The 2019 ICPC Caribbean National Contests

Real Contest Editorial

by

Marcelo Fornet (NEAR, Cuba)
Carlos Joa (INTEC, República Dominicana)
Carlos Toribio (Google, República Dominicana)
Elio Govea (UPR, Cuba)
José Carlos Gutiérrez (Universität Bremen, Cuba)
Rafael Fernández (Cuba)
Ariel Cruz (UH, Cuba)
Norge Vizcay (Cuba)
Daniel Otero (Universität Bremen, Cuba)
Leandro Castillo (Dropbox, USA)
Roberto Abreu (República Dominicana)
Aurora Gil (Cuba)
Reynaldo Gil (Datys, Cuba)
Dovier Ripoll (UCI, Cuba)

October 17th, 2019

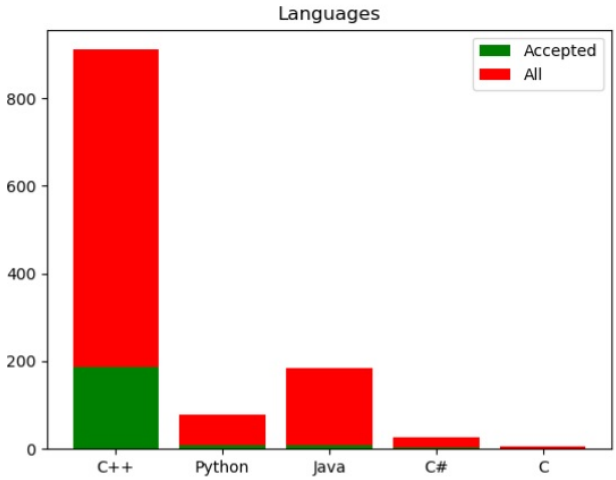
About this document

This document is an analysis of The 2019 ICPC Caribbean National Contests (Real). It describes a sketch of the solution for every problem. They are not intended to be a complete solution but just an outline of how the judge's solution looks like. If you spot an error, have a comment about the competition or this editorial, or just want to provide some feedback, please send an email to icpc.carib.cjc@gmail.com.

Judges Team

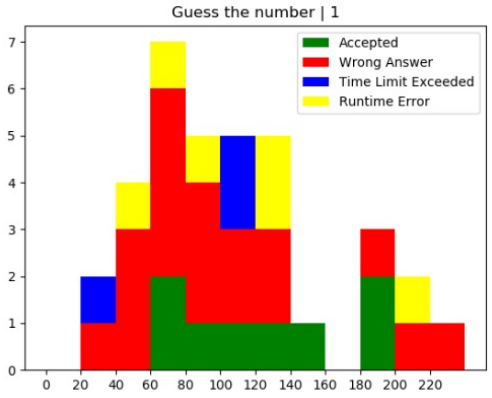
Summary

The following graph shows the number of correct submissions (green) and total submissions (red) per language.



Problem A: Guess the number

Author: Carlos Toribio
Solved: 8 teams
First solved: 61 minutes by DECB



This problem can be solved with dp. At any moment of time lets assume the set C contains all the numbers we have not guessed. Also let X be the number we want to guess and $dp_X(C)$ the expected value of the number of steps remaining to guess X on state C. Since X is fixed we will be using $dp(C) = dp_X(C)$ for simplicity.

The base case is when $X \notin C$ which implies $dp(C) = 0$ since we already guess the target number, so there no extra steps are needed. Then we have to solve only when $X \in C$ in which we will try any number in $[1, N]$.

Let $F(i, C)$ be the lowest number greater or equal to i that exists in C or -1 it none exist. The probability of guessing each i is $1/N$. For each i so that $F(i, C) = -1$ we will have to try again hence the expected will be $dp(C)$, but when $F(i, C)! = -1$ we will have the expected $dp(C \setminus F(i, C))$. Having this functions we can write the recurrence as:

$$E(i, C) = \begin{cases} dp(C \setminus F(i, C)) & F(i, C) \neq -1 \\ dp(C) & \text{otherwise} \end{cases}$$

$$dp(C) = \sum_{i=1}^{N} \frac{1 + E(i, C)}{N}$$

The only problem with this recurrence is that $dp(C)$ has infinite loop when $F(i, C) = -1$ So if we have the set of numbers that satisfy $F(i, C)! = -1$ as S:

$$S = \{i | F(i, C)! = -1\}$$

Then we can write the $dp(C)$ as:

