IPSC 2010

problems and sample solutions

Piece of cake!

Authors

Problemsetter: Michal 'mišof' Forišek
Task preparation: Monika Steinová, Michal 'mišof' Forišek

Problem statement

This morning Aunt Petunia baked a cake. The cake had the shape of a box with dimensions $A \times B \times C$ centimeters.

During the day, each of the D kids in the neighborhood stopped by for a slice of the cake. Every time a kid came for some cake, Aunt Petunia took her knife and made a single cut parallel to one of the sides of the cake. The piece she cut off the cake was always exactly 1 cm thick.

Problem specification

Given the values A, B, C, and D, compute the largest possible volume the cake could have at the end of the day.

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 the integers A, B, C, and D.

You may assume that $0 < A, B, C \leq 10^{18}$ and $0 \leq D \leq A + B + C - 3$.

In the easy input you may assume that $A, B, C \leq 1000$.

Output specification

For each test case output a single line with a single integer: the largest possible volume of the rest of the cake, after each of the kids got a slice.

You may assume that for each test case the answer will conveniently fit into a 64-bit signed integer variable.

Example

input

```
3

4 5 6 0

4 5 6 3

1 1 10 9
```

output

```
120
64
1
```

One way of optimally solving the second test case: first make two cuts parallel to the 4×5 side to obtain a $4 \times 5 \times 4$ cake, and then make a cut parallel to the current 4×4 side to obtain a cube with side 4 cm long.

Solution

First of all notice that the order in which Aunt Petunia cuts the slices of the cake is not important. (Of course, not important with respect to our problem. Some of the kids would disagree `:-)`.)

The only thing that matters is the number of cuts made in each of the three possible directions. Regardless of the order in which we make them the remaining cake looks always the same.

Solving the easy test case

One possible solution of this task is to make the above observation and then to use brute force. Try all possible numbers of cuts for each of the three sides of the cake (i.e., triplets of integers that sum up to D). For each possibility, compute the volume of the remaining cake and pick the maximum.

Another way of solving the easy test case was to realize that a greedy approach always works: Always make a cut that cuts away the smallest part of the cake. In other words, a cut orthogonal to the longest edge. The proof is shown in the next section.

A useful lemma

Out of all rectangles with the same perimeter, the one with the largest area is the square.

If the sides are required to be integers, the optimal one is the one that most resembles the square.

Proof: Take any rectangle $a \times b$ with $a \geq b+2$. This rectangle is not optimal, because of the rectangle $(a-1) \times (b+1)$. Its perimeter is the same, while its area is $(a-1)(b+1) = ab + a - b - 1 > ab$.

This means that the optimal rectangle is the only one that cannot be improved in the above way – the one closest to being a square.

Solving the hard test case

It is not hard to realize that every box $p \times q \times r$ that we can produce has the same value $p+q+r$. More precisely, we always have $p + q + r = A + B + C - D$. This is because each cut decreases one dimension by 1.

Now consider any final shape we can produce. If it contains two edges with lengths $x \geq y + 2$ such that y is **not** the original length of that edge, we can use the lemma to improve the volume: we will make one less cut that decreases the y edge, and one more that decreases x.

This observation proves the greedy algorithm: note that it produces essentially the only box that cannot be improved using the above observation.

In order to solve the hard test case, it was necessary to implement the greedy algorithm in some way that would be faster than the naive "one step at a time" method. It is actually doable in constant time, see our sample implementation for an example.

Quick growth

Authors

Problemsetter: Michal 'mišof' Forišek
Task preparation: Monika Steinová, Michal 'mišof' Forišek

Problem statement

The memory of Bob's computer contains two interesting things: an array of integers and a virus.

Each midnight the virus becomes active. It takes each array in memory and replaces it with a bunch of new arrays: one for each contiguous subarray of the original array.

For example, if today the memory contains a single array $(1, 2, 1, 3)$, tomorrow it will contain the following arrays: (1), (2), (1), (3), $(1, 2)$, $(2, 1)$, $(1, 3)$, $(1, 2, 1)$, $(2, 1, 3)$, and $(1, 2, 1, 3)$.

Problem specification

You are given the length N of Bob's original array, its contents and the number of days D.

Compute the sum of all elements of all arrays that will be in the memory of Bob's computer after D days. As this number can be huge, it is sufficient to compute the remainder it gives when divided by $10^9 + 9$.

Assume that the memory of Bob's computer is sufficiently large to accommodate all the arrays.

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 two lines. The first line contains the two integers N and D. The second line contains N small non-negative integers: the contents of the original array.

In the easy data set you may assume that the total number of elements in all arrays after D days will be at most 10 000 000.

In the hard data set you may assume that $N \leq 50$ and $D \leq 1000$.

Output specification

For each test case output a single line with a single integer: the sum of all elements in all arrays after D days, modulo $1,000,000,009$.

Example

input

```
3

4 1
1 2 1 3

1 10
47

2 10
1 2
```

output

```
34
47
33
```

The first test case corresponds to the example in the problem statement.

An additional challenge (just for fun)

The problem is solvable even if the constraints were, say, $N, D \leq 100\,000$. Can you see a solution that would be efficient enough to solve such a huge test case?

If you think you know such a solution, you can test it on the following input: $N = D = 100\,000$, array $= (1, 2, 3, \ldots, 99\,999, 100\,000)$. The correct output is $92\,629\,705$.

Solution

As stated in the task description: each midnight the virus replaces each array by all its possible contiguous subarrays.

The most straightforward approach is to simulate the process. This solution is very slow, but if you are patient, it should work for the easy data set. (A significant speedup can be achieved using the simple observation that arrays of length 1 just remain sitting in memory forever. Hence whenever we obtain an array of length 1, we can simply add its element to the result and throw it away.)

More simple observations

It should be clear that even after multiple days each array that is currently in memory represents a contiguous subarray of the single original array. Hence we can represent each array using only two indices – its start and end in the original array. This observation helps us write an even faster program, but this was not enough to Still, that observation alone is still not enough to solve the hard test case. To do so, we need to get rid of the exponential growth.

"Compressing" the memory

One way to solve the hard test case is to represent the current state of memory in a better way.

For example, consider the array $(1, 2, 1, 3)$ from the problem statement and try writing down what happens after two days. You should see many of the arrays multiple times. For example, the array $(2, 1)$ will be in memory four times.

And this is exactly the representation we need. We can store the contents of memory using a matrix of $O(N^2)$ numbers: for each pair of indices (i, j) we need to know how many times does the subarray $A[i : j]$ currently occur.

Given the current state of memory in this compressed form, we can easily produce the next state in the same form. At the end, for each subarray we just compute the sum of its elements and multiply it by the number of times this subarray currently occurs in the memory.

This gives us a solution with a polynomial time complexity, and it is fast enough to solve the hard test case.

Note that during the simulation phase we can compute the **subarray counts** modulo $P = 10^9 + 9$: if the same array occurs P times at any moment, we can throw away all P copies, because the sum of everything they'll produce will be divisible by P.

Dynamic programming

Another way of looking at the previous solution: what we are doing is just a form of dynamic programming. We may as well combine both steps of the previous solution into a single dynamic programming solution.

Let $A[d][i][j]$ be the answer for our problem, if the input array is the subarray starting with element i and ending with element j, and the number of days is d.

We are interested in the value $A[D][1][N]$. To compute it, we will compute all the values $A[d][i][j]$ for $0 \le d \le D$ and $1 \le i \le j \le N$.

One possible recurrence relation that allows us to compute these values efficiently looks as follows:

$$A[d+1][x][y] = \sum_{i=x}^{y} \sum_{j=i}^{y} A[d][i][j], \quad 1 \le x \le y \le N.$$

In words: To calculate the answer for the subarray from x to y and $d+1$ days, we consider all the arrays that will be produced from this array in the first day. For each of them we take the answer for d days, and add all these answers together.

As the calculation for the following day is dependent only on the previous day, it is sufficient to store the records only for the previous day and thus save space.

Faster solutions

The time complexity of the above solutions is $O(DN^4)$. We can speed it up to $O(DN^3)$ using a different recurrence relation:

$$A[d+1][x][y+1] = A[d+1][x][y] + \sum_{i=x}^{y+1} A[d][i][y+1].$$

We can even speed it up to $O(DN^2)$ using an even better recurrence relation

$$A[d+1][x][y] = A[d][x][y] + A[d+1][x+1][y] + A[d+1][x][y-1] - A[d+1][x+1][y-1].$$

(Try to prove that these recurrences actually work. Ideally, find a combinatiorial proof: what is represented by each term on the right hand side?)

Finally, here are some hints to solving the additional challenge:

For each x, compute the number of subarrays in the final state that contain the x-th element of the original array.

How to do this? For each such subarray, consider its entire history: in the beginning was the original array (from $a_0 = 1$ to $b_0 = N$), after the first step we had some subarray (from a_1 to b_1), this then produced another subarray (from a_2 to b_2), and so on. As the last subarray (from a_D to b_D) contains the x-th element, we get:

$$1 = a_0 \le a_1 \le \cdots \le a_D \le x \le b_D \le \cdots \le b_1 \le b_0 = N$$

All possible subarrays that contain x can be mapped to all such sequences of indices a_i, b_i.

Note that we can separately count the sequences of as and the sequences of bs, and then multiply the two counts. Also note that both sequences can be counted in the same way.

The number of valid sequences of as only depends on D and x. Can you work out the exact formula? (Hint: it's a single binomial coefficient).

Precompute factorials and their inverse elements to quickly evaluate binomial coefficients modulo a large prime.

Round and round it goes

Authors

Problemsetter: Michal 'mišof' Forišek
Task preparation: Michal 'mišof' Forišek, Monika Steinová

Problem statement

Last month, Dexter created the best infinite loop ever. However, his sister DeeDee used his computer to check her Facebook page and meanwhile she managed to erase two numbers from Dexter's program.

Here is the damaged program in Java. (Check the input file for the same code in other languages.)

```
public class R {

  static boolean magic(int x) {
    int foo = 0;
    while (x > 1)
      for (int y=2; ; ++y) {
        if (y==x) return foo<=0;
        if (magic(y) && x%y==0) { x/=y; ++foo; break; }
      }
    return true;
  }

  public static void main(String[] argv) {
    int where = ??????;
    int step = ??????;
    int best = where;
    while (step != 0) {
      where += step;
      if (where < best) { best=where; System.out.println("*"); }
      if (!magic(where)) break;
    }
  }
}
```

Problem specification

In both data sets your task is to replace the two "`??????`" strings by a pair of integers W and S. Note that these integers must be in a valid range for a signed 32-bit integer variable (e.g., `int` in Java). That is, your values must satisfy $-2^{31} \leq W, S \leq 2^{31} - 1$.

As the solution to the easy data set R1, submit any two integers W and S such that the program will run in an infinite loop if `where` is initialized to W and `step` to S.

As the solution to the hard data set R2, submit two integers W and S such that the program will run in an infinite loop if `where` is initialized to W and `step` to S. Additionally, out of all pairs (W, S) that satisfy the first condition, your values W and S must be such that the number of stars printed by the program is maximized. (Any such pair will be accepted.)

Input specification

For your convenience, the input file contains the program in multiple languages. All these programs are equivalent.

Output specification

Output a single line with two space-separated integers W and S.

Solution

Let's first focus on the function `magic`. Which integers are magic?

Clearly, all negative integers are magic, and so are 0 and 1.

We can now test the function on some small values. For example, if we run it for all numbers up to 20, we'll discover the following magic integers: 2, 3, 5, 7, 11, 13, 17, and 19. These are precisely the prime numbers. Which brings us to the hypothesis that for $x > 1$ the number x is magic if and only if it is prime.

Given this insight, the hypothesis is pretty easy to prove using mathematical induction. The function does the following: While $x > 1$, find the smallest prime divisor y of x, divide x by y and increment the counter. Return true if and only if the counter was never incremented – that is, if none of the numbers between 2 and $x - 1$ inclusive divide x.

(It should be obvious that neither y nor the counter can overflow during these operations.)

Solving the easy test case

Now we need to find a pair of integers W and S such that the cycle will only visit magic numbers.

Obviously, we can't have $S = 0$, as then the cycle does not run at all.

The easiest solution: let W be 0, 1, or any prime number, and let $S = -2^{31}$. The cycle will then alternately visit the values W (which is magic), and $W - 2^{31}$ (which is negative, hence magic). Note that $(W - 2^{31}) - 2^{31}$ overflows back to W.

This solves the easy data set.

A programmer's approach

It is possible to run the Sieve of Eratosthenes up to 2^{31} to find all magic numbers. The main bottleneck is probably memory – but if you use just one bit for each odd number in the range, 128 MB are enough. Our optimized implementation takes about 40 seconds on an ordinary machine.

Knowing (or assuming) some of the stuff outlined below it is then possible to write a function that tries all reasonable values of S and reports the largest cycle found.

However, the entire task can be solved completely without any programming. Below we explain how.

A closer look at infinite cycles

For the hard data set we need a little more insight into what happens during the cycle. Given some value of S, how many values will the cycle visit, and which values will they be?

First let's consider the case when S is odd. We claim that in this case the cycle will sooner or later visit each of the 2^{32} possible values. To prove this, it is enough to show that the shortest period of the sequence S, $2S$, $3S$, ... is exactly 2^{32}.

And that is quite obvious: if we have $aS \equiv bS \pmod{2^{32}}$, then 2^{32} divides $(a - b)S$. But as S is odd, this means that 2^{32} divides $a - b$.

Now for the general case: let $S = 2^a b$. It is now easily seen that in the sequence S, $2S$, $3S$, ..., all values are divisible by 2^a. Imagine that we now divide all values by 2^a. Using the same proof as above we can now show that the shortest period of this new sequence is 2^{32-a}. Hence the original sequence, when seen modulo 2^{32}, contains precisely all numbers divisible by 2^a.

