算法设计与分析

第3章 动态规划 (3)

谢晓芹
哈尔滨工程大学计算机科学与技术学院

0-1背包问题

- 真人游戏：在有限时间从奖池中取奖品

- 0-1背包问题是一类经典的组合优化问题
- 对0-1背包问题的研究可以广泛运用于资源分配、投资决策、货物装载等方面。
  - 处理机和数据库在分布式计算机系统上的分配问题
  - 项目选择的货物装载问题
  - 削减库存问题等

0-1背包问题

- 研究
  - 计算机学报
    - 基于离散差分演化的KPC问题降维建模与求解, 2019, 10:2267-2280
    - 基于遗传算法求解折扣{0-1}背包问题的研究，2016，12:2614-2630
    - 多重二次背包问题的量子进化求解算法,2016,8:1518-1529
    - 二次背包问题的一种快速解法,国防科大计算机学院 2004,9：1162 ~ 1169
    - 多维背包问题的一个蚁群优化算法，哈工大计算机学院 ,2008，5：810 ~ 819
    - NodeRank：一种高效软硬件划分算法. 陈志 武继刚 宋国治 陈金亮. 2013, 10, P2033-2040.
  - 软件学报
    - 求解背包问题的演化算法, 王熙照,贺毅朝. 2017, 28(1):1-16
    - 面向MIMO多跳无线网络的多用户视频传输优化方法, 北京大学 周超,张行功,郭宗明. 2013,24(2):279-294

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?
  - 每种物品要么装入，要么不装入
  - 不能装入多次，也不能装入部分
- 分析
  - 装，用1表示，不装，用0表示。
  - 问题实质是求$(x_1, x_2,..., x_n)$, $x_i \in \{0,1\}, 1 \le i \le n$
  - 例如：
    - 三个物品，重量和价值分别为$(w_0 = 3, v_0 = 4)$,$(w_1 = 4, v_1 = 5)$,$(w_2 = 5, v_2 = 6)$, 其中背包容量为C=10
    - 应该选物品(w_1, w_2), 此时$w_1 + w_2 = 9 < 10$, $v_1 + v_2 = 11$总价值最大

0-1背包问题

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

$$\max \sum_{i=1}^{n} v_i x_i, \quad \sum_{i=1}^{n} w_i x_i \le C$$

- 穷举法
  - 时间复杂度=O(2^n)

0-1背包问题

- 简化方法
  - 先找最优总价值
  - 再扩展算法找最优物品选择方案
- 策略
  - 考虑对前一个物品做出选择后，剩下的物品选择方案
  - 关键：子问题？

涉及哪些参数？

0-1背包问题

- 子问题分析
  - 设$(y_1,y_2,y_3,\ldots,y_n)$ 是所给问题的一个最优解,则$(y_2,y_3,\ldots,y_n)$ 是下面相应**子问题**的一个最优解:
  - 输入：可选物品为{2,...,n};
  - 输出： $y_2,y_3,\ldots,y_n$,满足

$$max\sum_{i=2}^{n}v_iy_i \qquad \sum_{i=2}^{n}w_iy_i \le C-w_1y_1$$

$$y_i \in \{0,1\}, 2 \le i \le n$$

注意：当物品放入或不放入后，包的容量发生变化

  - 反证法证明

0-1背包问题 - ①最优子结构性质

- 证明：假设$(y_2,y_3,\ldots,y_n)$不是最优解，则设 反证

  $(z_2,z_3,\ldots,z_n)$是最优解，其满足：

  $\sum_{i=2}^{n} v_i z_i > \sum_{i=2}^{n} v_i y_i$ 且 $w_1 y_1 + \sum_{i=2}^{n} w_i z_i \le C$

- 有： $v_1 y_1 + \sum_{i=2}^{n} v_i z_i > v_1 y_1 + \sum_{i=2}^{n} v_i y_i = \sum_{i=1}^{n} v_i y_i$ 构造

$$w_1 y_1 + \sum_{i=2}^{n} w_i z_i \le C$$

故$(y_1,z_2,\ldots,z_n)$是比 $(y_1,y_2,\ldots,y_n)$更优的解，产生矛盾。假设错误，命题得证。

0-1背包问题 - ①最优子结构性质

- 子问题定义
  - 输入：可选物品为{**i** ,i+1,...,n} ， $w_i>0$, $v_i>0$;
  - 输出：($\mathbf{x_i}$, x_{i+1}, …, x_n), $x_i \in \{0, 1\}$, 满足

