The 2019 ICPC Caribbean Local Contests

Real Contest Editorial

Editorial developers

Marcelo Fornet (NEAR, Cuba)

Carlos Joa (INTEC, República Dominicana)

Dovier Ripoll (UCI, Cuba)

Reynaldo Gil (Datys, Cuba)

Carlos Toribio (Google, República Dominicana)

Elio Govea (UPR, Cuba)

Rubén Alcolea (UCI, Cuba)

Ernesto Peña (UO, Cuba)

September 22nd, 2019

About this document

This document is an analysis of The 2019 ICPC Caribbean Local 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. For some of the problems, we are releasing some extra challenges. 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.

Judge Team

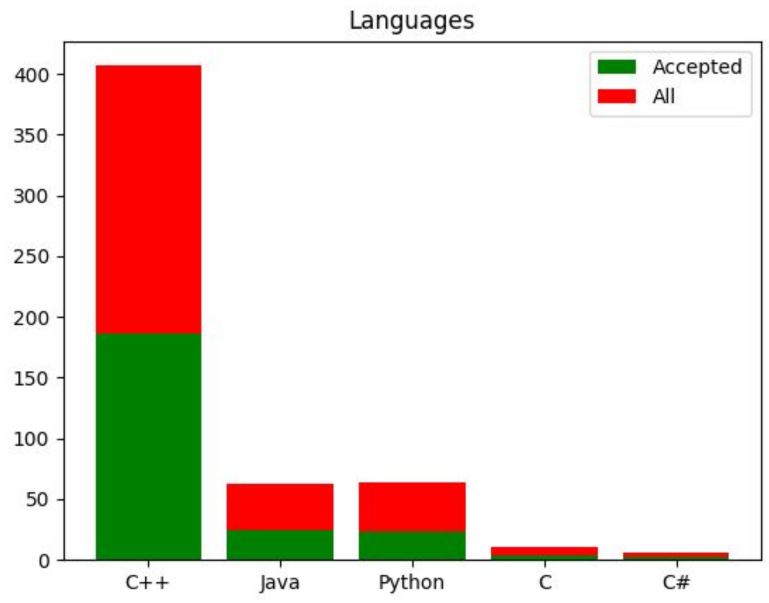
Summary

This year, for technical reasons, we use the brand new platform MOG instead of the well known legendary COJ. As judges, we were very pleased with the development and results of the contest regarding interaction with the platform. There were 40 teams participating officially in the contest and 49 teams competing as guests. If you were a participant/observer and had some issues with the platform, let us know so we can address them for future contests.

The contest began with an incredible start by team **UH++** solving the 3 easiest problems in 13 minutes (all first to solve). During the first hour, teams were working on the easiest problems, up until a few minutes after the first hour when team **UH_wake_me_up!** solved problem **H**. This year, for the first time, the set contained a problem with two versions: the easy version was noticeably easier compared with the second version. This required some strategic decisions from part of the teams, and we see how **UH_wake_me_up!** and **DECB** didn't solve both versions independently but went directly for the big pot. We think that strategies during the competition plays a big role in the final result, and in this case the strategy for **UH_wake_me_up!** turned out to be a winning one.

We commented the full contest in this post, so you can relive the excitement of the competition up until the frozen time. Watch in this video what happened during the frozen time. **Spoiler alert**: **Limitless** solved 3 problems in the last hour and went from 8th place to 2nd place. What an amazing climb! **UH_wake_me_up!** solved their 8th problem with 22 seconds to go at the end of the contest, very impressive.

