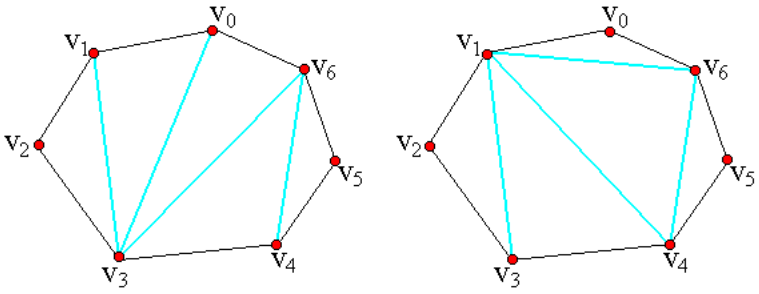
凸多边形最优三角剖分

凸多边形最优三角剖分

- 用多边形顶点的逆时针序列表示凸多边形，即$P = \{v_0, v_1, \ldots, v_{n-1}\}$表示具有$n$条边的凸多边形。
- 若v_i与v_j是多边形上不相邻的2个顶点，则线段v_iv_j称为多边形的一条弦。弦将多边形分割成2个多边形$\{vi, v_{i+1}, \ldots, v_j\}$和$\{vj, v_{j+1}, \ldots vi\}$。
- **多边形的三角剖分**是将多边形分割成互不相交的三角形的弦的集合T。

给定凸多边形P，以及定义在由多边形的边和弦组成的三角形上的权函数w。要求确定该凸多边形的三角剖分，使得即该三角剖分中诸三角形上权之和为最小。

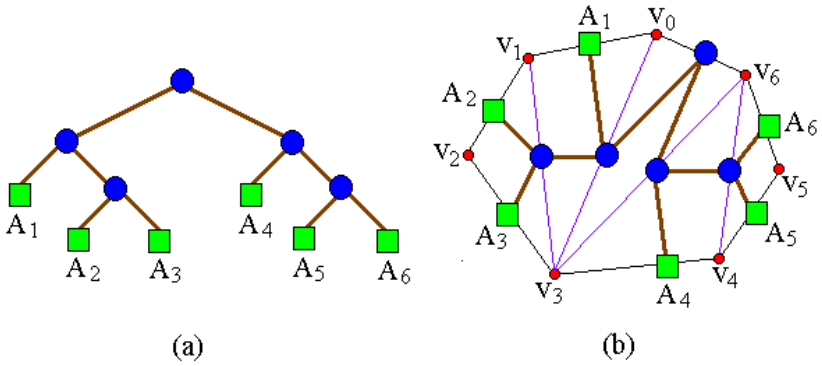


三角剖分的结构及其相关问题

•一个表达式的完全加括号方式相应于一棵完全二叉树，称为表达式的语法树。例如，完全加括号的矩阵连乘积$((A_1(A_2A_3))(A_4(A_5A_6)))$所相应的语法树如图 (a)所示。

•凸多边形$\{v_0, v_1, \ldots v_{n-1}\}$的三角剖分也可以用语法树表示。例如，图 (b)中凸多边形的三角剖分可用图 (a)所示的语法树表示。

•矩阵连乘积中的每个矩阵A_i对应于凸(n+1)边形中的一条边$v_{i-1}v_i$。三角剖分中的一条弦v_iv_j，i<j，对应于矩阵连乘积A[i+1:j]。



最优子结构性质

- 凸多边形的最优三角剖分问题有最优子结构性质。
- 事实上，若凸(n+1)边形$P=\{v_0,v_1,\ldots,v_{n-1}\}$的最优三角剖分T包含三角形$v_0v_kv_n$，$1\le k\le n-1$，则T的权为3个部分权的和：三角形$v_0v_kv_n$的权，子多边形$\{v_0,v_1,\ldots,v_k\}$和$\{v_k,v_{k+1},\ldots,v_n\}$的权之和。可以断言，由T所确定的这2个子多边形的三角剖分也是最优的。因为若有$\{v_0,v_1,\ldots,v_k\}$或$\{v_k,v_{k+1},\ldots,v_n\}$的更小权的三角剖分将导致T不是最优三角剖分的矛盾。

