

1. 第一层for 循环n，内层循环n,n-1,…1,所以T(n)=1=2+…+n=n(n-1)/2=O(n2)
2. 分治算法：和最邻近点对的算法类似，我们可以在k=n/2的位置将A=<a1,a2,…an>划分为A1和A2两半，于是A的最大子段和可能是三种情况：出现在A1部分、出现在A2部分、出现在横跨两边的中间部分。前两种情况恰好对应两个规模减半的子问题，第三种情况,设最大子段为[p..q]，则一定p≤k,q≥k+1，只需从A[k]和A[k+1]分别向前和向后求和，记下其最大的和S1,S2，则S1+S2就是横跨中心的最大和。算法如下：

‘

MaxSubSum(A,left,right){

输入：数组A,left、right分别是A的左右边界；

输出：A的最长子段和sum及子段的前后边界。

if left=right then return max(A[left],0)

k:=(left+right)/2

leftsum:=MaxSubSum(A,left,k)

Rightsum:=MaxSubSum(A,k+1,right)

S1:=A1[k] //从k向左的最大和,参考1)内循环改为for j=k to 1

S2:=A2[k+1] //从k向右的最大和,求s1,s2需扫描整个数组O(n)

//（边界已经定了，所以只需要内循环，不需要外循环）

Sum:=s1+s2

If leftsum>sum then sum:=leftsum

If rightsum>sum then sum:=rightsum

}

算法复杂度：T(n)=2T(n/2)+O(n)=O(nlogn)(主定理之情形2)

3)这个问题如何确定子问题是个有意思的事情，需要认真考虑。如我们可能很自然设C[i]=A[1..i]的最大和，但它不满足最优子结构。因为使A[1..i]达到最大和的子段未必包含A[i]，计算后面的问题不能直接把该子段直接组合进来。如A=<2,-5,8, 1,-9,4,6>，C[5]=8+1=9，在计算C[6]时不能把<8,1>直接组合进来，因为后面有-9，对连续和也有影响。例中，C[7]=4+6，C[6]显然不是4，而是8+1。所以这样定义的C不满足最优子结构。

（与上述解法不同的是，开始位置为不确定的值，而不是1） 按提示，定义b(j)= 为输入A[1..j]中必须包含A[j]的最大子段和，b[j]满足最优子结构性质：若b[j+1]最大，则b[j]必然最大，从b[j+1]= 得b[j+1]=b[j]+A[j+1]即可很易证明。

据此可得递推公式：b[j]=max{b[j-1]+A[j],A[j]},j=1,2,..n；b[0]=0.

这里还有一个问题：b[j]并不是A[1..j]的最大子段和，只是包含A[j]的最大子段和。但我们已得到b[1],b[2],…b[n]，恰好枚举了以任何元素为最后元素的的所有子段的最大和，A[1..n]的最大子段和一定在其中：对n个和比较取最大者即可。

算法描述：

MaxSum(A,n){

输入：数组A；输出：组大字段和sum，子段的最后位置c

Sum:=0

b:=0

for i:=1 to n do

if b>0 then b:=b+A[i]//前几项和为整数（不考虑当前值的正负是因为，如果是复数的话，后面跟最大sum比较的时候也被排除了，所以无需去做这个事情）

