University of Central Florida

2020 Local Programming Contest (Final Round)

Problems

Problem#	Difficulty Level	Filename	Problem Name
1	Easy	virus	Corona Virus Testing
2	Easy	elect	Presidential Election
3	Easy	microwave	Microwave Mishap
4	Medium	food	Food Display Arrangement
5	Medium	connect	Making Connections
6	Medium	books	Boxing Books
7	Medium-Hard	badtree	Bad Tree
8	Medium-Hard	hanoi	Lots of Towers of Hanoi
9	Medium-Hard	incoming	Paragliders and Aircraft
10	Hard	solitaire	Boring Solitaire
11	Hard	logging	Safe Logging
12	Hard	transport	Village Transportation

Call your program file: filename.c, filename.cpp, filename.java, or filename.py

For example, if you are solving Corona Virus Testing:

Call your program file: virus.c, virus.cpp, virus.java, or virus.py

Corona Virus Testing

filename: `virus`
Difficulty Level: `Easy`
Time Limit: `5 seconds`

Testing for Corona can be done individually, e.g., 100 people require 100 test kits. Alternatively, the test can be done in groups (pools), e.g., 100 people can be divided into five group of 20 people each and then using only one test kid per group. If one or more groups test positive, then individual tests are needed for each person in those group. So, for our example, five groups will need 5 test kits and let's say two groups test positive, so we would need additional 40 (2*20) test kits for a total of 45 (5+40) test kits.

The Problem:

Given the data for the two possible testing approaches, determine which approach will use fewer test kits.

The Input:

There is only one input line; it provides three integers: g ($2 \le g \le 50$), indicating the number of groups, p ($2 \le p \le 50$), indicating the number of people in each group, and t ($0 \le t \le g$), indicating how many groups tested positive (i.e., people in these groups need to be tested individually).

The Output:

Print 1 (one) if testing everyone individually will use fewer kits, 2 (two) if testing in groups will use fewer kits, and 0 (zero) if the two approaches use the same number of kits.

Sample Input	**Sample Output**
`40 3 38`	`1`
`10 20 2`	`2`
`20 10 18`	`0`

Presidential Election

filename: `elect`
Difficulty Level: `Easy`
Time Limit: `5 seconds`

Presidential election is coming up (November). In 2016, Clinton won the "*majority*" votes but Trump ended up with more "*electoral*" votes and won the race. (As a reminder, if a candidate receives more votes in a state, that candidate wins all the electoral votes for that state, i.e., electoral votes for a state are not divided proportionally based on the votes received by each candidate in that state.)

The Problem:

Election is in less than two months so let's predict the outcome! Given the voting data for each state, determine who wins the majority votes and who wins the electoral votes.

The Input:

The first input line contains an integer, n ($1 \le n \le 50$), indicating the number of states. Each of the next n input lines contains three integers, providing voting data for a state: e ($1 \le e \le 100$), indicating electoral votes for the state, v_1 ($0 \le v_1 \le 1000$), indicating votes for the first candidate, and v_2 ($0 \le v_2 \le 1000$; $v_2 \ne v_1$), indicating votes for the second candidate.

The Output:

Print 1 (one) if the first candidate wins both the majority votes and the electoral votes. Print 2 (two) if the second candidate wins both the majority votes and the electoral votes. Print 0 (zero) for all the other cases. Assume that if the total majority votes for the two candidates tie, neither one wins the majority. Similarly, if the total electoral votes for the two candidates tie, neither one wins the electoral.

Sample Input	**Sample Output**
`3`	`2`
`5 10 50`	
`15 30 60`	
`10 25 15`	
`3`	`0`
`5 48 50`	
`15 57 60`	
`10 25 15`	

Microwave Mishap

filename: `microwave`
Difficulty Level: `Easy`
Time Limit: `5 seconds`

Donald is very hungry. He grabs a TV dinner, puts it in the microwave, and enters 02:15 to cook it for 2 minutes and 15 seconds. Unknown to Donald, microwave takes these values as hours and minutes, i.e., microwave takes 02:15 as 2 hours and 15 minutes (not 2 minutes and 15 seconds). We need to tell Donald how much extra (i.e., the additional time) his food will be cooking. That is, we need to tell Donald that his food will cook 2 hours, 12 minutes, and 45 seconds more (longer) than what he was expecting!

The Problem:

Given the initial time in the form of *MM:SS*, where the input is actually taken as *HH:MM*, determine how much extra the food will be in the microwave. Provide this info in the form of *HH:MM:SS*.