In other words, if $S = 2^a b$, the cycle will visit exactly the same set of numbers as in the case where $S = 2^a$. For $b \neq 1$ the numbers will be visited in a different order, but right now this does not matter – if we want the cycle to be infinite, then *all the numbers* it visits must be magic.

Note that this observation was sufficient for the implementation-based solution to discover the longest cycle. We can simply find the smallest a for which there still is a $W \in \{0, 1, \ldots, 2^a - 1\}$ such that the cycle for W and $S = 2^a$ is infinite.

What's the longest infinite cycle?

Let $S = 2^a$ for some $a \in \{0, 1, \ldots, 31\}$. Then the cycle will visit 2^{31-a} negative numbers and 2^{31-a} non-negative ones, all equally spaced.

Clearly we can't have an infinite cycle for $a = 0$, as some of the numbers are not magic.

Now let $1 \leq a \leq 29$. As $a \leq 29$, the cycle visits at least four non-negative numbers. Let x be the smallest non-negative number visited by the cycle. If $x = 0$ or $x = 3$, the fourth non-negative number visited is divisible by 3 and larger than 3, hence it is not a prime, hence it is not magic. In all other cases one of the first three numbers visited has the same properties. (This comes from the fact that 2^a is not divisible by 3.)

Hence we can only see an infinite cycle if $a = 30$ or $a = 31$. We already saw what happens in the $a = 31$ case when solving the easy data set. Can $a = 30$ give us a longer cycle?

Sure it can. All we need is a starting value x such that $0 \leq x < 2^{30}$ and both x and $x + 2^{30}$ are magic.

There's plenty of such values. The easiest one to find is $x = 3$. (The number $2^{30} + 3 = 1\,073\,741\,827$ is indeed prime.)

Hence the longest infinite cycle visits exactly four values.

What's the maximum number of stars printed?

Clearly the program can only print a star if it reaches a previously unseen value. As each infinite cycle visits at most four distinct values, we surely cannot make the program print more than three stars. If we could make it print three starts, this would be optimal.

And such a solution is easy to achieve for any cycle of length four. For example, consider the cycle we found above. If we set $W = 2^{30} + 3$ and $S = -2^{30}$, we are done: In the first three steps the algorithm will visit the values 3, $3 - 2^{30}$, and $3 - 2^{31}$, and each time it will print a star.

This concludes our solution for the hard data set.

Antisort

Authors

Problemsetter: Michal 'mišof' Forišek
Task preparation: Michal 'Mic' Nánási

Problem statement

Usually, in programming contests the problem statement tells you what to do. We find that boring.

In this task we do the opposite. We will tell you something you **must not** do:

You must not sort the given sequence.

Problem specification

Given is a sequence S of distinct integers. Rearrange the sequence in any way. The only condition is that the resulting sequence must not be sorted – neither in ascending nor in descending order.

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 two lines. The first line contains a number N (between 3 and 1000, inclusive) giving the length of the sequence. The second line contains N space separated numbers that form the sequence. No two of these N numbers are equal, and all of these numbers fit into a 32-bit signed integer variable.

Output specification

The output format is the same as the input format, with one exception: The whitespaces may be arbitrary. Just make sure that every two tokens in the output are separated by at least one whitespace.

Each sequence in the output must contain the same numbers as the corresponding input sequence, and they must not be in sorted order.

Example

input

```
2

5
1 2 3 4 5

8
3 1 4 47 5 9 2 6
```

output

```
2

5
5 1 4 3 2

8
3 1 4 47
5 9 2 6
```

Solution

The solution is easy.

One possible approach is to randomly shuffle the sequence and hope that it will not be sorted. This should work on the easy data set. However, the hard data set contains about 20 sequences of length 3. If you randomly shuffle the sequence of length 3, in 2 out of 6 possible cases the sequence will be sorted – so unless you are really lucky, one of your random shuffles would fail.

Of course, the above solution can be easily fixed: `while sorted(sequence) do shuffle(sequence)`.

The above solution is really easy to implement, and you do not have to worry about tricky details. But for the ones who prefer avoiding unnecessary work, here's another approach: We can actually un-sort each sequence using a constant number of changes. For example, it is enough to sort the first three elements, and then to swap the first two.

Broadway

Authors

Problemsetter: Michal 'mišof' Forišek
Task preparation: Michal 'Mic' Nánási

Problem statement

The Manhattan in the New York City has really a nice topology. So nice, it is often idealized to a rectangular grid. If you want to go from corner $A = (A_x, A_y)$ to corner $B = (B_x, B_y)$, the shortest way has length $|A_x - B_x| + |A_y - B_y|$. Or so they told you in school.

The truth is, the correct definition of a Manhattan metric has to involve the Broadway – a road that leads across the neatly aligned system of streets and avenues. In this problem we finally correct this horrible mistake made by the mathematical community.

Problem specification

Given the two corners $A = (A_x, A_y), B = (B_x, B_y)$ and three rational numbers P, Q, R that describe the Broadway, your task is to find the length of the shortest path between points A and B.

The road network consists of the following roads:

- For each integer Z, there is an avenue described by the equation $x = Z$.
- For each integer Z, there is a street described by the equation $y = Z$.
- The Broadway is described by the equation $Px + Qy = R$.

When moving from A to B, we can only move along the roads and change roads at intersections.

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 one line containing seven numbers:
four integers A_x, A_y, B_x, B_y specifying the points $A = (A_x, A_y)$ and $B = (B_x, B_y)$,
and three rational numbers P, Q, R specifying the Broadway as explained above.

Output specification

For each test case output a single line containing the length of the shortest path from A to B. Output a sufficient number of decimal places. Your output will be judged as correct if it has an absolute or relative error at most 10^{-9}.

Example

input

```
2

2 0  -1 1  1.0 1.0 1.0

-2 3  4 -1  1.0 -0.1 0.47
```

output

```
3.414213562373
10
```

Solution

First we introduce some notations. The coordinates of point A are denoted A_x and A_y. Let A and B be two points. Their Manhattan distance is $d_m(A,B) = |A_x - B_x| + |A_y - B_y|$ and their Euclidean distance is $d(A,B) = \sqrt{(A_x - B_x)^2 + (A_y - B_y)^2}$.

Obviously correct solutions

The first observation we can make is that the shortest path from A to B is always entirely in the rectangle defined by points A and B. This is because if we take any path and replace the part outside the rectangle by a suitable part of the rectangle's boundary, we will get a shorter valid path.

One trivial solution is to construct a graph and find the shortest path in that graph. This solution worked for the easy data set. However, for the hard data set the number of vertices in such a graph would be roughly 10^{12}, which is way too much.

A better solution is based on the following observation: the shortest path through city enters the Broadway at most once. Again, this should be obvious: if we leave and later reenter the Broadway, this part of the path can be replaced by a shorter path along the Broadway.

Let $P = \{P_1, P_2, \ldots, P_K\}$ be the set of all crossings between Broadway and other roads in the rectangle. Then the shortest path between A and B has length

$$\min \left\{ d_m(A,B), \quad \min_{1 \le i,j \le K} d_m(A,P_i) + d(P_i,P_j) + d_m(P_j,B) \right\}$$

In words: Either we do not use the Broadway, or we go from A to some point P_i on the Broadway, then follow the Broadway to some P_j and from there use the streets and avenues again to get to B.

Still, trying out all pairs (i,j) in the above formula would be too slow. However, we can easily split the search into two parts. First, for each of the points P_i we will find the length D_i of the shortest path from A to P_i. This is just a shortest path problem in a simple graph with $K+1$ vertices (A and all of the P_i) and $2K-1$ edges (for each i an edge from A to P_i of length $d_m(A,P_i)$, and for each $i < K$ an edge between P_i and P_{i+1} of length $d(P_i,P_{i+1})$). Using a priority queue we can quickly compute all the values D_i. Then the answer is simply $\min_j D_j + d_m(P_j,B)$.

A fast and simple solution

Finally, there is a much simpler solution. The advantages: simplicity and speed. Disadvantage: it may not be immediately obvious that it always works. Anyway, the advantages are so significant that it was probably worth implementing it and trying to submit it even if you were not sure that it works.

The observation we base this solution on: If we actually use a segment P_iP_j of the Broadway, then there is an optimal solution in which both paths AP_i and P_jB are straight line segments.

This means that we only need to check four cases: both for A and for B there are exactly two points on the Broadway that are reachable using a straight line: one is reachable horizontally and one vertically. Hence this solution works in constant time.

Proof. Assume that the shortest path from A to B uses the segment P_iP_j of the Broadway. Consider the last straight segment of the path from A to P_i. Let Z be the last grid point on this path.

Extend the segment ZP_i to a line ℓ and examine the position of A with respect to this line. If it lies on ℓ, then the segment AP_i is the shortest way of getting from A to P_i. Now, A and P_j can not be on the same side of the line ℓ – otherwise we could shorten the path by moving P_i towards P_j. If the angle ZP_iP_j is acute, we can improve the path by entering the Broadway later, and if it is obtuse, we can improve the path by entering the Broadway earlier.

Cells

Authors

Problemsetter: Jozef 'YoYo' Šiška
Task preparation: Jozef 'YoYo' Šiška

Problem statement

Tim loves spreadsheets. Everything he does on a computer, he does in a spreadsheet. Track his expenses? Create a spreadsheet! Decide which car to buy? Create a spreadsheet to compare them! Make an inventory of his games? Create a spreadsheet! Decide which girl he loves most? ...

Unfortunately his spreadsheet software just crashed and he needs some of the data right now and does not have the time to install a competing office suit.

Problem specification

Given the formulas used in the cells of a spreadsheet, calculate the values of all the cells.

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 a single number N giving the number of expression. Each of the following N lines contains a single cell formula of the form "`CELL = EXPRESSION`", where `CELL` is the name of the cell and `EXPRESSION` is a mathematical expression consisting of cell names and the operators `+`, `-`, `*` and `/`. A cell name is a non-empty sequence of letters followed by a positive integer.

Each test case is correct: there are no cycles, and all cells referenced in expressions have definitions.

Evaluating expressions

When evaluating an expression, usual priorities apply: first we evaluate all `*` and `/` (left to right), and only then all `+` and `-` (again, left to right).

You may assume that the expressions are such that when evaluating the expression in correct order, the result and also all intermediate values will fit into 32-bit signed integer variables.

The operator `/` represents integer division which is always rounded **down**. The dividend will always be non-negative and the divisor will always be positive.

Output specification

For each test case output the calculated values of cells, one per line, in the form "`CELL = VALUE`". The rows in the output should be ordered alphabetically. (To compare two rows, take a look at the first character where they differ. The one with a smaller ASCII value goes first.)

Optionally, output a blank line between test cases.

Example

input

```
2

3
A47 = 5 + ZZ22
ZZ22 = 3
A9 = 13 + A47 * ZZ22

2
A1 = 4 / 7 + 4 / 7
B2 = 3 * 3 / 7
```

output

```
A47 = 8
A9 = 37
ZZ22 = 3

A1 = 0
B2 = 1
```

Solution

The straightforward way to solve this problem is to sort the cells so that we can evaluate them in the right order. This is called a *topological sort* and the easiest way to compute it is a depth-first search. Everytime we visit a new cell, we will first process all the cells it depends on and after that evaluate the cell itself.

For a "lazier" solution we can simply iterate through all the cells. In the first pass, we will compute at least the values of all cells that do not depend on any other cells. Each time we re-evaluate them, the cells already computed will keep their values and some others will become correct. If we repeat this enough times, all cells will eventually get computed. At least one cell will be computed in each step, thus for N cells we need to re-evaluate them at most N times.

The easiest way to parse and evaluate the cells is to use another program or tool that does it for us. We can, for example, take the input and turn it into a program that computes the answer. In most programming languages this requires just a few simple changes, such as declaring the cell names as variables and adding a semicolon at the end of each line.

Dragon Slayer

Authors

Problemsetter: Michal 'mišof' Forišek
Task preparation: Tomáš 'Tom' Záthurecký

Problem statement

You probably heard the story a thousand times: In some place far away, at some time long ago there was a kingdom ruled by a good king. This king had a daughter – a pretty princess. One day an evil dragon abducted the princess. A prince from some other kingdom decided to kill the dragon and save her. There was a fight, the dragon died, the princess was saved, and everybody lived happily ever after.

Of course, the story tellers have no idea how the fight with a dragon really looks like, so they make up a fake story where the prince uses a combination of brute force, expert sword handling and clever tricks to kill the dragon.

In reality, killing a dragon is a purely mathematical problem. The dragon has a variable (but always non-negative) number of heads. The only way to kill it is to reduce the number of its heads to zero.

The number of heads can only be changed by using a magic sword. Each magic sword is described by two parameters: c and g. Each time someone swings the magic sword, he must cut off **exactly** c heads. Afterwards, if the dragon is still alive (its number of heads is positive), it grows g new heads.

The sword cannot be used if there are less that c heads to cut. Note that the prince has a head. Thus he can kill a dragon with exactly $c - 1$ heads, but dies when doing so.

Problem specification

Our prince has found **two magic swords**: one with parameters c_1 and g_1, the other with parameters c_2 and g_2. The dragon has N heads.

Given the numbers N, c_1, g_1, c_2 and g_2, compute whether the prince can kill the dragon, and if so, whether he can survive the fight.

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 one line containing the numbers N, c_1, g_1, c_2 and g_2.

Output specification

For each test case output a single line containing one word: "`PRINCE`" if the prince can win the fight (i.e., kill the dragon and survive), "`DRAW`" if he can kill the dragon but has to die in order to do so, or "`DRAGON`" if the dragon cannot be killed.

Example

input

```
3

20 7 1 8 5

3 4 1 2 2

100 102 0 103 0
```

output

```
PRINCE
DRAW
DRAGON
```

In the first case, the prince can kill the dragon e.g. by using the swords in the order first, first, second.

In the second test case the second sword has no effect. As soon as the prince uses the first sword, both he and the dragon die.

In the third test case the prince cannot use any of the two swords.

