

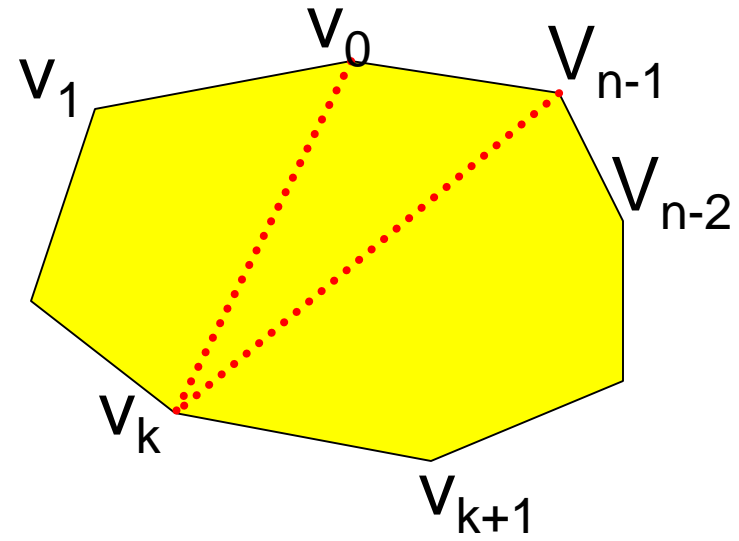
算法设计与分析

3. 动态规划 (Part 2)

3.5 凸多边形最优三角剖分

凸多边形最优三角剖分

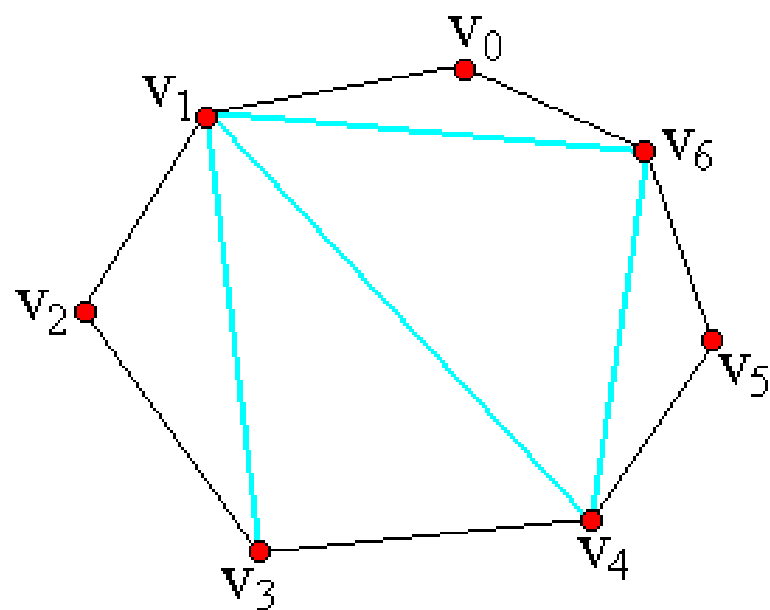
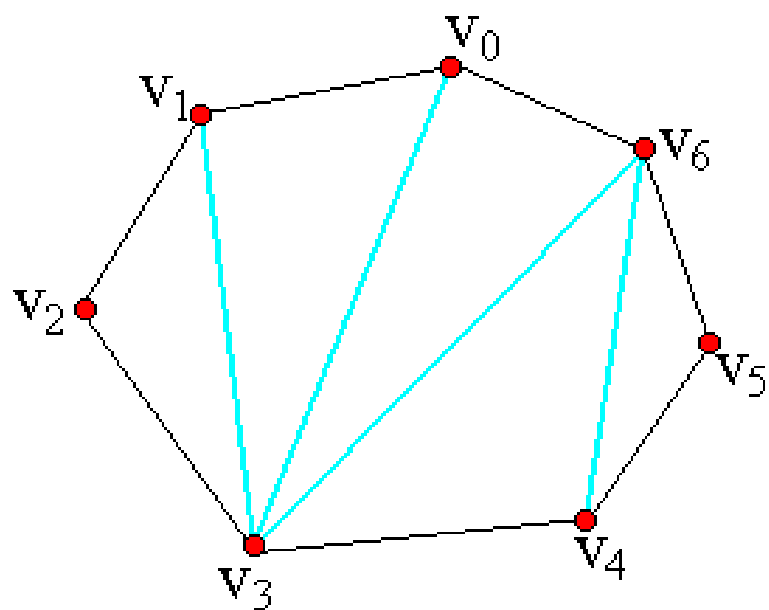
- **凸多边形**: 用多边形顶点的逆时针序列表示凸多边形, 即 $P=\{v_0, v_1, \dots, v_{n-1}\}$ 表示具有 n 条边的凸多边形。
- **弦**: 若 v_i 与 v_j 是多边形上不相邻的2个顶点, 则线段 $v_i v_j$ 称为多边形的一条弦。弦将多边形分割成2个多边形 $\{v_i, v_{i+1}, \dots, v_j\}$ 和 $\{v_j, v_{j+1}, \dots, v_i\}$ 。