最优三角剖分的递归结构

- 定义t[i][j]，1≤i<j≤n为凸子多边形{vi-1,vi,…,vj}的最优三角剖分所对应的权函数值，即其最优值。为方便起见，设退化的多边形{vi-1,vi}具有权值0。据此定义，要计算的凸(n+1)边形P的最优权值为t[1][n]。
- t[i][j]的值可以利用最优子结构性质递归地计算。
  - 当j-i≥1时，凸子多边形至少有3个顶点。由最优子结构性质，t[i][j]的值应为t[i][k]的值加上t[k+1][j]的值，再加上三角形$v_{i-1}v_kv_j$的权值，其中i≤k≤j-1。
  - 由于在计算时还不知道k的确切位置，而k的所有可能位置只有j-i个，因此可以在这j-i个位置中选出使t[i][j]值达到最小的位置。由此，t[i][j]可递归地定义为：

$$t[i][j]=\begin{cases}0 & i=j\\ \min\limits_{i\le k<j}\{t[i][k]+t[k+1][j]+w(v_{i-1}v_kv_j)\} & i<j\end{cases}$$

多边形游戏

多边形游戏

多边形游戏是一个单人玩的游戏，开始时有一个由n个顶点构成的多边形。每个顶点被赋予一个整数值，每条边被赋予一个运算符“+”或“*”。所有边依次用整数从1到n编号。

游戏第1步，将一条边删除。

随后n-1步按以下方式操作:

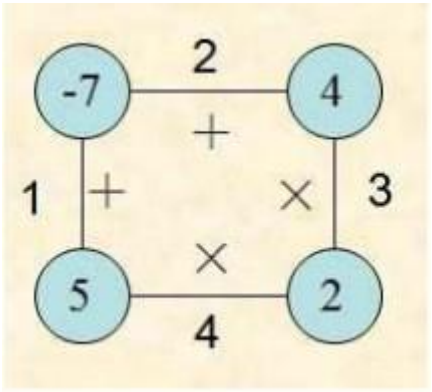
(1)选择一条边E以及由E连接着的2个顶点V1和V2;

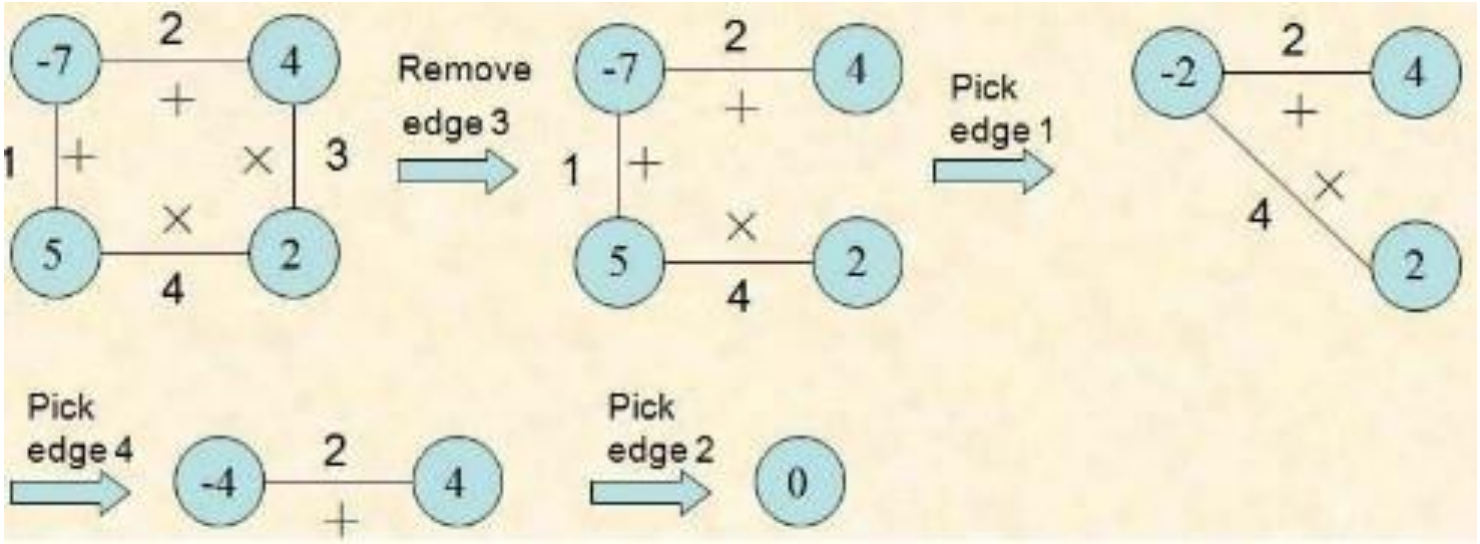
(2)用一个新的顶点取代边E以及由E连接着的2个顶点V1和V2。将由顶点V1和V2的整数值通过边E上的运算得到的结果赋予新顶点。

