题目描述

潜水员为了潜水要使用特殊的装备。他有一个带2种气体的气缸：一个为氧气，一个为氮气。让潜水员下潜的深度需要各种的数量的氧和氮。潜水员有一定数量的气缸。每个气缸都有重量和气体容量。潜水员为了完成他的工作需要特定数量的氧和氮。他完成工作所需气缸的总重的最低限度的是多少？

例如：潜水员有5个气缸。每行三个数字为：氧，氮的（升）量和气缸的重量：

3 36 120

10 25 129

5 50 250

1 45 130

4 20 119

如果潜水员需要5升的氧和60升的氮则总重最小为249（1，2或者4，5号气缸）。你的任务就是计算潜水员为了完成他的工作需要的气缸的重量的最低值。

输入

第一行有2整数m，n（1≤m≤21，1≤n≤79）。它们表示氧，氮各自需要的量。

第二行为整数k（1≤n≤1000）表示气缸的个数。

此后的k行，每行包括ai，bi，ci（1≤ai≤21，1≤bi≤79，1≤ci≤800）3个整数。这些各自是：第i个气缸里的氧和氮的容量及汽缸重量。

输出

仅一行包含一个整数，为潜水员完成工作所需的气缸的重量总和的最低值。

输入样例

```
5 60
5
3 36 120
10 25 129
5 50 250
1 45 130
4 20 119
```

输出样例

```
249
```

解析

最大值

在算法竞赛中，我们常常需要用到设置一个常量用来代表“无穷大”。

比如对于int类型的数，有的人会采用INT_MAX，即0x7fffffff作为无穷大。但是以INT_MAX为无穷大常常面临一个问题，即加一个其他的数会溢出。

而这种情况在动态规划，或者其他一些递推的算法中常常出现，很有可能导致算法出问题。

所以在算法竞赛中，我们常采用0x3f3f3f3f来作为无穷大。0x3f3f3f3f主要有如下好处：

0x3f3f3f3f的十进制为1061109567，和INT_MAX一个数量级，即10^9数量级，而一般场合下的数据都是小于10^9的。

0x3f3f3f3f * 2 = 2122219134，无穷大相加依然不会溢出。

可以使用memset(array, 0x3f, sizeof(array))来为数组设初值为0x3f3f3f3f，因为这个数的每个字节都是0x3f。

潜水员分析

因此，我们定义数组dp[i][j][k]来代表前i个气缸提供j升氧气和k升氮气的最小重量。

因为本题是求最小重量，因此初始化的时刻需要将数组内部的数据设置为最大，即设置为0x3f。需要注意的是，dp[0][0][0]处的值必须设置为0，可以理解为：不需要任何氧气和氮气的时刻，最小重量为0。

重点：任何算法都要弄清楚0状态时的初始值，千万不要弄错。

虽然本题求解的是最小值，但是并不妨碍我们使用01背包的计算方法来求解这道题，只是在计算过程中多了一个维度而已，同时去比较最小值即可。

具体状态分析：

1、所需要的氧气和氮气小于当前气缸能够提供的数量：直接选择重量最小的那个。

2、所需要的氧气和氮气大于当前气缸能够提供的数量：直接使用01背包的计算，注意是二维计算！

3、其中一种气体小于当前气缸能够提供的数量：将溢出的那一维度减到0就可以了。

朴素算法

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

using namespace std;

const int N = 110;
```

```
//第一维：为瓶子的索引
//第二维：氧气限制
//第三维：氮气限制
//前i个气缸提供j升氧气和k升氮气的最小重量
int f[1001][N][N];

int main() {
    //将所有数据设置为最大
    memset(f, 0x3f, sizeof f);
    int n, m, k;  //所需的氧气和氮气的数量，以及氧气瓶的数量
    cin >> n >> m >> k;
    //什么都不装入的时候，重量必须为0
    f[0][0][0] = 0;
    //模拟01背包，开始装入气缸
    //遍历所有的气缸
    for (int i = 1; i <= k; i++) {
        //当前瓶子的氧气含量、氮气含量、重量
        int co, cn, w;
        //输入氧气、氮气、瓶子重量
        cin >> co >> cn >> w;
        //遍历氧气
        for (int j = 0; j <= n; j++) {
            //遍历氮气
            for (int k = 0; k <= m; k++) {
                //如果瓶中所装载的氧气和氮气都超过了所需要的量时，
                //则需要选择一个重量最小的
                if (j < co && k < cn) {
                    f[i][j][k] = min(f[i - 1][j][k], w);
                }
                //氧气过量时，比较氮气。因为氮气过量，
                //因此直接回退到氧气为0的状态进行比较
                else if (j < co) {
                    f[i][j][k] = min(f[i - 1][j][k],
                                     f[i - 1][0][k - cn] + w);
                }
                //氮气过量时，比较氧气。因为氮气过量，
                //因此直接回退到氮气为0的状态进行比较
                else if (k < cn) {
                    f[i][j][k] = min(f[i - 1][j][k],
                                     f[i - 1][j - co][0] + w);

                }
                //如果瓶中所装载的氧气和氮气都未达到所需要的量时，
                //则按照01背包的计算方式进行累加和比较
                else {
                    f[i][j][k] = min(f[i - 1][j][k],
                                     f[i - 1][j - co][k - cn] + w);
                }
            }
```

```
            }
        }
        //最小值依然是在最后一个格子
        cout << f[k][n][m] << endl;

        return 0;
    }
```

降维优化

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

    using namespace std;

    const int N = 110;
    //第一维：氧气限制
    //第二维：氮气限制
    int f[N][N];

    int main() {
        memset(f, 0x3f, sizeof f);
        int n, m, k;  //所需的氧气和氮气的数量，以及氧气瓶的数量
        cin >> n >> m >> k;
        f[0][0] = 0;
        for (int i = 1; i <= k; i++) {
            //当前瓶子的氧气含量、氮气含量、重量
           int co, cn, w;
            cin >> co >> cn >> w;
            for (int j = n; j >= 0; j--) {
                for (int k = m; k >= 0; k--) {
                    //如果瓶中所装载的氧气和氮气都超过了所需要的量时，
                 //则需要选择一个重量最小的
                 if (j < co && k < cn) {
                        f[j][k] = min(f[j][k], w);
                    }
                        //氧气过量时，比较氮气
                 else if (j < co) {
                        f[j][k] = min(f[j][k], f[0][k - cn] + w);
                    }
                        //氮气过量时，比较氧气
                 else if (k < cn) {
                        f[j][k] = min(f[j][k], f[j - co][0] + w);
                    }
                        //如果瓶中所装载的氧气和氮气都未达到所需要的量时，
                     //则按照01背包的计算方式进行累加和比较
                 else {
                        f[j][k] = min(f[j][k], f[j - co][k - cn] + w);
                    }
```

```
                }
            }
        }
        cout << f[n][m] << endl;

        return 0;
}
```