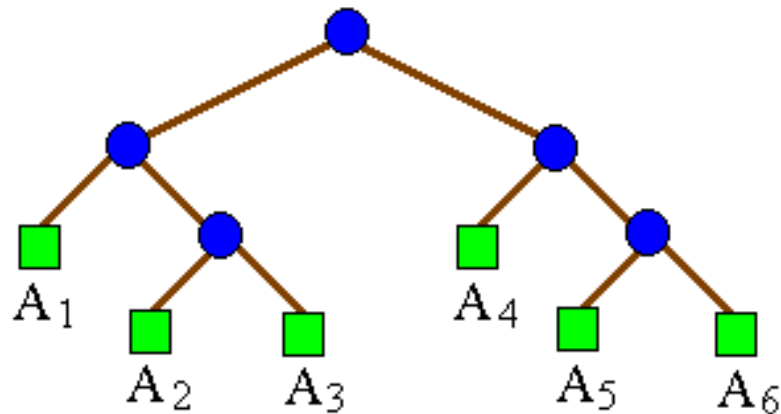
概念

- **多边形的三角剖分**：将多边形分割成互不相交的三角形的弦的集合 T 。
- **凸多边形最优三角剖分**：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，使得该三角剖分中诸三角形上权之和为最小。

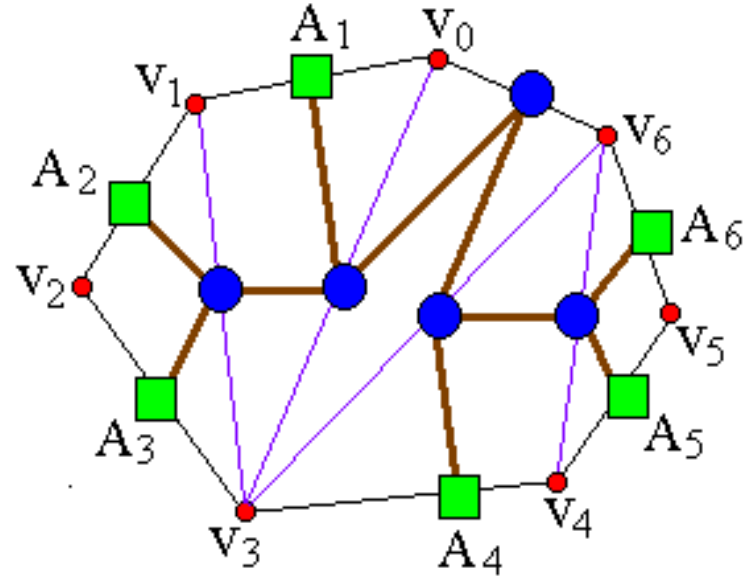
凸多边形三角剖分举例



三角剖分的结构及其相关问题



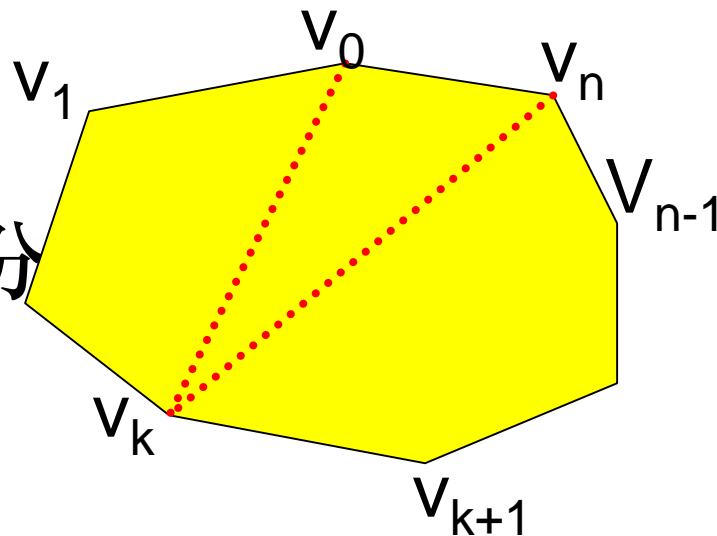
(a)



