背包九讲问题——超详细

Acwing背包题库

文章目录

一.01背包问题

问题描述

> 有 N 件物品和一个容量是 VV 的背包。每件物品只能使用一次。
> 第 i 件物品的体积是 vi，价值是 wi。
> 求解将哪些物品装入背包，可使这些物品的总体积不超过背包容量，且总价值最大。
> 输出最大价值。

输入格式

> 第一行两个整数，N，V，用空格隔开，分别表示物品数量和背包容积。
> 接下来有 N 行，每行两个整数 vi,wi，用空格隔开，分别表示第 i 件物品的体积和价值。

输出格式

> 输出一个整数，表示最大价值。

数据范围

> 0<N,V≤1000
> 0<vi,wi≤1000

输入样例

> 4 5
> 1 2
> 2 4

> 3 4
> 4 5

输出样例:

> 8

题解:

首先DP问题分为俩个步骤:

1.状态表示:首先考虑用几维的状态表示,然后考虑集合的含义,以及其属性(求Max, Min,数量等)

2.状态计算:DP问题一般都可以将大问题划分为小问题,从小问题下手,从而得到一般的状态转移方程

对于这道题,我们考虑:

> 1.首先声明一个数组**F(i, j)**表示选前 i 件物品,且背包容量为 j 时所能获得的最大价值 。
> 2.对于每个物品我们有拿或者不拿俩种选择:
> (1).j < w[i] 的情况,这时候背包容量不足以放下第 i 件物品,只能选择不拿
> (2).j >= w[i] 的情况,这时背包容量可以放下第 i 件物品,我们就要考虑拿这件物品是否能获取更大的价值。
> 如果拿取,则**F(i, j) = f(i - 1, j - v[i]) + w[i]**,即 **F(i, j)** 表示在上一状态中选了第i件物品,
> 如果不拿,则**F(i, j) = f(i - 1, j)**
> 拿或者不拿,就要看哪种方法得到的价值最大,即
> **F(i, j) = max(f(i - 1, j), f(i - 1, j - v[i]) + w[i])**

代码如下:

```
#include<bits/stdc++.h>
#define maxn 1000
using namespace std;
int v[maxn + 5], w[maxn + 5];
int dp[maxn + 5][maxn + 5];
int main()
{
    int n,W;
    scanf("%d%d",&n,&W);
    for(int i = 0; i < n; i++)
        scanf("%d%d",&v[i],&w[i]);
    for(int i = 0; i < n; i++){
        for(int j = 0; j <= W; j++){
            if(v[i] > j)dp[i + 1][j] = dp[i][j];
            else dp[i + 1][j] = max(dp[i][j], dp[i][j - v[i]] +w[i]);
        }
    }
    printf("%d\n",dp[n][W]);
    return 0;
}

```

01背包问题一维数组实现

状态转移方程如果是由上一层的状态得来的话，枚举体积的时候从大到小枚举，这样我们计算体积的时候，可以保证本层所用到的体积还没有被计算过

如果用的是本层的状态，枚举体积的时候就要从小到大枚举，这样我们计算体积的时候，可以保证所用到的体积是本层之前计算好的体积

```
#include<iostream>
using namespace std;
const int N = 1010;
int d[N];
```

```
int main()
{
    int n, V;
    cin >> n >> V;
    for(int i = 0; i < n; i++){
        int v, w;
        cin >> v >> w;
        for(int j = V; j >= v; j--)
            d[j] = max(d[j], d[j - v] + w);
    }
    cout << d[V] << endl;
    return 0;
}
```

二.01背包问题2

问题描述

> 有n个重量和价值分别为wi，vi的物品。从这些物品中挑选总重量不超过W的物品，求所有挑选方案中价值总和最大的方案

输入样例

> 4 5
> 2 1 3 2
> 3 2 4 2

输出样例

> 7

取值范围

> 1<=n<=100
> 1<=wi<=10^7
> 1<=vi<=100
> 1<=W<=10 ^9

分析:

这里与背包问题1不同的地方是修改了限制的条件，求解这一问题的复杂度是O（NW），对于这一问题的规模来讲就不够用了，相比较重量来说，价值的范围较小一些，所以可以改变DP的对象，背包问题1用DP来表示不同体积下的最大价值，这次我们不妨用DP来表示不同价值下的最小体积。
定义：F(i, j)表示前i个物品挑选出价值总和为j时的最小重量,(不存在是就是一个充分大的数INF)由于前0个物品都挑选不了 所以F(0, 0)=0, F(0, j)=INF

