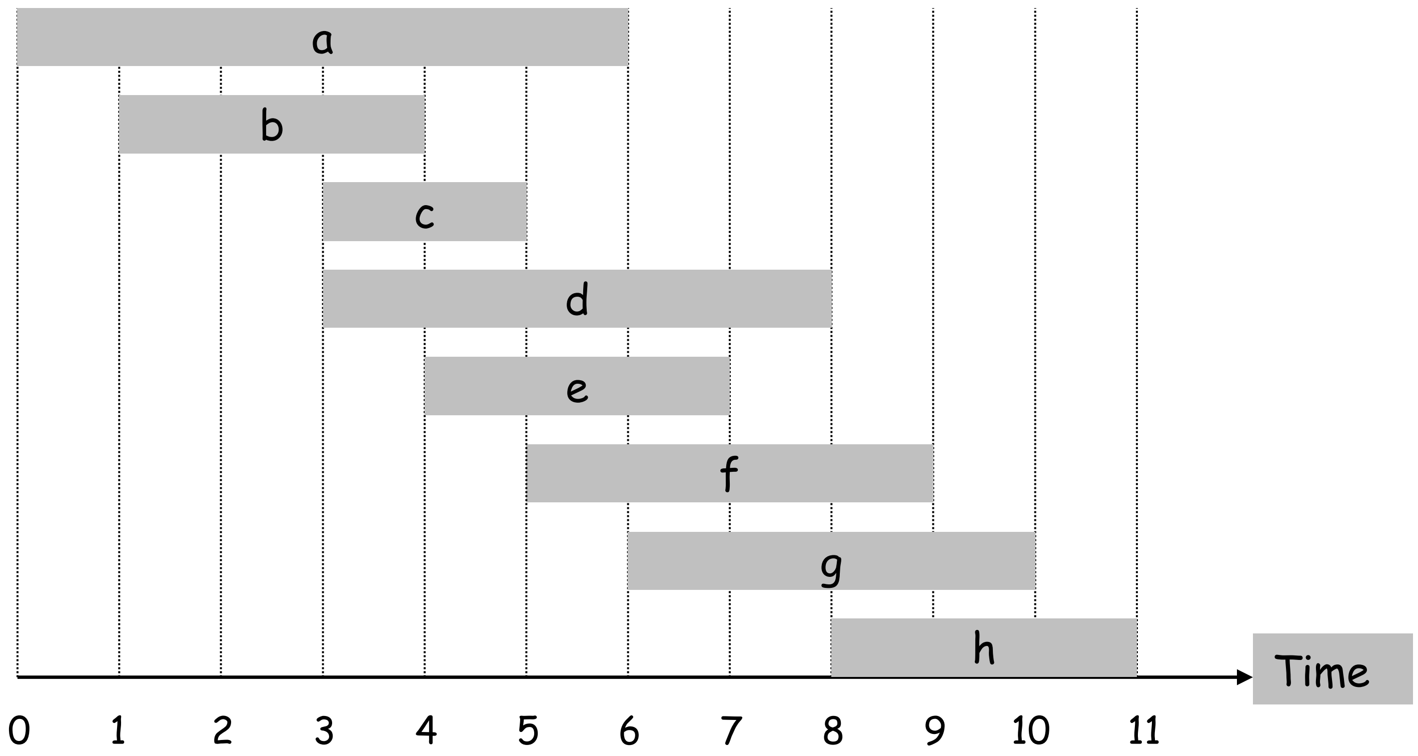
6.1 带权的区间调度

问题的提出

- 对于区间调度问题，不同区间有一样的权重，贪心算法可以得到最优解
- 如果不同区间有不同的权重：
  - 任务 j 在 s_j时间开始，f_j时刻结束，且其权重为v_j .
  - 两个任务如果对应的时间区间不相交，称为相容.
  - 目标：寻找最大的不相容的区间子集，使得所选区间的**权重之和**最大。

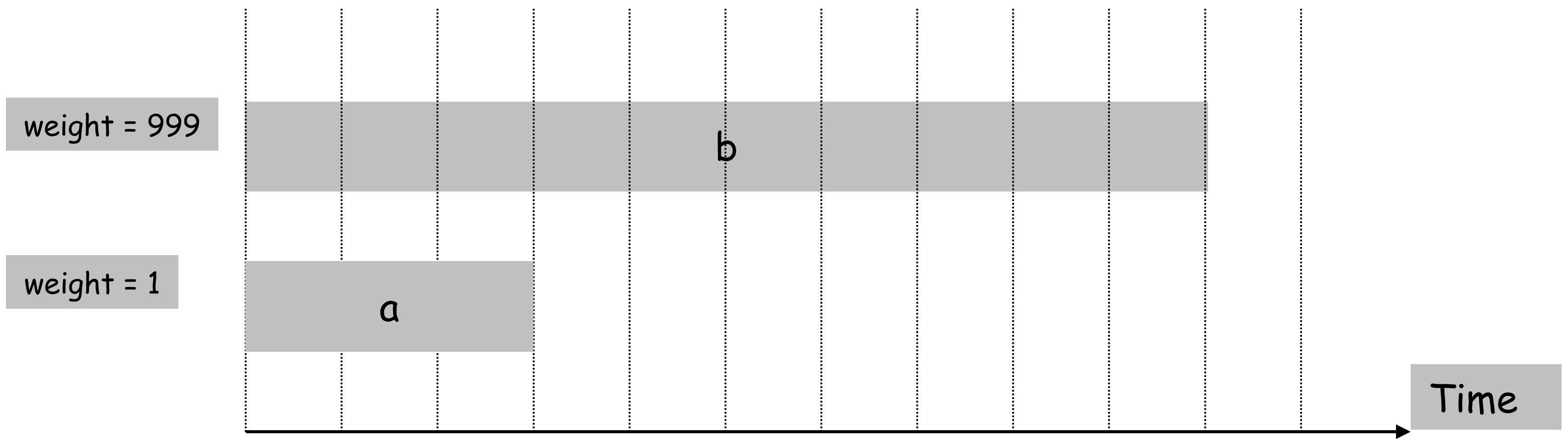
带权的区间调度

不同区间的影响是不一样的



带权的区间调度

- 贪心算法情形下，区间的权重可以看成都是1
- 对于区间权重不同的情形，贪心算法就失效了



解决之道：更加灵活的调度策略

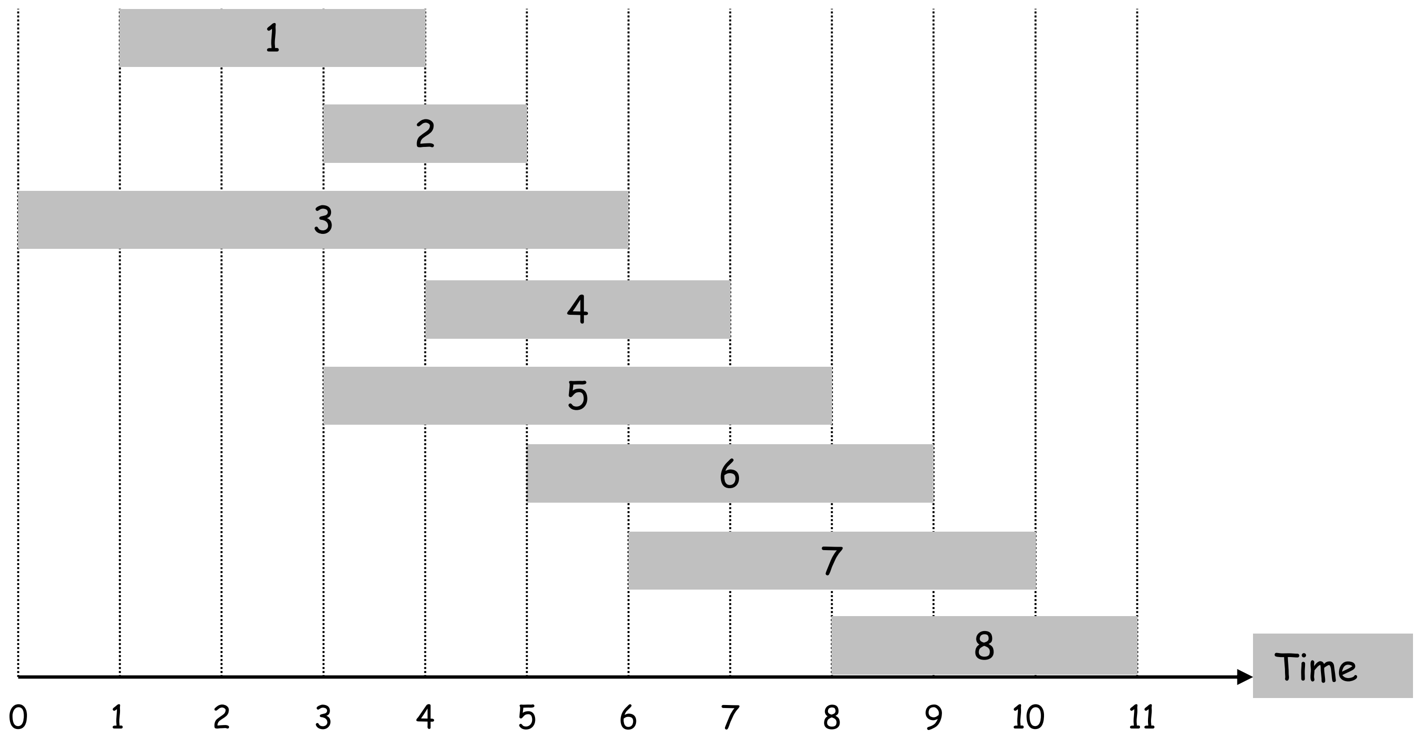
带权的区间调度

- 定义：任务需求按照结束时间排序：$f_1 \leq f_2 \leq \ldots \leq f_n$.
- 定义：我们对区间j定义p(j) 为使得区间i与j不相交的最大的标记i < j, 也就是说，i是最右边的与j不冲突的区间。

带权的区间调度

- Ex: p(8) , p(7) , p(2) =?

  p(8) = 5, p(7) = 3, p(2) = 0



带权的区间调度

- 分析最优解O的性质

  1.如果区间n∈O,那么O一定包含对于需求 {1,…,p(n)}所组成问题的一个最优解;

  2.如果n∉O,那么O等于对需求{1,…,n-1}所组成问题的最优解。

- 求区间{1,2,…,n}上的最优解包括查看形如{1,2,…,j}的**较小问题**的最优解。

带权的区间调度

- 对于任意在1与n之间的j值,令O_j表示对于有需求{1,…,j}所组成问题的最优解， 并且令OPT(j)表示这个解的值。
- 我们寻找的最优解就是O_n， 具有值OPT(n).
- 下面将关注OPT(j)的结构

带权的区间调度

- 情形1： OPT 包含任务需求 j.
  - 不会包含不相容的任务需求{ p(j) + 1, p(j) + 2, ..., j - 1 }
  - 包含剩下的任务需求1, 2, ..., p(j)的最优解
- 情形2： OPT 不包含任务需求 j.
  - 一定包含任务需求1, 2, ..., j-1的最优解

定理 6.1

$$OPT(j) = \begin{cases} 0 & \text{if } j = 0 \\ \max \{ v_j + OPT(p(j)),\ OPT(j-1) \} & \text{otherwise} \end{cases}$$

带权的区间调度

- 定理6.2 需求j属于集合{1,2,…,j} 上的最优解当且仅当

$$v_j + OPT(p(j)) \geq OPT(j-1)$$

上面定理构成了动态规划求解基础的最重要的部分：用较小子问题的最优解来表达更大规模问题的最优解的一个递推的等式。

带权的区间调度

- 根据上面的性质，首先我们先写出如下的算法

```
Input: n, s1,…,sn , f1,…,fn , v1,…,vn

Sort jobs by finish times so that f1 ≤ f2 ≤ ... ≤ fn.

Compute p(1), p(2), …, p(n)

Compute-Opt(j)
{
   if (j = 0)
      return 0
   else
      return max(vj + Compute-Opt(p(j)), Compute-Opt(j-1))
}
```

带权的区间调度

- 该算法的正确性：
- 定理6.3 对每个j=1,2,…,n, Compute-Opt(j)正确地计算了OPT(j).