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

- 子问题的变化因素
  - 可选物品数
  - 容量限制
- 子问题个数： O(nC)？

0-1背包问题 - ②递归关系

- 求解策略：
  - 先**求最优总价值**，再求物品选择方案
- 定义：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。

0-1背包问题 - ②递归关系

- 由最优子结构性质，建立计算最优总价值m(i，j)的递归式:

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

通过m(i+1, x)来计算m(i,x)

不包含物品i　　包含物品i

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

可选物品i

不可选物品i

注意：i之前的物品都已经做完选择

0-1背包问题 - ③计算最优值

- 计算最优值：动态规划填表方式计算

1，2，…，n

```
Knapsack(int w, int c, int n, type **m){
1.      int jMax=min(w[n]-1, c);    //第n个物品的重量和C选小值
2.      for(int j=0; j<=jMax; j++){m[n][j]=0;}
3.      for(int j=w[n];j<=c;j++) {m[n][j]=v[n];}
4.      for(int i=n-1;i>1;i--){  //物品从第n-1个开始选直到物品2
5.          jMax=min(w[i]-1,c);
6.          for (int j=0;j<=jMax;j++){  m[i][j]=m[i+1][j]; } //不选物品i
7.          for (int j=w[i];j<=c;j++){  //可以选物品i
8.                m[i][j]=max(m[i+1][j], m[i+1][j-w[i]]+v[i]); }  }
9.      m[1][c]=m[2][c];  //物品从第1个开始选的情况
10.     If (c>=w[1]){m[1][c]=
                    max(m[1][c], m[2][c-w[1]]+v[1]);}
}
```

边界条件

枚举变化的j

递归表达式中 不能选i物品的情况

i=1第一个物品选择情况，前面循环没有处理

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

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

0-1背包问题 - ④构造最优解

- 递归构造最优解
  - m[1][c]=m[2][c]，则x1=0,否则x1=1
  - 若x1=0　（没有选第一个物品）
    - 若m[2][c]=m[3][c],则x2=0,否则x2=1
  - 若x1=1　（选了第一个物品）
    - 若m[2][**c-w1**]=m[3][**c-w1**],则x2=0,否则x2=1
- 算法实现

0-1背包问题 - ④构造最优解

```
void Traceback(Type **m, int w, int c, int n, int x){
    for (int i=1; i<n; i++)
        if (m[i][c]==m[i+1][c]) x[i]=0;
        else { x[i]=1;
               c- =w[i];}
    x[n]=(m[n][c])?1:0;
}
```

注意：c是变化的

m[n][c]=1有值表示选了n物品， x[n]=1
m[n][c]=0无值表示没有选n物品, x[n]=0

0-1背包问题

- 时间复杂度分析：
  - 动态规划的本质是把所有前面已知的结果建成一个大表格,表格是迭代构造的.
  - 从m(i，j)的递归式容易看出，表格有nC项,每一项由其他两项在常数式时间内计算得到.算法需要O(nC)计算时间。
  - 回推构造最优解：O(n)计算时间
- 空间复杂度分析
  - O(nC)

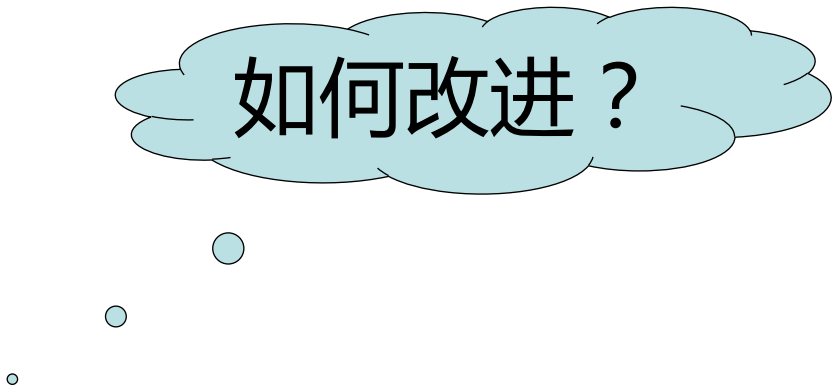
n个物品
C最大容量

0-1背包问题

	最优解
原问题： n个物品 $\{x_1,x_2,x_3,\ldots,x_n\}$ → $\{y_1,y_2,y_3,\ldots,y_n\}$ 满足 $\max \sum_{i=1}^{n} v_i x_i,\ \sum_{i=1}^{n} w_i x_i \le C$	$\{y_1,y_2,y_3,\ldots,y_n\}$
子问题： n-1个物品 $\{x_2,x_3,\ldots,x_n\}$ → $\{y_2,y_3,\ldots,y_n\}$ 满足 $\max \sum_{i=2}^{n} v_i x_i,\ \sum_{i=2}^{n} w_i x_i \le C - w_1 x_1$	$\{y_2,y_3,\ldots,y_n\}$