状态转移式为：F(i, j) = min(f(i - 1, j), F(i - 1,j - w[i]) + v[i])

代码如下:

```
#include<stdio.h>
#include<string.h>
#include<algorithm>
#include<iostream>
#define INF 1000000000
#define max_n 100
#define max_v 100
using namespace std;
int n,W;
```

```
int dp[max_n+5][max_n*max_v+5];
int w[max_n+5],v[max_n+5];
void solve()
{
    for(int i=0;i<n;i++){
        for(int j=0;j<=max_n*max_v+5;j++){
            if(j<v[i])dp[i+1][j]=dp[i][j];
            else dp[i+1][j]=min(dp[i][j],dp[i][j-v[i]]+w[i]);
        }
    }
}
int main()
{
    scanf("%d%d",&n,&W);
    for(int i=0;i<n;i++)
        scanf("%d",&w[i]);
    for(int i=0;i<n;i++)
        scanf("%d",&v[i]);
    fill(dp[0],dp[0]+max_n*max_v+5,INF);   //初始化
    dp[0][0]=0;
    solve();
    int res=0;
    for(int i=0;i<=max_n*max_v;i++){
        if(dp[n][i]<=W)res=i;
    }
    printf("%d\n",res);
    return 0;
}
```

三.完全背包问题

> 有 N 种物品和一个容量是 V 的背包，每种物品都有无限件可用。
> 第 i 种物品的体积是 vi，价值是 wi。
> 求解将哪些物品装入背包，可使这些物品的总体积不超过背包容量，且总价值最大。
> 输出最大价值。

输入格式

> 第一行两个整数，N，V，用空格隔开，分别表示物品种数和背包容积。
> 接下来有 N 行，每行两个整数 vi,wi，用空格隔开，分别表示第 i 种物品的体积和价值。

输出格式

> 输出一个整数，表示最大价值。

数据范围

> $0<N,V\le1000$
> $0<v_i,w_i\le1000$

输入样例

> 4 5
> 1 2
> 2 4

> 3 4
> 4 5

输出样例:

> 10

题解:

完全背包问题，物品有无限个，这里我们来考虑第i个物品选多少个

> 1.对于每种物品，我们有选和不选俩种选择，
> 如果不选，**F(i + 1 ,j) = F(i , j)**
> 如果选了，我们还要考虑选多少个，即**F(i + 1, j) = max(F(i + 1, j - v[i]) + w[i], F(i + 1, j))**
> 取俩种情况的最大值，变得到了状态转移方程:
> **F(i + 1, j) = max(F(i , j), F(i + 1, j - v[i]) + w[i])**

代码如下:

```
#include<bits/stdc++.h>
#define maxn 1000
using namespace std;
int v[maxn + 5], w[maxn + 5];
int dp[maxn + 5][maxn + 5];
int main()
{
    int n,V;
    scanf("%d%d",&n,&V);
```

```

    for(int i = 0; i < n; i++)
        scanf("%d%d",&v[i],&w[i]);
    for(int i = 0; i < n; i++){
        for(int j = 0; j <= V; j++){
            if(v[i] > j)dp[i + 1][j] = dp[i][j];
            else dp[i + 1][j] = max(dp[i][j], dp[i + 1][j - v[i]] + w[i]);
        }
    }
    printf("%d\n",dp[n][V]);
    return 0;
}
```

一维数组实现

上面我们说到：如果用的是本层的状态，枚举体积的时候就要从小到大枚举即可，这样我们计算体积的时候，可以保证所用到的体积是本层之前计算好的体积

代码如下：

```
#include<iostream>
using namespace std;
const int N = 1010;
int d[N];
int main()
{
    int n,V;
    cin >> n >> V;
    for(int i = 0; i < n; i++){
        int v, w;
```

```
        cin >> v >> w;
        for(int j = v; j <= V; j++)
            d[j] = max(d[j], d[j - v] + w);
    }
    cout << d[V] << endl;
    return 0;
}
```

四.多重背包问题 I

题目描述

> 有 N 种物品和一个容量是 V 的背包。
> 第 i 种物品最多有 si 件，每件体积是 vi，价值是 wi。
> 求解将哪些物品装入背包，可使物品体积总和不超过背包容量，且价值总和最大。
> 输出最大价值。

输入格式

> 第一行两个整数，N，V，用空格隔开，分别表示物品种数和背包容积。
> 接下来有 N 行，每行三个整数 vi,wi,si，用空格隔开，分别表示第 i 种物品的体积、价值和数量。