Solution

First we show how to answer the simpler question: "Is it possible to cut exactly K heads?". This is based mainly on heavy case analysis. We consider three main cases:

- $g_1 \geq c_1$ and $g_2 \geq c_2$:

  In this case the swords cannot decrease the number of heads. So the only possible strategy is to use them to increase the number of heads to c_1 or c_2 and then to cut off all heads with a single swing.

  Without loss of generality we can assume that $c_1 \leq c_2$. Now we can consider the following cases:

  - $K < c_1$: Both swords are unusable, we cannot cut K heads.
  - $K = c_1$: One swing with the first sword does the job.
  - $c_1 < K < c_2$: The only possible strategy is to use the first sword several times and then finish with the second sword. This can be done if and only if $g_1 > c_1$ (the first sword can increase the number of heads) and $c_2 - k$ is divisible by $g_1 - c_1$.
  - $K = c_2$: We can just use the second sword.
  - $c_2 < K$: Since the number of heads cannot be reduced, we cannot cut K heads.

- $g_1 < c_1$ and $g_2 \geq c_2$ (or the reverse case):

  The first sword can reduce the number of heads while the second one cannot. This can be split into more cases:

  - $g_2 = c_2$: The second sword has no effect (if not used in the last step). The only possible strategy is to use the first sword several times to reduce the number of heads to c_1 or c_2 and then to finish with the appropriate sword. Such reduction to c (where $c = c_1$ or $c = c_2$) is possible only if $K \geq c$ (we can **reduce** to c), $K - c$ is divisible by $c_1 - g_1$ and $c - g_1 > 0$ (we do not get into negative values when using the first sword).
  - $g_2 > c_2$ and $K < c_2$: The second sword caoont be used, so the only possible strategy is to use the first sword seversl times. This can be done if and only if $K > c_1$ and $K - c_1$ is divisible by $c_1 - g_1$.
  - $g_2 > c_2$ and $K = c_2$: Trivial, just use the second sword.
  - $g_2 > c_2$ and $K > c_2$: The general strategy is to use the second sword y times to increase the number of heads, then to use the first sword x times to reduce the number of heads to c (where $c = c_1$ or $c = c_2$) and then finish with appropriate sword. This can be done only if $c - g_1 > 0$ (we do not get into negative values when using the first sword) and the equation $K + (g_1 - c_1)x + (g_2 - c_2)y = c$ has an integer solution with both $x \geq 0$ and $y \geq 0$.

- $g_1 < c_1$ and $g_2 < c_2$:

  Both swords reduce the number of heads. Without loss of generality we can assume that $c_1 \leq c_2$. The general strategy is to use the second sword y times then to use the first sword x times to reduce the number of heads to c (where $c = c_1$ or $c = c_2$) and then to finish with the appropriate sword. We can always use the second sword before the first for the following reason: The second sword can be used earlier because there are more heads available (since both swords reduce the number of heads) and the first sword can be used later because some sword will be usable at the end (and $c_1 \leq c_2$).

  This strategy can be used if only if $K \geq c$ (since we want to **reduce** to c) and the equation $K + (g_1 - c_1)x + (g_2 - c_2)y = c$ has an integer solution with $x \geq 0$, $y \geq 0$ and $K + (g_2 - c_2)y - g_2 \geq 0$

if $y > 0$. The last condition tells us that we will not run into the negative values when using the second sword. It can be transformed to $(c_1 - g_1)x \geq (g_2 - c)$.

And now the solution is simple: If it is possible to cut exactly N heads prince wins. Otherwise if it is possible to cut exactly $N + 1$ heads the fight ends in a draw and if neither is possible the dragon wins.

The only thing left is to solve some equations, so let us look at them:

In the first case we have to solve the equation of the form $ax - by = n$ ($a = c_1 - g_1$, $b = g_2 - c_2$ and $n = K - c$) where $a > 0$, $b > 0$ and $n \geq 0$. We are looking for an integer solution satisfying $x \geq 0$ and $y \geq 0$. It is clear that if n is not divisible by $\gcd(a, b)$ this equation has no integer solution. But also if n is divisible by $\gcd(a, b)$ then this equation has an integer solution (which can be found by the Extended Euclidean algorithm (`http://en.wikipedia.org/wiki/Extended_Euclidean_algorithm`)). Let (x', y') be such a solution. We can see that also $(x' + b, y' + a)$ is also its solution. This means that the equation $ax - by = n$ has an integer solution satisfying $x \geq 0$ and $y \geq 0$ if and only if n is divisible by $\gcd(a, b)$.

In the second case we have to solve the equation of the form $ax + by = n$ ($a = c_1 - g_1$, $b = c_2 - g_2$ and $n = K - c$) where $a > 0$, $b > 0$ and $n \geq 0$. We are looking for an integer solution satisfying $x \geq 0$ and $y \geq 0$ and $ax > l$ ($l = g_2 - c$) if $y > 0$. Let us assume that n is divisible by $\gcd(a, b)$ (otherwise the solution does not exist) and let (x_1, y_1) and (x_2, y_2) be some integer solutions to this equation. We can see that $ax_1 + by_1 = n = ax_2 + by_2$ which means that $a(x_1 - x_2) = b(y_2 - y_1) = m$ is a multiple of both a and b and thus also a multiple of $\mathrm{lcm}(a, b)$. This means that any solution of this equation can be expressed as $(x_1 + k\frac{\mathrm{lcm}(a,b)}{a}, y_1 - k\frac{\mathrm{lcm}(a,b)}{b})$. So we can easily find a solution with minimal nonnegative y and check other conditions.

Another approach

For reasonably small input values the problem is also solvable using a graph traversal algorithm (such as breadth-first search). It is possible to prove a statement of the form "if there is a way to kill the dragon, there is a way such that the number of its heads never exceeds $f(n, c_1, g_1, c_2, g_2)$." This makes the state space finite, and allows us to search it.

The exact form of the smallest possible function f is left as an exercise to the reader. During the contest, you could for example try to guess a suitable upper bound of its value.

Eyjafjallajökull

Authors

Problemsetter: Vladimír Koutný
Task preparation: Vladimír Koutný

Problem statement

Sometimes it's hard to see the forest, because all the trees get in your way. Other times, it's hard to see the volcano, because of all the ash in the air.

Problem specification

The world is represented by a rectangle of $M \times N$ cells. There is exactly one volcano somewhere in the world. The volcano occupies one of the cells. Your task is to find this cell.

Note that there are two separate data sets. The easy data set and the hard data set are completely independent and use two different maps. You have to solve each of them separately.

In the beginning, there is no ash anywhere. Our volcano erupts in rounds that correspond to your submits, and there is a changing wind which then distributes the ash over the world.

In each your submit you can check K locations of the map to see if there is currently any ash present or not. These probes are binary – it will report ash presence if and only if the cell contains at least one unit of ash.

Each round consists of the following steps, in order.

1. You submit a list of probes.
2. The volcano releases 100 units of new ash into the air at its location (no wind applied yet).
3. Your probes are evaluated.
4. A new wind is calculated and applied to the whole map. (In each round the current wind is the same everywhere on the map.)

The answer you'll receive for your submit will contain all probe results and a description of the wind that was generated in step 4.

Wind model

The wind is specified by a 5×5 matrix called the *wind matrix*. Rows and columns of a wind matrix are labeled from -2 to 2.

A wind matrix specifies how to distribute the ash from any given cell into its surroundings. In each wind matrix the numbers are non-negative integers that sum up to 100, and they should be interpreted as percentages.

Formally, suppose that there are a units of ash in a given cell (x, y) before the wind blows. Let W be a wind matrix. After the wind described by W blows, the ash from (x, y) will be redistributed into cells (x', y') with $x - 2 \leq x' \leq x + 2$ and $y - 2 \leq y' \leq y + 2$. More precisely, the cell $(x + d_x, y + d_y)$ will receive $\lfloor a \cdot W_{d_x,d_y}/100 \rfloor$ units of ash from this cell. Any ash that leaves the map boundary in this way is gone forever. Also note that (due to rounding) the total amount of ash the cells receive may be less than a.

It is important to note that the ash from all cells is being redistributed at the same time. (The process resembles a blur operation in a graphics editor.)

Input specification

The input file contains 3 integers: M, N and K. The map size is $M \times N$ cells. In each submit, you may specify up to K probe locations.

The volcano will be placed in a cell such that in any direction there are at least two other cells before we reach the map boundary.

Output specification

In each submit you can specify up to K cell locations x_i, y_i ($0 \leq x_i < M$, $0 \leq y_i < N$). All values should be separated by whitespace. In case one of your probes matches the volcano location, the game ends and the data set is correctly solved.

Otherwise, you will receive a "`WRONG ANSWER`" message with the following information:
– the current round number,
– ash presence values for all your probes (0 = no ash present, 1 = some ash present),
– and a new wind matrix.

Submission limit

Please note that the submission limit for each data set is 10 submits, as is the case for any other task.

Be careful about the submissions you make. Try not to waste any, and always double-check whether the one you are about to send is correct.

Example submission (in the middle of a game)

input

```
1 3
4 9
```

output

```
Round: 3

Probe [1,3]: 0
Probe [4,9]: 1

Wind:
0 0 5 5 0
0 5 30 10 0
0 0 20 15 0
0 0 10 0 0
0 0 0 0 0
```

Solution

Our solution is based on the same idea as in similar search problems: We maintain the set of all possible (consistent with previous measurements) volcano positions and after any submit (and corresponding response from tester) we check which of them are still valid. Since the maps are small (20×20 in the hard case) we can use a brute-force approach here. For each cell C in the map we can compute the ash distribution for the case that the volcano is on cell C. With this information it is easy to determine which cells are are still valid candidates for volcano position (the response from tester matches the computed ash distribution).

The only problem to now is find the good positions for the probes. An optimal solution would select such positions that minimize the worst-case number of submissions required to find the volcano. But this seems to be very hard (at least we cannot see a way to compute it efficiently). Our solution tries to minimize the size of the set of all possible volcano positions after the worst-case tester response. It is not optimal, but it seems to be reasonably good (at least it is able to solve both test cases)

It works as follows: Consider the position p_1 of the first probe. It divides the set V of all possible volcano positions int two groups: V_1 in which there is some ash in position p_1 and V_0 in which there is no ash there. We choose p_1 in such a way that minimizes the size of larger of these groups. With the second probe (at position p_2) the situation is similar but now each of the two above groups is divided. So we have four groups of positions and we choose p_2 in such a way that minimizes the size of the largest of them. We can continue in this fashion until we find the positions for all K probes.

Of course it would be better to select all the probes at once, but searching through all possibilities would be too slow.

Flawed Code

Authors

Problemsetter: Michal 'mišof' Forišek
Task preparation: Michal 'mišof' Forišek

Problem statement

Michael is teaching an algorithms class. In the last lecture he gave his students the following problem:

> Given are several cubes of various sizes. You can make towers out of the cubes by placing them one atop another, in any order. Your task is to find the *largest possible* height H such that is is possible to build *two separate towers* of height H.
>
> For example, if the cube sizes are (8,6,3,3,3,2), then the solution is 11:
> we can build one tower as 8+3, the other as 6+3+2, and a cube of size 3 remains unused.

This task proved to be too difficult for his students, so each of them just implemented some clever hack that sometimes worked and sometimes failed. More precisely:

- Alla implemented a brute force solution. She sorts the cubes in descending order. If there are more than 15, she just takes the 15 largest cubes. Then she tries all possible ways how to build two towers and finds the best one in which the towers are of equal height.
- Bob finds and outputs the best solution that only leaves at most two cubes unused.

  To do so, he used dynamic programming to implement a function that takes a set of cubes and checks whether it is possible to build two equal towers using *all of them*. He then calls this function multiple times, each time with a different subset of the given set of cubes.
- Chermi loves greedy algorithms. His program finds the largest X for which his greedy algorithm manages to build two towers of height X each.

  His algorithm to build two towers of height X is simple: He first sorts the cubes in descending order, and then processes them one by one. Each time he checks whether he can add the cube to the currently smaller tower without exceeding X, and if he can, he does so.
- Dominika's solution is perhaps the most clever one. She used dynamic programming. For each H she computed the number of ways in which a tower of size H can be assembled. The answer she outputs is the largest X such that the tower of height $2X$ can be assembled, and the tower of height X can be assembled in at least two different ways.

Problem specification

You are given the four programs made by Michael's students (for your convenience, each of them is implemented in several languages). Your goal is to help Michael show the students their errors by providing inputs for which the students' solutions fail.

Input specification

The input file contains implementations of the four students' programs in C++, Pascal, and Python.

Output specification

As the output for the **easy** data set F1, output 4 lines, containing 2 valid test cases. Each test case consists of two lines. The first line contains the number N of cubes ($1 \leq N \leq 100$). The second line contains N space-separated integers – the cube sizes. The sum of all cube sizes must not exceed 10 000.

The submission will be judged as correct if it breaks at least three of our solutions. In other words, at most one of our four solutions can give the correct answer for both your test cases.

As the output for the **hard** data set F2, output 2 lines containing a single valid test case.

The submission will be judged as correct if it breaks all four solutions. That is, none of the four solutions should give the correct answer for your test case.

Solution

In this solution we first present an approach by hand and then we discuss how the answer could be found by a program.

Alla's solution

Any test case with an optimal solution that uses more than 15 cubes works.

Bob's solution

If the largest cube is larger than the sum of all other cubes, it can not be used in the solution. Hence Bob's solution will fail for any input of the form $(5000, 2500, 1250, \dots)$ where the remaining cubes form a perfectly solvable instance that sums to less than 1250.

Another way is to use the opposite end of the spectrum: to add small useless cubes. Start with any solvable instance, for example $(1, 1)$. Multiply all values by 10, and add cubes of sizes 4, 2, and 1. These three cubes are now completely useless. So another example where Bob's solution fails is the case $(10, 10, 4, 2, 1)$.

