

# 6 动态规划

## Dynamic Programming

# 引例：费氏数列

- 费氏数列是由13世纪的意大利数学家、来自Pisa的 Leonardo Fibonacci发现。
- 费氏数列是由0，1开始，之后的每一项等于前两项之和：  
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144.....。
- 这个数列有如下一些特性：
  - 前2个数相加等于第3个数
  - 前1个数除以后一个数越往后越无限接近于0.618 (黄金分割)
  - 相邻的两个比率必是一个小于0.618一个大于0.618
  - 后1个数除以前一个数越往后越无限接近于1.618
  - ...

$$f(n) = \begin{cases} 1 & , \text{ if } n = 1, 2 \\ f(n-1) + f(n-2) & , \text{ if } n \geq 3 \end{cases}$$

递归形式的算法：

```
procedure fib(n)
```

```
  if n=1 or n=2
```

```
    return 1
```

```
  else
```

```
    return fib(n-1)+fib(n-2)
```

优点：



简洁，容易书写以及调试。

缺点：

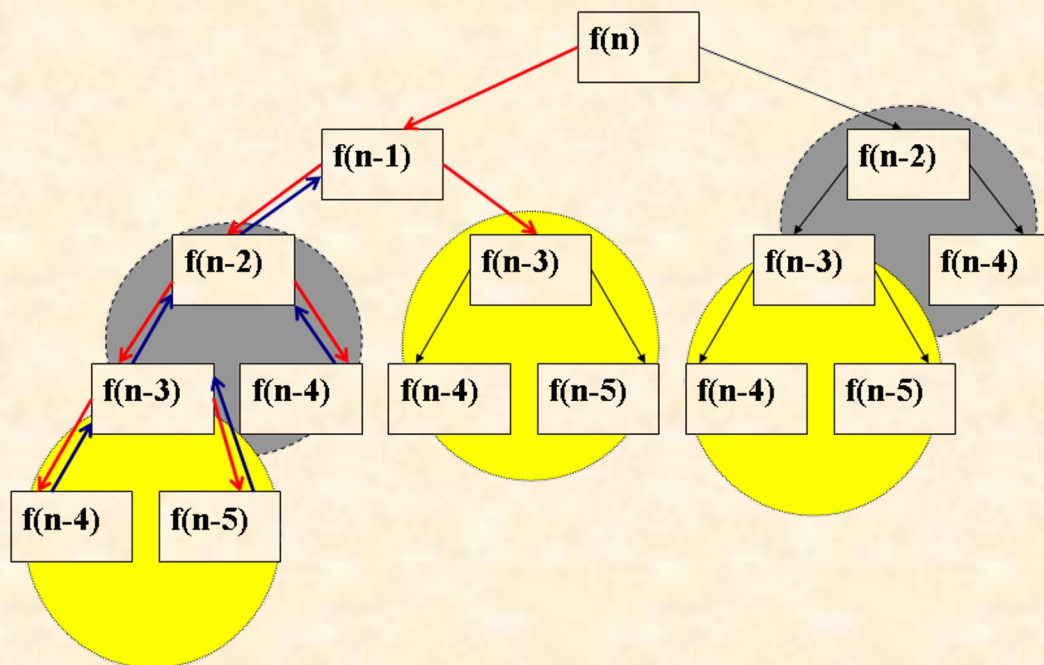


效率低下。

# 为何效率低下？


- 使用直观的方式分析

存在大量重复计算



- 使用时间复杂性的方式分析

$$T(n) = \begin{cases} 1 & \text{if } n = 1, 2 \\ T(n-1) + T(n-2) & \text{if } n \geq 3 \end{cases}$$


$$T(n) \approx \frac{1}{\sqrt{5}} \left( \frac{1 + \sqrt{5}}{2} \right)^n \approx 0.447(1.618)^n$$

即时间复杂度为输入规模的指数形式。当 $n=100$ 时，用递归求解的时间 $T(100) \approx 3.53 \times 10^{20}$ ，若每秒计算 $10^8$ 次，需111,935年！

# 解决方法

- 借助于变量存储中间计算结果，**消除重复计算**。代码片断如下：

```
f1 ← 1  
f2 ← 1  
for i ← 3 to n  
    result ← f1+f2  
    f1 ← f2  
    f2 ← result  
end for  
return result
```

$$T(n) = \Theta(n)$$

# 动态规划的基本思想

- 动态规划的实质是分治和消除冗余，是一种将问题实例分解为更小的、相似的子问题，并存储子问题的解以避免计算重复的子问题，来解决最优化问题的算法策略。
- 基本步骤：
  - 找出最优解的性质，并刻画其结构特征。
  - 递归地定义最优值。
  - 以自底向上的方式计算出最优值。
  - 根据计算最优值时得到的信息，构造最优解。

# 矩阵链相乘

- 给定 $n$ 个连乘的矩阵 $M_1 \cdot M_2 \dots M_{n-1} \cdot M_n$ ，问：所需要的最小乘法次数(最优值)是多少次？对应此最小乘法次数，矩阵是按照什么结合方式相乘(最优解)的？

---

$$(A)_{p \times q} \cdot (B)_{q \times r} \quad \text{所需要的乘法次数为:} \quad p \times r \times q = p \times q \times r$$

---

$$(M_1)_{2 \times 10} \cdot (M_2)_{10 \times 2} \cdot (M_3)_{2 \times 10} \begin{cases} \nearrow (M_1 \cdot M_2) \cdot M_3 & 2 \times 10 \times 2 + 2 \times 2 \times 10 = 80 \\ \searrow M_1 \cdot (M_2 \cdot M_3) & 10 \times 2 \times 10 + 2 \times 10 \times 10 = 400 \end{cases}$$

观察结论：多个矩阵连乘时，相乘的结合方式不同，所需要的乘法次数大不相同。



$M_1 \cdot M_2 \cdot M_3 \cdots M_n$  按照何种结合方式相乘, 所需要的乘法次数最少?

