IPSC 2008

problems and sample solutions

Perfect rhyme

Authors

Problemsetter: Michal Forišek
Task preparation: Milan Plžík, Michal Forišek

Problem statement

A perfect rhyme is not a crime,
it is something that exceeds time,
a bit of science, a piece of art,
soft as a pillow, sharp as a dart.

Everyone tried it, but only few chosen ones succeeded. It is a hard task with an unclear path, but a famous end – should you reach it. Many compare it to finding the Holy Grail, or even to finding Waldo. The task is to find a perfect rhyme.

Problem specification

Given is a wordlist L, and a word w. Your task is to find a word in L that forms a perfect rhyme with w. This word u is uniquely determined by these properties:

- It is in L.
- It is different from w.
- Their common suffix is as long as possible.
- Out of all words that satisfy the previous points, u is the lexicographically smallest one.

Notes

A prefix of a word is any string that can be obtained by repeatedly deleting the last letter of the word. Similarly, a suffix of a word is any string that can be obtained by repeatedly deleting the first letter of the word.

For example, consider the word `different`.

This word is both its own prefix and suffix. Its longest other prefix is `differen`, and its longest other suffix is `ifferent`. The string `rent` is its yet another, even shorter suffix. The strings `eent` and `iffe` are neither prefixes nor suffixes of the word `different`.

A word u is lexicographically smaller than a word v, if either u is a prefix of v, or if i is the first position where they differ, and the i-th letter of u is earlier in the alphabet than the i-th letter of v.

For example, `dog` is smaller than `dogs`, which is smaller than `dragon` (because `o` is less than `r`).

Input specification

The input file consists of two parts. The first part contains the wordlist L, one word per line. Each word consists of lowercase English letters only, and no two words are equal.

The first part is terminated by an empty line.

The second part follows, with one query word w per line.

Output specification

For each query in the input file output a single line with its perfect rhyme. The output must be in lowercase.

Example

input

```
perfect
rhyme
crime
time

crime
rhyme
```

output

```
time
crime
```

In the second test case, there were two candidates that had an equally long common suffix (crime and time), the lexicographically smaller one was chosen.

Solution

First of all, a brute force solution that answers a query by traversing the entire wordlist should work fine if the number of queries is small. This was enough to solve the easy input.

For the hard input, such a program probably would not finish in time until the end of the contest, thus we need to find a solution that's much faster.

We will first explain a solution that is fast enough and easy to implement.

We will start by reversing all words in our wordlist and sorting it. Now, words that rhyme are close in the sorted order, and the more they rhyme, the closer they are. More precisely, words with the same first K characters (which were originally the last K characters) always form a contiguous subsequence.

Now whenever processing a new query, we can easily find its place in the wordlist using binary search. Now, we can take a look on the words just before and after our position. No other word can have a longer common prefix with our word. Take the longer of these two prefixes. Now we can simply traverse the range of words that share it, and pick the first one lexicographically. (Note that in this step we compare the originals, not the reversed words.) As the wordlist contains English words, this range is never too long.

One solution with a better worst case time complexity is to insert the reversed words from the wordlist into a trie, process the trie bottom up so that in each vertex we know the lexicographically smallest of the words that end in its subtree, and then to look for the reverse of a query word in that trie.

Quiz

Authors

Problemsetter: Michal Forišek
Task preparation: Milan Plžík, Michal Forišek

Problem statement

In ancient Rome people used to say: "Historia magistra vitae." (History is the teacher of lives.) Knowing the history helps us to correctly decide in present, to avoid dangerous paths and sometimes it can even give us advantage against our opponents.

By taking part in this practice session, you'll gain such an advantage against teams that skip it – by getting to know the history of IPSC. Towards this end we prepared a small quiz for you.

Input specification

In the input file, each paragraph represents one quiz question. The paragraphs are separated by blank lines

Output specification

For each question, output a single line with the correct answer.
There are two types of answers:

- Numeric answers: The line must contain exactly one decimal number.
- Verbal answers: The line must contain one string of UPPERCASE English letters.

You have to answer all questions correctly in order to solve the given input. Note that exact spelling matters. Read carefully, good luck, and have fun while learning about the famous history of IPSC!

Example

input	output
How many letters does "IPSC" have? The first letter in Greek alphabet?	4 ALPHA

Solution

Here the solution was quite simple, all the answers could be found on the IPSC website. Googling with site:ipsc.ksp.sk could ocassionaly speed up solving, as could rewriting the URL instead of clicking. But the main thing to focus on was to read the questions carefully, and take care to format the answers properly.

Railroad map

Authors

Problemsetter: Lukáš Poláček
Task preparation: Lukáš Poláček

Problem statement

The Slovak national railroad company has recently built new tracks. They want to update their railroad map according to these changes. But they want the map to be as simple as possible. So they decided to remove from the map all the stations that have exactly two other direct connections to other stations (i.e., a single railroad passing through the station).

Problem specification

You will be given the complete map of Slovak railroads. It consists of railway stations numbered from 1 to N, and railroad segments between some pairs of these stations. For each railroad segment we are given its length.

Your task is to remove all such stations that are directly connected with exactly two other stations, and output the new map. The new map must contain correct distances between the remaining stations.

Input specification

The first line of the input file contains an integer T specifying the number of test cases. Each test case is preceded by a blank line.

Each test case begins with a line with two integers N and M. The number N denotes the number of stations and M is the number of railroad segments. M lines follow, each with 3 integers a, b, and c ($1 \leq a, b \leq N$) specifying that there is a railroad segment of length c connecting stations a and b.

You can assume that in each test case there is a path between every two stations. Also, there will always be at least 2 stations that are not directly connected to exactly two other stations.

Output specification

For each test case, the output shall consist of multiple lines. The first line shall contain a positive integer K – the number of railroads on the simplified map. Each of the next K lines shall contain three integers a, b, and c stating that there is a railroad of length c between stations a and b on the simplified map.

Print a blank line between outputs for different test cases.

Example

input

```
2

3 2
1 2 1
2 3 1

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

output

```
1
1 3 2

2
1 2 1
2 2 6
```

In the first case we removed station 2 beacuse it had exactly 2 direct connections.

In the second case we removed stations 3 and 4. We see that there is now a railroad from station 2 back to itself.

Solution

The input for this problem is a weighted graph. There were two quite obvious approaches that could be used for this problem.

One possible solution is to implement the procedure directly, each time we find a vertex of degree 2, remove it along with its both edges, and add a new edge instead. To implement this solution efficiently, we need a data structure that can store a dynamic graph that can have duplicate edges. (There are no such edges in the input, but they can be created during this process.) A simple (but not optimal) data structure is a multiset containing all edges.

The other solution is to use depth-first search (DFS). Start in a vertex that does not have degree 2. Now, each time the DFS enters a vertex with degree 2, we know that it will travel the entire path of such vertices. Each time the DFS enters a vertex that has a degree other than 2, we can output the edge of the new graph we just traversed.

To avoid special cases (two large vertices connected by an edge), it is best to preprocess the input graph – add a new vertex in the middle of each edge.

Army strength

Authors

Problemsetter: Michal Forišek
Task preparation: Peter Perešíni

Problem statement

The next MechaGodzilla invasion is on its way to Earth. And once again, Earth will be the battleground for an epic war.

MechaGodzilla's army consists of many nasty alien monsters, such as Space Godzilla, King Gidorah, and MechaGodzilla herself.

To stop them and defend Earth, Godzilla and her friends are preparing for the battle. (After this contest, you can visit `http://www.atari.com/godzilla/` to find out more about them.)

Problem specification

Each army consists of many different monsters. Each monster has a strength that can be described by a positive integer. (The larger the value, the stronger the monster.)

The war will consist of a series of battles. In each battle, the weakest of all the monsters that are still alive is killed.

If there are several weakest monsters, but all of them in the same army, one of them is killed at random. If both armies have at least one of the weakest monsters, a random weakest monster of MechaGodzilla's army is killed.

The war is over if in one of the armies all monsters are dead. The dead army lost, the other one won.

You are given the strengths of all the monsters. Find out who wins the war.

Input specification

The first line of the input file contains an integer T specifying the number of test cases. Each test case is preceded by a blank line.

Each test case starts with line containing two positive integers N_G and N_M – the number of monsters in Godzilla's and in MechaGodzilla's army. Two lines follow. The first one contains N_G positive integers – the strengths of the monsters in Godzilla's army. Similarly, the second one contains N_M positive integers – the strengths of the monsters in MechaGodzilla's army.

Output specification

For each test case, output a single line with a string that describes the outcome of the battle.

If it is sure that Godzilla's army wins, output the string "`Godzilla`".

If it is sure that MechaGodzilla's army wins, output the string "`MechaGodzilla`".

Otherwise, output the string "`uncertain`".

Example

input

```
2

1 1
1
1

3 2
1 3 2
5 5
```

output

```
Godzilla
MechaGodzilla
```

In the first test case, there are only two monsters, and they are equally strong. In this situation, MechaGodzilla's monster is killed and the war ends.

In the second test case, the war will consist of three battles, and in each of them one of Godzilla's monsters dies.

Solution

To solve the easy input simulating the war was enough. For the hard input it was better to realize that the order in which monsters are killed is simply the order according to their increasing strength.

Denote S_g the maximum strength of a monster from Godzilla's army and S_m the maximum strength of a MechaGodzilla's monster. If $S_g \geq S_m$, all MechaGodzilla's monsters will die before Godzilla's strongest one and thus Godzilla wins. Otherwise, MechaGodzilla wins for the same reason.

Breaking in

Authors

Problemsetter: Peter Košinár
Task preparation: Peter Košinár, Lukáš Poláček

Problem statement

Mayco has recently been hired as a security consultant for a well-known software company. At the moment, he's working on his first assignment – trying to determine which of the company's servers would be the best targets for potential attackers. It is a bit difficult, though, because some of the servers "trust" some of the others. If an attacker compromises a server, he or she can also freely access all servers that trust it (and servers that trust them, and so on).

By definition, the importance of a server S is the number of servers the attacker would be able to access if he compromised S. The most important servers are those with the highest importance. (Note that there can be more than one most important server. This is also illustrated in the example below.)

Problem specification

The network consists of N computers, numbered 1 to N, inclusive. The trust between computers is described by M ordered pairs (A, B) of numbers, denoting that computer A trusts computer B. The trust is not assumed to be mutual – i.e., if a computer A trusts computer B, it does not necessarily imply that computer B trusts computer A.

Input specification

The first line of the input file contains an integer T specifying the number of test cases. Each test case is preceded by a blank line.

Each test case starts with a line containing the numbers N and M. Each of the following M lines contains two integers, A and B, denoting that computer A trusts computer B.

Output specification

For each test case, the output shall contain one line with the numbers of all of the most important servers. The numbers must be listed in increasing order and separated by single spaces.

Example

input

```
2

5 4
3 1
3 2
4 3
5 3

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

output

```
1 2
4
```

In the first test case, compromising one of the servers 1 and 2 gives us access to the entire network.

