**深 圳 大 学 实 验 报 告**

**课程名称： 算法设计与分析**

**实验名称： 资源分配问题**

**学院： 计算机与软件学院 专业： 软件工程**

**报告人： 徐霞 学号： 2015150290 班级： 3**

**同组人： 无**

**指导教师： 杨烜**

**实验时间： 2017/10/30-2017/11/20**

**实验报告提交时间： 2017/11/20**

**教务处制**

1. **实验目的**

（1） 掌握动态规划算法设计思想。

（2） 掌握资源分配问题的动态规划解法。

1. **实验内容**
2. 某厂根据计划安排，拟将n台相同的设备分配给m个车间，各车间获得这种设备后，可以为国家提供盈利Ci j(i台设备提供给j号车间将得到的利润，1≤i≤n，1≤j≤m) 。问如何分配，才使国家得到最大的盈利？

2、 设计动态规划算法求解资源分配问题，写出求得最优值的递推公式。

3、 对小规模问题利用蛮力法验证动态规划方法求解的正确性（即最优分配方案、最优分配方案的值）。

4、 测试不同问题规模（按级数增长）的运行时间。

5、 能够实现的问题规模越大，成绩越高。

1. **实验过程**

**3.1 问题描述：**某厂根据计划安排，拟将n台相同的设备分配给m个车间，各车间获得这种设备后，可以为国家提供盈利Ci j(i台设备提供给j号车间将得到的利润，1≤i≤n，1≤j≤m) 。问如何分配，才使国家得到最大的盈利？

**3.2 动态规划思想：**

1. **刻画一个最优解的结构特征；**
2. **递归的定义最优解的值；**
3. **计算最优解的值；**
4. **利用计算出的信息构造一个最优解。**

**动态规划思想在资源分配问题中的实际应用**

进行如下变量定义：

C[i][j]：i台设备分配给j号车间所获利润；

V[i][j]：i台设备分配给前j个车间所获利润；

D[i][j]：D[i][j]=k，i个设备分给前j个车间，前j-1个车间分配的设备数。

1. **刻画一个最优解的结构特征**；

即寻找最优子结构，然后可以利用这种子结构从子问题的最优解构造出原问题的最优解。

最优子结构：

若使V[i][j]最大的分配方案是第j个车间分配了i-k台设备，前j-1个车间分配了k台设备，则对k个设备分配给前j-1个车间进行最优化分配；此时称V[i][j]具有最优子结构。下面进行证明：

**证明：V[i][j]的子结构V[k][j-1]是最优的。**

若V[k][j-1]不是最优的，则一定存在另一种分配方案，使得V[k][j-1]是最优的，记作max\_V[k][j-1]，此时对于i台设备分配给j个车间，有如下式子：

V[i][j]=V[k][j-1]+C[i-k][j]<max\_V[k][j-1]+C[i-k][j];

显然，此时V[i][j]不是最优的，与原假设矛盾。

所以命题得证，V[i][j]具有最优子结构。

1. **递归的定义最优解的值；**

用子问题的最优解来递归地定义原问题的最优解。

* + 1. i=0或j=0时，意味着0台设备分配给j个车间，或者i台设备分配给0个车间，此时显然V[i][j]=0;
    2. i>0且j>0时，假设使V[i]j]最大的分配方案是第j个车间分配了i-k台设备，前j-1个车间分配了k台设备，则有

V[i][j]= V[k][j-1]+C[i-k][j],其中0≤k≤i，k不是一下子就可以知道的，k是需要我们计算的，完整的公式应该是：

V[i][j]= 0i {V[i][j]，V[k][j-1]+C[i-k][j]}

因此，使获得利润V[i][j]最大的分配方案的递归求解公式变为：

0 ，i=0,或j=0时

V[i][j]=

0i { V[i][j]，V[k][j-1]+C[i-k][j]}，i>0且j>0时

1. **计算最优解的值；**

可以利用步骤2给出的公式写出一个算法，来计算规模为n个设备，m个车间的资源分配问题，并填好利润表V[i][j]，根据V[i][j]的定义，可知，V[n][m]即为最优解的值。