输出格式

> 输出一个整数，表示最大价值。

数据范围

> $0<N,V\le 100$
> $0<v_i,w_i,s_i\le 100$

输入样例

> 4 5
> 1 2 3
> 2 4 1
> 3 4 3
> 4 5 2

输出样例：

> 10

题解：

当成01背包问题来做即可，在枚举体积的时候在枚举一下该物品个数

状态转移方程： $F(i, j) = \max(F(i-1, j), F(i-1, j-k*v[i]) + k*w[i])$

一维数组实现，由于该状态转移用的是上一层的状态，所以枚举体积的时候，我们从大到小枚举，代码如下：

```
#include<iostream>
using namespace std;
const int N = 110;
```

```
int d[N];
int main()
{
    int n, V;
    cin >> n >> V;
    for(int i = 0; i < n; i++){
        int v, w, s;
        cin >> v >> w >> s;
        for(int j = V; j >= v; j--)
            for(int k = 1; k <= s && k * v <= j; k++)
                d[j] = max(d[j], d[j - k * v] + k * w);
    }
    cout << d[V] << endl;
    return 0;
}
```

五.多重背包问题II

题目描述

> 有 N 种物品和一个容量是 V 的背包
> 第 i 种物品最多有 si 件，每件体积是 vi，价值是 wi。
> 求解将哪些物品装入背包，可使物品体积总和不超过背包容量，且价值总和最大。
> 输出最大价值。

输入格式

> 第一行两个整数，N，V，用空格隔开，分别表示物品种数和背包容量。

> 接下来有 N 行，每行三个整数 vi,wi,si，用空格隔开，分别表示第 i 种物品的体积、价值和数量。

输出格式

> 输出一个整数，表示最大价值。

数据范围

> 0<N≤1000
> 0<V≤2000
> 0<vi,wi,si≤2000

提示：

本题考查多重背包的二进制优化方法。

输入样例

> 4 5
> 1 2 3
> 2 4 1
> 3 4 3
> 4 5 2

输出样例：

> 10

题解：

由于这题的数据范围有点大，直接暴力枚举会超时，那么我们就要想一个可以优化的方法，这里主要是对物品的个数进行拆分，将其变为01背包问题

二进制拆分法： 我们知道，从2^0, 2^1, $2^{2 \dots 2}$(k - 1)这k个数中选出任意个相加可以表示出0~2^k之间任何整数，所以我们可以对每一种物品就行二进制拆分，将其转化为01背包问题

代码如下：

```
#include<iostream>
using namespace std;
const int N = 1e6 + 9;
int a[N], b[N], d[N];
int main()
{
    int k = 0;
    int n, V;
    cin >> n >> V;
    for(int i = 0; i < n; i++){
        int v, w, s;
        scanf("%d%d%d", &v, &w, &s);
        for(int j = 1; j <= s; j <<= 1){//二进制拆分
            a[k] = j * v;    //用a数组来存体积
            b[k++] = j * w;  //b数组来存价值
            s -= j;
        }
        if(s > 0){
            a[k] = s * v;
```

```
                b[k++] = s * w;
            }
        }
        for(int i = 0; i < k; i++)//01背包
            for(int j = V; j >= a[i]; j--)
                d[j] = max(d[j], d[j - a[i]] + b[i]);
        cout << d[V] << endl;
        return 0;
}
```

六.分组背包问题

题目描述

> 有 N 组物品和一个容量是 V 的背包。
> 每组物品有若干个，同一组内的物品最多只能选一个。
> 每件物品的体积是 vij，价值是 wij，其中 i 是组号，j 是组内编号。
> 求解将哪些物品装入背包，可使物品总体积不超过背包容量，且总价值最大。
> 输出最大价值。

输入格式

> 第一行有两个整数 N，V，用空格隔开，分别表示物品组数和背包容量。
> 接下来有 N 组数据：
> 每组数据第一行有一个整数 Si，表示第 i 个物品组的物品数量；
> 每组数据接下来有 Si 行，每行有两个整数 vij,wij，用空格隔开，分别表示第 i 个物品组的第 j 个物品的体积和价值；

输出格式

> 输出一个整数，表示最大价值。

数据范围

> 0<N,V≤100
> 0<Si≤100
> 0<vij,wij≤100

输入样例

> 3 5
> 2
> 1 2
> 2 4
> 1
> 3 4
> 1
> 4 5

输出样例

> 8

题解：

跟完全背包问题类似

用F(i, j)来表示选前i组物品且体积为j时的价值最大值，我们先枚举每一组，由于F(i, j)的状态用的是上一层的状态，所以我们枚举体积的时候从大到小来枚举，再依次枚举每一组里的物品，找到体积为j时，选取i组中哪个物品的价值最大