0-1背包问题

- 存在问题
  - 要求物品重量w_i是**整数**
  - 当背包容量C很大时，算法所需计算时间较多.如,当$c>2^n$时，要$\Omega(n2^n)$ 计算时间。



0-1背包问题

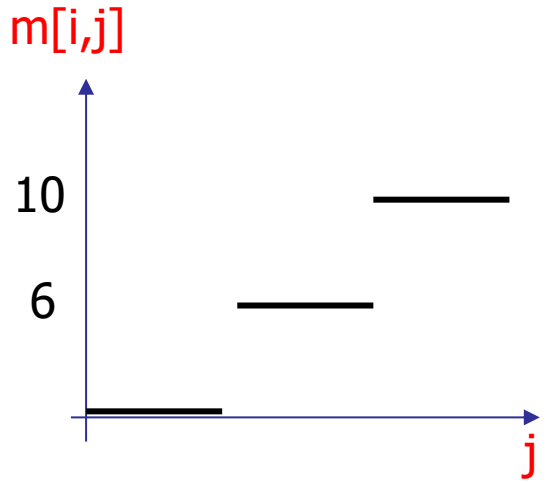
- 改进策略： 将问题转化为跳跃点问题
- 例
  - n =5, C=10, w={2,2,6,5,4}, v={6,3,5,4,6}
  - 当i=5时

$$m(5,j)=\begin{cases}6 & j\geq 4\\ 0 & 0\leq j<4\end{cases}$$

边界条件

  - 当i=4时?

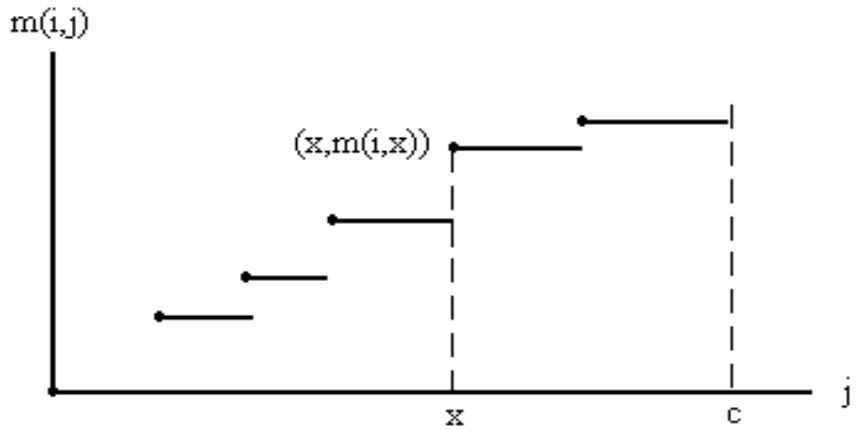
$$m(4,j)=\begin{cases}10 & j\geq 9\\ 6 & 4\leq j\leq 9\\ 0 & 0\leq j<4\end{cases}$$



0-1背包问题 - 算法改进

- 分析：
  - 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。
    - 表示：跳跃点坐标(s,t)=(j, m(i,j))
  - 在一般情况下，**函数m(i,j)由其全部跳跃点唯一确定**。
- 改进策略
  - j是**连续变量**时，对每一个确定的i(1≤i≤n)，用一个表p[i]存储函数m(i，j)的全部跳跃点。

j: 容量限制



0-1背包问题 - 算法改进

- 跳跃点的递归计算式的推导
  - 初始时p[n+1]={(0，0)}。
  - 表p[i]可根据计算m(i，j)的递归式递归地由表p[i+1]计算

物重为0,价值为0

0-1背包问题 - 算法改进

- 函数m(i,j) = MAX(m(i+1,j), m(i+1,j-w_i)+v_i)
  - 函数m(i,j)的全部跳跃点**p[i] = p[i+1] ∪ q[i+1]**
  - p[i+1]是函数m(i+1, j)的跳跃点集
  - q[i+1]是函数m(i+1, j-w_i)+v_i的跳跃点集
- (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]，
  - **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]}
- 控制点
  - 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。

