目录

咨询电话：18561566921

动态规划经典题目

1. 基础题目

1.1. 上帝选人

【问题描述】

世界上的人都有智商 IQ 和情商 EQ。我们用两个数字来表示人的智商和情商，数字大就代表其相应智商或情商高。现在你面前有 N 个人，这 N 个人的智商和情商均已知，请你选择出尽里多的人，要求选出的人中不存在任意两人 i 和 j，i 的智商大于 j 的智商但 i 的情商小于 j 的情商。

【输入】

第一行一个正整数 N，表示人的数里。

第二行至第 N+1 行，每行两个正整数，分别表示每个人的智商和情商。

【输出】

仅一行，为最多选出的人的个数。

【样例输入】

```
3
100 100
120 90
110 80
```

【样例输出】

```
2
```

【问题分析】

原题中的要求“不存在任意两人 i 和 j，i 的智商大于 j 的智商，但 i 的情商小于 j 的情商。”

将其转化成：就是要求不存在 i,j 满足：如果（iq[i] > iq[j] ），但(eq[i] < eq[j])。

即选出的人 i 和 j 要满足：(IQ[i]>=IQ[j])and(EQ[i]>=EQ[j]) 或者(IQ[i]<=IQ[j])and(EQ[i]<=EQ[j])

【解题思路】

1.将所有人 IQ（或者 EQ）从大到小排序。

2.求 EQ（或者 IQ ）的最长非递增子序列长度

【参考代码】

```
#include <cstdio>
#include <algorithm>
using namespace std;
struct node
{
    int iq,eq;
}e[1010];
int n,f[1010];
bool cmp(const node &x,const node &y)//排序比较函数，按 iq 降序排序
{
    return x.iq>y.iq;
}
int main()
{
    scanf("%d",&n);
    for(int i=1;i<=n;i++)
    {
        scanf("%d%d",&e[i].iq,&e[i].eq);
        f[i]=1;
    }
```


咨询电话：18561566921

```
    sort(e+1,e+n+1,cmp);
    for(int i=n;i;i--)
    {
        for(int j=i+1;j<=n;j++)
        {
            if(e[i].eq>=e[j].eq)
            {
                f[i]=max(f[i],f[j]+1);
            }
        }
    }
    int ans=f[1];
    for(int i=2;i<=n;i++)
    {
        ans=max(ans,f[i]);
    }
    printf("%d\n",ans);
    return 0;
}
```

1.2. 最大正方形面积

【问题描述】

给定一个 R 行 C 列的 01 矩阵，求一个最大的正方形全 1 子矩阵，并输出该最大正方形子矩阵的面积。

【输入】

第一行给出两个正整数 R,C，表示矩阵有 R 行 C 列；

接下来 R 行 C 列给出这个 01 矩阵，行内相邻两元素用一个空格隔开。

R,C<=1000。

【输出】

一个数，为该最大正方形子矩阵的面积。

【样例输入】

```
5 8
0 0 0 1 1 1 0 1
1 1 0 1 1 1 1 1
0 1 1 1 1 1 0 1
1 0 1 1 1 1 1 0
1 1 1 0 1 1 0 1
```

