Problem K. Knapsack

Input file:	*standard input*
Output file:	*standard output*
Time limit:	3 seconds
Memory limit:	512 mebibytes

```
To fit the best stuff in a sack
with limits on what you can pack
       recursion will do
        if memoized too
but DP puts less on the stack.
```

You are given a classic knapsack problem: a set of elements $(w_1, v_1), \ldots, (w_n, v_n)$ and a capacity W. Solve

$$\max_{\substack{S \subseteq [n] \\ \sum_{i \in S} w_i \le W}} \sum_{i \in S} v_i.$$

Here, $[n]$ is the list of integers from 1 through n.

You are guaranteed that $0 \le n \le 500$, $0 \le w_i \le W \le 10^{17}$, and $0 \le v_i \le 10^{16}$. Furthermore, you are guaranteed that the w_i have been "smoothed" as described in the next paragraph.

A problem instance complying with the above bounds is constructed. Then a randomizing filter is applied to the problem instance as follows: Let B be the weight of the element of maximum weight. Now the following update is applied to each weight:

$$w_i \leftarrow \min(w_i + \text{rand}(\lfloor .05B \rfloor), W)$$

Here $\text{rand}(A)$ is a function that generates a uniform random integer in $[0, A-1]$. This process is applied only to the w_i, not to W or any of the other values.

Input

The first line contains the two space-separated integers n and W. Each of the next n lines contains two space-separated integers w_i and v_i. The element weights were generated as described above. First the items are chosen in compliance with all the bounds. Then the smoothing algorithm is applied. The result is what your program will receive as input.

Output

Output one line with the value of the given knapsack problem instance.

Examples

standard input	standard output
3 5 1 2 2 3 3 4	7
3 302000 100588 2 200421 1000 30036 1	1002