(b)

- 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 $\{v_0, v_1, \dots, v_{n-1}\}$ 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



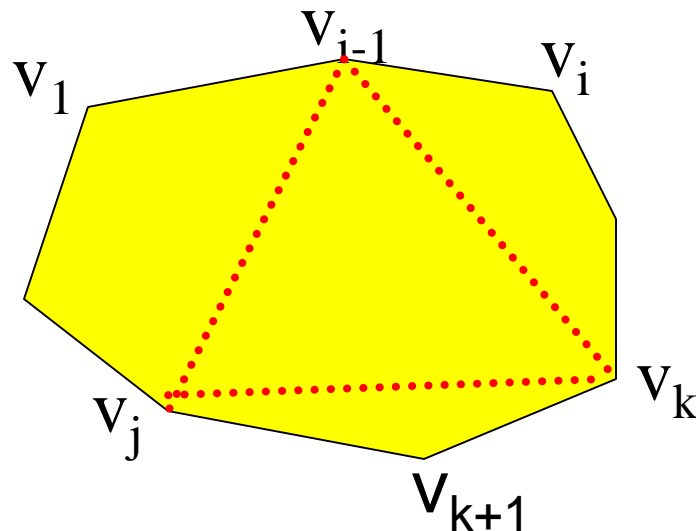
- 凸多边形的最优三角剖分结构性质。
- **理由：**若凸 $(n+1)$ 边形 $P=\{v_0, v_1, \dots, v_n\}$ 的最优三角剖分 T 包含三角形 $v_0 v_k v_n$, $1 \leq k \leq n-1$, 则 T 的权为3个部分权的和：**三角形 $v_0 v_k v_n$ 的权**, 子多边形 **$\{v_0, v_1, \dots, v_k\}$** 和 **$\{v_k, v_{k+1}, \dots, v_n\}$** 的权之和。
- 可以断言, 由 T 所确定的这2个子多边形的三角剖分也是最优的。
- 因为若有 $\{v_0, v_1, \dots, v_k\}$ 或 $\{v_k, v_{k+1}, \dots, v_n\}$ 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 $t[i][j]$, $1 \leq i < j \leq n$ 为凸子多边形 $\{v_{i-1}, v_i, \dots, v_j\}$ 的最优三角剖分所对应的权函数值, 即其最优值。
- 约定: 两个顶点的退化多边形 $\{v_{i-1}, v_i\}$ 具有权值 0。
- 问题转化为: 计算的凸 $(n+1)$ 边形 P 的最优权值为 $t[1][n]$ 。

递归关系

当 $j-i \geq 1$ 时，凸子多边形 $\{v_{i-1}, v_i, \dots, v_j\}$ 至少有3个顶点。对 $i \leq k \leq j-1$ 。



$t[i][j]$ 的值可递归计算：

$$t[i][j] = t[i][k] + t[k+1][j] + (\Delta v_{i-1} v_k v_j \text{ 的权值})$$

k 的确切位置待定，但 k 的所有可能位置只有 $j-i$ 个，从中可选出使 $t[i][j]$ 值达到最小的位置。

$t[i][j]$ 的递归定义

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

最优三角剖分最优值计算

- 仿矩阵连乘积问题
- 复杂性：时间 $O(n^3)$ ，空间 $O(n^2)$

最优三角剖分构造：略

3.9 流水作业调度

流水作业调度

- **问题描述：** n 个作业 $\{1, 2, \dots, n\}$ 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。
- **一般流水作业调度问题：** n 个作业 $\{1, 2, \dots, n\}$ 要在由 m 台机器 $M1, M2, \dots, M_m$ 组成的流水线上完成加工。要求确定这 n 个作业的最优加工顺序，使 m 台机器上作业所需的时间最少。

加工时间矩阵

- n 个作业 $\{1, 2, \dots, n\}$, m 台机器 M_1, M_2, \dots, M_m , 设 M_i 加工作业 j 所需的时间分别为 t_{ij} 。
- 加工时间矩阵: $T = (t_{ij})_{m \times n}$ 。
- n 个作业 $\{1, 2, \dots, n\}$, 2台机器 M_1, M_2 的时间矩阵

$$T = \begin{pmatrix} t_{11} & t_{12} & \dots & t_{1n} \\ t_{21} & t_{22} & \dots & t_{2n} \end{pmatrix}$$