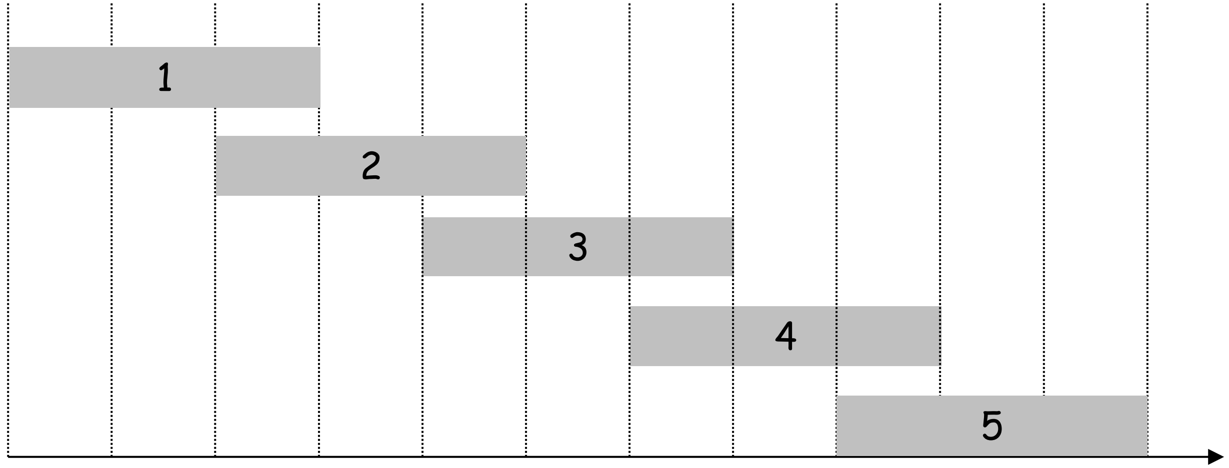
这个算法是不是一个好的**多项式阶**的算法呢?

最坏情形下这个算法将以指数时间运行!

带权的区间调度

- 观察　递归算法特别容易产生**冗余**的重复子问题计算，从而导致了指数阶算法.

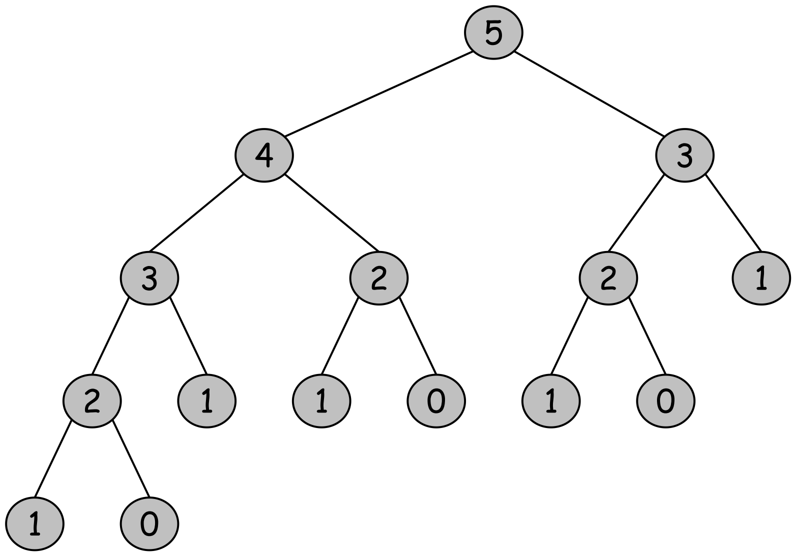
- Ex.



p(1) = 0, p(j) = j-2

带权的区间调度

- **递归调用的次数**增长就像Fibonacci 序列一样.



Fibonacci Sequence:

F(1)=F(2)=1,

F(n)=F(n-1)+F(n-2)

$$F(n) = (1/\sqrt{5}) * \left\{ \left(\frac{1+\sqrt{5}}{2} \right)^n - \left(\frac{1-\sqrt{5}}{2} \right)^n \right\}$$

带权的区间调度

- **递归的备忘录形式**
- 为了避免上面的重复计算，我们把中间计算的结果存储起来，需要的时候先查找是否计算过
- 下面将用到一个数组M[0…n]保存中间计算结果

带权的区间调度

```
Input: n, s1,…,sn , f1,…,fn , v1,…,vn

Sort jobs by finish times so that f1 ≤ f2 ≤ ... ≤ fn.
Compute p(1), p(2), …, p(n)

for j = 1 to n
   M[j] = empty
M[0] = 0                                ← 全局数组

M-Compute-Opt(j) {
   if (M[j] is empty)
   {
    M[j] = max(wj + M-Compute-Opt(p(j)),
                   M-Compute-Opt(j-1))
    }
   return M[j]
}
```

带权的区间调度

- 分析采用备忘录的算法
  - 按照结束时间排序： O(n log n).
  - 计算 p(·) :如果已对开始时间排序， 只需O(n)

定理6.4 M-Compute-Opt(n)的运行时间是O(n).

带权的区间调度

- 证明：对于M-Compute-Opt(j)：每一次调用所用的时间是O(1)，每次调用时
  - (i) 或者返回一个已经存在的值 M[j]
  - (ii)或者填入一个新项 M[j]，并且有两次递归调用
- 定义进展度量Φ 为M[]的非空元素数目.
  - 初始Φ = 0，全局中 Φ ≤ n;
  - Φ每次增加1；

  可以推导出至多会有2n 次递归调用.
- 所以 M-Compute-Opt(n)总的运行时间是 O(n)

带权的区间调度

- 预处理：算法实现的过程中先对开始时间和结束时间排序，便于后面的处理
- 一些编程语言，如Lisp，就可以自动实现备忘录的功能(built-in support for memorization)，因而有好的执行效率；但是在其他的一些语言中，比如Java，就没有实现这一功能。

带权的区间调度

- 前面的动态规划算法只计算了一个最优解的值，如果想要输出这组最优的区间的集合，如何处理？

利用存储的M[]数组进行后期处理

根据定理6.2，j属于一个区间集合{1,…,j}的最优解当且仅当 $v_j + OPT(p(j)) \geq OPT(j-1)$

带权的区间调度

```
Run M-Compute-Opt(n)
Run Find-Solution(n)

Find-Solution(j) {
   if (j = 0)
      output nothing
   else if (vj + M[p(j)] > M[j-1])
      print j
      Find-Solution(p(j))
   else
      Find-Solution(j-1)
}
```

算法的复杂度? O(n)

动态规划原理

- 对于带权区间调度问题，我们设计了一个多项式时间的算法：先设计一个（指数时间的)递归算法，然后通过备忘录将其转换成一个有效的递推算法，这个算法利用了全局数组M[]来记录递归函数值。
- 提出另外一种解决的思路：在**子问题上迭代**，而不是递归的计算解

动态规划原理

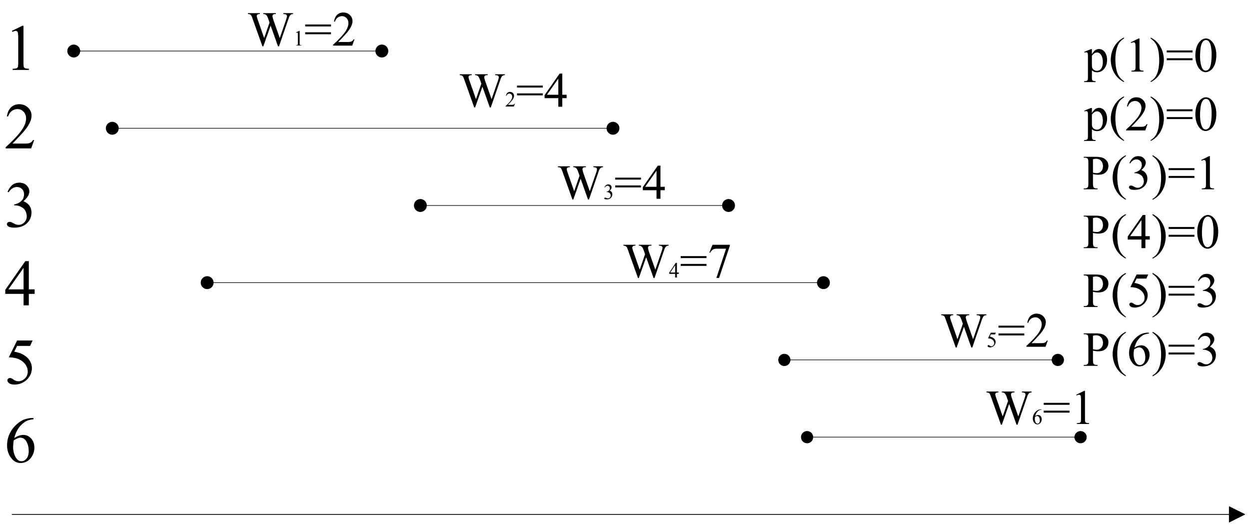
- 对于带权区间调度问题，我们可以不使用备忘录式的递归方法，而通过迭代算法直接计算M[]中的项。

```
Input: n, s1,…,sn , f1,…,fn , v1,…,vn

Sort jobs by finish times so that f1 ≤ f2 ≤ ... ≤ fn.

Compute p(1), p(2), …, p(n)

Iterative-Compute-Opt
{
   M[0] = 0
   for j = 1 to n
      M[j] = max(vj + M[p(j)], M[j-1])
}
```

自底向上

动态规划原理

- **Iterative-Compute-Opt的运行时间是O(n)，上面算法清楚的运行了n次迭代运算。**



M=

0	1	2	3	4	5	6
0	2					
0	2	4				
0	2	4	6			
0	2	4	6	7		
0	2	4	6	7	8	
0	2	4	6	7	8	8

动态规划原理与分治策略

- 相似之处
  - 通过子问题的解来求出全局问题的解
- 不同之处
  - 子问题*不独立*
  - 把子问题的解存在备忘录中

动态规划原理

动态规划类型问题:

1. 只存在多项式个子问题;
2. 容易从子问题的解计算初始问题的解;
3. 在子问题从“最小”到“最大”存在一种自然的顺序,与一个容易计算的**递推式**相联系,这个递推式允许从某些更小的子问题来确定一个子问题的解。

动态规划原理

动态规划的**过程**

1. 刻划最优解的结构；
2. 递归定义最优解的值；
3. 按照**自底向上**的方式或自顶向下记忆化的方式计算最优解的值并记录下求解的途径；
4. 按照求解途径给出最优解的形成过程