【样例输出】

```
9
```

【算法分析】

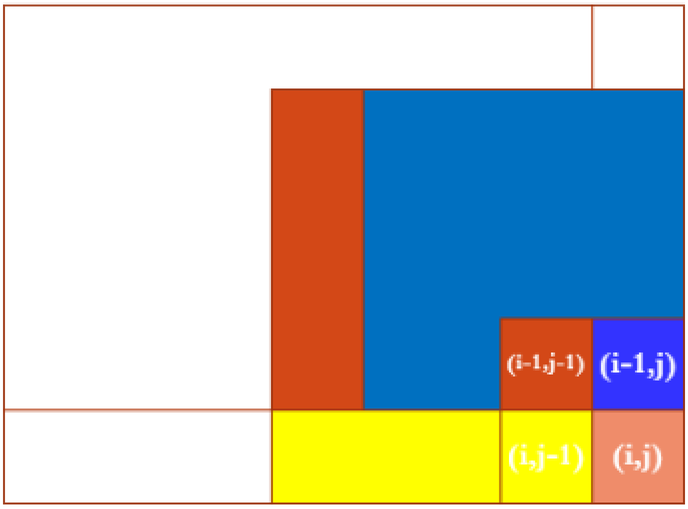
f[i][j]定义为以(i,j)为右下角顶点的最大正方形边长。

最大正方形边长：

ans=max{f[i][j]}　　1<=i<=R，1<=j<=C。

最大面积：ans*ans

if(a[i][j]==0)　　f[i][j]=0;



if(a[i][j]==1)

情况 1：f[i][j]=f[i-1][j]+1

情况 2：f[i][j]=f[i-1][j-1]+1

情况 3：f[i][j]=f[i][j-1]+1

综上三种情况：

当 a[i][j]=1

f[i][j]=min{f[i][j-1],f[i-1][j],f[i-1][j-1]}+1;

【参考代码】

```
#include<iostream>
#include<cstring>
using namespace std;
const int maxn=1010;
int a[maxn][maxn],f[maxn][maxn],r,c,mmax=0;
int main()
{
    cin>>r>>c;
    for(int i=1;i<=r;i++)
    {
        for(int j=1;j<=c;j++)
        {
            cin>>a[i][j];
        }
    }
    for(int i=1;i<=r;i++)
    {
        for(int j=1;j<=c;j++)
        {
            if(a[i][j]==1)
            {
                f[i][j]=min(min(f[i][j-1],f[i-1][j-1]),f[i-1][j])+1;
                mmax=max(f[i][j],mmax);
            }
        }
    }
    cout<<mmax*mmax<<endl;
    return 0;
```

```
}
```

1.1. 方格取数

【问题描述】

设有 N×N 的方格图，我们在其中的某些方格中填入正整数，而其它的方格中则放入数字 0。如下图所示：

A

0	0	0	0	0	0	0	0
0	0	13	0	0	6	0	0
0	0	0	0	7	0	0	0
0	0	0	14	0	0	0	0
0	21	0	0	0	4	0	0
0	0	15	0	0	0	0	0
0	14	0	0	0	0	0	0
0	0	0	0	0	0	0	0

B

某人从图中的左上角的 A 出发，可以向下行走，也可以向右行走，直到达右下角的 B 点。在走过的路上，他可以取走方格中的数(取走后的方格中将变为数字 0)。此人从 A 点到 B 点共走了两次，试找出两条这样的路径，使得取得的数字和为最大。

【输入】

第一行为一个整数 N（N≤10），表示 N×N 的方格图。接下来的每行有三个整数，第一个为行号数，第二个为列号数，第三个为在该行、该列上所放的数。一行 0 0 0 表示结束。

【输出】

包含一个整数，表示两条路径上取得的最大的和。

【样例输入】

```
8
2 3 13
2 6 6
3 5 7
4 4 14
5 2 21
5 6 4
6 3 15
7 2 14
0 0 0
```

【样例输出】

```
67
```

【算法分析】

一个四重循环枚举两条路分别走到的位置。由于每个点均从上或左继承而来，故内部有四个 if，分别表示两个点从上上、上左、左上、左左继承来时，加上当前两个点所取得的最大值。a[i][j]表示(i,j)格上的值，sum[i][j][h][k]表示第一条路走到(i,j)，第二条路走到(h,k)时的最优解。例如，sum[i][j][h][k] = max{sum[i][j][h][k], sum[i-1][j][h-1][k] + a[i][j] + a[h][k]}，表示两点均从上面位置走来。