In the second test case, compromising server 4 gives us access to servers 1, 2, and 3. All other servers are less important.

Solution

The network can be represented as a directed graph with N nodes and M arcs. The most important servers correspond to nodes v from which the largest number of nodes is reachable by a directed path.

The most straightforward approach is to use, for example, the Floyd-Warshall algorithm to determine for each node the complete list of nodes reachable from it. Then, finding the most important node(s) is a matter of simple counting. This yields an algorithm with $O(N^3)$ time complexity.

Unfortunately, this is too slow for the hard input case. A simple improvement is to replace the Floyd-Warshall algorithm by N simple searches (be it depth-first or breadth-first ones), starting from each node of the graph. If, instead of adjacency matrix, we store the list of directly reachable nodes in a list for each vertex, the the resulting complexity will be $O(NM)$. This is sufficient for our purposes, as the graphs given in the input files are relatively sparse.

Comparison mysteries

Authors

Problemsetter: Michal Forišek
Task preparation: Michal Forišek

Problem statement

For beginners in programming it often comes as a great surprise when numbers inside a computer refuse to behave the same way numbers in mathematics do. In this task, we will investigate two such cases.

The ideas needed to solve this task are neither platform-specific, nor language-specific. However, due to many subtle differences between various programming languages the best way to state this task was to pick a single language. In this case that language will be Java.

We will be using numeric variables only, and only the following six basic types:

- `byte`, a signed integer from -128 to 127, inclusive
- `short`, a signed integer from -32768 to 32767, inclusive
- `int`, a signed integer from -2147483648 to 2147483647, inclusive
- `long`, a signed integer from -9223372036854775808 to 9223372036854775807, inclusive
- `float`, a 32-bit floating point number (24 bits sign+mantissa, 8 bits exponent)
- `double`, a 64-bit floating point number (53 bits sign+mantissa, 11 bits exponent)

For each of these types, the Scanner class has a corresponding method (e.g., `nextByte` for `byte`) that scans the next token of the input as the given type.

Easy data set

Obviously, in mathematics the only solution to $x = -x$ is zero. Now consider the following Java code:

Java code template

```
import java.util.*;
public class C1 {
  public static void main(String args[]) {
    Scanner sc = new Scanner(System.in); sc.useLocale(Locale.US);
    ?????  x = sc.?????();
    System.out.println( (x==-x) + " " + (x!=0) );
  }
}
```

We would like you to show that there are cases when a non-zero number is equal to its own negation. Choose the type of the variable x and provide input to this program so that it outputs "`true true`".

Submit your answer as a text file with two lines. In the first line, write the type of the variable x, in the second line a string that shall be used as the input for your program.

An example submission follows. This submission is syntactically correct, but it does not produce the desired output.

Your submission

```
short
47
```

The corresponding Java code

```
import java.util.*;
public class C1 {
  public static void main(String args[]) {
    Scanner sc = new Scanner(System.in); sc.useLocale(Locale.US);
    short x = sc.nextShort();
    System.out.println( (x==-x) + " " + (x!=0) );
  }
}
```

Its output on your input

```
false true
```

Hard data set

An even more obvious fact is that whenever $x = y$ and $y = z$, we must have $x = z$ as well. Now consider the following Java code:

Java code template

```
import java.util.*;
public class C2 {
  public static void main(String args[]) {
    Scanner sc = new Scanner(System.in); sc.useLocale(Locale.US);
    ?????  x = sc.?????();
    ?????  y = sc.?????();
    ?????  z = sc.?????();
    System.out.println( (x==y) + " " + (y==z) + " " + (x==z) );
  }
}
```

We would like you to show that there are cases when equality is not transitive. Choose the types of the variables x, y, and z and provide input to this program so that it outputs "`true true false`".

Submit your answer as a text file with six lines. In the first three lines, write the types of the variables x, y, and z. In the second three lines write the strings that shall be used as the input for your program.

An example submission follows. This submission is syntactically correct, but it does not produce the desired output.

Your submission

```
short
short
int
47
47
47
```

The corresponding Java code

```
import java.util.*;
public class C2 {
  public static void main(String args[]) {
    Scanner sc = new Scanner(System.in); sc.useLocale(Locale.US);
    short x = sc.nextShort();
    short y = sc.nextShort();
    int z = sc.nextInt();
    System.out.println( (x==y) + " " + (y==z) + " " + (x==z) );
  }
}
```

Its output on your input

```
true true true
```

Applets

For easy experimentation we provide an applet for each of the subproblems, where you can set variable types and values, and let the applet produce the output for you.

- Applet for the easy data set: `C1applet.html`
- Applet for the hard data set: `C2applet.html`

Java documentation

This section is not strictly necessary to solve the problem. But in case someone feels the need to browse the Java documentation to check something out, we provide links to the online version of *The Java Language Specification, Third Edition* you might consider useful:

- Main page:
  `http://java.sun.com/docs/books/jls/third_edition/html/j3TOC.html`
- 4.2 Primitive Types and Values:
  `http://java.sun.com/docs/books/jls/third_edition/html/typesValues.html#4.2`
- 5.6 Numeric Promotions:
  `http://java.sun.com/docs/books/jls/third_edition/html/conversions.html#5.6`

Solution

Minus x equals x

For the easy data set, your task was to find a non-zero number that is equal to its negation.

The key trick is that for signed integer data types the set of possible values is asymetric. For example, the smallest possible `int` is -2^{31} and the largest is $2^{31}-1$. This leaves us with $2^{31}-1$ pairs $(x,-x)$, and two special values: 0 and -2^{31}.

The negation of -2^{31} would be 2^{31}. However, this does not fit into an `int`. More precisely, 2^{31} is one larger than the largest allowed value, and this overflows to the smallest allowed value: -2^{31} again.

(Essentially, all computations with b-bit integers are done modulo 2^b in the sense that results that differ by a multiple of 2^b will, due to overflow, be represented by the same bit pattern.)

Note that when comparing less-than-32 bit integer variables in Java, they get promoted to `int`s. This causes our solution to work only for `int`s and `long`s. Thus the only two solutions to this problem are: `int` $x = -2\,147\,483\,648$ and `long` $x = -9\,223\,372\,036\,854\,775\,808$.

Equality is not transitive

If comparing variables of different types, they are usually first both promoted to the larger of those two types. For example, when comparing a `short` and an `int`, the `short` is cast to an `int`, and then they are compared.

The problem is that sometimes there is no good choice – the two types can have incomparable ranges of possible values. In that case, the language designers have to pick one of them.

For example, this often happens when comparing an integer data type to a floating point one.

In Java and many other languages, floating point types get precedence. To quote the reference: If either operand is of type `double`, the other is converted to `double`. Otherwise, if either operand is of type `float`, the other is converted to `float`.

The problem here is that a `long` is a 64-bit integer, whereas a `double` only has 53 bits to store the digits. Thus there are values that can be stored in a `long` but can not be represented exactly in a `double`. For the same reason, we may lose precision when converting an `int` into a `float`.

This can cause two different `int`s to be converted into the same `float`. For example, the ints $1\,234\,567\,890$ and $1\,234\,567\,891$ will.

We can use this to construct a solution to the hard problem. One possible solution is: `int` $x = 1\,234\,567\,890$, `float` $y = 1\,234\,567\,890$, and `int` $z = 1\,234\,567\,891$. Both for `x==y` and `y==z` the `int` is converted to a `float`, and thanks to the precision loss equality holds. However, with `x==z` we are comparing two `int`s, and find that they differ.