6.2 分段的最小二乘

- 问题的提出

- 绘制在一组二维数轴上的科学统计数据，试图用一条“最佳拟合”的线穿过这些数据

- 统计学与数值分析中的基本问题

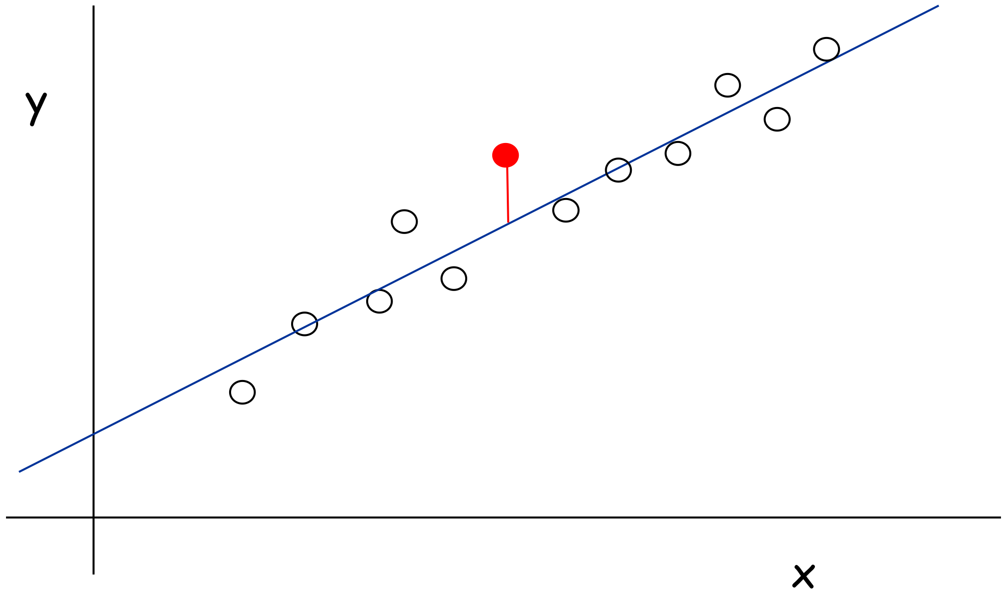
分段的最小二乘

- 给定平面上的由n个点组成的集合P:(x_1, y_1), (x_2, y_2) , . . . , (x_n, y_n).
- 假设$x_1 < x_2 < \ldots < x_n$
- 给定由方程y=ax+b定义的直线L, 我们说L相对于P的误差是它对于P中点的“**距离**”的平方和;
- 自然的目标是寻找具有最小误差的直线

分段的最小二乘

误差

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



分段的最小二乘

- 求极值的问题

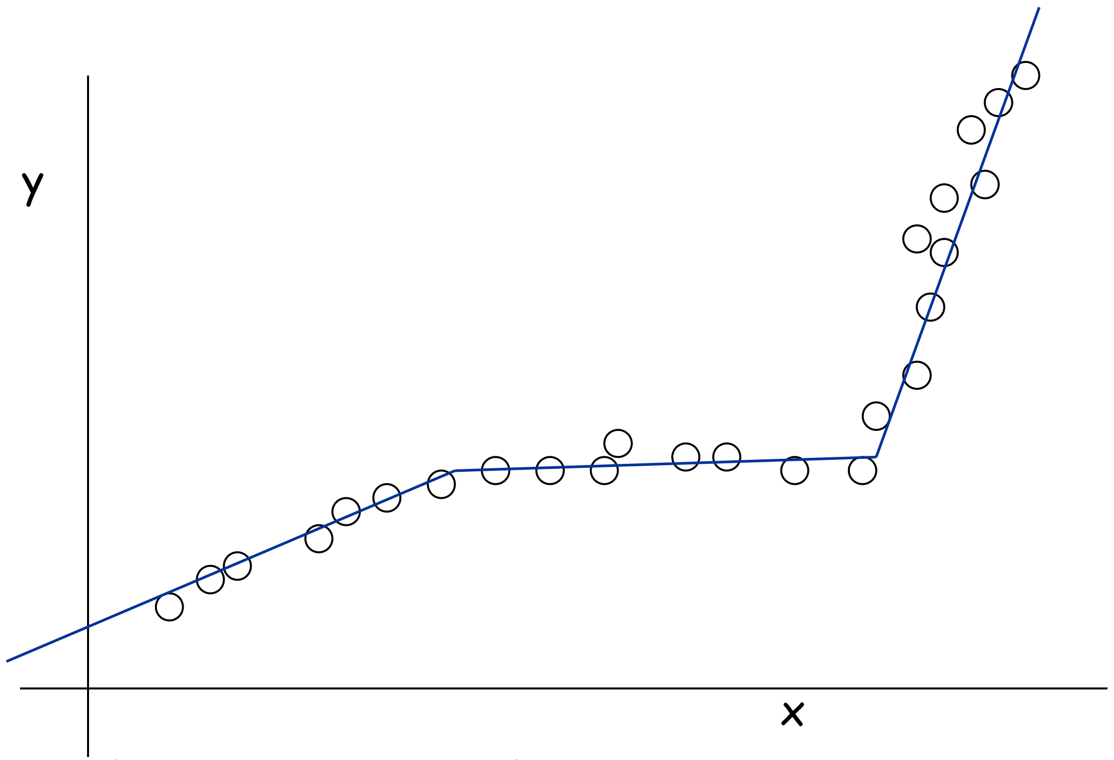
- 通过微积分中的方法，可以知道

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

分段的最小二乘

- 实际中，有些时候数据看起来不像在一条直线中，而是大致位于两条连续的直线中，如何把“最佳拟合”这个概念形式化?
- 使用最少的直线来拟合这些点

分段的最小二乘



如何在拟合的精确度和使用最少的直线数目之间找到一个平衡?

分段的最小二乘

- 分段的最小二乘问题，问题形式化
- 给定一组点$(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$，其中$x_1 < x_2 < \ldots < x_n$
- 寻找一组直线使得：在所有直线上的**平方误差之和**E，**直线的数目**L之间找到一个好的平衡
  - 因此定义罚分度量函数：$E + c\,L$，其中 $c > 0$是某一个常数.

分段的最小二乘

- 采用动态规划的方法找到一个多项式时间的算法
- 定义.
  - OPT(j) =对于点p_1, p_{i+1} , . . . , p_j的最优解
  - e(i, j) = 关系到p_i, p_{i+1} , . . . , p_j的任何直线的最小误差

递推式?

分段的最小二乘

- 定理6.6 如果最优划分的最后一段是p_i, p_{i+1} , . . . , p_n, 那么最优解的值是

$$OPT(n)= e(i, n) + c + OPT(i-1).$$

为得到OPT(j),对于某一个i, 我们应该以最好的方式产生最后的一段p_i, p_{i+1} , . . . , p_j.

分段的最小二乘

- 定理6.7 对于$p_1, \ldots, p_j$的子问题，

$$OPT(j) = \begin{cases} 0 & \text{if } j = 0 \\ \min\limits_{1 \le i \le j} \{\ e(i,j) + c + OPT(i-1)\ \} & \text{otherwise} \end{cases}$$

现在设计这个算法中的困难部分已经克服：我们将给出一个自底向上的算法

算法

```
INPUT: n, p1,…,pN , c

Segmented-Least-Squares() {
   M[0] = 0
   for j = 1 to n
      for i = 1 to j
         compute the least square error eij for
           the segment pi,…, pj

   for j = 1 to n
      M[j] = min 1 ≤ i ≤ j (eij + c + M[i-1])

   return M[n]
}
```

算法

复杂度分析

- 算法的前半部分，计算$O(n^2)$ 个$e(i, j)$，根据前面给出的公式计算每一对$e(i, j)$需要$O(n)$ 的复杂度;

  后半部分，总的代价应该是$O(n^2)$；
- 因此总的算法代价应该是**$O(n^3)$**；如果预先计算了统计信息$e(i, j)$，那么代价就是$O(n^2)$.

算法

- 通过数组M向回寻找计算最优的划分

Find-Segments(j)

If j=0 then

不用输出

Else

找一个使得e(i,j)+c+M[i-1]最小的i

输出这一段{p_i,...,p_j}作为直线段，以及输出递归函数Find-Segments(i-1)的结果

End if

6.4 子集和与背包

- 问题：有一台可以处理作业的机器，我们有一组需求{1,2,…,n}.对于某个数W,我们只能在时间0和时间W之间的区间使用这个资源。每个需求对应于一个需要w_i时间来处理的作业。我们的目标是处理这些作业，在“截至点”W之前能够使机器充分使用。

子集和问题

- 给定n个项{1,…,n},每个项有一个给定的非负的权w_i, 以及给定一个界W.我们想选择项的一个子集S使得$\sum_{i\in S} w_i \le W$，并且在这个前提下使得$\sum_{i\in S} w_i$达到最大。这个问题称为子集和问题。
- 这是更一般问题背包问题的特殊情形

背包问题

Knapsack Problem

- 用物品{1,…,n}的子集装入一个容量W的背包使得它装的最满（或者装入的价值最大）。
- 每一个需求i有一个值v_i与一个权w_i,在总权不超过W的限制下，选择一个最大总值的子集。

子集和与背包

约束条件： $\sum_{i \in S} w_i \le W$

要求，最大化 $\sum_{i \in S} v_i$

例子：

W = 11

Item	Value	Weight
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

子集和与背包

贪心策略可不可以呢?

- 比如最直观的方法是，按照单位价值v_i / w_i排序，然后从高到低选择物品，直到背包不能再装下为止。
- 选择{ 5, 2, 1 } 达到 35。

最大的价值?　　选择{ 3, 4 }，　价值 40.

⇒ 贪心算法得不到最优解!