当(i, j) <> (h, k))时

sum[i][j][h][k]:=max{sum[i-1][j][h-1][k],sum[i][j-1][h][k-1],sum[i-1][j][h][k-1],sum[i][j-1][h-1][k]}+a[i][j]+a[h][k];

当(i, j) = (h, k)时

sum[i][j][h][k]:=max{sum[i-1][j][h-1][k],sum[i][j-1][h][k-1],sum[i-1][j][h][k-1],sum[i][j-1][h-1][k]}+a[i]

[j];

【参考代码】

```
#include<cstdio>
#include<algorithm>
using namespace std;
int a[51][51];
int sum[51][51][51][51];
int n,i,j,h,k,x,y,z;
int main()
{
scanf("%d%d%d%d",&n,&x,&y,&z);
    while (x&&y&&z)
    {
        a[x][y]=z;
        scanf("%d%d%d",&x,&y,&z);
    }
for (int i=1;i<=n;i++)
{
    for (int j=1;j<=n;j++)
    {
        for (int h=1;h<=n;h++)
        {
            for (int k=1;k<=n;k++)
            {
                int tmp1=max(sum[i-1][j][h-1][k],sum[i][j-1][h][k-1]);
                int tmp2=max(sum[i-1][j][h][k-1],sum[i][j-1][h-1][k]);
                sum[i][j][h][k]=max(tmp1,tmp2)+a[i][j];
                if (i!=h && j!=k)
                {
                    sum[i][j][h][k]+=a[h][k];
                }
            }
        }
    }
}
printf("%d\n",sum[n][n][n][n]);
return 0;
}
```

【算法分析】

本题满足“曼哈顿距离”，可优化状态变量为三维数组。

【参考代码】

```
#include <bits/stdc++.h>
using namespace std;
int a[10][10];
int dp[20][10][10];
int f(int a,int b,int c,int d)
{
    return max(max(max(a,b),c),d);
}
int main()
```

```
{
    int n,x,y,z;
    cin>>n;
    while(1)
    {
        cin>>x>>y>>z;
        if(x==0 && y==0 && z==0)
        {
            break;
        }
        a[x][y]=z;
    }
    for(int k=1;k<=2*n;k++)///走 2*n 步到达右下角
    {
        for(int i=1;i<=k;i++)
        {
            for(int j=1;j<=k;j++)
            {
    dp[k][i][j]=f(dp[k-1][i][j],dp[k-1][i-1][j],dp[k-1][i][j-1],dp[k-1][i-1][j-1]);
                if(i==j) dp[k][i][j]+=a[k-i][i];///相同
                else dp[k][i][j]+=a[k-i][i]+a[k-j][j];
            }
        }
    }
    cout<<dp[2*n][n][n];
    return 0;
}
```

1.2. 数字三角形 2

【问题描述】

有一个数字三角形，从最顶层出发，每一步只能向左下或右下方向走。编程求从最顶层到最底层的一条路所经过位置上的数字之和模(%)100 的最大值。

【输入】

第一行：n(1<=n<=25),数字三角形共有 n 行；

以下 R 行：依次表示数字三角形中每行中的数字。

每个数都是非负的，且<=100.

```
    1
  99  98
1   1   1
```

【输出】

一个正整数，路径上数字之和 模 100 的最大值。

【样例输入】

```
3
1
90 9
9 10 99
```

【样例输出】

```
20
```

【算法分析】

错误分析：

定义 f[i][j]表示到达第 i 行第 j 列位置的最优值，则：

f[i][j]=max{(f[i-1][j-1]+a[i][j])%100,(f[i-1][j]+a[i][j])%100}，初始值：f[1][1]=a[1][1]，max{f[n][i]}即为所求。

上述分析错误，因为不具备最优子结构。

正确做法：

定义 f[i][j][k] 表示到达第 i 行第 j 列位置能否得到 k(0<=k<=99)。f[i][j][k]=f[i][j][k] || f[i-1][j-1][(k-a[i][j]+100)%100] ||f[i-1][j][(k-a[i][j] +100)%100];