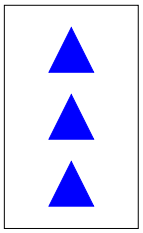
Discover all Sets

Authors

Problemsetter: Michal Forišek, Monika Steinová
Task preparation: Monika Steinová, Peter Perešíni, Michal Forišek

Problem statement

The game of Sets is a very popular card game (`http://en.wikipedia.org/wiki/Set_(game)`). The game is played with a special deck of cards, where each card has a unique picture on it. This is an example of such a card:



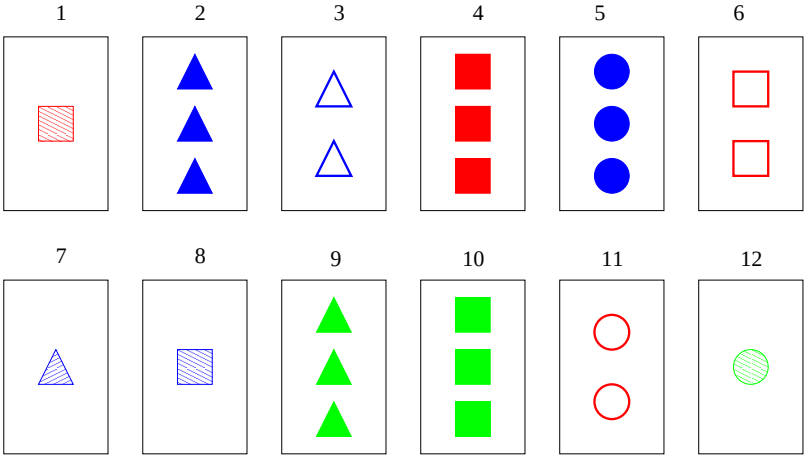
Each of the cards is uniquely identified by its four properties: the shape shown in the picture (square, triangle, circle), the count of these shapes (one, two, three), their color (red, green, blue), and the fill style (empty, striped or full).

For each combination of values there is exactly one such card in the deck. Thus the total number of cards in the deck of the original game is $3^4 = 81$.

The main concept of this card game is a **Set**. A Set consists of three cards such that for each property, they are either all the same, or all are different. In other words, three cards are **not** a Set if and only if you can say "Two of them are ... and one is not."

In the picture below, the cards 1, 4, and 6 form a Set: they are all red, all are squares, the counts are different, and so are the fill types.

The cards 2, 3, and 9 do not form a Set, for example because two of them are blue and one is not.



Problem specification

In this problem we will play a more general version of the game. There will be N card properties, and for each of these properties there will be M different values it can obtain. The deck will contain exactly one card for each possible combination of properties. Thus there will be M^N cards in the deck.

In this more general game, a Set consists of exactly M cards such that for **each** of the N properties the following holds: *Either all the cards have the same value of this property, or all of them have different values.*

You will be given the values N, M, a positive integer K, and a description of K different cards from the deck. Your task is to find the number of different Sets that can be created from the given cards. (Note that some of those Sets may overlap.)

Input specification

The first line of the input file contains an integer T specifying the number of test cases. Each test case is preceded by a blank line.

Each test case looks as follows: The first line contains three positive integer N, M and K giving the number of different properties of the cards, the number of different values of a single property, and the number of cards drawn.

Each of the next K lines describes one of these cards. The i-th of these lines ($1 \leq i \leq K$) contains exactly N numbers $p_{i,j}$ ($1 \leq j \leq N$), $1 \leq p_{i,j} \leq M$. Value $p_{i,j}$ is the value of the j-th property of the i-the card.

Output specification

For each test case output a single line with a single integer – the number of different Sets that can be made from the given cards.

Example

input

```
1

4 3 12
1 1 1 2
2 3 3 3
2 2 3 1
1 3 1 3
3 3 3 3
1 2 1 1
2 1 3 2
1 1 3 2
2 3 2 3
1 3 2 3
3 2 1 1
3 1 2 2
```

output

```
11
```

This example input corresponds to the image in the problem statement. The four properties are described in this order: shape (square, triangle, circle), count (one, two, three pieces), color (red, green, blue) and fill style (empty, striped, full).

These are the 11 different sets: (1,4,6), (1,7,12), (2,3,7), (2,6,12), (3,4,12), (3,5,8), (4,5,9), (5,11,12), (6,8,10), (7,10,11), and (8,9,11).

Solution

The straightforward solution of this problem is to try all possible sets of M cards and check whether they form a Set. This solution is very slow and thus we will not go into details.

A slightly better solution is to try all possible sets of $M-1$ cards. For each such a set S, it is clear whether cards in S can form a Set. If so, we can calculate the missing value of each property and thus compute the missing card that will create a Set. Note that such card is uniquely determined. It is enough to check whether it is among the K given cards, and for this, we can use a set or a hashmap.

Note that this solution counts each set M times, once for each missing card.

Both solutions can be improved by cutting the branches of the program in early stages when it is clear that the branch does not lead to a Set. This approach performs reasonably well on many inputs, but its performance is not guaranteed – and its time complexity is always at least linear in the number of Sets.

The sample solution of this problem uses this improvement, and the idea of the meet-in-the-middle approach for $M \geq 4$.

We can hash a set of cards to a NM-bit sequence of ones and zeroes that describes for each of the N properties which of the M values are represented in the set.

How can this help us? Suppose that we have two sets of cards C and D, with $|C|, |D| \geq 2$ and $|C| + |D| = M$. Do C and D form a Set together? The answer to this question is uniquely determined by their hashes.

For each set C of $X = \lfloor M/2 \rfloor$ cards that can form a Set we will determine its hash, and construct a hash table where we count the number of times each hash occured.

After the hash table is constructed we take all possible sets of cards of size $Y = \lceil M/2 \rceil$ that can form a Set. For each such set D: From the hash of D we can determine the hash of its matching set C. (The properties that are constant in D must be constant in C as well, with the same value. For the other properties we take the complement of the set of values.) From our hash table we know how many such sets C there are.

As in the previous solution, the resulting number of Sets need to be divided by $\binom{K}{\lceil M/2 \rceil}$, as we count each set that many times.

(The solution can be made even faster by counting each Set only once. Can you see how to do it?)

Expected road cost

Authors

Problemsetter: Michal Forišek
Task preparation: Michal Forišek, Tomáš Záthurecký

Problem statement

For many years, Absurdistan was famous for its camel caravans. However, the new Grand Vizier decided to make a radical step towards civilization – he will actually build some roads.

On the other hand, too many roads would unnecessarily confuse the local inhabitants. (That, and camels don't really like walking on roads.) Therefore the road network will, for now, have to be minimal, just enough to connect all the villages.

Preliminary investigations were made. For some pairs of villages we know a lower and an upper estimate of the cost of building a direct road between these two villages.

The Grand Vizier has made you personally responsible for preparing the budget. You know that after you prepare the budget, the exact cost for each potential road will be computed, and the best set of roads will be selected.

Problem specification

There are N villages in Absurdistan. There are M pairs of villages such that we can build a direct road that connects them. For each such road i, we know the minimal cost l_i and the maximal cost u_i.

The builders will first find the exact cost for each road, and then select the cheapest set of roads that connects all the villages.

Assuming that for each road its true cost is a real-valued random variable with uniform distribution over the interval $[l_i, u_i]$, compute the expected cost of the road network.

Notes

All roads are bidirectional.

The roads are built in such a way that they don't intersect anywhere, using bridges if necessary. Thus in order to connect N villages you have to build at least $N - 1$ roads.

Input specification

The first line of the input file contains an integer T specifying the number of test cases. Each test case is preceded by a blank line.

Each test case consists of several lines. The first line of a test case contains two integers N and M, giving the number of villages and the number of possible roads, respectively. The villages are numbered from 0 to $N - 1$. M lines follow, each of them describes one possible road. Each description consists of four integers: the villages connected x_i and y_i, and the minimum and maximum cost l_i and u_i.

Villages are numbered starting from zero. All costs are non-negative. The input graphs are somewhat special, it may be worth your while to examine the input data before attempting to solve the task.

Output specification

For each test case, the output shall contain one line with the expected cost of the cheapest road network. This value is always rational, and you are to output it as the simplest possible fraction.

Print the fraction as A/B, with no spaces. Even if $B = 1$, print the denominator.

If for some test case it is not possible to build any road network that connects all villages, output -1 instead.

Examples

input

```
4

3 2
0 1 0 9
1 2 10 11

4 2
0 1 10 11
1 2 10 12

3 3
0 1 0 1
1 2 2 2
0 2 3 3

3 3
0 1 0 1
1 2 0 1
0 2 0 1
```

output

```
15/1
-1
5/2
3/4
```

In the first test case, we have to build both roads. Their expected costs are $9/2$ and $21/2$, which gives a total of $30/2 = 15/1$.

In the second test case, we can not connect village 4 to the other villages.

In the third test case, the two cheapest roads are always $1 - 2$ and $2 - 3$, with total expected cost $1/2 + 2 = 5/2$.

The fourth test case illustrates that the builders first find out the exact road costs, and only then pick the best set. In this case, we generate three random numbers from $\langle 0, 1 \rangle$, and the result is the expected sum of the smaller two.

Solution

The problem statement asks us to compute the expected cost of the minimum spanning tree of the given graph.

The graphs in the easy input were relatively small, and simpler to solve due to the fact that all the edge costs were chosen uniformly from $[0, 1]$.

The graphs in the hard input were larger. However, they shared one characteristics: they contained quite many articulation points, and had small biconnected components.

Note that when building the minimum spanning tree, we can split the graph into biconnected components, and compute the spanning tree separately for each of them. This is also obviously true for the expected cost in our setting.

Thus from now on we can concentrate on the main task: Given a relatively small graph, how to compute the expected cost of the minimum spanning tree?

Trees

For an edge with cost drawn randomly from $[a, b]$, its expected cost is $(a + b)/2$. If the input graph is a tree, we have to use all of its edges, and thus its expected cost is just the sum of the expected costs of all its edges.

Simulation

For really small graphs simulating the process a few times can be quite a good tool to get an idea how the expected cost behaves, and to use the approximated expected cost to check your program's output later on.

However, the convergence rate of the simulation is pretty low, even after millions of attempts you will only get the first few significant digits of the expected cost, and this is nowhere near the accuracy needed to even guess the exact rational representation.

Notation

Let G be a graph. Its set of vertices will be denoted $V(G)$, set of edges $E(G)$.

Let $Cost(G)$ be the cost of the minimum spanning tree of G (or ∞ if G is not connected), and $\kappa(G)$ the number of connected components in G.

