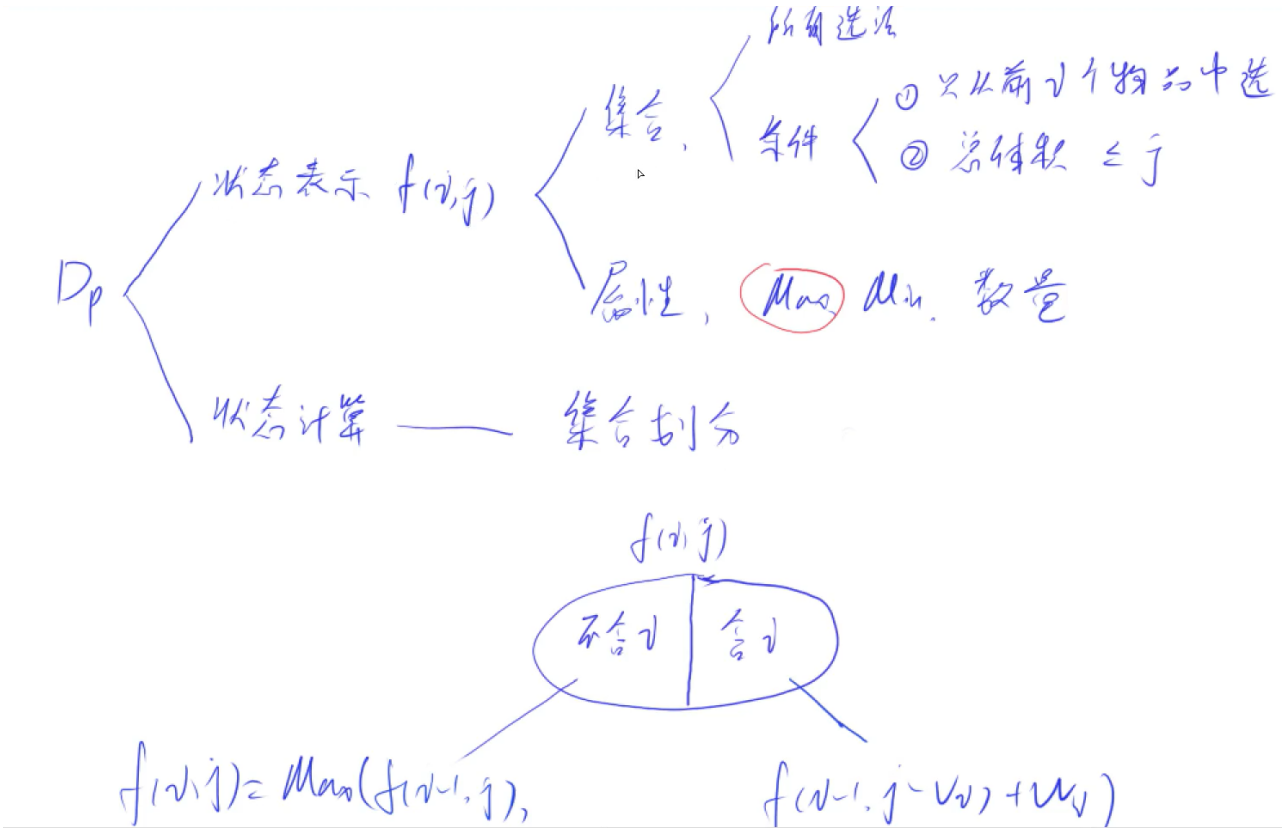
1：背包问题：

1）：0 - 1 背包问题：（特点：每件物品只能用一次）

闫氏DP分析法：



例题：

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。

输入样例

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

输出样例:

```
8
```

版本一：二维：

（1）状态f [i][j]定义：前 i 个物品，背包容量 j 下的最优解（最大价值）：

当前的状态依赖于之前的状态，可以理解为从初始状态 f [0][0] = 0开始决策，有 N 件物品，则需要 N 次决策，每一次对第 i 件物品的决策，状态 f [i][j]不断由之前的状态更新而来

（2）当前背包容量不够（ j < v [i]），没有可以选的方案，因此前 i 个物品最优解即为前 i−1 个物品最优解：对应代码：f [i][j] = f [i - 1][j]