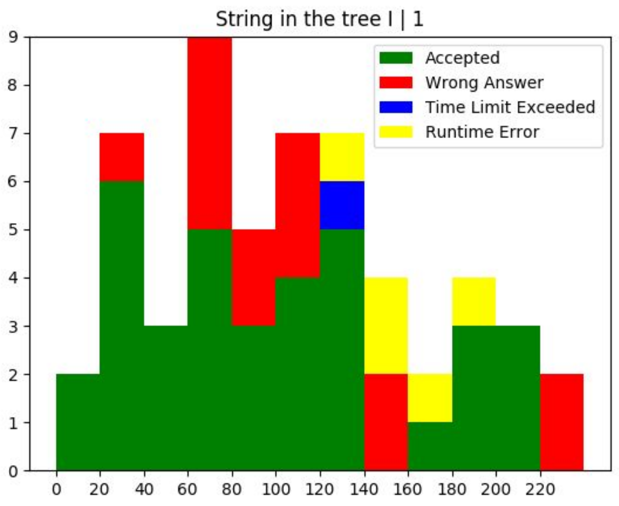
You can read statements of the problems in english and spanish using the following links: English, Spanish. During the competition, we see how C++ was by far the most popular language among contestants. This was also the case for judges, but we did our best to solve problems with java and python. Similar to Regional Contest and World Finals we guaranteed solutions in C++/Java, but unfortunately this was not the case for Python and other languages.



For every problem in this document, we provide length of the shortest submission from judges and from participants (in case they exist). Keep in mind that solutions were not written with the intention to be short (they can be shortened trivially in most cases), but it is reported to show the magnitude of the size of the code that solves each problem. The graph shown with the problems have the number of submissions during the competition time. They are grouped in intervals of 20 minutes.

The expected order of problems difficulties from judge point of view was **JG**|**AE**|**DHF**|**IKBC.** The real order (from number of accepted submissions per problem) was **GJA**|**EHD**|**FB**|**KIC**.

Problem A: String in the tree I



Author: Carlos J. Toribio

Solved/Tried by 31/36 teams

First solved after 13 minutes by ***UH++***

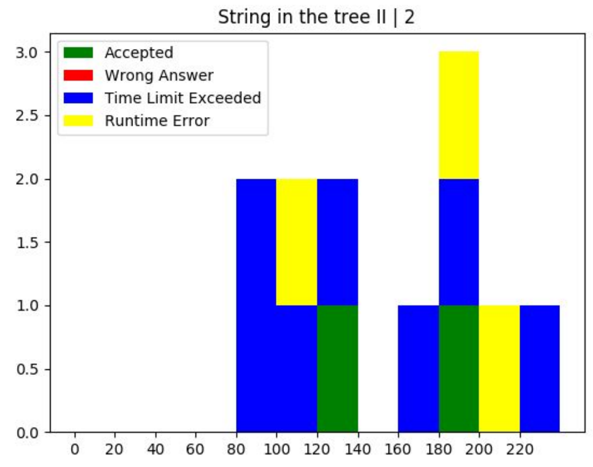
Shortest team solution: 633 bytes

Shortest judge solution: 495 bytes

SOLUTION: The constraints for the easy version allows to brute force the answer. We can check for every vertex, how many paths starts from it that contains the required pattern. This can be done easily using DFS algorithm. This solution runs in $O(N \cdot L)$ time and needs $O(N + L)$ space.

Problem B: String in the tree II

Author: Carlos J. Toribio
Solved/Tried by 2/11 teams
First solved after 123 minutes
by ***UH_wake_me_up!***
Shortest team solution: 3913 bytes
Shortest judge solution: 6183 bytes



This problem was intended to be one of the hardest of the competition. It requires advanced knowledge of graph (centroid decomposition) and string manipulation. It also requires heavy and careful implementation.

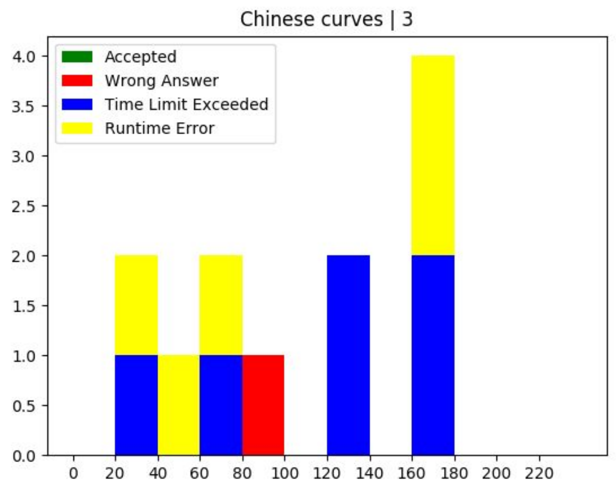
SOLUTION: First, let's fix a node ***u*** and count how many paths pass through this node that are equal to the pattern. For this, we enumerate all paths starting from this node (using DFS) and store in a frequency array if they are a valid prefix or a valid suffix (maybe both) of the pattern. While we are aggregating this information we should count how many (prefix, suffix) pairs exist such that prefix + ***u*** + suffix match the pattern. Take care of not counting paths that go through the same vertex below ***u***.

