Evolutionary Computation, 2017/2018
Programming assignment 3

Important information

- Deadline: 25/Oct/2017, 23:59.
- All problems must be submitted through Mooshak.
- Please go to `http://mooshak.deei.fct.ualg.pt/~mooshak/` and register in 'EC1718_P3' from the 'contest' list box.
  - You can submit programs in C, C++, Java, or Python.

About this assignment

The purpose of this programming assignment is to get you involved in programming more components of a simple genetic algorithm.

Similarly to the previous, you will be given pseudo-random numbers uniformly generated in [0..1), so that the output of your programs can be checked for correctness.

Problem A: Random permutation

Write a computer program that given a number N generates a random permutation of the integers ranging from 0 to $N-1$. In order to that, you should implement the following algorithm:

```
for( i=0; i<N; i++)
   v[i] = i;
for(i=0;i<N-1;i++) {
   r = obtain a uniformly generated random integer in [i..N-1];
   exchange the contents of v[i] with the contents of v[r]
}
```

On completion, the array v will contain a random permutation. The body of the second for loop shown above is executed $N-1$ times. Therefore you will be given $N-1$ pseudo-random numbers uniformly generated in [0..1) as part of the input. You should map each of these numbers into an apropriate distribution (just like you did in previous assignments).

Input

The first line contains the integer N. Each of the next $N-1$ lines contains contains a pseudo random number uniformly generated in [0..1).

You can assume $N \leq 100$.

Output

The output is the random permutation, one element per line, followed by a newline character.

Example

Sample Input

```
4
0.54353
0.23432
0.43434
```

Sample Output

```
2
1
0
3
```

Problem B: Tournament selection without replacement

Implement *tournament selection without replacement* as explained in class. You can assume that the population size N is a multiple of the tournament size s. You will need to generate a random permutation of the population s times. In each time, you use the generated permutation (from the beginning to the end) to perform N/s tournaments.

Input

The first line contains an integer N that specifies the size of the population. The second line contains an integer ℓ that specifies the string length. The third line contains an integer s that specifies the tournament size. Each of the next N lines contain a binary string of length ℓ, followed by a white space, followed by string's fitness value (a real number). Then, each of the next $s * (N - 1)$ lines contains a pseudo random number uniformly generated in [0..1). You should use these numbers to obtain random permutations, as needed.

You can assume that both N and ℓ are less or equal than 100, and that N is a multiple of s.

Output

The output should be the population of selected strings. You should send them to the output in the same order as the tournament winners are found.

Example

Sample Input

```
4
3
2
110 8.0
001 7.5
101 23.0
111 5.0
0.16758
0.54902
0.97234
0.25456
0.19763
0.83764
```

Sample Output

```
101
001
110
101
```

Problem C: One generation on onemax

Simulate one generation of a genetic algorithm operating on the *onemax* problem. After reading the input, you must generate a random population and evaluate the fitness of all its individuals. Then you will do one generation of a genetic algorithm. As you know, a generation consists of applying selection, crossover, mutation, and replacement.

As usual, you will be given a sufficient amount of pseudo-random numbers in [0..1) to allow your program to be tested automatically. For the purpose of this exercise, you must use:

- tournament selection without replacement with tournament size s
- onepoint crossover with probability P_c
- bit-flip mutation with probability P_m
- replace worse strategy with a replacement fraction of r

Input

The first line contains an integer N that specifies the size of the population. The second line contains an integer ℓ that specifies the string length. The third line contains an integer s that specifies the tournament size. The fourth line contains a real number P_m that specifies the mutation probability. The fifth line contains a real number P_c that specifies the crossover probability. The sixth line contains a real number r that specifies the fraction of the worse individuals from the old population that is going to be replaced by newly created individuals. (**Important: the actual number of individuals to be replaced should be rounded up to the nearest integer**). The seventh line contains an integer M that specifies the amount of random numbers that are given to you. Each of the next M lines contains a pseudo random number uniformly generated in [0..1).

You can assume that both N and ℓ are less or equal than 100, and that N is a multiple of s.