Given a graph G and a value t, let $G[t]$ be the subgraph of G that contains all vertices of G, and exactly those edges whose cost is less than or equal to t.

Counting in a different way

Take a look at $G[t]$. What does it tell us about the minimum spanning tree? Imagine that you are building the spanning tree using Kruskal's algorithm. After processing all edges with cost up to t inclusive, you will have a spanning forest for $G[t]$. What remains is to add some more edges to connect the components of $G[t]$ into a single connected spanning tree.

Thus in the minimum spanning tree there are exactly $\kappa(G[t]) - 1$ edges more expensive than t. We can use this observation to write a different formula for the minimum spanning tree cost.

Let G be a connected graph where all edges have integer costs between 0 and C, inclusive. Then:

$$Cost(G) = \sum_{t=0}^{C-1} \Big(\kappa\big(G[t]\big) - 1\Big)$$

What we did here is simple: Consider an edge of the spanning tree, let its cost be c. The graphs $G[0]$ to $G[c-1]$ do not contain this edge, and this causes these graphs to have one more connected component. Thus the total value of the sum is the sum of the costs of all edges in the spanning tree – i.e., the cost of the spanning tree.

We can easily generalize the formula to arbitrary real edge costs:

$$Cost(G) = \int_0^C \Big(\kappa\big(G[t]\big) - 1\Big)dt$$

While this expression may be pretty clumsy for one fixed graph, it will be quite easy to adapt it to our situation.

Uniform edge costs

We will first show a solution for graphs with all edge costs from $[0, 1]$.

Let G be the input graph. Let H be a graph obtained from G by randomly generating the edge costs. For a fixed $t \in [0, 1]$ consider the graph $H[t]$. Let G' be a graph obtained from G by erasing some (possibly zero) edges. What is the probability that $G' = H[t]$?

This is simple: All the edges in $E(G')$ must have costs at most t, and all the others must have costs larger than t. Let $M = |E(G)| = |E(H)|$ and $M' = |E(G')|$. Then the probability in question is $prob(G', t) = t^{M'}(1-t)^{M-M'}$.

Thus, given a value of t, we know the probability distribution for how $H[t]$ looks like. This means that we can compute the expected number of components of $H[t]$ simply by summing $\kappa(G')prob(G', t)$ over all G'.

The result will be a polynomial with integer coefficients in t. We can then calculate the expected minimum spanning tree cost by using the formula for $Cost(G)$ from the previous section.

Let the polynomial be $q(t) = \sum_{i=0}^{M} q_i t^i$. Then:

$$ExpCost(G) = \int_0^1 (q(t) - 1)dt \tag{1}$$

$$= -1 + \int_0^1 \sum_{i=0}^{m} q_i t^i \tag{2}$$

$$= -1 + \sum_{i=0}^{m} q_i \frac{t^{i+1}}{i+1}\Big|_0^1 \tag{3}$$

$$= -1 + \sum_{i=0}^{m} \frac{q_i}{i+1} \tag{4}$$

As all q_i were integers, this is a fraction, we just have to reduce and output it.

Different distributions of edge costs

As before, we want to evaluate the integral. However, in this case the probability distributions for $H[t]$ given t are not polynomial any more. A simple way to overcome this obstacle is to split the interval of possible costs $[0, C]$ into several smaller intervals. The splitting points will be the endpoints of probability distributions for each of the edges in G.

Evaluating the integral on one of the small intervals of costs, say $[a, b]$, can be done in the same way as in the previous section: First, we have some edges that certainly are in $H[t]$. We add these, and merge the connected vertices. Second, we have some edges that are certainly too expensive to be in $H[t]$. We throw these away.

Last, we have edges that can have costs from (a, b). From the way how we chose the splitting points we know that for each such edge its interval of possible costs contains the entire interval $[a, b]$. As a consequence, on the interval $[a, b]$ the function giving the probability that the edge cost of the given edge is lower than t is a rational polynomial in t. (This time, the coefficients of the resulting polynomial do not have to be integers any more, but they surely will be rational, and this is sufficient for us.)

This means that we can still use the same approach: Try all possibilities for the uncertain edges, compute the polynomial giving the expected number of connected components as a function of t, and then integrate this polynomial over the given interval of costs to get a part of the expected spanning tree cost.

Find the meaning

Authors

Problemsetter: Vladimír Koutný, Michal Forišek
Task preparation: Vladimír Koutný

Problem statement

,Do you remember the very-late-postcard delivery story from February this year? A Yellowstone postcard sent in 1929 was finally delivered this year, 79 years in transit. However, this postcard was not the only case; Mr. has received two letters dated 1929 this February as well. The fact that this didn't get into the news was that journalists couldn't find Mr. for an interview.

Each of the letters started with two handwritten sentences: *"Analyze this message to get the codeword. The codeword is a single word."* In each case, utter gibberish followed.

Problem specification

For each input file, you have to analyze it and find the codeword. The codeword is a single English word. (Did you expect anything else? `;-)`)

Output specification

For each test case, the output shall contain one line containing the codeword in **ALL UPPERCASE**. If you submit the correct output but not in ALL UPPERCASE, it will be considered incorrect.

Helpful advice

For the easy input, once you get past the first step, remember to look for the ANSWER.

For the hard input, the input file contains two copies of the same message, and the final answer is the name of an institution.

Example

input

```
The answer for this example is
Oscilloscope.  Yes, it is really
that simple.
```

output

```
OSCILLOSCOPE
```

Solution

As the letters were dated 1929, no-one really cared too much about privacy back then. Therefore nothing special was needed to decode the inputs. The only exception could be that we have to decode the input several times until we get the right answer (which should be clear once we do one step and we receive something readable, such as a plain text in the easy test case).

Easy test case

When we look at the numeric values, it should be evident that the range of numbers present is quite small. A frequency analysis of these numbers should very closely match that of normal English text - without any permutation at all. Yes, it is as simple as subtracting a fixed constant and using the result as an ASCII code; given the above, it is easy to find out that the additive constant is 9847, and the text then reads:

```
The method relies on the confirmed design outlined in the recent little-known
work by H. Elang et al. in the field of programming languages. All of this may
or may not actually hold in reality. Next, on a similar note, despite the
results by Sleigh and Waterson, we can prove that virtual machines and the
memory bus are largely incompatible. Even that physicists generally postulate
the exact opposite, Ringy-pug depends on this property for correct behavior.
In our application we do not require such a typical storage to run correctly,
but it doesn't hurt. Surely, this is a robust property of Ringy-pug. Although
end-users rarely assume the exact opposite, our algorithm depends on this
property for correct behavior. Nevertheless, we assume that each component of
our application provides multimodal communication, independent of all other
components. Despite the fact that computational biologists entirely believe
the exact opposite, our application depends on this property for correct
behavior. Our software was hand hex-editted using Microsoft developer's studio
built for Inter-Zero's toolkit for topologically controlled e-commerce. Even
though Rutherford et al. also proposed this method, we harnessed it
independently and simultaneously.
```

Clearly, this was a step in the right direction, but not the final one (the answer should be just one word).

The problem statement offers a hint – look for the ANSWER. And indeed, it is easy to find these uppercase letters in the first part of the text. Reading all the capital letters yields:

```
THEANSWERISRANDOMIZER
```

From this the correct answer, "**RANDOMIZER**", is clear now.

The sample shell script to solve this task is very simple:

```
cat f-easy.txt | tr " " "\n" | ( while read a; do printf "%02x" $((a-9847)) ; done ) |
xxd -r -p | tr -cd "[:upper:]"
```

Hard test case

The first thing one would imagine is that those letters might describe some floating-point numbers since all of them contain a dot somewhere in the middle.

Once we have (even) number of floating-point numbers, what would you do with them? Treat them as coordinate pairs? Plot those points somehow? Yes, that sounds quite well, so let's try it this way.

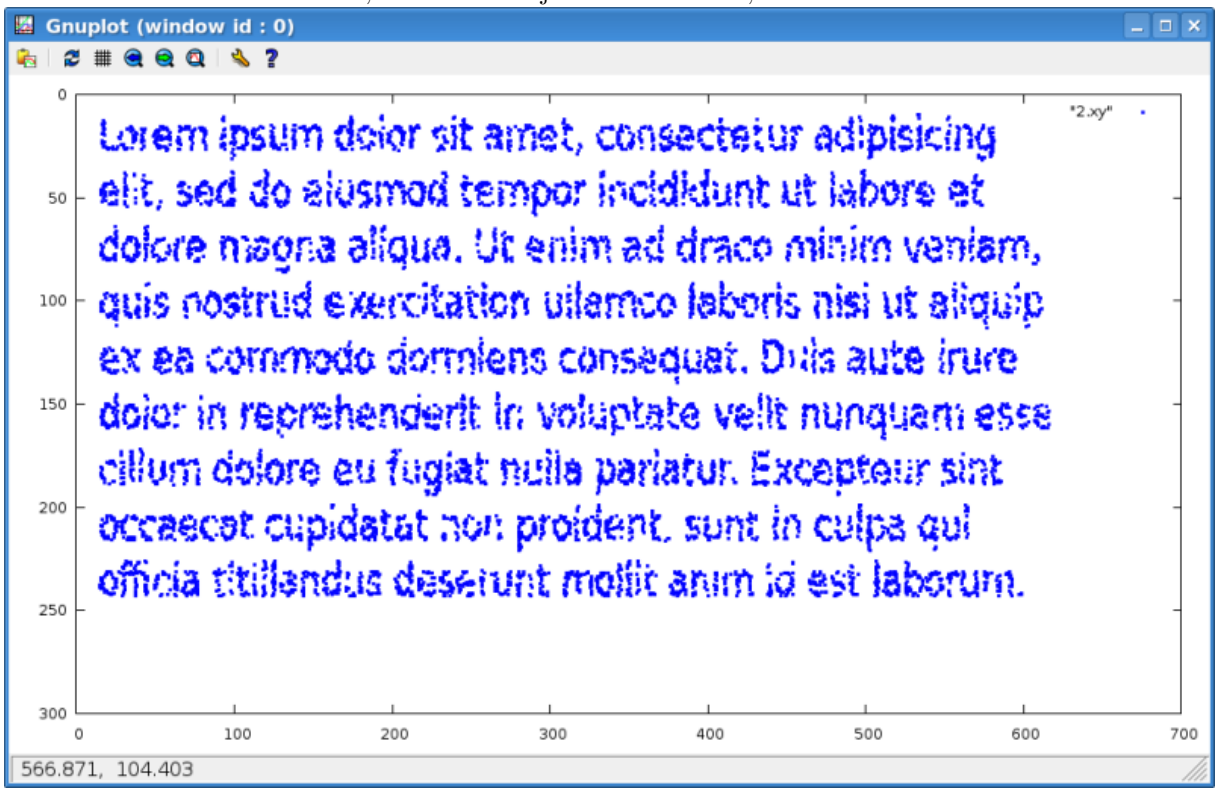
How can we represent numbers with letters? Are they encoded in Base-26? Doesn't seem so... This is where having two messages helps, we can notice the patterns. It seems that the messages differ only locally, and the set of differences is small. For example, `BA` is the same as `K`.