（3）当前背包容量够，可以选，因此需要决策选与不选第 i个物品：

选：f [i][j] = f [i - 1] [j - v [i]] + w [i]

不选：f [i][j] = f [i - 1][j]

决策是如何取到最大价值，因此以上两种情况取 max( )

```java
import java.util.*;
public class Main{
    public static int N = 1010;
    public static int[] v = new int[N];
    public static int[] w = new int[N];
    public static int[][] f = new int[N][N];
    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int m = sc.nextInt();
        for(int i = 1; i <= n; i++){
            v[i] = sc.nextInt();
            w[i] = sc.nextInt();
        }
        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] = Math.max(f[i][j], f[i - 1][j - v[i]] + w[i]);
                }
            }
        }
        System.out.println(f[n][m]);
    }
}
```

版本二：一维：

将状态 f [i][j]优化到一维 f [j]，实际上只需要做一个等价变形

为什么可以这样变形呢？我们定义的状态 f [i][j]可以求得任意合法的i与j最优解，但题目只需要求得最终状态f [n][m]，因此我们只需要一维的空间来更新状态

（1）状态 f [j]定义：N 件物品，背包容量 j 下的最优解

（2）注意枚举背包容量 j 必须从m开始

（3）为什么一维情况下枚举背包容量需要逆序？在二维情况下，状态 f [i][j]是由上一轮i - 1的状态得来的，f[i][j]与 f [i - 1][j]是独立的。而优化到一维后，如果我们还是正序，则有 f [较小体积] 更新到 f [较大体积]，则有可能本应该用第 i-1 轮的状态却用的是第 i 轮的状态

举个例子：

假设有3件物品，背包的总体积为10

```
物品     体积      价值
i = 1     4        5
i = 2     5        6
i = 3     6        7
```

因为 f [0] [j] 总共0件物品，所以最大价值为 0， 即 f [0][j] == 0 成立

```
如果 j 层循环是递增的：
    for (int i = 1; i <= n; i++) {
        for (int j = v[i]; j <= m; j++) {
            f[j] = max(f[j], f[j - v[i]] + w[i]);
        }
    }
```

```
当还未进入循环时：
    f[0] = 0;  f[1] = 0;  f[2] = 0;  f[3] = 0;  f[4] = 0;
    f[5] = 0;  f[6] = 0;  f[7] = 0;  f[8] = 0;  f[9] = 0; f[10] = 0;
    当进入循环 i == 1 时：
    f[4] = max(f[4], f[0] + 5); 即max(0, 5) = 5; 即f[4] = 5;
    f[5] = max(f[5], f[1] + 5); 即max(0, 5) = 5; 即f[5] = 5;
    f[6] = max(f[6], f[2] + 5); 即max(0, 5) = 5; 即f[6] = 5;
    f[7] = max(f[7], f[3] + 5); 即max(0, 5) = 5; 即f[7] = 5;
    重点来了！！！
    f[8] = max(f[8], f[4] + 5); 即max(0, 5 + 5) = 10; 即f[8] = 10;
    这里就已经出错了
    因为此时处于 i == 1 这一层，即物品只有一件，不存在单件物品满足价值为10
    所以已经出错了
```

```
如果 j 层循环是逆序的：
    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]);
        }
    }
```

```
当还未进入循环时：
    f[0] = 0;  f[1] = 0;  f[2] = 0;  f[3] = 0;  f[4] = 0;
    f[5] = 0;  f[6] = 0;  f[7] = 0;  f[8] = 0;  f[9] = 0; f[10] = 0;
    当进入循环 i == 1 时：w[i] = 5; v[i] = 4;
    j = 10：f[10] = max(f[10], f[6] + 5); 即max(0, 5) = 5; 即f[10] = 5;
    j = 9 ：f[9] = max(f[9], f[5] + 5); 即max(0, 5) = 5; 即f[9] = 5;
    j = 8 ：f[8] = max(f[8], f[4] + 5); 即max(0, 5) = 5; 即f[8] = 5;
    j = 7 ：f[7] = max(f[7], f[3] + 5); 即max(0, 5) = 5; 即f[7] = 5;
    j = 6 ：f[6] = max(f[6], f[2] + 5); 即max(0, 5) = 5; 即f[6] = 5;
    j = 5 ：f[5] = max(f[5], f[1] + 5); 即max(0, 5) = 5; 即f[5] = 5;
    j = 4 ：f[6] = max(f[4], f[0] + 5); 即max(0, 5) = 5; 即f[4] = 5;
    当进入循环 i == 2 时：w[i] = 6; v[i] = 5;
    j = 10：f[10] = max(f[10], f[5] + 6); 即max(5, 11) = 11; 即f[10] = 11;
    j = 9 ：f[9] = max(f[9], f[4] + 6); 即max(5, 11) = 5; 即f[9] = 11;
    j = 8 ：f[8] = max(f[8], f[3] + 6); 即max(5, 6) = 6; 即f[8] = 6;
    j = 7 ：f[7] = max(f[7], f[2] + 6); 即max(5, 6) = 6; 即f[7] = 6;
```

```
    j = 6 : f[6] = max(f[6], f[1] + 6); 即max(5, 6) = 6; 即f[6] = 6;
    j = 5 : f[5] = max(f[5], f[0] + 6); 即max(5, 6) = 6; 即f[5] = 6;
    当进入循环 i == 3 时: w[i] = 7; v[i] = 6;
    j = 10: f[10] = max(f[10], f[4] + 7); 即max(11, 12) = 12; 即f[10] = 12;
    j = 9 : f[9] = max(f[9], f[3] + 6); 即max(11, 6) = 11; 即f[9] = 11;
    j = 8 : f[8] = max(f[8], f[2] + 6); 即max(6, 6) = 6; 即f[8] = 6;
    j = 7 : f[7] = max(f[7], f[1] + 6); 即max(6, 6) = 6; 即f[7] = 6;
    j = 6 : f[6] = max(f[6], f[0] + 6); 即max(6, 6) = 6; 即f[6] = 6;
```

```java
import java.util.*;
public class Main{
    public static int N = 1010;
    public static int[] v = new int[N];
    public static int[] w = new int[N];
    public static int[] f = new int[N];
    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int m = sc.nextInt();
        for(int i = 1; i <= n; i++){
            v[i] = sc.nextInt();
            w[i] = sc.nextInt();
        }
        for(int i = 1; i <= n; i++){
            for(int j = m; j >= v[i]; j--){
                f[j] = Math.max(f[j], f[j - v[i]] + w[i]);
            }
        }
        System.out.println(f[m]);
    }
}
```