代码如下：

```
#include<iostream>
using namespace std;
const int N = 110;
int v[N][N], w[N][N];
int d[N], s[N];
int main()
{
    int n, V;
    cin >> n >> V;
    for(int i = 0; i < n; i++){
        cin >> s[i];
        for(int j = 0; j < s[i]; j++){
            cin >> v[i][j] >> w[i][j];
        }
    }

    for(int i = 0; i < n; i++)  //枚举每一组
        for(int j = V; j >= 0; j--)  //枚举体积
            for(int k = 0; k < s[i]; k++)//枚举第i组体积为j时，选取哪个物品价值最大
                if(v[i][k] <= j)
                    d[j] = max(d[j], d[j - v[i][k]] + w[i][k]);
    cout << d[V] << endl;

```

```
    return 0;
}
```

七.背包问题求方案数

有 N 件物品和一个容量是 V 的背包。每件物品只能使用一次。

第 i 件物品的体积是 vi，价值是 wi。

求解将哪些物品装入背包，可使这些物品的总体积不超过背包容量，且总价值最大。

输出 **最优选法的方案数**。注意答案可能很大，请输出答案模 109+7 的结果。

输入格式

第一行两个整数，N，V，用空格隔开，分别表示物品数量和背包容积。

接下来有 N 行，每行两个整数 vi,wi，用空格隔开，分别表示第 i 件物品的体积和价值。

输出格式

输出一个整数，表示 **方案数** 模 109+7 的结果。

数据范围

0<N,V≤1000
0<vi,wi≤1000

输入样例

```
4 5
1 2
2 4
3 4
4 6
```

输出样例：

```
2
```

题解：在01背包问题的基础上，添加一个num数组用来记录方案数即可

代码如下：

```
#include<bits/stdc++.h>
using namespace std;
const int N = 1e3 + 10, mod = 1000000007;
int w[N], v[N], dp[N], num[N];
int main()
{
    int n, V;
    scanf("%d%d", &n, &V);
    for(int i = 0; i < n; i++){
        scanf("%d%d", &v[i], &w[i]);
        num[i] = 1;
    }
    for(int i = 0; i < n; i++){
        for(int j = V; j >= v[i]; j--){
            if(dp[j] < dp[j - v[i]] + w[i]){//更新最大价值
```

```
                dp[j] = dp[j - v[i]] + w[i];
                num[j] = num[j - v[i]] % mod;//num数组记录更新方案数
            }
            else if(dp[j] == dp[j - v[i]] + w[i]){//如果相等
                num[j] = (num[j] + num[j - v[i]]) % mod;//方案数相加
            }
        }
    }
    cout << num[V] << endl;
    return 0;
}
```

八. 背包问题求具体方案

有 N 件物品和一个容量是 V 的背包。每件物品只能使用一次。

第 i 件物品的体积是 vi，价值是 wi。

求解将哪些物品装入背包，可使这些物品的总体积不超过背包容量，且总价值最大。

输出 **字典序最小的方案**。这里的字典序是指：所选物品的编号所构成的序列。物品的编号范围是 1…N。

输入格式

第一行两个整数，N，V，用空格隔开，分别表示物品数量和背包容积。

接下来有 N 行，每行两个整数 vi,wi，用空格隔开，分别表示第 i 件物品的体积和价值。

输出格式

输出一行，包含若干个用空格隔开的整数，表示最优解中所选物品的编号序列，且该编号序列的字典序最小。

物品编号范围是 1…N。

数据范围

> 0<N,V≤1000
> 0<vi,wi≤1000

输入样例

> 4 5
> 1 2
> 2 4
> 3 4
> 4 6

输出样例：

> 1 4

```
#include<iostream>
using namespace std;
const int N = 1010;
int v[N], w[N], d[N][N], ans[N];
int n, m;
int main()
{
```

```
    cin >> n >> m;
    for(int i = 1; i <= n; i++)
        cin >> v[i] >> w[i];
    for(int i = n; i >= 1; i--){
        for(int j = 0; j <= m; j++){
            d[i][j] = d[i + 1][j];
            if(j >= v[i])
                d[i][j] = max(d[i][j], d[i + 1][j - v[i]] + w[i]);
        }
    }
    int j = m;
    for(int i = 1; i <= n; i++){
        if(j >= v[i] && d[i][j] == d[i + 1][j - v[i]] + w[i]){
            cout << i << ' ';
            j -= v[i];
        }
    }
    return 0;
}
```

文章知识点与官方知识档案匹配，可进一步学习相关知识