初始值:f[1][1][a[1][1]]=true,其余均为 false。

【参考代码】

```
#include <cstdio>
#include <algorithm>
using namespace std;
const int M = 100;
bool f[30][30][M];
int n,a[30][30];
int main()
{
    scanf("%d", &n);
    for(int i=1;i<= n;i++)
    {
    for(int j=1;j<=i;j++)
    {
        scanf("%d", &a[i][j]);
    }
    }
    f[1][1][a[1][1]] = true;
    for(int i=1;i<=n;i++)
    {
        for(int j= 1;j<=i;j++)
        {
            for(int k=0;k<M;k++)
            {
                f[i][j][k]=f[i][j][k] || f[i-1][j-1][(k-(a[i][j]%M)+ M)%M]
                            ||f[i-1][j][(k-(a[i][j]%M)+M)%M];
            }
        }
    }
    int ans = 0;
    for(int i=1;i<=n;++i)
    {
        for(int j=0; j<M;++j)
        {
            if(f[n][i][j])
            {
                ans=max(ans,j);
            }
```

```
        }
    }
    printf("%d", ans);
    return 0;
}
```

1.3. 多重背包

有 N 种物品和一个容量为 V 的背包。第 i 种物品最多有 n[i]件可用，每件费用是 w[i]，价值是 c[i]。求解将哪些物品装入背包可使这些物品的费用总和不超过背包容量，且价值总和最大。

基本算法:

这题目和完全背包问题很类似。基本的方程只需将完全背包问题的方程略微一改即可，因为对于第 i 种物品有 n[i]+1 种策略：取 0 件，取 1 件……取 n[i]件。令 f[i][v]表示前 i 种物品恰放入一个容量为 v 的背包的最大权值，则：f[i][v]=max{f[i-1][v-k*w[i]]+k*c[i]|0<=k<=n[i]}。复杂度是 O(V*Σn[i])。

【问题描述】

为了庆贺班级在校运动会上取得全校第一名成绩，班主任决定开一场庆功会，为此拨款购买奖品犒劳运动员。期望拨款金额能购买最大价值的奖品，可以补充他们的精力和体力。

【输入】

第一行二个数 n(n<=500)，m(m<=6000)，其中 n 代表希望购买的奖品的种数，m 表示拨款金额。

接下来 n 行，每行 3 个数，v、w、s，分别表示第 i 种奖品的价格、价值（价格与价值是不同的概念）和可购买的数量（买 0 件到 s 件均可），其中 v<=100，w<=1000，s<=10。

【输出】

第一行：一个数，表示此次购买能获得的最大的价值（注意！不是价格）。

【样例输入】

```
5 1000
80 20 4
40 50 9
30 50 7
40 30 6
20 20 1
```

【样例输出】

```
1040
```

【参考代码】

【解法一】朴素算法

```
#include <cstdio>
#include <iostream>
using namespace std;
int v[6002], w[6002], s[6002];
int f[6002];
int n,m;
int main()
{
    scanf("%d%d",&n,&m);
    for (int i=1;i<=n;i++)
    {
        scanf("%d%d%d",&v[i],&w[i],&s[i]);
    }
    for(int i=1;i<=n;i++)
```

```
    {
        for(int j=m;j>=0;j--)
        {
            for(int k=0;k<=s[i];k++)
            {
                if(j-k*v[i]<0)
                {
                    break;
                }
                f[j]=max(f[j],f[j-k*v[i]]+k*w[i]);
            }
        }
    }
    printf("%d",f[m]);
    return 0;
}
```

【优化算法】进行二进制优化，转换为01背包

基本思想是转化为01背包求解，方法是：将第i种物品分成若干件物品，其中每件物品有一个系数，这件物品的费用和价值均是原来的费用和价值乘以这个系数。使这些系数分别为1,2,4,...,2^k-1,n[i]-2^k+1，且k是满足n[i]-2^k+1>0的最大整数(注意：这些系数已经可以组合出1~n[i]内的所有数字)。例如，如果n[i]为13，就将这种物品分成系数分别为1,2,4,6的四件物品。