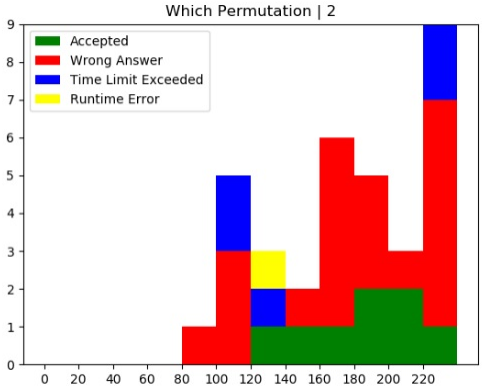
$$dp(C) = dp(C)\frac{N - |S|}{N} + \sum_{i \in S} \frac{1 + dp(C - F(i, C))}{N}$$

If we solve for $dp(C)$ we get:

$$
\begin{aligned}
dp(C) - dp(C)\frac{N-|S|}{N} &= \sum_{i \in S} \frac{1 + dp(C - F(i,C))}{N} \\
dp(C)(1 - \frac{N-|S|}{N}) &= \sum_{i \in S} \frac{1 + dp(C - F(i,C))}{N} \\
dp(C)\frac{|S|}{N} &= \sum_{i \in S} \frac{1 + dp(C - F(i,C))}{N} \\
dp(C) &= \frac{N}{|S|} \sum_{i \in S} \frac{1 + dp(C - F(i,C))}{N} \\
dp(C) &= \sum_{i \in S} \frac{1 + dp(C - F(i,C))}{|S|}
\end{aligned}
$$

Problem B: Which Permutation

Author: Carlos Toribio
Solved: 8 teams
First solved: 125 minutes by UH++



First lets see all the permutation with 3 distinct elements:

1 2 3	2 1 3	3 1 2
1 3 2	2 3 1	3 2 1

We can notice that after sorted lexicographically we can see each element of the array occupies the first position in 2 permutations. This varies when we have repeated elements

1 2 3 3	2 1 3 3	3 1 2 3
1 3 2 3	2 3 1 3	3 1 3 2
1 3 3 2	2 3 3 1	3 2 1 3
		3 2 3 1
		3 3 1 2
		3 3 2 1

As we can see now element 3 appears in the front in 6 permutation while 1 and 2 appear only in 2. Lets call this number of times as function P(x) which means, number of permutations starting with x. If we have $F_x[i]$ as a frequency array of all elements of the array after removing ONE x. Using combinatorics we know the number of permutations is:

$$P(x) = \frac{(\sum F_x[i])!}{\prod F_x[i]}$$

Having this, if we know that a given permutation starts with y then we know that all permutations starting with x $(0 \leq x < y)$. This would be:

$$\sum_{x=0}^{y-1} P(x)$$

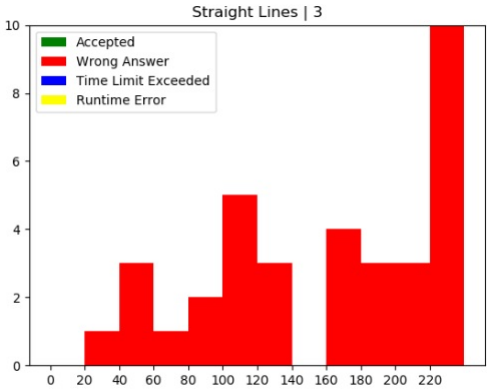
After this we remove y from the set of elements and we continue recursively (for the second position, then third and so on) solving the problem with the remaining elements. This means now we have to update all $F_x[y]$ for all x hence recomputing P(x) for every x. This means P(x) will suffer the following transformation:

$$P(x) = \frac{\left(\sum F_x[i]\right)!}{\prod F_x[i]} * \frac{F_x[y]}{\sum F_x[i]}$$

This will make the P(x) as if y never existed and we can continue doing sum of P(x) with the remaining elements. This range updates and range queries can be done using Segment Trees overall complexity of $O(NlogM)$ where M is the alphabet size and N is the number of elements.