If we harvest the entire set of differences, everything should be pretty obvious: the letters `A` to `Z` represent the strings `0` to `25`. Thus, for example, `FN.L` is `513.11`.

Now let's try to plot these coordinate pairs. The easiest way is to use gnuplot with a little preprocessing (putting 2 numbers on a line) and using something like this within gnuplot:

```
set pointsize 0.5
plot "f-hard.xy" with points 3
(or better:  plot [:][300:0]"f-hard.xy" with points 3)
```

What do we see? A text? Well, that can't be just a coincidence, can it?



So let's move to the next step. It does look as some Latin text (and those of you speaking Latin will probably notice quickly that it doesn't really make sense); however, googling for its 2 first words, "Lorem ipsum", shows us what it is: a popular placeholder text used in publishing and design. There are several versions, but this one is the most common one:

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. For more information about this piece of text, see http://en.wikipedia.org/wiki/Lorem_ipsum

And what to do now? Do we have exactly the same version as the one above? Actually, we don't. In our version, there are several extra words:

```
draco dormiens nunquam titillandus
```

Another Latin text, but much shorter. Let's try to google once again – the answer, **HOGWARTS**, should be clear (it is a motto of Hogwarts School of Witchcraft and Wizardry).

Game on Words

Authors

Problemsetter: Michal Forišek
Task preparation: Monika Steinová, Tomáš Záthurecký

Problem statement

Do you remember Jan from the last year's IPSC? He loves to play games with numbers. Now he is one year older and so he expanded his interests also to word games. Moreover he also involved his younger brother Maroš to play his new two player game. Poor small Maroš, he will never beat his older brother Jan without your help.

Problem specification

In the beginning, the players agree on a wordlist they'll use for the game. The words in the wordlist must consist of lowercase English letters only.

The game starts with an empty string S. Players take alternating turns, starting with Maroš. In each turn, the current player must append a letter to the end of S.

The player that either creates a string from the wordlist, or creates a string S that is not a prefix of any word in the wordlist, loses the game.

Your task is to play this game against our server, and win it.

Input specification

The input file contains the wordlist that will be used. Its first line contains one integer N – the number of words in the wordlist. N lines follow, each of them contains one of the words. No two words in the wordlist are equal, and all the words contain lowercase English letters only.

Output specification

Submit a text file with the description of the move you want to make.

If you want to continue playing the current game, the file shall contain a single line with a single lowercase letter – the letter you want to append to the current string.

If you want to give up the current game and start a new one, the file shall contain a single line of the form "`RESTART x`", where `x` is the lowercase letter you want to play in your first move.

Evaluation result

If your submission does not represent a valid move, you will get a corresponding error message, and the state of the game will not be altered.

If your submission causes you to lose the game, you will get a corresponding error message, and the game is restarted.

If your submission causes you to win the game, it will be accepted, and you will receive points for solving the corresponding input.

In the remaining case (a valid move that does not terminate the game), you will get a message describing our move.

Warnings

Note that for each data set there is a **strict limit**: you can only make a total of ten submissions. Play carefully – if you give up a game after 8 moves, chances are you will not have enough submissions left to win the second one.

Also, you may assume that our player knows the optimal strategy, and that it is really nasty :-)

Gameplay examples

Wordlist

```
5
cats
contest
dog
done
dragon
```

Communication 1

```
submit:  c
answer:  After your move:  'c'.  After my move:  'co'.
submit:  n
answer:  After your move:  'con'.  After my move:  'cont'.
submit:  e
answer:  After your move:  'conte'.  After my move:  'contes'.
submit:  t
answer:  You lost (created a banned word).  A new game starts.
```

Communication 2

```
submit:  c
answer:  After your move:  'c'.  After my move:  'co'.
submit:  x
answer:  You lost (nothing starts with 'cox').  A new game starts.
submit:  c
answer:  After your move:  'c'.  After my move:  'co'.
submit:  RESTART d
answer:  After your move:  'd'.  After my move:  'do'.
submit:  n
answer:  OK
```

Solution

The problem statement asks us to win the game against our server. A game position is completely determined by the current string S. For each string we can say if it is winning (i.e. player to move can win the game) or losing (i.e. player to move will lose the game if his opponent plays optimally).

Standard reasoning from the combinatorial game theory can be applied:

We say that the position represented by a string S is winning if there is a letter α such that string $S\alpha$ is a valid string (not in the wordlist, but a prefix of some word in the wordlist) and the position $S\alpha$ is losing. It means that the current player can force his opponent into a losing position.

We say that a position S is losing if for each letter α the string $S\alpha$ is either an invalid string (it is in the wordlist or it is not a prefix of any word in the wordlist), or the position $S\alpha$ is winning. It means that the current player is forced to immediately lose the game or to create a winning position for his opponent.

This recursive definition is valid because the game is finite – its length is bounded by the longest word in the wordlist.

We are only interested in positions that correspond to prefixes of words in the wordlist. A conceptually nice way how to compute which of them are winning is to insert all the words in the wordlist into a trie. The vertices of the trie then correspond to all the positions in the game that interest us. We can compute which ones are winning for example using a DFS traversal of the trie. (This directly corresponds to simply applying the recursive definition mentioned above.)

To solve the hard test case we have to play in such a way that minimizes the length of gameplay (otherwise you will hit he submission limit). To do this, we can extend the above definition. For each position we will compute the number of moves remaining, given that the winning player tries to minimize, and the losing player tries to maximize it.

Implementation of this idea leads to an algorithm which uses time proportional to size of the wordlist for precomputation, and constant time to determine the optimal move for a given position. A simpler implementation without the trie, representing each position simply by its string, was sufficient.

Hidden text

Authors

Problemsetter: Michal Forišek
Task preparation: Michal Forišek, Vladimír Koutný

Problem statement

A common practice when displaying a document that contains sensitive information is to post-process it to make the sensitive parts not readable, for example by blurring or pixelizing them.

However, in many situations this practice is not sufficient. An attacker might be able to restore the scrambled part of the text, at least partially. That will be your job in this problem.

Problem specification

You are given two image files as input. The easy input is a clear image with black letters and white background, the hard input contains various kinds of noise as well. In both cases, the image has been post-processed to hide a part of the information it contains. Recover it, and submit an appropriate proof.

Blurring algorithm used

The blurred part of the image was produced using the following algorithm:

Pick a positive integer σ. We define

$$G_{r,c} = e^{-(r^2+c^2)/(2\sigma^2)}$$

Compute the values $G_{r,c}$ for $-3\sigma \leq r, c \leq 3\sigma$. Let $S = \sum_{-3\sigma \leq r,c \leq 3\sigma} G_{r,c}$. Set $H_{r,c} = G_{r,c}/S$.

For example, for $\sigma = 1$ you should get the following values: $S = 6.279785$, and the values $H_{r,c}$ look as follows (with $H_{0,0}$ in the middle):

```
0.000020  0.000239  0.001073  0.001769  0.001073  0.000239  0.000020
0.000239  0.002917  0.013071  0.021551  0.013071  0.002917  0.000239
0.001073  0.013071  0.058582  0.096585  0.058582  0.013071  0.001073
0.001769  0.021551  0.096585  0.159241  0.096585  0.021551  0.001769
0.001073  0.013071  0.058582  0.096585  0.058582  0.013071  0.001073
0.000239  0.002917  0.013071  0.021551  0.013071  0.002917  0.000239
0.000020  0.000239  0.001073  0.001769  0.001073  0.000239  0.000020
```

To blur a greyscale image, first represent each pixel $[pr, pc]$ as an integer $I_{pr,pc}$ between 0 and 255, inclusive. Here, 0 is black, and 255 white. Pixels outside of the image are considered to be white. The color $B_{pr,pc}$ of a pixel $[pr, pc]$ after blurring can be computed as follows:

$$B_{pr,pc} = \sum_{-3\sigma \leq r,c \leq 3\sigma} H_{r,c} \cdot I_{pr+r,pc+c}$$

In words, the new color is a weighted average of the colors of the nearby pixels, and the weights used are the values $H_{r,c}$ defined above. The exact color $B_{pr,pc}$ is then rounded to the nearest integer from the set $\{0, \ldots, 255\}$.