穷举(蛮力)法:

1. 找出所有可能的相乘结合方式;
2. 计算每种相乘结合方式所需要的乘法次数;
3. 求min;

$f(n)$  表示  $n$  个矩阵连乘所有可能的结合方式, 下面设法求出其解析解。

$$\underbrace{(M_1 \cdot M_2 \cdot M_3 \cdots M_k)}_{f(k)} \cdot \underbrace{(M_{k+1} \cdots M_n)}_{f(n-k)}$$

$$f(n) = \sum_{k=1}^{n-1} f(k) \cdot f(n-k)$$

$$f(1) = 1, f(2) = 1, f(3) = 2$$

结论:

穷举法时间复杂度太高。

$$\implies f(n) = \frac{1}{n} C_{2n-2}^{n-1} \xrightarrow{n! \approx \sqrt{2\pi n} (n/2)^n}$$

课本46页

$$f(n) = \frac{(2n-2)!}{n((n-1)!)^2} \approx \frac{4^n}{4\sqrt{\pi n}^{1.5}} \implies f(n) = \Omega\left(\frac{4^n}{n^{1.5}}\right)$$



使用动态规划法:

$$M_1 \cdot M_2 \cdot M_3 \cdots M_n$$

$$r_1, r_2, r_3, \cdots r_n, r_{n+1}$$

$$(M_i)_{r_i \times r_{i+1}} \quad 1 \leq i \leq n$$

$$M_{i,j} = M_i \cdot M_{i+1} \cdots M_{j-1} \cdot M_j$$

$C[i, j]$ : 计算  $M_{i,j}$  所需的最小乘法次数。

$i = 1, j = n$  时, 原问题得解。

$$\underbrace{M_{i,j}}_{C[i,j]} = \underbrace{(M_i \cdot M_{i+1} \cdots M_{k-1})}_{C[i,k-1]} \cdot \underbrace{(M_k \cdot M_{k+1} \cdots M_{j-1} \cdot M_j)}_{C[k,j]}$$

$\longleftarrow k \longrightarrow$

$$C[i, j] = C[i, k-1] + C[k, j] + r_i \cdot r_k \cdot r_{j+1}$$

$$k = i + 1$$

$$C[i, j] = \min_{i < k \leq j} \{C[i, k-1] + C[k, j] + r_i \cdot r_k \cdot r_{j+1}\}$$

$$k = j$$

输入：  $r[1..n+1]$ ，表示  $n$  个矩阵规模的  $n+1$  个整数.

输出：  $n$  个矩阵连乘的最小乘法次数.

```
1. for  $i \leftarrow 1$  to  $n$  {填充对角线  $d_0$ }
2.    $C[i,i] \leftarrow 0$ 
3. end for
4. for  $d \leftarrow 1$  to  $n-1$  {填充对角线  $d_1$  到  $d_{n-1}$ }
5.   for  $i \leftarrow 1$  to  $n-d$  {填充对角线  $d_i$  的每个项目}
6.      $j \leftarrow i+d$  {该对角线上  $j,i$  满足的关系}
7.      $C[i,j] \leftarrow \infty$ 
8.     for  $k \leftarrow i+1$  to  $j$ 
9.        $C[i,j] \leftarrow \min\{ C[i,j], C[i,k-1] + C[k,j] + r_i \times r_k \times r_{j+1} \}$ 
10.    end for
11.  end for
12. end for
13. return  $C[1,n]$ 
```

$$T(n) = \sum_{d=1}^{n-1} \sum_{i=1}^{n-d} \sum_{\substack{j \\ k=i+1}}^j \textcolor{red}{c} = \sum_{d=1}^{n-1} \sum_{i=1}^{n-d} \sum_{\substack{i+d \\ k=i+1}}^{\textcolor{red}{i+d}} c = \sum_{d=1}^{n-1} \sum_{i=1}^{n-d} \sum_{\substack{d \\ k=1}}^d c = \Theta(n^3)$$

$$T(n) = \sum_{d=1}^{n-1} \sum_{i=1}^{n-d} \sum_{\substack{j \\ k=i+1}} c = \sum_{d=1}^{n-1} \sum_{i=1}^{n-d} \sum_{\substack{i+d \\ k=i+1}} c = \sum_{d=1}^{n-1} \sum_{i=1}^{n-d} \sum_{\substack{d \\ k=1}} c$$

$$= \sum_{d=1}^{n-1} \sum_{i=1}^{n-d} cd = c \sum_{d=1}^{n-1} \sum_{i=1}^{n-d} d$$

$$= c \left( \sum_{i=1}^{n-1} 1 + \sum_{i=1}^{n-2} 2 + \sum_{i=1}^{n-3} 3 + \dots + \sum_{i=1}^{n-(n-1)} (n-1) \right)$$

$$= c \left( (n-1) \cdot 1 + (n-2) \cdot 2 + (n-3) \cdot 3 + \dots + (n-(n-1)) \cdot (n-1) \right)$$

$$= c \left( n \cdot 1 + n \cdot 2 + n \cdot 3 + \dots + n \cdot (n-1) - 1 \cdot 1 - 2 \cdot 2 - (n-1) \cdot (n-1) \right)$$

$$= c \left( n(1 + 2 + 3 + \dots + (n-1)) - \sum_{k=1}^{n-1} k^2 \right)$$

$$= c \left( n \cdot \frac{n(n-1)}{2} - \frac{1}{6} (n-1)n(2n-1) \right)$$

$$= \frac{1}{6} (cn^3 - cn) = \Theta(n^3)$$

蛮力方法:

$$T(n) = \Omega\left(\frac{4^n}{n^{1.5}}\right)$$

# 一个实例

$$(M_1)_{5 \times 10} \cdot (M_2)_{10 \times 4} \cdot (M_3)_{4 \times 6} \cdot (M_4)_{6 \times 10} \cdot (M_5)_{10 \times 2}$$