子集和与背包

尝试动态规划的方法

- 借鉴带权区间调度问题，用符号OPT(i) 表示使用需求{1,…,i}的一个子集的可能的最好的解。
- 于是可以得到：
- 情形1：OPT 没有选择需求 i.
  - OPT 选择 { 1, 2, …, i-1 } 的最优子集

子集和与背包

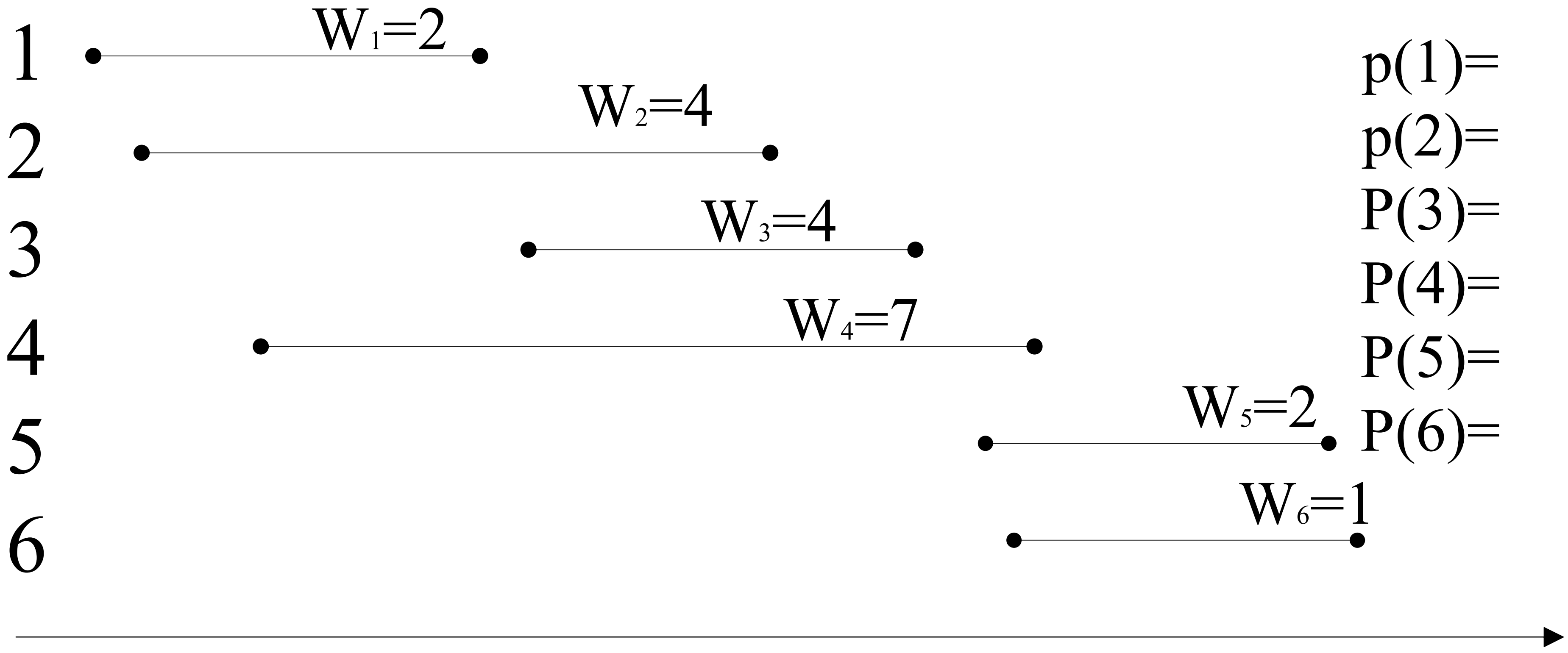
- 情形2 OPT选择了需求i, 那么

  我们不能拒绝任何别的需求，因此我们只有剩下W-w_i的权来接收其余需求的集合

$$S \subseteq \{1, \ldots, i-1\}$$

这种规划的方法不是很有效，提醒我们为找出OPT(n)的值，我们不仅需要物品子集作为参数，同时还需要剩余的总权(背包空间)作为参数；需要重新定义子问题

回顾（加权区间调度）



p(1)=
p(2)=
P(3)=
P(4)=
P(5)=
P(6)=

思考

在古代，有一种特殊的加密方式被用于传递秘密信息。这种方式将字母表中的每个字母映射为一个数字，具体规则如下：

- 'A' -> "1"
- 'B' -> "2"
- ...
- 'Z' -> "26"

这种方式在当时非常有效，因为只有知道映射规则的人才能解码这些信息。然而，随着时间的推移，一些密文被发现并保存了下来，但解码这些密文的方法却逐渐失传。

最近，考古学家在一次探险中发现了一本古老的密码本，其中记录了一些加密信息。这些信息不仅包含数字，还包含了一些特殊的符号，比如 ''。*根据密码本的注释，*' 可以表示从 '1' 到 '9' 的任意一个数字。

为了研究这些密文，你需要编写一个程序来计算解码这些密文的方法数。具体规则如下：

1. **数字解码**：
   - 每个数字可以单独解码为一个字母。
   - 如果两个数字组合起来在 1 到 26 之间，也可以解码为一个字母。例如，"11" 可以解码为 'K'。

任务

给定一个字符串 s，其中包含数字和 '*'，请你计算解码该字符串的所有可能方法数。

- **输入**：s = "11106"
- **输出**： 2
- **解释**："11106" 可以解码为 "AAJF"、"KJF" 等多种方式，总共有 2 种解码方法。

子集和与背包

- 动态规划方法：
- 定义OPT(i, w) 表示使用物品{1,…,i}的子集且所允许的最大权是w的最优解的值。

✓ 情形1 OPT 没有选择 i；

OPT 选择子集{ 1, 2, …, i-1 }，采用w 的最优解

✓ 情形2 OPT 选择 i；

余下部分OPT 选择子集{ 1, 2, …, i-1 }，采用$w-w_i$ 的最优解

子集和与背包

- 据此马上我们就得到了递推式

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

由此可见，增加一个变量重新进行动态规划，会变成更小的子问题，有助于设计动态规划算法。

算法

- 自底向上的算法
- 填满一个n*W的二维数组

```
Input: n, w1,…,wN, v1,…,vN

for w = 0 to W
   M[0, w] = 0

for i = 1 to n
   for w = 1 to W
      if (wi > w)
         M[i, w] = M[i-1, w]
      else
         M[i, w] = max {M[i-1, w], vi + M[i-1, w-wi ]}

return M[n, W]
```

算法

W + 1

n + 1

	0	1	2	3	4	5	6	7	8	9	10	11
ϕ	0	0	0	0	0	0	0	0	0	0	0	0
{ 1 }												
{ 1, 2 }												
{ 1, 2, 3 }												
{ 1, 2, 3, 4 }												
{ 1, 2, 3, 4, 5 }												

W = 11

Item	Value	Weight
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

算法

W + 1

n + 1

	0	1	2	3	4	5	6	7	8	9	10	11
ϕ	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: { 4, 3 }
value = 22 + 18 = 40

W = 11

Item	Value	Weight
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

算法

时间复杂度?

- 我们正在建立一个二位数组的表，并且我们在O(1)时间内计算每个值M[i,w];于是得到
- 定理6.9 上面算法正确的计算这个问题的最优值，并且运行在Θ(n W)时间。

算法

- 为了复原物品的最优集合，我们可以采用与前面类似的向回追踪的方法

- 定理6.10 给定子问题的最优值的表M,可以在O(n)时间内找到最优集合。

推广

- 运行时间不是n的多项式函数。相反，它是n与W的多项式函数，我们把这个算法称为**伪多项式**的。
- 这里的子集和问题，背包问题可以在$\Theta(n\ W)$代价内找到最优解
- 对于另外的子集和决策问题：（背包决策问题的特殊情况）任给自然数$w_1,\ldots,w_n$和目标值W,问$\{w_1,\ldots,w_n\}$有一个子集加起来恰好等于W吗?

NP-complete

6.5 RNA二级结构

问题描述

- Watson, Crick提出双螺旋的DNA结构

- 单链RNA分子倾向于弯回去并且与自己构成碱基对； 了解这种结构有助于了解分子行为的基础

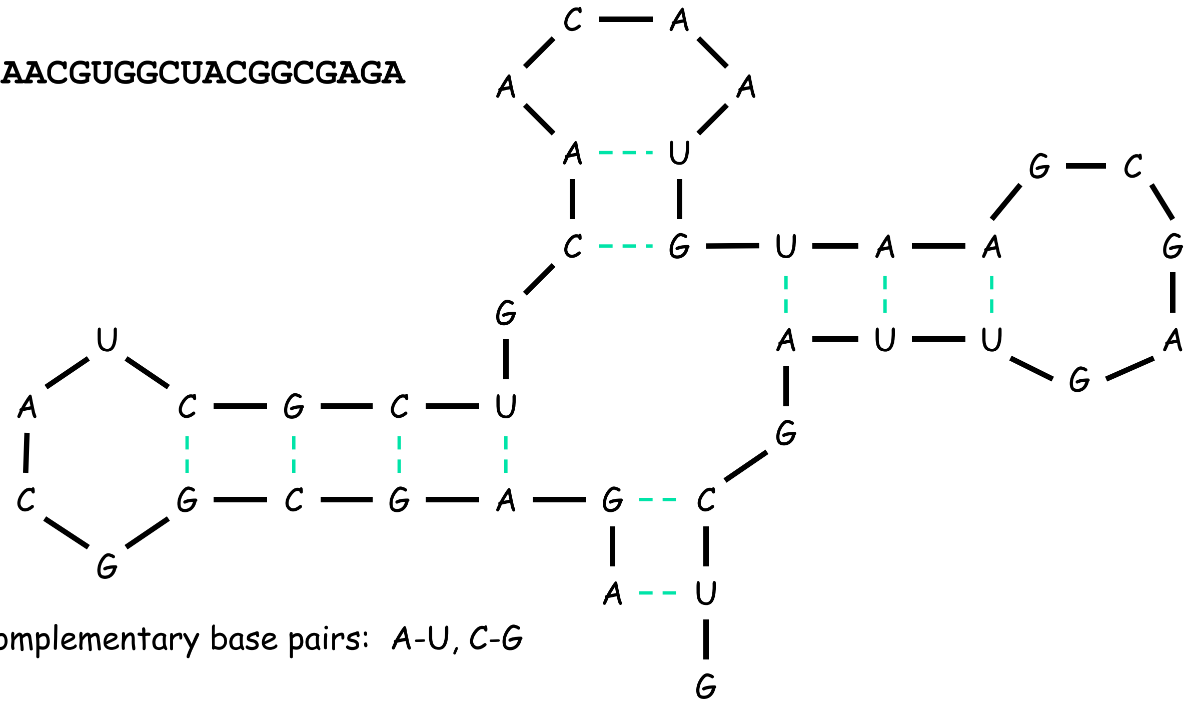
- 配对的四种基本单位{ A, G, C, U }.

