Statement

Problem A: Bijection

In this problem, you have to establish a bijection between combinations and pairs of a regular bracket sequence and an integer.

Details

In this problem, you have to establish a bijection between combinations and pairs of a regular bracket sequence and an integer.

- Combinations, $\mathrm{choose}(4, 2) = 6$:
  "RRUU", "RURU", "RUUR", "URRU", "URUR", "UURR".

Details

In this problem, you have to establish a bijection between combinations and pairs of a regular bracket sequence and an integer.

- Combinations, $\mathrm{choose}(4, 2) = 6$:
  "RRUU", "RURU", "RUUR", "URRU", "URUR", "UURR".
- Regular bracket sequences, $C_2 = 2$:
  "(())", "()()".
  Paired with an integer from $\{0, 1, 2\}$.

Details

In this problem, you have to establish a bijection between combinations and pairs of a regular bracket sequence and an integer.

- Combinations, $\mathrm{choose}(4, 2) = 6$:
  "RRUU", "RURU", "RUUR", "URRU", "URUR", "UURR".
- Regular bracket sequences, $C_2 = 2$:
  "(())", "()()".
  Paired with an integer from $\{0, 1, 2\}$.
- Combinations, $\mathrm{choose}(6, 3) = 20$:
  "RRRUUU", "RRURUU", "RRUURU", "RRUUUR", "RURRUU",
  "RURURU", "RURUUR", "RUURRU", "RUURUR", "RUUURR",
  "URRRUU", "URRURU", "URRUUR", "URURRU", "URURUR",
  "URUURR", "UURRRU", "UURURU", "UURUUR", "UUURRR".

Details

In this problem, you have to establish a bijection between combinations and pairs of a regular bracket sequence and an integer.

- Combinations, $\mathrm{choose}(4, 2) = 6$:
  "RRUU", "RURU", "RUUR", "URRU", "URUR", "UURR".
- Regular bracket sequences, $C_2 = 2$:
  "(())", "()()".
  Paired with an integer from $\{0, 1, 2\}$.
- Combinations, $\mathrm{choose}(6, 3) = 20$:
  "RRRUUU", "RRURUU", "RRUURU", "RRUUUR", "RURRUU",
  "RURURU", "RURUUR", "RUURRU", "RUURUR", "RUUURR",
  "URRRUU", "URRURU", "URRUUR", "URURRU", "URURUR",
  "URUURR", "UURRUU", "UURURU", "UURUUR", "UUURRR".
- Regular bracket sequences, $C_3 = 5$:
  "((()))", "(()())", "(())()", "()(())", "()()()".
  Paired with an integer from $\{0, 1, 2, 3\}$.

Enumeration

Solution 1: just number the objects lexicographically.

Enumeration

Solution 1: just number the objects lexicographically.

Requirements:

- big integers
- combination $\longleftrightarrow$ its number
- bracket sequence $\longleftrightarrow$ its number

Tedious but straightforward.
Note: using a non-default language might speed up development.

Proper Bijection

Solution 2: establish a natural bijection.

Proper Bijection

Solution 2: establish a natural bijection.

From path to bracket sequence.

- Write down the path, add an "`U`" at the end.
- Walk along the path and maintain the balance, where "`R`" adds $+1$ and "`U`" adds -1.
- Mark the first time when the balance was the least possible. The number k is the marked position minus one. The character before the marked position is "`U`".
- Now loop the path.
- Starting from the marked position, take $2n$ characters. Transform "`R`" $\to$ "`(`" and "`U`" $\to$ "`)`". This is guaranteed to be a regular bracket sequence.

Proper Bijection

Solution 2: establish a natural bijection.

From path to bracket sequence, example: "`RUURUR`".

- Write down the path, add an "`U`" at the end: "`RUURURU`".
- Walk along the path and maintain the balance, where "`R`" adds $+1$ and "`U`" adds -1: $\{+1, 0, -1, 0, -1, 0, -1\}$.
- Mark the first time when the balance was the least possible: "`RUU|RURU`". The number k is the marked position minus one: $k = 3 - 1 = 2$. The character before the marked position is "`U`".
- Now loop the path: "$\underbrace{\texttt{RUU|RURU}}\underbrace{\texttt{RUU|RURU}}\ldots$".
- Starting from the marked position, take $2n$ characters. Transform "`R`" $\to$ "`(`" and "`U`" $\to$ "`)`". This is guaranteed to be a regular bracket sequence: "`RURURU`" $\to$ "`()()()`".

