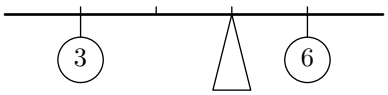
1. Lever

1 second 10 points

Let's consider a lever of length L that has N weights attached to it. For each weight, its location X_i (as distance from the left end of the lever to the point from which the weight hangs) and mass M_i are given.

Write a program to find the location (as distance from the left end of the lever) where the fulcrum should be placed for the lever to be balanced.



Input. The first line of the text file `kangsis.txt` contains two space-separated integers: L, the length of the lever ($0 < L \leq 100$), and N, the number of weights ($1 \leq N \leq 100$). Each of the following N lines also contains two space-separated integers: the position X_i ($0 \leq X_i \leq L$) and the mass M_i ($0 < M_i \leq 100$) of the weight i.

Output. The only line of the text file `kangval.txt` should contain a single real number: the location (as distance from the left end of the lever) of the fulcrum. The answer should not differ from the correct value by more than 0.01.

Example.

kangsis.txt	kangval.txt
5 2 1 3 4 6	3.0

Grading. In tests worth 5 points in total, $N \leq 5$.

2. Sorting Cards

1 second 20 points

A common notation for writing down hands in card games uses two characters to denote each card's rank and suit. The characters `AKQJT98765432` are used for ranks (ace, king, queen, jack, ten, 9, ..., 2) and the characters `shdc` for suits (spades, hearts, diamonds, clubs). For example, the queen of spades is written as `Qs` in this notation.

Write a program to sort a given set of cards first by suits and then within each suit by ranks (in the order given in the previous paragraph).

Input. The fist line of the text file `sortsis.txt` contains N, the number of cards ($1 \leq N \leq 52$) and the second line $2 \cdot N$ characters describing the N cards (all distinct).

Output. The only line of the text file `sortval.txt` should contain $2 \cdot N$ characters listing the members of the input set in the required order.

Example.

sortsis.txt	sortval.txt
5 3d8sAdTd6c	8sAdTd3d6c

3. Tower of Hanoi

1 second 30 points

Tower of Hanoi is a puzzle consisting of three rods and N disks. There's exactly one disk with each size $1 \ldots N$. Each disk has a hole in the middle and can slide onto any of the rods. Initially all the disks are stacked on the leftmost rod in decreasing order and the goal is to move them to the rightmost rod, using the middle rod as a temporary storage. With each move, the player may move the topmost disk from one rod to the top of the stack on another rod, but must never put a larger disk on top of a smaller one.

Write a program that reads some state of the puzzle and computes the minimal number of moves needed to solve it from the given state (that is, to move all the disks to the rightmost rod in the correct order).

Input. The first line of the text file `tornsis.txt` contains N_v, the number of disks on the leftmost rod, followed by N_v space-separated integers, the sizes of the disks, from bottom to top; the second line contains N_k, the number of disks on the middle rod, followed by their sizes; the third line contains N_p, the number of disks on the rightmost rod and their sizes ($0 \le N_v$, $0 \le N_k$, $0 \le N_p$, $N_v + N_k + N_p \le 30$). It may be assumed the state is valid (each disk $1 \ldots N_v + N_p + N_k$ occurs exactly once, and nowhere is a larger disk on top of a smaller one).

Output. The only line of the text file `tornval.txt` should contain a single integer: the number of moves needed to solve the puzzle, starting from the state given in the input file.

Example.

tornsis.txt	tornval.txt
2 2 1 1 3 0	4

The puzzle can be solved in four moves as follows: disk 3 from the middle rod to the right; disk 1 from left to middle; disk 2 from left to right; disk 1 from middle to right. The puzzle can't possibly be solved with three moves, as then each disk would have to be placed to the right rod the first time it is moved; but this is impossible as then disk 1 would end up below disk 2.

Grading. In tests worth 20 points in total, $N_v + N_k + N_p \le 10$.

4. Digit Sum

1 second 40 points

Digit sum of an integer is the sum of all its digits. For example, the digit sum of 123 is 1+2+3 = 6 and the digit sum of 99 is $9 + 9 = 18$.

Write a program that counts the number of non-negative integers smaller than the given one that have the same digit sum as the given one.

Input. The only line of the text file `ristsis.txt` contains the given integer N ($0 \le N \le 10^{18}$).

Output. The only line of the text file `ristval.txt` should contain the number of non-negative integers that are smaller than N and have the same digit sum as N.

Example.

ristsis.txt	ristval.txt
123	9

The smaller integers are 6, 15, 24, 33, 42, 51, 60, 105, and 114.

Grading. In tests worth 20 points in total, $N \le 10^9$; among them in tests worth 10 points, $N \le 10^3$.