else b:=A[i] //前几项和为负数,此时把和赋为当前值

endif

if b>sum then

sum:=b

c:=i

endif

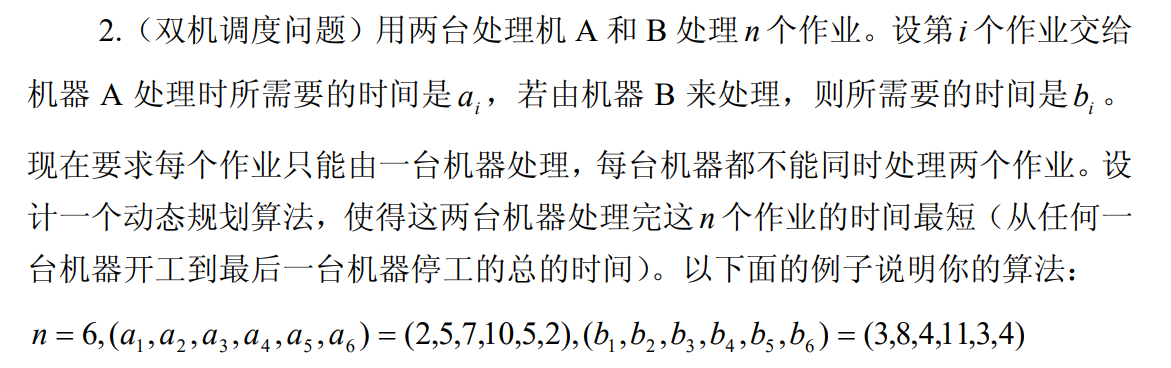
end {for}

return sum,c

}

尚有一点工作：从sum，c找到子段的左边界(略)。

算法复杂度：显然T(n)=O(n)。



1）算法思想：

在完成前k个作业时，设机器A工作了x 时间，则机器B此时最小的工作时间是x的一个函数。

设F[k][x]表示完成前k个作业、机器A工作了x 时间时，**机器B最小的工作时间**，

则F[k][x]=min{F[k-1][x]+bk , F[k-1][x-ak]} （前边，第k个工作b来做）（后边a来做）

其中F[k-1][x]+bk对应第 k个作业由机器 B 来处理（完成 k-1 个作业时机器A的工作时间仍是 x，则 B在 k-1 阶段用时F[k-1][x]；而F[k-1][x-ak]对应第 k个作业由机器A处理（完成k-1 个作业，机器A 工作时间是x-a[k]，而B完成k阶段与完成k-1 阶段用时相同为F[k-1][x-ak]。

则完成前 k 个作业所需的时间为：max{x,F[k][x]}。

根据递推关系，很容易证明问题满足最优子结构性质。

解为：min(max(x,f(N,x)),0<=x<=Σa(i)。

2）当处理第一个作业时，a[1]=2,b[1]=3;

机器A 所花费时间的所有可能值范围：0≤x ≤a[0].

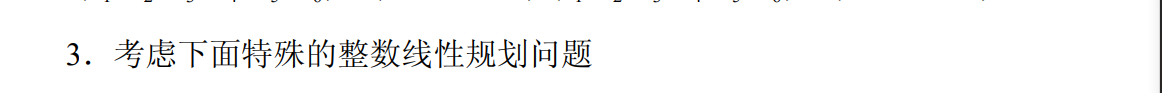
x<0 时，设F[0][x]= ∞，则max(x, ∞)= ∞；

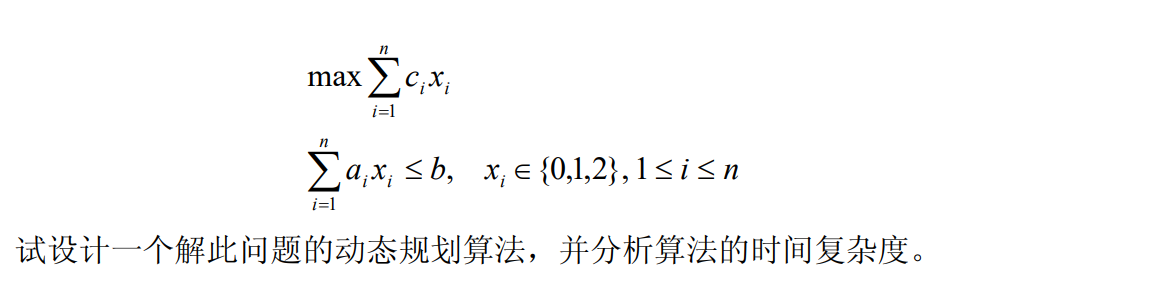
0≤x<2 时，F[1][x]=3，则Max(0,3)=3，

x≥2 时， F[1][x]= 0，则Max(2,0)=2；

处理第二个作业时：x 的取值范围是：0 <= x <= (a[0] + a[1])，

当x<0 时，记F[2][x] = ∞；以此类推下去(略)。





3. 解：

这是整数背包问题。类似0/1背包问题，易证满足最优子结构性质：设y1,y2,..,yn是原问题的最优解，则y1,y2,..,yn-1是下述子问题：

 , S.T. ,xi∈{0,1,2},1≦i≤n-1

的最优解。如不然，y/1,y/2,…,y/n-1是子问题的最优解，则

,则，y/1,y/2,…,y/n-1,yn将是原问题的可行解解，且，与y1,y2,…,yn是最优解矛盾。

递推公式：设m[k][x]表示容量约束x，可装入1,2,…k件物品的最优解，则

m[k][x]=max{m[k-1][x],m[k-1][x-ak]+ck,m[k-1][x-2ak]+2ck} ,0≤x≤b

m[0][x]=0,if x>=0; m[0][x]=-∞,ifx<0; m[k][<0]= -∞,k=1,2,..n

据此不难给出算法的描述。（略。仿照0/1背包问题，但那个程序是从后向前推的，上述递推是从前往后。本问题可以扩展到xi可以取值0,1,2，…m,，递推部分改为m[k][x]=max{m[k-1][x-xiak]+xick}，对0≤xi≤m取max即可）