Problem C: Straight lines

Author: Elio Govea
Solved: 0 teams
First solved: -



Each straight line is defined by two integers x and y. Two of this straight lines are parallels if $\frac{x_1}{y_1} = \frac{x_2}{y_2}$. Lets group the lines, two lines belong in the same group if are parallel. Given the problem's restrictions you can only draw one line from each group, so the answer to this question is the number of groups. Each group of lines have only one that $\gcd(x, y) = 1$, so we can count this lines only.
We will count the number of lines that $\gcd(x, y) = 1$ and $y < x$ (for the lines that $x > y$ the procedure is the same, and there are N lines that $x = y$). For every x there are $\phi(x)$ (this is the Euler's totient function) values of y that $\gcd(x, y) = 1$ so the total is $\sum_{x=2}^{N} \phi(x)$. Defined $\Phi(n)$ as $\sum_{i=1}^{n} \phi(n)$. Since $N <= 10^9$ a sub-linear solutions is expected, you can use that $\Phi(n) = \frac{n\cdot(n+1)}{2} - \sum_{g=2}^{n} \Phi(\left\lfloor \frac{n}{g} \right\rfloor)$, in the sum there are only $\mathcal{O}(\sqrt{n})$ different values {1, 2, ..., $\lfloor\sqrt{n}\rfloor$, $\left\lfloor \frac{n}{\lfloor\sqrt{n}\rfloor} \right\rfloor$, ... $\lfloor\frac{n}{2}\rfloor$}, all g from $\left\lfloor \frac{n}{\lfloor\frac{n}{g}\rfloor+1} \right\rfloor + 1$ to $\left\lfloor \frac{n}{\lfloor\frac{n}{g}\rfloor} \right\rfloor$ result in the same $\left\lfloor \frac{n}{g} \right\rfloor$ so if you know $\Phi(\left\lfloor \frac{n}{g} \right\rfloor)$ you can calculate $\Phi(n)$ in $\mathcal{O}(\sqrt{n})$, using memoization and the recursive formula leads to an overall $\mathcal{O}(n^{\frac{3}{4}})$.
You can count the number of ways using the Rule of product, since you can draw only one line from each group, w is the product of the amount of lines on each group. For the groups that $y < x$ for an x value there are $\phi(x)$ groups and each contains $\lfloor\frac{n}{x}\rfloor$ lines, so $w = n \cdot (\prod_{x=2}^{N} \lfloor\frac{N}{x}\rfloor^{\phi(x)})^2$. This expression have only $\mathcal{O}(\sqrt{n})$ different values, let v to be a value $\lfloor\frac{n}{x}\rfloor$ the product is $\lfloor\frac{n}{x}\rfloor^{\sum_{i=l}^{r}\phi(i)}$ (l and r are the bounds of the interval of all x values that lead to the same $\lfloor\frac{n}{l}\rfloor$), using Φ the formula goes like $\lfloor\frac{n}{x}\rfloor^{\Phi(r)-\Phi(l-1)}$, since $l-1$ and r are are the result of $\lfloor\frac{n}{k}\rfloor$ for some integer $k \in [1, n]$ we have $\Phi(l-1)$ and $\Phi(r)$ calculated from the previous part, leading to a $\mathcal{O}(\sqrt{N})$ complexity. Summarizing:

- $c = 1 + 2 \cdot \sum_{x=2}^{N} \phi(x) = 1 + 2 \cdot (\Phi(N) - 1)$
- $w = n \cdot (\prod_{x=2}^{N} \lfloor\frac{N}{x}\rfloor^{\phi(x)})^2$.

- Use the formula $\Phi(n) = \frac{n \cdot (n+1)}{2} - \sum_{g=2}^{n} \Phi(\left\lfloor \frac{n}{g} \right\rfloor)$ and memoization
- Expected $\mathcal{O}(N^{\frac{3}{4}})$ solution.

Problem D: Bridge

Author: Jose Carlos Gutierrez
Solved: 3 teams
First solved: 209 minutes by Pac-men ...

