T1

题目描述

设计一次竞赛使得选手最大可能的得分尽可能高。可供选择的题目共有N类，你可以从每一类题目中选出任意道题或不选组成一次竞赛。某一类题中的每一道题都需要相同的时间并能得到相同的分数。竞赛总时间不超过M分钟

输入格式

第一行两个整数M, N分别表示竞赛总时间与题目类数；

第2到N+1行每行两个整数，分别表示这类题每一道的分数和所需时间。

输出格式

给定条件下最大可能的得分

样例输入

```
300 4
100 60
250 120
120 100
35 20
```

样例输出

```
605
```

数据规模与约定

1 <= N <= 10,000

1 <= M <= 10,000

数据已加强4.18

T1算法分析

时间限制：1000MS　　内存限制：65536KB

- 属于完全背包
- 看到时间1s,内存65536KB=64MB，注意空间复杂度，所以选择一维状态

```
for(reg int i=1;i<=n;i++)
{
    scanf("%d%d",&c[i],&w[i]);
    for(reg int j=w[i];j<=m;j++)//完全背包正序枚举
    cmax(f[j],f[j-w[i]]+c[i]);
}
```

T2

再过 T 天就是 Kate 的生日了，Coffee 打算送它一些蛋糕。在 Coffee 和 Kate 的世界，蛋糕是做不出来的，而是种出来了。

Coffee 收藏着 N 个蛋糕种子。第 i 个蛋糕种子需要在室内培育连续 P_i 天才能长成幼苗，之后将它移栽到室外，再经过 Q_i 天后就会结出一个美味度为 R_i 的蛋糕。移栽蛋糕苗、为室外的蛋糕浇水以及收取成熟的蛋糕花费的时间对于 Coffee 来说可以忽略不计，但由于长成幼苗前的蛋糕比较娇嫩，照顾起来也更麻烦，Coffee 每天最多只会照顾一棵蛋糕幼苗。

更具体地，如果 Coffee 在第 t_0 天开始室内培育第 i 个蛋糕种子，那么它的室内培育工作会占用 Coffee 第 $t_0, t_0+1, \ldots, t_0+P_i-1$ 天，并会在 $t_0+P_i+Q_i-1$ 天结出蛋糕。在第 $t_0, t_0+1, \ldots, t_0+P_i-1$ 天，Coffee 不会开始或同时培养其他种子。

Coffee 希望在 Kate 生日时，送给 Kate 的蛋糕们的美味度总和尽量大。也就是说，在接下来的 T 天内，Kate 最多能收获蛋糕的美味度总和最大是多少？注意，即使一个蛋糕苗在第 T 天前已经被移出了室外，只要它在 Kate 的生日前没能结出蛋糕，它的美味度就不会被计算到总美味度中。

输入格式

第一行，两个用空格隔开的整数 N 与 T

第二到 $N+1$ 行，每行用三个数表示 P_i, Q_i, R_i

输出格式

一个整数，表示 Coffee 在 Kate 生日时，能送给 Kate 的蛋糕的美味度总和最大值

样例输入

```
3 6
6 1 100
2 2 20
3 1 30
```

样例输出

```
50
```

数据规模与约定

对于 20% 的数据，$1 \leqslant T \leqslant 1500$, $1 \leqslant N \leqslant 30$;

对于 100% 的数据，$1 \leqslant T \leqslant 20000$, $1 \leqslant N \leqslant 300$, $1 \leqslant P_i, Q_i, R_i \leqslant 2000$.

T2算法分析

- 若不考虑Qi的情况，这就是一道01背包问题的裸题。第i个物品消耗为Pi，价值为Ri。因此我们想到，在考虑第i个物品放或不放时，我们都要把当前背包的容积“腾出”Qi的空间来，保证幼苗能长成蛋糕。（因为我们知道，如果幼苗不能在T天内长出蛋糕，那我们种了还不如不种）。剩下的空间可以用来对“幼苗在室内的时间”做01背包。
- 首先发现这里有一个贪心的算法，即Qi较大的放在前面种，总比后面种更优。因此我们首先，对Qi进行从大到小的排序。保证Qi大的被先种掉。