记为:

$$T = \begin{pmatrix} a_1 & a_2 & \dots & a_n \\ b_1 & b_2 & \dots & b_n \end{pmatrix}$$

算法分析: $m=2$ 的情况

- 为叙述方便起见, 以下设
- 作业集合为 $J=\{J_1, J_2, \dots, J_n\}$

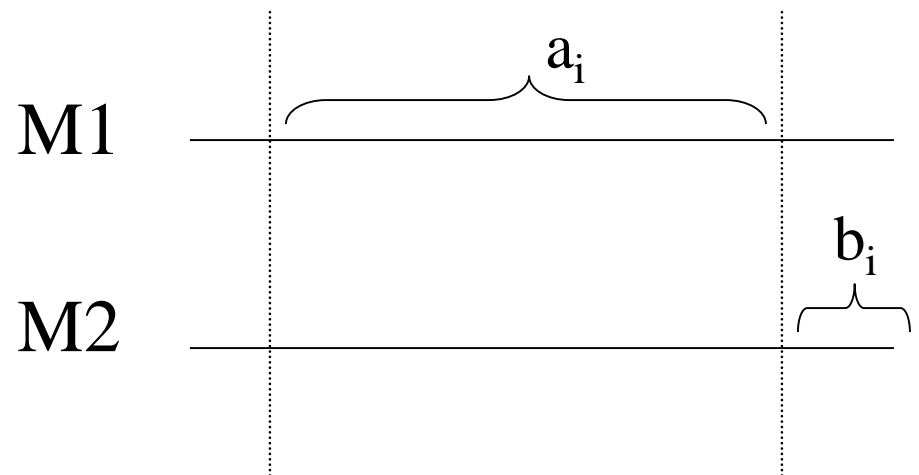
又设 $S \subseteq J, S_0=J$

约定: 为叙述方便。有时将作业集 J 仍用 $\{1, 2, 3, \dots, n\}$ 表示

分析

- 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

- 情况



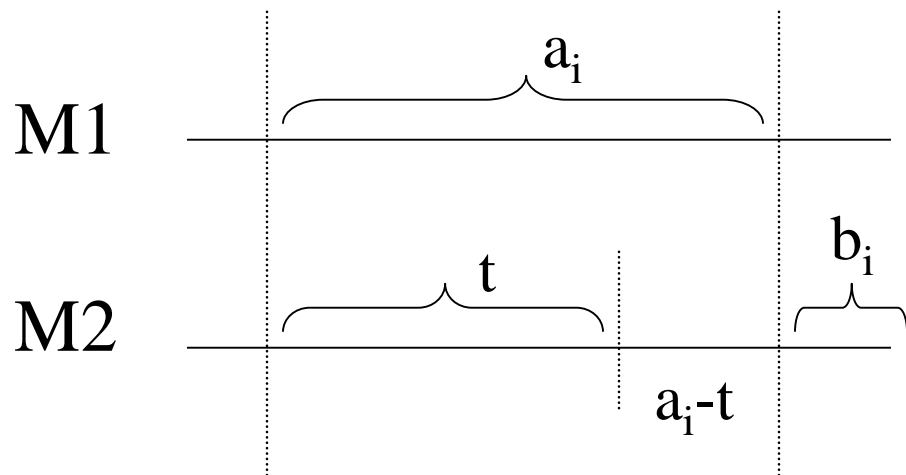
开始时的情况

- 设 π 是所给 n 个流水作业的一个最优调度，即已排好作业调度：
 $\pi(1), \pi(2), \dots, \pi(n)$ ，其中 $\pi(k) \in \{1, 2, \dots, n\}$ ，记 $i = \pi(1)$