Our goal in this task is to create an alternating sequence from a given set of colored cubes, i.e. one in which no number (representing the cube color) does not occur twice in a row. In addition, the starting and ending elements of the sequence are determined.

Let i_1 denote the number of available cubes of 1, i_2 denote the number of cubes of 2, and so on. The color to start the sequence should be marked with p, while the `final` color should be marked with q. In addition, k is the number of different cube colours, and n is the number of all cubes, i.e. $n = i_1 + i_2 + \ldots + i_k$. To simplify the description of the solution we assume from now on that the numbers of cubes of particular colours are sorted not decreasing, i.e. i_1 is the number of cubes most rarely found in the given set, and i_k is the number of the most popular cubes in the set.

Let's first solve a slightly simpler task: from a given set of cubes we want to create any alternating sequence - without a fixed initial and final element. It turns out that in this case it is easy to check if it is possible at all and, if it is, arrange such a sequence. The condition necessary and sufficient for the existence of an alternating sequence is as follows:

$$i_k \leq \frac{n+1}{2}.$$

The necessity results from the Pigeonhole principle. Sufficiency can be demonstrated by providing an algorithm for the construction of the sequence:

- Find j, and divide i_j in $i'_j + i''_j = i_j$ so that

  $$i_1 + i_2 + \ldots + i'_j + 1 = i''_j + i_{j+1} + \ldots + i_k$$

  if n is odd, or

  $$i_1 + i_2 + \ldots + i'_j = i''_j + i_{j+1} + \ldots + i_k$$

  if n is even.

- Let A be the sequence of i_1 cubes of color 1, i_2 cubes of color 2, ..., i'_j cubes of color j.

- Let B be the sequence of i''_j cubes of color j, i_{j+1} cubes of color $j+1$, ..., i_k cubes of color k.
- The final sequence will be formed alternating elements from A and B.

It turns out that the condition sufficient to construct an alternating sequence for any pair of extreme colours (referred to as p and q) is very similar to the condition necessary for the existence of any alternating sequence:

$$i_k \leq \frac{n-1}{2}$$

If $p = q$, there is a simple boundary condition to check: $i_p \geq 2$.

The construction of the target sequence begins with removing two elements from the set of cubes: one in p and one in q. Even if $p \neq k$ and $q \neq k$, the condition sufficient to build any sequence is still fulfilled. We can build an alternating sequence, and at the ends of this sequence, add the two previously removed p and q cubes.

Now the task is not to repeat the colour on the left or right edge of the sequence. Let us mark the colours of the cubes at the ends of the alternating sequence as p' and q'. We have the following cases:

- $\{p, q\} \cap \{p', q'\} = \emptyset$. In this case nothing has to be done.
- $p \neq q$ and $p' \neq q'$. In this case it may be that p or q equals to p' or q', so we may have to insert p (or q) on a particular side so that there is no repetition of the colour.
- $p' = q'$. The algorithm of alternating sequence construction given above will lead to such a situation only if the colour p' is the most numerous in the reduced sequence and this colour has the maximum allowed number of times. But it follows from this that neither p nor q are the same colour as p'. So this case is equivalent to the first one.
- We can now assume that $p' \neq q$, $p = q$ and p is the same to p' or q'. Without loss of generality let us assume that $p = p'$. So we have three elements of colour p and one colour q'. One cube of p and one cube of q' should be placed on one side. Now we have to push one p into a place in the middle of the string where two elements of a colour other than p are adjacent to each other. Such a place always exists - if it did not exist, then the colour element p would have to appear on every second position in the reduced sequence, and thus adding two additional colour elements p (i.e. elements p and q), we would disturb the inequality of the necessary condition.

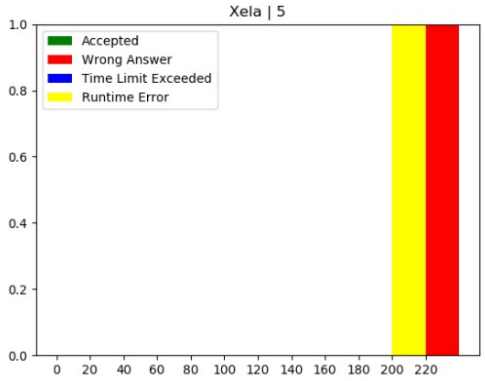
If the condition sufficient to create an alternating sequence for any p and q is not fulfilled, there are still a few more cases where the correct sequence can be arranged:

- For n odd and $i_k = \frac{n+1}{2}$ we have to place k coloured cubes on every second position, also the initial and final, so that the string is alternate. The solution therefore exists for $p = q = k$.