Solving the easy data set

To solve the easy data set, it was necessary to create two test cases. One of them had to break at least two of our solutions. Probably the easiest way to do so was to break Alla's and Bob's solutions using the same case.

The ideas above can easily be combined into a single test case.

For example, for the input $(5000, 2500, 1250, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1)$ the correct output is 7, but Alla's algorithm outputs 6 and Bob's outputs 0. Together with any one test case shown below, this test case solves the easy data set.

Chermi's solution

If there are exactly two equally large largest cubes, the algorithm will use them in different towers. We can break it by constructing a test case that is only solvable if they are in the same tower.

A simple test case with this property: $(3, 3, 2, 2, 2)$. The correct output is 6 (one of the towers is $3 + 3$, the other $2 + 2 + 2$), but Chermi's solution misses it and only finds the answer 5.

Dominika's solution

This solution is still incorrect. The fact that height X can be achieved in (at least) two ways still does not guarantee that it can be achieved in two *disjoint* ways. And not even the fact that $2X$ can be achieved helps – because $2X$ may be achieved using a different subset of cubes.

One small example when this occurs: $(10, 9, 8, 3, 3)$.

Clearly the only solution for this input are two towers of height 3 each. We cannot use any of the larger cubes.

Still, Dominika's solution would incorrectly output 12. This is because we can assemble a tower of height 24 (as $10 + 8 + 3 + 3$), and we can assemble a tower of height 12 in two ways (using 9 and one of the two 3s).

Combining the ideas

To solve the hard data set we need to combine the properties of all above inputs.

First of all, we can find a test case that breaks the last two solutions. One such case is $(5, 5, 4, 3, 2)$. The correct solution is $7 = 5 + 2 = 4 + 3$. Chermi's solution fails to find it and only returns 5. Dominika's solution is confused into thinking that the answer is 8, because it is possible to get 16 (as $5 + 5 + 4 + 2$) and to get 8 in two ways (as one of the 5s and the 3).

We can now augment this case to break Bob's and Alla's solution as well. For example, nothing changes if we multiply all item sizes by 100: the test case $(500, 500, 400, 300, 200)$ still breaks both Chermi's and Dominika's solutions.

We can now add some small useless items (these will cause Bob's solution to fail) and small useful items (these will cause Alla's solution to fail, as she won't consider them).

One such test case is $(500, 500, 400, 300, 200, 44, 22, 11, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1)$. This test case breaks all four solutions:

- The correct solution is $705 = 500 + 200 + 5 \times 1 = 400 + 300 + 5 \times 1$.
- Alla's solution outputs 703, as she does not use the last three 1s.
- Bob's solution outputs 0, as none of the items 44, 22, and 11 can be used in any valid solution.
- Chermi's solution outputs 505, because for any larger value it uses both cubes of size 500, and then it is unable to build two equal towers.
- Our construction guaranteed that Dominika's solution will output at least 800, therefore the output is surely incorrect. (The actual output is 987.)

Correct algorithm for the "two towers" problem

If we want to automate the searching for a case where all solutions fail, we need to know the right answer for that case – in other words, we need to solve the problem.

Our solution is based on dynamic programming. For each A and B we answer the question: "Suppose that I take the first A cubes and use some of them to build two towers whose heights differ by B. What's the maximum possible height of the taller tower?"

It is easier to compute these values in the forward direction: Once I know the optimal answer for some A and B, I can update three new values: either I don't use cube $A + 1$ at all, or I add it to the taller, or to the shorter tower.

Automated solving

When given the four programs, many solvers may be tempted simply to throw lots of random cases at them until a suitable one is found. This is, of course, another way of solving the task, but it's not completely straightforward.

The first thing to notice about the random inputs: the largest ones are not the hardest ones.

The simplest problem similar to the one we are solving is the Partition problem: given is a list of positive integers, divide them into two lists with sums as close to each other as possible. Sometimes this problem is stated as a decision problem: is it possible to create a perfect partition, where the sums of the two parts differ at most by one?

The Partition problem exhibits a phase transition. If we have few large items, it's almost never possible to split them evenly. But if we have many small items, it is almost always possible, and there are usually countless ways to do so.

For the Partition problem it is known that if we generate n numbers from the range 1 to 2^m, then for $m/n < 1$ almost all instances have lots of perfect partitions, while for $m/n > 1$ almost no instances

have a perfect partition. See `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.89.9577` for more on this.

We can observe a very similar pattern when dealing with our problem. Almost all instances with $N = 100$ have an optimal solution. (In our case, the optimal solution means that we compute the sum of all items. If it is even, there are many ways how to build two equal towers using all cubes. If it is odd, we throw away the smallest odd number and then we are in the same situation.)

Among other things, this means that Bob's and Dominika's solutions will work for almost all instances with large N. If we want a counterexample to these solutions, we have to look elsewhere.

On the other hand, to break Alla's solution we need at least 16 items. So the best place to look for hard cases is for N between, say, 16 and 20.

Another issue is how to generate the cube sizes. Uniformly at random is not exactly the way to go: essentially none of your items will be small.

Our simplest successful attempt to randomly generate the test case worked as follows:

- Uniformly at random, pick N between 16 and 20.
- To generate each of the N numbers, first uniformly at random pick D between 1 and 10, and then uniformly at random generate a number that has exactly D binary digits.
- Correctly solve the test case.
- Run Bob's solution. If it fails, run the other three to verify.

This approach can easily find a good test case in under a minute. Here's one randomly found test case: $(946, 508, 181, 46, 42, 39, 33, 23, 19, 9, 7, 7, 6, 5, 5, 2, 1)$. The correct answer is 212, but the four algorithms output 94, 0, 211, and 928.

Grid is good

Authors

Problemsetter: Michal 'mišof' Forišek
Task preparation: Peter 'Bob' Fulla

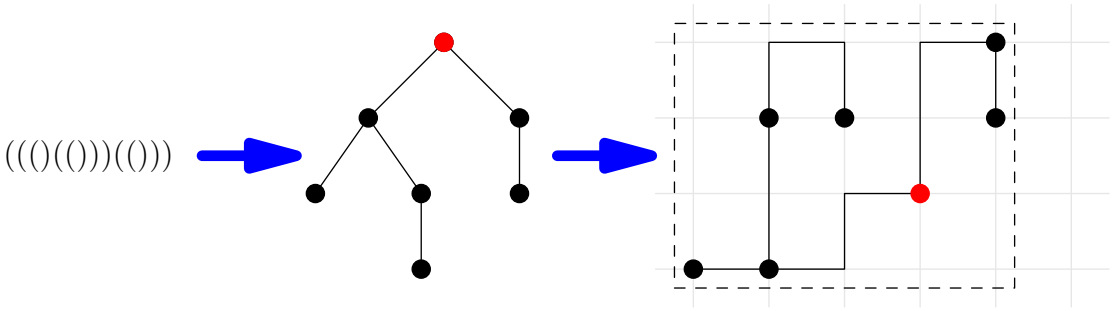
Problem statement

Little Gordon enjoys nature and especially trees. He would like to sketch some of his most favourite ones. However, it seems that he doesn't have enough grid paper – and that's where your help is needed...

Problem specification

You are given a rooted binary tree and a number K. Pick a rectangle on the grid with sides parallel to the coordinate axes. The chosen rectangle may contain at most K grid points (including those on its perimeter). Draw the tree into the rectangle.

You have to place each node onto one of the grid points in your chosen rectangle. Different nodes must be placed onto different points. You then have to draw each of the edges. Each edge must be drawn as a polyline that lies entirely on the grid and does not exit the rectangle. No two edges may share a common point other than an endpoint they have in common.



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 two lines. The first one contains the integer K, the second one a string of parentheses that encodes the tree.

An empty tree is encoded as an empty string. A tree consisting of a root node and its two subtrees is denoted `(LR)`, where `L` and `R` are the codes of those two subtrees. Note that a leaf has two empty subtrees, hence each leaf is denoted by the string `()`.

The trees in the hard data set are identical to those in the easy data set. The data sets differ only in the amount of grid points you can use (the number K).

Output specification

For each test case output a single line with two integers r and c: the number of rows and the number of columns of grid points your chosen rectangle contains. Note that rc must not exceed K.

Afterwards, output $2r-1$ rows, each containing $2c-1$ characters: an ASCII art drawing of the tree. In the drawing, grid points correspond to characters whose both coordinates are even (indexed from 0).

The root node should be marked as `H`. For each other vertex: if it is the first child of its parent, label it `L`, otherwise label it `R`. The edges are drawn using `-`, `|` and `+` symbols (ASCII 45, 124, and 43; follow the example output). All the remaining characters should be `.` (ASCII 46).

Any valid drawing will be accepted.

Example

input

```
2

4
(()())

36
(((()(()))(())))
```

output

```
2 2
H-R
|..
L..

4 5
..+-+.+-R
..|.|.|.|
..R.L.+.L
..|...|..
..+.+-H..
..|.|....
L-L-+....
```

In the second test case the chosen rectangle contains 20 grid points, which is less than the limit (which is 36).

The second test case and its solution that corresponds to this ASCII art drawing are shown in the picture in the problem statement.

Solution

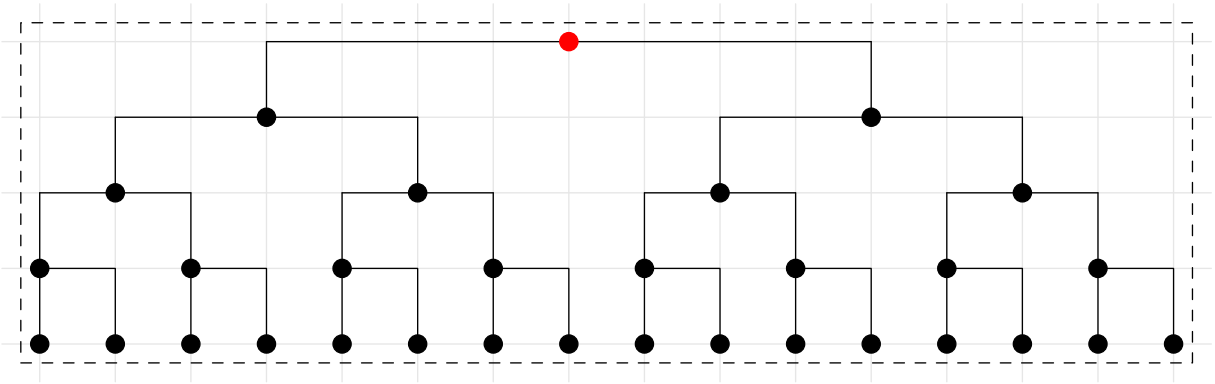
Let's take a look on binary trees in the input files. They are quite balanced: except for a few small test cases they look like perfect binary trees. Drawing small trees can be done by hand, therefore we will focus on large perfect trees.

First attempt

A simple way how to draw any binary tree into a grid is shown in the picture below. We divide the nodes into levels according to their distance from the root and we draw each level into a separate row in a grid.

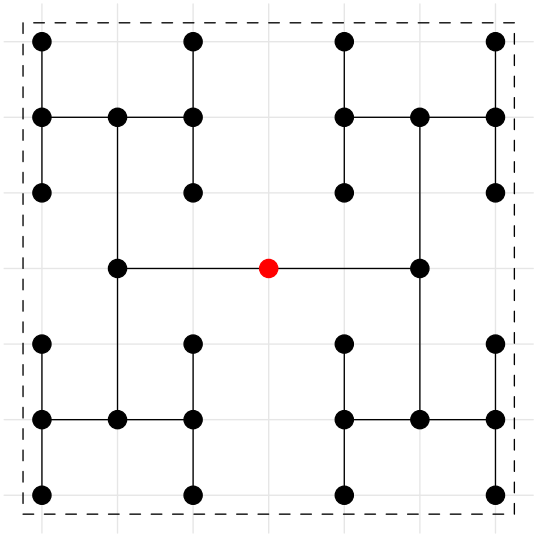
If the given tree is not perfect, we just omit missing parts.

A perfect binary tree with n nodes has height $\Theta(\log n)$ and $\Theta(n)$ leaves. Drawing a tree using this method requires $\Theta(n \log n)$ grid points.



Linear solution

A bit more complicated method for drawing binary trees is shown in the picture below.



An easy way how to implement it is using recursion. First we draw (recursively) the left subtree to the left half of the rectangle, then we draw the right subtree to the right half of the rectangle and after it the root node to the middle.

In the next level of recursion we draw the left subtree to the top half of the remaining rectangle and the right subtree to the bottom half. In the next level of recursion we again change mutual position of the children's subtrees (horizontal/vertical).

Let $T(n)$ denote the number of grid points in one row we need to draw a perfect binary tree with n nodes using this method. Then $T(1) = 1$, $T(3) = 3$ and $T(n) = 2T((n-3)/4) + 1$ for $n \geq 7$. By master theorem $T(n) = \Theta(\sqrt{n})$.

The number of grid points in one column is always less then or equal to $T(n)$, thus the total number of grid points needed is $\Theta(n)$.

Summary

The easy input was solvable by $\Theta(n \log n)$ algorithm, for the hard input some better approach was necessary (e.g. the presented linear solution). The small test cases in both input files required additional optimization by hand.

Hash

Authors

Problemsetter: Vladimír 'U$Amec' Boža
Task preparation: Vladimír 'U$Amec' Boža, Peter 'ppershing' Perešíni

Problem statement

When dealing with large files, hashing is a nice way how to reduce the file into a small *fingerprint* – a short string that somehow represents the contents of the entire file.

Hashing has multiple uses, for example it gives us a nice integrity check when transfering the file. It is also used in most digital signatures – as signing is usually slower than hashing, you first hash the file, and then sign the fingerprint.

Most of the hash functions have the property that even if just one byte of the file changes, we have to process the entire file again in order to find the new hash value.