$$r_1 = 5, r_2 = 10, r_3 = 4, r_4 = 6, r_5 = 10, r_6 = 2$$

$d = 0$

$d = 1$

$d = 2$

$d = 3$

$d = 4$

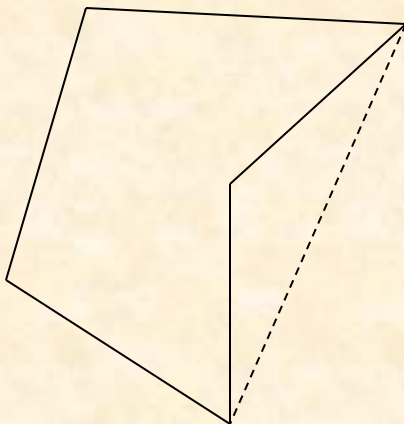
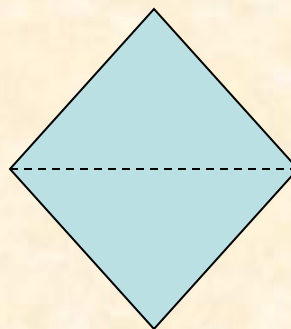
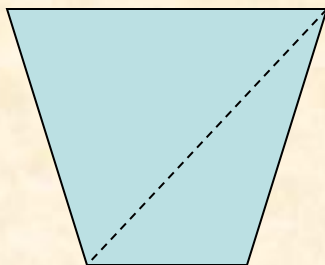
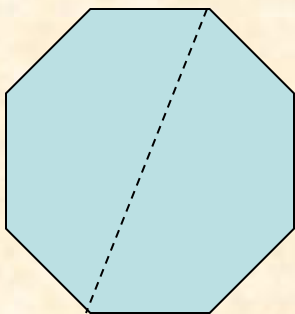
$C[1,1]=0$ ( $M_1$ )	$C[1,2]=200$ ( $M_1 M_2$ )	$C[1,3]=320$	$C[1,4]=620$	$C[1,5]=348$
	$C[2,2]=0$ ( $M_2$ )	$C[2,3]=240$ ( $M_2 M_3$ )	$C[2,4]=640$ ( $M_2$ ) ( $M_3 M_4$ )	$C[2,5]=248$
		$C[3,3]=0$ ( $M_3$ )	$C[3,4]=240$ ( $M_3 M_4$ )	$C[3,5]=168$
			$C[4,4]=0$ ( $M_4$ )	$C[4,5]=120$ ( $M_4 M_5$ )
				$C[5,5]=0$ ( $M_5$ )

$$C[2,4] = \min_{2 \leq k \leq 4} \{C[2, k-1] + C[k, 4] + r_2 \cdot r_k \cdot r_{4+1}\}$$

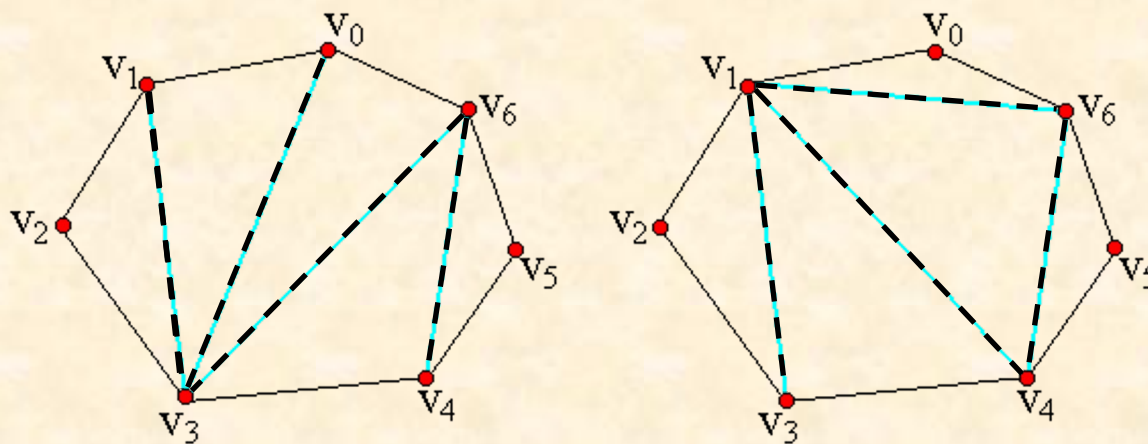
$$\begin{aligned} k=3 &\rightarrow C[2,4] = C[2,2] + C[3,4] + r_2 \cdot r_3 \cdot r_{4+1} = 0 + 240 + 10 \cdot 4 \cdot 10 = 640 \rightarrow (M_2) \cdot (M_3 \cdot M_4) \\ k=4 &\rightarrow C[2,4] = C[2,3] + C[4,4] + r_2 \cdot r_4 \cdot r_{4+1} = 240 + 0 + 10 \cdot 6 \cdot 10 = 840 \rightarrow (M_2 \cdot M_3) \cdot (M_4) \end{aligned} \} \min$$

# 平面凸多边形最优三角划分

- 平面多边形
  - 由在同一平面上，且不在同一直线上的多条线段首尾顺次连结且不相交所组成的图形称为平面多边形。
- 平面凸多边形
  - 弦：连接平面多边形的任意两个不同顶点的线段。
  - 平面凸多边形：如果一个平面多边形的任意一条弦，要么在该多边形的内部，要么恰好为该多边形的边，那么，称该平面多边形为凸的。否则，称该平面多边形为凹的。
- 三角划分
  - 将平面凸多边形分割成互不相交的三角形。



平面凸多边形的表示：用平面凸多边形顶点的逆时针序列表示凸多边形，即 $P=\{v_0, v_1, \dots, v_n\}$ 表示具有 $n+1$ 条边的平面凸多边形。



1. 给定一个平面凸多边形 $P$ ，其三角划分不是唯一的。
2. 给定平面凸多边形 $P$ ，对于 $P$ 的每种三角划分，可以定义一个权函数(例如：三角划分中所有三角形的边长之和)。
3. 最优三角划分：使得权函数取最小值的三角划分。