保证能够选物品i

表示：(容量,最优值)

意义：装载量多，价值却少，肯定不是最优解

0-1背包问题 - 算法改进

- p[i]= p[i+1] ∪ q[i+1] –受控点
  - 在递归地由表p[i+1]计算表p[i]时
    - 先由p[i+1]计算出q[i+1]
    - 然后合并表p[i+1]和表q[i+1]
    - 清除其中的受控跳跃点得到表p[i]。
- $q[i+1]=p[i+1]\oplus(w_i,v_i)$

  $=\{(j+w_i,m(i,j)+v_i)|(j,m(i,j))\in p[i+1]$
- p[n+1]={(0，0)}

0-1背包问题 - 举例

- n=5，c=10，w={2，2，6，5，4}，v={6，3，5，4，6}。
- $q[i+1]=p[i+1]\oplus(w_i,v_i)=\{(j+w_i,m(i,j)+v_i)|(j,m(i,j))\in p[i+1]$
- 设(a，b)和(c，d)是$p[i+1]\cup q[i+1]$中的2个跳跃点，
  则当c≥a且d<b时，(c，d)受控于(a，b)

① 初始，$p[6]=\{(0,0)\}$，$(w_5,v_5)=(4,6)$
因此，$q[6]=p[6]\oplus(w_5,v_5)=\{(4,6)\}$

② $p[5]=p[6]\cup q[6]=\{(0,0),(4,6)\}$，
$q[5]=p[5]\oplus(w_4,v_4)=\{(5,4),(9,10)\}$　　$(w_4,v_4)=(5,4)$
$p[5]\cup q[5]=\{(0,0),(4,6),(5,4),(9,10)\}$　　(4,6)控制(5,4)

③ $p[4]=\{(0,0),(4,6),(9,10)\}$,　　$(w_3,v_3)=(6,5)$
$q[4]=p[4]\oplus(w_3,v_3)=\{(6,5),(10,11),(\sout{15,15})\}$　　(15>10, (15,15)不是跳跃点

$p[3]=\{(0,0),(4,6),\sout{(6,5)},(9,10),(10,11)\}$

0-1背包问题 - 举例

- **n=5，c=10，w={2，2，6，5，4}，v={6，3，5，4，6}。**
- **q[i+1]=p[i+1]⊕(wi,vi)={(j+wi,m(i,j)+vi)|(j,m(i,j))∈p[i+1]**
- **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)**