(3)所有边都被删除，游戏结束。游戏的得分就是所剩顶点上的整数值。

问题: 对于给定的多边形，计算最高得分。

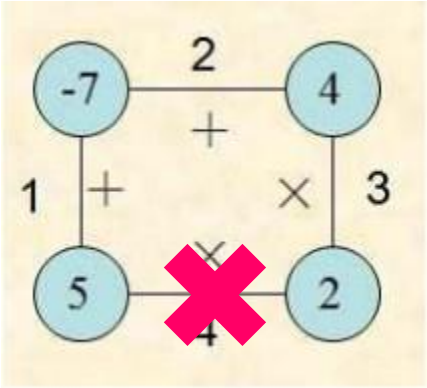
多边形游戏


-7
2
4
1
3
5
2
4


-7
2
4
1
3
5
2
4
Remove
edge 3
-7
2
4
1
5
2
4
Pick
edge 1
-2
2
4
4
2
Pick
edge 4
-4
2
4
Pick
edge 2
0

最优子结构性质

- 在所给多边形中，从顶点i(1≤i≤n)开始，长度为j(链中有j个顶点)的顺时针链p(i，j) 可表示为v[i]，op[i+1]，…，v[i+j-1]。
- 如果这条链的最后一次合并运算在op[i+s]处发生(1≤s≤j-1)，则可在op[i+s]处将链分割为2个子链p(i，s)和p(i+s，j-s)。



最优子结构性质

- 设m1是对子链p(i，s)的任意一种合并方式得到的值，而a和b分别是在所有可能的合并中得到的最小值和最大值。
- m2是p(i+s，j-s)的任意一种合并方式得到的值，而c和d分别是在所有可能的合并中得到的最小值和最大值。
- 依此定义有a≤m1≤b，c≤m2≤d

(1)当op[i+s]='+'时，显然有a+c≤m≤b+d

(2)当op[i+s]='*'时，有min{ac，ad，bc，bd}≤m≤max{ac，ad，bc，bd}

- 换句话说，主链的最大值和最小值可由子链的最大值和最小值得到。

图像压缩

图像压缩

图象的变位压缩存储格式将所给的象素点序列$\{p_1, p_2, \ldots, p_n\}$，$0 \le p_i \le 255$分割成m个连续段$S_1, S_2, \ldots, S_m$。第i个象素段$S_i$中$(1 \le i \le m)$，有$l[i]$个象素,且该段中每个象素都只用b[i]位表示。设 $t[i] = \sum_{k=1}^{i-1} l[k]$，则第i个象素段$S_i$为