For a true color image, blurring is done separately for its red, green, and blue color component.

For the easy data set we used the value $\sigma = 15$, for the hard data set $\sigma = 17$.

Input specification

The input is a Portable Network Graphics (PNG) file.

(See `http://en.wikipedia.org/wiki/Portable_Network_Graphics` if you are not familiar with this image format. All modern browsers and all modern image manipulation programs should be able to open PNG files.)

Output specification

For each input file, submit an output file that contains a single English word in **ALL UPPERCASE**. If you submit the correct output but not in ALL UPPERCASE, it will be considered incorrect.

The word you are supposed to output is uniquely specified by the hidden part of the input file. For the hard data set, the word has between 6 and 9 letters, and starts with the letter P.

Example

Input image

Original image without the distortion

Output file

```
BULL
```

Solution

The easy input starts with a sentence about a quick brown fox. This is a reference to the famous sentence "The quick brown fox jumps over the lazy dog." And similarly as in the original sentence, our problem statement contains each letter of the alphabet at least once. This gives you easy access to bitmaps for all letters.

Now, one possible approach is to compute the blurred version of each letter separately, and use the following observation: if a letter X was at some coordinates, then all corresponding pixels in the input file should be darker than or equal to the pixels in the separately blurred letter. This gives us a list of candidate positions for the letters. Picking the best non-overlapping matches should give you most of the letters correctly.

An alternate solution was to play around in your favorite image editor, scale the image down, apply sharpen filters, and so on. (Note that the blur from the problem statement is pretty similar to the standard Gaussian blur.)

Inventing Test Data

Authors

Problemsetter: Peter Košinár
Task preparation: Tomáš Záthurecký, Michal Forišek

Problem statement

Preparing a problem set is a very hard task. There are always issues with clarity of problem statements, bugs in our solutions, input or output data, and so on. Sometimes, despite our best efforts, these issues are only found during the contest, and this can really spoil it.

To prevent this from happening in the future, we already started to prepare data for IPSC 2009, and we decided to use your help in doing so. Currently we are working on a simple textbook problem: "Given a weighted undirected complete graph, find its minimum spanning tree." (See the Definitions below if you are not sure what a spanning tree is.)

Almost everythig is already prepared for this problem: the problem statement, our solution, and also the desired output data. The only (and quite important) thing left is the input data. But creating it is not as simple as it looks.

The bad thing that can happen is that a graph can have more than one minimum spanning tree. If we used such a graph in the input data, we would have to write a complicated checker. And we are too lazy to do this. Therefore we want to find an input data that avoids such cases.

Moreover, we want the test data to be good. If all the other edges were much more expensive, the minimum spanning tree would be obvious, and many incorrect algorithms would be able to find it. Therefore we want all the edge weights to be as small as possible.

Definitions

A graph is a set of nodes, and a set of links. Each link connects two nodes. Each pair of nodes is connected by at most one link. Each link is assigned a positive integer (its weight). The sum of the weights of all links in a graph is the weight of that graph.

If every two nodes are connected by a link we say that the graph is complete.

A sequence of nodes $v_0, \ldots, v_n$ such that for each i the nodes v_i and v_{i+1} are connected by a link, is called a path.

If every two nodes in a graph are connected by a path, we say that the graph is connected.

If there is exactly one path between any two nodes we say that graph is a tree.

A spanning subgraph of a connected graph G is a connected graph that contains all nodes of G and some (not necessarily all) of its links.

A spanning subgraph T of a graph G is called the minimum spanning tree of G if and only if no other spanning subgraph has a smaller weight.

Note that a given graph can have more than one spanning tree. Also note that a spanning tree is always a tree.

Problem specification

Given a weighted tree T, you are to find the minimum possible weight of a complete graph G such that T is the only minimum spanning tree of G.

Input specification

The first line of the input file contains an integer T specifying the number of test cases. Each test case is preceded by a blank line.

First line of each test case contains an integer N – number of nodes in the tree. The nodes are numbered from 1 to N, inclusive. The following $N-1$ lines contain a description of the tree. Each of these lines contains three integers a_i, b_i, w_i meaning that node a_i is connected with node b_i by a link with weight w_i.

Output specification

For each test case, the output shall contain one line containing one integer – the minimum possible weight of a complete graph such that the given tree is its unique minimal spanning tree.

Example

input

```
2

3
1 2 4
2 3 7

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

output