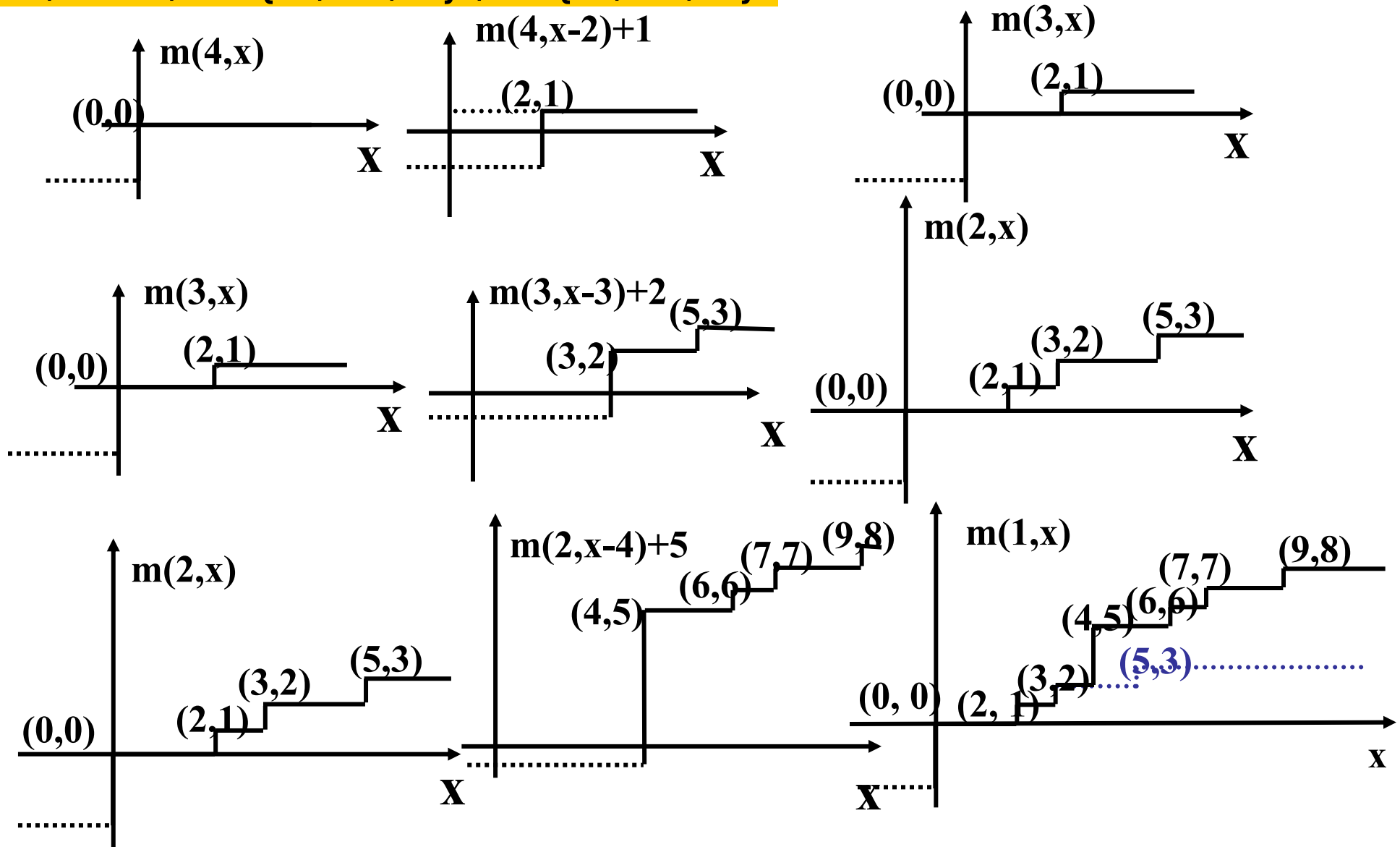
④ p[3]={(0，0)，(4，6)，(9，10)，(10，11)}
q[3]=p[3]⊕(w_2,v_2)= p[3]⊕(2,3)={(2, 3)，(6, 9)}

⑤ p[2]={(0, 0)，(2, 3)，(4, 6)，(6, 9)，(9, 10)，(10, 11)}
q[2]=p[2]⊕(w_1,v_1)= p[2]⊕(2,6)={(2, 6)，(4, 9)，(6, 12)，(8, 15)}

⑥ p[1]={(0，0)，(2，6)，(4，9)，(6，12)，**(8，15)**}

⑦ 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

n=3 , c=6 , w={4 , 3 , 2} , v={5 , 2 , 1}



0-1背包问题 - 算法复杂度分析

- 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

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

若m(1,c)=m(2,c) 则 x1=0， 否则x1=1

0-1背包问题 - 算法复杂度分析

- 当所给物品的重量是整数时，（|p[i]|）≤c+1, 1≤i≤n)

- 改进后算法的计算时间复杂性为

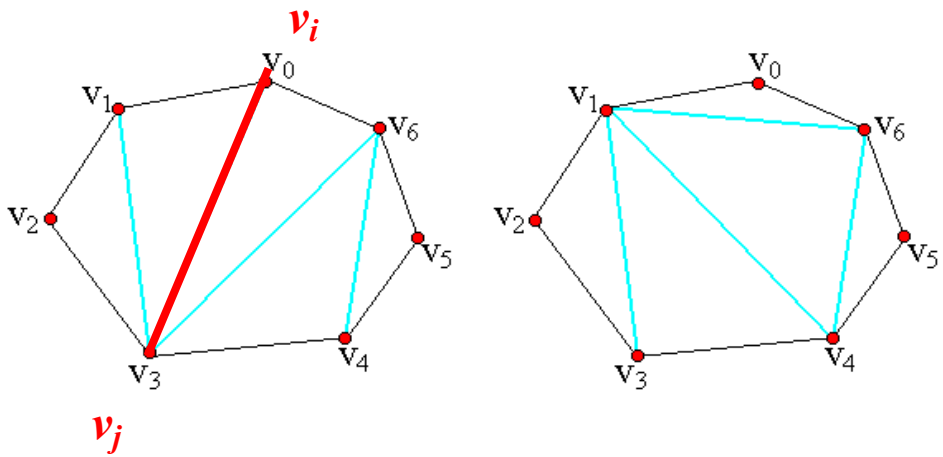
$$O(\min\{nc,2^n\})$$

凸多边形最优三角剖分

- 应用
  - Catalan数
  - 多边形三角剖分是数字城市研究许多工作的前提
    - 城市景观三维重建中的三角剖分算法
  - 基于图像特征和三角剖分的水印算法
  - 基于三角剖分的小脑模型在增强学习中的应用
  - 传感网中的动态Delauanay三角剖分算法
  - ……