After we count paths passing through ***u***, we can remove this vertex and solve the problem in the remaining subtrees. To fit this solution in time, the special vertex can't be chosen arbitrarily, otherwise the solution would be quadratic in the number of nodes. Instead we should choose the centroid of the tree. This is a node such that, if it were the root of the tree, every subtree hanging from it has size at most half of the entire tree size. You can prove such node always exists and can be found in O(n).

To match prefixes and suffixes you can use hash or other fancy suffix data structures such as suffix automata. Overall complexity is $O(n \cdot log(n))$.

Problem C: Chinese curves

Author: Reynaldo Gil, Carlos J. Toribio
Solved/Tried by 0/4 teams
First solved: -
Shortest team solution: -
Shortest judge solution: 2148 bytes



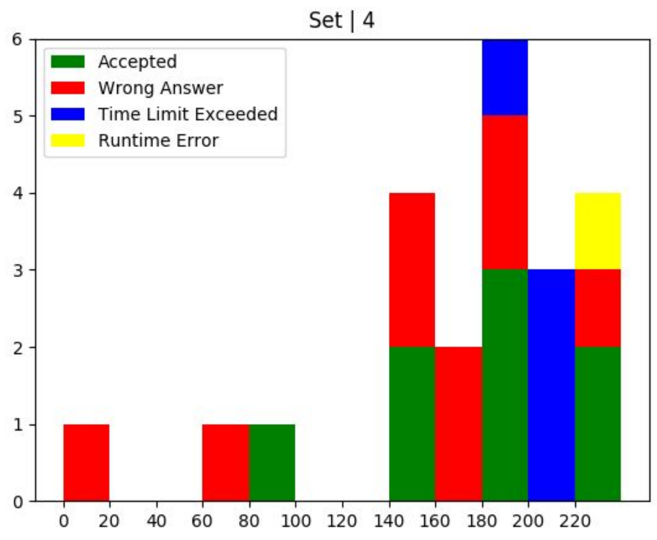
This problem was intended to be the hardest of the competition. All submissions during the contest used brute force, but the constraints of this problem didn't allow such submissions to fit in time.

SOLUTION: We can use a brute force algorithm for x coordinates with absolute value less than 40. For the remaining coordinates, the first factor in the curves' equations, $arctan(e^x + a)$, is almost constant, so it can be assumed constant.
Therefore, it is necessary to work with functions of the form $\sqrt{b \cdot x^2 + c}$ that fulfills (in the range x > 0) that each pair of functions have a single intersection point. For this part, queries can be answered using data structures similar to those used with straight lines, for example, lichao-tree.

The time complexity expected in the solution is O(n log(n)) and the spatial one is O(n).

Problem D: Set

Author: Elio Govea
Solved/Tried by 8/12 teams
First solved after 89 minutes
by ***◔Pac-men •••***
Shortest team solution: 673 bytes
Shortest judge solution: 908 bytes



SOLUTION: If L < N, there is no solution. Otherwise let $S_{min} = \sum_{i=1}^{n} i$ and $S_{max} = \sum_{i=L-N+1}^{L} i$. If $S < S_{min}$ or $S_{max} < S$, there is no solution.

For any other case, there is at least one solution. If $S = S_{min}$, then R={1, 2, ..., N}, and for any other $S \leq S_{max,}$ it is possible to build a solution from the solution for S - 1 because there will always exist at least an element x such that $x + 1 \leq L$ and $x + 1 \notin R$ (changing x by $x + 1$ in R).