In this task we investigate some options how to design a hash function that would allow us to update the hash value quickly after we make a change to the hashed file. We will use the following approach:

1. We split the file into blocks $w_1, \ldots, w_n$ of size 256 bytes (for simplicity we will assume that the length of the file is divisible by 256).
2. We calculate the hash of each block: $H_i = H(w_i)$, where H is some cryptographic hash function whose output is m bits long. We will use the well-known MD5 hash function with $m = 128$ bits.
3. We merge the block hashes together. This can be done in two simple ways. We can either use bitwise xor (so the final hash would be $X = H_1 \oplus H_2 \oplus \cdots \oplus H_n$), or we can add the individual hashes as numbers, computing modulo 2^m (so the final hash would be $A = (H_1 + H_2 + \cdots + H_n) \bmod 2^m$). We will call the two approaches XOR-HASH and ADD-HASH, respectively.

Problem specification

Invert the hash functions described in the problem statement. That is, both for XOR-HASH and for ADD-HASH you should provide a file with a prescribed hash value.

Note that while there are known attacks to find collisions in MD5, so far it is not known how to invert this function efficiently.

Input specification

You are given a sample implementation of the MD5 hash function along with a testing program and example output files for a team with TIS 12345678901234567890.

Output specification

As the output for the **easy** data set H1, output the hex dump of a file which has $n = 256$ blocks (i.e., its size is 65536 bytes). After calculating the XOR-HASH of this file, the hexadecimal notation of the result must be `00000 00000 00XXX XXXXX XXXXX XXXXX XX`, where the string of 20 `X`s should be replaced by your TIS.

As the output for the **hard** data set H2, output the hex dump of a file which has $n = 128$ blocks (i.e., its size is 32768 bytes). The ADD-HASH of this file must be `00000 00000 00XXX XXXXX XXXXX XXXXX XX`, where the string of 20 `X`s should be replaced by your TIS.

The hex dump of a B-byte file should consist of B pairs of hexadecimal digits separated by any whitespace. For example, if the first three bytes of your file have ASCII values 97, 32, 10, the hex dump of this file would start "`61 20 0a`".

Instead of the provided MD5 implementation, you should be able to use any software that computes MD5 hashes of files. If your operating system contains the `od` application (octal dump, a part of GNU coreutils), you can easily create the hex dump of a file using the command `od -v -A n -t x1`.

Solution

Easy input solution

Let's take 2 strings: $s^0 = s^0_1 s^0_2 \dots s^0_{256}$, $s^1 = s^1_1 s^1_2 \dots s^1_{256}$. And hash their blocks, so we get: $h^i_j = H(s^i_j)$. Now we want to find $y_1, y_2, \dots, y_{256} \in \{0, 1\}$ such that:

$$h^{y_1}_1 \oplus h^{y_2}_2 \oplus \dots h^{y_{256}}_{256} = X$$

where X is our desired result (basically our idea is to select suitable blocks from 2 strings). Now we know that: $h^{y_j}_j = h^1_j \cdot y_j + h^0_j \cdot (1 - y_j)$ (this holds because y_j can be just $0, 1$).

A XOR is bitwise independent operation. So let $h^{i(b)}_j$ be b^{th} bit of h^i_j. Now we get (in field Z^2 - or just computing modulo 2):

$$h^{1(b)}_1 y_1 + h^{0(b)}_1 (1 - y_1) + \dots + h^{1(b)}_{256} y_{256} + h^{0(b)}_{256} (1 - y_{256}) = X^{(b)}$$

for $b = 1, 2, \dots, 128$. This is just system of 128 equations with 256 unknowns and can be solved by Gausian elimination.

Our output will then be $s^{y_1}_1 \dots s^{y_{256}}_{256}$.

Hard input solution

We will not use any property of MD5 hash function. So firstly we generate sets $L_1, L_2, \dots, L_{128}$, where L_i contains m hashes of random strings. Out task is to find such $x_1, x_2, \dots, x_{128}$, that $x_i \in L_i$ and $(x_1 + x_2 + \dots + x_{128}) \bmod 2^{128} = X$ (X is our desired output) or more generally find set $L_{1,128}$ where: $L_{1,128} = \{(x_1, x_2, \dots, x_{128}) | (x_1 + x_2 + \dots + x_{128}) \bmod 2^{128} = X\}$.

Let's make our task little bit easier. We will have only 4 sets, and we want $(x_1 + x_2 + x_3 + x_4) \bmod 2^{128} = 0$.

We can generate sets of size 2^{32}, then make sets $L_{1,2} = \{(x_1, x_2) | x_1 \in L_1 \wedge x_2 \in L_2\}$, $L_{3,4} = \{(x_3, x_4) | x_3 \in L_3 \wedge x_4 \in L_4\}$, which will have size 2^{64} and then join this two sets into $L_{1,4}$. This joining can be done in 2^{64} time, because we can fill hash table with elements of $L_{1,2}$ and then process elements of $L_{3,4}$ by looking for coresponding element is hash table. And we will have good chance of finding solution because $2^{64} \cdot 2^{64} = 2^{128}$.

But 2^{64} is still pretty much. We can make other extreme situation, where we want $L_{1,2} = \{(x_1, x_2) | x_1 + x_2 = 0 \wedge x_i \in L_i\}$, now we need L_i of size 2^{64}, which is still not good.

Maybe something in the middle will be better. For example $L_{1,2}$ contains elements, whose sum end by l zeroes. Let $low_l(x)$ be the last l bits of x. Let: $L \bowtie_l L' = \{(x, x') | x \in L \wedge x' \in L' \wedge low_l(x + x') = 0\}$.

Then we make: $L_{1,2} = L_1 \bowtie_{128/3} L_2$, $L_{3,4} = L_3 \bowtie_{128/3} L_4$ and $L_{1,4} = L_{1,2} \bowtie_{128} L_{3,4}$. Size of L_i will be $2^{128/3}$. Let's see what we will get. Expected size of $L_{1,2}$ will be $2^{128/3} \cdot 2^{128/3} / 2^{128/3} = 2^{128/3}$ (number of pairs divided by chance of having last 128/3 bits zero). Size of $L_{3,4}$ will be same. And expected size of $L_{1,4}$ will be $2^{128/3} \cdot 2^{128/3}\ 2^{2 \cdot 128/3} = 1$ (we need first $2 \cdot 128/3$ bits zeroed). If we count time complexity we get $7 \cdot 2^{128/3} = 7 \cdot 2^{43}$, which is better.

We can generalize this construction for 128 sets. Just we will have bigger tree and first join will be on last 16 bites, next on last 32 bites (but we have last 16 already zeroed, so we are joining 16 bites), $\cdot$, and at last we join 96 bites and finnaly 128 bites.

Now we have just one problem. We can make $x_1 + x_2 + \dots + x_{128} = 0$. But how to make $x_1 + x_2 + \dots + x_{128} = X$? It's easy. We just alter one set, for example L_1 as $L'_1 = \{x | x = y - X, y \in L_1\}$ and find zero solution for $L'_1, L_2, L_3, \dots, L_{128}$.

Literatúra

[1] Wagner, *Generalized birthday problem* http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.94.6281&rep=rep1&type=pdf

[2] M Bellare, D Micciancio, *A new paradigm for collision-free hashing: Incrementality at reduced cost* http://taz.newffr.com/TAZ/Cryptologie/papers/inchash.pdf

Inside job

Authors

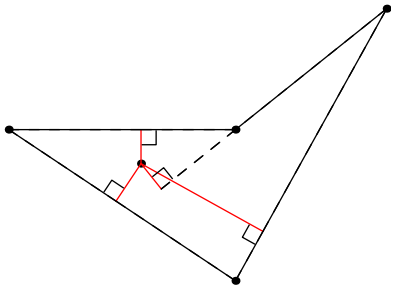
Problemsetter: Michal 'mišof' Forišek
Task preparation: Michal 'mišof' Forišek, Lukáš Poláček

Problem statement

Jan is throwing darts into a polygon painted on the wall. His aim is really lousy. We will assume that the points he hits are uniformly distributed inside the polygon.

(Formally, let A be the area of the polygon. Given any part of the polygon with area B, the probability that Jan hits the given part is B/A.)

Each time Jan throws a dart, Monika measures the altitudes from the point where the dart landed onto all sides of the polygon. In other words, for each side of the polygon she measures the distance between the dart and the line that contains the given side. She then adds all the distances together and announces the result.



Monika computes the total length of the red lines in the image.

Problem specification

Given is the polygon. Compute the expected value of the number Monika will calculate.
(If you do not know what an expected value is, imagine it as the average of very many attempts.)

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 number of vertices N. Each of the next N lines contains two integers: the coordinates of one point. You may assume that $N \leq 500$ and that all polygons are simple: the boundary of a polygon never touches or intersects itself.

The polygons in the easy input are special: They are convex, N is even, and each pair of opposite sides is parallel.

Output specification

For each test case output a single line with the expected value of the sum of all distances. The number must be printed with sufficiently many decimal places (we recommend 10 or more). Answers that have an absolute or relative error at most 10^{-9} will be accepted.

Example

input

```
2

3
0 0
3 0
3 4

3
4 2
6 1
7 3
```

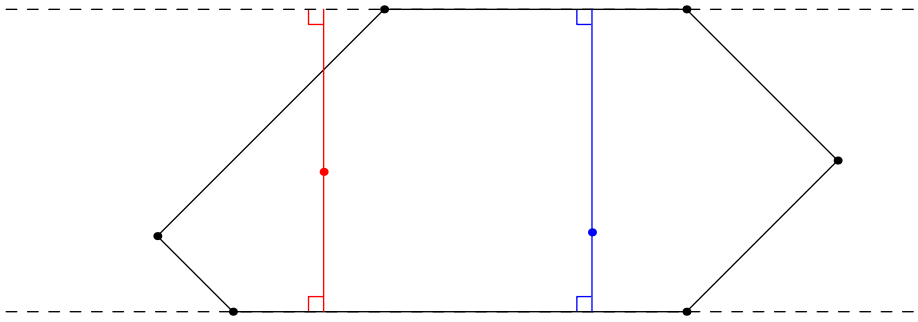
output

```
3.133333333333
2.017758261695
```

Solution

To solve the easy data set, let's examine the constraint for this data set carefully: The polygons in the easy data set are special: They are convex, N is even, and each pair of opposite sides is parallel.

Well, so we know that each pair of opposite sides is parallel. How should this help us? Let's draw a picture and focus on one such pair:



As the polygon is convex, the entire polygon lies in the strip determined by the lines on which our two sides lie (these are the dashed lines in the picture).

Now, regardless of where we pick the point (two possible choices are shown in red and blue) the sum of distances from the point to these two sides will always be the same: it will be equal to the distance of those two lines.

This gives us a really simple solution: for each pair of opposite sides, calculate the distance of the lines on which they lie, and add all these distances together to obtain the answer.

(Another way of stating the same thing: pick any point inside the polygon and compute the sum of altitudes from this point onto all sides. This sum is the same for all points in the polygon, hence this is the answer we seek.)

Hard data set

In the hard data set the polygons were way more tricky, most of them were not even convex. We will present a solution with time complexity $O(N^3 \log N)$. This solution is way from being optimal, but it is fast enough to solve the data set in a matter of seconds, and simple enough to be implemented from scratch.

First of all, note that by linearity of expectation the expected value of the sum of all distances is equal to the sum of expected values of all distances. This means that we can split the problem into N subproblems: for each of the N edges we will separately compute the expected altitude from a random point onto that edge, and the sum of these expected distances is our answer.

How to compute the answer for a single edge? We can translate and rotate the polygon so that our edge lies on the x axis. Now imagine that we take our polygon and cut it along a line that is parallel to the x axis and passes through the point $(0, y)$. The intersection of this line and our polygon is a (possibly empty) set of disjoint line segments. Let $f(y)$ be the sum of their lengths for a given y. Let S be the area of the polygon.

We can now express the expected distance as

$$\frac{1}{S} \int_{-\infty}^{\infty} |y| \cdot f(y) \, \mathrm{d}y$$

Of course, this integral over an infinite continuous domain is just the most general way of describing the solution. First of all, it is sufficient to integrate over a finite range, namely $[\min y_i, \max y_i]$, where y_i are the vertical coordinates of all vertices of the polygon.

As the second step, imagine that before we compute the integral, we cut the polygon into horizontal slices by lines that pass through all its vertices. Now we can compute the integral over the entire range as the sum of integrals over each slice.

Why does this help us? The reason is that inside each slice the function f is linear. All we have to do is to find its exact form, and then we can easily compute the value of the integral.

In our implementation we proceed as follows:

- Sort the y coordinates of all vertices.
- For each non-empty slice (y_{lo}, y_{hi}):
  - Create the list of all edges that cross this slice.
  - Sort this list to put the edges into the order in which they cross the slice.
  - Using the sorted list, compute $x_{lo} = f(y_{lo})$ and $x_{hi} = f(y_{hi})$.
  - Using x_{lo}, x_{hi}, y_{lo}, and y_{hi} compute p and q such that on (y_{lo}, y_{hi}) we have $f(y) = py + q$.
  - Evaluate the integral on (y_{lo}, y_{hi}).

(One technical note. Formally, the value x_{lo} computed from the list of edges that cross the slice may differ from the actual $f(y_{lo})$ if our polygon contains a horizontal edge at y_{lo}. Correctly, the above paragraph should claim that $x_{lo} = \lim_{y \to y_{lo}+} f(y)$. This is merely a technicality that does not affect the correctness of our solution.)

Faster solutions

The task is pretty similar to simply computing the area of the polygon. And the analogy continues: the division into strips is not necessary. It just helped us visualize the solution.

Instead, we can compute one almost identical integral for each original edge, and adding up these (with opposite signs for edges going up and down). It's not even necessary to sort the edges. This solution, while conceptually a bit more difficult, is significantly faster – its time complexity being $O(n^2)$ – and simpler to implement.