凸多边形最优三角剖分

- 问题：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。
  - 用多边形顶点的逆时针序列表示凸多边形，即P={v_0,v_1,…,v_{n-1}}表示具有n条边的凸多边形
    - 凸多边形：边界上或内部任意两点所连成的直线段上所有点均在多边形的内部或边界上
  - 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v_iv_j将多边形分割成2个多边形{v_i,v_{i+1},…,v_j}和{v_j,v_{j+1},…v_i}。



凸多边形最优三角剖分

- 输入:
  - 给定凸多边形P=$\{v_0, v_1, \ldots, v_{n-1}\}$，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。

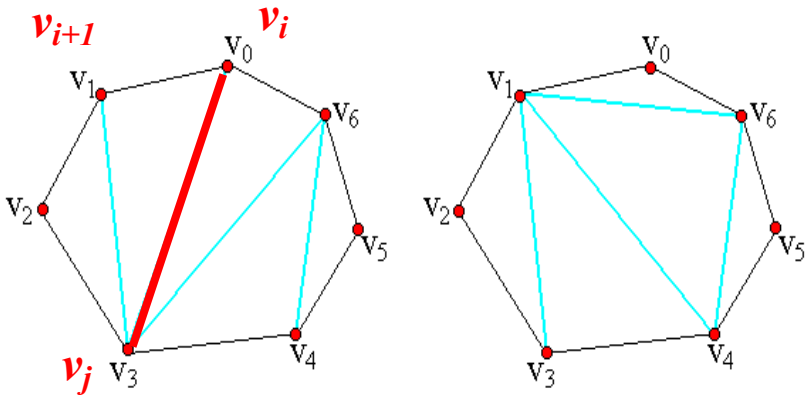
边界上或内部任意两点连成的直线段上所有点均在多边形的内部或边界上

- 输出:
  - 要求确定该凸多边形的三角剖分T，使得即该三角剖分中诸三角形上权之和为最小

可以表示为：三角形的集合或弦的集合

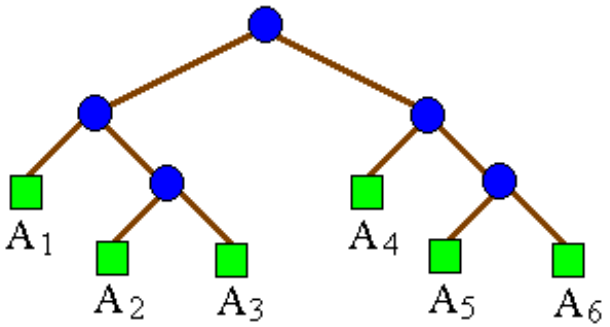
- 例如
  - 弦v_0v_3分割多边形$\{v_0, v_1, \ldots, v_6\}$为：$\{v_0, v_1, v_2, v_3\}$和$\{v_3, v_4, v_5, v_6, v_0\}$
  - 不同拆分数为c_n

n=1, $c_n = 1$→ 1个点
n=2, $c_n = 1$→ 1条线段
n=3, $c_n = 1$→ 1个三角形
n=4, $c_n = 2$→ 四边形
n=5, $c_n = 5$



三角剖分的结构及其相关问题

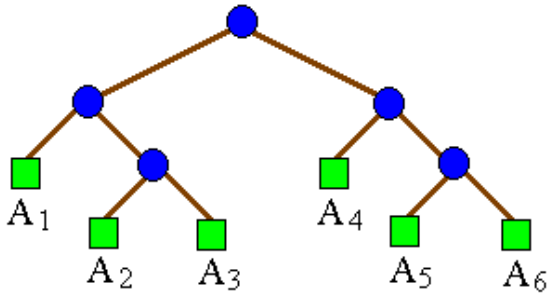
- 一个**表达式**的完全加括号方式相应于一棵完全二叉树，称为表达式的**语法树**。
  - 叶结点: 表达式中一个原子
  - 例如: (a1*(a2+a3))-(a4*(a5+a6))
- 完全加括号的矩阵连乘积$((A_1(A_2A_3))(A_4(A_5A_6)))$ 相应的语法树为
  - 叶结点: 一个矩阵