2）：完全背包问题：（特点：每种物品可以无限使用）

基本思路：

集合的划分依据：第 i 种物品可以使用多少个

例题：

版本一：二维：

```java
import java.util.*;
public class Main{
    public static int N = 1010;
    public static int[] v = new int[N];
    public static int[] w = new int[N];
    public static int[][] f = new int[N][N];
    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int m = sc.nextInt();
        for(int i = 1; i <= n; i++){
            v[i] = sc.nextInt();
            w[i] = sc.nextInt();
        }
        for(int i = 1; i <= n; i++){
            for(int j = 0; j <= m; j++){
                for(int k = 0; k * v[i] <= j; k++){
                    f[i][j] = Math.max(f[i][j], f[i - 1][j - k * v[i]] + k * w[i]);
                }
            }
        }
        System.out.println(f[n][m]);
    }
}
```

优化思路：

列举一下更新次序的内部关系：

```
f[i , j] = max( f[i-1,j] , f[i-1,j-v]+ w ,  f[i-1,j-2*v]+ 2*w , f[i-1,j-3*v]+ 3*w , .....)
f[i , j - v]= max(          f[i-1,j-v]   ,   f[i-1,j-2*v]+ w ,   f[i-1,j-3*v]+ 2*w , .....)
由上两式，可得出如下递推关系：
f[i][j]= max(f[i,j-v]+ w , f[i-1][j])
```

有了上面的关系，那么其实k循环可以不要了，核心代码优化成这样：

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

```
import java.util.*;
public class Main{
    public static int N = 1010;
    public static int[][] f = new int[N][N];
    public static int[] v = new int[N];
    public static int[] w = new int[N];
    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int m = sc.nextInt();
        for(int i = 1; i <= n; i++){
            v[i] = sc.nextInt();
            w[i] = sc.nextInt();
        }
        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] = Math.max(f[i][j], f[i][j - v[i]] + w[i]);
                }
            }
        }
        System.out.println(f[n][m]);
    }
}
```