The Input:

Input will consist of a single line in the form of *MM:SS*, representing what Donald has entered. *MM* and *SS* will be in the range of 00 and 59 (inclusive), but they both will not be 00 at the same time, i.e., the total time will be positive, i.e., input will not be 00:00.

The Output:

Output should contain a single line in the form of *HH:MM:SS*, indicating the additional time the food will cook in the microwave. Note that you need to print two digits for each part. Also note that *MM* and *SS* must be in the range of 00 to 59.

Sample Input	**Sample Output**
`05:00`	`04:55:00`
`13:37`	`13:23:23`
`00:10`	`00:09:50`

Food Display Arrangement

filename: `food`
Difficulty Level: `Medium`
Time Limit: `10 seconds`

Your friend, Thomas, is working on Food Display Arrangements (FDA). He has all the food lined up on a long row (table). His job requires that he arranges the FDA in an aesthetically pleasing manner. An FDA is aesthetically pleasing if all the food of the same type is grouped together, i.e., all the food of the same type are next to each other. Thomas can reorganize the FDA as follows: pick up all the food of one type and place it on either end of the table, i.e., place it at the beginning of the table or at the end of the table. Thomas wants to know the minimum number of reorganization steps needed to make the FDA aesthetically pleasing. Note that you don't need to tell him the specific steps, only the least number of steps.

The Problem:

Given a display of food by their types, determine the minimum number of times necessary to move all food of the same type to the end or the beginning of the display to ensure that all food of the same type is grouped together. Assume that the display can be extended at the ends to contain any amount of moved food.

The Input:

The first input line contains an integer, n ($1 \le n \le 100{,}000$), representing the number of food items in the display. The next input line contains n space separated integers, a_i ($1 \le a_i \le 1{,}000{,}000{,}000$), representing the id of the i^{th} food item in the display.

The Output:

Print the minimum number of times necessary to move all food of the same type to the beginning or the end to make the FDA aesthetically pleasing.

Sample Input	**Sample Output**
15 8 8 2 8 7 8 1 1 3 7 3 4 2 4 7	2
10 1 2 3 4 5 1 2 3 4 5	4

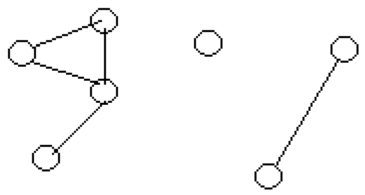
Explanation of the first Sample Input/Output: We can move all food of type 2 to either end and all food of type 7 to either end, for a total of 2 moves.

Making Connections

filename: `connect`
Difficulty Level: `Medium`
Time Limit: `5 seconds`

Given that everything is online these days, connectivity is a must. A computer network can be modeled as a graph, where each computer is a vertex and each direct network connection between pairs of computers is an undirected edge.

Consider the process of building a computer network. At the very beginning there will be n computers, with no connections between any of them. Then, as time goes on, pairs of computers are chosen, one pair at a time, and a direct network connection between them is added. In the middle of such a process, we might get the following graph modeling the current connections:



This network currently has three groups: the first group has 4 computers that can communicate with each other directly or indirectly, the second group consists of 1 computer by itself, and the third group has 2 computers that can communicate with each other. So, a group is each of the separate components of the graph.

We can define the average connectivity of a network as the sum of the group sizes squared, divided by the number of components. For the example graph shown above, the current connectivity equals $(4^2 + 1^2 + 2^2)/3 = 21/3 = 7$.

As a network is being built, the project manager would like to know the average connectivity of the network at that given snapshot of time. Write a program to handle the queries as the network is being constructed.

The Problem:

Given a network of n initially separate computers, along with a sequence of steps, where each step is either a pair of computers being connected or a query about the average connectivity as of that moment, process each step (i.e., connect the two computers or answer the query).

The Input:

The first input line contains two space separated integers: n ($1 \leq n \leq 10^5$), representing the number of computers, and m ($1 \leq m \leq 3\times10^5$), representing the total number of steps (each step being either connect two computers or a query on the average connectivity as of that moment). Assume that the computers are numbered 1 through *n*, inclusive.

Each of the following *m* input lines contains information about one operation, in the order that they occur. Each of these lines will start with a single integer, either 1 or 2, providing the step type. If the step type is 1, it means that a pair of computers is being connected with a direct connection. The value of 1 will be followed by *u* and *v* ($1 \le u, v \le n, u \ne v$), representing the pair of computers being connected with a direct connection. If the step type is 2, this is a query and no other information will be on that input line.