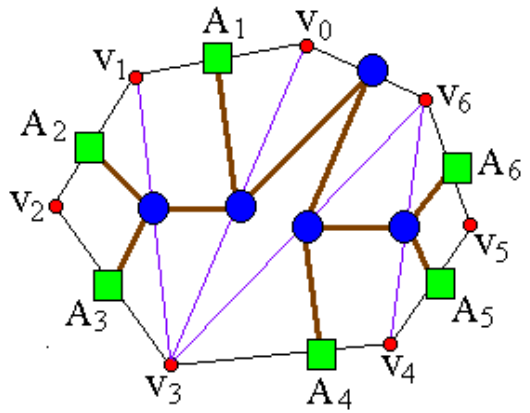
三角剖分的结构及其相关问题

- 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, 如图
  - 一个凸n多边形的三角剖分对应一棵有**n-1**个叶结点的语法树
- 与矩阵连乘积问题的比较
  - 每个矩阵A_i 对应于凸(n+1)边形中的一条边$v_{i-1}v_i$
  - 矩阵连乘积A[i+1:j]对应于三角剖分中的一条弦v_iv_j ，i<j
- 例子 ：凸七边形$\{v_0, v_1, \ldots, v_6\}$， 划分T=$\{v_0v_1, v_1v_3, v_3v_6, v_4v_6\}$

语法树	凸多边形
根节点	边v_0v_6
内节点	三角剖分的弦
叶子节点	除v_0v_6外的各边



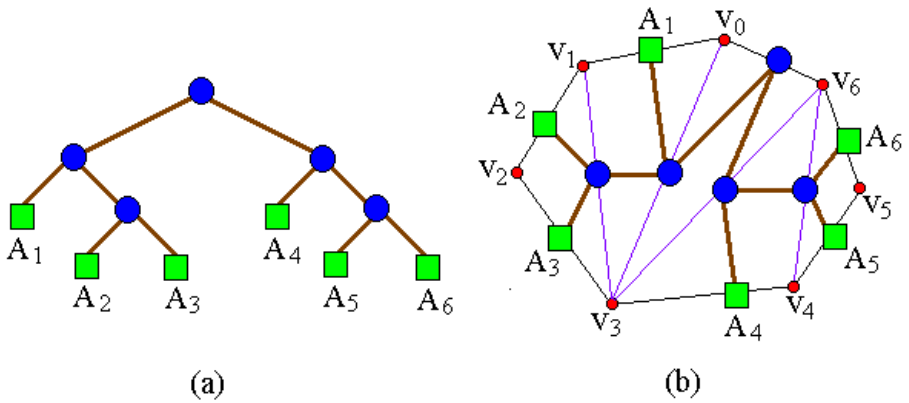
(a)



(b)

三角剖分的结构及其相关问题

- 与矩阵连乘问题比较分析
  - 矩阵链$A_1A_2\ldots A_n$，定义相应凸(n+1)边形P=$\{v_0,v_1,\ldots,v_n\}$
  - 每个矩阵A_i 对应凸多边形边$v_{i-1}v_i$
  - 矩阵连乘积A[i+1,...j]对应三角剖分的一条弦v_iv_j
  - 矩阵A_i的维数是p_{i-1}* p_i，定义三角形$v_iv_jv_k$的权函数值= $p_ip_jp_k$
- 凸多边形P的最优三角剖分对应的语法树给出矩阵链$A_1A_2\ldots A_n$的最优完全加括号形式



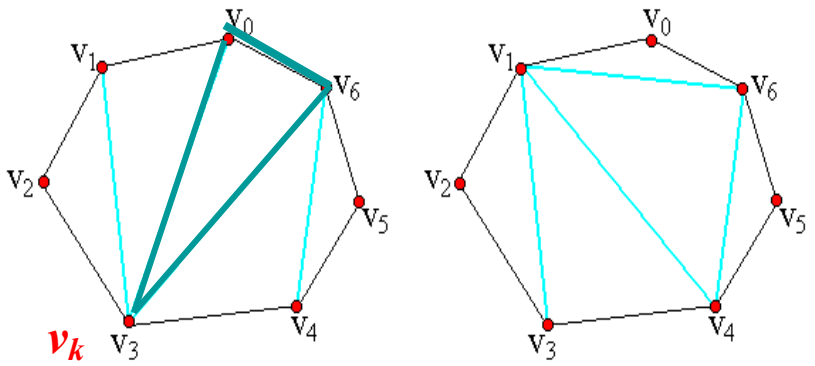
步骤1：最优子结构性质

- 最优三角剖分问题
  - 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$\Sigma_{s\in ST}W(s)$最小，其中ST 是T 所对应的三角形集合
- 分析
  - 是否满足动态规划算法的两个要素？