- For n even and $i_k = \frac{n}{2}$ at least one of the cubes p, q must be k. Moreover, if both are k coloured, the correct sequence can be arranged only if and when $i_{k-1} = \frac{n}{2}$, that is, when there are at least three different colors of the cubes. If however $k = 2$ and $i_1 = i_2 = \frac{n}{2}$, the only alternating sequence alternates colors 1 and 2, so the beginning and the end must have such colors.

Problem E: Xela

Author: Marcelo Fornet
Solved: 0 teams
First solved: -



This problem doesn't allow a naive interpreter that just simulate the given instructions. With the given limits (at most 100 lines) there can exist 99 nested loops with repeat parameter R, so any naive interpreter should do R^{99} operations, which will not fit in time. The problem offers a few clues about the expected solution in its constraints.

- There are only 26 different variables (all lowercase English characters).
- It is not allowed to multiply by another variable.
- It is not allowed to loop using a variable as a parameter.

All operations changes variable linearly in terms of the previous values of each variable, so they can be expressed nice in terms of matrix. For example, let's see how to rewrite some of the operations like matrix.

Assume we have only two variables a, b for simplicity. Use a vector with three terms such that the first term has the value of a, the second term has the value of b and the third term has always the value 1 (this last term will be useful to add constant values). Initially we start with the vector $< 0, 0, 1 >$ since variables starts at value 0.

If we have operation `a += b` this can be rewritten as $a' = a + b$ (a' is the value of a after executing the operation). This can expressed as:

$$\begin{bmatrix} a' \\ b' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ 1 \end{bmatrix}$$

This matrix changes value a as expected and left unchanged variable b and the constant value 1. Subtraction by another variable is done in the same way, but instead of using 1 you should use -1.

For operation `b -= 314`, it is rewritten as $b' = b - 314$

$$\begin{bmatrix} a' \\ b' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -314 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ 1 \end{bmatrix}$$

Notice how we make use of the constant value to add/subtract by a constant. Assignment of the form `a = 123` is already a linear equation and it just overwrite current value at a.

$$\begin{bmatrix} a' \\ b' \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 123 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ 1 \end{bmatrix}$$

Finally for multiplying one variable by a constant `b *= 11` this is rewritten as $b' = b \cdot 11$ which in matrix notation would be:

$$\begin{bmatrix} a' \\ b' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 11 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ 1 \end{bmatrix}$$

Notice that multiplying two variables isn't a linear operation (so this is the where the second hint is relevant).

Now each operation can be written as a matrix, so if we want to apply three operations O_1, O_2, O_3 sequentially, this is equivalent to build the respective matrix $M_{O_1}, M_{O_2}, M_{O_3}$ and applying this matrix to the initial vector: $v' = (M_{O_3} \cdot (M_{O_2} \cdot (M_{O_1} \cdot v)))$. Notice that matrix aren't commutative so they should be multiplied in the same order as the operations are executed. But matrix are associative, so we can remove parenthesis from the previous equation, and moreover this will allow us to group parenthesis as we wish: $v' = (M_{O_3} \cdot M_{O_2} \cdot M_{O_1}) \cdot v = M \cdot v$. For a block of three operations we calculate the matrix $M = M_{O_3} \cdot M_{O_2} \cdot M_{O_1}$ that apply this operations sequentially to the initial variables. With loops we just want to repeat the same operations k times, and it is equivalent to multiply the values of the variables v before entering the loop by M k times: $v' = M \cdot M \cdot ... \cdot M \cdot v = M^k \cdot v$. We can do binary exponentiation to find the value of M^k in $O(n^3 \cdot \log k)$ where n is the size of the matrix. This comes from the fact that we need $\log k$ multiplications, and the multiplication of two matrix takes $O(n^3)$ operations.