5. Eating Scarves

5 seconds 50 points

Märt the fashion artist has in his wardrobe N beautiful scarves that he wears to the fashion meetups on Saturdays. Each scarf has a specific fashion value and Märt gets corresponding amount of cool points for wearing it to a meetup. Wearing the same scarf twice would be a huge *faux pas*, however, that Märt would never commit. He puts the used scarves back to the wardrobe, but will not wear them again.

In Märt's wardrobe lives Kärt the moth who eats a hole into one of the scraves every Sunday. A scarf with a hole in it can't be worn any more. Kärt does not care about the fashion value and eats the scarves randomly. The probability of picking a scarf is proportional to its length. Kärt may also eat again a scarf that already has a hole in it.

Clearly, Märt would be able to attend at most N meetups, if Kärt would only eat used scarves. But if Kärt sometimes eats new scarves, the number of meetups will decrease accordingly.

Write a program to compute the amount of cool points Märt can expect to collect on average if he will use the optimal strategy to pick the scarves, but Kärt will eat them randomly. The process starts on a Monday, so all the scarves will be intact for the first meetup.

Input. The first line of the text file `sallsis.txt` contains N, the number of scarves ($1 \leq N \leq 20$). Each of the following N lines contains two integers M_i and P_i ($1 \leq M_i \leq 100$, $1 \leq P_i \leq 20$), the fashion value and length of a scarf.

Output. The only line of the text file `sallval.txt` should contain one real number, the mean of total cool points Märt can expect to collect if he picks the scarves optimally, but Kärt eats them randomly. The answer should not differ from the correct value by more than 0.001.

Example.

sallsis.txt	sallval.txt
3	48.333
10 3	
20 2	
30 1	

Given the input data, Märt's best strategy would be to pick the second scarf for the first meetup. After that:

- with probability $\frac{1}{2}$ the hole will be in the first scarf, only the third one will be available for the second meetup, and consequently he will get 50 cool points;
- with probability $\frac{1}{3}$ the hole will be in the second scarf and there will be two scarves left; he'll pick the third one for the second meetup and with probability $\frac{1}{2}$ will be able to use the first one as well; expected total is 55 cool points;
- with probability $\frac{1}{6}$ the hole will be in the third scarf, only the first one will be available, and he will get 30 cool points.

The weighted mean of these is $48\frac{1}{3}$ cool points.

Example.

sallsis.txt	sallval.txt
4	84.94144
10 10	
20 9	
30 5	
40 1	

Grading. In tests worth 25 points in total, $N \leq 10$.

6. Area

Testing 50 points

An engineering bureau commissioned a software house to develop a program to compute the area of the intersection of a triangle and a quadrangle. The program is required to be able to handle all kinds of triangles and quadrangles.

Create a test data set that the engineering bureau can use to verify the correctness of the program before they pay the software house.

Input. The first line of the text file `pindsis.txt` contains six space-separated integers A_x, A_y, B_x, B_y, C_x, C_y: the coordinates of the vertices of the triangle ABC. The second line contains in similar fashion the coordinates of the vertices of the quadrangle $DEFG$ in the order in which they appear on the edge of the quadrangle. All coordinates are integers whose absolute values do not exceed 100.

Output. The only line of the text file `pindval.txt` should contain a single real number: the area of the intersection of the two figures described in the input file; the answer should not differ from the correct value by more than 0.01.

Example.

`pindsis.txt`
```
1 1 5 1 3 5
1 3 5 3 5 6 1 6
```

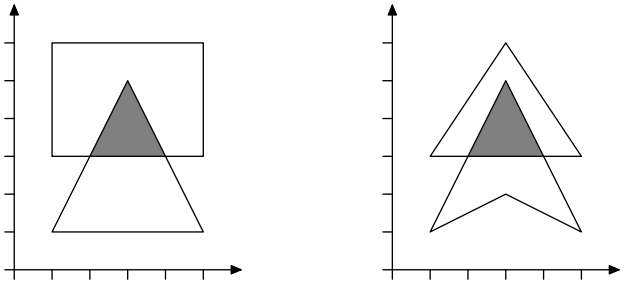
`pindval.txt`
```
2.0
```

Example.

`pindsis.txt`
```
1 3 5 3 3 6
1 1 3 2 5 1 3 5
```

`pindval.txt`
```
2.0
```

The figure below illustrates the first example on the left and the second one on the right.



Grading. In this task you should submit one ZIP archive containing up to 15 input files named `pindsis01.txt`, `pindsis02.txt`, ... and corresponding output files named `pindval01.txt`, `pindval02.txt`, ...

If a pair of files is not a valid input file and a valid output file corresponding to it, they earn no points. Correct pairs of files will be used to evaluate programs with known bugs in them and the contestant will receive points for each of these programs that gives a wrong answer or terminates abnormally in at least one of the contestant's test cases.