?子问题是什么

步骤1：最优子结构性质

- 最优子结构分析
  - 若凸(n+1)边形P={$v_0,v_1,\ldots,v_n$}的最优三角剖分T包含三角形$v_0v_kv_n$，1≤k≤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
  - W(T)=W({$v_0,v_1,\ldots,v_k$})+W({$v_k,v_{k+1},\ldots,v_n$})+W({$v_0v_kv_n$})
  - 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。

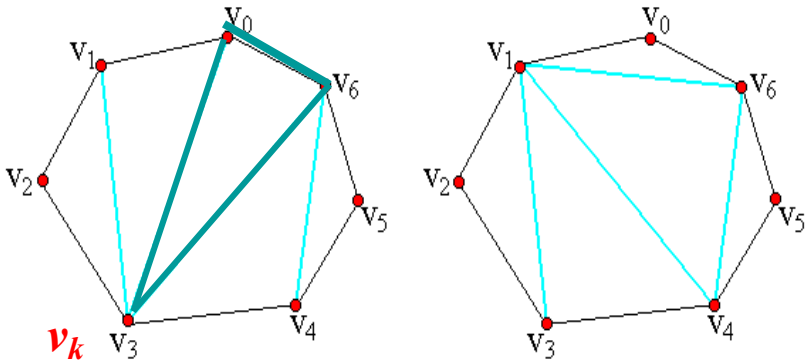


步骤2：最优三角剖分的递归结构

- 子问题 ：求P={v_{i-1},v_i,…,v_j}的最优三角剖分
  - 划分方法：三角形$v_{i-1}v_kv_j$ ， 关键怎么找v_k！
- 求解策略：先找最优权值和，再找剖分
  - 定义t[i][j] (1≤i<j≤n) 为凸子多边形{v_{i-1},v_i,…,v_j}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{v_{i-1},v_i}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]。

原问题： ={v_0,v_i,…,v_n}
子问题： ={v_{i-1},v_i,…,v_j}
子问题个数：$\Theta(n^2)$

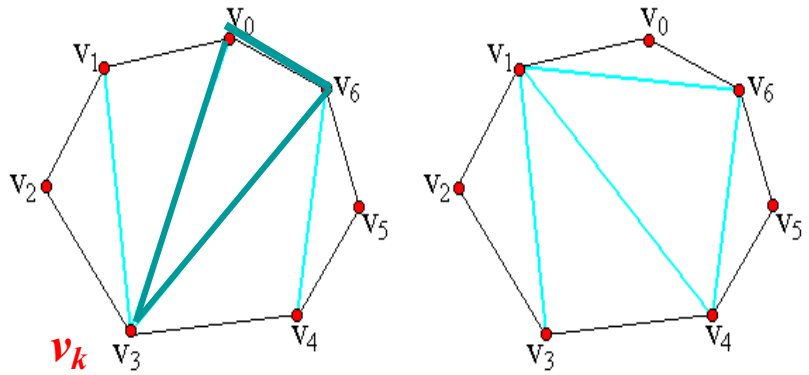
i=j时只有一条线段



步骤2：最优三角剖分的递归结构

- 当j-i≥1时，凸子多边形至少有3个顶点(v_{i-1},v_i, v_j)
- 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，而k的所有可能位置只有j-i个，由此，t[i][j]可递归地定义为：

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



凸多边形最优三角剖分

- 思考题：给定凸六边形，三角形权值为边长之和，假设边长为1，为方便计算，设另两类弦长分别为2和3.请给出该凸六边形的最优三角剖分方法。

- 提示：凸六边形P=$\{v_0, v_1, v_2, v_3, v_4, v_5\}$, 求t[1][5]

小结

- 0-1背包问题
  - 描述
  - 动态规划算法
  - 跳跃点方法
- 凸多边形三角剖分问题
  - 描述
  - 子问题的描述
  - 问题的转化：语法树
- 难点
  - 跳跃点方法