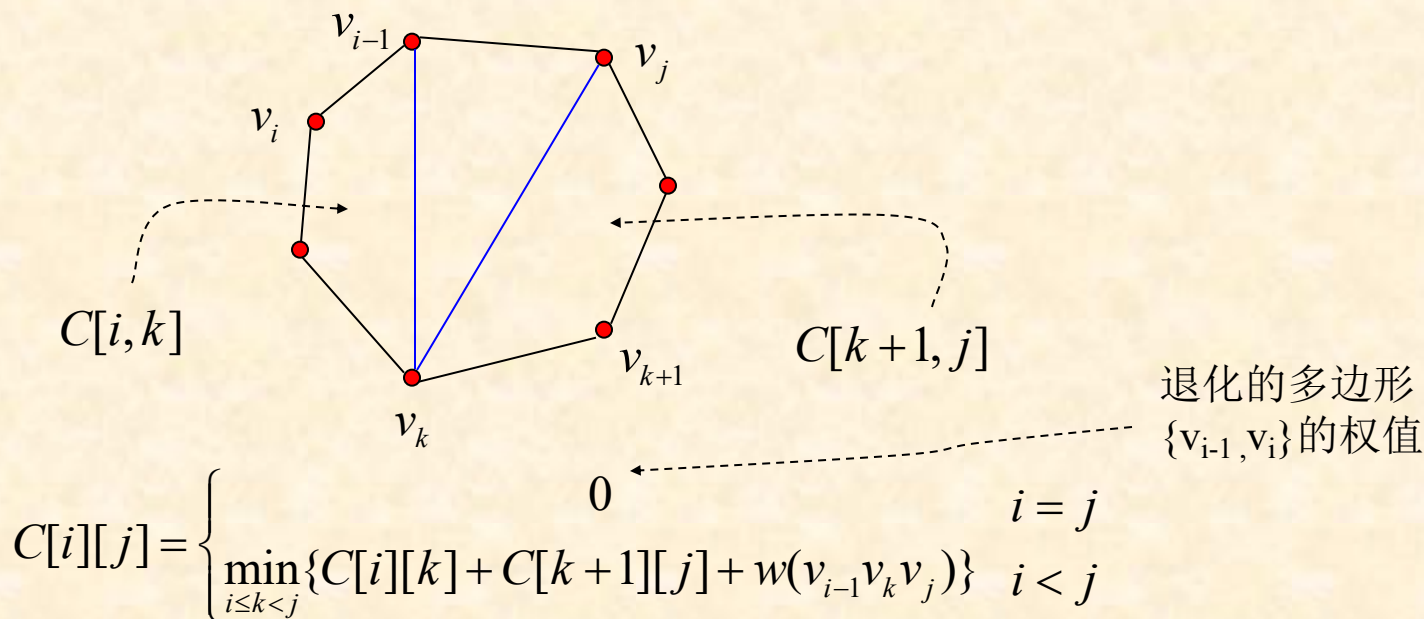


问题：给定平面凸多边形 $P=\{v_0, v_1, \dots, v_n\}$ ，求： $P$ 的最优三角划分。  
 (不妨假设：权函数定义为三角划分中所有三角形的边长之和)  
 求解目标：权函数的最小值，以及对应该最小值的三角划分方式。

↓ 类比

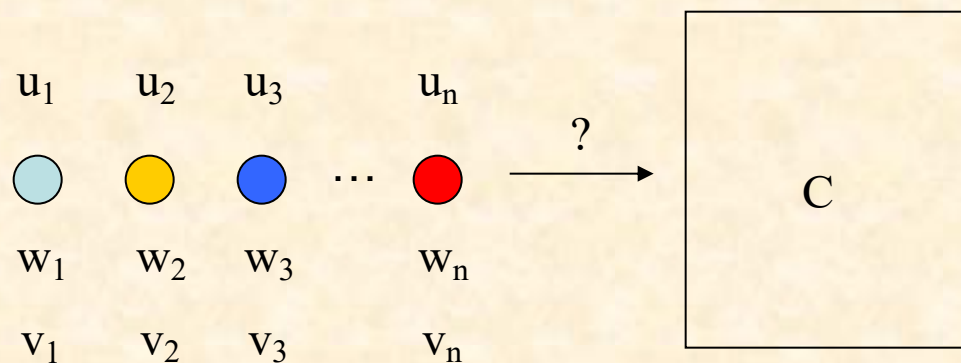
矩阵连乘：最小乘法次数，以及对应该最小乘法次数的矩阵结合方式。

若 $P=\{v_0, v_1, \dots, v_n\}$ 是一个凸多边形，那么 $\{v_{i-1}, v_i, \dots, v_j\}$ 所构成的必定也是一个凸多边形。定义 $C[i, j]$ 为子凸多边形 $\{v_{i-1}, v_i, \dots, v_j\}$ 的最优三角划分所对应的权函数值，即其最优值。



# 0-1背包问题

- 给定 $n$ 个物品  $\{u_1, u_2, \dots, u_n\}$  和一个背包，物品 $i$  的重量为  $w_i$ ，价值为  $v_i$ ，已知背包的承重量为  $C$ 。问：在不撑破背包的条件下，选择哪些物品装入背包，得到的总价值最大？
- 之所以称为0-1 背包问题，是因为一个物品要么装入、要么不装入，这两种状态分别用1 和0 表示。



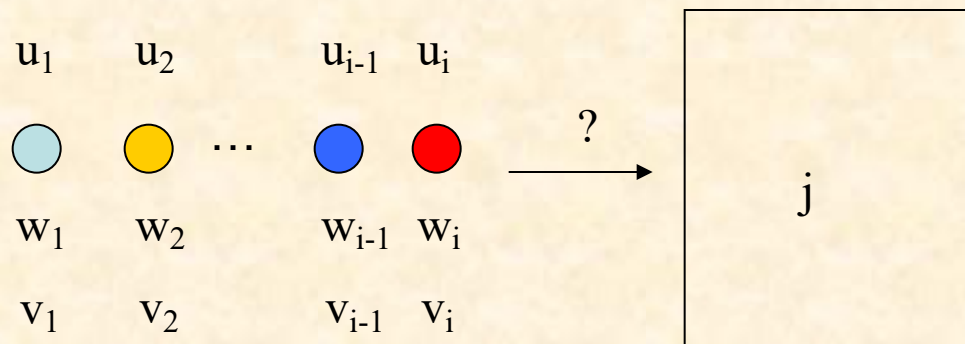
0-1背包问题的形式化描述:

给定 $C > 0$ ,  $w_i > 0$ ,  $v_i > 0$ ,  $1 \leq i \leq n$ , 找出一个 $n$  元的0-1 向量 $(x_1, x_2, \dots, x_n)$ ,  $x_i \in \{0, 1\}$ ,  $1 \leq i \leq n$ , 求如下优化问题:

$$\max \sum_{i=1}^n v_i x_i$$

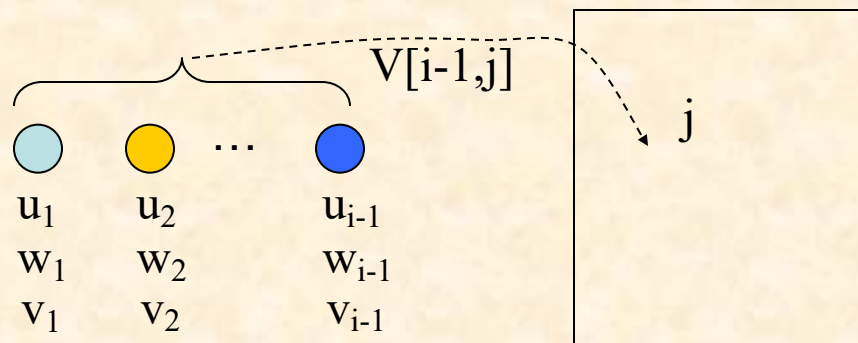
$$s.t. \quad \sum_{i=1}^n w_i x_i \leq C, \quad x_i \in \{0, 1\}, 1 \leq i \leq n$$

设 $V[i,j]$ 表示从前 $i$ 个物品 $\{u_1, u_2, \dots, u_i\}$ 中取出一部分装入承重量为 $j$ 的背包所能取得的最大价值。那么，当 $i=n, j=C$ 时， $V[n,C]$ 就是原问题的解。