这样就将第i种物品分成了O(logn[i])种物品,将原问题转化为了复杂度为O(V*Σlogn[i])的01背包问题,有很大改进。

【参考代码】

```
#include<cstdio>
using namespace std;
int v[10001],w[10001],f[6001];
int n,m,nl;
int max(int a,int b)
{
    return a>b?a:b;
}
int main()
{
    scanf("%d%d",&n,&m);
    for(int i=1;i<=n;i++)
    {
        int x,y,s,t=1;
        scanf("%d%d%d",&x,&y,&s);
        while(s>=t)
        {
            v[++nl]=x*t;//相当于nl++;v[nl]=x*t;
            w[nl]=y*t;
            s-=t;
            t*=2;
        }
        v[++nl]=x*s;
        w[nl]=y*s;//把s以2的指数分堆：1，2，4，…，2^(k-1)，s-2^k+1,
    }
    for(int i=1;i<=nl;i++)
    {
        for(int j=m;j>=v[i];j--)
        {
            f[j]=max(f[j],f[j-v[i]]+w[i]);
        }
    }
    printf("%d\n",f[m]);
```

```
    return 0;
}
```

1.4. 混合背包

如果将 01 背包、完全背包、多重背包混合起来。也就是说，有的物品只可以取一次（01 背包），有的物品可以取无限次（完全背包），有的物品可以取的次数有一个上限（多重背包）。

01 背包与完全背包的混合

考虑到在 01 背包和完全背包中最后给出的伪代码只有一处不同，故如果只有两类物品：一类物品只能取一次，另一类物品可以取无限次，那么只需在对每个物品应用转移方程时，根据物品的类别选用顺序或逆序的循环即可，复杂度是 O(VN)。

伪代码如下：

```
for i=1..N
    if 第 i 件物品是 01 背包
        for v=V..0
            f[v]=max{f[v],f[v-w[i]]+c[i]};
    else if 第 i 件物品是完全背包
        for v=0..V
        f[v]=max{f[v],f[v-w[i]]+c[i]};
```

再加上多重背包

如果再加上有的物品最多可以取有限次，那么原则上也可以给出 O(VN)的解法：遇到多重背包类型的物品用单调队列解即可。但如果不考虑超过 NOIP 范围的算法的话，用多重背包中将每个这类物品分成 O(log n[i])个 01 背包的物品的方法也已经很优了。

【问题描述】

一个旅行者有一个最多能用 V 公斤的背包，现在有 n 件物品，它们的重量分别是 W1，W2，...，Wn，它们的价值分别为 C1,C2,...,Cn。有的物品只可以取一次（01 背包），有的物品可以取无限次（完全背包），有的物品可以取的次数有一个上限（多重背包）。求解将哪些物品装入背包可使这些物品的费用总和不超过背包容量，且价值总和最大。

【输入】

第一行：二个整数，V(背包容量，V<=200)，N(物品数量，N<=30)；

第 2..N+1 行：每行三个整数 Wi,Ci,Pi，前两个整数分别表示每个物品的重量，价值，第三个整数若为 0，则说明此物品可以购买无数件，若为其他数字，则为此物品可购买的最多件数(Pi)。

