第 3 章 动态规划

3.1 内 容 提 要

1. 基本概念

动态规划(dynamic programming)是一种求解多阶段决策(优化)问题的算法设计技术,其主要思想是:将原问题归约为规模较小、结构相同的子问题,建立原问题与子问题优化函数间的依赖关系. 从规模最小的子问题开始,利用上述依赖关系求解规模更大的子问题,直到得到原始问题的解为止.

动态规划算法的适用条件 适用于求解多阶段决策(优化)问题,该问题的解可以表示为一个决策序列,且满足优化原则(或最优子结构性质),即一个最优决策序列的任何子序列本身一定是相对于子序列的初始和结束状态的最优决策序列.

对于规模大的实例,对存储空间的需求是该算法的瓶颈.

主要设计步骤

1. 写出所要求解的组合优化问题的目标函数和约束条件.

2. 确定子问题的结构与边界,将问题求解转变成多步判断的过程.

3. 定义优化函数,以该函数的极大值或者极小值作为判断的依据,确定是否满足优化原则.

4. 列出有关优化函数的递推关系和边界条件.

5. 根据问题的解的不同情况,考虑是否需要设立标记函数.

6. 从初值开始,自底向上计算每个子问题的优化函数值(有的需要同时计算标记函数),并以备忘录的方式存储所有的中间结果.

7. 如果有标记函数,则根据标记函数逐步追踪问题的解.

2. 时间复杂度的分析方法

时间复杂度取决于备忘录(包括优化函数及标记函数)中每个项的计算工作量,以及用标记函数追踪解的工作量. 大多数情况下,追踪解的工作量不超过优化函数计算的工作量,因此可以根据递推关系确定备忘录中每个项的计算工作量,然后对这些工作量求和.

3. 典型的动态规划算法

矩阵链相乘 MatrixChain(P,n) 给定矩阵链 $A_1..n=A_1A_2\cdots A_n$,其行和列数为向量 $P=<P_0,P_1,\cdots,P_n>$,其中 P_0 是 A_1 的行数,$P_i(i=1,2,\cdots,n-1)$是 A_i 的列数和 A_{i+1}的行数,P_n 是 A_n 的列数. 通过加括号确定具有最少乘法次数的运算顺序.

设子问题 $A_{i..j}$ 是矩阵链 $A_iA_{i+1}\cdots A_j$ 的计算，令 $m[i,j]$ 是计算 $A_{i..j}$ 的最少的乘法次数，则

$$m[i,j]=\begin{cases}0 & i=j\\ \min\limits_{i\leqslant k<j}\{m[i,k]+m[k+1,j]+P_{i-1}P_kP_j\} & i<j\end{cases},\ 1\leqslant i\leqslant j\leqslant n$$

用标记函数 $s[i,j]$ 记录使得优化函数 $m[i,j]$ 达到最小值时的 k 值，可用于追踪加括号的位置，备忘录是 $n\times n$ 的 $m[i,j]$ 表和 $s[i,j]$ 表，$i,j=1,2,\cdots,n$，时间是 $O(n^3)$.

背包问题(knapsack problem)　一个旅行者准备随身携带一个背包. 可以放入背包的物品有 n 种，每种物品的重量和价值分别为 w_j 和 v_j，$j=1,2,\cdots,n$. 如果每种物品放入背包的数量不限，背包的最大重量限制是 b，怎样选择放入背包的物品以使得背包的价值最大？

使用组合优化问题的描述方法，设 x_j 表示装入背包的第 j 种物品的数量，背包问题可以描述为

$$\begin{aligned}&\text{目标函数}\quad \max\sum_{j=1}^{n}v_jx_j\\ &\text{约束条件}\quad \sum_{j=1}^{n}w_jx_j\leqslant b\\ &\qquad\qquad x_j\in\mathbf{N},\quad j=1,2,\cdots,n\end{aligned}$$

当每种物品只有 1 个，即 x_j 只能取 0 或 1 时，称为 **0-1 背包问题.**

使用动态规划的方法求解背包问题. 设 $F_k(y)$ 表示只允许装前 k 种物品，背包总重不超过 y 时背包的最大价值，则

$$\begin{aligned}&F_k(y)=\max\{F_{k-1}(y),F_k(y-w_k)+v_k\},\ k=1,2,\cdots,n,\ y=1,2,\cdots,b\\ &F_0(y)=0 \qquad 0\leqslant y\leqslant b\\ &F_k(0)=0 \qquad 0\leqslant k\leqslant n\\ &F_k(y)=-\infty \qquad y<0\end{aligned}$$

根据上面的递推式，当 $k=1$ 时，可以得到

$$F_1(y)=\left\lfloor\frac{y}{w_1}\right\rfloor v_1$$

令 $i_k(y)$ 表示在计算优化函数值 $F_k(y)$ 时所用到物品的最大标号，则

$$i_k(y)=\begin{cases}i_{k-1}(y) & F_{k-1}(y)>F_k(y-w_k)+v_k\\ k & F_{k-1}(y)\leqslant F_k(y-w_k)+v_k\end{cases},\ 1<k\leqslant n,\ y=1,2,\cdots,b$$

$$i_1(y)=\begin{cases}1 & y\geqslant w_1\\ 0 & y<w_1\end{cases}$$

该算法最坏情况下的时间复杂度为 $O(nb)$.

最长公共子序列 LCS(X,Y,m,n)　给定长度分别为 m 和 n 的序列 X 和 Y，求 X 与 Y 的最长公共子序列 Z.

设 $X_i=<x_1,x_2,\cdots,x_i>$，$Y_j=<y_1,y_2,\cdots,y_j>$，设 $C[i,j]$ 表示 X_i 与 Y_j 的最长公共子序列的长度，则

$$C[i,j]=\begin{cases}C[i-1,j-1]+1 & \text{如果 } i,j>0,\ x_i=y_j\\ \max\{C[i,j-1],C[i-1,j]\} & \text{如果 } i,j>0,\ x_i\neq y_j\end{cases}$$

$$i=1,2,\cdots,m,\quad j=1,2,\cdots,n$$

$$C[0,j]=C[i,0]=0,\quad \text{对一切 } 1\leqslant i\leqslant m,\ 1\leqslant j\leqslant n$$

令 $B[i,j]$为标记函数. 在归约为子问题时,如果 $C[i,j]$归约为 $C[i-1,j-1]$,则 $B[i,j]$="↖";若归约为 $C[i,j-1]$,则 $B[i,j]$="←";若归约为 $C[i-1,j]$,则 $B[i,j]$="↑". 算法最坏情况下的时间复杂度为 $O(mn)$.

图像压缩 Compress(P,n) 给定图像的灰度值序列 $P=<p_1,p_2,\cdots,p_n>$,其中 p_i 为第i 个像素的灰度值,采用变位压缩存储格式,如何对 P 进行分段,以使得存储 P 占用的二进制位数达到最少?

设 $S[i]$表示输入为$<p_1,p_2,\cdots,p_i>$时按最优分段存储所使用的二进制位数,则

$$S[i]=\min_{1\leqslant j\leqslant \min\{i,256\}}\{S[i-j]+j*b[i-j+1,i]+11\}\quad i=2,3,\cdots,n$$

$$S[1]=b[1,1]+11$$

其中 $b[i-j+1,i]$表示最后一段$<p_{i-j+1},\cdots,p_i>$中的最大灰度值在存储时所占用的二进制位数. 11 是存储该分段信息的额外开销. 用标记函数 $l[i]$记录进行最优划分时最后一段的像素个数,可用于追踪达到最小存储位数时的分段位置. 该算法最坏情况下的时间复杂度为 $O(n)$.

最大子段和 MaxSum(A,n) 给定 n 个整数的序列 $A=<a_1,a_2,\cdots,a_n>$,求

$$\max\Big\{0,\max_{1\leqslant i\leqslant j\leqslant n}\sum_{k=i}^{j}a_k\Big\}$$

定义 $C[i]$是输入 $A[1..i]$中必须包含元素 $A[i]$的最大子段和,最优解为 OPT(A),则

$$C[i]=\max\{C[i-1]+A[i],A[i]\}\quad i=2,\cdots,n$$

$$C[1]=A[1]$$

$$\mathrm{OPT}(A)=\max_{1\leqslant i\leqslant n}\{C[i]\}$$

该算法的时间复杂度是 $O(n)$.