It is possible to build the solution as it was described before, starting with R = {1, 2, ..., N}, and always taking the greatest of all x (let's call this element f(R)) as it will repeat that the number x that is taken and increased in one, in the next step it will be selected again unless it no longer meets the conditions. It will occur at most L - N times. Let's call a big step the fact of taking *f(x)* and change it for *f(x)* + *L* - *N*, and *q* the number of big steps necessary to find the solution, then *q* can be computed with the formula $q = \lfloor \frac{S-Smin}{L-N} \rfloor$ (Notice that after q big steps, R has the form {1, ..., N - q} ∪ {L - q + 1, ..., L} and *f(R)* = *N* - *q*). After that, it is needed at most one more step in which *f(R)* is replaced by *f(R)* + *k*, where *k* is the difference to obtain the required sum (*k* < *L* - *N*).

Overall complexity of described solution is $O(1)$. Solutions that use binary search will also run fast enough. In general, sublinear solutions are required to solve the problem.

Problem E: Rotate circles

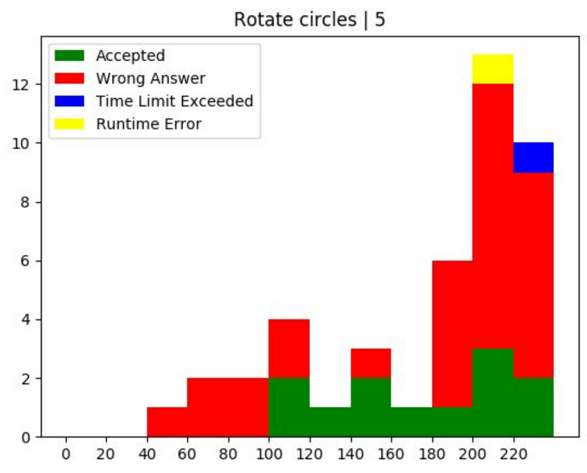
Author: Marcelo Fornet

Solved/Tried by 12/21 teams

First solved after 107 minutes by ***Tommy***

Shortest team solution: 1246 bytes

Shortest judge solution: 1374 bytes

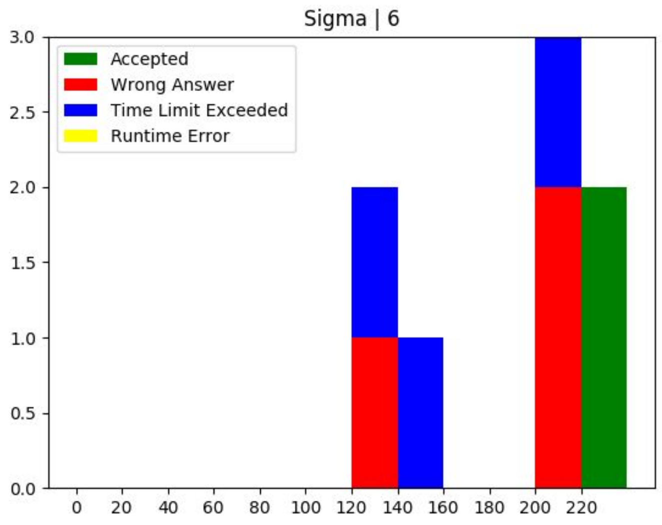


SOLUTION: Since colors from all regions are different for every circle, after fixing one circle, there is at most one valid rotation for every other circle. The solution is to fix one circle, and try to find if other circles can be rotated in such a way that satisfy the condition of the problem. The answer is always an integer between **0** and **4**. Solutions using backtracking will work in time, since the branching factor is only four in the first step, in further steps it will be at most 1.

Overall complexity is $O(N \cdot M)$.

Problem F: Sigma

Author: Marcelo Fornet
Solved/Tried by 2/3 teams
First solved after 221 minutes
*by **Limitless***
Shortest team solution: 3834 bytes
Shortest judge solution: 3235 bytes



SOLUTION: There is always a solution to exactly one of the two children. Let's build the following directed weighted graph: Split and replace every node **u** into two nodes **u_b** and **u_e**. For every node **u** in the original graph, create a directed edge from **u_b** to **u_e** with capacity 1. For every edge in the original graph between nodes **u** and **v,** create a directed edge in the new graph from **u_e** to **v_b** with capacity ∞.