Note: It's possible that the input contains multiple direct connections for the same pair of computers; the extra direct connections between the same two computers do not have any effects. It's also possible that, at the end of the process, not all *n* computers are connected in the same component, i.e., there may be more than one component even at the end of the process.

The Output:

For each query, output the average connectivity of the network at that point in time as a fraction in lowest terms on a line by itself. Specifically, output two integers, *x* and *y*, with the character '/' in between, indicating that the average connectivity of the network at the time is *x* divided by *y* such that *x* and *y* share no common factors, e.g., 21/12 is not correct (should be 7/4).

Sample Input **Sample Output**

```
7 9
2
1 1 2
1 1 3
2
1 3 4
1 2 3
2
1 6 7
2
```

```
1/1
13/5
19/4
7/1
```

```
4 9
1 1 2
2
1 3 4
2
1 2 3
2
1 1 4
1 2 4
2
```

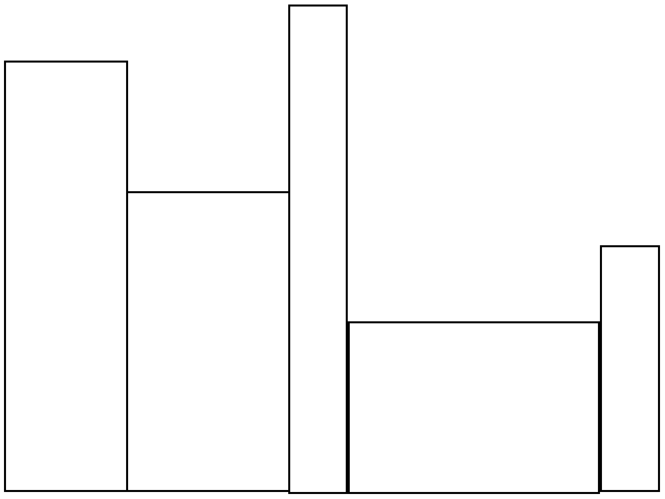
```
2/1
4/1
16/1
16/1
```

UCF Local Contest (Final Round) — September 19, 2020

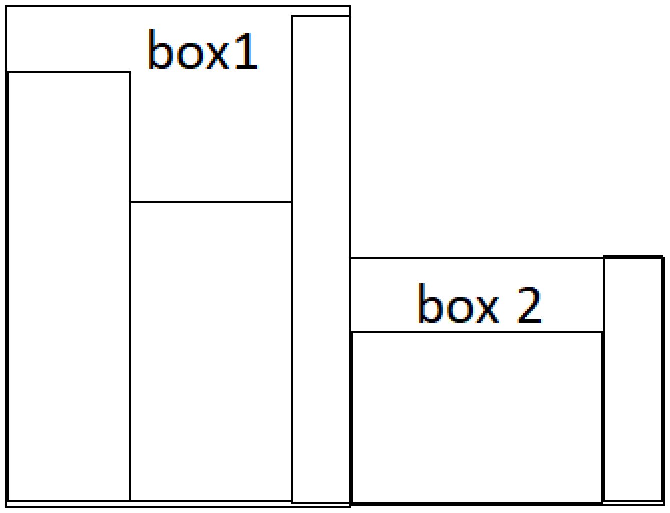
Boxing Books

filename: `books`
Difficulty Level: `Medium`
Time Limit: `5 seconds`

You have many books all in a row on a single long shelf and will need to box them up for a move. Each book is the same depth, but the books have different heights and widths. Here is a quick illustration of several books on a shelf:



Unfortunately, the moving company is limiting the number of boxes you can use and charges based on the height and width of each box. In addition, they are quite lazy and refuse to grab books from different areas of the shelf for each box, i.e., they only grab a sequence of adjacent books for each box. And, they don't want to change the orientation of any book (to save space) when placing it in a box. (Unfortunately, during the COVID-19 crisis, this was the only company you could find, so you'll have to deal with their idiosyncrasies for this problem.) Thus, all you are allowed to tell the moving company is how many adjacent books belong in each box, starting from the left part of the shelf. For example, if you were only allowed two boxes for the picture above, you could choose to put the first three books in the first box and the last two books in the second box as illustrated below:



The charge for each box is simply the height of the box times its width. As can be seen in the diagram, the height of each box is simply the maximum height of the books in that box and the width of the box is the sum of the widths of the books in the box.