最优二分检索树 设 $S=<x_1,x_2,\cdots,x_n>$是数据集,称$(-\infty,x_1),(x_1,x_2),\cdots,(x_{n-1},x_n),(x_n,+\infty)$为第 0,1,2,…,$n$ 个空隙. 与 S 相关的存取概率分布是 $P=<a_0,b_1,a_1,b_2,\cdots,a_i,b_{i+1},\cdots,b_n,a_n>$,其中 x 处在 x_i 的概率是 $b_i(i=1,2,\cdots,n)$,处在第 j 个空隙的概率是 $a_j(j=0,1,\cdots,n)$. 求一棵关于 S 的二分检索树,使得平均比较次数

$$t=\sum_{i=1}^{n}b_i(1+d(x_i))+\sum_{j=0}^{n}a_jd(L_j)$$

达到最小,其中 $d(x_i)$表示树中数据结点 x_i 的深度,$d(L_j)$表示树中空隙结点 L_j 的深度.

令 $S[i,j]=<x_i,x_{i+1},\cdots,x_j>$表示 S 以 i 和 j 作为边界的子数据集,$P[i,j]=<a_{i-1},b_i,a_{i+1},b_{i+1},\cdots,b_j,a_{j+1}>$是子数据集 $S[i,j]$所对应的存取概率分布,$P[i,j]$与 $S[i,j]$一起构成以 i 和 j 作为边界的子问题. $m[i,j]$是针对这个子问题的最优二分检索树的平均比较次数,则

$$m[i,j]=\min_{i\leqslant k\leqslant j}\{m[i,k-1]+m[k+1,j]+w[i,j]\}\quad 1\leqslant i\leqslant j\leqslant n$$

$$m[i,i-1]=0\quad i=1,2,\cdots,n$$

其中

$$w[i,j]=\sum_{p=i-1}^{j}a_p+\sum_{q=i}^{j}b_q$$

是 $P[i,j]$中所有元素(包括数据元素与空隙)的概率之和. 算法最坏情况下的时间复杂度是 $O(n^3)$.

3.2 习　　题

对于本章与算法设计有关的习题,解题要求如下:必须对优化函数和标记函数的含义给出说明,列出子问题计算中所使用关于优化函数及标记函数的递推关系和初值. 根据题目要求给出迭代实现的伪码并估计算法的复杂度. 如果题目给出具体的输入实例,应该根据给定实例进行计算并给出相关的备忘录表和最后的解.

3.1 用动态规划算法求解下面的组合优化问题,

$$\max \quad g_1(x_1)+g_2(x_2)+g_3(x_3)$$
$$x_1^2+x_2^2+x_3^2 \leqslant 10$$
$$x_1,x_2,x_3 \text{ 为非负整数}$$

其中函数 $g_1(x),g_2(x),g_3(x)$的值给在表 3.1 中.

表 3.1　函数值

x	$g_1(x)$	$g_2(x)$	$g_3(x)$	x	$g_1(x)$	$g_2(x)$	$g_3(x)$
0	2	5	8	2	7	16	17
1	4	10	12	3	11	20	22

3.2 设 $A=<x_1,x_2,\cdots,x_n>$是 n 个不等的整数构成的序列,A 的一个单调递增子序列是序列$<x_{i_1},x_{i_2},\cdots,x_{i_k}>$,使得 $i_1<i_2<\cdots<i_k$,且 $x_{i_1}<x_{i_2}<\cdots<x_{i_k}$. 子序列$<x_{i_1},x_{i_2},\cdots,x_{i_k}>$的长度是含有的整数个数 k. 例如 $A=<1,5,3,8,10,6,4,9>$,它的长为 4 的递增子序列是:$<1,5,8,10>$,$<1,5,8,9>$,…. 设计一个算法求 A 的一个最长的单调递增子序列,分析算法的时间复杂度. 设算法的输入实例是 $A=<2,8,4,-4,5,9,11>$,给出算法的计算过程和最后的解.

3.3 有 n 个底面为长方形的货柜需要租用库房存放. 如果每个货柜都必须放在地面上,且所有货柜的底面宽度都等于库房的宽度,那么第 i 个货柜占用库房面积大小只需要用它的底面长度 l_i 来表示,$i=1,2,\cdots,n$. 设库房总长度是 $D\left(l_i \leqslant D \text{ 且 } \sum_{i=1}^{n} l_i > D\right)$. 设第 i 号货柜的仓储收益是 v_i,若要求库房出租的收益达到最大,问如何选择放入库房的货柜?若 $l_1,l_2,\cdots,l_n,D$ 都是正整数,设计一个算法求解这个问题,给出算法的伪码描述并估计算法最坏情况下的时间复杂度.

3.4 设有 n 项任务,加工时间分别表示为正整数 $t_1,t_2,\cdots,t_n$. 现有 2 台同样的机器,从 0 时刻开始安排对这些任务的加工. 规定只要有待加工的任务,任何机器就不得闲置. 如果直到时刻 T 所有任务都完成了,总的加工时间就等于 T. 设计一个算法找到使得总加工时间 T 达到最小的调度方案. 设给定实例如下:

$$t_1=1,\quad t_2=5,\quad t_3=2,\quad t_4=10,\quad t_5=3$$

试给出一个加工时间最少的调度方案. 给出计算过程和问题的解.

3.5 设有 n 种不同面值的硬币,第 i 种硬币的币值是 v_k(其中 $v_1=1$),重量是 w_i,$i=1,2,\cdots,n$ 且现在购买某些总价值为 y 的商品,需要用这些硬币付款,如果每种钱币使用的个数不限,那么如何选择付款的方法使得付出钱币的总重量最轻?设计一个求解该问题的算法,给出算法的伪码描述并分析算法的时间复杂度.假设问题的输入实例是:

$$v_1=1,\quad v_2=4,\quad v_3=6,\quad v_4=8$$
$$w_1=1,\quad w_2=2,\quad w_3=4,\quad w_4=6$$
$$y=12$$

给出算法在该实例上计算的备忘录表和标记函数表,并说明付线的方法.

3.6 n 种币值 x_1, x_2, …, x_n 和总钱数 M 都是正整数.如果每种币值的钱币至多使用1次,问:对于 M 是否可以有一种找零钱的方法?设计一个算法求解上述问题.说明算法的设计思想,分析算法最坏情况下的时间复杂度.

3.7 在一条直线的公路两旁有 n 个位置 x_1, x_2, …, x_n 可以开商店,在位置 x_i 开商店的预期收益是 p_i, $i=1,2,\cdots,n$.如果任何两个商店之间的距离必须至少为 d 千米,那么如何选择开设商店的位置使得总收益达到最大?

(1) 用组合最优化方法对该问题建模,写出目标函数与约束条件.

(2) 设计一个算法求解该问题,说明算法设计思想,分析算法最坏情况下的时间复杂度.

3.8 设 $A=\{a_1,a_2,\cdots,a_n\}$ 是正整数的集合,且 $\sum_{i=1}^{n}a_i=N$,设计一个算法判断是否能够把 A 划分成两个子集 A_1 和 A_2,使得 A_1 中的数之和与 A_2 中的数之和相等?说明算法的设计思想,估计算法最坏情况下的时间复杂度.

3.9 有 n 项作业的集合 $J=\{1,2,\cdots,n\}$,每项作业 i 有加工时间 $t(i)\in\mathbf{Z}^+$,$t(1)\leqslant t(2)\leqslant\cdots\leqslant t(n)$,效益值 $v(i)$,任务的结束时间 $D\in\mathbf{Z}^+$,其中 $\mathbf{Z}^+$ 表示正整数集合.一个可行调度是对 J 的子集 A 中任务的一个安排,对于 $i\in A$,$f(i)$ 是开始时间,且满足下述条件:

$$f(i)+t(i)\leqslant f(j)\text{ 或者 }f(j)+t(j)\leqslant f(i),\ j\neq i,\ i,j\in A$$
$$\sum_{k\in A}t(k)\leqslant D$$

设机器从0时刻开动,只要有作业就不闲置,求具有最大总效益的调度.给出算法的伪码,分析算法的时间复杂度.

3.10 把0-1背包问题加以推广.设有 n 种物品,第 i 种物品的价值是 v_i,重量是 w_i,体积是 c_i,且装入背包的重量限制是 W,体积是 V.问如何选择装入背包的物品使得其总重不超过 W,总体积不超过 V 且价值达到最大?设计一个动态规划算法求解这个问题,说明算法的时间复杂度.