进一步优化：

```
for(int i = 1 ; i<= n ;i++)
    for(int j = v[i] ; j<= m ;j++){
```

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

```
import java.util.*;
public class Main{
    public static int N = 1010;
    public static int[] f = new int[N];
    public static int[] v = new int[N];
    public static int[] w = new int[N];
    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int m = sc.nextInt();
        for(int i = 1; i <= n; i++){
            v[i] = sc.nextInt();
            w[i] = sc.nextInt();
        }
        for(int i = 1; i <= n; i++){
            for(int j = v[i]; j <= m; j++){
                f[j] = Math.max(f[j], f[j - v[i]] + w[i]);
            }
        }
        System.out.println(f[m]);
    }
}
```

3）：多重背包问题：（每种物品的件数有限制）

例题：

```
import java.util.*;
public class Main{
    public static int N = 110;
    public static int[][] f = new int[N][N];
    public static int[] w = new int[N];
    public static int[] v = new int[N];
    public static int[] s = new int[N];
    public static void main(String[] args){
```

```
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int m = sc.nextInt();
        for(int i = 1; i <= n; i++){
            v[i] = sc.nextInt();
            w[i] = sc.nextInt();
            s[i] = sc.nextInt();
        }
        for(int i = 1; i <= n; i++){
            for(int j = 0; j <= m; j++){
                for(int k = 0; k <= s[i] && k * v[i] <= j; k++){
                    f[i][j] = Math.max(f[i][j], f[i - 1][j - k * v[i]] + k * w[i]);
                }
            }
        }
        System.out.println(f[n][m]);
    }
}
```

例题：

二进制优化分析：

二进制优化思维就是：现在给出一堆苹果和10个箱子，选出n个苹果。将这一堆苹果分别按照1,2,4,8,16....512分到10个箱子里，那么由于任何一个数字x∈[0,1023]，都可以从这10个箱子里的苹果数量表示出来，但是这样选择的次数就是 ≤10次

比如：

如果要拿1001次苹果，传统就是要拿1001次；按照二进制的思维，就是拿7个箱子就行（分别是装有512、256、128、64、32、8、1512、256、128、64、32、8、1个苹果的这7个箱子）这样一来，1001次操作就变成7次操作就行了

```
import java.util.*;
public class Main{
    public static int N = 120010;
    public static int M = 2010;
    public static int[] v = new int[N];
    public static int[] w = new int[N];
    public static int[] f = new int[N];
    public static void main(String[] args){
```

```
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int m = sc.nextInt();
        int cnt = 0;
        for(int i = 1; i <= n; i++){
            int a = sc.nextInt();
            int b = sc.nextInt();
            int s = sc.nextInt();
            int k = 1;
            while(k <= s){
                cnt ++;
                v[cnt] = a * k;
                w[cnt] = b * k;
                s -= k;
                k *= 2;
            }
            if(s > 0){
                cnt ++;
                v[cnt] = a * s;
                w[cnt] = b * s;
            }
        }
        n = cnt;
        for(int i = 1; i <= n; i++){
            for(int j = m; j >= v[i]; j--){
                f[j] = Math.max(f[j], f[j - v[i]] + w[i]);
            }
        }
        System.out.println(f[m]);
    }
}
```

4)：分组背包问题：

集合划分依据：第 i 种物品选哪个

```
import java.util.*;
public class Main{
    public static int N = 110;
```

```
    public static int[] f = new int[N];
    public static int[][] w = new int[N][N];
    public static int[][] v = new int[N][N];
    public static int[] s = new int[N];
    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int m = sc.nextInt();
        for(int i = 1; i <= n; i++){
            s[i] = sc.nextInt();
            for(int j = 1; j <= s[i]; j++){
                v[i][j] = sc.nextInt();
                w[i][j] = sc.nextInt();
            }
        }
        for(int i = 1; i <= n; i++){
            for(int j = m; j >= 0; j--){
                for(int k = 0; k <= s[i]; k++){
                    if(j >= v[i][k]){
                        f[j] = Math.max(f[j], f[j - v[i][k]] + w[i][k]);
                    }
                }
            }
        }
        System.out.println(f[m]);
    }
}
```