Naturally, you would like to minimize the cost of boxing up the books given the ridiculous restrictions of the moving company. Write a program to do this.

The Problem:

Given a list of the dimensions of *n* books on a shelf, as well as the number of boxes, *k*, to put the books in, determine the minimum cost to box the books in exactly *k* non-empty boxes. Note that you must use exactly *k* boxes, not less and not more.

The Input:

The first input line contains two integers: n ($1 \leq n \leq 1000$), representing the number of books, and k ($1 \leq k \leq n$), representing the number of boxes to put the books in. The following *n* input lines contain the dimensions of the books, one book per line, in the order they appear on the shelf, from left to right. The ith of these input lines contains two integers, w_i ($1 \leq w_i \leq 10^6$) and h_i ($1 \leq h_i \leq 10^6$), representing (respectively) the width and height of the ith book on the shelf from the left.

The Output:

On a line by itself, print the minimum cost of boxing the books.

Sample Input **Sample Output**

Sample Input	Sample Output

5 2
3 10
4 7
1 12
6 4
1 6
	138

5 5
2 6
1 8
3 4
2 12
3 9
	83

Explanation of the first Sample Input/Output: Putting the first three books in one box and the last two books in the second box will result in the minimum cost.

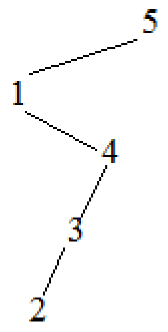
Box_1: `(3 + 4 + 1) * 12 = 96`
Box_2: `(6 + 1) * 6 = 42`
Total: `96 + 42 = 138`

Bad Tree

filename: `badtree`
Difficulty Level: `Medium-Hard`
Time Limit: `1 second`

Binary Search Trees are supposed to speed up searches for items but this happens only when the height of the tree is much less than the total number of values stored in the tree. And, in programming contests, judges unfortunately try to make the "worst case data". So invariably, in binary search tree problems, judges will make data where n nodes get inserted into a tree in a particular order so that the height of the tree is the worst case, $n - 1$.

Without loss of generality, let's assume that the n items to be inserted into a binary search tree are 1, 2, 3, …, n. For $n = 5$, if we insert the values in this order: 5, 1, 4, 3, and 2, we get the following binary search tree of height 4:



In fact, there are quite a few orderings of the first n positive integers that, when inserted into a binary search tree, create a binary search tree of height $n - 1$. Since you aspire to be a great judge one day for this contest, write a program to generate the k^{th} lexicographical permutation of the first n positive integers that, when inserted into a binary search tree in that order, generates a binary search tree of height $n - 1$.

The Problem:

Given a positive integer n, and another positive integer k, determine the k^{th} permutation, in lexicographical ordering, that when the items are inserted into a binary search tree in that order, generates a tree of height $n - 1$.

The Input:

There is only one input line; it contains two space separated integers: n ($1 \le n \le 100$), representing the number of nodes in the binary search tree, and k ($1 \le k \le 10^{18}$), where we desire the k^{th} lexicographical permutation of the first n integers which creates a binary search tree of height $n - 1$, when inserted in the order given in the permutation.

The Output:

Print the k^{th} lexicographical permutation of the integers 1 through n of the permutations which, when the values are inserted into a binary search tree, create a tree of height $n - 1$. Output the permutation on a single line, following each number in the permutation with a space. If no such permutation exists, output –1 instead.

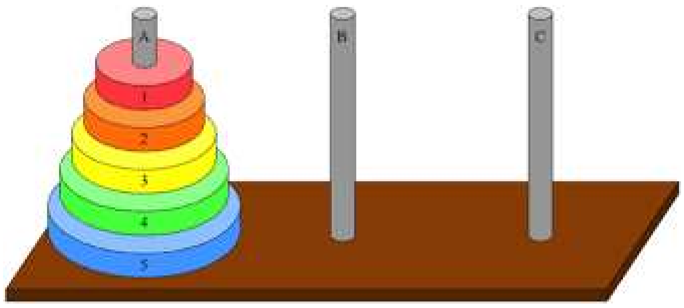
Sample Input	**Sample Output**
`5 12`	`5 1 4 3 2`
`4 2`	`1 2 4 3`
`6 1`	`1 2 3 4 5 6`
`3 50`	`-1`