3.11 有 n 个分别排好序的整数数组 $A_0,A_1,\cdots,A_{n-1}$,其中 A_i 含有 x_i 个整数,$i=0,1,\cdots,n-1$.已知这些数组顺序存放在一个圆环上,现在要将这些数组合并成一个排好序的大数组,且每次只能把两个在圆环上处于相邻位置的数组合并.问如何选择这 $n-1$ 次合并的次序以使得合并时在最坏情况下总的比较次数达到最少?设计一个动态规划算法求解这个问题,说明算法的时间复杂度.

3.12 设 A 是顶点为 $1,2,\cdots,n$ 的凸多边形,可以用不在内部相交的 $n-3$ 条对角线将

A 划分成三角形，图 3.1 就是五边形的所有的划分方案. 假设凸 n 边形的边及对角线的长度 d_{ij} 都是给定的正整数，$1\leqslant i<j\leqslant n$. 划分后三角形 ijk 的权值等于其周长，求具有最小权值的划分方案. 设计一个动态规划算法求解这个问题，说明算法的时间复杂度.

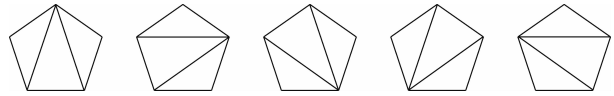

图 3.1　五边形的划分方案

3.13　图的连通性问题是图论研究的重要问题之一，在实际中有着广泛的应用. 例如通信网络的连通问题，运输路线的规划问题等. 一个著名的检查图的连通性的算法就是 Warshall 算法. 假设 $D=<V,E>$ 是顶点集为 $V=\{x_1,x_2,\cdots,x_n\}$，边集为 E 的有向图. $n\times n$ 的 0-1 矩阵 $M=(r_{ij})$ 是 D 的矩阵表示. 考虑 $n+1$ 个矩阵构成的序列 $M_0,M_1,\cdots,M_n$，将矩阵 M_k 的 i 行 j 列的元素记作 $M_k[i,j]$. 对于 $k=0,1,\cdots,n$，$M_k[i,j]=1$ 当且仅当在图中存在一条从 x_i 到 x_j 的路径，并且这条路径除端点外中间只经过 $\{x_1,x_2,\cdots,x_k\}$ 中的顶点. 不难看出 M_0 就是 M，而在 M_n 中如果 $M_n[i,j]=1$，则说明 D 中 x_i 与 x_j 是连通的. Warshall 算法从 M_0 开始，顺序计算 $M_1,M_2,\cdots$，直到 M_n 为止. 利用动态规划的迭代实现方法来实现 Warshall 算法，给出算法的伪码表示并分析算法的时间复杂度. 假设某有向网络的结点是 a,b,c,d，已知网络的矩阵表示是

$$M=\begin{bmatrix}0&1&1&0\\0&0&1&0\\0&0&0&1\\0&0&1&0\end{bmatrix}$$

给出算法在这个实例的计算过程和结果.

3.14　考虑 3.3.7 节中提到的带权编辑距离问题. 设 $S_1[1..n]$ 和 $S_2[1..m]$ 表示两个序列，假设插入和删除操作的权是 d，替换操作的权是 r. 令 $C[i,j]$ 表示序列 $S_1[1..i]$ 和 $S_2[1..j]$ 的最小权编辑距离，设计一个算法求解该问题，给出关于 $C[i,j]$ 的递推关系并分析算法的时间复杂度.

3.15　某机器每天接受大量加工任务，第 i 天需要加工的任务数是 x_i. 随着机器连续运行时间的增加，处理能力越来越低，需要花 1 天时间对机器进行检修，以提高处理能力. 检修当天必须停工，重启后的第 i 天能够加工的任务数是 s_i，且满足

$$s_1>s_2>\cdots>s_n>0$$

我们的问题是：给定 $x_1,x_2,\cdots,x_n,s_1,s_2,\cdots,s_n$，如何安排机器的检修时间，以使得在 n 天内加工的任务数达到最大？设计一个算法求解该问题.

3.16　设 P 是一台 Internet 上的 Web 服务器. $T=\{1,2,\cdots,n\}$ 是 n 个下载请求的集合，$\forall i\in T,a_i\in\mathbf{Z}^+$ 表示下载请求 i 所申请的带宽. 已知服务器的最大带宽是正整数 K. 我们的目标是使带宽得到最大限度的利用，即确定 T 的一个子集 S，使得 $\sum\limits_{i\in S}a_i\leqslant K$，且 $K-\sum\limits_{i\in S}a_i$ 的值达到最小. 设计一个算法求解服务器下载问题，用文字说明算法的主要设计思想和步骤，给出最坏情况下的时间复杂度.

3.17 有正实数构成的数字三角形排列形式如图3.2所示. 第一行的数为a_{11};第二行的数从左到右依次为a_{21},a_{22};以此类推,第n行的数为a_{n1},a_{n2},…,a_{nn}. 从a_{11}开始,每一行的数a_{ij}只有两条边可以分别通向下一行的两个数$a_{(i+1)j}$和$a_{(i+1)(j+1)}$. 请设计一个算法,计算出从a_{11}通到a_{n1},a_{n2},…,a_{nn}中某个数的一条路径,并且使得该路径上的数之和达到最小.

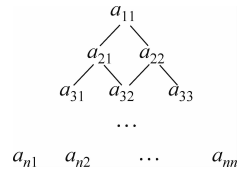

图3.2 数字三角形

3.18 设集合$A=\{a,b,c\}$,A中元素的运算满足$aa=ab=bb=b$,$ac=bc=ca=a$,$ba=cb=cc=c$. 给定A中n个字符组成的串$x_1x_2\cdots x_n$,设计一个算法,检查是否存在一种运算顺序使得这个串的运算结果等于a. 如果存在,则回答"1",否则回答"0". 例如串$x=bbbba$,因为存在下述运算顺序,使得

$$(b(bb))(ba)=(bb)(ba)=b(ba)=bc=a$$

因此回答"1". 而对串bca,由于$(bc)a=aa=b$,$b(ca)=ba=c$,因此回答"0". 说明算法的设计思想,并给出最坏情况下的时间复杂度.

3.3 习题解答与分析

3.1 设$F_k(y)$表示对$x_1,x_2,\cdots,x_k$赋值,且$x_1^2+x_2^2+\cdots+x_k^2\leqslant y$时所得到的目标函数的最大值,递推关系和初值是

$$F_k(y)=\max_{0\leqslant x_k\leqslant\lfloor\sqrt{y}\rfloor}\{F_{k-1}(y-x_k^2)+g_k(x_k)\}$$

$$F_1(y)=g_1(\lfloor\sqrt{y}\rfloor)$$

针对给定实例的计算过程如下.

$k=1$:

$F_1(0)=2$ $x_1=0$

$F_1(1)=F_1(2)=F_1(3)=g_1(1)=4$ $x_1=1$

$F_1(4)=F_1(5)=F_1(6)=F_1(7)=F_1(8)=g_1(2)=7$ $x_1=2$

$F_1(9)=F_1(10)=g_1(3)=11$ $x_1=3$

$k=2$:

$F_2(0)=\max\{F_1(0)+g_2(0)\}=7$ $x_2=0$

$F_2(1)=\max\{F_1(1)+g_2(0),F_1(0)+g_2(1)\}=12$ $x_2=1$

$F_2(2)=\max\{F_1(2)+g_2(0),F_1(1)+g_2(1)\}=14$ $x_2=1$

$F_2(3)=\max\{F_1(3)+g_2(0),F_1(2)+g_2(1)\}=14$ $x_2=1$

$F_2(4)=\max\{F_1(4)+g_2(0),F_1(3)+g_2(1),F_1(0)+g_2(2)\}=18$ $x_2=2$

$F_2(5)=\max\{F_1(5)+g_2(0),F_1(4)+g_2(1),F_1(1)+g_2(2)\}=20$ $x_2=2$

$F_2(6)=\max\{F_1(6)+g_2(0),F_1(5)+g_2(1),F_1(2)+g_2(2)\}=20$ $x_2=2$

$F_2(7)=\max\{F_1(7)+g_2(0),F_1(6)+g_2(1),F_1(3)+g_2(2)\}=20$ $x_2=2$

$F_2(8)=\max\{F_1(8)+g_2(0),F_1(7)+g_2(1),F_1(4)+g_2(2)\}=23$ $x_2=2$

$F_2(9)=\max\{F_1(9)+g_2(0),F_1(8)+g_2(1),F_1(5)+g_2(2),F_1(0)+g_2(3)\}=23$ $x_2=2$