设A=<x1,x2,…,xn>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<xi1,xi2,…,xik>，i1<i2<…<ik，且xi1<xi2<…<xik，子序列含有k个整数。例如，A=<1,5,3,8,10,6,4,9>，它的长度为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。对输入实例A=<2,8,4,-4,5,9,11>，给出算法的计算过程和最后的解。

解：

使用动态规划设计技术。对于i=1,2,…n，考虑以xi为最后项的最长递增子序列的长度C[i]。如果在xi前存在xj<xi，则C[i]=max{C[j]}+1,；否则C[j]=1。显然，若C[i]的子序列是i1i2…iki，则C[ik]的子序列是i1i2…ik，满足最优子结构性质。因此C[1]=1，对i>1，

C[i]= C[i]=max{C[j]}+1,存在j,1<=j<I,xj<xi；或C[i]=1，任给j，1<=j<I,xj>xi；

所求最长子序列的长度是C=max{C[i]|i=1,2…n}。

算法设计时用K[i]记录C[i]最大时的j，C[i]=1时，K[i]=0；则从K[]可追踪出解序列：K[i],K[K[i]]….。算法描述略，参考习题五、1,3）。

算法复杂度：对每个i需要检索比i小的所有j，需O(n)时间，i取值n种，所以T(n)=O(n2)。

对于给定的实例A=<2,8,4,-4,5,9,11> ，计算过程如下：

C[1]=1，K[1]=0

C[2]=max{C[1]+1)=2，K[2]=1

C[3]=max{C[1]+1}=2，K[3]=1

C[4]=1, K[4]=0

C[5]=max{C[1]+1,C[3]+1,C[4]+1}=3，K[5]=3

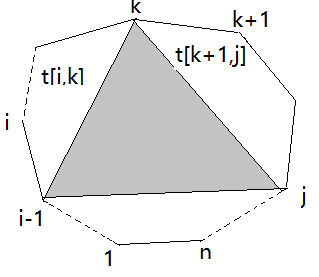
C[6]=max{ C[1]+1,C[2]+1,C[3]+1,C[4]+1,C[5]+1}=4 ，K[6]=5

C[7]=max{ C[1]+1,C[2]+1,C[3]+1,C[4]+1,C[5]+1,C[6]+1}=5，K[7]=6

C[7]=5是最大值。子序列追踪过程：11，A[K[7]]=A[6]=9,A[K[6]]=A[5]=5,A[K[5]]=A[3]=4,A[K[3]]=A[1]=2，得解为<2,4,5,9,11>。

设A是顶点为1,2,…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，如下图是5边形的所有划分方案。假设凸n边形的边及对角线的长度dij都是给定的正整数，1≤i<j≤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，分析算法复杂度。（提示：参考矩阵连乘问题）。

如图所示，n边形的顶点是1,2,..,n，顶点i-1,i,…j构成的凸多边形记做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,…,j-1中任选点k，它与底边(i-1)j构成一个三角形，如图中灰色三角形所示。这个三角形将A[i,j]划分为两个凸多变形：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]。根据动态规划的思想，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+dkj+d(i-1)j

其中d(i-1)k+dkj+d(i-1)j是三角形(i-1)kj的周长。于是得到递推关系：

t[i,j]=min{t[I,k]+t[k+1,j]+ d(i-1)k+dkj+d(i-1)j} for i≤k≤j；当i<j时；

t[i,j]=0，当i=j时。

有了这个递推关系，不难证明，若最优划分t[i,j]与(i-1)j相连的三角形第三顶是k，则这个划分也是A[i,k]和A[k+1,j]的最优划分。不难看出，这个递推关系与矩阵连乘算法的递推式非常相似，定义s[I,j=k记录得到最小权值的k的位置，算法描述参考矩阵连乘的迭代算法(略)。算法最坏复杂度O(n2)。