UCF Local Contest (Final Round) — September 19, 2020

Lots of Towers of Hanoi

filename: `hanoi`
Difficulty Level: `Medium-Hard`
Time Limit: `10 seconds`

In the original Towers of Hanoi game, there are three towers of disks. In the initial configuration, all of the disks are on a single tower such that the radii of the disks are sorted in order from smallest to largest, from top to bottom. Here is a picture of the starting position of the original game:



The disks are typically labeled 1 to *n*, where *n* is the number of disks, with 1 representing the smallest disk and *n* representing the largest disk. In this diagram, the towers are labeled A, B and C. The goal of the game is to transfer all the disks to a different tower via a sequence of moves. On a single move, one may move the top disk of any tower onto the top disk of another tower, provided that the disk being placed on top is smaller than the disk below it. Alternatively, one may move the top disk of any tower onto an empty tower. To signify a move, it's good enough to simply specify a starting and ending tower. For example, in the above diagram, performing the moves A to B followed by A to C, followed by B to C would result in disks 3, 4, and 5 on tower A and disks 1 and 2 on tower C.

Luckily, the manufacturers of the game have splurged on the number of towers and have produced a set with *k* towers instead of just 3 towers. Due to the generality, the towers are labeled 1 through *k*, instead of A, B and C. The manufacturers have decided to include $n = \frac{k(k-1)}{2}$ disks with the set. The directions on the box containing the puzzle claim that the puzzle can be solved with $2(k\text{-}1)^2$ or fewer moves. Write a program to prove the manufacturer's conclusion.

The Problem:

Given the value of *k*, the number of towers in the Towers of Hanoi game, the number of the starting tower of all the disks, *s,* and the number of the ending tower of all the disks, *e*, create a sequence of at most $2(k\text{-}1)^2$ moves that solves the puzzle.

The Input:

The first and only input line contains three integers: k ($3 \le k \le 1000$), representing the number of towers in the game, s ($1 \le s \le k$), representing the starting tower for the $\frac{k(k-1)}{2}$ disks, and e ($1 \le e \le k$, $e \ne s$), representing the ending tower for the $\frac{k(k-1)}{2}$ disks.

The Output:

The first line of output should contain a single positive integer, m, number of moves in your solution. The following m lines should contain the contents of each move, in the sequence they are made to solve the puzzle. For each move, simply output two space separated integers, a and b ($a \ne b$), representing the starting and ending towers, respectively, for that particular move.

NOTES:

- There are many possible solutions for most puzzles. Any valid solution will be accepted.
- You do not have to provide the optimal solution; your solution will be accepted as long as it is valid and has $2(k\text{-}1)^2$ or fewer moves.

Sample Input **Sample Output**

3 1 2	7
	1 2
	1 3
	2 3
	1 2
	3 1
	3 2
	1 2

3 1 3	7
	1 3
	1 2
	3 2
	1 3
	2 1
	2 3
	1 3

UCF Local Contest (Final Round) — September 19, 2020

Paragliders and Aircraft

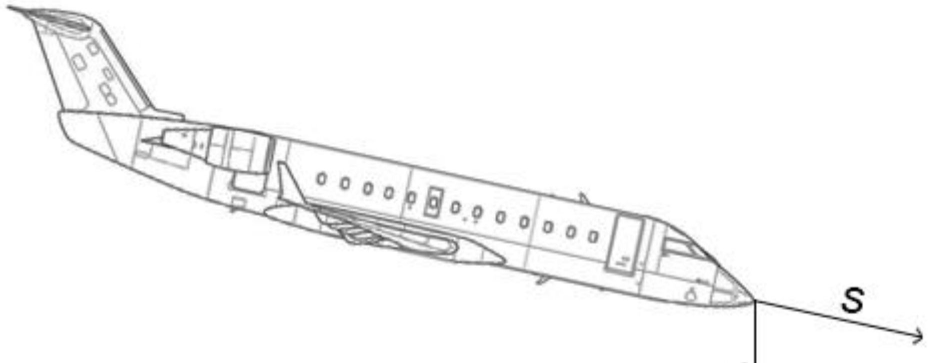
filename: `incoming`
Difficulty Level: `Medium-Hard`
Time Limit: `5 seconds`

A friend of yours is a paraglider pilot who flies at Tiger Mountain outside of Seattle, WA. The site is popular and, unfortunately, also in the landing approach patterns of some flights heading to Sea-Tac airport or Boeing field. This could obviously lead to dangerous situations involving high-flying paragliders and low-flying jets. Your friend, knowing you're a code guru, has asked for your help in developing an early warning system to let paraglider pilots know when jet traffic might intrude upon their airspace.