$F_2(10)=\max\{F_1(10)+g_2(0),F_1(9)+g_2(1),F_1(6)+g_2(2),F_1(1)+g_2(3)\}=24$ $x_2=3$

$k=3$：

$F_3(0)=\max\{F_2(0)+g_3(0)\}=15$ $x_3=0$

$F_3(1)=\max\{F_2(1)+g_3(0),F_2(0)+g_3(1)\}=20$ $x_3=0$

$F_3(2)=\max\{F_2(2)+g_3(0),F_2(1)+g_3(1)\}=24$ $x_3=1$

$F_3(3)=\max\{F_2(3)+g_3(0),F_2(2)+g_3(1)\}=26$ $x_3=1$

$F_3(4)=\max\{F_2(4)+g_3(0),F_2(3)+g_3(1),F_2(0)+g_3(2)\}=26$ $x_3=1$

$F_3(5)=\max\{F_2(5)+g_3(0),F_2(4)+g_3(1),F_2(1)+g_3(2)\}=30$ $x_3=1$

$F_3(6)=\max\{F_2(6)+g_3(0),F_2(5)+g_3(1),F_2(2)+g_3(2)\}=32$ $x_3=1$

$F_3(7)=\max\{F_2(7)+g_3(0),F_2(6)+g_3(1),F_2(3)+g_3(2)\}=32$ $x_3=1$

$F_3(8)=\max\{F_2(8)+g_3(0),F_2(7)+g_3(1),F_2(4)+g_3(2)\}=35$ $x_3=2$

$F_3(9)=\max\{F_2(9)+g_3(0),F_2(8)+g_3(1),F_2(5)+g_3(2),F_2(0)+g_3(3)\}=37$ $x_3=2$

$F_3(10)=\max\{F_2(10)+g_3(0),F_2(9)+g_3(1),F_2(6)+g_3(2),F_2(1)+g_3(3)\}=37$ $x_3=2$

关于中间结果的备忘录如表 3.2 所示.

表 3.2 备忘录

y	$k=1$		$k=2$		$k=3$	
	$F_1(y)$	x_1	$F_2(y)$	x_2	$F_3(y)$	x_3
0	2	0	7	0	15	0
1	4	1	12	1	20	0
2	4	1	14	1	24	1
3	4	1	14	1	26	1
4	7	2	18	2	26	1
5	7	2	20	2	30	1
6	7	2	20	2	32	1
7	7	2	20	2	32	1
8	7	2	23	2	35	2
9	11	3	23	2	37	2
10	11	3	24	3	37	2

从而得到，$F_3(10)=37$，此刻 $x_3=2$，于是

$$x_1^2+x_2^2\leqslant 10-2^2=6$$

再查 $F_2(6)=20$，此刻 $x_2=2$，于是

$$x_1^2\leqslant 6-2^2=2$$

再查 $F_1(2)=4$ 得 $x_1=1$. 问题的解是：在 $x_1=1,x_2=2,x_3=2$ 时得到 $g_1(x_1)+g_2(x_2)+g_3(x_3)$的最大值 37.

3.2 使用动态规划设计技术. 对于 $i=1,2,\cdots,n$，考虑以 x_i 作为最后项的最长递增子序列的长度 $C[i]$. 如果在 x_i 项前面存在 $x_j<x_i$，那么 $C[i]=\max\{C[j]\}+1$；否则 $C[i]=1$. 因此

$$C[i]=\begin{cases}\max\{C[j]\}+1 & \exists j(1\leqslant j<i,x_j<x_i)\\ 1 & \forall j(1\leqslant j<i,x_j>x_i)\end{cases},\ i>1$$

$$C[1]=1$$

在计算 $C[i]$时,用 $k[i]$记录 $C[i]$取得最大值时的 j 的值;如果不存在这样的 j,令 $k[i]=0$. 这个记录用于追踪解. 所求的最长递增子序列的长度是

$$C=\max\{C[i]\mid i=1,2,\cdots,n\}$$

对每个 i,需要检索比 i 小的所有的 j,需要 $O(n)$时间,i 的取值有 n 种,于是算法时间复杂度是 $W(n)=O(n^2)$.

对于给定的实例 $A=<2,8,4,-4,5,9,11>$,具体的计算过程如下:

$C[1]=1$

$C[2]=\max\{C[1]+1\}=2$ $\quad k[2]=1$

$C[3]=\max\{C[1]+1\}=2$ $\quad k[3]=1$

$C[4]=1$ $\quad k[4]=0$

$C[5]=\max\{C[1]+1,C[3]+1,C[4]+1\}=3$ $\quad k[5]=3$

$C[6]=\max\{C[1]+1,C[2]+1,C[3]+1,C[4]+1,C[5]+1\}=4$ $\quad k[6]=5$

$C[7]=\max\{C[1]+1,C[2]+1,C[3]+1,C[4]+1,C[5]+1,C[6]+1\}=5$ $\quad k[7]=6$

在 $C[1],C[2],\cdots,C[7]$中,$C[7]=5$ 是最大值,这意味着 A 的第 7 项 $x_7=11$ 是最长递增子序列的最后项,长度是 5. 子序列的构造从后向前进行,开始是 11,追踪过程是:

$k[7]=6\Rightarrow x_6=9$, $\quad k[6]=5\Rightarrow x_5=5$, $\quad k[5]=3\Rightarrow x_3=4$, $\quad k[3]=1\Rightarrow x_1=2$

于是得到解是:$<2,4,5,9,11>$,长度是 5. 本题可以有多个解.

3.3 类似于 0-1 背包问题,库房的长度相当于背包的重量限制,每个货柜的收益相当于物品的价值. 于是问题是:

$$\max\sum_{i=1}^{n}v_ix_i$$

$$\sum_{i=1}^{n}l_ix_i\leqslant D,\quad x_i=0,1$$

令 $C[k,y]$是只允许装前 k 个货柜,库房长度为 y 时的最大收益,那么有

$$C[k,y]=\begin{cases}C[k-1,y] & y<l_k\\ \max\{C[k-1,y],C[k-1,y-l_k]+v_k\} & D\geqslant y\geqslant l_k\end{cases},\ k>1$$

$$C[1,y]=\begin{cases}v_1 & y\geqslant l_1\\ 0 & y<l_1\end{cases}$$

算法的伪码是:

```
Store
输入:数组 L[1..n],V[1..n],D      //L 和 V 是货柜长度和价值序列,D 为库房长度
输出:最大的收益 C[n,D]
1. for y←1 to D
2.     C[1,y]←V[1]
3. for k←2 to n
4.     for y←1 to D
5.         C[k,y]←C[k-1,y]
6.         i[k,y]←i[k-1,y]
```

```
7.        if y≥L[k] and C[k-1,y-L[k]]+V[k]>C[k-1,y]
8.        then C[k,y]←C[k-1,y-L[k]]+V[k]
9.             i[k,y]←k
```

算法在第1行时间为$O(D)$,第3行和第4行的循环进行$O(nD)$次,循环内部是常数时间的操作,于是算法最坏情况下的时间复杂度是$O(nD)$.

3.4 设任务集$J=\{1,2,\cdots,n\}$,机器为M_1和M_2. 一个调度是函数$f: J\to\{1,2\}$,如果$f(i)=1$,那么任务i将分配到M_1上;如果$f(i)=2$,则分配在M_2上. 因为任务之间的安排没有偏序约束,所以只要f给定,即安排在M_1和M_2上的任务确定,无论M_1和M_2上的任务怎样排序,每台机器的加工时间都是不变的. 两台机器的加工时间分别为

$$D_1(f)=\sum_{f(i)=1}t_i,\quad D_2(f)=\sum_{f(i)=2}t_i$$

调度f的总加工时间是

$$D(f)=\max\{D_1,D_2\}$$

问题的解是求一个使得$D(f)$达到最小值的调度f.

命题 3.1 令$T=\left\lfloor\frac{1}{2}\sum_{i=1}^{n}t_i\right\rfloor$,对于给定输入,一定存在一个最优解$f^*$,使得$D_1(f^*)\leqslant T$且$D_1(f^*)$达到最大.

