组装最大可靠性设备

题目描述：

一个设备由 N 种类型元器件组成(每种类型元器件只需要一个，类型 type 编号从 0~N-1)，每个元器件均有可靠性属性 reliability，**可靠性越高的器件其价格 price 越贵**。而设备的可靠性由组成设备的所有器件中可靠性最低的器件决定。

给定预算 S，购买 N 种元器件（**每种类型元器件都需要购买一个**），在不超过预算的情况下，请给出能够组成的设备的最大可靠性。

输入描述：

S N // S 总的预算，N 元器件的种类

total // 元器件的总数，每种型号的元器件可以有多种；此后有 total 行具体器件的数据

type reliability price // type 整数类型，代表元器件的类型编号从 0 ~ N-1；reliability 整数类型，代表元器件的可靠性；price 整数类型，代表元器件的价格

输出描述：

符合预算的设备的最大可靠性，如果预算无法买齐 N 种器件，则返回 -1

补充说明：

0 <= S,price <= 10000000;

0 <= N <= 100;

0 <= type <= N-1;

0 <= total <= 100000;

0 < reliability <= 100000;

示例 1

输入：

```
500 3
```

```
6

0 80 100

0 90 200

1 50 50

1 70 210

2 50 100

2 60 150
```

输出：

```
60
```

说明：

预算 500，设备需要 3 种元件组成，方案 类型 0 的第一个(可靠性 80)，类型 1 的第二个(可靠性 70)，类型 2 的第二个(可靠性 60) 可以使设备的可靠性最大 60

示例 2

输入：

```
100 1
1
0 90 200
```

输出：

```
-1
```

说明：

组成设备需要 1 个元件，但是元件价格大于预算，因此无法组成设备，返回 -1

```
import sys
from collections import defaultdict

def read_data():
    S, N = list(map(int, sys.stdin.readline().strip().split()))
    total = int(sys.stdin.readline().strip())
    types = []
    reliabilities = []
```

```
    prices = []

    for _ in range(total):
        t, r, p = list(map(int, sys.stdin.readline().strip().split()))
        types.append(t)
        reliabilities.append(r)
        prices.append(p)

    return S, N, types, reliabilities, prices

def solve(input_data):
    S, N, types, reliabilities, prices = input_data
    # dp[i][j]  使用前 i 种元器件填满 j 的背包的最大可靠性
    # dp[i][j] = max(min(dp[i-1][j-prices[k]], reliabilies[i-1]) k from 0 to K

    data = defaultdict(list)
    for i in range(len(types)):
        data[types[i]].append([reliabilities[i], prices[i]])

#       print(data)
    dp = [[0 for _ in range(S+1)] for _ in range(N+1)]

    for j in range(S+1):
        for r, p in data[0]:
            if p <= j:
                dp[1][j] = max(dp[1][j], r)

    data = list(data.values())
    for i in range(2, N+1):
        for j in range(1, S+1):
            dp[i][j] = 0
            for r, p in data[i-1]:
                if j > p and dp[i-1][j-p] != 0    and min(dp[i-1][j-p], r) > dp[i][j]:
#                     dp[i][j] = max(min(dp[i-1][j-p], r), dp[i][j])
                    dp[i][j] = min(dp[i-1][j-p], r)
#       print(dp[3])
    if dp[-1][-1] == 0:
        print(-1)
        return
    print(dp[-1][-1])


# solve(read_data())
```

```
def solve2(input_data):
    S, N, types, reliabilities, prices = input_data
    # dp[i][j]  使用前 i 种元器件填满 j 的背包的最大可靠性
    # dp[i][j] = max(min(dp[i-1][j-prices[k]], reliabilies[i-1]) k from 0 to K

    data = defaultdict(list)
    for i in range(len(types)):
        data[types[i]].append([reliabilities[i], prices[i]])

    data = list(data.values())
#     print(data)
    dp = []
    tmp = {}
    for r, p in data[0]:
        if S-p >=0:
            tmp[r] = S-p
    if not tmp:
        print(-1)
        return
    dp.append(tmp)

    for i in range(1, N):
        cur_data = {}
        for last_r, left_p in dp[-1].items():
            for cur_r, cur_p in data[i]:
                if cur_p <= left_p:
                    cur_r = min(cur_r, last_r)
                    if cur_r in cur_data:
                        cur_data[cur_r] = max(cur_data[cur_r], left_p-cur_p)
                    else:
                        cur_data[cur_r] = left_p-cur_p
        dp.append(cur_data)

#     print(dp)
    ans = list(dp[-1].keys())
    if not dp[-1] or max(ans) == 0:
        print(-1)
    else:
        print(max(ans))

solve2(read_data())
```