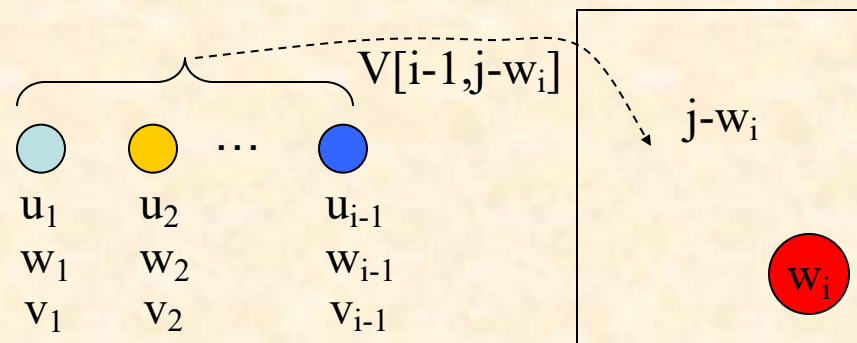


$$V[i, j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ V[i-1, j] & \text{if } j < w_i \\ \max\{V[i-1, j], V[i-1, j-w_i] + v_i\} & \text{if } i > 0 \text{ and } j \geq w_i \end{cases}$$

Case 1:



Case 2:



输入：物品集合  $\{u_1, u_2, \dots, u_n\}$ ，重量分别为  $w_1, w_2, \dots, w_n$ ，价值分别为  $v_1, v_2, \dots, v_n$ ，承重量为  $C$  的背包

输出：背包所能装物品的最大价值

```
1. for  $i \leftarrow 0$  to  $n$ 
2.  $V[i, 0] \leftarrow 0$ 
3. end for
4. for  $j \leftarrow 0$  to  $C$ 
5.  $V[0, j] \leftarrow 0$ 
6. end for
7. for  $i \leftarrow 1$  to  $n$  //前 $i$ 个物品
8.  for  $j \leftarrow 1$  to  $C$  //承重量 $C$ 与物品重量 $w_i$ 均为整数，故 $j$ 为整数
9.     $V[i, j] \leftarrow V[i-1, j]$ 
10.   if  $w_i \leq j$  then  $V[i, j] \leftarrow \max\{V[i, j], V[i-1, j-w_i] + v_i\}$ 
11.   end if
12. end for
13. end for
14. return  $V[n, C]$ 
```

$$T(n) = \Theta(nC)$$

## 一个实例

背包的承重量为 $C=9$ ；给定4个物品，重量( $w$ )分别为2, 3, 4, 5；价值( $v$ )依次为3, 4, 5, 7。问：背包中最多能装的物品的总价值是对少？

	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	0	3	3	3	3	3	3	3	3
2	0	0	3	4	4	7	7	7	7	7
3	0	0	3	4	5	7	8	9	9	12
4	0	0	3	4	5	7	8	10	11	12

因为 $w_3=4 \leq j=7$ ;

所以 $V[3, 7] = \max\{V[3-1, 7], V[3-1, 7-w_3]+v_3\}$

$= \max\{V[2, 7], V[2, 3]+5\} = \max\{7, 4+5\} = 9$

# 最长公共子序列问题

- 给定两个定义在字符集 $\Sigma$ 上的字符串A和B，长度分别为n和m，现在要求它们的最长公共子序列的长度值(最优值)，以及对应的子序列(最优解)。
- 子序列

$A = a_1 a_2 \cdots a_n$  的一个子序列是形如下式的一个字符串： $a_{i_1} a_{i_2} \cdots a_{i_k}$ ，其中  
 $1 \leq i_1 < i_2 < \cdots < i_k \leq n$

例如： $A = zxy$  子序列可以是：“”，z, x, y, zx, zy, xy, zxy。

但是xz, yz, xyz不是它的子序列。

$$C_n^0 + C_n^1 + \cdots + C_n^n = 2^n$$



- 穷举法(Brute-Force):
  - 找出A字符串所有可能的子序列( $2^n$ );
  - 对于A的每一个子序列, 判断其是否是B的一个子序列,需要的时间为 $\Theta(m)$ ;
  - 求max;总的时间为 $\Theta(m 2^n)$ .

$$A = a_1 a_2 \cdots a_n \quad B = b_1 b_2 \cdots b_m$$


---

$$\left. \begin{array}{l} a_1 a_2 \cdots a_{i-1} a_i \\ b_1 b_2 \cdots b_{j-1} b_j \end{array} \right\} \xrightarrow{\text{最长公共子序列的长度值}} C[i, j]$$

$$C[i, j] = \begin{cases} 0 & \text{if } i == 0 \text{ or } j == 0 \\ C[i-1, j-1] + 1 & \text{if } i > 0 \& j > 0 \& a_i == b_j \\ \max \{C[i, j-1], C[i-1, j]\} & \text{if } i > 0 \& j > 0 \& a_i \neq b_j \end{cases}$$

$$\begin{array}{l} \underline{a_1 a_2 \cdots a_{i-1} a_i} \\ \underline{b_1 b_2 \cdots b_{j-1} b_j} \end{array} \Rightarrow C[i, j-1]$$

$$\begin{array}{l} \underline{a_1 a_2 \cdots a_{i-1} a_i} \\ \underline{b_1 b_2 \cdots b_{j-1} b_j} \end{array} \Rightarrow C[i-1, j]$$

}  $\xrightarrow{\text{max}}$

输入：两个字符串A, B, 长度分别为n, m.