In this new graph, find the maximum flow from node **F_e** to node **M_b** setting weight of each edge as the capacity. Assume that maximum flow equals G. If $G \le K$ there is a solution for Fito's puzzle. Otherwise ($K + 1 \le G$), there is a solution for Maria's puzzle.

Let’s build the solution for each puzzle. In both cases, we will be using the residual network associated to the graph after finding the maximum flow.

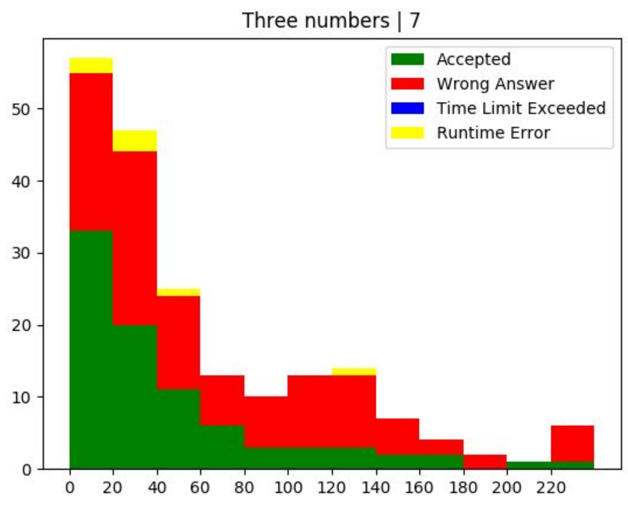
Fito's puzzle: To solve Fito’s puzzle, we need to find the minimum cut of this graph. Note that edges belonging to the cut will have capacity 1. Otherwise the cut would have value ∞, which we know is not optimal. Each edge with capacity 1 is associated with a node in the original graph, and those nodes are the ones that need to be removed.

Maria's puzzle: If maximum flow is equal to G, we can extract G paths in this network following augmenting paths. Note that two paths won't pass through the same node since capacity of edges associated with nodes on the original graph is 1.

To find the residual network of the maximum flow, we can use Dinic algorithm. In practice, instead of using ∞ capacity, using a large number such as N (or even 2) will do the work.

In graphs such that augmenting paths are bounded with 1 Dinic algorithm has asymptotic behavior $O(n^{\frac{3}{2}})$. Overall complexity is dominated by finding max flow in the network. Our solutions with Ford Fulkerson don't pass in time, since there exists some pathological cases in which the algorithm behavior is $O(n^2)$ where n is the number of nodes in the original graph.

Problem G: Three numbers



Author: Marcelo Fornet

Solved/Tried by 84/90 teams.

*First solved after 2 minutes by **UH++***

Shortest team solution: 211 bytes

Shortest judge solution: 99 bytes

This was intended to be one of the easiest problems of the competition. As expected there were several solutions without loops, but it remains a tricky problem if implemented without care.

SOLUTION: Iterate through every triplet and check if it is valid or not. Report any valid triplet. On the other hand, you can prove that among the triplets: (1, 2, 3), (2, 3, 4), (3, 4, 5), (4, 5, 6) there is at least one valid triplet, so this problem can be solved without loops by checking at most 4 triplets.

BONUS: What is the shortest solution you can write without using loops?

Problem H: Queries with recurrences

Author: Rubén Alcolea

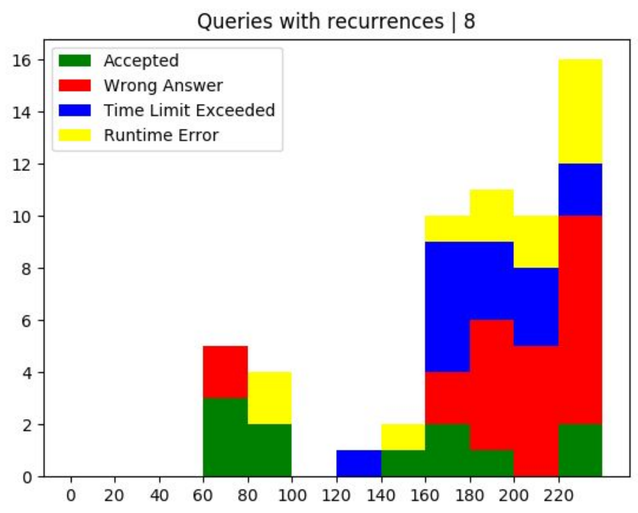
Solved/Tried by 11/23 teams.