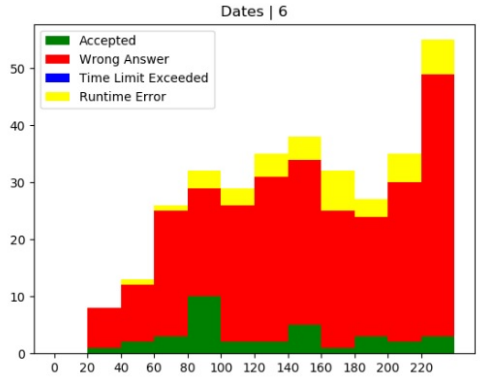
The real matrix have size 27 (the number of lowercase English letters plus a constant value). The expected complexity for this problem was $O(m \cdot n^3 \cdot \log k)$ (m the number of lines, n the size of the alphabet (26 in this case) and k the max value of the repeat parameter).

All operation are taken modulo 998244353, so numbers are guaranteed not to become extremely large. Remember that internal operations in the matrix should use this module (which is fine). Beware of the case of negative values, instead of using $-v$ use $998244353 - v$ instead.

Parsing the input was an extra ~~intimidating~~ challenge.

Problem F: Dates

Author: Marcelo Fornet
Solved: 31 teams
First solved: 34 minutes by Pac-men ...



This problem could be solved by mere brute force. But this is the first problem which have many caveats such February having 28 or 29 days or months having 31 or 30 days.

Since the constraints are rather small (only 300 days are given in the input) and there are at most 366 days in a year, we can just simulate if the given sequence match the sequence generated starting each day. The answer will be the last month of all valid sequence (don't forget to sort them).

We don't know beforehand which year this sequence took place, so you should test with February having 28 days and 29 days. The easiest way to do that is trying all sequences that start at every possible day for two consecutive years, such that the second year is leap. This will cover the case where the leap year happened at the beginning of the sequence, at the end of the sequence or didn't happen.

For every month there is at most one valid starting date, so the number of sequence that should be checked against the given sequence is at most $2 \cdot 12$, one starting at every month during two years. To solve this problem it was very convenient `datetime` built in functions in languages such as Python and Java. No accepted solution to this problem used it during the competition.

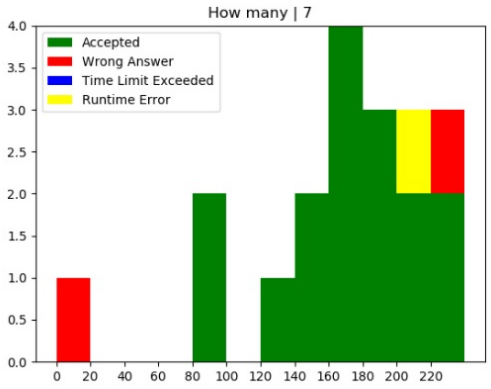
Solution using `datetime` library:

```
import datetime

n = int(input())
days = list(map(int, input().split()))

answer = set()

for month in range(1, 13):
    for year in range(2003, 2005):
        try:
            start = datetime.datetime(year, month, days[0])
        except ValueError:
```

```
            continue

        valid = True

        for delta in range(n):
            cur = start + datetime.timedelta(days=delta)

            if cur.day != days[delta]:
                valid = False
                break

        if valid:
            last = start + datetime.timedelta(days=n - 1)
            answer.add(last.month)

MONTHS = ['','January','February','March',
          'April','May','June','July',
          'August','September','October',
          'November','December']

print(len(answer))

for x in sorted(answer):
    print(MONTHS[x])
```

Problem G: How Many

Author: Ariel Cruz
Solved: 16 teams
First solved: 92 minutes by Team3C-1



To solve this problem, the amount of invalid matches where all the elements of set B are matched with exactly one element of set A will be calculated, and will be subtracted from the total of possible matches that meet this property.

The total of possible matches following this rule is m^n because each of the m elements of B can be matched with each of the n elements of A. So the problem is reduced to finding invalid matches where each element of B is matched with exactly one element of A.

An invalid match is anyone who leaves elements of A without being paired with any element of B.

This can happen by leaving at least one element unpaired. To calculate this, each of the m elements is fixed and the rest of the $m-1$ can be matched in $(m-1)^n$ ways. So this can be done in $m(m-1)^n$ ways.