- 考虑完以上问题，背包模型就来了：用f[i][j]表示当前做到第i个物品，恰巧能在第j天移出室外（为什么要这么设定，而不是按最原始的那种意义来定义f[i][j]呢？因为这个背包对于物品放进的时间先后有要求。），收获的美味度最大值。则对于第i个物品我们有：
- for(int j=t-q[i];j>=p[i];j--)
- //j的上界必须是t-q[i],即“腾出q[i]天”保证所有种的幼苗能长成蛋糕，当j>t-q[i]时，种的蛋糕长不成熟，种了不如不种。
- f[i][j]=max(f[i-1][j],f[i-1][j-p[i]]+r[i]);//01背包，第i种种子种，或者不种

T2算法分析2

- 综上所述，我们这道题算法的本质就是，在保证种子能成熟的情况下才考虑种该种子。即对于每一个物品，我们考虑放不放时，都要先“腾出”Qi天的背包空间来保证种子可以成熟。在种后一个种子时（即状态转移时），我们要从种完上一个种子后新“开辟”的背包空间中，腾出Q[i]天来让室外幼苗成熟。

T2核心代码实现

```
scanf("%d %d",&n,&t);
for(int i=1;i<=n;i++)
    scanf("%d %d %d",&c[i].p,&c[i].q,&c[i].r);
std::sort(c+1,c+1+n,cmp);//贪心法把q排序
for(int i=1;i<=n;i++)
  for(int j=t-c[i].q;j>=c[i].p;j--)//考虑每个物品时，都先腾出c[i].q空间
  f[j]=max(f[j],f[j-c[i].p]+c[i].r);//滚动数组进行01背包
for(int i=1;i<=t;i++)
ans=max(ans,f[i]);
printf("%d",ans);//输出答案
```

T3算法分析

- 写这题你需要背包的知识。
- 由题意可以联想到01背包。
- 按两座塔的需要加维（很多背包的提高题目都可以通过给数组加维度解决），
- 再在边界（每座塔高度<=1000)
- 和枚举顺序上做一些修改，就可以了。

T3代码实现

```
    for(int i=1;i<=n;++i){
        cin >> a[i];
        sum += a[i];
    }
    memset(f,-INF,sizeof(f));
    f[0][0] = 0;
    for(int i=1;i<=n;++i){
        for(int j=0;j<=sum;++j){
            f[i][j] = max(f[i-1][j],f[i-1][j+a[i]]);
            if(j>=a[i]) f[i][j] = max(f[i][j],f[i-1][j-a[i]]+a[i]);
            else f[i][j] = max(f[i][j],f[i-1][a[i]-j]+j);
        }
    }
    if(f[n][0]>0) printf("%d\n",f[n][0]);
    else printf("Impossible");
```

T4

题目描述

身为房产业界的巨鳄，黄地产不但自己盖房，卖房，同时也买房。说的文艺一些，也就是对业界同行的楼盘进行投资。

然而，当今国内的房产市场楼盘质量良莠不齐，有的像楼倒倒楼歪歪这样的豆腐渣工程，也有的是屹立不倒的黑长直。黄地产通过对市场敏锐的观察和精确的分析，将所有的楼盘分成 n 个等级，第 i 个等级一共有 c_i 个楼盘可供投资，而且同时具有相同的价格 b_i 。也就是说，每个等级内都有一定数量的楼盘，同时价格也是一致的。当然，一个楼盘只能投资一次。

当然，之前说过了，对于黄地产这样的高富帅，钱根本不是问题。可是也正因为这样，所以不能都买那些便宜货，以免被人耻笑。所以，黄地产决定一次性拨出 m 的钱用于投资楼盘，并且要恰好全部花光。可是，每投资一个楼盘就会花费黄地产 1 点的精力值，黄地产可不想太累，所以你能告诉他，把钱花光最少要消耗多少精力值吗？

输入格式

第一行，一个整数 n，表示楼盘一共有 n 个等级；

第二行，n 个整数，第 i 个整数表示第 i 等级的每个楼盘的价格 b_i；

第三行，n 个整数，第 i 个整数表示第 i 等级中的楼盘数量 c_i；

第四行，一个整数 m，表示黄地产拨出来的钱数。

输出格式

一行，仅包含一个整数，表示黄地产把钱花光最少需要消耗的精力值。

样例输入

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

样例输出

```
3
```

数据规模与约定

数据规模：

对于 40% 的数据，有 $1 \leq n \leq 10$;

对于 100% 的数据，有 $1 \leq n \leq 200, 1 \leq m \leq 20000, 1 \leq b_i \leq 20000, 1 \leq c_i \leq 20000$.