Can you see a way how to improve it even more? At the moment, we don't `:-)`

Jimmy the Acting Teacher

Authors

Problemsetter: Lukáš Poláček
Task preparation: Lukáš Poláček

Problem statement

Jimmy is an acting teacher. He wants to introduce a new acting exercise to his class. There are exactly N men and N women taking his course. There are N tables in the room. Each table is assigned a dialogue for a man and woman. The dialogues are different at different tables. Jimmy wants all students to try every dialogue exactly once. He also wants every man to play with every woman, so that he can see which actors should play together in a play.

Jimmy wants to keep things as simple as possible. First, he took a black marker and labelled the tables from 1 to N. Then he took a blue marker and a red marker and wrote a blue number and a red number onto each table. Whenever a couple finishes the dialogue, the man reads the blue number on their table and goes to the table with that number, and the woman does the same with the red number.

Each actor chooses one table at the beginning of the class and will start by playing the dialogue at the chosen table. Each dialogue takes 5 minutes. After the 5 minutes are over, all students move simultaneously according to the blue and red numbers. All movement happens in an instant.

Problem specification

Finding the right 2 numbers for each table seems to be a very difficult task. Please help Jimmy. For a given N, find one way how the blue and red numbers may look like.

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 the integer N denoting the size of the class.

Output specification

For each test case output a single line with word "`Impossible`" if it is impossible to find the right 2 numbers for every table. Otherwise output 2 lines. The first line will contain a space-separated list of blue numbers, the second line will contain a space-separated list of red numbers. (The i-th number in each list is the one written on table i.) Output a blank line between test cases.

Example

input

```
3

2

3

5
```

output

```
Impossible

2 3 1
3 1 2

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

In the first test case there is no solution: after the first dialogue both actors at table 1 must move to a different table in order not to play the same dialogue twice. But then they will face each other again.

In the second test case in the first round we have pairs $(1,1),(2,2),(3,3)$ *playing together (assuming that we numbered both men and women 1, 2, 3 according to their starting table). In the second round pairs* $(3,2),(1,3),(2,1)$ *play together. Finally, in the third round pairs* $(2,3),(3,1),(1,2)$ *play together. Note that everyone played with all possible partners and visited all tables.*

Solution

Let us first show a solution when N is odd. Let us number the tables $0, \ldots, N-1$. Let $\pi(i) = (i+1) \bmod N$ and $\varphi(i) = (i+2) \bmod N$. We will prove that these two permutations are the solution for the problem when N is odd. It is easy to see, that every man and every woman will visit every table. Let us number men and women by the number of the table where they played first. Man with number m will visit tables $m, m+1, \ldots, N, 1, \ldots, m-1$. Woman with number w will visit tables $w, w+2 \bmod N, w+4 \bmod N, \ldots, w+2(N-1) \bmod N$. No two tables have the same number in this sequence, since $w+2j \equiv w+2k \pmod N$ is equivalent to $2k \equiv 2j \pmod N$. Since $0 \leq j, k \leq N-1$ and N is odd, this can only happen for $k = j$. Hence each woman will visit all tables.

After k rounds, man m will visit table $m+k$. A woman with number $(m+k)-2k \equiv m-k \pmod N$ will be playing with him. Thus a man m will play with women $m, m-1, m-2, \ldots, 1, N, \ldots, m+1$. This concludes the proof for N odd.

For N even, we will prove that no solution exists by contradiction. Assume that there are two permutations π and φ, that are the solution. Let us renumber tables as follows. Pick a man and label the tables with numbers $0, \ldots, N-1$ in the order he visits them. Since we changed the labeling, we will get new permutations π' and φ'. Let us number men and women by the number of the table where they played first, as we did in the odd case. Note that man m will play at tables $m, m+1, \ldots, N, 1, \ldots, m-1$. A woman w will play at $w, \varphi'(w), \varphi'^2(w), \ldots, \varphi'^{N-1}(w)$. After k rounds, she will be at table $\varphi'^k(w)$ and will a meet man with number $(\varphi'^k(w) - k) \bmod N$. We assume that a woman will meet all men, thus numbers $(\varphi'^k(w) - k) \bmod N$ will be all different. Hence

$$\sum_{k=0}^{N-1} (\varphi'^k(w) - k) \equiv (0 + 1 + \cdots + (N-1)) \equiv N(N-1)/2 \pmod N.$$

Since a woman visits all tables, numbers $\varphi'^k(w)$ are also all different and we have

$$\sum_{k=0}^{N-1} \varphi'^k(w) \equiv 0 + 1 + \cdots + (N-1) \pmod N.$$

Moreover, $\sum_{k=0}^{N-1} k \equiv 0 + 1 + \cdots + (N-1) \pmod N$. By subtracting these two equivalences we get

$$\sum_{k=0}^{N-1} (\varphi'^k(w) - k) \equiv 0 \pmod N.$$

We showed before, that this sum is also equal to $N(N-1)/2 \bmod N$. However, $0 \not\equiv N(N-1)/2 \pmod N$ when N is even. This is because for $N = 2M$, $2M(2M-1)/2 = 2M^2 - M$ and $2M^2$ is divisible by N while $-M$ is not. Hence $2M^2 - M$ is not divisible by N and we get a contradiction.

Klingon High Council Training

Authors

Problemsetter: Michal 'mišof' Forišek
Task preparation: Michal 'mišof' Forišek

Problem statement

Tired of being seen as the endless supply of brute force for the United Federation of Planets, the Klingon High Council has decided to start a training program aimed on spaceship battle tactics. The main part of the program is a game that is played by two of the trainees.

The game is played using a fleet of N spaceships. The space is divided into a cube grid of sectors. Each sector is uniquely identified by its three integer coordinates. Each sector may contain multiple spaceships. During the game the spaceships are traveling between sectors in order to reach a target planet that is located in the sector $(0, 0, 0)$. When all the ships reach this sector, the game is over and the player who finished the maneuver is decorated as the winner.

Each turn of the game looks as follows: The player whose turn it is must first **contact** between 1 and K ships, and then **order** each of the contacted ships to move. The turn ends when all ships finish executing their orders.

Different ships may receive different orders. Each ship must be ordered to move into a sector that is strictly closer to the target planet than its current sector. (When measuring the distance between sectors, we measure the Euclidean distance of their centers.)

Problem specification

Given are the number of ships N, the limit on moves per turn K and the initial sectors of all ships. Compute which of the two players will win the game if both players play optimally.

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 number of ships N and the limit K. The i-th of the next N lines contains three integers: the coordinates of the sector where the i-th ship starts.

In the easy input you may assume that the coordinates are small and that in each test case either $K = 1$ or $K = N$.

Output specification

For each test case output a single line with the string `Player 1` if the first player has a winning strategy and the string `Player 2` otherwise.

Example

input

```
2

3 2
2 1 0
0 0 0
47 42 47

2 1
1 0 0
0 1 0
```

output

```
Player 1
Player 2
```

In the first test case the first player can move both ships #1 and #3 straight to $(0, 0, 0)$*. If he does so, he wins.*

In the second test case the first player must move one of the ships to $(0, 0, 0)$*. The second player then moves the other one and wins.*

Solution

This was more or less a standard combinatorial game theory task. Still, there were some aspects (in particular the "K moves at once" part) that made this problem challenging.

A simplification of the game

Imagine that we are about to move a ship into the sector $(5, 0, 0)$. Nothing would change if we moved it to one of the sectors $(0, 5, 0)$, $(0, 3, 4)$, $(4, 0, 3)$, etc. This is because each of these sectors has exactly the same distance from the target. If sending the ship to $(5, 0, 0)$ was a valid order, then sending it to any other sector in the same distance is valid as well. And afterwards the set of valid orders is the same for any of these sectors.

So instead of saying that a ship is in the sector $(5, 0, 0)$, we only need the information that its distance from the target is 5.

Of course, some of the distances are irrational, for example the distance between the sector $(3, 0, 2)$ and the target is $\sqrt{3^2 + 0^2 + 2^2} = \sqrt{13}$. To get rid of the square roots, we will represent each sector by the square of its distance from the target.

This gives us the following simplified version of our game:

Let $Z = \{x^2 + y^2 + z^2 \mid x, y, z \in \mathbb{Z}\}$ be the set of all valid squares of distances. We have N tokens, each of them placed on an element of Z. Each token can only be moved onto a smaller element of Z.

And that's not all. We can order the elements of Z and replace them by all non-negative integers. In this way we get a game that's again a bit simpler: We have N tokens, each of them placed on an element of $\mathbb{N}_0$. Each token can only be moved onto a smaller number.

And this is clearly the standard game of NIM – except for the constraint K.

In this way we managed to separate the task into two parts: first we have to "translate" the sector coordinates into NIM heap sizes, and then we have to solve the NIM game.

A simple way to translate the coordinates

Let the initial sectors be (x_i, y_i, z_i) and let $M = \max_i x_i^2 + y_i^2 + z_i^2$ be the square distance to the furthest sector.

In order to translate the coordinates into heap sizes, we will simply generate all valid square distances up to M.

This can be done in the following way:

```
for (int x=0; x*x <= M; ++x)
  for (int y=0; x*x + y*y <= M; ++y)
    for (int z=0; x*x + y*y + z*z <= M; ++z)
      add_distance( x*x + y*y + z*z );
```

As none of the coordinates x, y, z can exceed $\sqrt{M}$, the number of possibilities examined will be $O(M^{3/2})$, which is surely fast enough for the easy data set.

For the remaining implementation details see the sample implementation.

Moving all ships at a same time

We will now solve the case $K = N$. This case is really simple: player 1 wins in the first move.

Multiple ships, moving one at a time

Now suppose we have N ships and have $K = 1$, i.e., we are only allowed to move one ship in each turn. This is the standard game of NIM, with a well-known solution: *The position is losing for the player to move if and only if the bitwise xor of all heap sizes is zero.*

We decide to omit the proof, as later we will prove a more general version of this statement.

A closer look at sums of three squares

If you took the time to implement the slower method that generates all possible square distances, you probably noticed that almost all non-negative integers appear among them. In order to speed up this part of the solution we need to know precisely which integers are missing.

This question is related to a famous result from number theory: Lagrange's Four Squares Theorem that states: any non-negative integer can be written as the sum of at most four squares.

In our case we are looking precisely for the "worst cases" of this theorem: the numbers for which three squares are not enough. Let's call these numbers *complicated*.

First of all, note that each number of the form $8t + 7$ is complicated.

Why is it so? Modulo 4, each even square is congruent to 0 and each odd square to 1. And as $8t+7 \equiv 3 \pmod 4$, we need at least three odd squares. Let $8t+7 = (2a+1)^2 + (2b+1)^2 + (2c+1)^2$. This simplifies to $2t + 1 = a(a + 1) + b(b + 1) + c(c + 1)$, and this equation has no integer solutions: the left side is odd, the right side is even.

Similarly, each number of the form $4^s(8t+7)$ is complicated. This is because if it were not complicated, then all squares that form it must be even. We can divide both sides by 4, and repeat this process until we eliminate s. Then we reach a contradiction anyway.

It can be proved that there are no other complicated numbers – for any other number three squares are enough. (This was first proved by Legendre in 1798, an omission in the proof was later corrected by Gauss.) We omit the proof.

A faster way to translate the coordinates

Legendre's theorem can be used to translate the coordinates into heap sizes efficiently: Suppose that we have a ship at coordinates (x, y, z). Then its square distance is $d = x^2+y^2+z^2$. What's the corresponding heap size? Clearly, it can be determined as $d - f(d)$, where $f(d)$ is the count of complicated numbers that do not exceed d. We will now show how to compute $f(d)$ efficiently.

Based on Legendre's theorem, $f(d)$ can be expressed as the number of integer pairs (s, t) such that $s, t \geq 0$ and $4^s(8t + 7) \leq d$.

To count these pairs, we will try all possible values of s. Note that there are just $O(\log d)$ such values. For a fixed s, we need to count all valid t.

Rewriting the inequality, we get $t \leq ((d/4^s) - 7)/8$. This means that there are $\lfloor((d/4^s) - 7)/8\rfloor + 1$ possible values of t for a fixed s.

Solving the general case

Suppose that we take all the heap sizes. Write them in binary and align them so that digits of the same order are in the same column.
We now claim the following:
The position is losing if and only if the number of 1s in each column is divisible by $K + 1$.

