Tower Challenge

You will have n blocks which have a hole. A height of a block is h_i, and the height of a hole is $d_i(< h_i, h_{i+1})$. The lower side of the block $i+1$ and the upper side of block i interlock, however, the lower side of the block $i+1$ cannot join with any other blocks.

Therefore, when block j is placed on block i to form a sub-tower, the height will be $h_i - d_i + h_{i+1}$ where $j = i+1$ or $h_i + h_j$ where $j \neq i$, $j \neq i+1$ as show in fig 1.

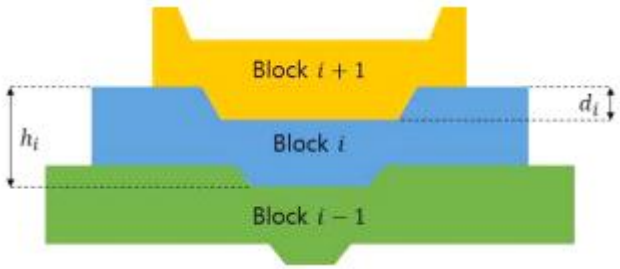

Fig 1 Example of placing consecutive blocks

Now we are to stack the blocks by following the rules below:
1) If $i < j$ then block i must be placed below bock j.
2) The height of the tower cannot excced H.

You cannot place more than one block on another and tilt any of the blocks.
If you have chosen to build the tower in order of block 1, block 2, block 5, block 6, block 7, block 9, then the height will be $h_1 - d_1 + h_2 + h_5 - d_5 + h_6 - d_6 + h_7 + h_9$.

Given h_i, d_i, H, write an algorithm that find the number of all possible combinations of the blocks. You are to print mod 1,000,000 of it due to the possibility of integer overflow.

[Constraints]

- $10 \leq n \leq 1000$ in $\mathbb{N}$.
- $100 \leq H \leq 10{,}000$ in $\mathbb{N}$.
- $1 \leq h_i \leq 100$ $(0 \leq i \leq n-1)$ in $\mathbb{N}$.
- $0 \leq d_i \leq \min(h_i, h_{i+1}) - 1$ where $(0 \leq i \leq n-2)$ in $\mathbb{N}$.
- Memory usage : ~ 128 MB.
- Time limit for solving 10 test cases : ~ 1 second.

[Input]

An input file “input.txt” will be given with 10 test cases. The first line of a test case contains the number of blocks n and max tower height H, the second line is a sequence of h_i and the

third line is a sequence of d_i separated by space.

[Output]
For each case, you should print the case number as #x where x is the index of the case. Then, print (the number of all possible combinations) mod (1M) followed by space.

You must produce your results as “output.txt”

[Example]
Input (input.txt)

```
10 100                                                          ← First Case
49 64 69 76 45 40 50 36 57 87
18 13 58 30 30 31 26 32 50
20 200                                                          ← Second Case
75 87 34 5 24 16 7 91 50 13 15 66 94 93 37 63 19 80 12 62
65 17 4 3 12 1 0 26 3 0 4 4 69 3 3 1 11 9 1
...
```

Output (output.txt)

```
#1 34
#2 20825
...
```