$$S_i = \{p_{t[i]+1}, \cdots, p_{t[i]+l[i]}\}$$

设 $h_i = \left\lceil \log\left(\max_{t[i]+1 \le k \le t[i]+l[i]} p_k + 1 \right) \right\rceil$，则$h_i \le b[i] \le 8$。因此需要用3位表示b[i],如果限制$1 \le l[i] \le 255$，则需要用8位表示$l[i]$。因此，第i个象素段所需的存储空间为$l[i]$*b[i]+11位。按此格式存储象素序列$\{p_1, p_2, \ldots, p_n\}$，需要 $\sum_{i=1}^{m} l[i]*b[i]+11m$ 位的存储空间。

图象压缩问题要求确定象素序列$\{p_1, p_2, \ldots, p_n\}$的最优分段，使得依此分段所需的存储空间最少。每个分段的长度不超过256位。

图像压缩

设$l[i], b[i], 1 \le i \le m$, 是$\{p_1, p_2, \ldots, p_n\}$的最优分段。显而易见，l[1]，b[1]是$\{p_1, \ldots, p_{l[1]}\}$的最优分段，且$l[i], b[i], 2 \le i \le m$，是$\{p_{l[1]+1}, \ldots, p_n\}$的最优分段。即图象压缩问题满足最优子结构性质。

设s[i]，1≤i≤n，是象素序列$\{p_1, \ldots, p_i\}$的最优分段所需的存储位数。由最优子结构性质易知：

$$s[i] = \min_{1 \le k \le \min\{i, 256\}} \{s[i-k] + k * \mathrm{bmax}(i-k+1, i)\} + 11$$

其中 $\mathrm{bmax}(i, j) = \left\lceil \log\left(\max_{i \le k \le j} \{p_k\} + 1 \right) \right\rceil$

算法复杂度分析：
由于算法中对k的循环次数不超这256，故对每一个确定的i，可在时间O(1)内完成的计算。因此整个算法所需的计算时间为O(n)。

流水作业调度

流水作业调度

n个作业{1，2，…，n}要在由2台机器M1和M2组成的流水线上完成加工。每个作业加工的顺序都是先在M1上加工，然后在M2上加工。M1和M2加工作业i所需的时间分别为a_i和b_i。
流水作业调度问题要求确定这n个作业的最优加工顺序，使得从第一个作业在机器M1上开始加工，到最后一个作业在机器M2上加工完成所需的时间最少。

分析：

- 直观上，一个最优调度应使机器M1没有空闲时间，且机器M2的空闲时间最少。在一般情况下，机器M2上会有机器空闲和作业积压2种情况。
- 设全部作业的集合为N={1，2，…，n}。$S \subseteq N$是N的作业子集。在一般情况下，机器M1开始加工S中作业时，机器M2还在加工其它作业，要等时间t后才可利用。将这种情况下完成S中作业所需的最短时间记为T(S, t)。流水作业调度问题的最优值为T(N,0)。

流水作业调度

设π是所给n个流水作业的一个最优调度，它所需的加工时间为 $a_{\pi(1)}+T'$。其中T'是在机器M2的等待时间为$b_{\pi(1)}$时，安排作业π(2)，…，π(n)所需的时间。

记S=N-{π(1)}，**则有$T'=T(S, b_{\pi(1)})$**。

证明：事实上，由T的定义知$T'\geq T(S, b_{\pi(1)})$。若$T'>T(S, b_{\pi(1)})$，设π'是作业集S在机器M2的等待时间为$b_{\pi(1)}$情况下的一个最优调度。则π(1)，π'(2)，…，π'(n)是N的一个调度，且该调度所需的时间为$a_{\pi(1)}+T(S,b_{\pi(1)})<a_{\pi(1)}+T'$。这与π是N的最优调度矛盾。故$T'\leq T(S, b_{\pi(1)})$。从而$T'=T(S, b_{\pi(1)})$。这就证明了流水作业调度问题具有最优子结构的性质。