证 假设一个最优解f满足$D_1(f)>D_2(f)$. 交换M_1和M_2的任务,得到解f^*,那么f^*的总加工时间与f的总加工时间相等,因此f^*也是最优解且$D_1(f^*)\leqslant D_2(f^*)$. 由于

$$D_1(f^*)+D_2(f^*)=\sum_{i=1}^{n}t_i$$

且$D_1(f^*)\leqslant D_2(f^*)$,于是$D_1(f^*)\leqslant\frac{1}{2}\sum_{i=1}^{n}t_i$. 由于所有任务的加工时间$t_i$都是正整数,因此$D_1(f^*)\leqslant T$.

f^*的总加工时间$D(f^*)=D_2(f^*)$. 作为最优调度,$D_2(f^*)$一定达到最小,而$D_1(f^*)+D_2(f^*)$对所有的调度都是一样的,当$D_2(f^*)$达到最小时,$D_1(f^*)$必然达到最大.

根据命题3.1,可以设计一个动态规划算法找一个解f^*,使得$D_1(f^*)\leqslant T$且$D_1(f^*)$达到最大. 令$x_i=1$当且仅当$f(i)=1$,原问题变成如下组合优化问题:

$$\max\sum_{i=1}^{n}x_it_i$$

$$x_i=0,1,\quad i=1,2,\cdots,n$$

$$\sum_{i=1}^{n}t_ix_i\leqslant T,\quad T=\left\lfloor\frac{1}{2}\sum_{i=1}^{n}t_i\right\rfloor$$

这是一个0-1背包问题,令$F[k,y]$表示考虑前k个任务,在M_1的加工时间不超过y的情况下其加工时间$\sum_{i=1}^{k}t_ix_i$的最大值. 那么有如下公式:

$$F[k,y]=\max\{F[k-1,y],F[k-1,y-t_k]+t_k\},\quad k>1,\quad T\geqslant y>0$$

$$F[1,y]=\begin{cases}t_1 & \text{如果 } y\geqslant t_1\\ 0 & \text{否则}\end{cases}$$

$$F[k,0]=0$$

$$F[k,y]=-\infty \quad \text{如果 } y<0$$

在 $F[k,y]$计算时,设立标记函数 $i[k,y]$

$$i[k,y]=\begin{cases} i[k-1,y] & \text{如果 } F[k-1,y-t_k]+t_k<F[k-1,y] \\ k & \text{否则} \end{cases},\ k>1,\ T\geqslant y>0$$

$$i[1,y]=\begin{cases} 1 & y\geqslant t_1 \\ 0 & \text{否则} \end{cases}$$

最坏情况下的时间复杂度为 $O(nT)$,其中 $T=\left\lfloor \frac{1}{2}\sum_{i=1}^{n}t_i \right\rfloor$.

具体实例的计算过程:

$$T=\left\lfloor \frac{1}{2}(1+5+2+10+3) \right\rfloor=\lfloor 10.5 \rfloor=10$$

备忘录和标记函数如表 3.3 和表 3.4 所示.

表 3.3 $F[k,y]$

k \ y	1	2	3	4	5	6	7	8	9	10
1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	1	5	6	6	6	6	6
3	1	2	3	3	5	6	7	8	8	8
4	1	2	3	3	5	6	7	8	8	10
5	1	2	3	3	5	6	7	8	9	10

表 3.4 $i[k,y]$

k \ y	1	2	3	4	5	6	7	8	9	10
1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	1	2	2	2	2	2	2
3	1	3	3	3	2	2	3	3	3	3
4	1	3	3	3	2	2	3	3	3	4
5	1	3	3	3	5	5	3	5	5	5

由 $F[5,10]=10$ 可知,M_1 的加工时间 $D_1(f)=10$,于是 f 的总加工时间

$$D_2(f)=21-D_1(f)=21-10=11$$

由 $i[5,10]=5$ 可知 $x_5=1$,接着查

$$i[4,10-t_5]=i[4,7]=3$$

可知 $x_3=1$,接着查

$$i[2,7-t_3]=i[2,5]=2$$

可知 $x_2=1$,接着查

$$i[1,5-t_2]=i[1,0]=0$$

追踪完成,于是调度是

$$f(2)=f(3)=f(5)=1,\quad f(1)=f(4)=2,\quad D_2=11$$

如果 $i[k,y]$的定义不同,可能得到另外一个解:

$$f(4)=1,\quad f(1)=f(2)=f(3)=f(5)=2,\quad D_2=11$$

3.5 设 x_i 表示第 i 种硬币使用的个数,$i=1,2,\cdots,n$. 该问题的描述为

$$\min\sum_{i=1}^{n}w_ix_i$$

$$\sum_{i=1}^{n}v_ix_i=y\quad x_i\text{ 为非负整数}$$

令 $F_k(x)$表示只允许使用前 k 种钱,总付款为 x 时所使用零钱的最轻重量,则

$$F_k(x)=\min\{F_{k-1}(x),F_k(x-v_k)+w_k\},\quad k>1,\ 0<x\leqslant y$$

$$F_1(x)=w_1\left\lfloor\frac{x}{v_1}\right\rfloor=w_1x$$

$$F_k(0)=0$$

$$F_k(x)=+\infty,\quad x<0$$

设立标记函数 $t_k(y)$记录 $F_k(y)$取得最小值时最大币值的标号是否为 k.

$$t_k(x)=\begin{cases}k & F_k(x-v_k)+w_k\leqslant F_{k-1}(x)\\ t_{k-1}(x) & \text{否则}\end{cases},\quad k>1,\ 0<x\leqslant y$$

$$t_1(x)=1,\quad 0<x\leqslant y$$

$$t_k(0)=0,\quad k=1,2,\cdots,n$$

算法的伪码是:

```
Coin
输入:w[1..n],v[1..n],y      //w,v 分别为硬币的重量和币值数组,y 是付款数
输出:F[i,j],t[i,j] i=1,2,…,n,j=1,2,…,y
1. for j←1 to y do
2.     F[1,j]←j * w[1]
3.     t[1,j]←1
4. for i←2 to n do
5.     for  j←1 to y do
6.         F[i,j]←F[i−1,j]
7.         t[i,j]←t[i−1,j]
8.         if F[i,j−v[i]]+w[i]≤F[i−1,j]
9.         then F[i,j]←F[i,j−v[i]]+w[i]
10.                t[i,j]←i
11. return 二维数组 F,t
```

算法的时间复杂度主要取决于第 4 行和第 5 行的 for 循环,内部工作量是常数时间,算法的时间是 $O(ny)$. 通过数组 t 追踪解的过程比较简单,时间不超过 $O(ny)$. 于是算法最坏情况下的时间复杂度是 $O(ny)$.

针对给定实例,算法的备忘录如表3.5和表3.6所示.

表3.5 $F_k(x)$

k \ x	1	2	3	4	5	6	7	8	9	10	11	12
1	1	2	3	4	5	6	7	8	9	10	11	12
2	1	2	3	2	3	4	5	4	5	6	7	6
3	1	2	3	2	3	4	5	4	5	6	7	6
4	1	2	3	2	3	4	5	4	5	6	7	6

表3.6 $t_k(x)$

k \ x	1	2	3	4	5	6	7	8	9	10	11	12
1	1	1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	2	2	2	2	2	2	2	2	2
3	1	1	1	2	2	3	3	2	2	3	3	2
4	1	1	1	2	2	3	3	2	2	3	3	2

问题实例的解是:最轻重量是6,由$t_4(12)=2$知道$x_4=0,x_3=0,x_2\geqslant 1$,再由

$$t_2(12-4)=t_2(8)=2\Rightarrow x_2\geqslant 2$$

$$t_2(8-4)=t_2(4)=2\Rightarrow x_2\geqslant 3$$

$$t_2(4-4)=t_2(0)=0\Rightarrow x_2=3,\quad x_1=0$$

从而得到$x_1=x_3=x_4=0,x_2=3$,只用了3枚币值为4的硬币.

3.6 使用动态规划算法.令$F_k(y)=0,1$,

$F_k(y)=1\Leftrightarrow$用前k种钱币、总钱数为y时可以付款

那么$F_k(y)$满足如下递推方程:

$$F_k(y)=\max\{F_{k-1}(y),F_{k-1}(y-v_k)\},\quad k=2,3,\cdots,n,\ y=1,2,\cdots,M$$

$$F_1(y)=\begin{cases}1 & \text{若 } y=v_1\\ 0 & \text{否则}\end{cases}$$

$$F_k(y)=0\quad \text{若 } y<0,\ k=2,3,\cdots,M$$

设立标记函数$i_k(y)$

$$i_k(y)=\begin{cases}k & \text{若 } F_{k-1}(y-v_k)=1\\ i_{k-1}(y) & \text{否则}\end{cases}$$

算法计算出$F_n(M)$.如果$F_n(M)=1$,则存在一种找零钱的方法.所使用的零钱币值可根据$i_k(n)$的值向前追踪得到.算法的时间复杂度为$T(n)=O(nM)$.