Proper Bijection

Solution 2: establish a natural bijection.

From bracket sequence to path.

- Write down the sequence, add an “)” at the end.
- Now loop the sequence.
- Mark the position $2n - k$.
- Starting from the marked position, take $2n$ characters. Transform “(” $\rightarrow$ “R” and “)” $\rightarrow$ “U”.

Proper Bijection

Solution 2: establish a natural bijection.

From bracket sequence to path, example: "()()()", $k = 2$.

- Write down the sequence, add an ")" at the end: "()()())".
- Now loop the sequence: "$\underbrace{\texttt{()()())}}\underbrace{\texttt{()()())}}$...".
- Mark the position $2n - k = 6 - 2 = 4$: "$\underbrace{\texttt{()()|())}}\underbrace{\texttt{()()())}}$...".
- Starting from the marked position, take $2n$ characters. Transform "(" $\rightarrow$ "R" and ")" $\rightarrow$ "U": "())()(" $\rightarrow$ "RUURUR".

Other Bijections

Solution 3: other bijections are possible.

For example, there is one based on *exceedance*, see here:
`https://en.wikipedia.org/wiki/Catalan_number#Third_proof`.

Statement

Problem B: Rectangle Tree

In this problem, you have to rebuild a binary rooted tree of combinatorial rectangles so that it has logarithmic height compared to its size.

Definitions

In this problem, you have to rebuild a binary rooted tree of combinatorial rectangles so that it has logarithmic height compared to its size.

Just like a rectangle consisting of board squares is (consecutive subset) $\times$ (consecutive subset), a *combinatorial rectangle* is (arbitrary subset) $\times$ (arbitrary subset).

Definitions

In this problem, you have to rebuild a binary rooted tree of combinatorial rectangles so that it has logarithmic height compared to its size.

Just like a rectangle consisting of board squares is (consecutive subset) $\times$ (consecutive subset), a *combinatorial rectangle* is (arbitrary subset) $\times$ (arbitrary subset).

We have an $n \times n$ board consisting of colored squares.
We also have a rooted tree of combinatorial rectangles.
The root corresponds to the whole board.
Each vertex either is a leaf or has two children.
Children form a partition of the parent.
For each leaf, the cells it contains are colored the same.

Definitions

In this problem, you have to rebuild a binary rooted tree of combinatorial rectangles so that it has logarithmic height compared to its size.

Just like a rectangle consisting of board squares is (consecutive subset) $\times$ (consecutive subset), a *combinatorial rectangle* is (arbitrary subset) $\times$ (arbitrary subset).

We have an $n \times n$ board consisting of colored squares.
We also have a rooted tree of combinatorial rectangles.
The root corresponds to the whole board.
Each vertex either is a leaf or has two children.
Children form a partition of the parent.
For each leaf, the cells it contains are colored the same.

Our task is to construct a tree with the same properties.
Its size must not increase too much.
Its height must be logarithmic in terms of size.

Solution: Restricton

Let us define *restriction* of a tree to a combinatorial rectangle $A \times B$:

- Intersect all combinatorial rectangles in the tree with $A \times B$.
- Remove all empty rectangles.
- If only one child is not empty, promote it to parent.

Solution

Let U be the whole set $[0, n-1]$.

Solution

Let U be the whole set $[0, n-1]$.

Now, construct the new balanced tree as follows:

- Find the center C of the tree: its subtrees are of size $\leqslant S/3$, but its own subtree is of size $\geqslant S/3$.
- Let the rectangle in C be $A \times B$.

Solution

Let U be the whole set $[0, n-1]$.

Now, construct the new balanced tree as follows:

- Find the center C of the tree: its subtrees are of size $\leqslant S/3$, but its own subtree is of size $\geqslant S/3$.
- Let the rectangle in C be $A \times B$.
- Then, the new root is $U \times U$.
- The left child is $A \times U$.
- Its children are $A \times B$ and $A \times (U \setminus B)$.
- The right child is $(U \setminus A) \times U$.