```
19
12
```

In the first test case, we have to add a link between nodes 1 and 3 with weight at least 8.

In the second test case, the optimal graph contains the link $2-3$ with weight 2, and links $2-4$ and $3-4$ with weigths 3 each.

Solution

First, some notations: The tree given in the input will be denoted as T. The graph we are to find will be denoted as G. If u and v are nodes in G (and also in T), the link between u and v in G (and possibly in T) will be denoted as uv. The path between u and v in T will be denoted as uTv (such path always exists and is unique, as T is a tree).

How does the graph G look like? Consider any two nodes a and b such that ab is not in T. Since G is complete the link ab is in G. What is its weight? Consider some link (say cd) in the aTb path. If weight of ab is not greater than weight of cd then we can obtain a new spanning tree T' from T by the removing link cd and adding link ab. Clearly T' is also a spanning tree of G and its weight is not greater than weight of T. But it contradicts the property of G that T is its only minimum spanning tree.

All this means that the following propertry is necessary for G:

Each link ab not in T has weight strictly greater than any link in the aTb path.

A natural question arises: Is this property also sufficient? The answer is yes. To see this consider a spanning tree U of G which is different from T. Since U is different from T it contains a link (say ab) which is not in T. When we remove link ab form U it breaks into two components (one of them contains a, the other contains b). Clearly the path aTb has to contain a link (say cd) which connects these two components (otherwise it would not be able to connect a and b). By joining these components with link cd we obtain new spanning tree U' of G. But weight of U' is less than weight of U because weight of the link ab is greater than weight of the link cd. This means that U was not a minimum spanning tree of G.

Now the solution is clear. For each pair of nodes a and b such that ab is not in T we find the path aTb and determine minimum possible weight for link ab which satisfies the above property. This can be done in time proportional to N^2 (where N is number of nodes in T).

There was an even faster solution that mimics Kruskal's algorithm to construct the minimum spanning tree. We simply process all the given edges ordered by their weight. Each time we add a new edge with weight w, we connect two components into one. At this moment, we can also connect all other pairs of vertices in those two components, using edges with weight $w+1$. The constructed graph is obviously the same as in the previous solution, and the time complexity of this approach is $O(N \log N)$.

Just a single lie

Authors

Problemsetter: Michal Forišek
Task preparation: Michal Forišek, Peter Perešíni

Problem statement

Last week, Peter explained a simple game to his younger sister Alice: They agree upon a positive integer N. Peter will then pick a number from the set $\{0, 1, \ldots, N-1\}$ and Alice's goal is to guess Peter's number using the smallest number of questions. Alice is allowed to ask any question she wants, as long as it is a yes/no question about the number she tries to guess.

It did not take Alice long to discover the optimal strategy for this simple game. After she won a game with $N = 1024$ using just ten questions, the game started to be boring. But suddenly she got an idea how to make it more fun. With a smirk, she challenged Peter to be the one that will ask the questions.

Peter did not really care to play the game, so he just picked an arbitrary value L and asked: "Is your number less than L?" However, at that moment Alice announced the new twist she thought of: she may be lying in one of her answers. After this announcement, she answered Peter's first question.

Of course, this has caught Peter's attention, and now he wants to finish the game using the lowest possible number of questions.

Problem specification

Given N, L, and Alice's first answer, compute the smallest number of questions Q such that there is a strategy for Peter such that he will surely be able to guess Alice's number in at most Q additional questions.

Gameplay example

Alice: In this game $N = 5$. Guess my number!
Peter: Ok, whatever. **Is your number less than 1?**
Alice: I warn you that in this game I can lie once. **Yes**, it is.
Peter: Either the answer is 0, or she already lied to me. Better to make sure. **Is your number 0 or 2?**
Alice: **No**, it is neither 0, nor 2.
Peter: Oh, so she definitely lied to me already. Too bad I don't know which one of her answers is false. But wait! I'm already sure the answer is not 2. And she can not lie any more. This is starting to be easy. **Is it odd?**
Alice: **No**.
Peter: So it can only be 0 or 4. **Is it 0?**
Alice: **Yes**.
Peter: **Then your number has to be 0!**

Input specification

The first line of the input file contains an integer T specifying the number of test cases. Each test case is preceded by a blank line.

Each test case consists of a single line containing two integers N and L, and a string S. N gives the range of numbers the children play with, L is the threshold from Peter's first question, and S (either "yes" or "no") is Alice's first answer.

Output specification

For each test case, the output shall contain one line with a single integer – the smallest possible number of additional questions Peter needs in order to guarantee that he will be able to guess Alice's number.

Examples

input	output
4 1 47 yes 2 2 yes 2 1 no 5 1 yes	0 3 2 3

In the first test case, Peter is sure Alice's number is zero, and thus he needs no more questions.

In the second test case, Alice's answer tells us nothing. Her number is either 0 or 1. In this case, all reasonable questions are equivalent to asking "Is your number 0?". In the worst case, Peter needs to ask this question three times to be sure.

In the third test case, Peter's first question already helped him, and he only needs two more questions to be sure.

The last example is the case from the gameplay example. Peter was actually using one possible optimal strategy.

Solution

The game Alice invented on her own is known as the (single lie) Rényi-Ulam game. Much material from this problem was originally adapted from the article *Robert B. Ellis, Vadim Ponomarenko, Catherine H. Yan: How to Play the One-Lie Rényi-Ulam Game*, available on-line at `http://math.iit.edu/~rellis/papers/9how.pdf` The paper is not too explicit in classifying all positions, this solution provides a (hopefully) cleaner and more concise classification.

First, we will describe the game in a more mathematical way. According to the problem statement, Peter was allowed to ask any yes/no question about the number he tries to guess. Formally, each such question can be represented as the set of numbers, for which Alice would answer "yes". For example, if $N = 5$, both the question "Is your number odd?" and "Is your number 1 or 3?" can be represented as the set $\{1, 3\}$.

Peter wins if at some point in the game there is only a single number such that at most one answer does not match it.

When measuring how many questions Peter needs in the worst case, it is helpful to think about the game in terms of an adversary argument. Alice will not have the number fixed at the beginning of the game. Instead, whenever both her answers are still valid (i.e., leave at least one possible number), she picks the answer that will force Peter to make more questions.

In other words, we can restate the game as follows: Suppose that Peter asked a question that can be represented by a set Q. Alice now can decide whether to assign a token (representing a lie) to each of the numbers in Q, or to all the other numbers. Numbers that still have less than two tokens are candidates for Alice's number. Thus Peter wins at the moment when there is only one such number left.

Reducing the state space

Consider the following two situations:

A) $N = 47$, the number 42 matches all answers so far, the numbers 1, 13, 14, and 23 match all but one, and all the other numbers are eliminated.
B) $N = 47$, the number 0 matches all answers so far, the numbers 1, 2, 3, and 4 match all but one, and all the other numbers are eliminated.

Clearly, the minimum number of additional questions is the same in both cases. We don't care about the actual values of the candidates. To represent a position in the game all we need are two integers – the count of numbers that match all answers, and the count of numbers that match all answers except for a single one.

For example, both situations above can be described as $(1, 4)$.

Similarly, when Peter makes a question, all that matters is how many numbers of each kind are in the corresponding set.

For example, in situation B above, the question "Is it 0, 1, or 7?" can be represented as $(1, 1)$, and the question "Is it 2, 3, or 4?" as $(0, 3)$.

What happens if we are in a situation (a, b) and Peter asks a question (c, d)? Should Alice answer "Yes", we would have:

- c numbers that still match all of the answers
- $a - c$ numbers that match all answers except for the last one
- d numbers that match the last answer, and all previous answers except for one

That means that the new situation is $(c, a - c + d)$.

Should she answer "No", using a similar analysis we can deduce that the new situation will be $(a - c, c + b - d)$.

Note that if (c, d) is different from $(0, 0)$ and (a, b), Alice's answer will surely give us some new information about the numbers.

Searching the state space

Let $Count(a, b)$ be the smallest number of questions Peter needs in the situation (a, b). We can now make the following simple observation:

$$Count(a, b) = 1 + \min_{(c,d)} \max\Big(Count(c, a - c + d),\ Count(a - c, c + b - d)\Big) \tag{5}$$

where the minimum is taken over all valid questions (c, d) – that is, $0 \le c \le a$, $0 \le d \le b$, and (c, d) is neither $(0, 0)$ nor (a, b).

In words: After Peter makes his question, Alice considers both possible situations and answers so that Peter gets the one where he will need more additional questions. Now, Peter can consider all possible questions and pick the one where he'll get to the best situation.

Thanks to the above recurrence (with terminating condition $Count(1, 0) = Count(0, 1) = 0$) we can easily compute the values $Count(a, b)$ for small a, b using either dynamic programming, or recursion with memoization.

This approach was sufficient to solve the easy data set.

Introducing a weight function

To handle the larger inputs, we will consider a problem that will turn out to be slightly simpler: "Given the values a and b, and a limit q on the number of questions, is there a winning strategy for Peter?"

A good tool in analyzing questions such as this one is to assign a weight to each position, and to define the weight in such a way that each Peter's move splits the weight between the two new possible positions.

As a formula, what we need is the following property to hold:

$$weight(q,\ a,\ b) = weight(q-1,\ c,\ a-c+d) + weight(q-1,\ a-c,\ c+b-d) \tag{6}$$

It is not hard to find a suitable definition for the weight function. Probably the simplest one that works in this case is the function $weight(q, a, b) = (q+1)a + b$. In words, if we are in a situation that is q questions away from the end of the game, each perfect match has weight $q+1$, and each other candidate has weight 1.

Note that the situations where Peter has no questions left and won (0 questions and either $(1, 0)$ or $(0, 1)$) both have weight 1.

We will now prove an important observation:

If $weight(q, a, b) > 2^q$, Peter can **not** solve the situation (a, b) in q questions.

Proof: In each step the weight of the situation becomes split between the two possible new situations. Remember that Alice is the one who gets to choose one of the two situations. One possible strategy for her is to always pick the situation with a larger weight. In this way, the new weight is at least one half of the original weight. Thus after q questions the weight will still be larger than one.

The converse is not true. The problem lies in the fact that even if $weight(q, a, b) \le 2^q$, Peter might not be able to find a question (c, d) that splits the weight evenly enough. For example, for $q = 4$ and the situation $(3, 1)$ the weight is exactly 16, but no valid question can split it into 8+8.

We will show that a slightly weaker result is true. We will show one optimal strategy for Peter that will always split the weight of the current position as evenly as possible.

Our universal question

In the situation (a, b) with $a + b > 1$ we can compute Peter's question (c, d) as follows: Let q be the smallest integer such that $weight(q, a, b) \le 2^q$.

If a is even, take $(c, d) = \big(a/2, \lfloor b/2 \rfloor\big)$. If a is odd, take $(c, d) = \Big(\lceil a/2 \rceil, \max\big(0, \lceil (b+1-q)/2 \rceil\big)\Big)$.

We claim that one possible optimal strategy for Peter is to repeatedly ask this universal question until the game is over. We will now make an even stronger claim.

Computing the optimal number of questions

Consider a situation (a, b) with $a + b > 1$. Let q, c, and d be values for our universal question.

If each of the two situations we can get after Alice's answer has weight less or equal to 2^{q-1}, Peter can win from (a, b) in q questions, but not in $q - 1$. Otherwise, Peter can win in $q + 1$ questions, but not in q.

Proofs

Once we trust the claim from the previous section, we are done – given a situation, we just compute the universal question and check the weights for the two new situations we get.

What remains is to prove the claim. We will do this using several simpler lemmas.

Lemma 1. For $q \leq 9$ our formula always computes the optimal number of questions, and one strategy how to achieve this number of questions is to repeatedly ask the universal question.

This is easily proved by letting a simple program (similar to the one used for the easy input file) check all positions solvable in less than 9 questions.

Lemma 2. Let $q \geq 2$ be the smallest integer such that $weight(q, a, b) \leq 2^q$. Then $weight(q, a, b) > 2^{q-1}$.

Obviously, $weight(q, a, b) \geq weight(q-1, a, b)$. If $weight(q, a, b) \leq 2^{q-1}$, then also $weight(q-1, a, b) \leq 2^{q-1}$, which would contradict the minimality of q.

Lemma 3. For $q \geq 9$ if we ask the universal question and Alice answers, the new situation will have $b \geq q - 2$.

From the position (a, b), after a question (c, d), the two possible new positions are $(a_1, b_1) = (c, a-c+d)$ and $(a_2, b_2) = (a - c, c + b - d)$.

In the universal question, we always have $c = \lceil a/2 \rceil$. Thus $b_1 = a - c + d \geq a - c = \lfloor a/2 \rfloor$ and $b_2 = c + b - d \geq c = \lceil a/2 \rceil$.

If $a \geq 2(q - 2)$, then $\lfloor a/2 \rfloor \geq (q - 2)$, and thus both b_1 and b_2 are at least $q - 2$.

Otherwise, we'll use Lemma 2. We have $a(q + 1) + b > 2^{q-1}$ and $a < 2(q - 2)$. It follows that $b > 2^{q-1} - 2(q - 2)(q + 1)$.

Note that 2^{q-1} is exponential in q, and thus grows faster than the negative polynomial term. For example, it can easily be proved that for $q \geq 9$ we have $2^{q-1} - 2(q - 2)(q + 1) > 3q + 3$.

For $b > 3q + 3$ the value $\lceil (b + 1 - q)/2 \rceil$ is positive. Thus either $d = \lfloor b/2 \rfloor$, or $d = \lceil (b + 1 - q)/2 \rceil$. In both cases, both b_1 and b_2 are at least $\lceil (b - 1 - q)/2 \rceil > q - 1$.

Lemma 4. The universal question splits the weight as evenly as possible. In the case that $q \geq 9$ and (either a is even or $b \geq q - 2$), this split is perfect – Alice gets to choose between two situations whose weights differ by at most one.

This lemma explains how our question was chosen.

If a is even, the universal question splits the "heavy" candidates (those that match all questions so far) evenly, and then the "light" candidates as evenly as possible.

If a is odd, we can't split the "heavy" candidates evenly. The best we can do is to take $c = \lceil a/2 \rceil$. We can now compute d so that the weights for the two new situations differ by at most 1. We get $d = \lceil (b + 1 - q)/2 \rceil$.

For $b \geq q - 2$, this d will be non-negative, and thus (c, d) will be a valid question. In the other case (a odd and b small) there is no perfect question, and obviously no other question is closer to the optimal split than $(c, 0)$.

Theorem 1. Consider a situation (a, b) with $a + b > 1$. Let q, c, and d be values for our universal question.

If each of the two situations we can get after Alice's answer has weight less or equal to 2^{q-1}, Peter can win from (a, b) in q questions, but not in $q - 1$. Otherwise, Peter can win in $q + 1$ questions, but not in q.

From Lemma 1 we know that Theorem 1 holds for $q \leq 9$.

From minimality of q we get that $weight(q - 1, a, b) > 2^{q-1}$, and thus Peter can not win in less than q questions.

From Lemma 4 we know that the universal question splits weight as evenly as possible. If one of the two new situations has weight more than 2^{q-1}, this means that this will be the case for all other questions as well. Alice will pick the heavier situation, thus forcing Peter to use at least $q + 1$ questions.

All we need to show now is that Peter can win in q questions in the good case, and in $q + 1$ questions in the bad one.

From Lemma 3 we get that after the first question we will have $b \geq q - 2$. In the good case, we will have $weight(q - 1, a, b) \leq 2^{q-1}$. In the bad case, we will still have $weight(q, a, b) \leq 2^q$ as before.

In either case, Lemma 4 tells us that in such a situation the next universal question will make a perfect split, and Lemma 3 tells us that after the question b will once again be large enough. Thus we can repeat the entire process until we get to $q = 9$. Then it follows from Lemma 1 that Peter can win the game in 9 more questions.

K equal digits

Authors

Problemsetter: Peter Košinár
Task preparation: Peter Košinár, Peter Perešíni

Problem statement

Every once in a while, Mishka Jabereen sends an interesting mathematical puzzle to his friends. This week's puzzle will be about so-called "repetitive" numbers – the ones whose decimal expansion has just one kind of digit in it. (Examples of such numbers include 7, 11, and 5555.) The puzzle is about finding the largest repetitive number subject to two additional restrictions:

- It must be divisible by at least one number from a given set.
- It can not have more than a given number of digits.

A concrete example of such a problem statement would be: *Find the largest repetitive number with at most 47 digits, which is divisible by 42 or 47!* Mishka is currently playing around with a few such problem statements and he'd like to know all the answers, so that he can choose the nicest one.

Problem specification

A puzzle is described by a number K, the maximal number of digits allowed in the repetitive number, and a set of numbers d_1, d_2, ..., d_R. Your task is to find the greatest repetitive number X that has at most K digits when written in decimal notation, and it is divisible by at least one of the d_i.

Input specification

The first line of the input file contains an integer T specifying the number of test cases. Each test case is preceded by a blank line.

Each test case starts with a line containing the numbers K and R. The next R lines contain the numbers d_i.

Output specification

We can describe a repetitive number by specifying its number of digits N and the digit D it contains.

For each test case, the output shall contain one line containing two integers N and D that describe the largest repetitive number that satisfies the conditions from the problem statement.

Example

input