【输出】

仅一行，一个数，表示最大总价值。

【样例输入】

```
10 3
2 1 5
3 3 1
4 5 4
```

【样例输出】

```
11
```

【样例解释】

选第一件物品 1 件和第三件物品 2 件。

【参考代码】

```
#include<cstdio>
using namespace std;
int m, n;
```

```
int w[31], c[31], p[31], f[201];
int maxx(int x,int y)
{
    return x>y?x:y;
}
int main()
{
    scanf("%d%d",&m,&n);
    for(int i=1;i<=n;i++)
    {
        scanf("%d%d%d",&w[i],&c[i],&p[i]);
    }
    for(int i=1;i<=n;i++)
    {
        if(p[i]==0)//完全背包
        {
            for(int j=w[i];j<=m;j++)
            {
                f[j]=maxx(f[j],f[j-w[i]]+c[i]);
            }
        }
        else
        {
            for(int j=1;j<=p[i];j++)//01 背包和多重背包
            {
                for (int k = m; k >= w[i]; k--)
                {
                    f[k] = maxx(f[k],f[k-w[i]]+c[i]);
                }
            }
        }
    }
    printf("%d",f[m]);
    return 0;
}
```

【参考代码二】转换成多重背包

```
#include<cstdio>
using namespace std;
int m, n;
int w[31], c[31], p[31], f[201];
int maxx(int x,int y)
{
    return x>y?x:y;
}
int main()
{
    scanf("%d%d",&m,&n);
    for(int i=1;i<=n;i++)
    {
        scanf("%d%d%d",&w[i],&c[i],&p[i]);
        if(p[i]==0)//转换成多重背包，未做数据清洗
        {
            p[i]=m/w[i];
        }
    }
    for(int i=1;i<=n;i++)
    {
        for(int j=1;j<=p[i];j++)//01 背包和多重背包
        {
```

```
            for (int k = m; k >= w[i]; k--)
            {
                f[k] = maxx(f[k],f[k-w[i]]+c[i]);
            }
        }
    }
    printf("%d",f[m]);
    return 0;
}
```

1.5. 有依赖的背包问题

这种背包问题的物品间存在某种“依赖”的关系。也就是说，i 依赖于 j，表示若选物品 i，则必须选物品 j。为了简化起见，我们先设没有某个物品既依赖于别的物品，又被别的物品所依赖；另外，没有某件物品同时依赖多件物品。

NOIP2006 中金明的预算方案一题就是此类问题。

【问题描述】

金明今天很开心，家里购置的新房就要领钥匙了，新房里有一间金明自己专用的很宽敞的房间。更让他高兴的是，妈妈昨天对他说：“你的房间需要购买哪些物品，怎么布置，你说了算，只要不超过 N 元钱就行”。今天一早，金明就开始做预算了，他把想买的物品分为两类：主件与附件，附件是从属于某个主件的，下表就是一些主件与附件的例子：

主件 附件

电脑 打印机，扫描仪

书柜 图书

书桌 台灯，文具

工作椅 无

如果要买归类为附件的物品，必须先买该附件所属的主件。每个主件可以有 0 个、1 个或 2 个附件。附件不再有从属于自己的附件。金明想买的东西很多，肯定会超过妈妈限定的 N 元。于是，他把每件物品规定了一个重要度，分为 5 等：用整数 1-5 表示，第 5 等最重要。他还从因特网上查到了每件物品的价格（都是 10 元的整数倍）。他希望在不超过 N 元（可以等于 N 元）的前提下，**使每件物品的价格与重要度的乘积的总和最大。**

设第 j 件物品的价格为 v[j]，重要度为 w[j]，共选中了 k 件物品，编号依次为 j1,j2,…,jk 则所求的总和为：

v[j1]×w[j1]+v[j2]×w[j2]+…+v[jk]×w[jk]。

请你帮助金明设计一个满足要求的购物单。

【输入】

第 1 行，为两个正整数，用一个空格隔开：

N，m（其中 N(<32000)表示总钱数，m(<60)为希望购买物品的个数。）

从第 2 行到第 m+1 行，第 j 行给出了编号为 j-1 的物品的基本数据，每行有 3 个非负整数：v，p，q（其中 v 表示该物品的价格（v<10000），p 表示该物品的重要度（1-5），q 表示该物品是主件还是附件。如果=0，表示该物品为主件，如果 q>0，表示该物品为附件，q 是所属主件的编号）。