```
101010
 10001
  1111
     1
111000
100110
```

For example, let $N = 6$ and $K = 2$. Suppose that the heap sizes are 42, 17, 15, 1, 56, and 38. Then the aligned binary representations look as shown on the right:

In the second column from the left the number of 1s is not divisible by 3, so we claim that this is a winning position.

We will now prove the above statement. But before we do so, please convince yourself that it is correct both in the case $K = 1$ and in the case $K = N$.

Let's call the positions that satisfy our criterion *blue* and all other positions *red*.

In order to prove that blue positions are losing and red positions are winning, we need to prove three statements.

- All terminal positions are blue.

  This is trivially true, the only terminal position is when all heap sizes are zero.

- From each blue position all moves lead to red positions.

  Consider how the binary representations change. More precisely, look at the leftmost column in which something changes. In this column, several 1s change into 0s – this is because the heap sizes can only decrease. The number of 1s that changes in this way is between 1 and K, inclusive. And as their count was divisible by $K + 1$ before the change, it can not be divisible by $K + 1$ after the change.

- From each red position there is a move that leads into a blue position.

  Again, consider the binary representation. We will show how it can be used to construct such a move.

  We will process the columns from the left to the right. During the computation we will be labeling some of the rows as active. In the end, the active rows will correspond to the heaps that changed.

  In the beginning, no row is active. Whenever we process a column, we can make two types of changes. First, in a row that is not active we can change a 1 into a 0. This will make the row active for all future columns. Second, in a row that is already active we may make any change we want to.

  Note that the above rules ensure that we will *decrease* the changed numbers – in each row the most significant digit that changes goes from 1 to 0.

  We now need to show that we can turn any red position into some blue position, while making at most K rows active.

  To do so, each column will be processed in the following way: Let R be the number of rows that are already active. First, we look at the $N - R$ inactive rows and compute the number of 1s we see in the current column. Let X be this value modulo $K + 1$.

  If $X \geq K + 1 - R$, we can fix the current column by adjusting the active rows only: we will set $K + 1 - X$ of them to 1 and the remaining ones to 0.

This leaves the case $X \leq K - R$. In this case we pick any X inactive rows that contain 1s in the current column, and change these 1s into 0s. This makes the chosen rows active for future columns. Note that the number of active rows will not exceed K.

This concludes our proof.

Algorithm summary

Using Legendre's theorem we can quickly convert sector coordinates into corresponding heap sizes. We then convert the heap sizes into binary, and check whether $K + 1$ divides the number of occurrences of the digit 1 at each position. If yes, Player 2 wins, if no, Player 1 has a winning strategy.

Lovely stamps

Authors

Problemsetter: Peter 'ppershing' Perešíni
Task preparation: Peter 'ppershing' Perešíni, Vladimír 'U$Amec' Boža

Problem statement

Little Peter is collecting stamps. Recently, he went shopping with his mother Lucia and guess what: As they were going around the post office, he started to blackmail his mom, as only the little boys can. At the post office, they were selling N different types of one-dollar stamps and M different types of two-dollar stamps.

Peter got exactly K dollars from his mom, and he wants to spend all of them on stamps. Note that he can buy more stamps of the same type. You may assume that the post office has an infinite stock of each type of stamp.

Now, Peter is wondering in how many ways he can buy the stamps.

Problem specification

Given are the integers N, M, K, and a prime number P.

Your task is to compute the value Z mod P, where Z is the (possibly huge) number of ways in which Peter can spend all K dollars on stamps.

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 the integers N, M, K, and P.
You may assume that $3 \leq P \leq 1\,000\,000$.
For the small input set, you may assume that $0 \leq N, M \leq 1000$ and $1 \leq K \leq 1000$.
For the large input set, you may assume $0 \leq N, M \leq 300$ and $1 \leq K \leq 1\,000\,000\,000\,000 = 10^{12}$.

Output specification

For each test case output a single line with a single integer: The number of different ways to buy stamps, modulo P.

Example

input