黄地产：感觉钱一点也不多吗？？告诉你吧，其实这题所有与钱有关的变量（m 和 b_i）都是以万做单位的！！哈哈屌丝们自卑去吧！！（出题人表示自卑去了……）

T4思路分析

- 多重背包问题。
- 如果把它当成0-1背包来做，枚举每种楼盘购买的数量，复杂度为O(nm(c1+c2+...+cn))，常数写得好可以得到60分。
- 所以我们来优化一下。
- 可以发现，对于任意的十进制数，可以使用一个对应的二进制数来表示。
- 比如10 = 1010(2) = 1*2^3+0*2^2+1*2^1+0*2^0。
- 如果把这个式子看成【购买10套房子】的话，就相当于【购买6套房子与购买4套房子】。
- 所以，我们把n个楼盘拆分成log2n个楼盘，然后对这些拆分后的楼盘进行0-1背包即可，复杂度O(mlog(c1+c2+...+cn))，既mlog2$\sum_{i=1}^{n} ci$，就可以AC了~

代码实现

```
f[0]=0;
for(int i=1;i<=n;i++)
{
    k=1;
    while(c[i]>0)
    {
        if(k>c[i])
        k=c[i];
        c[i]-=k;
        for(int j=m;j>=k*b[i];j--)
        if(f[j]>f[j-k*b[i]]+k)
        f[j]=f[j-k*b[i]]+k;
        k=k*2;

    }
}
```

多重背包拓展应用

P1776 宝物筛选

题目描述

复制Markdown 展开

终于，破解了千年的难题。小 FF 找到了王室的宝物室，里面堆满了无数价值连城的宝物。

这下小 FF 可发财了，嘎嘎。但是这里的宝物实在是太多了，小 FF 的采集车似乎装不下那么多宝物。看来小 FF 只能含泪舍弃其中的一部分宝物了。

小 FF 对洞穴里的宝物进行了整理，他发现每样宝物都有一件或者多件。他粗略估算了下每样宝物的价值，之后开始了宝物筛选工作：小 FF 有一个最大载重为 W 的采集车，洞穴里总共有 n 种宝物，每种宝物的价值为 v_i，重量为 w_i，每种宝物有 m_i 件。小 FF 希望在采集车不超载的前提下，选择一些宝物装进采集车，使得它们的价值和最大。

输入格式

第一行为一个整数 n 和 W，分别表示宝物种数和采集车的最大载重。

接下来 n 行每行三个整数 v_i, w_i, m_i。

输出格式

输出仅一个整数，表示在采集车不超载的情况下收集的宝物的最大价值。

输入输出样例

输入 #1 复制

```
4 20
3 9 3
5 9 1
9 4 2
8 1 3
```

输出 #1 复制

```
47
```

说明/提示

对于 30% 的数据，$n \le \sum m_i \le 10^4$，$0 \le W \le 10^3$。

多重背包二进制拆分

```
for(int i=1;i<=n;i++){
    cin>>v>>w>>m;
    while(k<=m){
        cnt++;
        w1[cnt]=w*k;
        v1[cnt]=v*k;
        m-=k;
        k*=2;
    }
    if(m){
        cnt++;
        w1[cnt]=w*m;
        v1[cnt]=v*m;
    }
    k=1;
}
```

有限件物品，拆分为仅1件的不同物品

- 用01背包

```
23      for(int i=1;i<=cnt;i++){
24          for(int j=W;j>=w1[i];j--){
25              f[j]=max(f[j],f[j-w1[i]]+v1[i]);
26          }
27      }
28      cout<<f[W];
```