In general, following the same idea, the amount of invalid matches leaving at least i unmatched elements is $\binom{m}{i}(m-i)^n$.

In this way using the inclusion-exclusion principle the amount of invalid matches is:

$$\sum_{i=1}^{m-1}(-1)^i\binom{m}{i}(m-i)^n$$

So the general formula subtracting that from the total of matching is:

$$m^n-\sum_{i=1}^{m-1}(-1)^i\binom{m}{i}(m-i)^n$$

Or what is the same:

$$\sum_{i=0}^{m-1}(-1)^i\binom{m}{i}(m-i)^n$$

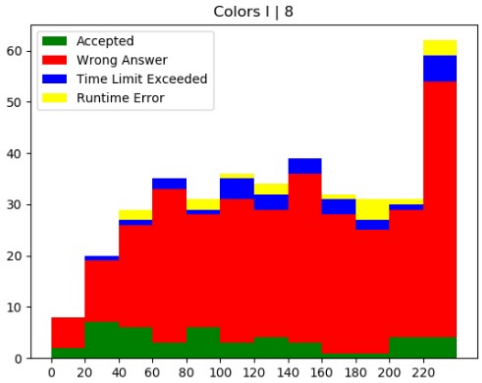
To calculate this value efficiently you need to perform a fast exponentiation in a time complexity of $O(\log(n))$. It is important to keep in mind that since you have to calculate

the result through a module, you cannot divide, but calculate the modular inverse. Since the module is a prime number, we have left that the modular inverse of the number x is equal to the rest of the division of $x^{(mod-2)}$ with mod. It is important to do that equally with fast exponentiation.

In general, the time complexity of this algorithm is $O(m \log(n))$.

Problem H: Color I

Author: Jose Carlos Gutierrez
Solved: 42 teams
First solved: 9 minutes by **Limitless**



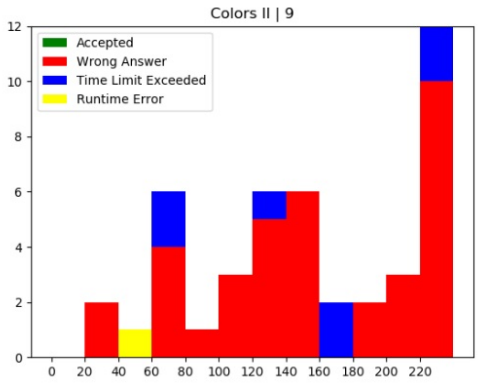
This problem was easy compared to the second version of the problem. First let's assume all points lie in a horizontal line. In this case the two farthest points would be the one with min X and the one with max X. Let's call these two points P_{min} and P_{max}. If these two points have different colors then their distance: $dist(P_{min}, P_{max})$ is the answer. If they have the same color then exactly one of them is included in the farthest pair. So we test every other point that have a different color than these two and compute farthest distance like this:

$$answer = \max\Big(\max\big(dist(P_{min}, x)\big), \max\big(dist(P_{max}, x)\big)\Big)$$

To extend the problem beyond a horizontal line we could get P_{min} and P_{max} using the comparator (sort by X if equal then sort by Y) and then run the above algorithm, for an overall complexity of $O(N \log N)$.

Problem I: Color II

Author: Jose Carlos Gutierrez
Solved: 0 teams
First solved: -



We consider $n \leq 250000$ points on the plane. Each of these points is assigned a certain color, but not that all points are of the same color. We are interested in determining the largest possible distance (Euclidean) between any pair of different-colored points.

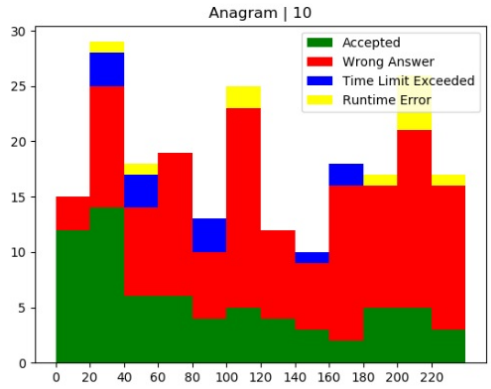
It's probably worth starting by considering the version in which we only have two possible colors. Let's call them blue and red. It is quite easy to notice that it is enough to consider only the blue points lying on the convex hull of all blue points and only the red points lying on the convex hull of all red points. So we're really working with two convex polygons. Find the largest distance between convex polygons is a well know problem in computational geometry, and can be solved in linear time, further details can be found in this paper.