You've researched and found a web API that can give you real-time flight data on nearby air traffic, and another that lets you send text messages to the pilots. Now all you have to do is to figure out which flights could cause a problem.

The Problem:

Treat the airspace as a 3D coordinate space measured in feet. (This is reasonable because, for small distances, you can interpret latitude and longitude as coordinates on a plane.) You can assume the paragliders will stay in a 3D cylinder with a center, a radius, and lower and upper altitude bounds. Flight data will contain each aircraft's position, altitude, heading, velocity, and descent rate. Heading is in degrees where 0 degrees is along the positive *x* axis and 90 degrees is along the positive *y* axis. Airspeed is in feet per second, along vector *s* in the diagram; descent is along vector *d*.



If an aircraft's flight path will intersect the bounded cylinder, compute the entry and exit time in seconds from the current time, and output a message with those times. Otherwise, output a message to ignore the aircraft.

The Input:

The first input line describes the airspace; it contains five floating point numbers: the (x_c, y_c) coordinates of the center of the cylinder in feet ($-50{,}000 \le x_c, y_c \le 50{,}000$), the radius ($r$) of the cylinder in feet ($1 \le r \le 10{,}000$), and the lower (l) and upper (u) bounds of the cylinder in feet ($0 \le l < u \le 10{,}000$).

The second input line contains a single integer, n ($1 \le n \le 100$), indicating the number of aircraft to process.

The next n input lines describe the incoming aircraft. Each line provides data about an aircraft and contains seven numbers: an integer flight number f ($1{,}000 \le f \le 9{,}999$), the ($x_a$, y_a) position of the aircraft ($-200{,}000 \le x_a, y_a \le 200{,}000$), the heading ($h$) of the aircraft in degrees ($0 \le h < 360$), the altitude (a) in feet ($1{,}000 \le a \le 35{,}000$), the speed ($s$) in feet per second ($100 \le s \le 10{,}000$), and the descent rate (d) in feet per second ($0 \le d \le s$). All inputs for the incoming aircraft, other than the flight number, are given in floating point.

Assume that the aircraft will never start inside the cylinder. Also assume that, if an aircraft enters the paragliders' cylinder, it will be within the cylinder at multiple points.

The Output:

If the aircraft will not enter the cylinder, print the message:
```
Flight f is safe.
```
where f is the flight number.

If the aircraft will intersect the cylinder, print the message:
```
Incoming! Flight f enters at t0 and exits at t1.
```
where f is the flight number, t_0 is the time in seconds when the aircraft will enter the cylinder, and t_1 is the time in seconds when the aircraft will exit the cylinder. Print the times rounded to two decimal places (i.e., the time 0.274 would be printed as 0.27; the time 0.275 would be printed as 0.28). If the aircraft grazes the cylinder (with an error of 10^{-6}), it is considered to have entered and exited.

Sample Input 1:

```
0.0 0.0 1000.0 0.0 10000.0
2
1200 -5000.0 0.0 0.0 7500.0 5000.0 500.0
2400 -5000.0 0.0 90.0 7500.0 5000.0 500.0
```

Sample Output 1:

```
Incoming! Flight 1200 enters at 0.80 and exits at 1.21.
Flight 2400 is safe.
```

Sample Input 2:

```
-1000.0 1000.0 2000.0 1000.0 10000.0
1
1000 -6000.0 1000.0 0.0 12000.0 4472.0 2000.0
```

Sample Output 2:

```
Incoming! Flight 1000 enters at 1.00 and exits at 1.75.
```

Aircraft image credit:

https://www.norebbo.com/2015/04/bombardier-canadair-regional-jet-200-blank-illustration-templates/

Boring Solitaire

filename: `solitaire`
Difficulty Level: `Hard`
Time Limit: `1 second`

During quarantine, Stacking-Knightro (SK) made a new card game. SK respects social distancing and, since SK didn't have other people around, the game is a type of solitaire. SK will shuffle a deck of cards. Then, at every step, SK takes the card at the top of the deck. When taking the card, SK will place the card on top of one of the existing piles of cards or will make a new pile with this card as the top. SK can only place a card on top of a pile if the value of the card is not less than the card at the top of the pile. The goal of the game is to minimize the number of piles when the deck is empty.

