ICPC Europe Regionals

icpc global sponsor
programming tools

icpc
international collegiate
programming contest

icpc.foundation

icpc diamond
multi-regional sponsor

The 2022 ICPC Southwestern Europe Regional Contest

Official Solutions

A Walking Boy

Author: Federico Glaudo
Preparation: Andrea Ciprietti

Notice that you may assume that the judge has sent a message at minute 0 and at minute 1440 and this does not change the answer (but it simplifies the reasoning and the implementation).

Let us consider two consecutive messages sent by the judge, at times $s < t$.

If $t - s < 120$, then the judge cannot have walked the dog between the two messages.

If $120 \leq t - s < 240$, then the judge may have walked at most once between the two messages.

If $240 \leq t - s$, then the judge may have walked the dog two times between the two messages.

Hence, if $a_{i+1} - a_i \geq 240$ for some i, then the answer is `YES`. If $a_{i+1} - a_i \geq 120$ for two distinct values of i, then the answer is `YES`. Otherwise the answer is `NO`.

$\boxed{\text{B}}$ Uniform Chemistry

Author: Federico Glaudo and Petr Mitrichev
Preparation: Petr Mitrichev

The standard approach for this type of problem would be to use dynamic programming to compute the probability that each researcher wins the prize for each possible state of the process — the set of elements that each researcher has. However, the number of such sets can be as high as 100^{10}, which is clearly too big for us to process them all one by one.

Therefore the key idea is to use the independence of the processes that each researcher follows. Suppose we compute for each researcher i and each year j the probability p_{ij} that the i-th researcher discovers element n in the j-th year. In order for them to win the SWERC prize when doing so, the other researchers must have not discovered element n yet. The probability that researcher k has not yet discovered element n by the j-th year can be computed as $1 - \sum_{t<j} p_{kt}$, and because of the independence of different researchers, the probability that all other researchers have not yet discovered element n is equal to $\prod_{k\neq i}(1 - \sum_{t<j} p_{kt})$, and the probability that the i-th researcher wins the SWERC prize is equal to

$$\sum_j p_{ij} \prod_{k\neq i} \left(1 - \sum_{t<j} p_{kt}\right)$$

Now we just need to compute the probability p_{ij} that the i-th researcher discovers element n in the j-th year for all i and j. This can be done using dynamic programming that computes q_{ij}: the probability that a researcher that has element $n - i$ in the beginning (in other words, is i elements away from the goal) discovers element n in the j-th year. The probabilities that we want are then found as $p_{ij} = q_{n-s_i,j}$.

From the fusion experiment definition we directly obtain:

$$q_{ij} = \frac{1}{i} \sum_{0\leq k<i} q_{k,j-1}$$

We can then apply this formula in increasing order of i, or in increasing order of j, to find all those probabilities, starting from $q_{00} = 1$, $q_{i0} = 0$ for $i > 0$, and then apply the formula that combines those values into our answer.

The dynamic programming has $O(n^2)$ states, and each state is processed in $O(n)$, so its running time is $O(n^3)$. The final formula is computed in $O(n^2m)$ for each of the m researchers, so the overall running time is $O(n^2(n + m^2))$, which is fast enough for $n \leq 100$, $m \leq 10$.

$\boxed{\text{C}}$ Another Wine Tasting Event

Author: Andrea Ciprietti
Preparation: Andrea Ciprietti

For an interval $[l, r]$ ($1 \leq l \leq r \leq 2n-1$), let $w(l, r)$ be the number of `W`'s in the substring $s_l \dots s_r$.

We are going to show that $x = \max_{1 \leq l \leq n} w(l,\, l+n-1)$ is a solution (that is, there are at least n intervals of length $\geq n$ containing exactly x `W`'s).

Let k be an index such that $w(k,\, k+n-1) = x$. The idea is to find $n-1$ other intervals with the same number of `W`'s by "sliding" the original interval $[k,\, k+n-1]$ in a clever way.

For each $l = 1, 2, \dots, k-1$, let L_l be the shortest interval starting at l with $w(L_l) = x$. Such interval exists because $w(l,\, k+n-1) \geq w(k,\, k+n-1) = x$ and moreover the length of L_l cannot be smaller than n by definition of x. Similarly, for each $r = k+n, k+n+1, \dots, 2n-1$, let R_r be the shortest interval ending at r with $w(R_r) = x$.

We have constructed a family of n intervals: $L_1, L_2, \dots, L_{k-1}, [k,\, k+n-1], R_{k+n}, R_{k+n+1}, \dots, R_{2n-1}$. All of them have length $\geq n$ and all of them contain exactly x `W`'s. Are they all distinct? Clearly $L_i \neq L_j$ and $R_i \neq R_j$ whenever $i \neq j$. Moreover, $[k,\, k+n-1]$ is different from all the other intervals. Could it be that $L_l = R_r$? No, because that would mean that such interval is $[l, r]$ with $l < k$ and $k+n-1 < r$, but then $[k,\, k+n-1]$ would be strictly contained inside $[l, r]$, violating the minimality of L_l and R_r.