First solved after 64 minutes

*by **UH_wake_me_up!***

Shortest team solution: 2553 bytes

Shortest judge solution: 2749 bytes



SOLUTION: This is a classical range query problem that can be solved with segment tree and lazy propagation. Here, the main challenge is efficiently compute the value F_n before each update operation.

The iterative solution to compute F_n does not pass the time limit. Instead, we can compute F_n using recurrence relation and matrix exponentiation in O(logN) without affecting the expected complexity (O(logN)) of the segment tree. F_n can be expressed in matrix notation in the following way:

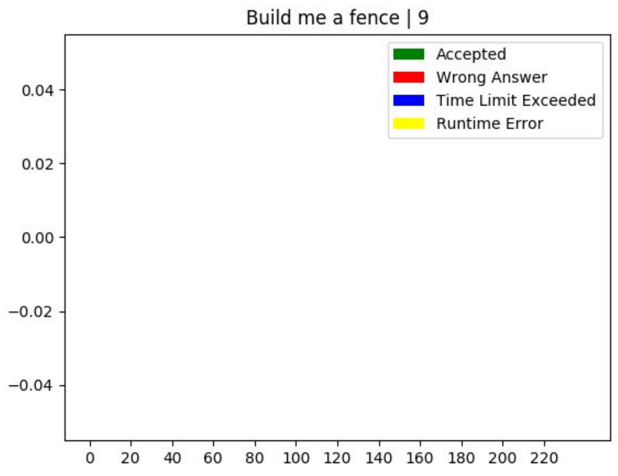
$$\begin{bmatrix} f_n \\ f_{n-1} \\ k \end{bmatrix} = \begin{bmatrix} a & b & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}^{N-1} \times \begin{bmatrix} f_{n-1} \\ f_{n-2} \\ k \end{bmatrix}$$

$$\begin{bmatrix} f_n \\ f_{n-1} \\ k \end{bmatrix} = \begin{bmatrix} a & b & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}^{N-1} \times \begin{bmatrix} k \\ 0 \\ k \end{bmatrix}$$

The final solution uses a segment tree with lazy propagation and also includes necessary methods to compute F_n before each update operation. The final complexity is O(Q logN).

Problem I: Build me a fence

Author: Carlos J. Toribio
Solved/Tried by 0/1 teams.
First solved: ∞
Shortest team solution: ∞
Shortest judge solution: 3631 bytes



The original version of this problem asked to find the polygon with largest area. The conditions were relaxed to find any valid non-degenerate polygon. The goal of this change was to make this problem more accessible to participants, but it seems we failed to do so. The submission graph is impressively empty (the first submission for this problem was 1 second before the competition ended, and the verdict was Compilation Error). Why are contestants so scared about geometry? It doesn't bite :-)

This problem was a step further of problem C (Can you draw it?) from the warmup contest.

Solution 1:
We can prove that there is always a polygon that fulfills these conditions and whose points touch a single circle and this is the polygon with the largest area. This demonstration is not very complex and it can be found here.

After knowing this, we can do a binary search over the radius of the circle, then we put each side in that circle in such a way that each side is a chord of the circle and therefore it occupies an arc. The sum total of all arcs must be 360 degrees. It is possible to determine the angle covered by that chord with the radius and the length of the chord (side). If the total angle is greater than 360, then the radius is too small, otherwise the radius is too big.

There is a special case: if using the smallest possible radius ($max(a_i)/2$) we cannot build the polygon (the angle of the arcs adds up to less than 360), then we must assume that there is a side covering more than 180 degrees and it is the biggest side. If F(r, l) represents the angle covered by that side, then we can compute 360 - F(r, l) (only for the biggest side) and it will be the angle covered by the biggest side. So, we can apply the previous binary search with that additional condition.

