Rod Collecting

Objective
Give practice with Dynamic Programming.
Give practice with inverse State DPs.

Story
You make it back to the chamber with the rods. Someone has come through and pilfered the golden chain. Luckily it appears that the door stayed open since the whole place is collapsing. The rods themselves seem to have shaken loose from their fixtures.

You find yourself thinking how lucky you were before deciding to take some of the rods that are left over. Basically, grab anything that is not nailed down before it is consumed by the desert. Each rod has some weight and value. You know how much you can carry. Your job is to determine the maximum value of rods you can take with you. Simple enough right?

Problem
Given your carrying capacity and the rod descriptions in terms of their weight and value determine the highest value sum you can take in one trip without exceeding your carrying capacity.

Input
Input will begin with a line containing 2 positive integers, ***C*** and ***R*** (***C*** ≤ 1,000,000; ***R*** ≤ 1,000), representing the amount of weight you can carry in grams (you've been working out) and the number of rods respectively. The following ***r*** lines will each contain 2 values representing a rod. The first rod value is a positive integer, ***v*** (***v*** ≤ 100), representing the value of rod. The second value is a positive floating pointer, ***w*** (2 ≤ 1,000), representing the weight of the rod in grams. The weight will be described with exactly 3 digits after the decimal point.

Output
Output should contain 1 line with a single integer representing the maximum value sum you can carry without exceeding your carry capacity.

Sample Input	Sample Output

`100 4`
`19 33.450`
`52 64.900`
`23 31.100`
`48 37.731`
	`75`

`100 7`
`74 80.583`
`86 83.109`
`60 8.689`
`36 38.940`
`49 52.681`
`83 74.350`
`61 21.270`
	`170`

Explanation

Case 1

We select the 2nd and 3rd rod. The total value is 52 + 23 = 75. The total weight is 64.9 + 31.1 = 96.0, which is less than or equal to 100

Note: If you are using the floating point conversion hint we are really working with a capacity of 100,000, and the weight of the rods is 96,000.

Case 2

We select the 3rd, 5th, and 7th rods. The total value is 60 + 49 + 61 = 170. The total weight is 8.689 + 52.681 + 21.270 = 82.640, which is less than or equal to 100.

Hints

0-1 Knapsack: This is very much like the 0-1 knapsack with extra constraints.

States With Floating Points: The straight forward DP (which you should not use; see Reverse State Hint) uses the weight as the state, where the value stored is the maximum value (like a 1-0 knapsack). The problem is that the weight is a floating point, and accessing an array using a float is tricky. To allow for handling the decimal values I recommend “multiplying” the decimal values by a large enough power of 10 such that we work with whole values (ints/longs). Since the decimals all have 3 spots, we should be able to multiply by 1000.

Reverse State: In the straight forward 0-1 knapsack DP the number of states we need is the number of possible unique weights In this case since the knapsack can carry up to 1,000,000,000 milligrams, the number of states is roughly 10^9. Unfortunately we cannot store that many integers in an array. To fix this we can store the current value sum of the rods as the state instead of the weight. The value will only go up to 100 times 1,000 which is 10^5 (usable by an array in java). The array itself will hold onto the minimum weight necessary to reach said rod value sum. To pull out the answer we only need to find the high spot in the array (value sum), that stores an answer (weight sum) less than that of the given capacity.

Space saving: Since we don’t need to pull out the answer, I suggest not storing a table that includes both the rod current at and the value. That will break the bank in terms of memory. Instead use an iterative DP with a simple array to compute your answer.

Grading Criteria

- Good comments, whitespace, and variable names
    - 15 points
- No extra input output (e.g. input prompts, “Please enter the number of friends:”)
    - 10 points
- Use non-floating points data types to store the weights
    - 10 points
- Use a dynamic programming “table” (1-d array is preferable) to store the answers
    - 10 points
- Extract the answer from the “table”
    - 5 points
- Programs will be tested on 10 cases
    - 5 points each

No points will be awarded to programs that do not compile using javac.

Sometime a requested technique will be given, and solutions without the requested technique will have their maximum points total reduced. For this problem use dynamic programming. ***Without this programs will earn at most 50 points!***

Any case that causes a program to return a non-zero return code will be treated as wrong. Additionally, any case that takes longer than the maximum allowed time (the max of {5 times my solutions time, 10 seconds}) will also be treated as wrong.

No partial credit will be awarded for an incorrect case.