01背包问题

问题分析

求物品的总价值

- 采用动态规划的思想，维护一个二维数组dp[i][j]
- 该数组表示为当有前i个物品选择且背包容量为j时最优价值

  例如，

  物品的价值：8,50,12,20,10

  物品的重量：3,20,5,10,5

  如`dp[1][1]`表示物品的选择范围只有第一个，且背包现在的容量为1，那么最大价值为0

  如`dp[1][3]` 表示物品的选择范围只有第一个，且背包现在的容量为3，那么最大价值为8

  如`dp[2][20]` 表示物品的选择范围有第一和第二个，且背包现在的容量为20，那么最大价值为50
- 原问题的解即为dp[i][j]数组中的最大值即为`dp[n][c]` （n为物品的数量，c为背包的重量)
- 最长连续上升子序列可以通过s[i]中的下标以及该元素中的值决定

  ## 动态规划的原理

  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**

  ## 背包问题的解决过程

  在解决问题之前，为描述方便，首先定义一些变量：**Vi表示第 i 个物品的价值，Wi表示第 i 个物品的体积，定义V(i,j)：当前背包容量 j，前 i 个物品最佳组合对应的价值，**同时背包问题抽象化（X1，X2，…，Xn，其中 Xi 取0或1，表示第 i 个物品选或不选）。

  1、建立模型，即求max(V1X1+V2X2+…+VnXn)；

  2、寻找约束条件，W1X1+W2X2+…+WnXn<capacity；

  3、**寻找递推关系式**，面对当前商品有两种可能性：

  - **包的容量比该商品体积小，装不下，此时的价值与前i-1个的价值是一样的，即V(i,j)=V(i-1,j)；**
  - **还有足够的容量可以装该商品，但装了也不一定达到当前最优价值，所以在装与不装之间选择最优的一个，即V(i,j)=max｛V(i-1,j)，V(i-1,j-w(i))+v(i)｝。**

  其中V(i-1,j)表示不装，V(i-1,j-w(i))+v(i) 表示装了第i个商品，背包容量减少w(i)，但价值增加了v(i)；

  由此可以得出递推关系式：

  - j<w(i)　V(i,j)=V(i-1,j)
  - j>=w(i)　V(i,j)=max｛V(i-1,j)，V(i-1,j-w(i))+v(i)｝
  - ## 背包问题最优解回溯

通过上面的方法可以求出背包问题的最优解，但还不知道这个最优解由哪些商品组成，故要根据最优解回溯找出解的组成，根据填表的原理可以有如下的寻解方式：

- **V(i,j)=V(i-1,j)时，说明没有选择第i 个商品，则回到V(i-1,j)；**
- **V(i,j)=V(i-1,j-w(i))+v(i)时，说明装了第i个商品，该商品是最优解组成的一部分，随后我们得回到装该商品之前，即回到V(i-1,j-w(i))；**
- **一直遍历到i = 0结束为止，所有解的组成都会找到。**

就拿上面的例子来说吧：

- 最优解为V(4,8)=10，而V(4,8)!=V(3,8)却有V(4,8)=V(3,8-w(4))+v(4)=V(3,3)+6=4+6=10，所以第4件商品被选中，并且回到V(3,8-w(4))=V(3,3)；
- 有V(3,3)=V(2,3)=4，所以第3件商品没被选择，回到V(2,3)；
- 而V(2,3)!=V(1,3)却有V(2,3)=V(1,3-w(2))+v(2)=V(1,0)+4=0+4=4，所以第2件商品被选中，并且回到V(1,3-w(2))=V(1,0)；
- 有V(1,0)=V(0,0)=0，所以第1件商品没被选择。

```cpp
#include <algorithm>

using namespace std;

const int N = 1010;

int n, m;
int f[N][N];
int v[N], w[N];

int main()
{
    cin >> n >> m;
    for (int i = 1; i <= n; i ++ ) cin >> v[i] >> w[i];

    for (int i = 1; i <= n; i ++ )
        for (int j = 0; j <= m; j ++ )
        {
            f[i][j] = f[i - 1][j];
            if (j >= v[i])
                f[i][j] = max(f[i][j], f[i - 1][j - v[i]] + w[i]);
        }

    int res = 0;
    for (int i = 0; i <= m; i ++ ) res = max(res, f[n][i]);

    cout << res << endl;
    return 0;
}
```

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 v_i，价值是 w_i。

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 v_i, w_i，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

$0 < N, V \le 1000$
$0 < v_i, w_i \le 1000$

输入样例

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

输出样例:

```
8
```

时/空限制:

总通过数:

总尝试数:

来源:

算法标签

思路:

二维数组+动规

状态转移方程:
定义 `f[i][j]` :前i个物品，背包容量j下的最优解

1）当前背包容量不够（`j < w[i]`），为前i-1个物品最优解：`f[i][j] = f[i-1][j]`
2）当前背包容量够，判断选与不选第i个物品
选：`f[i][j] = f[i-1][j-w[i]] + v[i]`
不选：`f[i][j] = f[i-1][j]`

```
`#include<bits/stdc++.h>

using namespace std;

const int MAXN = 1005;
int w[MAXN];    // 重量
int v[MAXN];    // 价值
int f[MAXN][MAXN];  // f[i][j], j重量下前i个物品的最大价值

int main()
{
    int n, m;
    cin >> n >> m;
    for(int i = 1; i <= n; ++i)
        cin >> w[i] >> v[i];
```

```
for(int i = 1; i <= n; ++i)
    for(int j = 1; j <= m; ++j)
    {
        //  当前重量装不进，价值等于前i-1个物品
        if(j < w[i])
            f[i][j] = f[i-1][j];
        // 能装，需判断
        else
            f[i][j] = max(f[i-1][j], f[i-1][j-w[i]] + v[i]);
    }

cout << f[n][m];
return 0;
```

}

作者：深蓝
链接：https://www.acwing.com/solution/content/1374/
来源：AcWing

简化

```
#include <iostream>
#include <cstring>
#include <algorithm>

using namespace std;

const int N = 1010;

int n, m;
int f[N];
int v[N], w[N];

int main()
{
    cin >> n >> m;
    for (int i = 1; i <= n; i ++ ) cin >> v[i] >> w[i];

    for (int i = 1; i <= n; i ++ )
        for (int j = m; j >= v[i]; j -- )
            f[j] = max(f[j], f[j - v[i]] + w[i]);

    cout << f[m] << endl;
    return 0;
}
```

优化

(动规+一维数组)

状态转移方程为：`f[j] = max(f[j], f[j-w[i]] + v[i]`

```
for(int i = 1; i <= n; ++i)
{
    for(int j = m; j >= 0; --j)
        if(j >= w[i])
```

```
            f[j] = max(f[j], f[j-w[i]] + v[i]);
}
```