Output

The output consists of 2 lines. In each line you should print the generation number followed by a colon and a white space, followed by the maximum, average, and minimum fitness of the population, all separated by a white space. Each of these numbers

must be reported with a precision of 2 decimal points. Each line ends with a newline character.

Example

Sample Input

```
4
5
2
0.1
0.8
0.5
50
0.1842014
0.8733817
0.9268632
0.7895050
0.2096997
0.4234094
0.2420872
0.7599762
0.9202078
0.9319989
0.1048958
0.9837330
0.6003861
0.6891934
0.2730424
0.0237503
0.1719036
0.1845107
0.0707562
0.1989022
0.9498948
0.8825581
0.1542663
0.7533399
0.3839350
0.7957093
0.4856689
0.6366967
0.9608628
0.2218308
0.3110664
0.0930620
```

```
0.0936589
0.1247946
0.4232356
0.3207462
0.7813338
0.8771974
0.0559059
0.6109542
0.3072115
0.3035363
0.5351927
0.9829322
0.1411754
0.7349411
0.1547252
0.4669782
0.5017778
0.3788434
```

Sample Output

```
0: 5.00 2.75 2.00
1: 5.00 3.25 2.00
```

Problem D: A genetic algorithm on onemax

This is similar to the previous exercise, but this time you will be simulating multiple generations, just like in a complete genetic algorithm (GA). Again you will be simulating your GA on the *onemax* problem.

Another difference from the previous exercise is that this time you will be using a *full replacement* strategy (it's just a particular case of the replace worse strategy with $r = 1.0$. You will be using the same selection, crossover, and mutation operators from the previous exercise.

Input

The first line contains an integer N that specifies the size of the population. The second line contains an integer ℓ that specifies the string length. The third line contains an integer s that specifies the tournament size. The fourth line contains a real number P_m that specifies the mutation probability. The fifth line contains a real number P_c that specifies the crossover probability. The sixth line contains an integer g that specifies the number of generations (in addition to generation 0 which consists of a randomly generated population) to be performed. The seventh line contains an integer M that specifies the amount of random numbers that are given to you. Each of the next M lines contains a pseudo random number uniformly generated in [0..1).

You can assume that both N and ℓ are less or equal than 100, and that N is a multiple of s. You can also assume that g is less or equal than 30.

Output

The output consists of $1 + g$ lines. In each line you should print the generation number followed by a colon and a white space, followed by the maximum, average, and minimum fitness of the population, all separated by a white space. Each of these numbers must be reported with a precision of 2 decimal points. Each line ends with a newline character.

Example

Sample Input

```
4
5
2
0.1
0.7
2
80
0.4966927
0.9141867
0.7361678
0.7719724
0.5399551
0.0258011
0.6395234
0.4703913
0.8671000
0.3499480
0.5758458
0.2406369
0.3847257
0.0844225
0.8892065
0.8930698
0.8245111
0.5576897
0.0908595
0.0757619
0.3298830
0.3434172
0.8136623
0.2217014
```

0.1355392
0.0079407
0.4598270
0.3130233
0.9831832
0.3598858
0.6011862
0.1363906
0.3167899
0.2878979
0.7007233
0.0569271
0.7740724
0.8350804
0.1966355
0.8533989
0.0758357
0.5713643
0.9196748
0.9747114
0.9748282
0.9368755
0.0657496
0.0532079
0.2653307
0.4134482
0.8245831
0.7687343
0.1166590
0.6877130
0.3926467
0.2129672
0.3391955
0.8585698
0.9829627
0.6544835
0.9038520
0.0409920
0.9531806
0.1065779
0.2549971
0.7363358
0.5950064
0.2724407
0.9103595

```
0.4113093
0.8750638
0.1972820
0.7193735
0.5101380
0.8885934
0.5898347
0.3517897
0.5299424
0.7411353
0.2613552
```

Sample Output

```
0: 3.00 2.25 1.00
1: 4.00 2.75 1.00
2: 4.00 3.75 3.00
```