由流水作业调度问题的最优子结构性质可知，

$$T(N,0)=\min_{1\leq i\leq n}\{a_i+T(N-\{i\},b_i)\}$$

$$T(S,t)=\min_{i\in S}\{a_i+T(S-\{i\},b_i+\max\{t-a_i,0\})\}$$

Johnson不等式

对递归式的深入分析表明，算法可进一步得到简化。
设π是作业集S在机器M2的等待时间为t时的任一最优调度。
若π(1)=i，π(2)=j。则由动态规划递归式可得：

$$T(S,t) = a_i + T(S-\{i\}, b_i + \max\{t-a_i, 0\})$$
$$= a_i + a_j + T(S-\{i,j\}, t_{ij})$$

其中，

$$t_{ij} = b_j + \max\{b_i + \max\{t-a_i,0\} - a_j, 0\}$$
$$= b_j + b_i - a_j + \max\{\max\{t-a_i,0\}, a_j - b_i\}$$
$$= b_j + b_i - a_j + \max\{t-a_i, a_j - b_i, 0\}$$
$$= b_j + b_i - a_j - a_i + \max\{t, a_i + a_j - b_i, a_i\}$$

如果作业i和j满足$\min\{b_i,a_j\} \geq \min\{b_j,a_i\}$，则称作业i和j满足Johnson不等式。

流水作业调度的Johnson法则

$$t_{ij} = b_j + b_i - a_j - a_i + \max\{t, a_i + a_j - b_i, a_i\}$$

交换作业i和作业j的加工顺序，得到作业集S的另一调度，它所需的加工时间为$T'(S,t) = a_i + a_j + T(S - \{i,j\}, t_{ji})$

其中，$t_{ji} = b_j + b_i - a_j - a_i + \max\{t, a_i + a_j - b_j, a_j\}$

当作业i和j满足Johnson不等式时，有

$$\max\{-b_i, -a_j\} \le \max\{-b_j, -a_i\}$$

$$a_i + a_j + \max\{-b_i, -a_j\} \le a_i + a_j + \max\{-b_j, -a_i\}$$

$$\max\{a_i + a_j - b_i, a_i\} \le \max\{a_i + a_j - b_j, a_j\}$$

$$\max\{t, a_i + a_j - b_i, a_i\} \le \max\{t, a_i + a_j - b_j, a_j\}$$

由此可见当作业i和作业j不满足Johnson不等式时，交换它们的加工顺序后，不增加加工时间。对于流水作业调度问题，必存在最优调度π，使得作业π(i)和π(i+1)满足Johnson不等式。进一步还可以证明，调度满足Johnson法则当且仅当对任意i<j有

$$\min\{b_{\pi(i)}, a_{\pi(j)}\} \ge \min\{b_{\pi(j)}, a_{\pi(i)}\}$$

由此可知，**所有满足Johnson法则的调度均为最优调度。**

算法描述

流水作业调度问题的Johnson算法

(1)令 $N_1=\{i \mid a_i < b_i\}, N_2=\{i \mid a_i \geq b_i\}$;

(2)将N_1中作业依a_i的非减序排序；将N_2中作业依b_i的非增序排序；

(3)N_1中作业接N_2中作业构成满足Johnson法则的最优调度。

算法复杂度分析：

算法的主要计算时间花在对作业集的排序。因此，在最坏情况下算法所需的计算时间为**O(nlogn)**。所需的空间为**O(n)**。

0-1背包问题

0-1 背包问题

给定n种物品和一背包。物品i的重量是w_i，其价值为v_i，背包的容量为C。问应如何选择装入背包的物品，使得装入背包中物品的总价值最大?

n=3，c=6，w={4，3，2}，v={5，2，1}

0-1背包问题是一个特殊的整数规划问题。

$$\max \sum_{i=1}^{n} V_i X_i$$

$$\begin{cases} \sum_{i=1}^{n} w_i x_i \le C \\ x_i \in \{0,1\}, 1 \le i \le n \end{cases}$$