D Crossing the Railways

Author: Cesc Folch
Preparation: Cesc Folch

First of all, let us transform the given problem to a geometry problem. We consider the plane where the x coordinate represents the distance of Isona from the first validating machine (remember that Isona is always in the straight segment between the two validating machines), and the y coordinate represents time.

Thus, the point (a, b) of the plane corresponds to being a meters away from the first validating machine after b seconds. If we represent trains in this plane they are vertical segments from (r_i, a_i) to (r_i, b_i).

Let us look at how is represented the fact that Isona is m_j meters away from the validating machine and starts moving at constant speed v_j at time t_j, and does so for s_j seconds. We get a straight segment with endpoints (m_j, t_j) and $(m_j + v_j \cdot s_j,\, t_j + s_j)$.

So, the movement of Isona at constant speed for a period of time is represented by a straight segment.

The constraint on the speed of isona corresponds to a constraint on the slope of the segment. Let us suppose that a segment passes through the points (m_1, t_1) and (m_2, t_2) with $t_1 < t_2$. Since Isona cannot run backward, it must hold $m_1 \leq m_2$. Isona's constant speed is $v_I = \frac{m_2 - m_1}{t_2 - t_1}$. From the statement we know that $\frac{1}{v} \geq v_I \geq 0$, so we can write $\frac{t_2 - t_1}{m_2 - m_1} \geq v$. That means that the slope of the segment must be greater or equal then v.

From now on we will consider a segment to be *valid* if it does not cross any train segment (take into account that train segments are open, so the endpoints are not considered part of the segment) and has a valid slope (that is $m_1 = m_2$ or $\frac{t_2 - t_1}{m_2 - m_1} \geq v$).

The problem is equivalent to finding the minimum number of segments that constitute a path from $x = 0$ to $x = m + 1$ that does not intersect any train segment. This statement is similar to a BFS where the nodes are the valid segments and the edges the intersection points. The source node is the segment $(0, 0) - (0, s)$ and the target node is $(m + 1, 0) - (m + 1, s)$. The issue is that we have infinitely many valid segments. The following lemma allows us to consider only finitely many segments.

Lemma. A *special* segment is a valid segment such that at least one of the following holds:

- It contains $(0, 0)$ and has slope equal to v.
- It contains the upper endpoint of a train (i.e., (r_i, b_i)) and has slope equal to v.
- It contains two points (r_i, a_i) and (r_j, b_j) with $r_i < r_j$.

We can construct an optimal solution using only special segments.

From the previous lemma we see that we only need to consider $O(n^2)$ segments, and we can construct the full set of segments from the lemma in time $O(m \cdot n^2 \cdot \log(n))$, as for each valid segment we only need to check if it crosses any train segment, which can be done using binary search in each rail in total time $O(m \cdot \log(n))$.

But there may be $O(n^4)$ intersection points, so running a BFS on this graph is too slow. So let us see when Isona shall change speed in an optimal solution. She may change speed at a given railway or between two railways. It is never convienient to change speed between the first validating machine and the first railway or between the last railway and the second validating machine.

Lemma. Any solution using only special segments changes speed at most once between any two railways.

Proof. Any solution changing speed twice between two rails is using a segment which is strictly contained between two railways, hence it cannot be a special segment because, by definition, any special segment has at least one point with coordinate x that is integer.

With this new observation, we can now proceed to describe an efficient algorithm. For each $i = 1, \dots, m$, and for each segment j, we keep a value $d_j^{(i)}$ that denotes the minimum number of changes of speed that are necessary to reach this segment, starting from $x = 0$, considering only the region $0 \le x \le i$. If we know $d_j^{(m)}$ then we know the answer to the problem.

Given $d_j^{(i)}$ for all segments j, we will efficiently compute $d_j^{(i+1)}$ for all segments j.

Let us see how to deal with the intersections (the changes of speed) between two railways (or on one railway).

Let us consider all the special segments that intersect both railway i and railway $i+1$; these segments will be indexed by an integer j (from here on, we consider only these segments, as the other ones are not important for the changes that happen between railway i and railway $i+1$). Let us identify the j-th segment with a pair of integers (a_j, b_j) where a_j and b_j are the positions that the segment s_j has if we order the segments by the y coordinate of the crossing point with $x = i$ and $x = i + 1$ respectively. From now on we will only use the pair (a_j, b_j) to work with the segments.

Notice that having the pairs (a_j, b_j) is sufficient to tell if two segments intersect (taking care of intersections on the railways is a technical detail we skip in this explanation). If we have two segments j, k with $a_j > a_k$ and $b_j < b_k$ it means that they cross. Also if $a_j < a_k$ and $b_j > b_k$, then they cross.

For all segments j, it is trivial that $d_j^{(i+1)} \le d_j^{(i)}$. Moreover, if segment j and segment k cross, then necessarily $d_j^{(i+1)} < d_k^{(i)} + 1$. These two observations allow us to compute $d_j^{(i+1)}$, but how can we do it efficiently?