Stacking-Knightro started getting pretty good "score" before too long, so the game became pretty boring once SK figured out the "trick". SK wants to know how many different starting card arrangements will end up with at most K piles if the game is played optimally at each step, i.e., each card is placed such that it will result in the best outcome.

The Problem:

We have a deck of cards with values 1 through V and a number of suits S, i.e., there are "(V × S)!" arrangements of the cards in the deck (note that "!" refers to the factorial of an integer). Using the game description above and a desired maximum of K piles, you are to determine how many different starting card orders will end up with at most K piles. Since this value can be large, print the output mod 1,000,000,007.

The Input:

There is only one input line; it contains three integers: V ($1 \leq V \leq 100$), representing the number of card values (values are 1 through V), S ($1 \leq S \leq 10{,}000$), representing the number of suits, and K ($1 \leq K \leq V \leq 100$), representing the maximum number of piles allowed.

The Output:

Print how many different starting card orders will end up with at most K piles.

Sample Input **Sample Output**

2 2 1	4
4 3 3	763937568
3 1 3	6

Explanation of the first Sample Input/Output:

There are two card values, two suites, and one pile. There are four card arrangements to win the solitaire:

1. 1_{s1} 1_{s2} 2_{s1} 2_{s2}
2. 1_{s1} 1_{s2} 2_{s2} 2_{s1}
3. 1_{s2} 1_{s1} 2_{s1} 2_{s2}
4. 1_{s2} 1_{s1} 2_{s2} 2_{s1}

Safe Logging

filename: `logging`
Difficulty Level: `Hard`
Time Limit: `5 seconds`

Consider a tree (not necessarily a binary tree). We will use the term *neighbor* to refer to the parent and children of a node, i.e., nodes with direct connection to the node. Some of the nodes in the tree have a log. All logs have two colors: Red at the top-half and Black at the bottom-half.

Our strong friend, Lumber-Knight (LK), needs to cut all these logs in half. When a log is cut, the Red (top) part falls into a neighbor but the neighbor cannot contain a Black part. This means each node will eventually contain only one Black log or several (zero or more) Red logs. Note that the Black logs do not move from node to node. Note also that when a Red log falls into a neighbor, it does not move anymore.

Lumber-Knight, besides being strong, is also very stylish. He does not want any node with Black log to have two (or more) neighbors with Red logs. Your job is to determine all different ways the Red logs can be moved to (cut and fall into) neighbors to meet the LK's style. Since the number of ways can be quite large, output the number of ways modulo 1,000,000,007.

The Problem:

Given a tree description and locations of the logs, determine the number of ways in which Lumber-Knight can cut the logs and move the Red logs into neighbors such that no node with Black log has two (or more) neighbors with Red logs.

The Input:

The first input line contains an integer, n ($1 \leq n \leq 100{,}000$), representing the number of nodes in the tree (assume the tree nodes are numbered 1 to n, with 1 being the tree root). Each of the following $n - 1$ input lines contains two integers, a_i and b_i ($1 \leq a_i, b_i \leq n$; $a_i \neq b_i$), representing a direct connection between two nodes.

Following the tree description will be an integer, m ($1 \leq m \leq n$), representing the number of tree nodes with a log. The following input line contains m distinct integers (i.e., no duplicates), representing the tree nodes that contain a log.

The Output:

Print a single integer representing the number of ways the m logs can be cut and the Red part moved to a neighbor such that no node with a Black log has two (or more) neighbors with Red part. If it is not possible to accomplish the task, print 0 (zero).

Two cutting sequences are considered different if there is a node with different contents in the two sequences, e.g., the node is empty in one case and contains Red log(s) in the other case.

Sample Input	**Sample Output**

`5`
`1 2`
`2 3`
`3 4`
`4 5`
`3`
`1 3 4`
	`1`

`6`
`1 2`
`2 3`
`3 4`
`3 5`
`5 6`
`2`
`2 5`
	`2`

UCF Local Contest (Final Round) — September 19, 2020

Village Transportation

filename: `transport`
Difficulty Level: `Hard`
Time Limit: `5 seconds`

This set started with a problem on presidential election and will end with another problem on the election!

Our good friend, Democracy-Forever (DF), wants to make sure everyone will have a chance to vote. DF has found out that there are a few villages, the first village has a Voting Station, the remaining villages do not have voting stations and do not currently have connection (roads) to get to the first village (the only Voting Station). He has received government funding to build roads for all the villagers to get to the station (first village) and vote. Hopefully, the funding is enough!

