307-15-背包的二进制优化

P2851 [USACO06DEC]最少的硬币The Fewest Coins

题目描述

Farmer John has gone to town to buy some farm supplies. Being a very efficient man, he always pays for his goods in such a way that the smallest number of coins changes hands, i.e., the number of coins he uses to pay plus the number of coins he receives in change is minimized. Help him to determine what this minimum number is.

FJ wants to buy T (1 ⩽ T ⩽ 10,000) cents of supplies. The currency system has N (1 ⩽ N ⩽ 100) different coins, with values V1, V2, ..., VN (1 ⩽ Vi ⩽ 120). Farmer John is carrying C1 coins of value V1, C2 coins of value V2, ..., and CN coins of value VN (0 ⩽ Ci ⩽ 10,000). The shopkeeper has an unlimited supply of all the coins, and always makes change in the most efficient manner (although Farmer John must be sure to pay in a way that makes it possible to make the correct change).

农夫John想到镇上买些补给。为了高效地完成任务，他想使硬币的转手次数最少。即使他交付的硬 币数与找零得到的的硬币数最少。

John想要买价值为T的东西。有N(1<=n<=100)种货币参与流通，面值分别为V1,V2..Vn(1<=Vi<=120)。John有Ci个面值为Vi的硬币(0<=Ci<=10000)。

我们假设店主有无限多的硬币， 并总按最优方案找零。**注意无解输出-1。**

输入格式

Line 1: Two space-separated integers: N and T.

Line 2: N space-separated integers, respectively V1, V2, ..., VN coins (V1, ...VN)

Line 3: N space-separated integers, respectively C1, C2, ..., CN

输出格式

Line 1: A line containing a single integer, the minimum number of coins involved in a payment and change-making. If it is impossible for Farmer John to pay and receive exact change, output -1.

输入输出样例

输入 #1 复制

```
3 70
5 25 50
5 2 1
```

输出 #1 复制

```
3
```

说明/提示

Farmer John pays 75 cents using a 50 cents and a 25 cents coin, and receives a 5 cents coin in change, for a total of 3 coins used in the transaction.

题解

这道题其实就是一个背包问题，这个背包分为两部分：

1. 希望付的钱用最少的硬币的多重背包问题
2. 希望找钱找的最少的硬币的完全背包问题

而两个问题结合起来，就是找到付相同的钱两个问题之和的最小值就是答案。

但是在做背包之前会发现一个问题：就是不知道最大的可能金额会是多少，因为时间复杂度是$O(nCV_{\max})$，在这里，可以使用鸽笼原理来证明$V_{\max}=t+\max_i\{V_i\}^2$。

证明：

设货币面值为$V_1\le V_2\le\dots V_n$，$(V_{max}=V_n)$那么当老板需要找V_{max}^2的钱时，至少需要V_{max}枚硬币。设一个前缀和数组为$S_0\ S_1\ S_2\dots S_n\ (S_0=0)$，其中共有$V_{max}+1$个元素。根据抽屉原理，至少有两个前缀和与$V_{max}$同余，则它们的差$=x\times V_{max}$($x$为正整数且$x\ge 1$)。显然当$V_{max}$的数量足够时，将这一部分替换成$x$个面值为$V_{max}$是最优的（不要纠结真正采用的方案）。对于$V_{n-1}$，同理。所以在$V_{max}^2$的范围内一定可以找出最优方案，对于更大的$t+V_{max}^2$也一定可以。

还有一点需要注意的就是这里的部分背包的时间复杂度就算在限制了最大金额之后，肯定还是会超时，所以需要寻找更好地解决方法，即使用二进制优化。

简单的来讲就是使用二进制将所有组合全部考虑到，这不难理解。其实它就等于是打乱了原来方程计算的顺序，但达到了原来计算的效果并且只需要花费$\log n$的时间。

以下为代码：

```cpp
#include <cstdio>
#include <cstring>
```

```cpp
const int maxT = 10000+10;
const int maxn = 100+5;
const int maxv = 120;
int f[maxT + maxv * maxv],g[maxT + maxv * maxv];//f[i]、g[i]分别表示John和店长付i元钱最少需要用的硬币
int v[maxn], c[maxn];//如题意所示
inline int max(int x, int y) { return x > y ? x : y; }
inline int min(int x, int y) { return x < y ? x : y; }
int main()
{
    int n, t;
    scanf("%d%d", &n, &t);
    for (int i = 1; i <= n; i ++)
        scanf("%d", &v[i]);
    int sum = 0,mx = 0;
    for (int i = 1; i <= n; i ++)
    {
        scanf("%d",&c[i]);
        sum += c[i] * v[i];
        mx = max(mx, v[i] * v[i]);
    }
    if (sum < t)//买不起，退了
    {
        printf("-1\n");
        return 0;
    }
    memset(g, 0x3f, sizeof(g));
    memset(f, 0x3f, sizeof(f));
    g[0] = 0;
    f[0] = 0;
    for (int i = 1; i <= n; i ++)
        for (int j = v[i]; j <= mx; j ++)//Rob找j元钱所用的最小钱数
            g[j] = min(g[j], g[j - v[i]] + 1);
    // 实际上应该是g[i][j]=min(g[i-1][j], g[i][j-v[i]+1])
    // g[i][j]表示考虑到第i个物品支付j元的最少硬币数
    // 但是因为第一维存储的信息用不到
    // 且更新前g[i][j]记录的就是g[i-1][j]的信息
    // 所以可以只用一维
    // 其实就是一个完全背包


    for (int i=1;i<=n;i++)
    {
        for (int j=1;j<=c[i];j<<=1)
        {
            for (int k = t + mx; k >= j * v[i]; k --)
            //倒过来更新（实际上是拆分成01背包的形式）
                f[k] = min(f[k], f[k - j * v[i]] + j);
            c[i] -= j;//相当于用拆分的物品进行了一次更新，要从数量中减去
```

```
        }
        if (c[i]) // 还有剩余的没更新
            for (int k = t + mx; k >= c[i] * v[i]; k --)
                f[k] = min(f[k], f[k - c[i] * v[i]] + c[i]);
    }
    // 就是二进制优化的多重背包问题
    int ans = 0x3f3f3f3f;
    for (int i = t; i <= t + mx; i ++)
        ans = min(ans, f[i] + g[i - t]);
    printf("%d\n", ans == 0x3f3f3f3f ? -1 : ans);
    return 0;
}
```