We shall use a Fenwick tree (or a segment tree) to efficiently compute the minimum over an interval.

First, for each j, we consider the crossings such that $a_j > a_k$ and $b_j < b_k$. We process them in increasing order of a_j. When processing the j-th segment, we update $d_j^{(i+1)}$ with the minimum over the interval $[b_j + 1, \infty]$ (if such value is smaller than the current value of d'_j). Then, we update the Fenwick with the value $d_j^{(i)} + 1$ at position b_j.

The other type of crossings, those such that $a_j < a_k$ and $b_j > b_k$, can be processes analogously by considering the segments in increasing order of b_j.

This allow us to process all the crossings between two railways in $O(n^2 \cdot \log(n))$. So, at the end, the total time complexity is $O(m \cdot n^2 \cdot \log(n))$.

$\boxed{\text{E}}$ Spinach Pizza

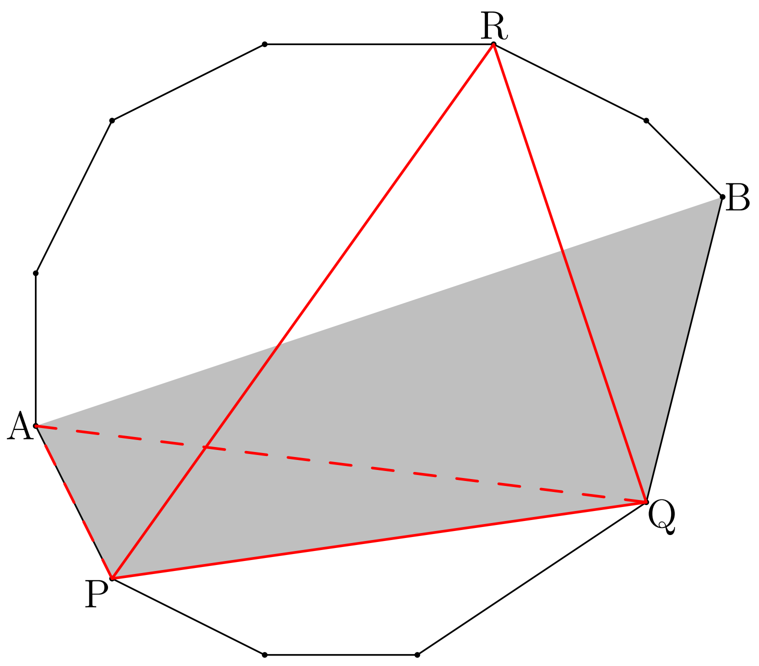
Author: Gerard Orriols
Preparation: Gerard Orriols

This problem has a greedy solution. The idea is that, at any moment, if the player who has to eat picks the triangle with smallest area among those that can be eaten in that moment, then any triangle chosen later will have area not less than it.

Therefore if n is even, Alberto can win by choosing the smallest possible slice at each turn, since the quantity eaten by Beatrice in the next turn will always be greater or equal and thus, since they eat the same number of slices, Alberto will eat no more in total. On the other hand, for n odd, no matter what slice Alberto chooses at the beginning, Beatrice can apply the explained strategy for the remaining number of turns, which is even. In this case Alberto will eat the initial amount plus at least the quantity Beatrice eats in the remaining turns, and therefore a total area strictly bigger than hers.

Now we prove the above claim. More precisely, given a convex polygon with vertices $P_1, \dots, P_n$, the minimum area of any triangle with vertices P_i, P_j, P_k is attained by a triangle determined by two consecutive edges in the polygon (we will call such triangles *edible*). Given a non-edible triangle, it is clear that we can label its vertices P, Q and R in such a way that neither of the edges PR and QR belong to the polygon. Then it is enough to show that we can change R by a vertex adjacent to P or Q and the remaining triangle will have at most the initial area. Indeed, by repeating this argument at most twice we can start with any triangle and obtain an edible triangle without increasing its area.

Let A and B be the vertices in the same side of the line PQ as R which are adjacent to P and Q, respectively. Then one of the triangles PQA or PQB has less area than PQR: since they all have the common base PQ, the area is proportional to the height with respect to the line PQ. However, if the height of R were less than that of A and B, then we would be able to write R as a convex combination of A, B, P and Q, which contradicts the strict convexity of the polygon.



$\boxed{\text{F}}$ Beppa and SwerChat

Author: Andrea Ciprietti
Preparation: Lifu Jin

Instead of finding the minimum number of members that must have been online at least once between 9:00 and 22:00, let us study the complement of that set: What is the maximum number of members that have never been online? We have the following observations:

- Members that have never been online must be a suffix of the sequence b. It cannot happen that member b_i has been online but b_{i-1} has not, for $2 \leq i \leq n$.
- Consider two members x and y who have never been online. Their relative order in a and b is the same. Indeed, if someone else goes online the relative order of x and y in the list does not change.

From the above observations, we deduce that the members who have not been online form a suffix of b that is also a subsequence of a. Let $p : \{1, 2, \dots, n\} \to \{1, 2, \dots, n\}$ be the function such that $a_{p(x)} = x$; we say that a sequence s of length m is a subsequence of a if $p(s_{i-1}) < p(s_i)$, for $2 \leq i \leq m$.