找寻最优子结构

- 设$(y_1, y_2, \ldots\ldots y_n)$是$x_1 \sim x_n$的一个最优解，则可以推断，$(y_2, y_3, \ldots\ldots y_n)$是一个子问题的最优解。

$$\max \sum_{i=2}^{n} v_i y_i$$

$$\begin{cases} \sum_{i=1}^{n} w_i y_i \leq C - w_1 y_1 \\ y_i \in \{0,1\}, 1 \leq i \leq n \end{cases}$$

0-1背包问题

设所给0-1背包问题的子问题

$$\max \sum_{k=i}^{n} v_k x_k$$

$$\begin{cases} \sum_{k=i}^{n} w_k x_k \le j \\ x_k \in \{0,1\}, i \le k \le n \end{cases}$$

的最优值为m(i，j)，即m(i，j)是背包容量为j，可选择物品为i，i+1，…，n时0-1背包问题的最优值。

0-1背包问题

设所给0-1背包问题的子问题

$$\max \sum_{k=i}^{n} v_k x_k \qquad \begin{cases} \sum_{k=i}^{n} w_k x_k \le j \\ x_k \in \{0,1\}, i \le k \le n \end{cases}$$

的最优值为m(i，j)，即m(i，j)是背包容量为j，可选择物品为i，i+1，…，n时0-1背包问题的最优值。由0-1背包问题的最优子结构性质，可以建立计算m(i，j)的递归式如下。

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

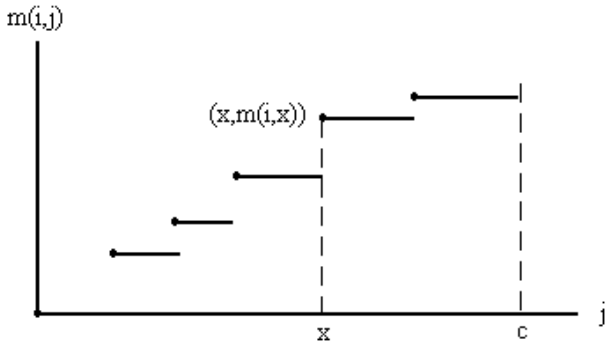
$$m(n,j) = \begin{cases} v_n & j \ge w_n \\ 0 & 0 \le j < w_n \end{cases}$$

0-1背包问题

算法复杂度分析：
从m(i，j)的递归式容易看出，算法需要O(nc)计算时间。当背包容量c很大时，算法需要的计算时间较多。例如，当$c>2^n$时，算法需要$\Omega(n2^n)$计算时间。

算法改进

由m(i, j)的递归式容易证明，在一般情况下，对每一个确定的i(1≤i≤n)，函数m(i, j)是关于变量j的阶梯状单调不减函数。跳跃点是这一类函数的描述特征。在一般情况下，函数m(i, j)由其全部跳跃点唯一确定。如图所示。