输出：X和Y的最长公共子序列长度.

```
1.  for i ← 0 to n
2.    C[i,0] ← 0
3.  end for
4.  for j ← 0 to m
5.    C[0,j] ← 0
6.  end for
7.  for i ← 1 to n
8.    for j ← 1 to m
9.      if  $a_i = b_j$  then  $C[i, j] \leftarrow C[i-1, j-1] + 1$ 
10.     else  $C[i, j] \leftarrow \max\{C[i, j-1], C[i-1, j]\}$ 
11.     end if
12.    end for
13.  end for
14. return C[n,m]
```

$$T(n) = \Theta(nm)$$

一个实例：

$$C[i, j] = \begin{cases} 0 & \text{if } i=0 \text{ or } j=0 \\ C[i-1, j-1] + 1 & \text{if } i>0 \& j>0 \& a_i == b_j \\ \max\{C[i, j-1], C[i-1, j]\} & \text{if } i>0 \& j>0 \& a_i \neq b_j \end{cases}$$

$A = xyxxz \quad B = zxzyyz$

使用一个 $(n+1) \times (m+1)$ 的表格来进行计算。  
逐行填满表格，问题得解。

	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	0	1	1	1	1	1
2	0	0	1	1	2	2	2
3	0	0	1	1	2	2	2
4	0	0	1	1	2	2	2
5	0	1	1	2	2	2	3

因为 $a_3 \neq b_4$ ，所以 $C[3, 4] = \max\{C[3, 3], C[2, 4]\} = \max\{1, 2\} = 2$

因为 $a_5 == b_6$ ，所以 $C[5, 6] = C[4, 5] + 1 = 2 + 1 = 3$

# 高性能计算机任务分配问题（二次动态规划）

## 问题描述：

现在有一项大型计算任务需要完成。这项大型计算任务包含  $n_A$  个 **A 类子任务** 和  $n_B$  个 **B 类子任务**；这些子任务**互不相关**，并且每个 A 类子任务的工作量都是一样的，每个 B 类子任务的工作量也是一样的(每个 A 和 B 类子任务的工作量不一定相同)。各个子任务之间没有执行顺序上的要求。

假设超级计算机拥有  $p$  个计算节点，每个节点都包括一个**串行处理器**、**本地主存**和**高速 cache**。然而，由于常年使用和不连贯的升级，各个计算节点的**计算能力并不一样**。

一个节点的**计算能力**包括如下几个方面：每个节点都有三种工作状态：**待机状态**、**A 类状态**和**B 类状态**。其中，执行 A 类任务时为 A 类状态；执行 B 类任务时为 B 类状态；待机状态下不执行计算。所有的处理器在开始工作之前都处于待机状态，而从其它的状态转入 A 类状态或 B 类状态(包括 A 类状态和 B 类状态之间相互转换)，都要花费一定的启动时间，并且对于不同的处理节点，这个状态转换启动时间不一定相同。用两个正整数  $t_i^A$  和  $t_i^B (i=1,2,\dots,p)$  分别表示计算节点  $i$  转入 A 类状态和 B 类状态所需要的启动时间(单位：ns)。

一个节点在连续处理同一类任务的时候，执行时间(不含状态转换的时间)随任务量(这一类子任务的数目)的平方增长，即：

若节点 $i$ 连续处理 $x$ 个 A 类子任务，则对应的执行时间为： $t=k_i^A \cdot x^2$ ；

类似地，若节点 $i$ 连续处理 $x$ 个 B 类子任务，对应的执行时间为： $t=k_i^B \cdot x^2$ 。

其中， $k_i^A$ 和 $k_i^B$ 是系数，单位是 ns， $i=1,2,\dots,p$ 。

任务分配必须在所有计算开始之前完成。所谓任务分配，即给每个计算节点设置一个任务队列，队列由一串 A 类和 B 类子任务组成，两类子任务可以交错排列。

计算开始后，各计算节点分别从各自的子任务队列中顺序读取计算任务并执行，队列中连续的同类子任务将由该计算节点一次性读出，队列中一串连续的同类子任务不能被分成两部分执行。



### 编程任务:

现在需要你编写程序，给这  $p$  个节点安排计算任务，使得这个工程计算任务能够尽早完成。假定任务安排好后不再变动，而且所有的节点都同时开始运行，任务安排的目标是使最后结束计算的节点的完成时间尽可能早。

### 输入输出:

输入文件名是 `hpc.in`。

文件的第一行是对计算任务的描述，包括两个正整数  $n_A$  和  $n_B$ ，分别是 A 类和 B 类子任务的数目，两个整数之间由一个空格隔开。

文件的后面部分是对此计算机的描述：

文件第二行是一个整数  $p$ ，即计算节点的数目。

随后连续的  $p$  行按顺序分别描述各个节点的信息，第  $i$  个节点由第  $i+2$  行描述，该行包括下述四个正整数（相邻两个整数之间有一个空格）： $t_i^A$   $t_i^B$   $k_i^A$   $k_i^B$

### 输出文件:

输出文件名是 `hpc.out`。其中只有一行，包含有一个正整数，即从各节点开始计算到任务完成所用的时间。

### 样例:

设输入文件 `hpc.in` 为

```
5 5
3
15 10 6 4
70 100 7 2
30 70 1 6
```

对应的输出文件 `hpc.out` 为

93

数据说明:

$$1 \leq n_A \leq 60, 1 \leq n_B \leq 60$$

$$1 \leq p \leq 20$$

$$1 \leq t_A \leq 1000, 1 \leq t_B \leq 1000, 1 \leq k_A \leq 50, 1 \leq k_B \leq 50$$

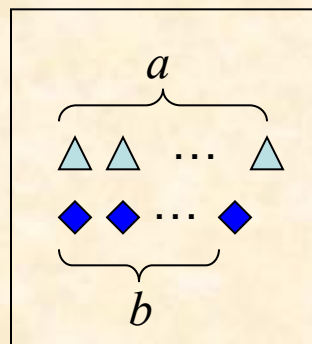


$(n_A, n_B)$

△ - 1份A类子任务

◆ - 1份B类子任务

任务  $(a, b)$



计算节点



第i个节点所需  
最短计算时间:

$f_i(a, b) \rightarrow ?$

$p=1$

$f_1(n_A, n_B)$

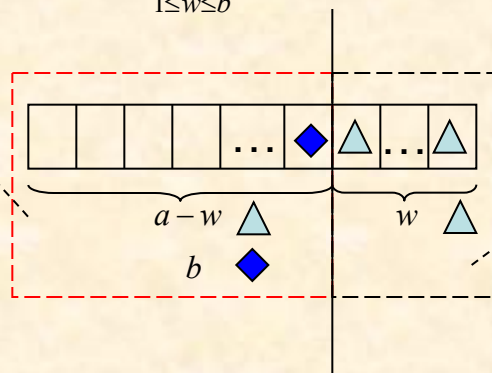
分析:节点i完成所分配的任务时,最后所完成的子任务不是A类子任务就是B类子任务.