It turns out that any "suffix of b that is a subsequence of a" can be a valid subset of members who have never been online as the following construction shows. For any such suffix $b_t, b_{t+1}, \dots, b_n$, it could be that members $b_{t-1}, b_{t-2}, \dots, b_1$ went online *in this order* between 9:00 and 22:00, leading to a valid list b of last seen online at 22:00.

Thus, the answer to the problem is the length of the *longest* suffix of b that is also a subsequence of a. Let us explain how to find such length quickly.

We first preprocess the sequence a to find the corresponding function p. Then, we iterate from b_n to b_1. If, for some i we discover that $p(b_i) < p(b_{i-1})$, then we know that b_{i-1} must have been online, and we denote this position i as r. In this way, $b_r, b_{r+1}, \dots, b_n$ is the sought longest suffix of b. The minimum number of members that must have been online at least once between 9:00 and 22:00 is then $r - 1$.

G Parmigiana With Seafood

Author: Simon Mauras
Preparation: Simon Mauras

First, as for many tree problem, let us solve it first on chains, then generalize the solution for trees.

Game on a path

First, start with several observations on Alessandro's strategy.

Definition (bad pair). We say that two ingredients form a bad pair if they are at odd distance of each other.

Lemma 1. Alessandro can choose the best of the following strategies:

- First, he can include any ingredient which is a terminal ingredient at the begining of the game.
- Second, if n is even, then he can make sure to include ingredient n.
- Third, if n is odd and given a bad pair, he can always make sure that one of the two ingredient is included in the recipe.

Proof. The first observation is easy as Alessandro plays first. For the second observation, Alessandro's strategy is to include ingredient n whenever allowed to, and otherwise to select an ingredient which does not make n a terminal. Observe that when only 3 ingredients remain, such that n is the middle one, then it must be Bianca's turn, and Alessandro will be able to pick n on the next turn. For the third observation, the same argument hold, and it must be Bianca's turn when the only possible moves make either x or y terminal. An alternative explaination is to look at all the ingredients on the path between x and y (including x and y) as one very large ingredient (changing the parity of the number of items), and to apply the second observation.

Now, we have to check whether or not Alessandro can do even better, by looking at Bianca's strategy.

Lemma 2. Assuming that n is odd, consider a set S of non-terminal vertices which contain no bad pair of ingredients. Bianca can make sure that each ingredient of S is discarded.

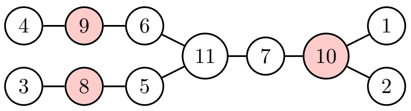
Proof. Bianca's strategy is (i) to select and discard ingredients from S whenever possible, and (ii) to only select ingredients which do not make any ingredient of S terminal. Observe that ingredients of S are not adjacent and thus (i) does not contradict (ii). To prove (ii), observe that whenever it is Bianca's turn the number of remaining ingredients is even. If only two ingredient remains then at most one can be in S and (ii) holds. If the chain has four or more ingredients then it has the form

$$a - x - \cdots - y - b$$

where x and y are at odd distance. In particular, either x or y is not is S, and Bianca can select and discard the corresponding neighbour and (ii) holds. Finally, we can prove by induction that whenever Alessandro plays, none of the terminal ingredients is in S, concluding the proof.

Game on a tree

We now move on to the same question, when the graph of ingredients is a tree. One can check that Lemma 1 still apply, with the exact same proof. However, the proof of Lemma 2 relied on the chain structure and does not hold for trees, as the following example shows.



Nodes in S colored in red. If Alessandro selects either 1 or 2, then Bianca will have to make one of the red ingredients terminal. This is caused by the fact that there is an even number of ingredients between red ingredients. Equivalently, this happens because there is a *bad triple*, as formalized in the following Definition and Lemma.

Definition (bad triple). We say that three ingredients x, y and z form a bad triple if there exist one indredient m whose removal would disconnect x, y and z (m is sometimes called the median), and such that all three ingredients x, y and z are at even distance of m.

In the example above, ingredients $x = 8$, $y = 9$ and $z = 10$ form a bad triple, as they are all three at distance two of their median $m = 11$.

Lemma 3. If n is odd and a bad triple is given, Alessandro can make sure one of the three ingredients of the triple is included in the recipe.

Proof. If one player is forced to make one of the three ingredients a terminal, then an even number of ingredient remains, and thus it is Bianca's turn to play.

Finally, we show that Alessandro cannot do better than combining strategies from Lemmas 1 and 3.

Lemma 4. Assuming that n is odd, consider a set S of non-terminal vertices which contain no bad pair and no bad triple of ingredients. Bianca can make sure that each ingredient of S is discarded.

Proof. Once again, Bianca's strategy is (i) to select and discard ingredients from S whenever possible, and (ii) to only select ingredients which do not make any ingredient of S terminal. To prove that (ii) holds, one can use an induction on $|S|$, which is left as an exercise.