7. Black Box

1 second 75 points

Various kinds of checksums are used to detect errors when data is entered, transmitted, and stored. For example, to protect the message M in transit, the sender computes $S = f(M)$ using a function f previously agreed upon and instead of just the message M transmits the pair $\langle M, S \rangle$. When the recipient gets the pair $\langle M', S' \rangle$, they can check whether $f(M') = S'$. If the equality does not hold, there has surely been a transmission error and the recipient could, for example, ask the sender to re-transmit.

If the equality holds, it does not fully quarantee that the message has been received intact. It is possible that both M and S have changed in transit (either by an unfortunate coincidence or due to someone's malicious intent) in such a way that the checksum verification does not detect it. The probability of such false positive result can be reduced by choosing the function f appropriately. In this task we examine various types of functions used to compute checksums.

Parity Bit.[1] The simplest checksum is a 1-bit value picked according to the number of 1-bits in the binary representation of the transmitted data. Several variations of the scheme have been used over time: in some of them the total number of 1-bits in the data and the checkum is to be even, in some it is to be odd; in some the check bit is appended to the data, in some it is prepended. In any case the checksum in this scheme has only two possible values, so the probability of false positives is rather high, 50%. On the other hand, it is very easy to implement this type of checksums in hardware.

Digit Sum.[2] Sending a digit sum of the data instead of the parity bit is somewhat more reliable. For example, if a communications channel transmits zeroes for all bits, and the duration of this problem is such that an even number of 1-bits is replaced with 0-bits, then any kind of parity bit will fail to detect the mistake, but the digit sum of transmitted data will surely decrease as the result of such an error.

Check Digit.[3] A plain digit sum fails to detect changes in the ordering of characters in the message (a type of error that can easily happen when data is entered by humans). To detect these errors, for example the account and routing numbers used by banks have check digits added to them in such a way that when the digits in the number are multiplied by the weights $3, 7, 1, 3, 7, 1, \ldots$ before summing, then for a correct account or routing number the result is always a multiple of 10.

CRC.[4] Much more complex than the schemes discussed so far are the CRC-sums. To compute those, the transmitted bits are viewed as coefficients of a polyomial P and the checksum is computed as the remainder of dividing P by a fixed polynomial G (just as for integers P and G it would possible to find the quotient Q and the remainder R such that $P = G \cdot Q + R$, a similar thing can be done with polynomials; it's worth pointing out that the coefficients of these polynomials are treated as 1-bit integers, with $1 + 1 = 0$ due to overflow). This scheme has also been used in many variations over the time, with different lengths of checksums generated by different divisors, and also the order in which bits are extacted from a byte or bytes from a multi-byte checksum has varied.

Cryptographic Hash Functions.[5] The checksums described so far have all been designed to detect accidental errors and offer little protection against intentional attacks. Indeed, the

[1] http://en.wikipedia.org/wiki/Parity_bit
[2] http://en.wikipedia.org/wiki/Digit_sum
[3] http://en.wikipedia.org/wiki/Check_digit
[4] http://en.wikipedia.org/wiki/Cyclic_redundancy_check
[5] http://en.wikipedia.org/wiki/Cryptographic_hash_function

attacker could just re-compute the checksum after altering the message and send the modified message along with the new checksum. To prevent this, the checksum scheme needs two additional properties.

First, the function f has to be chosen so that it would be very difficult to change the message in such a way that the checksum of the modified message would remain the same as the original. Cryptographic hash functions, among which MD5, SHA1, and the SHA2 family are most widely known, have this property.

Second, the sender and recipient need to share a password that the attacker does not know. In the simplest form, the password P could be used as follows. Instead of $f(M)$, the sender computes the checksum as $S = f(MP)$ (the password P is appended to the message M before the checksum is computed), but transmits still only $\langle M, S \rangle$. Now if the attackers want to replace the original message M with a modified one M', they can't do this, because without knowing the password P, they can't compute $f(M'P)$. The recipient, however, can re-compute the checksum from the received message and the known password.

There are five web apps at `http://prog.offline.ee/html2/kast.cgi` that compute checksums based on the schemes just explained. Determine for each one, which variant of which scheme (and for the hash function based one, also which password) is used and write a program to compute exactly the same sums.

Grading. Each of the five apps is a separate sub-task. A separate program has to be submitted for each sub-task and will be graded separately. The programs should use files for input and output, not web interfaces.

Input. The first line of the text file `kastsis.txt` contains N, the number of messages ($1 \leq N \leq 100$) and each of the following N lines contains one message. It may be assumed that

- all messages contain only printable characters of the 7-bit ASCII;
- the length of any message does not exceed 80 characters;
- only messages that would be accepted by the corresponding web app will be used.

Output. The text file `kastval.txt` should contain exactly N lines. The i-th line of output should contain the message from the input line $i+1$ with the checksum added in the exact same format as used by the corresponding web app.

Example.

`kastsis.txt`	`kastval.txt`
`2`	`246`
`123`	`912`
`456`	