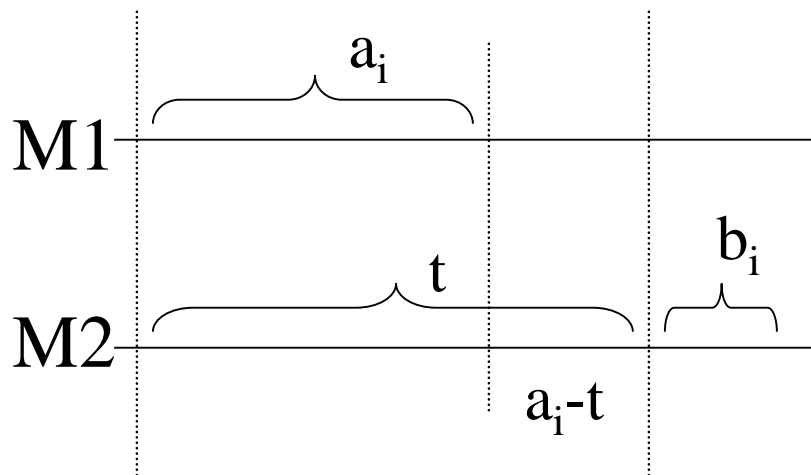
一般情况-在M1做作业i的情况

在M2上未完成前面作业时，M1上做作业i的情况

• 情况1



• 情况2



算法分析

- 一般情况
 - 设机器M1开始加工S中作业时，机器M2还在加工其他作业，要等时间 t 后才可利用,完成S中作业所需的最短时间记为 $T(S,t)$ 。
- 流水作业调度问题变为：求最优值为 $T(J,0)$ 。

最优子结构性质

- 设 π 是所给 n 个流水作业的一个最优调度，即已排好作业调度：

$\pi(1), \pi(2), \dots, \pi(n)$ ，其中 $\pi(i) \in \{1, 2, \dots, n\}$

- 设机器M1开始加工J中第一个作业 $J_{\pi(1)}$ 时，机器M2可能在等待，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 $J_{\pi(2)}, \dots, J_{\pi(n)}$ 所需的时间，**这是最好的时间**。
- 记 $S = J - \{J_{\pi(1)}\}$ ，则可证明：

$$T' = T(S, b_{\pi(1)})。$$

$T'=T(S, b_{\pi(1)})$ 的证明

- 注: $T(S, b_{\pi(1)})$ 是完成S中所有作业的最少时间
- 证明: 由 T' 的定义知 $T' \geq T(S, b_{\pi(1)})$ 。
- 若真 $T' > T(S, b_{\pi(1)})$, 设 π' 是作业集 $S = J - \{J_{\pi(1)}\}$ 在机器M2的等待时间为 $b_{\pi(1)}$ 情况下的一个最优调度。则 $\pi(1), \pi'(2), \dots, \pi'(n)$ 是J的一个调度 π' , 且该调度 π' 所需的时间为 $a_{\pi(1)} + T(S, b_{\pi(1)}) < a_{\pi(1)} + T'$ 。
- 这与 π 是J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。

结论

- $T(J,0)=a_{\pi(1)}+T(J-\{J_{\pi(1)}\},b_{\pi(1)})$

如未知作业 $J_{\pi(1)}$ 是否应该排在第一位，则应考虑所有任务

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

递归关系

- $T(J,0)=\min_{\{1 \leq i \leq n\}}\{a_i+T(J-\{J_i\},b_i)\}$
- 需要计算: $T(J-\{J_i\},b_i)$