3.7 (1)设$y_i=1$表示选择了位置i,$y_i=0$表示没有选位置i,问题的目标函数和约束条件是:

$$\max\sum_{i=1}^{n}p_iy_i$$

$$|x_i-x_j|\geqslant d,\quad i,j=1,2,\cdots,n,\ i\neq j,\ y_i=y_j=1$$

(2) 排序使得 $x_1<x_2<\cdots<x_n$，$F_k(y)$表示考虑前 k 个位置，距原点最远到 y 的范围内开设商店的最大收益. 那么 $F_k(y)$满足如下递推方程：

$$F_k(y)=\begin{cases}\max\{F_{k-1}(y),F_{k-1}(x_k-d)+p_k\} & \text{当 } x_k\leqslant y\\ F_{k-1}(y) & \text{否则}\end{cases}$$

$$F_1(y)=\begin{cases}p_1 & \text{当 } x_1\leqslant y\\ 0 & \text{否则}\end{cases}$$

如下定义标记函数：

$$i_k(y)=\begin{cases}k & \text{当 } F_{k-1}(x_k-d)+p_k\geqslant F_{k-1}(y)\\ i_{k-1}(y) & \text{否则}\end{cases}$$

$$i_1(y)=\begin{cases}1 & \text{当 } x_1\leqslant y\\ 0 & \text{否则}\end{cases}$$

算法的时间复杂度为 $O(nD+n\log n)$，其中 $D=\max\{x_i\mid i=1,2,\cdots,n\}$.

3.8 令 $B=\lfloor N/2\rfloor$，那么 $\min\{\sum\limits_{a_i\in A_1}a_i,\sum\limits_{a_j\in A_2}a_j\}\leqslant B$. 不妨设 $\sum\limits_{a_i\in A_1}a_i\leqslant\sum\limits_{a_j\in A_2}a_j$. 那么问题变成求 A 的子集 A_1 使得

$$\max\sum_{a_i\in A_1}a_i$$

$$\sum_{a_i\in A_1}a_i\leqslant B$$

那么该问题等价于双机调度问题(习题 3.4). 通过动态规划算法求出最优解 A_1，如果 A_1 中的数之和 $\sum\limits_{a_i\in A_1}a_i=B$，那么输出“Yes”，否则输出“No”. 该算法的时间复杂度为 $O(nN)$.

3.9 与 0-1 背包问题类似，使用动态规划方法. 令 $N_j(d)$表示考虑作业集$\{1,2,\cdots,j\}$、结束时间为 d 的最优调度的效益，那么

$$N_j(d)=\begin{cases}\max\{N_{j-1}(d),N_{j-1}(d-t(j))+v_j\} & d\geqslant t(j)\\ N_{j-1}(d) & d<t(j)\end{cases}$$

$$N_1(d)=\begin{cases}v_1 & t(1)\leqslant d\\ 0 & t(1)>d\end{cases}$$

$$N_j(0)=0$$

$$N_j(d)=-\infty \quad d<0$$

自底向上计算，存储使用备忘录方法. 可以使用标记函数 $B(j)$记录使得 $N_j(d)$达到最大时是否

$$N_{j-1}(d-t(j))+v_j>N_{j-1}(d)$$

如果是，则 $B(j)=j$；否则 $B(j)=B(j-1)$.

算法的伪码是

输入：加工时间 $t[1..n]$，效益 $v[1..n]$，结束时间 D

输出：最优效益 $N[i,j]$，标记函数 $B[i,j]$，$i=1,2,\cdots n$，$j=1,2,\cdots,D$

```
1.  for d←1 to t[1]−1
2.      N[1,d]←0,B[1]←0
3.  for d←t[1] to D
```

```
4.      N[1,d]←v[1],B[1]←1
5.  for j←2 to n
6.      for d←1 to D
7.          N[j,d]←N[j-1,d]
8.          B[j,d]←B[j-1,d]
9.          if d≥t[j] and N[j-1,d-t[j]]+v[j]>N[j-1,d]
10.         then   N[j,d]←N[j-1,d-t[j]]+v[j]
11.                B[j,d]←j
```

得到最大效益 $N[n,D]$后,通过对 $B[n,D]$的追踪,可以得到问题的解. 算法的主要工作是第5行和第6行的for循环,需要执行 $O(nD)$次,循环体内的工作量是常数,追踪解的工作量不大,于是算法最坏情况下的时间复杂度是 $O(nD)$.

3.10 设 x_i 表示装入背包的第 i 种物品个数,$i=1,2,\cdots,n$,推广的0-1背包问题的描述是

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

$$\sum_{i=1}^{n} w_i x_i \leqslant W$$

$$\sum_{i=1}^{n} c_i x_i \leqslant V$$

$$x_i = 0,1$$

设 $m[i,j,k]$表示使用前 i 种物品、背包重量限制为 j、容积为 k 时的最大价值,其中 $i=1,2,\cdots,n$,$j=1,2,\cdots,W$,$k=1,2,\cdots,V$,那么递推方程是:

$$m[i,j,k]=\begin{cases}\max\{m[i-1,j,k],m[i-1,j-w_i,k-c_i]+v_i\} & \text{若 } j\geqslant w_i \text{ 且 } k\geqslant c_i\\ m[i-1,j,k] & \text{否则}\end{cases}$$

$$i=2,\cdots,n,\quad j=1,2,\cdots,W,\quad k=1,2,\cdots,V$$

$$m[1,j,k]=\begin{cases}v_1 & \text{如果 } j\geqslant w_1 \text{ 且 } k\geqslant c_1\\ 0 & \text{否则}\end{cases}$$

$$m[i,0,k]=m[i,j,0]=0$$

$$m[i,j,k]=-\infty \quad j<0 \text{ 或 } k<0$$

与前面的背包问题类似,可定义标记函数 $t[i,j,k]$用于追踪问题的解. 其中

$$t[i,j,k]=\begin{cases}i & \text{如果 } m[i-1,j,k]\leqslant m[i-1,j-w_i,k-c_i]+v_i, j\geqslant w_i, k\geqslant c_i\\ t[i-1,j,k] & \text{否则}\end{cases}$$

$$i=2,\cdots,n,\quad j=1,2,\cdots,W,\quad k=1,2,\cdots,V$$

$$t[1,j,k]=\begin{cases}1 & \text{如果 } j\geqslant w_1 \text{ 且 } k\geqslant c_1\\ 0 & \text{否则}\end{cases}$$

该算法最坏情况下的时间复杂度是 $O(nWV)$.

3.11 设 $X=\{x_0,x_1,\cdots,x_{n-1}\}$顺时针排列在圆环上. 类似于矩阵链乘法的动态规划算法,用 i 和 j 界定子问题的边界. X_{ij} 表示按顺时针合并$\{x_i,\cdots,x_j\}$后的数组,其含有的整数个数是 n_{ij},完成合并在最坏情况下所需的最少比较次数记作 $m[i,j]$,其中 $i=0,1,\cdots,n-1$,$j=0,1,\cdots,n-1$.

考虑数组 X_{ik} 和 $X_{(k+1)j}$ 合并成 X_{ij} 的比较次数. 因为它们都是排好序的,比较次数至多是元素总数减 1,于是合并这两个数组的比较次数在最坏情况下是

$$n_{ik}+n_{(k+1)j}-1$$

与矩阵链乘法的不同在于:矩阵链的子问题的边界是排列成一条线,保持 $i\leqslant j$;而这里是环,可能出现 $i>j$ 的情况. 如图 3.3 所示,左边的合并对应的是 $i<j$ 的子问题,右边的合并对应的是 $i>j$ 的子问题. 在 $i>j$ 时相应的递推方程就不一样了.