【输出】

一个正整数，为不超过总钱数的物品的价格与重要度乘积的总和的最大值（<200000）。

【样例输入】

```
1000 5
800 2 0
400 5 1
300 5 1
400 3 0
500 2 0
```

【样例输出】

```
2200
```

【算法分析】

考虑到每个主件最多只有两个附件，因此我们可以通过转化，把原问题转化为 01 背包问题来解决，在用 01 背包之前我们需要对输入数据进行处理，把每一种物品归类，即：把每一个主件和它的附件看作一类物品。处理好之后，我们就可以使用 01 背包算法了。在取某件物品时，我们只需要从以下四种方案中取最大的那种方案：只取主件、取主件＋附件 1、取主件＋附件 2、取主件＋附件 1＋附件 2。很容易得到如下状态转移方程：

f[i,j]=max{f[i-1,j]， f[i][j]

f[i-1,j-a[i,0]]+a[i,0]*b[i,0]，

f[i-1,j-a[i,0]-a[i,1]]+a[i,0]*b[i,0]+a[i,1]*b[i,1]，

f[i-1,j-a[i,0]-a[i,2]]+a[i,0]*b[i,0]+a[i,2]*b[i,2]，

f[i-1,j-a[i,0]-a[i,1]-a[i,2]]+a[i,0]*b[i,0]+a[i,1]*b[i,1]+a[i,2]*b[i,2]}

其中，f[i,j]表示用 j 元钱，买前 i 类物品，所得的最大值，a[i,0]表示第 i 类物品主件的价格，a[i,1]表示第 i 类物品第 1 个附件的价格，a[i,2]表示第 i 类物品第 2 个附件的价格，b[i,0],b[i,1],b[i,2]分别表示主件、第 1 个附件和第 2 个附件的重要度。

【参考代码】

```
#include <iostream>
using namespace std;
int w[65][3],v[65][3],f[65][3205];
int main()
{
    int n,m,c,p,q,i,j,t;
    cin>>n>>m;
    n/=10; //都是 10 的整数倍，因此可以节约空间和时间
    for(i=1;i<=m;i++)
    {
        cin>>c>>p>>q;
        c/=10;
        if(q==0)
        {
            w[i][q]=c;
            v[i][q]=c*p;//同步计算该物品的价格*重要度
        }
        else if(w[q][1]==0)//如果是第一个附件
        {
            w[q][1]=c;
            v[q][1]=c*p;
        }
        else //是第二个附件
        {
            w[q][2]=c;
            v[q][2]=c*p;
        }
    }
    for(i=1;i<=m;i++)
    {
        for(j=0;j<=n;j++)
        {
            f[i][j]=f[i-1][j];
            if(j>=w[i][0]) //放主件
            {
                t=f[i-1][j-w[i][0]]+v[i][0];
                if(t>f[i][j])
                {
```

```
                f[i][j]=t;
            }
        }
        if(j>=w[i][0]+w[i][1])//放主件+第一个附件
        {
            t=f[i-1][j-w[i][0]-w[i][1]]+v[i][0]+v[i][1];
            if(t>f[i][j])
            {
                f[i][j]=t;
            }
        }
        if(j>=w[i][0]+w[i][2])//放主件+第二个附件
        {
            t=f[i-1][j-w[i][0]-w[i][2]]+v[i][0]+v[i][2];
            if(t>f[i][j])
            {
                f[i][j]=t;
            }
        }
        if(j>=w[i][0]+w[i][1]+w[i][2])//放主件+两个附件
        {
            t=f[i-1][j-w[i][0]-w[i][1]-w[i][2]]+v[i][0]+v[i][1]+v[i][2];
            if(t>f[i][j])
            {
                f[i][j]=t;
            }
        }
    }
  }
  cout<<f[m][n]*10<<endl;
  return 0;
}
```