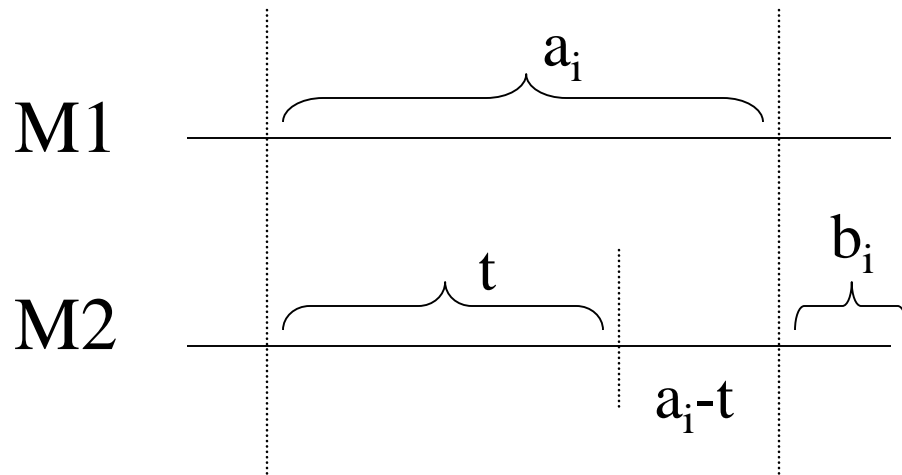
一般情况: 设 $S \subseteq J$, $S_0=J$, 则可证

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

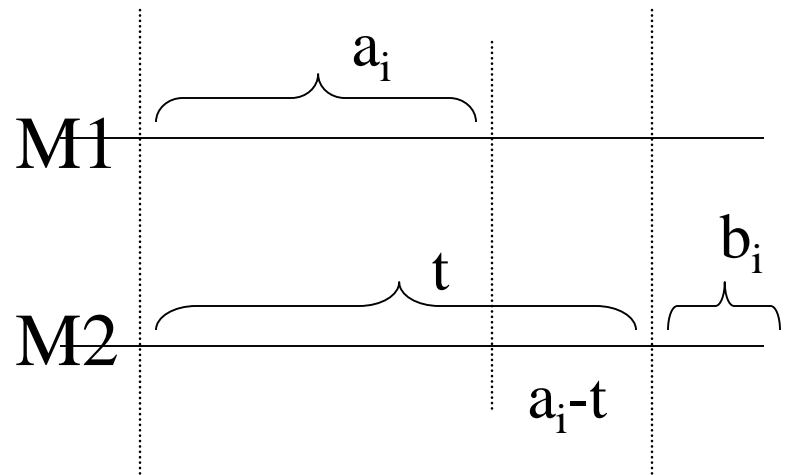
$$T(S,t) = \min\{a_i + T(S - \{J_i\}), b_i + \max(t - a_i, 0)\}$$

图例

• 情况1



• 情况2



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 M_2 的等待时间为 t 时的任一最优调度。若 $\pi(1)=i$, $\pi(2)=j$ 。则由动态规划递归式可得：

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

其中，

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

t_{ij} 演算

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

$$= \begin{cases} t + b_i + b_j - a_i - a_j & \text{若 } \max(t, a_i, a_i + a_j - b_i) = t \\ b_i + b_j - a_i & \text{若 } \max(t, a_i, a_i + a_j - b_i) = a_i \\ b_j & \text{若 } \max(t, a_i, a_i + a_j - b_i) = a_i + a_j - b_i \end{cases}$$

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

当作业i和j满足Johnson不等式时

- 如交换作业 J_i 和作业 J_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

$$T'(S,t) = a_i + a_j + T(S - \{J_i, J_j\}, t_{ji})$$

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 J_i 和 J_j 满足Johnson不等式时，有

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

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

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

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

当作业i和j满足Johnson不等式时

$$T(S,t) \leq T'(S,t)$$

此时，作业 J_i 排在作业 J_j 的前面

即：当作业 J_i 和作业 J_j 不满足Johnson不等式时，
只要交换它们的加工顺序后，不增加加工时间。

流水作业调度的Johnson法则

- 结论：**对于流水作业调度问题，必存在最优调度 π ，使得作业 $J_{\pi(i)}$ 和 $J_{\pi(i+1)}$ 满足Johnson不等式。

Johnson法则：当调度 π ，对任何 i ，作业 $J_{\pi(i)}$ 和 $J_{\pi(i+1)}$ 满足Johnson不等式

$$\min\{b_{\pi(i)}, a_{\pi(i+1)}\} \geq \min\{b_{\pi(i+1)}, a_{\pi(i)}\}$$

称调度 π 满足Johnson法则

可证明：调度 π 满足Johnson法则，当且仅当，对任意 $i < j$ 有

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

结论：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 的升序排序；将 N_2 中作业依 b_i 的降序排序；
- (3) N_1 中作业接 N_2 中作业构成满足Johnson法则的最优调度。

算法举例

| | J_1 | J_2 | J_3 | J_4 | J_5 | J_6 |
|----|-------|-------|-------|-------|-------|-------|
| 印刷 | 3 | 12 | 5 | 2 | 9 | 12 |
| 装订 | 8 | 10 | 9 | 6 | 3 | 1 |

- $N_1 = \{1, 3, 4\}$, $N_2 = \{2, 5, 6\}$
- N_1 按 a_i 升序: J_4, J_1, J_3 ,
- N_2 按 b_i 降序: J_2, J_5, J_6
- 合并: $J_4, J_1, J_3, J_2, J_5, J_6$

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 $O(n \log n)$ 。所需的空间为 $O(n)$ 。

3.10 0-1背包问题

0-1背包问题

- 假设给定 n 个物体和一个背包，物体 i 的重量为 w_i ，价值为 v_i ($i=1,2,\dots,n$)，背包能容纳的物体重量为 c ，要从这 n 个物体中选出若干件放入背包，使得放入物体的总重量小于等于 c ，而总价值达到最大
- 如果用 $x_i=1$ 表示将第 i 件物体放入背包，用 $x_i=0$ 表示未放入，则问题变为选择一组 x_i ($i=0,1$) 使得

$$w_{\mathbf{x}} = \sum_{i=1}^n w_i x_i \leq c, \quad v_{\mathbf{x}} = \sum_{i=1}^n v_i x_i, \quad \text{并且达到最大}$$