Solution 2:
If we assume the biggest side as a_0, then there is a number j such that:

$$\left|\sum_{i=1}^{j} a_i - \sum_{i=j+1}^{N-1} a_i\right| \leq a_0 \tag{1}$$

Therefore, it is possible to form a triangle such that the sides are:

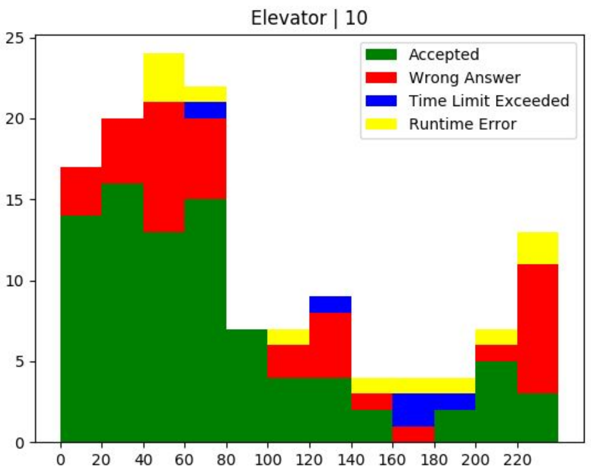
$$a_0$$

$$\sum_{i=1}^{j} a_i \tag{2}$$

$$\sum_{i=j+1}^{N-1} a_i \tag{3}$$

The only problem is when the two sides in expression 1 are strictly equal. In that case, it is possible to take two sides of (2) or (3) and to do an elbow, then replace those two sides with the length of that elbow and it will be less than a_0 instead of equal.

Problem J: Elevator



Author: Reynaldo Gil
Solved/Tried by 82/87 teams.
First solved after 6 minutes by ***UH++***
Shortest team solution: 64 bytes
Shortest judge solution: 103 bytes

SOLUTION: This problem can be solved using a greedy algorithm. In each delivery, we move the elevator to the highest floor where there are still missing packages to be delivered and we carry up as many packages as possible. This strategy is always optimal.

The number of deliveries can also be computed as (m / k) + (m % k != 0), where (m % k) represents the remainder of the integer division.

Problem K: Polygon

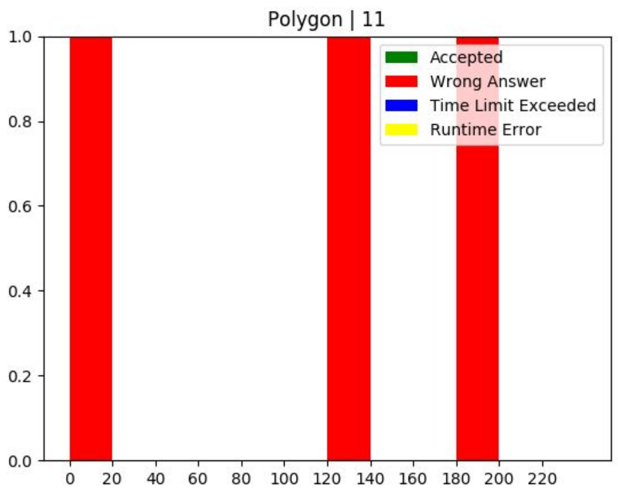
Author: Ernesto Peña

Solved/Tried by 0/3 teams.

First solved: ∞

Shortest team solution: ∞

Shortest judge solution: 1620 bytes



For this problem, we propose solutions for a version with number of vertices up to 20 and then the version with number of vertices up to 300. Both versions require clever insights and use of Dynamic Programming to speed up the solution. There is a part of the problem which requires some geometry checks, but submissions during the competitions missed that.

SOLUTION: The solution for this problem is composed of two parts. The first one is determining which diagonals we can draw in a valid triangulation, and the second part is calculating an optimal triangulation.