RNA二级结构

- RNA可以看成定义在字母表{ A, C, G, U }

上的字符串B = $b_1 b_2 \dots b_n$

Ex: GUCGAUUGAGCGAAUGUAACAACGUGGCUACGGCGAGA



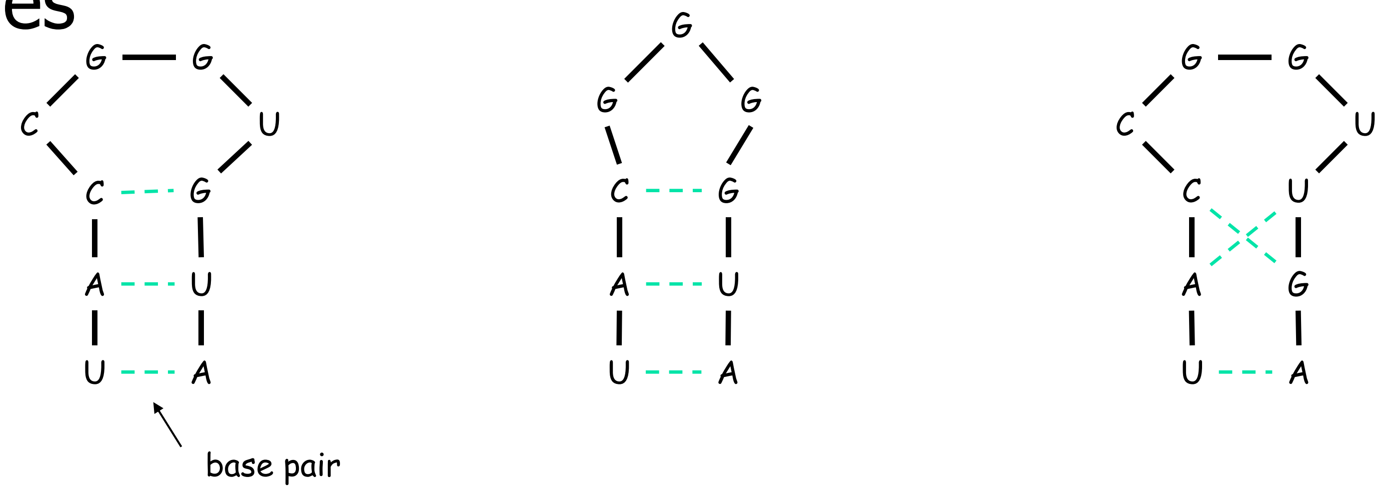
complementary base pairs: A-U, C-G

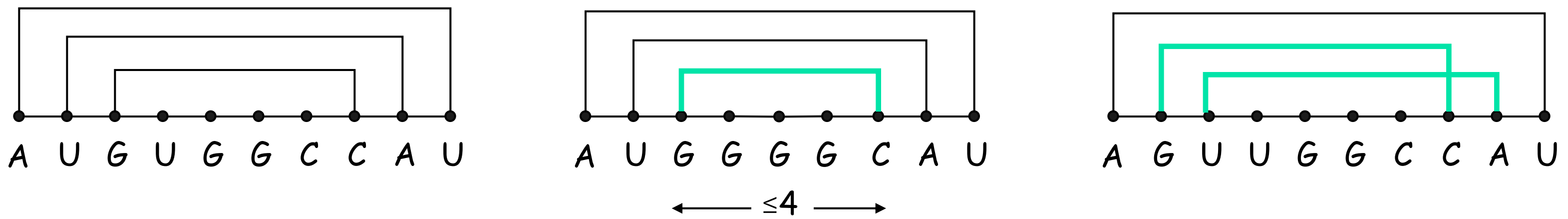
RNA二级结构

- 二级结构. 一些配对$S = \{ (b_i, b_j) \}$满足:
  - [Watson-Crick.] S是一个配对集合，其中的每一个配对符合S Watson-Crick 配对规则: A-U, U-A, C-G, G-C.
  - [没有尖的转弯.] S中每个对的两端被至少四个插入的碱基所分割. 即如果 $(b_i, b_j) \in S$, 那么$i < j - 4$.
  - [不交叉.] 如果 (b_i, b_j) 与 (b_k, b_l)是 S中的两个对, 那么我们不能有$i < k < j < l$.
  - 没有碱基出现在一个以上的对中。

RNA二级结构

Examples





ok

尖的转弯

交叉

RNA二级结构

- 对一个RNA分子，在所有可能的二级结构中，那些与生理学条件下产生的那个最象？
- 通常假设单螺旋RNA分子将构成具有**最优总自由能**的二级结构，这样最稳定
- **近似**认为二级结构的自由能只是与它包含的碱基对的个数成正比

RNA二级结构

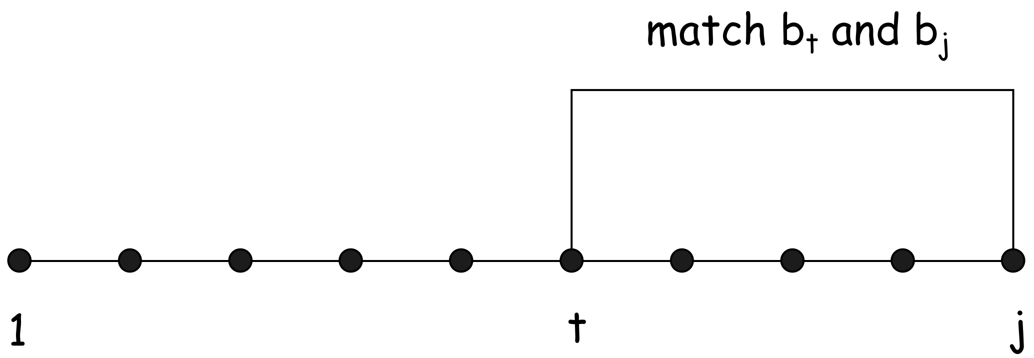
- 现在可以把基本的RNA二级结构预测问题描述的很客观:

- 对一个单螺旋RNA分子 $B = b_1 b_2 \ldots b_n$, 确定具有最大碱基配对个数的二级结构S.

- 动态规划的第一个尝试
- 定义OPT(j)是在$b_1b_2\ldots b_j$上的二级结构中最多的碱基对数。
- 我们可以知道：j<=5， OPT(j)=0;
- 下面开始尝试按照较小的子问题的解来写出OPT(j)的递推式

设计与分析算法

- 情形 1　 j不包含在一个对里面
  只需对OPT(j-1)考虑我们的解；
- 情形 2　 对某个t<j-4, j与t配对

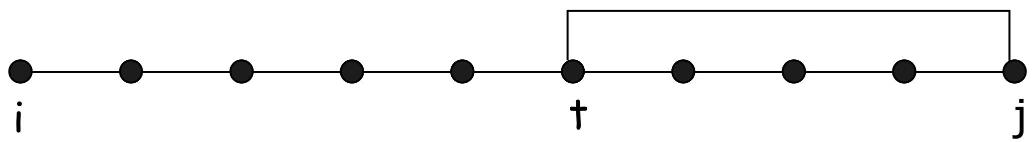


问题的困难：

- 根据不交叉条件，知道碱基对一个在$b_1b_2\ldots b_{t-1}$上（OPT(t-1)），另一个碱基对在$b_{t+1}b_{t+2}\ldots b_{n-1}$上。
- 后一个子问题不在我们定义的子问题类型里面
- 因此需要增加一个变量，处理不从b_1开始的子问题

区间上的动态规划

- 定义OPT(i, j) 表示在$b_i b_{i+1} \ldots b_j$上的二级结构中碱基对的最大数目。
  - 情形 1. 如果 $i \geq j - 4$.
    - 根据没有尖的弯的规则，OPT(i, j) = 0.
  - 情形 2. 碱基b_j 没有配对.
    - OPT(i, j) = OPT(i, j-1)
  - 情形 3. 碱基 b_j 与碱基 b_t配对，其中$i \leq t < j - 4$.
- 根据不交叉的限制就把问题分成两个子问题
  - $OPT(i, j) = 1 + \max_t \{ OPT(i, t-1) + OPT(t+1, j-1) \}$



- 定理６．１３

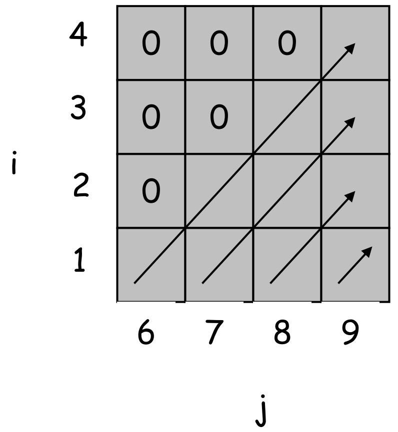
$$OPT(i, j) = \max\{ OPT(i, j-1), 1 + \max_t \{ OPT(i, t-1) + OPT(t+1, j-1) \}\},$$

其中$\max_t$遍取所有的t,使得b_t与b_j是一对被允许的碱基.

- 确认我们建立子问题解的适当顺序
- 按照区间**长度增长**的顺序建立解

```
RNA(b1,…,bn) {
   for k = 5, 6, …, n-1
      for i = 1, 2, …, n-k
         j = i + k
         Compute M[i, j]

   return M[1, n]
}
```

using recurrence



设计与分析算法

- 算法时间复杂度
- 存在$O(n^2)$个子问题;每次求值用$O(n)$时间
- 总运行时间是$O(n^3)$

动态规划方法小结

- 动态规划技术
  - 两种情况里面挑选最优解：带权的区间调度.
  - 多种情况里面挑选最优解：分段的最小二乘.
  - 增加一个变量：背包问题.
  - 在区间上的动态规划：RNA 二级结构.

- 自顶向下(递归) vs 自底向上(循环)：不同的思路

古代密文解码

在古代，有一种特殊的加密方式被用于传递秘密信息。这种方式将字母表中的每个字母映射为一个数字，具体规则如下：

- A -> “1”
- B -> “2”
- ...
- Z -> “26”

这种方式在当时非常有效，因为只有知道映射规则的人才能解码这些信息。然而，随着时间的推移，一些密文被发现并保存了下来，但解码这些密文的方法却逐渐失传。

最近，考古学家在一次探险中发现了一本古老的密码本，其中记录了一些加密信息。这些信息不仅包含数字，还包含了一些特殊的符号，比如“*”。根据密码本的注释，“*” 可以表示从 1 到 9 的任意一个数字。

为了研究这些密文，你需要编写一个程序来计算解码这些密文的方法数。具体规则如下：

1. **数字解码**：
   - 每个数字可以单独解码为一个字母。
   - 如果两个数字组合起来在 1 到 26 之间，也可以解码为一个字母。例如，”11” 可以解码为 ‘K’。
2. **特殊符号 ‘*’**：
   - ‘*’ 可以表示从 ‘1’ 到 ‘9’ 的任意一个数字。
   - 例如，”1*” 可以表示 “11”、”12”、”13”、”14”、”15”、”16”、”17”、”18” 或 “19”。
3. **无效解码**：
   - 任何以 ‘0’ 开头的数字组合是无效的。例如，”06” 不能解码为 ‘F’。

任务

给定一个字符串 s，其中包含数字和 “*”，请你计算解码该字符串的所有可能方法数。

示例

示例 1

- **输入**：s = “1*”
- **输出**：18

6.6 序列比对

- 比较单词的相似度
- 确定串之间的相似性是分子生物学的核心计算问题之一，可以用来判别基因类似的片段，确定某种遗传的规律
- 应用
  - Basis for Unix diff.
  - Speech recognition.
  - Computational biology

序列比对

- 例子:

  ocurrance

  occurrence

  有多相似?

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

5 mismatches, 1 gap

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

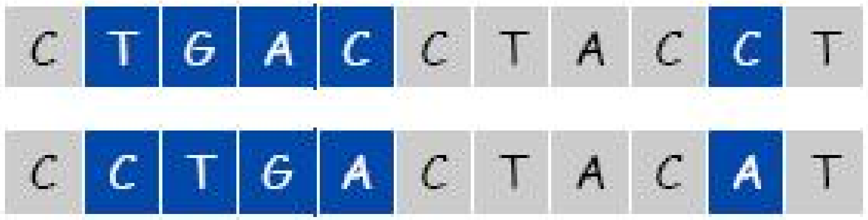
1 mismatch, 1 gap

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

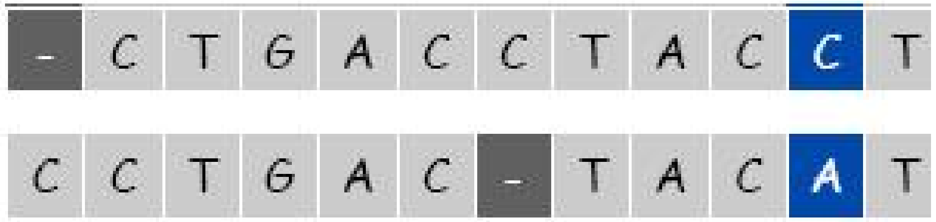
0 mismatches, 3 gaps

序列比对

- 编辑距离 [Levenshtein 1966, Needleman-Wunsch 1970]
  - 空隙罚分 δ; 不匹配罚分 α_{pq}.
  - 总的罚分= 空隙罚分和不匹配罚分之和.