0-1背包问题的数学表示

$$\begin{aligned} & \max \sum_{i=1}^n v_i x_i \\ & \begin{cases} \sum_{i=1}^n w_i x_i \leq C \\ x_i \in \{0,1\}, 1 \leq i \leq n \end{cases} \end{aligned}$$

- **0-1背包问题是一个特殊的整数规划问题**

0-1背包问题的最优子结构性质

- 设 (y_1, y_2, \dots, y_n) 是所给0-1背包问题的一个最优解，满足：

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

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

- 则 (y_2, \dots, y_n) 是下面相应子问题的一个最优解：

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

为什么？

$$\begin{cases} \sum_{i=2}^n w_i x_i \leq C - w_1 y_1 \\ x_i \in \{0,1\}, 2 \leq i \leq n \end{cases}$$

递归关系

- 设所给0-1背包问题的子问题

$$\max \sum_{k=i}^n v_k x_k \quad \left\{ \begin{array}{l} \sum_{k=i}^n w_k x_k \leq j \\ x_k \in \{0,1\}, i \leq k \leq n \end{array} \right.$$

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 $i, i+1, \dots, n$ 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，有计算 $m(i, j)$ 的递归式：

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

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

举例：求0-1背包问题

$$\left\{ \begin{array}{l} Z = \max \{ 2x_1 + 3x_2 + 2x_3 \} \\ x_1 + 2x_2 + 5x_3 \leq 6 \\ x_i \in \{0, 1\}, \quad 1 \leq i \leq 3 \end{array} \right.$$

- 答案： $z=6$, $x_1=1$, $x_2=0$, $x_3=1$

0-1背包问题的动态规划法

```
#define jmax 100
#define nmax 100
float m[nmax][jmax];
void Knapsack(float p[],int w[],int c,int n){
    int j=0,i=0;
    for(j=0;j<jmax;j++) m[n][j]=0;
    for(j=w[n];j<=c;j++) m[n][j]=v[n];
    for(i=n-1;i>1;i--){
        for(j=0;j<jmax;j++) m[i][j]=m[i+1][j];
        for(j=w[i];j<=c;j++)
            m[i][j]=max(m[i+1][j],(m[i+1][j-w[i]]+v[i]));
    }
    m[1][c]=m[2][c];
    if(c>=w[1])
        m[1][c]=max(m[1][c],(m[2][c-w[1]]+v[1]));
}
```

得到解向量

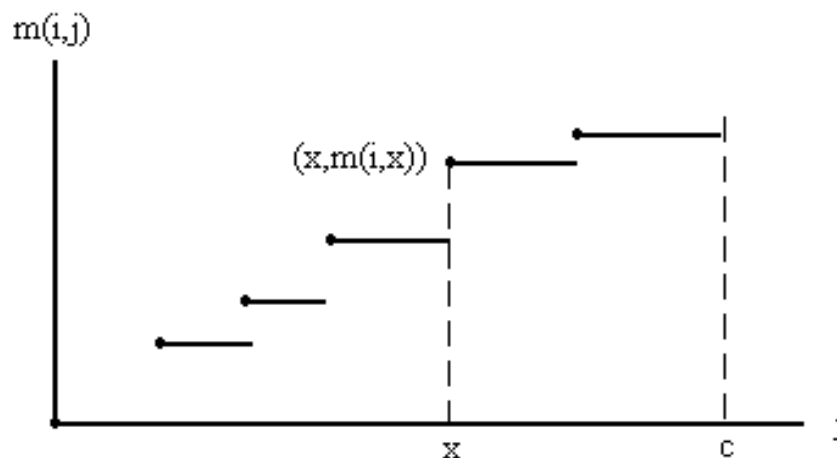
```
void traceback(int w[],int c,int n,int x[])
    /*得到解向量x*/
{   int i=0;
    for(i=1;i<n;i++)
        if(m[i][c]==m[i+1][c])    x[i]=0;
        else{
            x[i]=1;    c-=w[i];
        }
    x[n]=m[n][c]?1:0;
}
```


算法复杂度分析

- 从 $m(i, j)$ 的递归式容易看出，程序有两次循环，一次关于 i ($\leq n$)，一次关于 j ($\leq c$)。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--自学

- 由 $m(i,j)$ 的递归式，易证在一般情况下，对每一个确定的 $i(1 \leq i \leq n)$ ，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 $m(i,j)$ 由其全部跳跃点惟一确定。如图所示。



对每一个确定的 $i(1 \leq i \leq n)$ ，用一个表 $p[i]$ 存储函数 $m(i, j)$ 的全部跳跃点。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 $p[n+1]=\{(0, 0)\}$ 。