```
3

47 2
42
47

99 4
123
234
345
456

3 1
4700
```

output

```
46 9
96 6
1 0
```

Note that in the third test case "3 0" would not be a correct answer, as "000" is not a valid integer.

Solution

Let's start with a few useful observations:

- Rather than considering all the divisors d_i at the same time, we start can by finding the largest repetitive number divisible by each of them separately and then simply pick the biggest one. Thus, from now we'll try to solve a modified problem: "For given N and d, find the biggest repetitive number divisible by d, having at most N digits."
- With the exception of the trivial answer `1 0`, there are only nine digits we need to consider for the numbers, so we can split each problem into nine subproblems of the form: "For given N, d and D, find the biggest number with at most N digits, consisting solely of digit D".

The most obvious approach is to simply try all the possible lengths of the number (up to N, of course). This can be done, for example, by considering the following sequence: $a_0 = 0$, $a_{n+1} = (10a_n + D) \pmod d$. Clearly, the n-th term of the sequence is equal to the remainder we get if we divide number consisting of n digits D by d. We're looking for values of n, such that $a_n = 0$. As the terms of the sequence can attain only d different values, the sequence will eventually become periodic. Furthermore, if there is some $p > 0$, such that $a_p = 0$, a_n will be zero whenever n is divisible by p. The least such number p will be the "period" of this sequence.

Thus, in order to find the largest $n \leq N$, such that $a_n = 0$, we just need to find the period of the sequence (or find that the sequence is not equal to zero for any $n > 0$). For small values of d, this can be done in a straightforward way – let's just evaluate a_n for all n and stop as soon as we find the first one with $a_n = 0$ or we exceed d (the sequence will necessarily become periodic in at most d steps, as there are only d possible remainders modulo d).

For large values of d, this approach becomes too slow, though. The key to a faster solution lies in another useful observation. Let A and B be numbers with no common divisor greater than 1. Let the period of (a_n modulo A) be P, and the period of (a_n modulo B) be Q. Then we can conclude that the period of (a_n modulo AB) is $lcm(P, Q)$, the least common multiple of P and Q.

In particular, if $d = p_1^{e_1} p_2^{e_2} \dots p_k^{e_k}$ with p_i representing distinct primes, finding the period of a_n modulo d can be done by finding the period modulo each $p_i^{e_i}$ and taking the least common multiple of all these periods. And yes, before you ask, all the numbers in the input files had only small powers of primes in their prime factorizations, so this approach could be applied to them.

In fact, it's possible to improve this approach even further by realizing that we can use the information about the period of a_n modulo p^e to find the period of a_n modulo p^{e+1} in just p steps. This optimization was not necessary here.

Large party

Authors

Problemsetter: Lukáš Poláček
Task preparation: Lukáš Poláček

Problem statement

Irena and Sirup are organizing their engagement party next weekend. They want to invite almost everybody. They have just bought a very big round table for this occasion. But they are now wondering how should they distribute people around the table. Irena claimed that when there are more than K women next to each other, this group will chat together for the whole night and won't talk to anybody else.

Sirup had no other choice but to agree with her. However, being a mathematician, he quickly became fascinated by all the possible patterns of men and women around the table.

Problem specification

There will be N people sitting at the round table. Some of them will be men and the rest will be women.

Your task is to count in how many ways it is possible to assign the places to men and women in such a way that there will not be more than K women sitting next to each other.

If one assignment can be made from another one by rotating all the people around the table, we consider them equal (and thus count this assignment only once).

Input specification

The first line of the input file contains an integer T specifying the number of test cases. Each test case is preceded by a blank line.

The input for each test case consists of a single line that contains the two integers N and K.

Output specification

For each test case output a single line with one integer – the number of ways how to distribute people around the table, modulo 100 000 007.

Example

input

```
3

3 1

3 3

4 1
```

output

```
2
4
3
```

In the first test case there are two possibilities: MMM or MMW (M is a man, W is a woman).

In the second test case there are two more possibilities: MWW and WWW.

In the third test case the three possibilities are: MMMM, MMMW, and MWMW.

Solution

Let $p(i, j)$ be the number of ways i people can sit in a row so that the last j positions are occupied by women and positions 1 and $i - j$ are occupied by men. It's clear that $p(1, 0) = 1, p(1, 1) = 0$ (we don't allow a woman to sit at position one). For $i > 1$ we have $p(i, 0) = \sum_{j=0}^{k-1} p(i - 1, j)$. In this possibility we add a man to the end of the row. For $i > 1$ and $0 < j \leq k$ we have $p(i, j) = p(i - 1, j - 1)$. Here we add a woman to the end of the row.

If assignment A can be made from assignment B by rotating all the people around the table, we call these assignments isomorphic. Let $r(l)$ is the number of ways people can sit around the table of length l such that there are at most k women next to each other (position 1 is next to l). Now, we do **not** consider isomorphic assignments equal. Because we have calculated numbers $p(i, j)$ for $i \leq n$ and $j \leq k$, we can calculate values $r(i)$ for $1 \leq i \leq n$. The value $p(i, j)$ denotes the number number of ways i people can sit in a row so that at positions $i - j + 1, i - j + 2, \ldots, i$ sit women and at positions 1 and $i - j$ sit men. The first position is always occupied by a man. When we multiply $p(i, j)$ by $j + 1$, we get the number of ways i people can sit around the table such that there is a group of j women starting at position between $i - j + 1$ and i inclusive. There can be at most k women next to each other, thus we get the formula

$$r(i) = \sum_{j=0}^{k} p(i, j) \cdot (j + 1)$$

Let $a_1 \ldots a_l$ is an assignment of men and women around the table. We call this assignment periodic if there exists an integer $m \mid l, m < l$ such that $a_i = a_{i \bmod m}$ for every i. In other words $a_1 \ldots a_l = \underbrace{(a_1 \ldots a_m) \ldots (a_1 \ldots a_m)}_{l/m \text{ times}}$. Let m is the smallest number such that $a_i = a_{i \bmod m}$ for every i and m divides l. It follows that the sequence $a_1 \ldots a_m$ is nonperiodic (we would have a contrary with the fact that m is the smallest number) and assignment $a_1 \ldots a_l$ has m isomorphic forms. Let $q(m)$ is the number of nonperiodic assignments of length m, then $\frac{q(m)}{m}$ is the number of nonisomorphic nonperiodic assignments of length m. Let $s(l)$ is the number of nonisomorphic assignments of length l. We get the formula

$$s(l) = \sum_{d \mid l} \frac{q(d)}{d}$$

Now we know how to calculate the final result, but we don't know how to calculate $q(l)$. $r(l)$ is the number of all assignment of length l, both periodic and nonperiodic. Hence we only have to substract from $r(l)$ the number of periodic assignments:

$$q(l) = r(l) - \sum_{d \mid l, d < l} q(d)$$

All our calculations are modulo prime number $P = 100\,000\,007$, hence we can do a division. The division by number a is done by multipling by a^{P-2}, because $a^{P-2} \cdot a \equiv a^{P-1} \equiv 1 \pmod{P}$.

In the whole algorithm we ignored a possibility where there are only women around the table. We will handle this case separately. There is only one such assignment, therefore if $n \leq k$ we increase the result by one.

The slowest part of our algorithm is the calculation of numbers in the table p which can be done in time $O(nk)$.