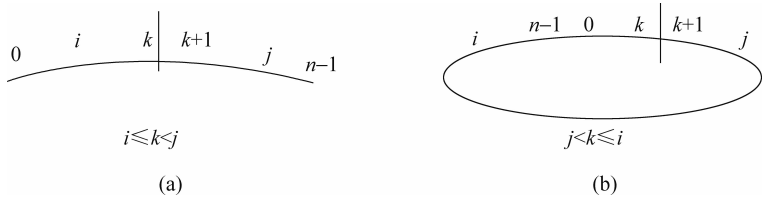

图 3.3　子问题的归约

令 $m[i,j]$ 表示在最坏情况下合并成 X_{ij} 所需要的最少的比较次数. 考虑在 $i<j$ 的情况,X_{ik} 与 $X_{(k+1)j}$ 合并的比较次数至多是 $n_{ik}+n_{(k+1)j}-1$,于是有

$$m[i,j]=\min_{i\leqslant k<j}\{m[i,k]+m[k+1,j]\}+\sum_{t=i}^{j}n_t-1$$

当 $i>j$ 时,如图 3.3(b)所示,当 k 取 $n-1$ 时,$k+1=0$,这时候优化函数的和应该是

$$m[i,n-1]+m[0,j]=m[i,k]+m[(k+1)\bmod n,j]$$

而为计算合并的比较次数,需要分成两段求和,即

$$\sum_{t=i}^{n-1}n_t+\sum_{t=0}^{j}n_t-1$$

于是有

$$m[i,j]=\min_{i\leqslant k<j}\{m[i,k]+m[k+1,j]\}+\sum_{t=i}^{j}n_t-1,\quad i<j$$

$$m[i,j]=\min_{\substack{i\leqslant k\leqslant n-1\\0\leqslant k<j}}\{m[i,k]+m[(k+1)\bmod n,j]\}+\sum_{t=i}^{n-1}n_t+\sum_{t=0}^{j}n_t-1,\quad i>j$$

$$m[i,i]=0,\quad i=0,1,\cdots,n-1$$

上述公式中的 $m[i,i]$是递推关系的初值. 算法将从规模为 1 的子问题开始迭代计算,直到规模为 n 的问题为止. 在矩阵链相乘问题中,规模为 n 的子问题只有一个;而在环形排列的合并问题中,规模为 n 的子问题可能以 $i=0,1,\cdots,n-1$ 中的任何位置为起点,恰好有 n 个. 我们需要从这 n 个子问题的解中找到具有最小比较次数的解. 于是最少的比较次数是

$$m=\min_{0\leqslant i\leqslant n-1}\{m[i,(i+n-1)\bmod n]\}$$

与矩阵链乘法相似,计算 $m[i,j]$过程中需要用标记函数 $s[i,j]$记录使得 $m[i,j]$取得最小值的 k. 以便找到最优的合并次序. 与矩阵链链乘法问题类似,该算法最坏情况下的时间复杂度是 $O(n^3)$.

3.12　如图 3.4 所示,n 边形的顶点是 $1,2,\cdots,n$. 顶点 $i-1,i,\cdots,j$ 构成的凸多边形记作 $A[i,j]$,于是原始问题就是 $A[2,n]$.

考虑子问题$A[i,j]$的划分,假设它的所有划分方案中的最小权值是$t[i,j]$. 从$i,i+1,\cdots,j-1$中任选顶点k,它与底边$(i-1)j$构成一个三角形(图3.4中的灰色三角形). 这个三角形将$A[i,j]$划分成两个凸多边形:$A[i,k]$和$A[k+1,j]$,从而产生了两个子问题. 这两个凸多边形的划分方案的最小权值分别是$t[i,k]$和$t[k+1,j]$. 根据动态规划思想,$A[i,j]$相对于这个k的划分方案的最小权值是

$$t[i,k]+t[k+1,j]+d_{(i-1)k}+d_{kj}+d_{(i-1)j}$$

其中$d_{(i-1)k}+d_{kj}+d_{(i-1)j}$是三角形$(i-1)kj$的周长,于是得到递推关系:

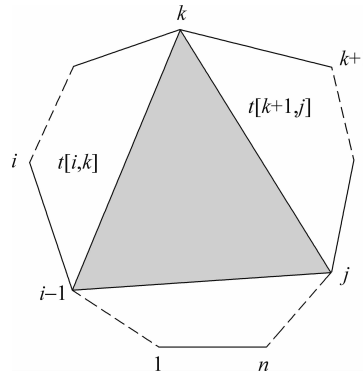

图3.4 子问题归约

$$t[i,j]=\begin{cases}0 & i=j\\ \min\limits_{i\leqslant k<j}\{t[i,k]+t[k+1,j]+d_{(i-1)k}+d_{kj}+d_{(i-1)j}\} & i<j\end{cases}$$

$$t[i,i]=0$$

不难看出,这个递推关系与矩阵链相乘算法的递推式非常相似,可以定义标记函数记下得到最小权值的k的位置,算法最坏情况下的时间复杂度是$O(n^3)$.

3.13 子问题分别对应于$M_0,M_1,\cdots,M_n$的计算. M_0是输入. 在第k步计算M_{k+1}的关键是找到它与前面矩阵的依赖关系. 假设M_k已经计算完毕,如何计算M_{k+1}呢?这需要对于每组i,j,确定$M_{k+1}[i,j]$是否为1. 条件是:

$M_{k+1}[i,j]=1\Leftrightarrow$在$D$中存在一条从$x_i$到$x_j$且只经过$\{x_1,x_2,\cdots,x_k,x_{k+1}\}$中顶点的路径. 可以将这种路径分成两类:

第一类是只经过$\{x_1,x_2,\cdots,x_k\}$中顶点的路径,这时$M_k[i,j]=1$.

第二类是经过顶点x_{k+1}的路径. 因为回路可以从路径中删除,因此只需考虑经过x_{k+1}一次的路径. 这条路径可以分成两段,从x_i到x_{k+1},再从x_{k+1}到x_j,因此有$M_k[i,k+1]=1$和$M_k[k+1,j]=1$. 这就是对于第二类路径的判别条件,即:

$$M_{k+1}[i,j]=1\Leftrightarrow M_k[i,k+1]=1\ \wedge\ M_k[k+1,j]=1$$

算法 Warshall

输入:M //图D的邻接矩阵

输出:M_t //图D的连通矩阵

```
1.  Mt←M
2.  for k←1 to n do
3.      for i←1 to n do
4.          for j←1 to n do
5.              Mt[i,j]←Mt[i,j] + Mt[i,k] * Mt[k,j]
```

注意,上述算法中矩阵加法和乘法$*$中的元素相加都使用逻辑加,即$1+0=0+1=1+1=1,0+0=0$.

下面分析算法的时间复杂度,在Warshall算法中,第2行、第3行、第4行都是n步的循环,因此第5行总共被执行n^3次,而每次只做1次乘法和1次加法,因此Warshall算法的

时间复杂度是 $O(n^3)$.

对于给定实例,利用 Warshall 算法计算的矩阵序列如下所示:

$$M_0=\begin{bmatrix}0&1&1&0\\0&0&1&0\\0&0&0&1\\0&0&1&0\end{bmatrix},\quad M_1=\begin{bmatrix}0&1&1&0\\0&0&1&0\\0&0&0&1\\0&0&1&0\end{bmatrix},\quad M_2=\begin{bmatrix}0&1&1&0\\0&0&1&0\\0&0&0&1\\0&0&1&0\end{bmatrix}$$

$$M_3=\begin{bmatrix}0&1&1&1\\0&0&1&1\\0&0&0&1\\0&0&1&1\end{bmatrix},\quad M_4=\begin{bmatrix}0&1&1&1\\0&0&1&1\\0&0&1&1\\0&0&1&1\end{bmatrix}$$

因此,a 可以连通到 b、c 和 d;b 可以连通到 c 和 d;c 可以连通到 d; d 可以连通到 c.

3.14 使用动态规划算法求解这个问题. 先考虑子问题划分. 设 $S_1[1..i]$和 $S_2[1..j]$表示两个子序列,$C[i,j]$表示 $S_1[1..i]$和 $S_2[1..j]$的最小编辑距离. 考虑最后一对字符,如果 $S_1[i]$被删除,那么最小编辑距离是 $S_1[i-1]$和 $S_2[j]$的最小编辑距离加 d;如果在 $S_1[i]$后面插入 $S_2[j]$,那么编辑距离是 $S_1[i]$和 $S_2[j-1]$的最小编辑距离加 d;如果 $S_1[i]$被替换成 $S_2[j]$,那么最小编辑距离是 $S_1[i-1]$与 $S_2[j-1]$的最小编辑距离加 r;如果 $S_1[i]=S_2[j]$,那么最小编辑距离是 $S_1[i-1]$与 $S_2[j-1]$的最小编辑距离. 递推关系如下:

$$C[i,j]=\min\{C[i-1,j]+d,C[i,j-1]+d,C[i-1,j-1]+t[i,j]\}$$

$$i=1,2,\cdots,n,\quad j=1,2,\cdots,m$$

$$t[i,j]=\begin{cases}0, & S_1[i]=S_2[j]\\ r, & S_1[i]\neq S_2[j]\end{cases}$$

$$C[0,j]=jd$$

$$C[i,0]=id$$

可以定义标记函数记录得到最优 $C[i,j]$时所对应的选择(删除、插入、替换等),算法的时间复杂度是 $O(nm)$.