跳过

旅行推销员问题--补充

- **问题描述：**设有 n 个城市，已知任意两城市间之距离。现有一推销员想从某一城市出发巡回经过每一城市（且每城市只经过一次），最后又回到出发点，问如何找一条最短路径。

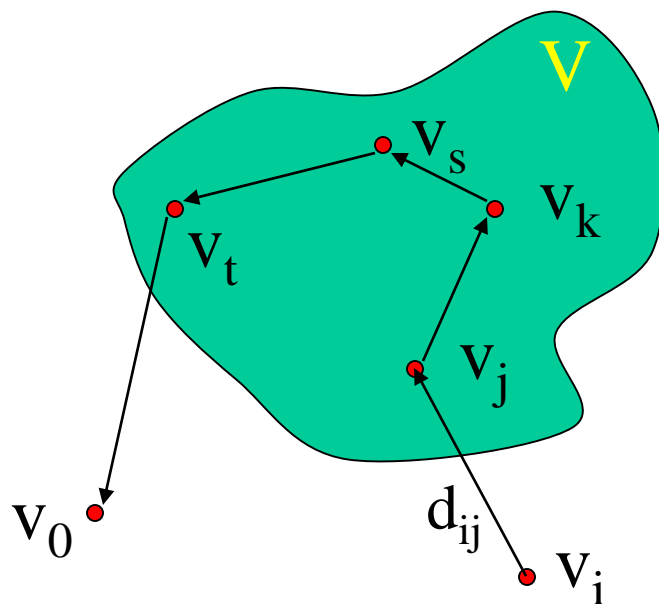
记号

- 设城市 v_i 与 v_j 城市的代价为 d_{ij} 。用 $d_{ij}=\max$ (或 ∞)表示 v_i 与到 v_j 无通路。
- 用 D 记代价矩阵,
- $V_0=\{v_0, v_1, v_2, \dots, v_n\}$, $V \subseteq V_0$

例: $D = \begin{pmatrix} 0 & 8 & 5 & 6 \\ 6 & 0 & 8 & 5 \\ 7 & 9 & 0 & 5 \\ 9 & 7 & 8 & 0 \end{pmatrix}$

算法分析

- 设 $T(v_i, V)$ 表示从点 v_i 出发，经过 V 中点各一次，最后返回到起点 v_0 的最短路径长。



- 则
$$T(v_i, V) = \min_{v_j \in V} \{ d_{ij} + T(v_j, V \setminus \{v_i\}) \}$$

旅行推销员问题求法

- 要求: $T(v_0, \{v_1, v_2, \dots, v_n\})$

利用:

$$T(v_0, V) = \min_{v_j \in V} \{d_{ij} + T(v_j, V \setminus \{v_i\})\}$$

可采用自顶向下记忆式求解方法，用数组 pos 记录商人的走向。

伪代码

```
Function T(k:integer;V:set of 1..n):real;  
  Var a,b:real; i,j:integer;  
      next[1..n,set of [1..n]]:integer;  
Begin  if (V=[]) then a:=d[i,0]  
      else begin a:=inf;  
              for i:=1 to n do  
                if (i in V) then do  
                  begin  
                    b:=d[k,I]+T(i,V\{i});  
                    if a>b then  
                      begin a:=b; next[k,V]:=i; end;  
                    end;  
                  end;  
                end;  
              return a;  
            End;
```

Pos的确定

Procedure route;

Begin V:=[1..n];

pos[0]:=0; i:=0;

while (V<>[]) do

begin

V:=V\[pos[i]];

i:=i+1;

pos[i]:=next[pos[i-1],V];

end;

End;

复杂性

- 对 i , 确定 $T(i, V)$ 有多大?

- 第1层 $T(0, V)$

1个

- 第2层 $T(i, V \setminus \{i\})$

nC_{n-1}^{n-1} 个

- 第3层 $T(i, V \setminus \{i, j\})$

nC_{n-1}^{n-2} 个

- 第 k 层 $T(i, V \setminus \{i, j\})$

nC_{n-1}^{n-k+1} 个

- 总数 $=1+n2^{n-1}$

练习

1. 根据所给条件求0-1背包问题，要求写出完整求解过程（自底向上）。

$$\left\{ \begin{array}{l} F = \max \{ 3x_1 + 5x_2 + 4x_3 \} \\ 3x_1 + 2x_2 + 4x_3 \leq 10 \\ x_i \in \{0, 1\}, \quad 1 \leq i \leq 3 \end{array} \right.$$