For the first part we must handle several cases:

1. A diagonal $\overline{ab}$ passes through vertex c of the polygon. In this case we should handle diagonal $\overline{ab}$ as two separate diagonals, $\overline{ac}$ and $\overline{ab}$.
2. A diagonal intersects any segment of the polygon. This diagonal is obviously impossible to draw without cutting the polygon.
3. At this point, if the current diagonal has passed cases 1 and 2, it means that is strictly inside or outside the polygon. So, we should check if it's inside. For this, we could just check if some point of the segment is an interior point or not.

This part can be done in O(n^3).

Now we will try to solve the second part of the problem. To do this we should find an optimal triangulation.

For n $\leq$ 20:
What we are going to do is to try all possible triangulation and get the optimal minimum one. For this we are going to use bitmasks and dynamic programming.

To solve the problem for a particular polygon, we are going to check every valid diagonal and for each one, to split the polygon in two polygons that we are going to solve recursively. The current polygon is composed by the active bits in the mask, and two vertices are adjacent in our polygon if between them all bits are zeroes. Of course, this mask should be cyclic, since we are handling a polygon, and all masks may not be reachable, except if the polygon is convex. When we reach a mask (polygon) consisting only of three bits, we must return the cost of such triangle.

To construct the two parts of the divided polygon for a fixed diagonal, we could just travel around the border of the polygon from one endpoint of the diagonal until we reach the other endpoint. This part has linear complexity, and fixing all diagonals is O(n^2), so, taking into account that we are going to do this for each possible polygon (all 2^n masks), the overall complexity of the solution is O($n^3 + 2^n \cdot n^3$), considering we are memoizing the best solution for each mask to avoid recalculating any state.

Unfortunately, this is a bit slow for getting Accepted. But we can improve this solution with the following observation. Suppose we are going to triangulate a polygon where vertices ***v*** and ***u*** are adjacent. This means that ***v*** and ***u*** must share a triangle and there must be a diagonal that goes through vertex ***v*** or vertex ***u*** if the polygon is not a triangle itself. Now we can just find a pair of adjacent vertices and try all possible diagonals for these two vertices. Now the number of diagonals we are going to check for each mask is linear and so, we remove an *O*(*n*) factor of the solution, getting a complexity of $O(n^3 + 2^n \bullet n^2)$.

Another way to improve the solution is to handle the diagonals from vertices ***v*** and ***u*** in order (clockwise or counterclockwise) of the polygon vertices, and keep track of the two parts of the divided polygon as we change from one diagonal to another. Thus, we avoid to travel around the border of the polygon and remove another O(*n*) factor, achieving a total complexity of $O(n^3 + 2^n \bullet n)$.

For n $\leq$ 300:
For this version of the problem we should improve even further the solution. To do so, we are going to use dynamic programming, but with a different approach. Our DP state is going to be a pair (*a* , *b*), meaning that we are going to triangulate the polygon composed by all points in between points *a* and *b* and delimited by diagonal $\overline{ab}$. Thus the initial state could be pair (0, *n-1*), meaning that we would like to triangulate polygon composed by all points and delimited by diagonal $\overline{0,\ n-1}$, that is a side in this case. So in this version, we should consider the sides of the polygon as possible diagonals.

Since *a* and *b* are consecutive points in the polygon, they must share a triangle in the triangulation that uses diagonal $\overline{ab}$, and since every polygon is possible to triangulate (proof to this fact can be found in [1]), there is always a vertex ***c*** such that we can draw diagonals $\overline{ac}$ and $\overline{cb}$. By doing this we have the triangle Δacb fixed and we can split our problem in solving pairs (*a*, *c*) and (*c*, *b*), which are sub-problems of the original problem and we can solve them recursively.

The amount of states is $O(n^2)$, since we should consider every pair of points (*a*, *b*), and we need O(*n*) to consider every possible point *c*. So, complexity for this part is $O(n^3)$ and the total complexity for our solution is also $O(n^3)$, which is good enough for this

[1] Mark de Berg · Otfried Cheong. Marc van Kreveld · Mark Overmars. Computational Geometry. Algorithms and Applications. Third Edition

version.

BONUS: If the polygon is convex we can disregard coordinates of the vertex, can you see why?