$\alpha_{TC} + \alpha_{GT} + \alpha_{AG} + 2\alpha_{CA}$



$2\delta + \alpha_{CA}$

序列比对

- 目标：给定两个字符串$X = x_1\ x_2\ .\ .\ .\ x_m$；$Y = y_1\ y_2\ .\ .\ .\ y_n$，寻找最小罚分的比对方式.
- 定义. 配对x_i-y_j，$x_{i'}$-$y_{j'}$ 称为交叉，如果 $i < i'$, 但是 $j > j'$.
- 定义. 一个比对M是一些有序配对 x_i-y_j 的集合，每一项至多出现在一个配对中，而且没有配对交叉。

序列比对

比对罚分

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

Ex: CTACCG **vs.** TACATG.

$M = x_2\text{-}y_1, x_3\text{-}y_2, x_4\text{-}y_3, x_5\text{-}y_4, x_6\text{-}y_6.$

x_1	x_2	x_3	x_4	x_5		x_6
C	T	A	C	C	-	G
-	T	A	C	A	T	G
	y_1	y_2	y_3	y_4	y_5	y_6

序列比对

- 定理6.15 在一个最优比对M中，至少下述情况之一为真

i. （m，n）在M中;

ii. X的第m个位置没有被匹配;

iii. Y的第n个位置没有被匹配.

序列比对

- 定义 $OPT(i, j) = x_1\ x_2 \ldots x_i$ 与 $y_1\ y_2 \ldots y_j$比对的最小罚分
  - Case 1: x_i-y_j在OPT中
    - x_i- y_j不匹配罚分+ $x_1\ x_2 \ldots x_{i-1}$ 和 $y_1\ y_2 \ldots y_{j-1}$ 最小比对罚分
  - Case 2a: OPT 中 x_i 没有匹配
    - x_i处间隙罚分 + $x_1\ x_2 \ldots x_{i-1}$ 和 $y_1\ y_2 \ldots y_{j-1}\ y_j$最小比对罚分
  - Case 2b: OPT中 y_j没有匹配
    - y_j处间隙罚分+ $x_1\ x_2 \ldots x_{i-1}\ x_i$ 和 $y_1\ y_2 \ldots y_{j-1}$ 最小比对罚分

序列比对

- 所以递归式应该是

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

定理6.16 对于*i,j>=1,OPT(i,j)=* $\min\begin{cases}\alpha_{x_i y_j}+OPT(i-1,j-1)\\ \delta+OPT(i-1,j)\\ \delta+OPT(i,j-1)\end{cases}$

此外，(i,j)在最优比对中，当且仅当达到上面最小值。

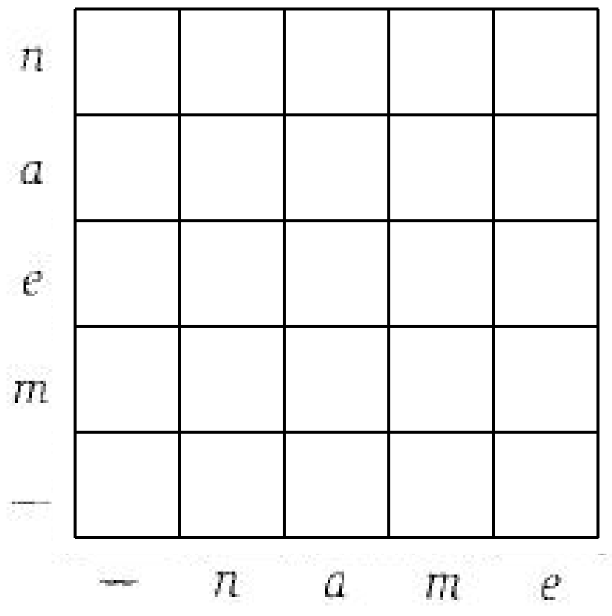
序列比对算法

```
Sequence-Alignment(m, n, x1x2...xm, y1y2...yn, δ, α) {
   for i = 0 to m
      M[0, i] = iδ
   for j = 0 to n
      M[j, 0] = jδ

   for i = 1 to m
      for j = 1 to n
         M[i, j] = min(α[xi, yj] + M[i-1, j-1],
                       δ + M[i-1, j],
                       δ + M[i, j-1])
   return M[m, n]
}
```

时间，空间复杂度？　　　$\Theta(mn)$

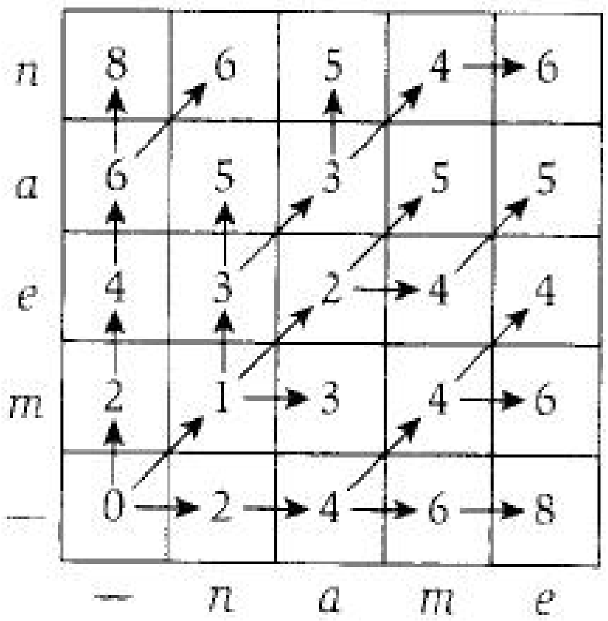
分析算法

- 建立一个二维方格图G_{XY}, 用来表示状态之间的转换。
- 例子：间隙罚分$\delta = 2$,元音与不同元音配对，辅音与不同辅音配对不匹配罚分1；元音与辅音配对罚分3.



分析算法

- 建立一个二维方格图G_{XY}，用来表示状态之间的转换。
- 例子：间隙罚分$\delta = 2$,元音与不同元音配对，辅音与不同辅音配对不匹配罚分1；元音与辅音配对罚分3.



分析算法

- 定理6.17 令f(i,j)表示G_{XY}中从(0,0)到(i,j)的一条路径**最小的代价**，那么对于所有的i,j, 我们有f(i,j)=OPT(i,j).

英语单词或者句子：m, n ≤ 10.

计算生物学：m = n = 100,000.

时间复杂度尚可承受，空间复杂度实在太大!

新的解决办法?

6.7 通过分治策略在**线性空间**的序列比对

- 我们可不可以在上面算法的基础之上，避免O(mn)的空间复杂度？

  计算最优比对值，比对结构

- 如果只关心最优比对的值，而不是比对结构本身，那么容易达到线性空间，算法如下：

线性空间的序列比对

Space-Efficient-Alignment(X,Y)

数组B[0…m,0…1]

初始化对每个i令B[i,0]=i δ

For j=1,…,n

B[0,1] = j δ

For i=1,…,m

B[i,1]= $\min[\alpha_{x_j y_j} + B(i-1,0), \delta + B(i-1,1), \delta + B(i,0)]$

Endfor

将B的第1列移到第0列来为下一次迭代留出空间

对每个i修改B[i,0]=B[i,1]

Endfor

线性空间的序列比对

- 容易验证当这个算法完成时，$i=0,1,\ldots,m$数组的项$B[i,1]$保存了$OPT(i,n)$的值。

- $O(mn)$的时间与$O(m)$的空间
- 但是对于恢复比对本身，这个算法就不可行了

？ 求解比对结构所需的最小空间

$O(m+n)$

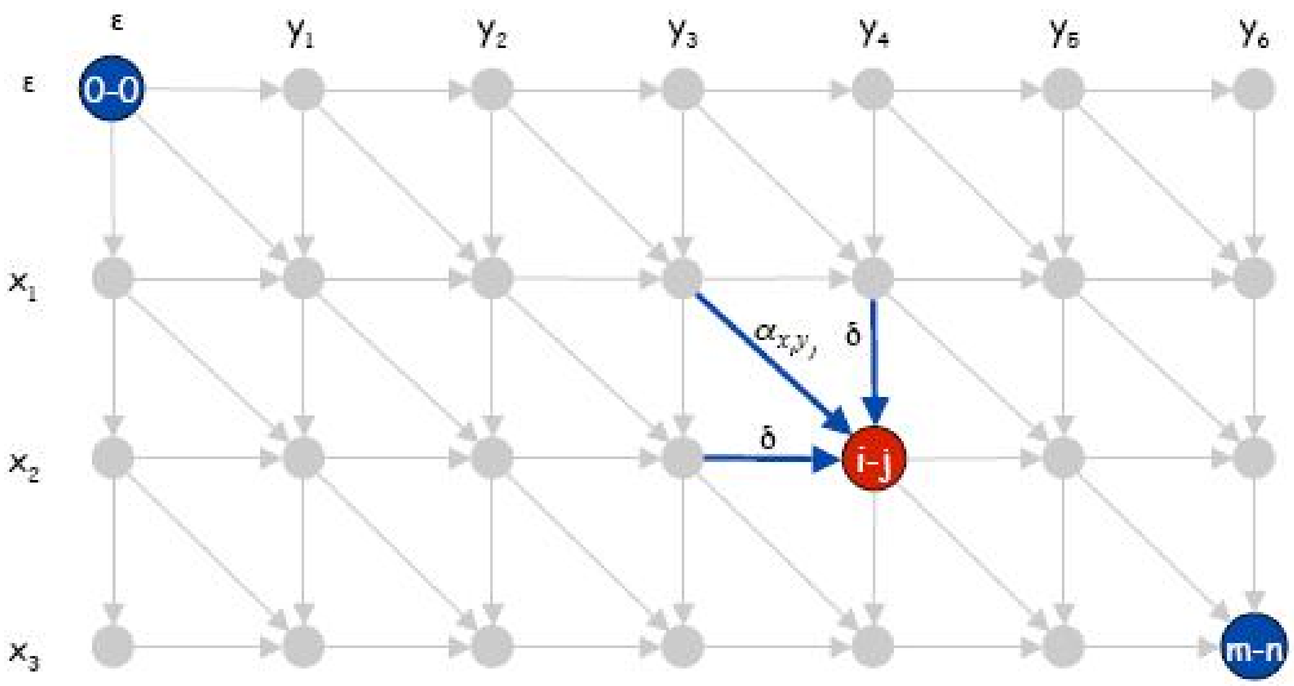
线性空间的序列比对

- 定理. [Hirschberg 1975] 最优序列比对算法可以在 O(m + n) 空间， O(mn)时间复杂度内完成.