观察2给出的式子，发现V[i][j]的求解，只需要V表的j-1列的值，即计算大问题V[i][j]时，依赖于其子问题V[k][j-1]的解，所以在计算大问题的最优解时，先解决小问题的最优解，由小问题的最优解构造出大问题的最优解。

在生成V[i][j]的时候，会产生一个k值，此值就是使V[i][j]达到最大的分配方案，用一个二维数组D来记录，即D[i][j]=k,表示i个设备分给前j个车间，前j-1个车间分配的设备数。以便利用D表构造出使V[n][m]达到最大的分配方案。

1. **利用计算出的信息构造一个最优解。**

在步骤3中，我们已经生成了D表，D表记录了资源分配的最优解，如何根据D来求解出最优解呢？

首先明确D[i][j]=k的含义：i个设备分给前j个车间，前j-1个车间分配的设备数是k。

令k为设备数，初始时k=N，则：

第m个车间分配的设备数： k-D[k][m]，还剩k=D[k][m];

第m-1个车间分配的设备数：k-D[k][m-1],还剩k=D[k][m-1]

第m-2个车间分配的设备数：k-D[k][m-2],还剩k=D[k][m-2];

……

第2个车间分配的设备数： k-D[k][2],还剩k=D[k][2];

第1个车间分配的设备数： k-D[k][1].

**3.3 动态规划数据测试**

**3.3.1优化前：**

动态规划伪代码：

For i 1 to n //n为设备数

  For j  1 to m //m为车间数

    Max=0;

    For k 0 to i

      If Max< V[k][j-1]+C[i-k][j]  //递归式

        V[i][j]= V[k][j-1]+C[i-k][j] //记下最大利润

        D[i][j]=k;//记下路径

蛮力法伪代码：

brute(int \*x,int t)

if t>N

if（count(x)>max) //计算利润

max=count(x)

else

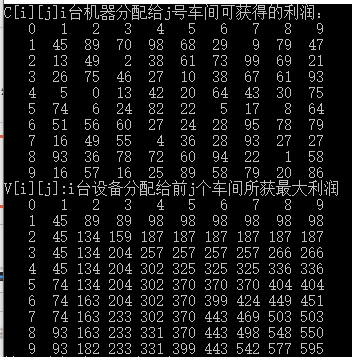
for i 0 to N

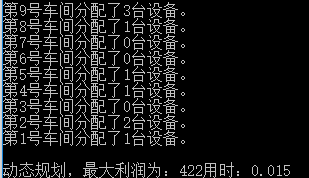
x[t]=i //第t个车间的设备数i

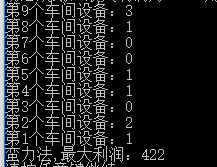
brute(x,t+1)//摆放第t+1个车间的设备

数据测试：

对于小规模数据，使用蛮力法验证动态规划法的正确性：



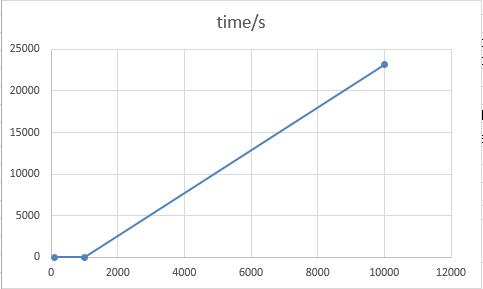




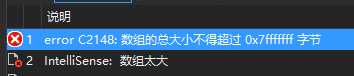
由此可见动态规划算法的正确性。

下面是动态规划算法的测试数据：





当数据规模为n=10000,m=10000时，已经耗时6.4h，当规模为n=20000，m=20000时耗时太长，没有时间等待，当规模为n=30000，m=30000时，直接爆出如下错误:



如果我们使用二维数组来存储利润表C，最大利润表V，以及分配表D，那么我们能跑的数据规模是不大的，因为收到了数组总大小的限制，如上图，此时如果想要跑更大规模的数据，则需要将这三个数组优化成一维数组。