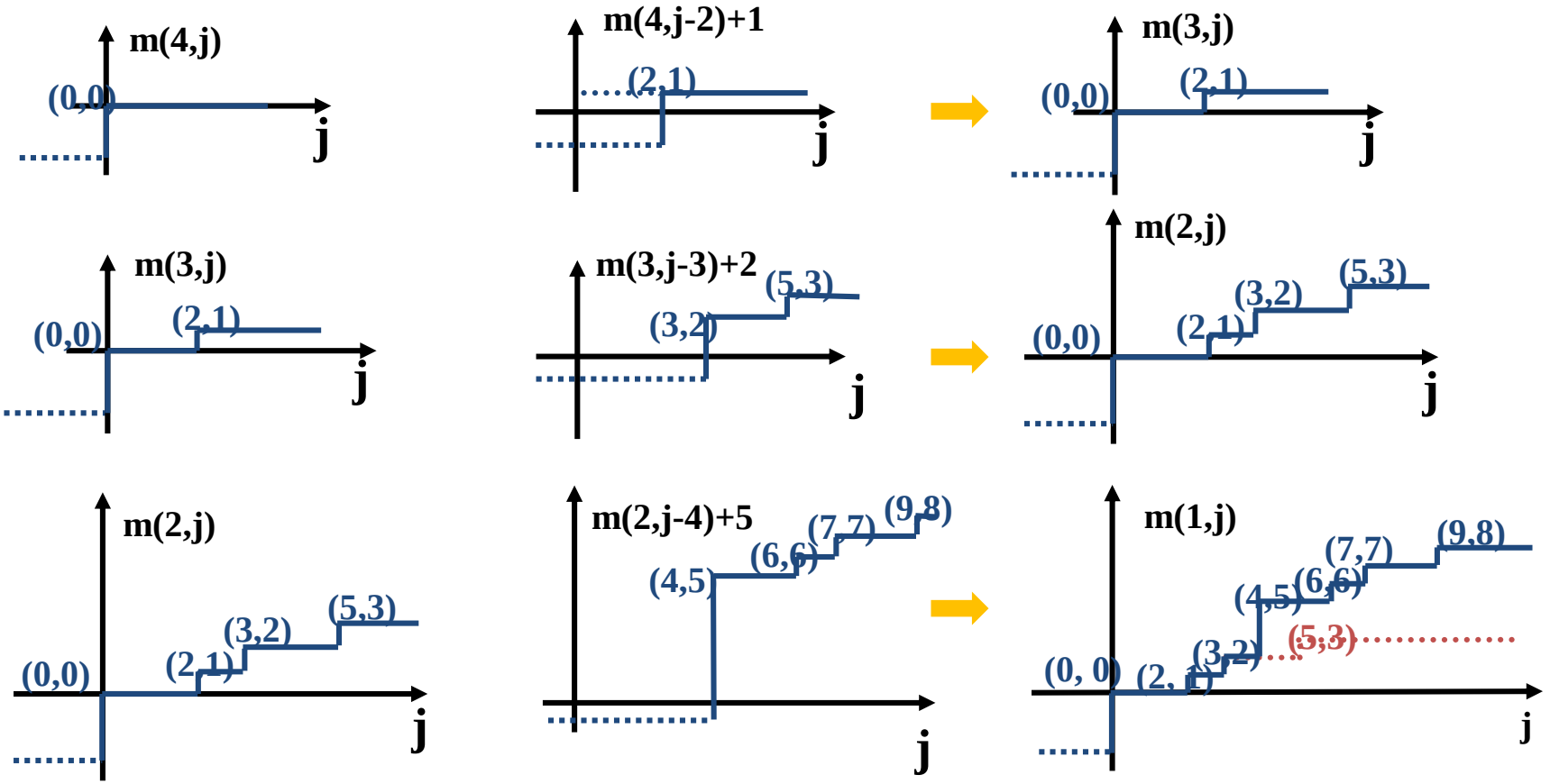


对每一个确定的i(1≤i≤n)，用一个链表p[i]存储函数m(i, j)的全部跳跃点。链表p[i]可依计算m(i, j)的递归式递归地由表p[i+1]计算，初始时p[n+1]={(0, 0)}。

一个例子

$$m(i,j)=\begin{cases}\max\{m(i+1,j),m(i+1,j-w_i)+v_i\} & j\geq w_i\\ m(i+1,j) & 0\leq j<w_i\end{cases}$$

n=3，c=6，w={4，3，2}，v={5，2，1}。



算法改进

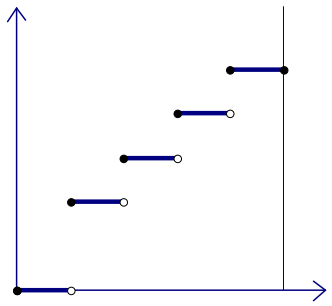
$$m(i,j)=\begin{cases}\max\{m(i+1,j),m(i+1,j-w_i)+v_i\} & j\geq w_i\\ m(i+1,j) & 0\leq j<w_i\end{cases}$$