  - 可以很聪明的把**动态规划**算法和**分治算法**结合起来

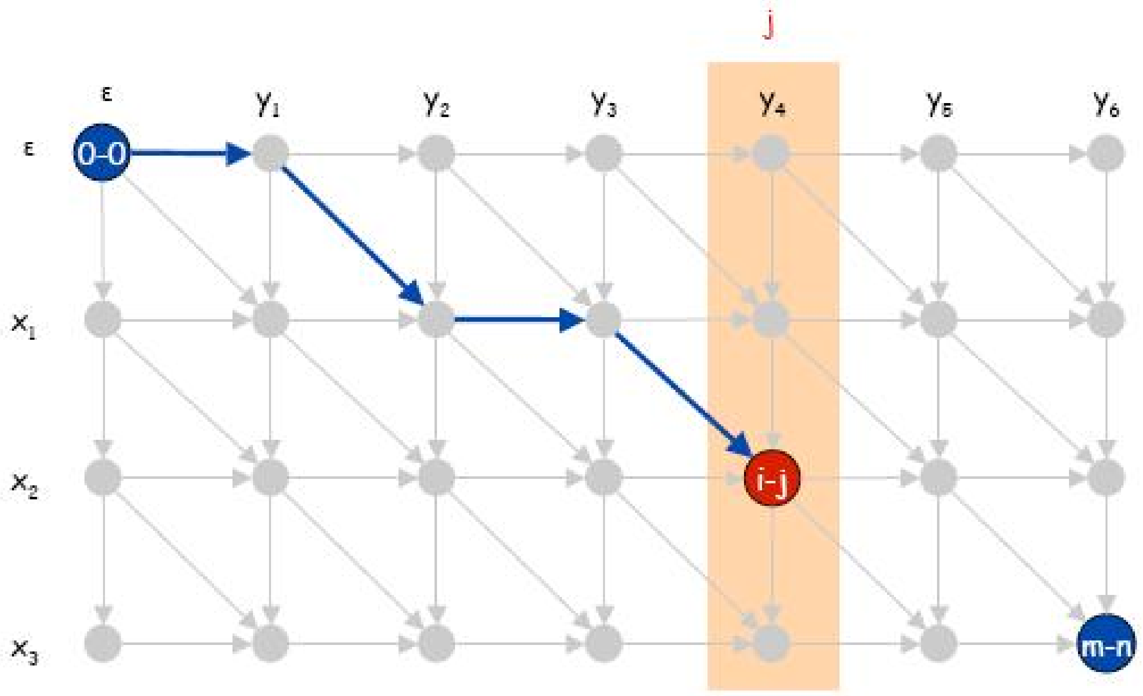
线性空间的序列比对

- 定义f(i,j)表示图G_{XY}中从(0,0)到(i,j)的最短路径的长度，也就是OPT(i,j).



线性空间的序列比对

- 对于图G_{XY}，设 f(i, j) 是从 (0,0) 到(i, j)的最短路径，对于任何的j， 可以在O(mn) 时间和O(m + n)空间代价内计算f (·, j) 。



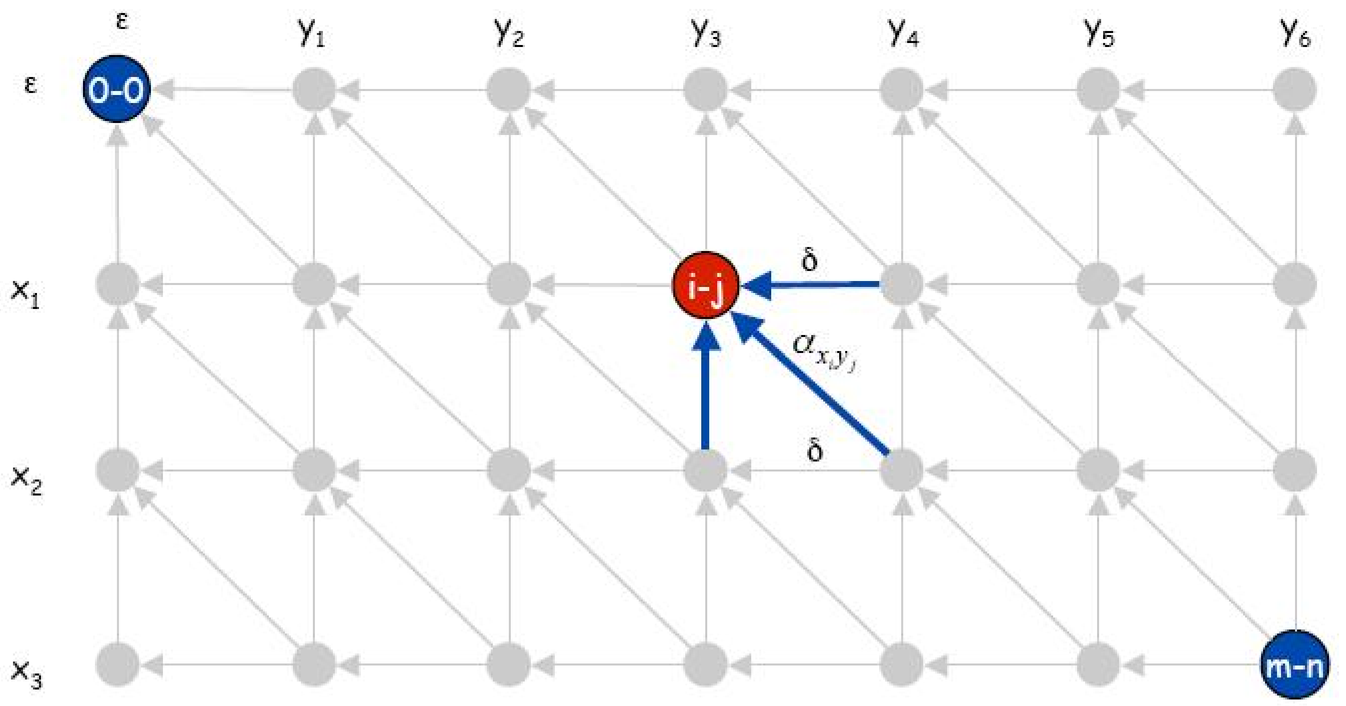
线性空间的序列比对

- 动态规划的逆向公式
- 定义g(i,j)是在G_{XY}中从(i,j)到(m,n)最短路径的长度。
- 关于g有下面的递推式：
- 定理6.18 对于i<m与j<n,我们有

$$g[i,j]=\min[\alpha_{x_{i+1}y_{j+1}}+g(i+1,j+1),\delta+g(i,j+1),\delta+g(i+1,j)].$$

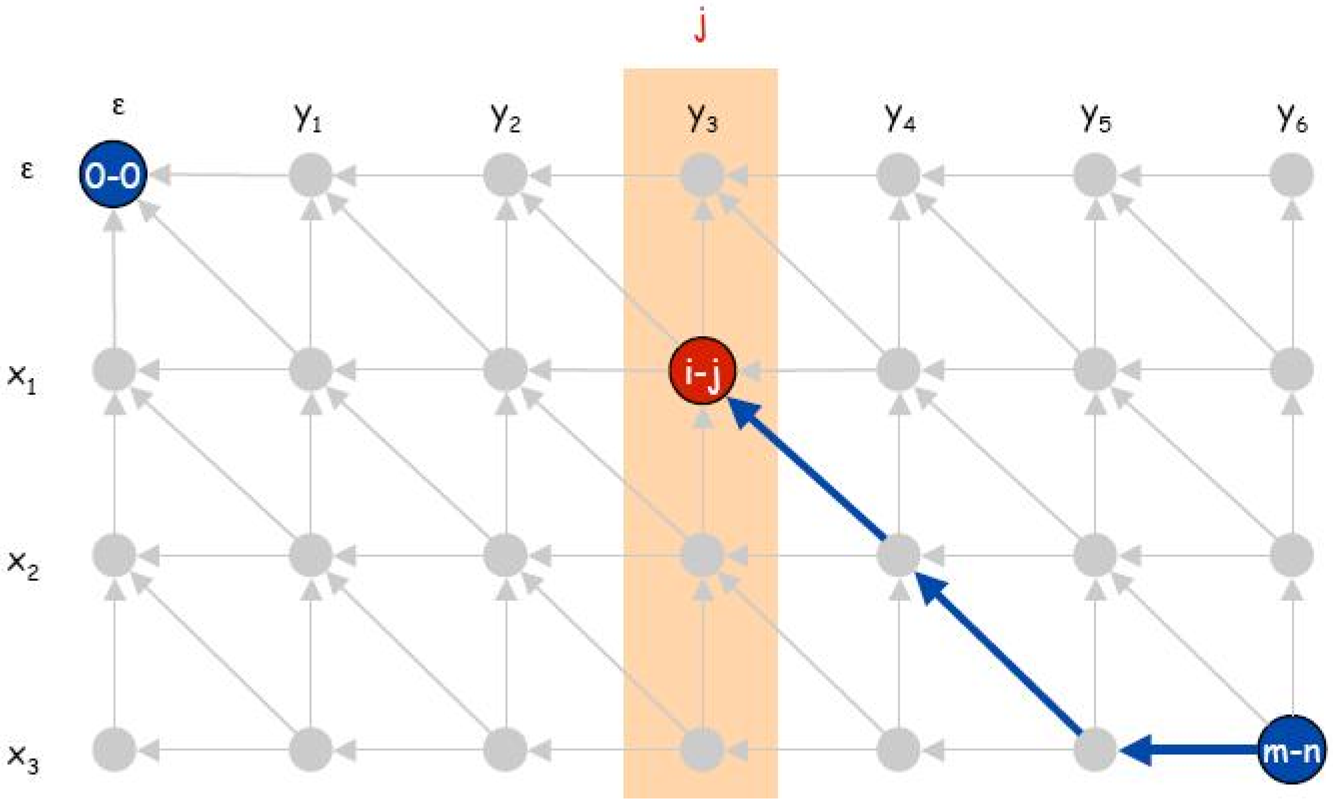
线性空间的序列比对

- 设 g(i, j)是从(i, j) 到(m, n)的最短路径.
- 可以把边的方向反转，把(m, n) 看成“(0, 0) ”,采用类似的计算方法



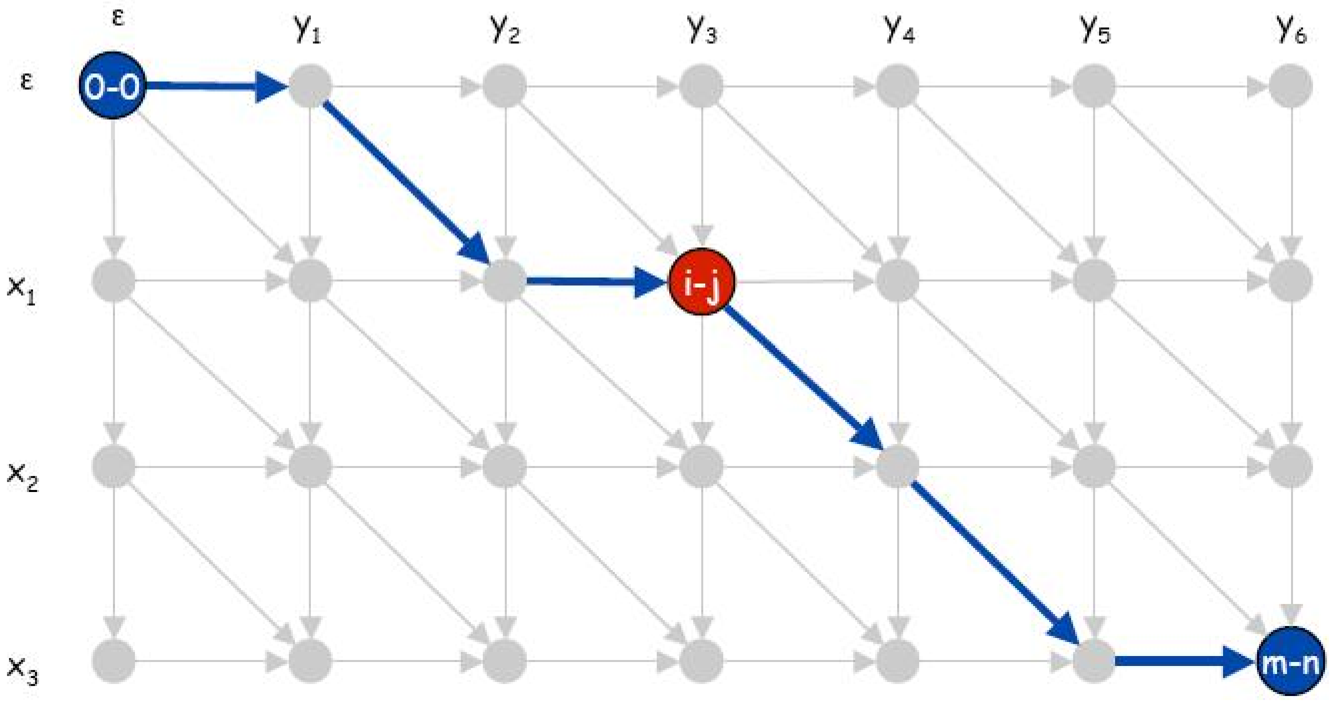
线性空间的序列比对

- 设 g(i, j)是从(i, j) 到(m, n)的最短路径.
- 对于任何的j, 可以在O(mn) 时间， O(m + n) 空间复杂度计算g(·, j).