Therefore if both players play optimally, the largest index included by Alessandro is characterized by Lemma 1 and 3. More precisely:

- if n is even then the answer is n,
- otherwise, it is the maximum between
  - the index of any terminal ingredient,
  - the index of any ingredient at odd distance to n,
  - $\min(x, y)$ for any bad triple (x, y, n),
  - $\min(x, y, z)$ for any bad triple (x, y, z) whose median is n.

In particular, this can be computed in linear time with a dfs in the tree (rooted in ingredient n).

H Controllers

Author: Stefanie Zbinden
Preparation: Stefanie Zbinden

Denote by p the number of + in s and by m the number of - in s and by tot the value $p - m$. We want to figure out whether we can win the game for a single controller with values x and y on the two buttons. If we only press the button with value x written on it, then our sum at the end is $(p - m) \cdot x = tot \cdot x$. So if $tot = 0$ we can always win the game and from here on we assume that $tot \neq 0$.

Assume we press the button with value x k_1 times when the symbol + comes up and k_2 times when the symbol - comes up. This means we press the button with value y $p - k_1$ times when the symbol + comes up and $m - k_2$ times when the symbol - comes up. Hence our total sum in the end is $k_1 \cdot x - k_2 \cdot x + (p - k_1)y - (m - k_2)y$.

We can define $k = k_1 - k_2$ and rewrite this as $k \cdot x + (tot - k)y$. Rewriting this even further gives that the sum is 0 if and only if $k(y - x) = tot \cdot y$. So if $x = y$ we cannot win the game (recall that we are assuming here that $tot \neq 0$). If $tot \cdot y$ is not divisible by $(y - x)$ we cannot win the game either.

Further, if $x \neq y$ we win if and only if $k = tot \cdot y/(y - x)$.

Can we get such value of k? Since $0 \leq k_1 \leq p$ and $0 \leq k_2 \leq m$, $k = k_1 - k_2$ can have any integer value in the range $[-m, p]$. Or in other words, we can win the game if and only if $tot \cdot y/(y - x)$ is an integer between $-m$ and p.

$\boxed{\text{I}}$ Library game

Author: Andrea Ciprietti
Preparation: Andrea Ciprietti

Sort the numbers $x_1, x_2, \dots, x_n$ in decreasing order, i.e. $x_1 \geq x_2 \geq \cdots \geq x_n$. For a real number x, let $\lfloor x \rfloor$ and $\lceil x \rceil$ denote, respectively the floor function of x (that is, the largest integer which does not exceed x) and the ceil function of x (that is, the smallest integer which is not less than x).

Lemma. Bernardo has a winning strategy if and only if there exists an index $1 \leq k \leq n$ such that $x_k > \lfloor m/k \rfloor$.

Proof. We will show the correctness of the criterion above by exhibiting a winning strategy for both Alessia (when the condition is not satisfied) and Bernardo (when the condition is satisfied).

- First, suppose that such an index k does not exist. Alessia will play by choosing the number x_i in her i-th turn (recall that the x_i's are sorted decreasingly). Let us show that, at the beginning of Alessia's i-th turn, there is necessarily an interval of length x_i that does not contain any number selected by Bernardo. Indeed, Bernardo has so far selected $i-1$ numbers, which form i "gaps" among the numbers $1, 2, \dots, m$. By the pigeonhole principle, one of these gaps contains at least $\left\lceil \frac{m-i+1}{i} \right\rceil = \left\lfloor \frac{m}{i} \right\rfloor \geq x_i$, where the last inequality follows from our hypothesis. Then, Alessia can safely choose this interval.

- Now suppose that $x_k > \lfloor m/k \rfloor$ for some k. Bernardo will play every turn as follows: if the interval chosen by Alessia contains at least one multiple of x_k, he selects one of those multiples; otherwise, he selects any number in the interval. Note that, every time Alessia chooses an x_i with $i \leq k$, whatever interval she chooses next will contain a multiple of x_k (because $x_i \geq x_k$). Since there are $\lfloor m/x_k \rfloor$ multiples of x_k in $[1, m]$, and the condition $x_k > \lfloor m/k \rfloor$ is equivalent to $k > \lfloor m/x_k \rfloor$, again by the pigeonhole principle there will necessarily be a turn where the multiple of x_k selected by Bernardo was already selected previously.

Implementing the strategies described is not difficult, due to the generous constraints that allow for implementations with $\mathcal{O}(nm)$ runtime.

Remark. In fact, if a "bad" k exists it is necessarily greater than 1, since $a_1 \leq m$ by the problem assumptions.

Remark. In the proofs above, we took for granted that $\left\lceil \frac{m-i+1}{i} \right\rceil = \left\lfloor \frac{m}{i} \right\rfloor$ and that $x_k > \lfloor m/k \rfloor$ if and only if $k > \lfloor m/x_k \rfloor$. The former is the well-known $\left\lceil \frac{a}{b} \right\rceil = \left\lfloor \frac{a+b-1}{b} \right\rfloor$ when a, b are integers (just replace a with $m-i+1$ and b with i). The latter can be proved by observing that both inequalities are equivalent to $k \cdot x_k > m$.