```
3

0 10 2 47

2 2 4 47

5 5 10 47
```

output

```
10
14
6
```

In the first test case, we must buy one 2-dollar stamp and there are 10 types.

In the second test case, we have these options:
– buy two 2-dollar stamps: 3 ways to do so

– *buy a 2-dollar stamp and two 1-dollar stamps:* $2 \times 3 = 6$ *ways to do so*
– *buy four 1-dollar stamps: 5 ways to do so*
Therefore the answer is $(3 + 6 + 5) \bmod 47 = 14 \bmod 47 = 14$.

Solution

Simple solution

The first solution you should consider is solution based on dynamic programming. To be more precise – pretend we know how many different ways for buying t 1-dollar stamps are there and denote it f_t. Similarly, denote the number of different ways of buying 2-dollar stamps using t dollars as g_t. Clearly, the result is

$$result = \sum_{i=0,2,4,6,\ldots}^{k} f_{k-i} \cdot g_i$$

So if we can compute numbers f_i, g_i for arbitrary $i \geq 0$, then we can obtain final result in time $O(k)$. The question is, how to compute f_i and g_i. One can notice, that $g_{2i+1} = 0$ and $g_{2i} = f_i$, so it is necessary only to compute f_i for $i \leq k$.

The simpliest solution for computing f_i should use the same principle, i.e. dynamic programming. Let $f_{i,t}$ be a number of ways to buy i stamps when we have t different types of stamps available. Formula for computing $f_{i,t}$ can be derived from simple observation – when we want to buy i stamps of t types, let's buy $0 \leq j \leq i$ stamps of first $t-1$ types and add $i-j$ stamps of t-th type. This can be expressed as

$$f_{i,t} = \sum_{j=0}^{i} f_{j,k-1}$$

Dynamic programing solves this in $O(i \cdot k)$ time. Applying it to our problem, the total running time is $O((n+m)k)$. This is enough for the easy test case.

But it can be done in another way: Computing $f_{i,t}$ is equivalent to computing, how many different solutions have the system $c_1 + c_2 + c_3 + \cdots + c_t = i$ with restrictions that $0 \leq c_j$ and $c_j \in Z$ for $j \in 1, 2, \cdots, t$. This is well known combinatorial problem and it's solution is

$$f_{i,j} = \binom{i+j-1}{j-1}$$

For details, see [1]. Now

$$result = \sum_{i=0,2,4,6,\ldots}^{k} f_{k-2i,n} \cdot g_{2i,m} = \sum_{i=0}^{\lfloor \frac{k}{2} \rfloor} \binom{k-2i+n-1}{n-1} \binom{i+m-1}{m-1}$$

This gives us (with the knowledge that successive values of $\binom{i+j-1}{j-1}$ can be computed from previous values in constant time) solution which runs in $O(k)$ time.

Little bit faster solution

It can be proven that:

$$\binom{x}{y} = \binom{x+p^t}{y} \bmod p \tag{1}$$

where t is such that $p^t > y$. To prove this, just express all terms in binomial in form of $x = p^s q$ where p does not divide q and show, that result will not change. Then

$$\binom{x}{y} = \frac{(x-y+1)(x-y+2)\ldots(x-1)x}{y!}$$

and

$$\binom{x+p^t}{y} = \frac{(x+p^t-y+1)(x+p^t-y+2)\dots(x+p^t-1)(x+p^t)}{y!}$$

Now, some of the values in the numerator are divided by some power of p from denominator (Well, the result si integer and therefore exists some pairing between powers in terms in numerator and denominator such that power of p in term in numerator is at least power of p in corresponding term in denominator). And the power in terms in denominator is always strictly less than t because $p^t > y$. Therefore, $\frac{x-y+i}{p^j q} \bmod p$ is the same as $\frac{x+p^t-y+i}{p^j q} = \frac{x-y+i}{p^j q} + p^{t-j} \bmod p$, because we can assume that $t-j > 0$. This implies (1).

Now if we take our sum:

$$result = \sum_{i=0}^{\lfloor \frac{k}{2} \rfloor} \binom{n+k-2i-1}{n-1}\binom{m+i-1}{m-1} \bmod p$$

and t such that $p^t > m$ and $p^t > n$, then following holds:

$$\binom{n+k-2i-1}{n-1}\binom{m+i-1}{m-1} = \binom{n+k-2(i+p^t)-1}{n-1}\binom{m+(i+p^t)-1}{m-1} \bmod p$$

So we can split our sum into p^t groups for which the value of binomials are are same (mod$p$) and calculate the total sum as separatelly.

$$result = \sum_{i=0}^{p^t-1} \binom{n+k-2i-1}{n-1}\binom{m+i-1}{m-1} \left\lfloor \frac{\lfloor k/2 \rfloor - i}{p^t} + 1 \right\rfloor$$

If we take the smallest t such that $p^t > m$ and $p^t > n$, then our solution will have time complexity $O(p^t) = O((m+n)p)$ or $O((m+n)^2 + p)$. This was enough to solve the hard input. But let's not stop here.

A bit of math

Now it's time to go deeper in mathematics. Consider polynomials

$$P(x) = 1 + x + x^2 + x^3 + x^4 + x^5 + ...$$
$$Q(x) = 1 + x^2 + x^4 + x^6 + x^8 + x^{10} + ...$$
$$R(x) = 1 - x + x^2 - x^3 + x^4 - x^5 + ...$$

What we do now is in mathematics called generating functions. The idea behind is something like: Take polynomial $P(x)^2$. What reperesents coefficient before x^k in it? One can notice that it represents a number of ways how to put together different terms of x together. But this is exactly what we want – it's the same situation as if we are buying stamps. We want to know in how many ways we can put different types together. So we know that coefficient before x^k of polynomial $P(x)^2$ represents the number of ways how to buy stamps if we have k types of stamps.

Extending this idea, the result can be obtained as coefficient before x^k of polynomial $P(x)^n \cdot Q(x)^m$. And here starts second approach to this problem. We want to know how resulting polynomial looks like. We can start with simpler problem: computing $P(x)^n$ for any n. One way is simply try to guess the

formula after a first few powers:

$$P(x)^1 = 1 + x + x^2 + x^3 + x^4 + x^5 + \dots$$
$$P(x)^2 = 1 + 2x + 3x^3 + 4x^3 + 5x^4 + 6x^5 + \dots$$
$$P(x)^3 = 1 + 3x + 6x^2 + 10x^3 + 15x^4 + 21x^5 + \dots$$
$$P(x)^4 = 1 + 4x + 10x^2 + 20x^3 + 35x^4 + 56x^5 + \dots$$

It can be shown by induction, that

$$P(x)^n = \frac{1}{1 \cdot 2 \cdot 3 \cdots (n-1)} \Big[\big(1 \cdot 2 \cdot 3 \cdots (n-1)\big) + \big(2 \cdot 3 \cdots n\big)x + \big(3 \cdot 4 \cdots (n+1)\big)x^2 + \dots \Big]$$

More formally

$$P(x)^n = \sum_{i=0}^{\infty} w_{n,i} x^i$$

where

$$w_{n,i} = \frac{(i+1) \cdot (i+2) \cdots (i+n-1)}{(n-1)!} = \binom{i+n-1}{n-1} \tag{2}$$

Proof: Base of induction is $w_{1,i} = \binom{i}{0} = 1$ which holds for all i.
Now suppose that $w_{n,i} = \binom{i+n-1}{n-1}$. Then from the way how polynomials multiply we have

$$w_{n+1,i} = \sum_{j=0}^{i} w_{n,j} = \sum_{j=0}^{i} \binom{j+n-1}{n-1} = \sum_{j=0}^{i} \binom{j+n-1}{j}$$

Now consider binomial identity $\binom{x}{y} = \binom{x-1}{y} + \binom{x-1}{y-1}$ and apply it:

$$\begin{aligned}
\sum_{j=0}^{i} \binom{j+n-1}{j} &= \binom{n-1}{0} + \binom{n}{1} + \binom{n+1}{2} + \binom{n+2}{3} + \cdots + \binom{i+n-1}{i} \\
&= \binom{n-1}{0} - \binom{n}{0} + \binom{n}{0} + \binom{n}{1} + \binom{n+1}{2} + \binom{n+2}{3} + \cdots + \binom{i+n-1}{i} \\
&= 1 - 1 + \binom{n+1}{1} + \binom{n+1}{2} + \binom{n+2}{3} + \cdots + \binom{i+n-1}{i} \\
&= \binom{n+2}{2} + \binom{n+2}{3} + \cdots + \binom{i+n-1}{i} \\
&\vdots \\
&= \binom{i+n-1}{i-1} + \binom{i+n-1}{i} \\
&= \binom{i+n-1}{i}
\end{aligned}$$

And therefore the induction step is prooved.

Similarly, we can show that

$$R(x)^n = \sum_{i=0}^{\infty} (-1)^i w_{n,i} x^i$$

What about $Q(x)^n$? We should use either the formula for $P(x)^n$ substituting x^2 for x or do it with trick. In first case, we know $P(x)^n$ and $Q(x)^n$ so we can easily build up coefficient before x^k in resulting product. This approach gives again $O((n+m)k)$ time (because computing one coefficient take $O(n)$ or $O(m)$ time. But it can be easily reduced to $O(k)$ time, as we can compute coefficients of $P(x)^n$ and $Q(x)^m$ step by step (we take advantage of small changes in formula for $w_{n,i}$). So we have another approach to $O(k)$ solution.

Better solution

Of course, $O(k)$ is not fast enough. There exists even faster solution. The idea behind is a bit tricky. At first, we rewrite $P(x)$ and $Q(x)$ to another form. It is clear that P and Q are infinite geometric progressions. We can formally sum them and obtain

$$\begin{aligned} P(x) &= \frac{1}{1-x} \\ Q(x) &= \frac{1}{1-x^2} = \frac{1}{(1-x)(1+x)} \\ R(x) &= \frac{1}{1+x} \end{aligned}$$

Together, polynomial $P(x)^n Q(x)^m$ which we are interested in can be written as

$$P(x)^n Q(x)^m = P(x)^{n+m} R(x)^m = \frac{1}{(1-x)^{n+m}(1+x)^m}$$

Let's consider theorem about partial fractions in algebra. It says, that every such a polynomial fraction can be rewritten as sum of simpler polynomial fractions, namely that there exists coefficients $c_1, \ldots, c_{n+m}$ and $d_1, \ldots d_m$ such that

$$\begin{aligned} \frac{1}{(1-x)^{n+m}(1+x)^m} =& \frac{c_1}{1-x} + \frac{c_2}{(1-x)^2} + \cdots + \frac{c_{n+m}}{(1-x)^{n+m}} \\ &+ \frac{d_1}{1+x} + \frac{d_2}{(1+x)^2} + \cdots + \frac{d_m}{(1+x)^m} \end{aligned}$$

Proof of this theorem can be found in [2]. It is not clear however, how to find such a coefficients $c_1, \ldots, c_{n+m}, d_1, \ldots, d_m$. Substituting x by $1, 2, \ldots, n+m$ we obtain a system of $n+m$ equations with $n+m$ unknowns. This system can be solved by Gaussian elimination [3] in $O((n+m)^3)$ total time. Note that we do not need to use big numbers as all computations can be done modulo p.[1]

Now, when we have coefficients, we can reconstruct final polynomial as

$$ResultPoly(x) = c_1 P(x) + c_2 P(x)^2 + \cdots + c_{n+m} P(x)^{n+m} + d_1 R(x) + d_2 R(x)^2 + \cdots + d_m R(x)^m$$

and we are interested in computing coefficient before x^k in this polynomial.

$$result = \sum_{i=1}^{n+m} c_i \binom{k+i-1}{i-1} + (-1)^k \sum_{i=1}^{m} d_i \binom{k+i-1}{i-1} \tag{3}$$

[1] Except for division by multiple of p. This issue is addressed at the end of this paper.

Computing coefficients of x^k in $P(x)^t$ and $R(x)^t$ and thus computing the result can be done in $O((n+m)^2)$ time. Therefore the result can be obtained in $O((n+m)^3)$.

Even better

Surprisingly it can be done even better. Idea is to compute the solution of the system of equations faster. If you play a with this system of equations for a while, you should guess the solution (at least the general shape of it). If $n > 0$ or $m > 0$ then solution can be obtained from following formula when we set $a = m$ and $b = n + m$

$$\begin{aligned}\frac{1}{(1+x)^a(1-x)^b} = &\sum_{i=1}^{a} \frac{1}{(1+x)^i}\binom{a+b-i-1}{a-i}\frac{1}{2^{a+b-i}} + \\ &\sum_{i=1}^{b} \frac{1}{(1-x)^i}\binom{a+b-i-1}{b-i}\frac{1}{2^{a+b-i}}\end{aligned} \tag{4}$$

Proof of formula (4): We can multiply (4) by $(1+x)^a(1-x)^b$ and thus obtain

$$\begin{aligned}1 = &\sum_{i=1}^{a} (1-x)^b(1+x)^{a-i}\binom{a+b-i-1}{n-i}\frac{1}{2^{a+b-i}} + \\ &\sum_{i=1}^{b} (1-x)^{b-i}(1+x)^a\binom{a+b-i-1}{b-i}\frac{1}{2^{a+b-i}}\end{aligned}$$

Changing the sum index from i to $a-i$ (and $b-i$ in the second sum) we obtain

$$\begin{aligned}1 = &\sum_{t=0}^{a-1} (1-x)^b(1+x)^t\binom{t+b-1}{t}\frac{1}{2^{t+b}} + \\ &\sum_{t=0}^{b-1} (1-x)^t(1+x)^a\binom{t+a-1}{t}\frac{1}{2^{t+a}}\end{aligned} \tag{5}$$

Before we proof (5), it will be useful to state and prove following lemma:

Useful lemma For all $a \geq 0, b > 0$ holds

$$\sum_{t=0}^{b-1}\left(\frac{1-x}{2}\right)^t\left[\frac{1+x}{2}\binom{t+a}{t} - \binom{t+a-1}{t}\right] = -(1-x)^b\binom{a+b-1}{a}\frac{1}{2^b} \tag{6}$$

Proof of lemma: Proof by induction (over b). Induction base is $b = 1$.

$$\left(\frac{1-x}{2}\right)^0\left[\frac{1+x}{2}\binom{a}{0} - \binom{a-1}{0}\right] = \frac{1+x}{2} - 1 = -(1-x)\binom{a}{a}\frac{1}{2}$$

Note that $\binom{-1}{0} = 1$.

Induction step: Suppose that (6) holds for b. Then for $b+1$ we have

$$\begin{aligned}
&\sum_{t=0}^{b}\left(\frac{1-x}{2}\right)^t\left[\frac{1+x}{2}\binom{t+a}{t}-\binom{t+a-1}{t}\right]\\
=&\sum_{t=0}^{b-1}\left(\frac{1-x}{2}\right)^t\left[\frac{1+x}{2}\binom{t+a}{t}-\binom{t+a-1}{t}\right]\\
&+\left(\frac{1-x}{2}\right)^b\left[\frac{1+x}{2}\binom{a+b}{b}-\binom{a+b-1}{b}\right]\\
=&\{\text{by induction}\}\\
&-(1-x)^b\binom{a+b-1}{a}\frac{1}{2^b}\\
&+\left(\frac{1-x}{2}\right)^b\left[\frac{1+x}{2}\binom{a+b}{b}-\binom{a+b-1}{b}\right]\\
=&\left(\frac{1-x}{2}\right)^b\left[\frac{1+x}{2}\binom{a+b}{b}-\binom{a+b-1}{b}-\binom{a+b-1}{a}\right]\\
=&\left(\frac{1-x}{2}\right)^b\left[\frac{1+x}{2}\binom{a+b}{b}-\binom{a+b-1}{b}-\binom{a+b-1}{b-1}\right]\\
=&\left(\frac{1-x}{2}\right)^b\left[\frac{1+x}{2}\binom{a+b}{b}-\binom{a+b}{b}\right]\\
=&\left(\frac{1-x}{2}\right)^b\frac{x-1}{2}\binom{a+b}{b}\\
=&-(1-x)^{b+1}\binom{a+b}{a}\frac{1}{2^{b+1}}
\end{aligned}$$

Therefore lemma holds.

Proof of equation (5): Again by induction. We start with postponing induction base and go directly for induction step. Supposing that 5 holds for a, we want to proove that it holds also for $a+1$.

$$\begin{aligned}
&\sum_{t=0}^{a}(1-x)^b(1+x)^t\binom{t+b-1}{t}\frac{1}{2^{t+b}}+\sum_{t=0}^{b-1}(1-x)^t(1+x)^{a+1}\binom{t+a}{t}\frac{1}{2^{t+a+1}}\\
=&\sum_{t=0}^{a-1}(1-x)^b(1+x)^t\binom{t+b-1}{t}\frac{1}{2^{t+b}}+\sum_{t=0}^{b-1}(1-x)^t(1+x)^{a+1}\binom{t+a}{t}\frac{1}{2^{t+a+1}}\\
&+(1-x)^b(1+x)^a\binom{a+b-1}{a}\frac{1}{2^{a+b}}\\
=&\{\text{by induction}\}\\
&1-\sum_{t=0}^{b-1}(1-x)^t(1+x)^a\binom{t+a-1}{t}\frac{1}{2^{t+a}}\\
&+\sum_{t=0}^{b-1}(1-x)^t(1+x)^{a+1}\binom{t+a}{t}\frac{1}{2^{t+a+1}}\\
&+(1-x)^b(1+x)^a\binom{a+b-1}{n}\frac{1}{2^{a+b}}\\
=&1+\frac{(1+x)^a}{2^a}\left[\sum_{t=0}^{b-1}\left(\frac{1-x}{2}\right)^t\left(\frac{1+x}{2}\binom{t+a}{t}-\binom{t+a-1}{t}\right)\right]\\
&+\frac{(1+x)^a}{2^a}\left((1-x)^b\binom{a+b-1}{n}\frac{1}{2^b}\right)+\\
=&\{\text{by lemma}\}\\
&1+\frac{(1+x)^a}{2^a}\cdot 0\\
=&1
\end{aligned}$$

So if $b\geq 1$, induction step for a holds.
Similarly, we can prove, that if $a\geq 1$, induction step holds for b. Now back to induction base Induction base: for $a=0,b=1$ we obtain from (5)

$$\sum_{t=0}^{-1}(\cdots)+(1-x)^0(1+x)^0\binom{-1}{0}\frac{1}{2^0}=0+1 \tag{7}$$

Analogically, for $a=1,b=0$ we can verify, that (5) holds. Finally, if we put together these two induction bases and two induction steps, we obtain proof that (5) holds for $a\geq 0, b\geq 0, a+b>0$.

Completing final solution

Now we are almost done. We have

$$\begin{aligned}
\frac{1}{(1+x)^a(1-x)^b}=&\sum_{i=1}^{a}\frac{1}{(1+x)^i}\binom{a+b-i-1}{a-i}\frac{1}{2^{a+b-i}}+\\
&\sum_{i=1}^{b}\frac{1}{(1-x)^i}\binom{a+b-i-1}{b-i}\frac{1}{2^{a+b-i}}
\end{aligned}$$

where $a = n + m$ and $b = m$. Therefore, coefficients c_i, d_i which we are interested in are

$$c_i = \binom{n+2m-i-1}{n+m-i} \frac{1}{2^{n+2m-i}}, \quad i = 1, \ldots, n+m$$
$$d_i = \binom{n+2m-i-1}{m-1} \frac{1}{2^{n+2m-i}}, \quad i = 1, \ldots, m$$

Final result can be determined from (3) by substituting for c_i, d_i:

$$result = \sum_{i=1}^{n+m} \binom{i+k-1}{i-1} \binom{n+2m-i-1}{n+m-i} \frac{1}{2^{n+2m-i}} + \\ + (-1)^k \sum_{i=1}^{m} \binom{i+k-1}{i-1} \binom{n+2m-i-1}{m-i} \frac{1}{2^{n+2m-i}}$$

This two sums can be evaluated in linear time, because value of x-th term can be computed from $(x-1)$-th term in constant time and first term can be computed in linear time. First sum have starting point

$$s_1 = \binom{n+2m-2}{n+m-1} \frac{1}{2^{n+2m-1}}$$

and step-changes are

$$\frac{s_{i+1}}{s_i} = \frac{\binom{i+k}{i}\binom{n+2m-i-2}{n+m-i-1}\frac{1}{2^{n+2m-i-1}}}{\binom{i+k-1}{i-1}\binom{n+2m-i-1}{n+m-i}\frac{1}{2^{n+2m-i}}} = \frac{2(n+m-i)(i+k)}{i(n+2m-i-1)}$$

Analogically, terms in second sum can be evaluated as

$$t_1 = \binom{n+2m-2}{m-1} \frac{1}{2^{n+2m-1}}$$

and step change

$$\frac{t_{i+1}}{t_i} = \frac{2(m-i)(i+k)}{i(n+2m-i-1)}$$

So, now we have solution that runs in $O(n+m+p)$ time (Time $O(p)$ is needed to precompute table of inverses.) or in time $O((n+m)\log p)$ if we do not precompute the whole table of inverses and just use extended Euclid algorithm to compute inverse of specific numbers.

Hidden catch

We have very pretty equation to compute. But there is still one hidden catch. We want to compute it over field Z_p (over modulo p). And as you can notice, if p is small, we can very easily divide by multiple of p, which is indeed dividing by zero. This undesired behaviour can be however eliminated using more complex information. We will be storing not only values modulo prime p but also how many times it is multiplied by p. Formally, we will store number as $x \cdot p^y$ where $0 < x < p$ and $0 \leq y$. Now, both multiplying and dividing also with multiples of p can be done ellegantly - new x is $x_1 \cdot /x_2 (\mod p)$ and $y = y_1 \pm y_2$. Value of this number is 0 if $y > 0$ and x if $y = 0$.

Precalculation of table of inverses in linear time

To obtain described algorithm in linear time, it is necessary to divide in constant time. This can be a problem, because inverse element in Z_p can't be generaly computed in constant time. One possible trick is to precompute all inverses. Precomputing should be fast enough also for large primes p. Ideally in linear time. Surprisingly, algorithm doing this isn't hard. Let's start with only one known inverse – inverse of 1 is 1. Now, let's pick up number u. Consider making stack of values $u, u^2, u^3, u^4, \ldots$ until we reach u^i for which we know the inverse. At this point, $inv[u^{i-1}] = inv[u^i] * u$. And from inverse of u^{i-1} we can compute inverse of u^{i-2} and so on until the stack is empty. If we repeat this procedure for all values of $u \neq 0$ we have completed table of inverses. To prove linearity of this algorithm, we should count number of reads and writes to table of inverses. For each write operation, we must have done at most two readings from table. And because number of written elements is p, the whole algorithm is linear to p.

Literatúra

[1] Wikipedia, *Multiset* http://en.wikipedia.org/wiki/Multiset (27.May 2007)

[2] Feldman, *Partial fraction decomposition* http://www.math.ubc.ca/~feldman/m101/partdecomp.pdf (27.May 2007)

[3] Wikipedia, *Gaussian elimination* http://en.wikipedia.org/wiki/Gaussian_elimination (27.May 2007)

Mass psychoanalysis

Authors

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

Problem statement

> **Note.** This is a special problem. No points can be gained for this problem.
> The only thing you can gain is negative penalty time.

In Asimov's famous series of books on the Foundation one of the main characters, Hari Seldon, is credited with inventing psychohistory: a scientific discipline that can use statistics to predict the behavior of large groups of people. We are now asking you to lay the foundations of psychohistory for real. Your goal is to predict the behavior of your fellow contestants as good as you can.

We will be playing a simple game. In the game you have a single submit. You can use this submit at any point during the game. If you do so, we will record the exact **time** between the start of the game and the moment of your submission. During the game these submits are **not shown** in the standings.

Once the game is over, we will then compute the **average** of all the submission times. Your goal in this game: your time must be as close as possible to **two thirds of that average**.

Example

Three teams take part in the game. The first team submits 500 seconds after the game starts, the second team submits at 600 seconds and the third one at 1690 seconds.

The average submission time is $(500 + 600 + 1690)/3 = 2790/3 = 930$ seconds. Two thirds of that average is $930 \cdot (2/3) = 620$. The second team's submission time is closest to 620.

Input specification

You may submit anything you wish to (e.g., an empty file). The content of your submission does not matter, only the time is important.

The **first game** starts at the beginning of the contest and ends after two hours. During these two hours you may submit the **easy data set** M1 in order to play this game.

After two hours the first round will be evaluated and the results **will be made public**.

The **second game** starts at 2h30 and ends at 4h30 into the contest. (I.e., it starts in the middle of the contest and ends 30 minutes before its end.) During these two hours you may submit the **hard data set** M2 in order to play this game. (Note that the time you choose is the time since the beginning of the **game**, not the beginning of the **contest**.)

After the second game ends, it will be evaluated and the results will be made public.

Evaluation

In the first game the following scoring is used. For each team only the best option applies.

- Any team within 5 seconds of the goal gains -120 penalty minutes.

- Any team within 30 seconds of the goal gains -80 penalty minutes.
- Any team within a minute of the goal gains -60 penalty minutes.
- Any team within two minutes of the goal gains -40 penalty minutes.
- Any team within five minutes of the goal gains -20 penalty minutes.

In the second game the scoring will be double (i.e., -240 for an exact guess, etc.).

Solution

This is a placeholder. We will publish some data and graphs later.