3.15 **方法一**:设 $F[j]$表示第 $1,2,\cdots,j$ 天加工任务的最大数目. 如果在第 i 天进行检修,那么从第 $i+1$ 天到第 j 天连续工作的加工量用 $w[i+1,j]$表示,则

$$w[i+1,j]=\sum_{k=i+1}^{j}\min\{x_k,s_{k-i}\},\quad 0\leqslant i<j\leqslant n$$

不难看出,$w[i+1,j]$满足如下递推方程:

$$w[i+1,j]=w[i+1,j-1]+\min\{x_j,s_{j-i}\},\quad 0\leqslant i<j\leqslant n,\ j>1$$

$$w[i+1,i+1]=\min\{x_{i+1},s_1\},\quad i=0,1,\cdots,n$$

假设最后一次检修机器在第 i 天,那么从第 $i+1$ 天到第 j 天连续加工任务总数为 $w[i+1,j]$,而检修前加工任务数至多是 $F[i-1]$. 于是当最后一次检修发生在第 i 天时,到第 j 天为止,加工任务数至多是 $F[i-1]+w[i+1,j]$. 考虑到所有可能的检修时间 $i(i=1,2,\cdots,j-1)$,或者在第 j 天之前根本不做检修的情况,可以得到如下递推关系:

$$F[j]=\max\begin{cases}\max\limits_{0<i<j}\{F[i-1]+w[i+1,j]\} & j>1\\ w[1,j] & j>0\end{cases}$$

$$F[0]=0$$

可以定义标记函数记录得到最优 $F[j]$时的检修时间 i. 如果每次都重新计算 $w[i+1,j]$,算法最坏情况下的时间复杂度是 $O(n^3)$. 如果在预处理时根据 $w[i.j]$的递推方程计算所有的 $w[i,j]$,$1\leqslant i\leqslant j\leqslant n$,并把计算结果存成备忘录,在计算 $F[j]$时直接查找相应的值,那么计算 $F[j]$的时间可以降为 $O(n^2)$,而预处理的时间也是 $O(n^2)$,因此算法最坏情况下的时间复杂度是 $O(n^2)$.

方法二:参照矩阵链相乘的方式划分子问题,该算法的时间复杂度高一些.

首先证明下述命题.

命题 3.2 在最优解中不会出现连续两天检修的情况.

证 假若不然,在一个最优解 T 的第 i 和 $i+1$ 天都进行检修. 那么,当去掉第 i 天的检修后,不会减少从第 $i+1$ 天到第 n 天的加工量;也不会减少第 1 天到第 $i-1$ 天的加工量;只有第 i 天的加工量由 0 增加到 $\min\{x_i,s_t\}$,其中 s_t 表示第 i 天机器具有的加工能力. 于是总加工量将大于原来的加工量,从而与 T 的最优性矛盾.

通过类似的分析还可以证明,检修也不会发生在子问题的最后一天.

利用上述性质,可以从检修的时间点 k 将原问题划分成两个子问题. 设 $G[i,j]$表示第 i 天到第 j 天的最大加工任务数. 如果在第 k 天进行检修,$i<k<j$,那么最大加工任务数等于 $G[i,k-1]+G[k+1,j]$. 如果在第 i 天到第 j 天的时间内没有发生检修,那么加工任务数是 $w[i,j]$,于是得到如下递推方程:

$$G[i,j]=\max\{w[i,j],\max_{i<k<j}\{G[i,k-1]+G[k+1,j]\},\quad 1\leqslant i<k<j\leqslant n$$

$$G[i,i]=w[i,i],\quad i=1,2,\cdots,n$$

标记函数的设定可参照矩阵链相乘问题的做法,只是当最优解的第 i 到第 j 天之间没有进行检修时将标记函数设为 0. 该算法最坏情况下的时间复杂度是 $O(n^3)$.

3.16 设 $x_1,x_2,\cdots,x_n$ 是 0 或 1,$x_i=1$ 当且仅当下载请求 i 被批准. 那么

$$\max\sum_{i=1}^{n}a_ix_i$$

$$\sum_{i=1}^{n}a_ix_i\leqslant K,\quad x_i=0,1$$

类似于 0-1 背包问题,令 $F_i(y)$表示考虑前 i 个申请,带宽限制为 y 时的最大带宽使用量,则有如下递推式:

$$F_i(y)=\max\{F_{i-1}(y),F_{i-1}(y-a_i)+a_i\},\quad i>1,\ 0<y\leqslant K$$

$$F_1(y)=\begin{cases}a_1, & y\geqslant a_1\\ 0, & y<a_1\end{cases}$$

$$F_i(0)=0$$

$$F_i(y)=-\infty,\quad y<0$$

与前面背包问题类似可设立标记函数,自底向上顺序处理 $i=1,2,\cdots,n$ 的情况. 对于给定的 i,令 $y=1,2,\cdots,K$,依次确定各个子问题的 $F_i(y)$值,最后由标记函数得到 x_i 的值. 算法时间复杂度为 $O(nK)$.

3.17 令 $F[i,j]$表示 a_{11} 到 a_{ij} 的路径上的数的最小和,则

$$F[i,j]=\min\{F[i-1,j],F[i-1,j-1]\}+a_{ij},\quad i=2,3,\cdots,n,\ j=2,3,\cdots,i-1$$

$$F[i,1]=F[i-1,1]+a_{i1},\quad i=2,3,\cdots,n$$
$$F[i,i]=F[i-1,i-1]+a_{ii},\quad i=2,3,\cdots,n$$
$$F[1,1]=a_{11}$$

问题的最优路径上的和是

$$\min\{F[n,j]\mid j=1,2,\cdots,n\}$$

可以定义标记函数 $k[i,j]$,记录得到最优 $F[i,j]$时的路径选择,即

$$k[i,j]=\begin{cases}j & \text{若 } F[i-1,j]<F[i-1,j-1]\\ j-1 & \text{否则}\end{cases}$$

算法的时间复杂度为 $O(n^2)$.

3.18 本题不是优化问题. 也可以使用动态规划的设计思想,通过多阶段决策来求解. 与一般优化问题的区别在于:当把一个问题的计算与子问题的计算建立递推关系的时候,不是只考虑达到最优的那个子问题的结果,而是考虑所有可能的子问题的结果. 考虑到每个串计算的结果至多是 3 种,即 a、b 或 c,为了后面的计算,需要把它们全部保留下来. 因此,可以把记录优化函数的一个极大或极小值推广到记录有限个值. 只要用含有 3 个项的数组替换备忘录中的一个项,就可以使用类似于矩阵链相乘的方法划分子问题,并利用动态规划方法设计算法.

令 $X[i,j]$是串 $x_i\cdots x_j$ 的可能的运算结果(由于计算顺序不同,结果可能不唯一,但至多为 3 个结果,即 a,b,c). 设计存储 $X[i,j]$的结构为 3×2 阵列,分别记录 3 个可能的运算结果(可能取 a、b 或 c)和得到这个结果时所对应的划分位置 k. 自底向上计算,通过较小子问题的结果来计算较大子问题的结果,其递推公式是:

$$X[i,j]=X[i,k]X[k+1.j],\quad 1\leqslant i<j\leqslant n,\ k=i,i+1,\cdots,j-1$$
$$X[i,i]=x_i,\quad i=1,2,\cdots,n$$

伪码如下:

```
1.  for i←1 to n
2.      X[i,i]←x_i
3.  for r←2 to n−1                          //计算所有大小为 r 的子问题
4.      for i←1   to n−r+1                  //子问题的前边界 i
5.          j←i+r−1                         //子问题的后边界 j
6.          for k←i to j−1                  //在 k 位置划分成两个更小的子问题
7.              X[i,j]←X[i,k]X[k+1,j]
                                    //记录 X[i,j]可能的结果和划分位置 k,至多为 3 个
8.  for k←1 to n−1                          //规模为 n 的原始问题
9.      if X[1,k]X[k+1,n]="a" then return "1"   //存在计算顺序结果为 a
10. return "0"                              //没有结果等于 a
```

与矩阵链计算类似,可通过所记录的 k 值追踪出相应的运算顺序. 该算法最坏情况下的时间复杂度为 $O(n^3)$.