Understanding the game's criterion

How does one come up with Bernardo's winning condition $x_k > \lfloor m/k \rfloor$? While everyone has their own combination of methods, intuition and luck, in this problem it can be particularly useful to focus on small values of n, and specifically on the case $n = 2$ ($n = 1$ is not interesting at all: Alessia always wins!).

For $n = 2$, it might be easier to visualize what is going on, and to figure out that Bernardo wants to select the first number so that it is as close as possible to $m/2$ (indeed, he wants to minimize the largest of the two gaps that the selected number creates). After understanding this case, one can try to generalize to other values of n.

$\boxed{\text{J}}$ Italian Data Centers

Author: Pedro Paredes
Preparation: Pedro Paredes

This problem is ultimately about graphs, so let's start by translating the statement into a more formal language using graphs. Let $G = (V, E)$ be a graph with $|V| = n$ vertices and $|E| = m$ edges, where each vertex is colored in one of three colors. The problem statement describes an operation we shall call *doubling*, since in a way it "doubles" the graph. The double of G, denoted by $d(G)$, is a graph with $2n$ vertices and $2m + n$ edges defined as such:

1. For each $v \in V$, the double graph $d(G)$ contains two copies of v of the same color, v_1 and v_2. There is also an edge connecting v_1 and v_2 in $d(G)$.
2. For each $\{v, u\} \in E$: if v and u have the same color, then $d(G)$ contains an edge between v_1 and u_1, and an edge between v_2 and u_2; otherwise, $d(G)$ contains an edge between v_1 and u_2, and an edge between v_2 and u_1.

Now consider applying this doubling operation k times to G. Our goal is to find the diameter of the resulting graph, i.e. the largest distance (shortest path) between any two vertices in the graph $\underbrace{d(d(\cdots d(}_{k\text{ times}}G)\cdots))$. Let G_i be the graph after i doubling operations, so $G_i = \underbrace{d(d(\cdots d(}_{i\text{ times}}G)\cdots))$.

Dissecting the problem

At first glance the problem sounds really hard, one has to compute the diameter of an exponentially sized graph ($2^k n$ vertices), so we have to study the structure of the problem to gain some intuition.

Observation 1. Each vertex of G is copied 2^k times in G_k, so we can describe each vertex in G_k by a vertex in G and a length-k bitstring. For example, $(v, \texttt{011})$ would represent $((v_1)_2)_2$, so the second copy of the second copy of the first copy of v.

Before we continue with our study of the doubling process let's make one definition.

Definition 1. The *hypercube graph* of dimension n is a graph with 2^n vertices given by all length-n bitstrings such that there is an edge between two vertices if their associated bitstrings differ in exactly one element (i.e. if the Hamming distance between them is 1).

Observation 2. The edges in G_k between copies of a vertex v from G form an hypercube graph of dimension k, in other words, each subgraph of G_k induced by all of the copies of a single vertex forms an hypercube graph of dimension k.

This is easy to see inductively, but for completeness here is a proof: consider the subgraph in G_{i-1} induced by all the copies of $v \in V$ and assume this forms an hypercube graph of dimension $i - 1$. When forming G_i notice that since all of the vertices have the same color, step 2 of the doubling operation creates two copies of subgraph induced by the copies of v in G_{i-1}. The first copy has all the vertices whose bitstring starts with a `0` in the G_i representation, and the second copy has are all the vertices whose bitstring starts with a `1`. All the vertices in the first copy differ in the first bit from the ones on the second copy, so to form an hypercube graph of dimension i we need to add an edge between vertices with the same $i - 1$ last bits, which is exactly what step 1 of the doubling operation does.

Now that we understand the structure of copies of a single vertex, let's describe the edges between copies of different vertices of G. To make our notation easier to read, we refer to a vertex from G_i as (v, b), where $v \in V$ and b is a length-i bitstring.

Observation 3. Assume that $\{u, v\} \in E$. Consider a vertex (u, b) from G_k, where b is a length-k bitstring. (u, b) is connected to exactly one copy of v which is (v, b) if v and u have the same color, or $(v, \overline{b})$ if they have different colors, where $\overline{b}$ is the bitwise-negation of b.

We can see why this is true by noting that if v and u have the same color, then all the copies of u will be connected to the corresponding copy of v. If v and u have different colors, then we can see the observation by thinking inductively. Let $b^{(i)}$ be the length-i suffix of i. When creating G_i from G_{i-1} we go from $(u, b^{(i-1)})$ to $(u, b^{(i)})$ by adding b_i to the beginning of $b^{(i-1)}$. Since u and v have different colors, we connect $(u, b^{(i)})$ to $(v, \overline{b_i}\overline{b^{(i-1)}}) = (v, \overline{b^{(i)}})$.

