动态规划复习：

1、问题描述：某个工厂计划要采购n个设备，每个设备的采购价格为$P_i(i=1,...,n)$，装备该设备后能产生的效益为$M_i(i=1,...,n)$。因为全球金融危机的影响，该工厂大幅缩减了预算，用于本次采购的预算总金额为C。请设计采购方案使得产生的效益最大。要求算法的时间复杂度是多项式级别复杂度。 输入：第一行是整数n和总金额C；第二行是n个整数$P_i(i=1,...,n)$；第三行是效益$M_i(i=1,...,n)$，整数之间用空格隔开。 输出：本次采购能产生的最大效益的总值。

参考答案：评分指导： 本题是0-1背包问题的具体实例，可以应用动态规划算法求解，其时间复杂度为O(NC). 评分标准： 如果明确用动态规划算法，最优子结构、状态表示和转移方程的描述正确，则可得18分。 代码正确，描述清晰，则可得30分。 参考代码如下： double purchase(int numItems,int *w,double *v,int capacity){ int i,j; double Val[MaxC]; memset(Val,0,sizeof(Val)); for(i=0;i<numItems;i++) for(j=capacity;j>=0;j--) if(j>=w[i] && Val[j]<Val[j-w[i]]+v[i]) Val[j]=Val[j-w[i]]+v[i]; return Val[capacity]; }

例如：

N=4， C=8

n	1	2	3	4
P	2	3	4	5
M	3	4	5	6

动态规划：

动态规划与分治法类似，都是把大问题拆分成小问题，通过寻找大问题与小问题的递推关系，解决一个个小问题，最终达到解决原问题的效果。但不同的是，分治法在子问题和子子问题等上被重复计算了很多次，而动态规划则具有记忆性，通过填写表把所有已经解决的子问题答案纪录下来，在新问题里需要用到的子问题可以直接提取，避免了重复计算，从而节约了时间，所以在问题满足最优性原理之后，用动态规划解决问题的核心就在于填表，表填写完毕，最优解也就找到。

递归关系：

1) j <P[i]   V(i,j)=V(i-1,j)

2) j>=P(i)   V(i,j)=max｛V(i-1,j)，V(i-1,j-P(i))+v(i)｝

最基本的方法就是填表

i/j	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0								
2	0								
3	0								
4	0								

表格 含义是竖轴来说 当预算为j时分别对应的能买的最大收益的物品，

例如当i=1，j=1时P[1]=2,M[1]=3;因为j<W[1],所以买不到，所以val[1,1]=0,

V(1,1)=V(1-1,1)=0

当i=1，j=2，P[1]=2,v[1]=3,有j=p[1],所以一可以进货物品1，收益就是

V(1,2)=max｛V(1-1,2)，V(1-1,2-P(1))+v(1)｝=max｛0，0+3｝=3;

如此下去，填到最后一个，i=4，j=8，P(4)=5，v(4)=6，有j>P(4)，故V(4,8)=max｛V(4-1,8)，V(4-1,8-P(4))+v(4)｝=max｛9，4+6｝=10

i/j	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	3	3	3	3	3	3	3
2	0	0	3	4	4	7	7	7	7
3	0	0	3	4	5	7	8	9	9
4	0	0	3	4	5	7	8	9	10

```
void FindMax()//动态规划
{
    int i,j;
    //填表
    for(i=1;i<=n;i++)
    {
        for(j=1;j<=c;j++)
        {
            if(j<P[i])//包装不进
            {
                V[i][j]=V[i-1][j];
            }
            else//能装
            {
                if(V[i-1][j]>V[i-1][j-P[i]]+v[i])//不装价值大
                {
                    V[i][j]=V[i-1][j];
                }
                else//前i-1个物品的最优解与第i个物品的价值之和更大
                {
                    V[i][j]=V[i-1][j-P[i]]+v[i];
                }
```

```
                }
            }
        }
    }
```

空间优化后：

空间优化其实就由于每次的变化都只是变化上一步之后的数组；所以二维数组可以用一维来记录每次的最终的结果

v(4,8)=max { v(4-1,8), v(4-1,8-P(4))+v(4) } =max {9, 4+6} =10

i/j	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	3	3	3	3	3	3	3
2	0	0	3	4	4	7	7	7	7
3	0	0	3	4	5	7	8	9	9
4	0	0	3	4	5	7	8	9	10

如图 就是第三行的数据依赖在第二行的数据，所以只需要记录最近一行的数据就行；

```
void FindMaxBetter()//优化空间后的动态规划
{
    int i,j;
    for(i=1;i<=n;i++)
    {
        for(j=c;j>=0;j--)
        {
            if(B[j]<=B[j-P[i]]+v[i] && j-w[i]>=0 )//二维变一维
            {
                B[j]=B[j-P[i]]+v[i];
            }
        }
    }
}
```