Democracy-Forever has determined the cost of constructing a one-way road between certain pairs of villages. DF needs to decide which roads to build to be sure everyone will be able to get to the Voting Station (directly or indirectly). It is, of course, desirable to have as much money left in DF's pocket as possible at the end!

What complicates the decision process (which roads to build) is that if a road originates from a village, the Leader of the village wants some Royalty; the Leader won't allow the road construction otherwise. The Royalty is not, unfortunately, a fixed amount; it is based on a future forecast. More specifically, it is a factor of how much DF will have in his pocket at the end. For example, the solutions (needed roads) for the first two Sample Input are shown on the last page (Node 1 is the Voting Station and the remaining nodes represent the remaining villages). For the first example:

- The original budget is \$100.
- It costs \$15 to build the road from 2 to 1 and the Leader wants 4 times what is in DF's pocket at the end (\$10). So, the total cost for that road is \$15 + (4 × \$10) = \$55.
- It costs \$5 to build the road from 3 to 1 and the Leader wants 3 times what is in DF's pocket at the end, so the total is \$5 + (3 × \$10) = \$35.
- The two roads will get all the villagers to the Voting Station. The total expenditure is \$55 + \$35 = \$90.
- The money left for DF at the end is \$100 – \$90 = \$10.

The Problem:

Democracy-Forever would like to know, given the budget (government funding) for the roads and the various cost associated with the road construction, the maximum amount he can pocket at the end. DF's primary concern is, of course, for all the villagers to be able to get to the Voting Station.

The Input:

The first input line contains an integer, M ($1 \leq M \leq 1{,}000{,}000{,}000{,}000$), representing the funding for the project, i.e., the budget (dollar amount).

The second input line contains two integers: V ($2 \leq V \leq 1{,}000$), representing the number of villages (Node 1 is the Voting Station and Nodes 2–V are the remaining villages), and R ($1 \leq R \leq 1{,}000$; $R \geq (V - 1)$), representing the number of roads that can be constructed.

Each of the following R input lines contains a road description. Each line contains four integers: b ($1 \leq b \leq V$), representing the origin for a road, e ($1 \leq e \leq V$; $e \neq b$), representing the destination for a road, p ($0 \leq p \leq 1{,}000$), representing the cost for the road construction, and f ($0 \leq \text{f} \leq 1{,}000$), representing the Royalty factor for the road.

Assume that, if all the roads are constructed, all the villagers will be able to get to the Voting Station (directly or indirectly).

The Output:

Print the maximum amount Democracy-Forever can have in his pocket at the end. If the funded budget is not enough to build all the needed roads, print the value 0 (zero).

Print the answer as a floating point. Answers within 10^{-7} absolute or relative of the judge output will be considered correct.

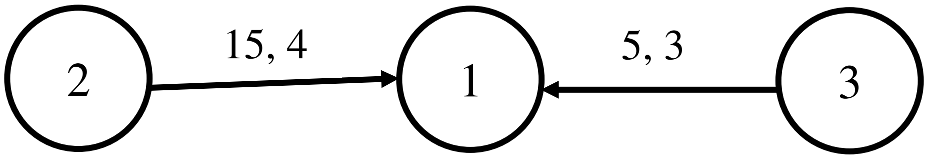
Sample Input	**Sample Output**

`100`
`3 5`
`1 2 0 7`
`2 1 15 4`
`2 3 12 5`
`3 1 5 3`
`3 2 20 2`
	`10.0`

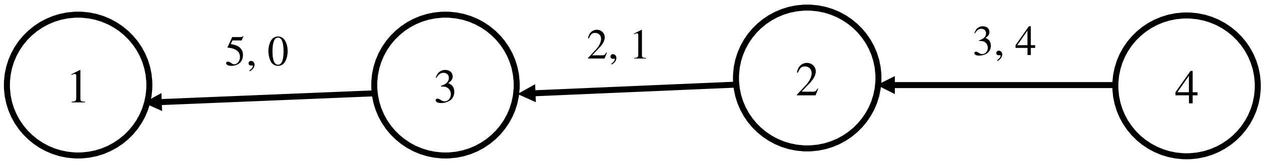
`13`
`4 4`
`1 4 9 10`
`2 3 2 1`
`3 1 5 0`
`4 2 3 4`
	`0.5`

`5`
`3 2`
`2 1 100 50`
`3 1 200 30`
	`0.0`

Solution to the first Sample Input:



Solution to the second Sample Input:


1
5, 0
3
2, 1
2
3, 4
4