$P_i(a, b, 0)$  — 节点i完成任务(a,b)所花的最短时间,  
(最后完成的一项子任务是A类子任务)

$P_i(a, b, 1)$  — 节点i完成任务(a,b)所花的最短时间,  
(最后完成的一项子任务是B类子任务)

$$P_i(a, b, 0) = \min_{1 \leq w \leq a} \{P_i(a-w, b, 1) + t_i^A + k_i^A \cdot w^2\}$$

$$P_i(a, b, 1) = \min_{1 \leq w \leq b} \{P_i(a, b-w, 0) + t_i^B + k_i^B \cdot w^2\}$$



$$f_i(a, b) = \min \{P_i(a, b, 0), P_i(a, b, 1)\}$$

边界值:

$$\begin{cases} P_i(a,b,0) = 0 \\ P_i(a,b,1) = 0 \end{cases}, a = 0 \wedge b = 0$$

$$\begin{cases} P_i(a,b,0) = \infty \\ P_i(a,b,1) = t_i^B + k_i^B \cdot b^2 \end{cases}, a = 0 \wedge b \neq 0$$

$$\begin{cases} P_i(a,b,0) = t_i^A + k_i^A \cdot a^2 \\ P_i(a,b,1) = \infty \end{cases}, a \neq 0 \wedge b = 0$$

$\infty$ 表示无穷大, 表示此情形是不可能发生的。

现在我们已经知道节点*i*完成任务(a,b)所需要的最短时间为 $f_i(a,b)$ , 下面的问题就变成, 有  $(n_A, n_B)$  项任务, 有*p*个节点可以并行地完成该任务, 每个节点完成任务(a,b)所需要的最短时间为 $f_i(a,b)$ 已知, 如何分配任务使得完成全部任务的时间最短。这样这个问题已经变成一个很简单且经典的动态规划问题了(类似于背包问题)。

设 $C_i(a,b)$ 表示将任务(a,b)分配给前*i*个节点, 并行完成这些任务所需要的最短的时间。则原来题目所求就是 $C_p(n_A, n_B)$ 。我们可以写出递归式:

$$C_i(a,b) = \min \{ \max \{ C_{i-1}(a - w_1, b - w_2), f_i(w_1, w_2) \} \}$$

其中:  $0 \leq w_1 \leq a, 0 \leq w_2 \leq b, (w_1 \neq 0 \vee w_2 \neq 0) \wedge (w_1 \neq a \vee w_2 \neq b)$

$$C_i(a,b) = \min\{\max\{C_{i-1}(a-w_1, b-w_2), f_i(w_1, w_2)\}\}$$

解释:如果从1到i的节点(前i个节点)要完成的任务总数是(a,b)的话, 我们可以分配给第i个节点( $w_1, w_2$ )的任务, 它所需的最短时间为 $f_i(w_1, w_2)$ , 这在前面已经求出来了; 而前1到i-1个节点总共要完成任务( $a-w_1, b-w_2$ ), 所需最短时间为 $C_{i-1}(a-w_1, b-w_2)$ 。因为是所有的节点是并行地完成任务, 所以从1到i的节点完成任务(a,b)所需的最短时间就是:

$\max\{C_{i-1}(a-w_1, b-w_2), f_i(w_1, w_2)\}$ 。

$$0 \leq w_1 \leq a, 0 \leq w_2 \leq b, (w_1 \neq 0 \vee w_2 \neq 0) \wedge (w_1 \neq a \vee w_2 \neq b)$$

节点i上不能什么任务也不分配
不能将所有任务分配给节点i

解释: $w_1$ 可以取0到a,  $w_2$ 可以取0到b, 但是 $w_1, w_2$ 不能同时取上界a,b, 也不能同时取下界0.

**整数划分问题：**将正整数 $K$ 表示为 $p$ 个整数的和：

$$K=k_1+k_2+\dots+k_p, \text{ 其中 } (k_1 \geq k_2 \geq \dots \geq k_p)$$

正整数 $K$ 的这种表示叫做正整数划分(partition)。

正整数 $K$ 的不同的划分的个数叫做的划分数，记做 $p(K)$ 。

比如对于6，可以进行如下不同的划分：

6; -----最大加数为6

5+1; -----最大加数为5

4+2; 4+1+1; -----最大加数为4

3+3; 3+2+1; 3+1+1+1; -----最大加数为3

2+2+2; 2+2+1+1; 2+1+1+1+1; -----最大加数为2

1+1+1+1+1+1 -----最大加数为1

因此， $p(K)=11$ 。也就是说，6的划分数是11。

请设计一个算法，对于一个给定的正整数(输入)，求出该整数的划分数(输出)。

分析  $K=k_1+k_2+\dots+k_p$ , 其中  $(k_1 \geq k_2 \geq \dots \geq k_p)$

比如对于6, 可以进行如下不同的划分:

6;	-----最大加数为6
5+1;	-----最大加数为5
4+2; 4+1+1;	-----最大加数为4
3+3; 3+2+1; 3+1+1+1;	-----最大加数为3
2+2+2; 2+2+1+1; 2+1+1+1+1;	-----最大加数为2
1+1+1+1+1+1	-----最大加数为1

- 在正整数 $K$ 的所有划分中, 我们将最大加数 $(k_1)$  不大于(小于或等于)  $m$ 的划分数记做 $p(K,m)$
- 那么当 $m=K$ 时, 就是问题的解。
- 例如 $p(6,1)=1$ ;  $p(6,2)=4$ ;  $p(6,3)=7$ ;  $p(6,4)=9$ ;  $p(6,5)=10$ ;  $p(6,6)=11$

6;	-----最大加数为6
5+ <u>1</u> ;	-----最大加数为5
4+ <u>2</u> ; 4+ <u>1+1</u> ;	-----最大加数为4
3+ <u>3</u> ; 3+ <u>2+1</u> ; 3+ <u>1+1+1</u> ;	-----最大加数为3    m
2+ <u>2+2</u> ; 2+ <u>2+1+1</u> ; 2+ <u>1+1+1+1</u> ;	-----最大加数为2
1+ <u>1+1+1+1+1+1</u>	-----最大加数为1

$$p(K, m) = \begin{cases} \text{-----}, & \text{if } (m = 1) \\ \text{-----}, & \text{if } (m > K) \\ \text{-----}, & \text{if } (m = K) \\ \text{-----}, & \text{if } (1 < m < K) \end{cases}$$