Observation 4. Let $(u_1, b_u^1), (u_2, b_u^2), \dots, (u_\ell, b_u^\ell)$ be a path in G_k. Then there is a valid path of the same length $(v_1, b_v^1), (v_2, b_v^2), \dots, (v_\ell, b_v^\ell)$ and an index i such that $v_1 \neq v_2$ and $v_2 \neq v_3$, $\dots$, $v_{i-1} \neq v_i$ and $v_i = v_{i+1} = \dots = v_\ell$, so we first take all steps that move between vertices of G and then we take only steps in copies of v_i.

We obtain this by "rearranging" the steps in the first path. Given the symmetry of the graph, if we take one step between two copies of the same vertex too early, we can take that step at the end instead.

Computing the diameter

Let's start by describing the distance between two vertices (u, b_u) and (v, b_v) from G_k, where b_u and b_v are length-k bitstrings and $u, v \in V$ (not necessarily connected or distinct). But first, a quick definition:

Definition 2. Let P be a path in G between u and v. We call P an *even* path if the number of multicolored edges it uses is even, i.e. the number of times it walks to a vertex of a different color from the previous one is even. We analogously define *odd* paths.

Observation 5. The length of the shortest path between (u, b_u) and (v, b_v) is given by the shortest of the following:

- The length of the shortest even path between u and v plus $\Delta(b_u, b_v)$, where $\Delta(x, y)$ represents the Hamming distance between x and y, i.e. the number of positions at which the corresponding bits differ.
- The length of the shortest odd path between u and v plus $\Delta(b_u, \overline{b_v})$.

Proof. Consider a path in G_k of length ℓ from (u, b_u) to (v, b_v), say $(u_1, b_u^1), (u_2, b_u^2), \dots, (u_\ell, b_u^\ell)$, where each pair of consecutive vertices is connected and $(u_1, b_u^1) = (u, b_u)$ and $(u_\ell, b_u^\ell) = (u_v, b_v)$. Without loss of generality, we can assume that there is an i such that $u_1 \neq u_2$ and $u_2 \neq u_3$, $\dots$, $u_{i-1} \neq u_i$ and $u_i = u_{i+1} = \dots = v$, by Observation 4.

We can "project" this path down to G, i.e. drop all bitstring labels to obtain $u_1, u_2, \dots, u_\ell$. Note that this projected path is a path between u and v. If the projected path is an even path, then using Observation 3 we conclude that the ith vertex is (v, b_u), otherwise it is $(v, \overline{b_u})$. In the even case, the distance between (v, b_u) and (v, b_v) is given by $\Delta(b_u, b_v)$, and in the odd case the distance between $(v, \overline{b_u})$ and (v, b_v) is $\Delta(\overline{b_u}, b_v)$. This concludes the proof. □

Now, we can use this observation to describe the solution to the problem. First, because of the symmetries of the doubling operation, note that given a path $(u_1, b_u^1), (u_2, b_u^2), \dots, (u_\ell, b_u^\ell)$, there is another valid path $(u_1, \texttt{00}\dots\texttt{0}), (u_2, (b_u^2)'), \dots, (u_\ell, (b_u^\ell)')$, which is obtained by inverting all the `1` bits of b_1. This means that to find the diameter of G_k we only need to look at paths that start on vertices of the form $(v, \texttt{00}\ldots\texttt{0})$.

Given u, v from G, let's try to determine the b that maximizes the distance between $(u, \texttt{00}\ldots\texttt{0})$ and (v, b). From Observation 4 and 5, we know that we either first take an even path from $(u, \texttt{00}\ldots\texttt{0})$ to $(v, \texttt{00}\ldots\texttt{0})$, or an odd path from $(u, \texttt{00}\ldots\texttt{0})$ to $(v, \texttt{11}\ldots\texttt{1})$. Let $|b| = x$, i.e. the number of 1s in b is x. Then, the distance from $(v, \texttt{00}\ldots\texttt{0})$ to (v, b) is given by x and the distance from $(v, \texttt{11}\ldots\texttt{1})$ to (v, b) is given by $k - x$. Let's denote $\delta_e(u, v)$ be the shortest even path between u and v, and define $\delta_o(u, v)$ analogously. Then the maximum distance between $(u, \texttt{00}\ldots\texttt{0})$ and (v, b) for any b, is given by $\max_{0\le x\le k}\{\delta_e(u, v) + x, \delta_o(u, v) + k - x\}$.

So here is a possible algorithm:

Algorithm. For every pair of vertices u, v from G, compute the length of the shortest even path $(\delta_e(u, v))$, and the shortest odd path between them $(\delta_e(u, v))$. We can do this by running a Breadth-First Search starting at u that also keeps track of the parity of the number of times we've traversed multicolored edges. Then we compute $\max_{0\le x\le k}\{\delta_e(u, v) + x, \delta_o(u, v) + k - x\}$ and output the maximum of this over all pairs.

Note that this algorithm runs in time $O(nm + n^2k)$, since we need one BFS per vertex (which takes $O(nm)$ time) and then for each pair of vertices we need to compute $\max_{0\le x\le k}\{\delta_e(u, v)+x, \delta_o(u, v)+k - x\}$ (which takes $O(n^2k)$ time). This can be improved to $O(nm + n^2)$, but that isn't necessary for this problem since $n, k \le 100$.