**3.3.2 优化思想：**

对C，V优化：观察递推式：0i {V[k][j-1]+C[i-k][j]}，

对C表，计算V[i][j]时，我们只用到了C[i-k][j],即当计算V表中第j列的值时，只需用到C表中第j列的数据，所以C表数据我们只需用一个大小为N+1的一维数组即可。

对V表，计算V[i][j]时，我们只用到了V[k][j-1],即当计算V表中第j列的值时，只需用到V表中第j-1列的数据，所以V表数据我们只需用一个大小为N+1的一维数组即可。

注意：为避免数据覆盖丢失，j应该是 N to 1！

原因：此时我们是使用一维数组V存储最大利润，假设现在数组V存储的是原二维数组中第j-1列的数据，则在计算第j列数据时，设k1<k2，如果先计算V[k1]，则新的V[k1]会覆盖原始的V[k1]的值，那么在计算新的V[k2]的时候，他要用到第j-1列的数据V[k1]时，此时这个值已经被更新了，所以计算出来的值是不对的，所以为了避免数据丢失，V填表时应该是V[N] to V[1]。

对D表，**D矩阵是稀疏矩阵，可以将其压缩，存储为字符串形式**：

Temp=D[i][j],若数据可以跑到50000\*50000，那么temp最大为50000，有5个数字，所以给每个数字预留5个字符的位置，

1. Temp<=9，将数字1-9表示为字符 b-j；
2. 10<=Temp<=99,此时有3个0，将3个0表示为字符X;
3. 100<=Temp<=999,此时有2个0，将2个0表示为字符Y;
4. 1000<=Temp<=9999,此时有1个0，将1个0表示为字符0；
5. 10000<=Temp<=50000,此时将数字转为字符串表示即可。

**注意：按列存储，一列存储为一个字符串；**

原因：明确D[i][j]的意义：D[i][j]=k，i个设备分给前j个车间，前j-1个车间分配的设备数。在使用表D时，我们首先取n个设备分配给m个车间，前j-1个车间获得的设备数，即D[n][m]，则可以知道第m个车间获得的设备数为n-D[n][m]，接下来应该取k=D[n][m]个设备分配给m-1个车间，前m-2个车间获得的设备数，即取D[k][m-1]，可以看出，每列的数据都只取一次一个，所以按列存储的时候比较容易在构造一个最优解。

优化后伪代码：

For i 1 to m //m为车间数

For m 1 to n //生成一列利润

C[m]=rand()%100

  For j  n to 1 //n为设备数

b=“” D\_temp=0

    For k 0 to j

      If V[j]< V[j-k]+C[k]  //递归式

        V[j] = V[j - k] + C[k] //记下最大利润

        D\_temp=k

b=“”

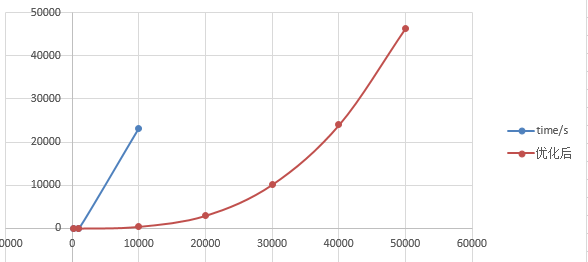
b.append(toString(D\_temp))

D[i]=append(b)

**3.3.3 优化后数据测试：**

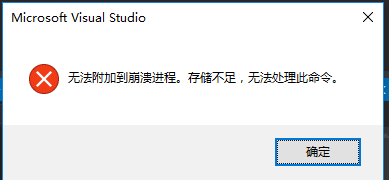
****



****

我的运行内存为8G，可以跑到N=50000,M=50000的规模，优化前只能跑到N=10000，M=10000，且二者在相同规模时，其效率也有明显差距，显然优化后代码跑的更快。

当运行到N=60000，M=60000规模时，运行到一半，内存爆掉了：



**结论：**