- 函数m(i,j)是由函数m(i+1,j)与函数m(i+1,j-w_i)+v_i作max运算得到的。因此，函数m(i,j)的全部跳跃点包含于函数m(i+1, j)的跳跃点集p[i+1]与函数m(i+1, j-w_i)+ v_i的跳跃点集q[i+1]的并集中。
  - 易知，(s,t)∈q[i+1]当且仅当w_i ≤s≤c且(s- w_i,t- v_i)∈p[i+1]。
  - 因此，容易由p[i+1]确定跳跃点集q[i+1]如下q[i+1]=p[i+1]⊕(w_i, v_i)={(j+ w_i,m(i,j)+ v_i)|(j,m(i,j))∈p[i+1]}
- 另一方面，设(a，b)和(c，d)是p[i+1]∪q[i+1]中的2个跳跃点，则当c≥a且d<b时，(c，d)受控于(a，b)，从而(c，d)不是p[i]中的跳跃点。除受控跳跃点外，p[i+1]∪q[i+1]中的其它跳跃点均为p[i]中的跳跃点。
- 由此可见，在递归地由表p[i+1]计算表p[i]时，可先由p[i+1]计算出q[i+1]，然后合并表p[i+1]和表q[i+1]，并清除其中的受控跳跃点得到表p[i]。

一个例子

n=5，c=10，w={2，2，6，5，4}，v={6，3，5，4，6}。

初始时p[6]={(0,0)}，(w5,v5)=(4,6)。因此，
q[6]=p[6]⊕(w5,v5)={(4,6)}。
p[5]={(0,0),(4,6)}。
q[5]=p[5]⊕(w4,v4)={(5,4),(9,10)}。从跳跃点集p[5]与q[5]的并集p[5]∪q[5]={(0,0),(4,6),(5,4),(9,10)}中看到跳跃点(5,4)受控于跳跃点(4,6)。将受控跳跃点(5,4)清除后，得到
p[4]={(0,0),(4,6),(9,10)}
q[4]=p[4]⊕(6，5)={(6，5)，(10，11)}
p[3]={(0，0)，(4，6)，(9，10)，(10，11)}
q[3]=p[3]⊕(2，3)={(2，3)，(6，9)}
p[2]={(0，0)，(2，3)，(4，6)，(6，9)，(9，10)，(10，11)}
q[2]=p[2]⊕(2，6)={(2，6)，(4，9)，(6，12)，(8，15)}
p[1]={(0，0)，(2，6)，(4，9)，(6，12)，(8，15)}
p[1]的最后的那个跳跃点(8,15)给出所求的最优值为m(1,c)=15。



算法复杂度分析

上述算法的主要计算量在于计算跳跃点集p[i](1≤i≤n)。由于q[i+1]=p[i+1]⊕(w_i，v_i)，故计算q[i+1]需要O(|p[i+1]|)计算时间。合并p[i+1]和q[i+1]并清除受控跳跃点也需要O(|p[i+1]|)计算时间。从跳跃点集p[i]的定义可以看出，p[i]中的跳跃点相应于$x_i,\ldots,x_n$的0/1赋值。因此，p[i]中跳跃点个数不超过2^{n-i+1}。由此可见，算法计算跳跃点集p[i]所花费的计算时间为

$$O\left(\sum_{i=2}^{n}|p[i+1]|\right)=O\left(\sum_{i=2}^{n}2^{n-i}\right)=O(2^n)$$

从而，改进后算法的计算时间复杂性为$O(2^n)$。当所给物品的重量w_i(1≤i≤n)是整数时，|p[i]|≤c+1，(1≤i≤n)。在这种情况下，改进后算法的计算时间复杂性为$O(\min\{nc, 2^n\})$。