Homework 10, CSE 232

Due October 29 Note: On most of the problem sets through the semester, I'll put a horizontal line with "Optional" under it. Any problems below this section are encouraged - I think they're interesting and will help you learn the subject - but not necessary to complete in order to get credit for the homework.

Problem 1

This week's problem is Google Code Jam Round 1A 2012 - Password Problem. Here's the problem statement:

> I have a really long password, and sometimes I make a mistake when I type it. Right now I've typed part of my password, but I might have made some mistakes. In particular, I might have pressed the wrong key while typing one or more of the previous characters. Given how likely I was to get each character right, what should I do?
>
> I have three options:
>
> Finish typing the password, then press "enter". I know I'll type the rest of the characters perfectly. If it turns out that one of the earlier characters was wrong, I'll have to retype the whole thing and hit "enter" again – but I know I'll get it right the second time. Hit "backspace" some number of times, deleting the last character(s) I typed, and then complete the password and press "enter" as in option 1. If one of the characters I didn't delete was wrong, I'll have to retype the whole thing and press "enter", knowing I'll get it right the second time. Give up by pressing "enter", retyping the password from the start, and pressing "enter" again. I know I'll get it right this time. I want to minimize the expected number of keystrokes needed. Each character in the password costs 1 keystroke; each "backspace" costs 1 keystroke; pressing "enter" to complete an attempt or to give up costs 1 keystroke.
>
> *Note:* The "expected" number of keystrokes is the average number of keystrokes that would be needed if the same situation occurred a very large number of times. See the example below.

Example

> Suppose my password is "guest" and I have already typed the first two characters, but I had a 40% chance of making a mistake when typing each of them. Then there are four cases:
>
> I typed "gu" without error. This occurs with probability $0.6 \times 0.6 = 0.36$.
>
> I typed the 'g' correctly but I made a mistake typing the 'u'. Then I have two letters typed still, but the second one is wrong: "gX". (Here, the 'X' character represents a mistyped letter.) This occurs with probability $0.6 \times 0.4 = 0.24$.
>
> I typed the 'u' correctly but I made a mistake typing the 'g': "Xu". This occurs with probability $0.4 \times 0.6 = 0.24$.
>
> I made a mistake typing both letters, so I have two incorrect letters: "XX". This occurs with probability $0.4 \times 0.4 = 0.16$.
>
> I don't know how many mistakes I actually made, but for any strategy, I can calculate the expected number of keys required to use it. This is shown in the table below:

	gu	*gX*	*Xu*	*XX*	*Expected*
Probability	0.36	0.24	0.24	0.16	–
Keystrokes if I keep typing	4	10	10	10	7.84
Keystrokes if I press backspace once	6	6	12	12	8.4
Keystrokes if I press backspace twice	8	8	8	8	8
Keystrokes if I press enter right away	7	7	7	7	7

(1)

If I keep typing, then there is an 0.36 probability that I will need 4 keystrokes, and an 0.64 probability that I will need 10 keystrokes. If I repeated the trial many times, then I would use 4 keystrokes 36% of the time, and 10 keystrokes the remaining 64% of the time, so the average number of keystrokes needed would be $0.36 \times 4 + 0.64 \times 10 = 7.84$. In this case however, it is better to just press enter right away, which requires 7 keystrokes.

Input

The first line of the input gives the number of test cases, T. T test cases follow. Each test case begins with a line containing two integers, A and B. A is the number of characters that I have already typed, and B is the total number of characters in my password.

This is followed by a line containing A real numbers: $p_1, p_2, \ldots, p_A$. p_i represents the probability that I correctly typed the ith letter in my password. These real numbers will consist of decimal digits and at most one decimal point. The decimal point will never be the first or the last character in a number.

Output

For each test case, output one line containing "Case #x: y", where x is the case number (starting from 1) and y is the expected number of additional keystrokes I need, not counting the letters I have typed so far, and assuming I choose the optimal strategy. y must be correct to within an absolute or relative error of 10^{-6}.

Limits

$1 \leq \mathrm{T} \leq 20$.

$0 \leq p_i \leq 1$ for all i.

Small dataset

$1 \leq \mathrm{A} \leq 3$.

$\mathrm{A} < \mathrm{B} \leq 100$.

Large dataset

$1 \leq \mathrm{A} \leq 99999$. $\mathrm{A} < \mathrm{B} \leq 100000$.

a) The problem describes three strategies. The first strategy is to keep typing the password, retyping the password after if you get it wrong the first time around. How many keystrokes will this strategy take if you made a mistake in the first A characters? How about if you got all the first A characters correct?

You'll always take $B - A + 1$ keystrokes to finish the current password and then press enter. If the first A characters were correct, we're done. If not, it takes an additional $B + 1$ keystrokes to retype the password, for a total of $2B - A + 2$ keystrokes.

b) The second strategy is to press backspace k times, where $k \in [1, A]$, removing the last k characters of the password, then completing the password and retyping the whole thing if incorrect. How many keystrokes will this strategy take if there is a mistake in the first $A - k$ characters? How about if the first $A - k$ characters are correct?

First we press backspace k times, then we must retype those k characters, taking $B - A + 2k + 1$ keystrokes to complete the current password and then press enter. If there was a mistake, then we'll take an additional $B + 1$ keystrokes to retype the password, for a total of $2B - A + 2k + 2$ keystrokes.

c) The final strategy is to press enter right now, get the password wrong, and then retype it. How many keystrokes will this take?

1 keystroke for the enter, and then $B + 1$ for the correct password, for a total of $B + 2$ keystrokes.

d) Given the (independent) probabilities p_k that we typed the k_{th} character correctly, explain how to calculate the probability P_k that we got all of the first k characters typed correctly. How can we calculate this efficiently for all values of k between 1 and A?

We can do this by simply multiplying the probabilities of getting each of the first k characters correct. If we did this separately for each value of k, this would take $O(A^2)$ time. We can do it instead in $O(A)$ time by initializing P to 1.0. We then multiply by p_1 to get P_1. Then we multiply by p_2, giving $p_1 * p_2 = P_2$. Then multiply by p_3, giving $p_1 * p_2 * p_3 = P_3$, and so forth.

e) Using the expected value calculations we discussed in class and your work above, write an efficient solution to find the best strategy for a given input, submit your solution to the Google Code Jam, and attach it to your homework.

In []:
```
import sys

def solve_prob(A, B, probs):
    #First option: press enter immediately
    mincost = B + 2

    #Cycle through deleting from A to 0 characters
    prob = 1.0
    for i, k in enumerate(range(A+1)[::-1]):
        cost = prob * (B - A + 2 * k + 1) + \
               (1 - prob) * (2 * B - A + 2 * k + 2)
        mincost = min(mincost, cost)
        try:
            prob = prob * probs[i]
        except IndexError:
            pass #we're looping through 1 more than probs has
    return mincost

infile = open("%s" % sys.argv[1], 'r')
outfile = open("%s.out" % sys.argv[1][:-3], 'w')
cases = int(infile.readline().strip())
for i in range(cases):
    A, B = [int(num) for num in infile.readline().split()]
    probs = [float(num) for num in infile.readline().split()]
    output = solve_prob(A, B, probs)
    outfile.write("Case #%i: %.6f\n" % (i+1, output))
    print "Case #%i: %.6f" % (i+1, output)
infile.close()
outfile.close()
```

Optional