K Train Splitting

Author: Alex Danilyuk
Preparation: Alex Danilyuk

The problem is equivalent to:

Formal statement: Given a connected graph, paint its edges with several colors so that the edges of any single color do not make the graph connected, but any 2 colors together make the graph connected.

This is a constructive problem and there can be a lot of different approaches. We left the limitations small on purpose, so that you could let your imagination run wild. We will describe a simple solution which works in time proportional to the size of the input.

How can we make sure the graph is not connected? Take a vertex and remove all incident edges. So if we paint edges incident to one vertex with color 1 and all other edges with color 2, then color 2 will not connect the graph, while colors 1 and 2 together will connect. And what about color 1 alone? It will connect the graph if and only if the chosen vertex was connected to all of the other vertices, so if we can find a vertex that is not connected to all of the other vertices, we are done.

Unless the graph is complete, i.e., it contains all possible edges, we can find such a vertex. If, on the other hand, the graph is complete, we can paint the edges from one vertex with two different colors (at least one of such edges with one color and at least one of such edges with the other), and all the other edges with the third color. You can verify that all the conditions are satisfied by this coloring.

The complexity of this solution is $O(n + m)$.

$\boxed{\text{L}}$ Vittorio Plays with LEGO Bricks

Author: Giovanni Paolini
Preparation: Alex Danilyuk

Each purple brick needs to be at the top of a chain of h bricks at heights $0, 1, \dots, h$ such that the x coordinates of consecutive bricks differ by at most 1. In an optimal structure, we may assume that every brick belongs to at least one of the n chains, and that any two chains that differ at some height h' also differ at all heights $\geq h'$.

We say that two chains diverge at height h' if their bricks coincide up to height $h' - 1$ but not at height $\geq h'$. In particular, two chains with different bricks at height 0 are said to diverge at height 0. For now, let's pretend that bricks can partially overlap with each other, so that there are no further constraints on how chains can be formed; at the end we will show how to account for overlapping bricks.

For $1 \leq l \leq r \leq n$, denote by $f(l, r)$ the minimum number of additional bricks needed to support the purple bricks $l, l+1, \dots, r$. We will recursively compute $f(l, r)$. For $l = r$, a single chain is needed and so $f(l, r) = h$.

Suppose now that $l < r$. For any m with $l \leq m < r$, we can build a structure as follows. First, place $f(l, m)$ blocks to support the purple bricks $l, l+1, \dots, m$. It is easy to see that we can modify such a structure so that the chain supporting the leftmost purple brick always climbs to the left, i.e., it consists of the bricks at positions $(x_l{+}h, 0, 0), (x_l{+}h{-}1, 0, 1), \dots, (x_l, 0, h)$. Similarly, add $f(m{+}1, r)$ bricks to support the purple bricks $m+1, m+2, \dots, r$, while ensuring that the chain supporting the rightmost purple brick always climbs to the right. If we set $h_{lr} := \max(0, h{+}1{-}\lceil (x_r{-}x_l)/2 \rceil)$, then it is possible to further change the leftmost and rightmost chains so that they coincide at heights $0, 1, \dots, h_{lr} - 1$ (and in fact diverge at height h_{lr}). We obtained a valid structure to support the purple bricks $l, l+1, \dots, r$ consisting of $f(l, m) + f(m+1, r) - h_{lr}$ additional bricks.

Conversely, suppose to have an optimal structure to support the purple bricks $l, l+1, \dots, r$. Let m be the index of any purple brick such that the m-th and the $(m+1)$-th chain diverge at the lowest possible height h'. In particular, all chains coincide at heights $0, 1, \dots, h' - 1$. Additionally, the first m chains share no bricks with the other chains at heights $\geq h'$. Note that $h' \leq h_{lr}$, otherwise it would not be possible to support both the l-th and the r-th purple bricks. Therefore, the number of non-purple bricks is at least $f(l, m) + f(m+1, r) - h' \geq f(l, m) + f(m+1, r) - h_{lr}$.

This allows us to use dynamic programming to calculate $f(l, r)$ through the following recursive formula:

$$f(l, r) = \min_{l \leq m < r} f(l, m) + f(m+1, r) - h_{lr}.$$

In particular, we can compute the answer $f(1, n)$ in $O(n^3)$ time.

To conclude, we now show that any optimal structure with overlapping bricks can be modified into an optimal structure with the same number of bricks which do not overlap. Suppose to have two overlapping bricks with x coordinates $\overline{x}$ and $\overline{x} + 1$ (and same height). There can't be any brick at the same height and with x coordinate equal to $\overline{x} - 2$, $\overline{x} - 1$, $\overline{x} + 2$, or $\overline{x} + 3$, otherwise we could remove one of the two overlapping bricks (it would not be needed to support the bricks above). Then we can move the left brick to position $\overline{x} - 1$ or the right brick to position $\overline{x} + 2$ (at least one of the two works, based on the position of the bricks in the row immediately below).