Nice, but we can't limit ourselves to two colors. However, this is not a big problem. Let's say that there are more colors. We can deal with this issue in several ways, one can be apply divide and conquer. But the easy way is the following one.

Generate sets of points $A_0, A_1, \ldots, A_{\lfloor \log_2 n \rfloor}$, $B_0, B_1, \ldots, B_{\lfloor \log_2 n \rfloor}$. The set A_i will be formed for all points whose color have the i-th bit with 0. The set B_i will be formed for all points whose color have the i-th bit with 1. For example, if the color of a point is $11 = 1011_2$, the point is included in the sets B_0, B_1, A_2, B_3. Once we have those sets, we can find the farthest points between a point of A_i and a point of B_i for each possible i. The farthest pair among all those will be the answer. Is easy to see that we are considering all pairs of different colors using this approach.

Problem J: Anagram

Author: Marcelo Fornet
Solved: 65 teams
First solved: 8 minutes by **UH++**



This problem was designed to be the easiest one. In order to count how many pairs of segments there are we could consider first the length of the segment. So we fix the length to X and then try all possible pairs of segments of size x. This could be done by iterating all the starting points i and j so that the pair would be strings and $S[i, i+x-1]$ and $S[j, j+x-1]$. For each pair we form using the previous method we can test if they are anagrams. This can be done in multiple ways:

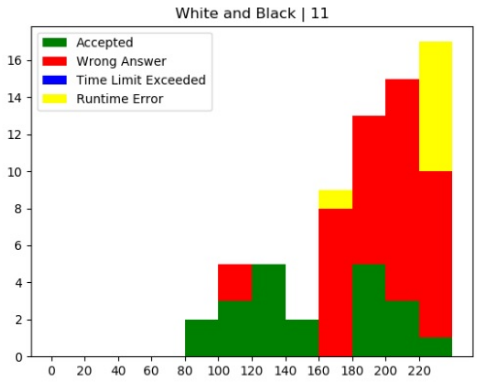
- Sort both and test if they are equal ($O(N \log N)$)
- Using c++ builtin function `is_permutation` ($O(N^2)$)
- Using a frequency array: ($O(N)$)

Using the worst we would end up with N^5 which was enough to get AC due to the low constant.

Faster Solution: You could compute the hash of the frequency array of all the segments, this could be done in then check for collisions for an overall complexity of using a hash map. Note that computing the hash of each frequency array can be done in $O(1)$ if we know the hash of the sub-array without the last element.

Problem K: White and Black

Author: Marcelo Fornet
Solved: 21 teams
First solved: 87 minutes by Pac-men ...



This problem is equivalent to determining if the next graph is bipartite and finding a coloration of it. Each set is a node in the network. Two sets must have two different colors if the intersection between the two sets is $k-1$ in size, so we add an edge between sets whose intersection is $k-1$ in size.

Complexity: Building the graph can be done using brute-force in $O(n^2k^2)$. To find if it is bipartite the cost is $O(n+m)$ where m is the amount of edges which is $O(n^2)$.

Explanation

Here we should assign a binary color to each set. It is easier to identify what should not happen. The thing is if there are two sets A and B such that they share $k-1$ elements then they **cannot** be of the same color. Because if they are then the person will choose those $k-1$ shared elements and you will not know if the set he has is A or B because they have the same color.

Let's call this pair enemies. So enemies cannot have the same color. If we get all pair of enemies we will have a graph and any two adjacent nodes should **not** be of the same color. This is literally two coloring of a graph. If the graph has two coloring then we can assign black and white colors so that every set does not have enemies of the same color and hence can be distinguished by knowing $k-1$ elements.

The graph can be built in $O(n^2k^2)$ and the two coloring can be computed in $O(n^2)$ which is the number of possible pairs.