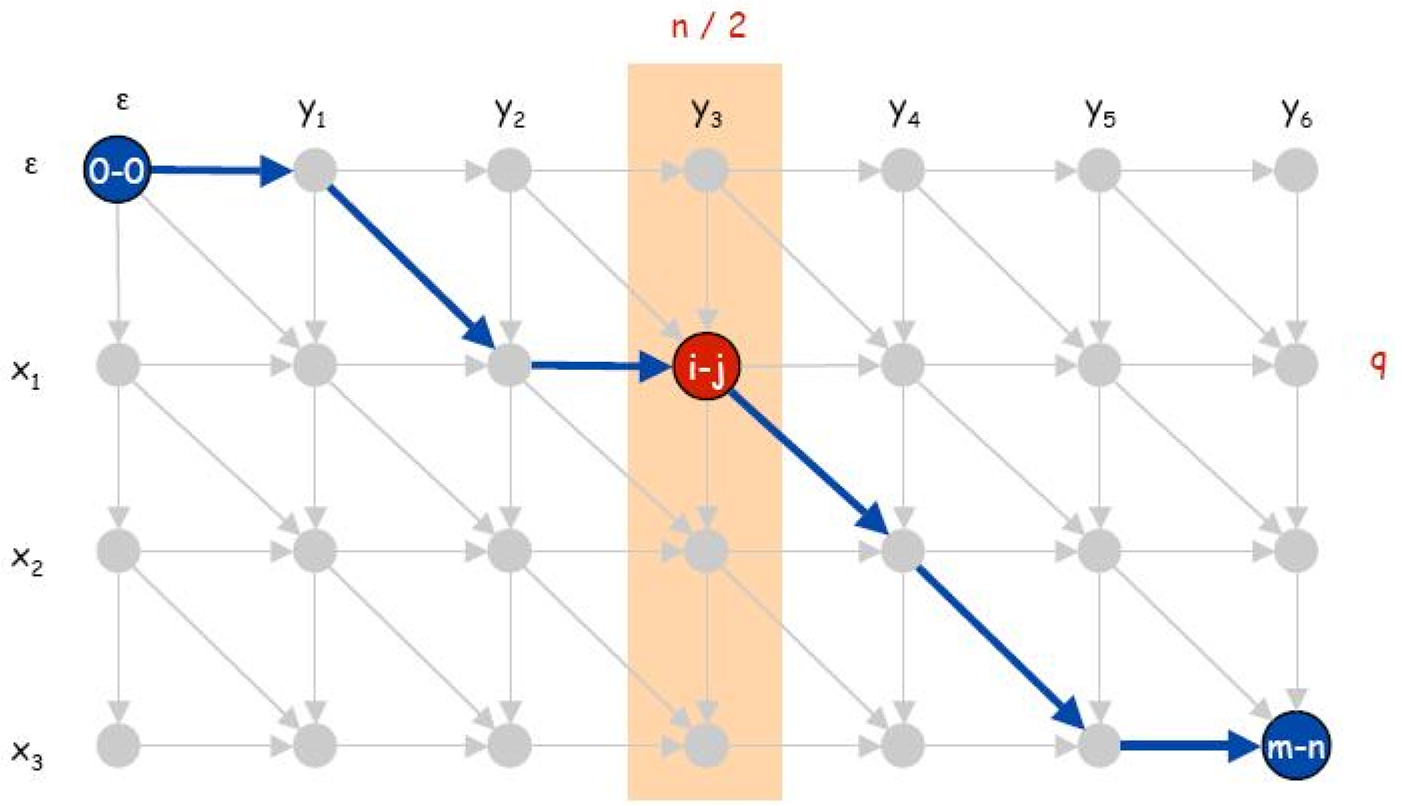
线性空间的序列比对

- 命题6.19 在G_{XY}中通过(i,j)的“角到角”的最短路径的长度是f(i,j)+g(i,j).



线性空间的序列比对

- 定理6.20 令k 表示在{0,…,n}中的任何数， 且令q是使得f(q,k)+g(q,k)达到最小的指标， 那么存在一条通过结点(q,k)的最小长度的角到角的路径。



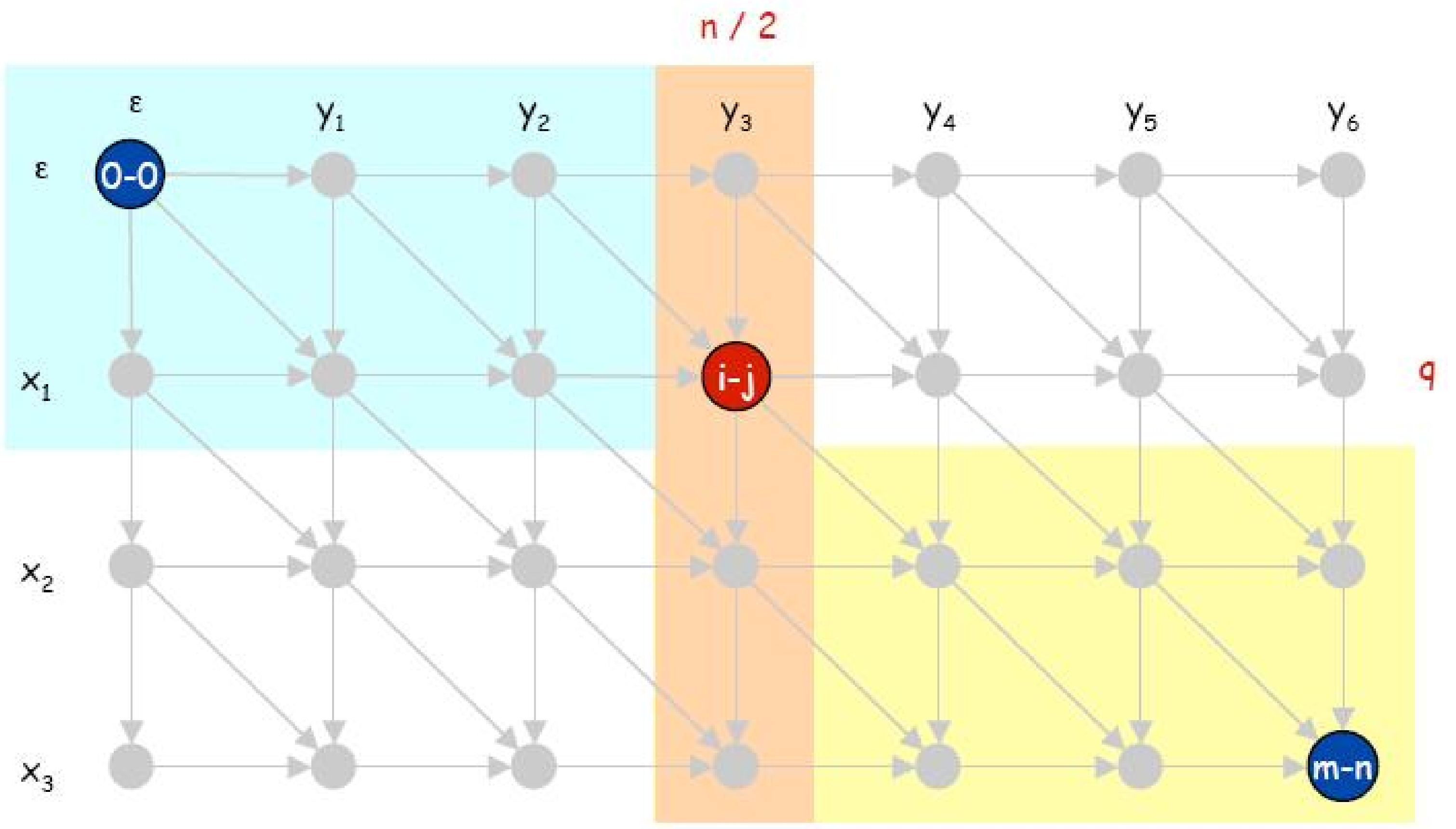
线性空间的序列比对

- 分治策略

- 划分：采用**动态规划算法**，寻找q使得 $f(q, n/2) + g(q, n/2)$最小.
  - 对 x_q and $y_{n/2}$进行比对。

- 处理：*维护全局可访问的表*来保存中间节点

线性空间的序列比对



线性空间的序列比对算法

Divide-and-Conquer-Alignment(X,Y)
  令m是X中的符号个数
  令n是Y中的符合个数
  If m<=1或N<=2, then
    使用Alignment(X,Y)计算最优比对
  调用Space-Efficient-Alignment(X,Y[1:n/2])
  调用Backward-Space-Efficient-Alignment(X,Y[n/2+1:n])
  令q是使得f(q,n/2)+g(q,n/2)达到最小的指标
  把(q,n/2)加到全局表P中
  Divide-and-Conquer-Alignment(X[1:q],Y[1:n/2])
  Divide-and-Conquer-Alignment(X[q:m],Y[n/2:n])
  返回P

分析算法

一个粗糙的估计

- 设T(m, n) 是长度至多为m,n的序列比对算法的最大运行时间。那么T(m, n) = O(mn log n).

$$T(m,n) \le 2T(m, n/2) + O(mn) \Rightarrow T(m,n) = O(mn \log n)$$

分析算法

- 注意到这个估计很大，因为子问题实际上规模是 (q, n/2)，(m - q, n/2).

$$T(m,2) \le cm$$
$$T(2,n) \le cn$$
$$T(m,n) \le cmn + T(q, n/2) + T(m-q, n/2)$$

分析算法

- 定理6.21 算法Divide-and-Conquer在长度为m与n的串上运行时间是O(mn).

证明：采用**待定系数法**式的归纳法。

选择常数c使得

$$
\begin{aligned}
T(m,\ 2) &\le cm \\
T(2,\ n) &\le cn \\
T(m,\ n) &\le cmn + T(q,\ n/2) + T(m-q,\ n/2)
\end{aligned}
$$

初始情形m = 2， n = 2成立

假设T(m', n') ≤ 2cm'n'对m'<m,n'<n成立,那么

$$
\begin{aligned}
T(m,n) &\le T(q,n/2) + T(m-q,n/2) + cmn \\
&\le 2cqn/2 + 2c(m-q)n/2 + cmn \\
&= cqn + cmn - cqn + cmn \\
&= 2cmn
\end{aligned}
$$