那么，可以按照如下关系来建立 $p(K,m)$ 的递归关系式：

(1): 当 $m=1$ 时(最大加数不大于1的时候),  $p(K,1)=1$ 只有1种划分形式, 也即 $1+1+\dots+1+1$ 。

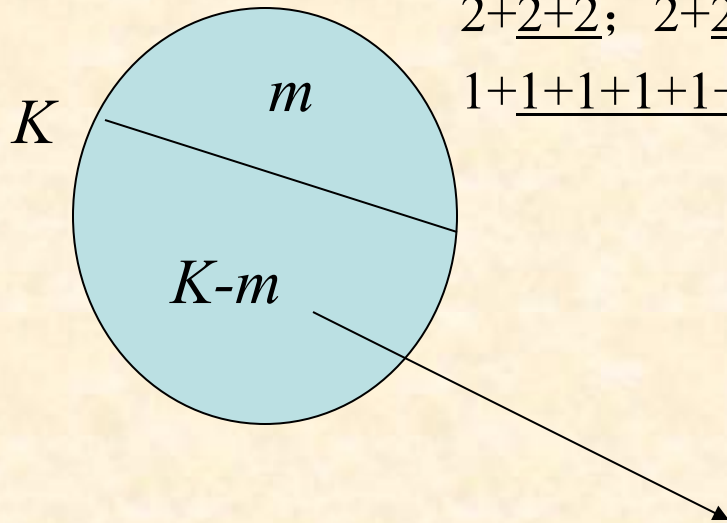
(2): 当 $m>K$ 时, 等价于 $m=K$ 。即 $p(K,m)=p(K,K)$ 。

(3): 当 $m=K$ 的时候, 正整数 $K$ 的划分数 = 最大加数为 $K$ 的划分数(1) + 最大加数不大于 $K-1$ 的划分数( $p(K,K-1)$ )。

(4): 当 $K>m>1$ 的时候, 正整数 $K$ 的划分数 = 最大加数为 $m$ 的划分数(??) + 最大加数不大于 $m-1$ 的划分数( $p(K,m-1)$ )



6;	-----最大加数为6
5+ <u>1</u> ;	-----最大加数为5
4+ <u>2</u> ; 4+ <u>1+1</u> ;	-----最大加数为4
3+ <u>3</u> ; 3+ <u>2+1</u> ; 3+ <u>1+1+1</u> ;	-----最大加数为3 $m=3$
2+ <u>2+2</u> ; 2+ <u>2+1+1</u> ; 2+ <u>1+1+1+1</u> ;	-----最大加数为2 $m=2$
1+ <u>1+1+1+1+1+1</u>	-----最大加数为1



整数 $K-m$ ，最大加数不大于 $m$ 的划分数是 $p(K-m, m)$

因此，对于整数 $K$ ，最大加数为 $m$ 的划分数就等价于：  
 $p(K-m, m)$

$$p(K, m) = \begin{cases} 1, & \text{if } (m = 1) \\ p(K, K), & \text{if } (m > K) \\ 1 + p(K, K - 1), & \text{if } (m = K) \\ p(K - m, m) + p(K, m - 1), & \text{if } (1 < m < K) \end{cases}$$

Algorithm	partitionNum(K, m)
1	If $K == 1 \parallel m == 1$
2	Return 1;
3	If $K < m$
4	Return partitionNum(K,K);
5	If $K == m$
6	Return $1 + \text{partitionNum}(K, K-1)$ ;
7	If $K > m \ \&\& \ m > 1$
8	Return $\text{partitionNum}(K, m-1) + \text{partitionNum}(K-m, m)$ ;

子问题重复，直接上述递归，性能不高。怎么办？



# 动态规划

		<i>m</i>					
		1	2	3	4	5	6
<i>K</i>	1	1	1	1	1	1	1
	2	1					
	3	1					
	4	1			$p(K,m)$		
	5	1					
	6	1	↓	↓			$p(K,K)$

$$p(K,m) = \begin{cases} 1, & \text{if } (m = 1) \\ p(K,K), & \text{if } (m > K) \\ 1 + p(K, K - 1), & \text{if } (m = K) \\ p(K - m, m) + p(K, m - 1), & \text{if } (1 < m < K) \end{cases}$$

Algorithm	partitionNum(K,m)
1	For i = 1 to K
2	$p[i,1] = 1; \quad p[1,i] = 1;$
	End for
3	For j = 2 to m
4	For i = 2 to K
	If $i < j$
	$p[i,j] = p[1,i];$
5	If $i == j$
6	$p[i,j] = 1 + p[i,i-1];$
7	If $i > j$
	$p[i,j] = p[i, j-1] + p[i-j, j];$
	End for
	End for
8	Return $p[K,m]$

# 总结

- 动态规划的本质和思想
- 具体问题具体分析
  - 形式化
  - 写出递归形式（如何从大问题转化为小问题）
  - 写出算法（边界和递推）
- 具体问题：
  - 矩阵链相乘
  - 平面凸多边形最优三角划分
  - 0-1背包问题
  - 最长公共子序列问题
  - 高性能计算机任务分配问题
  - 整数划分问题

## 作业一：

### – 最长公共子序列问题

- 输入：第一行：随机产生序列**A**（**m**个字符）；随机产生序列**B**（**m**个字符）
- 输出：序列**A**和序列**B**的最长公共子序列（相同的机器、相同的编程语言）

问题1: 规模**m**为**5**时，穷举法和用动态规划的时间分别是多少？

问题2: 规模**m**为**25**时，穷举法和用动态规划的时间分别是多少？



# 作业二：

## 编程任务：

现在需要你编写程序，给这  $p$  个节点安排计算任务，使得这个工程计算任务能够尽早完成。假定任务安排好后不再变动，而且所有的节点都同时开始运行，任务安排的目标是使最后结束计算的节点的完成时间尽可能早。

## 输入输出：

输入文件名是 `hpc.in`。

文件的第一行是对计算任务的描述，包括两个正整数  $n_A$  和  $n_B$ ，分别是 A 类和 B 类子任务的数目，两个整数之间由一个空格隔开。

文件的后面部分是对此计算机的描述：

文件第二行是一个整数  $p$ ，即计算节点的数目。

随后连续的  $p$  行按顺序分别描述各个节点的信息，第  $i$  个节点由第  $i+2$  行描述，该行包括下述四个正整数（相邻两个整数之间有一个空格）： $t_i^A$   $t_i^B$   $k_i^A$   $k_i^B$

## 输出文件：

输出文件名是 `hpc.out`。其中只有一行，包含有一个正整数，即从各节点开始计算到任务完成所用的时间。

## 样例：

设输入文件 `hpc.in` 为

5 5

3

15 10 6 4

70 100 7 2

30 70 1 6

对应的输出文件 `hpc.out` 为

93

数据说明：

$$1 \leq n_A \leq 60, 1 \leq n_B \leq 60$$

$$1 \leq p \leq 20$$

$$1 \leq t_A \leq 1000, 1 \leq t_B \leq 1000, 1 \leq k_A \leq 50, 1 \leq k_B \leq 50$$