Solution

Let U be the whole set $[0, n-1]$.

Now, construct the new balanced tree as follows:

- Find the center C of the tree: its subtrees are of size $\leqslant S/3$, but its own subtree is of size $\geqslant S/3$.
- Let the rectangle in C be $A \times B$.
- Then, the new root is $U \times U$.
- The left child is $A \times U$.
- Its children are $A \times B$ and $A \times (U \setminus B)$.
- The right child is $(U \setminus A) \times U$.
- For each of the three lower vertices, restrict the whole initial tree to its combinatorial rectangle, then construct a balanced subtree from it, with respective rectangles instead of $U \times U$.

Further Reading

Further reading: the topic can be googled as "balancing communication protocol".

Statement

Problem C: Fastest Arrival

In this problem, you have to find an integer point in a disc that is closest to a given point on the plane.

It Can't Be Too Far

In this problem, you have to find an integer point in a disc that is closest to a given point on the plane.

- Degenerate case: the given point is in the disc already. Answer is 0.

It Can't Be Too Far

In this problem, you have to find an integer point in a disc that is closest to a given point on the plane.

- Degenerate case: the given point is in the disc already. Answer is 0.
- Let our point be P, and the disc have center C and radius r.

It Can't Be Too Far

In this problem, you have to find an integer point in a disc that is closest to a given point on the plane.

- Degenerate case: the given point is in the disc already. Answer is 0.
- Let our point be P, and the disc have center C and radius r.
- Find the point M where PC intersects the circle. Denote $d = |PM|$.

It Can't Be Too Far

In this problem, you have to find an integer point in a disc that is closest to a given point on the plane.

- Degenerate case: the given point is in the disc already. Answer is 0.
- Let our point be P, and the disc have center C and radius r.
- Find the point M where PC intersects the circle. Denote $d = |PM|$.
- Let us move along the circumference. How far can the desired integer point be?

It Can't Be Too Far

In this problem, you have to find an integer point in a disc that is closest to a given point on the plane.

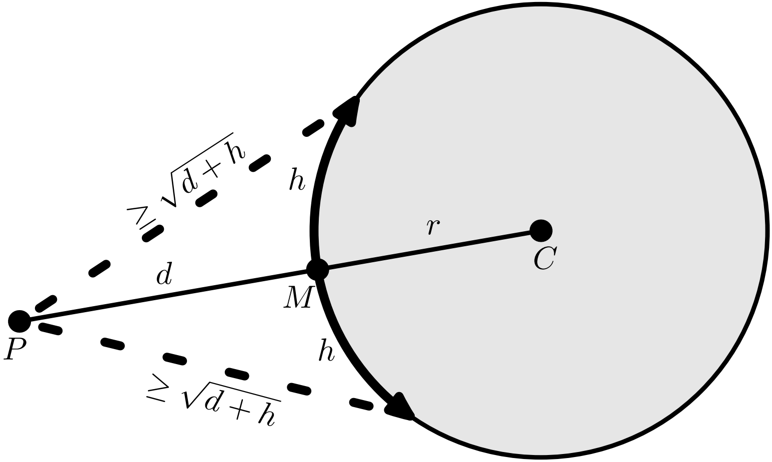
- Degenerate case: the given point is in the disc already. Answer is 0.
- Let our point be P, and the disc have center C and radius r.
- Find the point M where PC intersects the circle. Denote $d = |PM|$.
- Let us move along the circumference. How far can the desired integer point be?
- If we walk h units from M along the circumference, the distance from P will become more than $\sqrt{d^2 + h^2}$.

It Can't Be Too Far

In this problem, you have to find an integer point in a disc that is closest to a given point on the plane.

- Degenerate case: the given point is in the disc already. Answer is 0.
- Let our point be P, and the disc have center C and radius r.
- Find the point M where PC intersects the circle. Denote $d = |PM|$.
- Let us move along the circumference. How far can the desired integer point be?
- If we walk h units from M along the circumference, the distance from P will become more than $\sqrt{d^2 + h^2}$.
- When $h \geqslant 2\sqrt{d} + 1$, that distance is at least $d + 1$. So we are farther from P already than some integer point in the circle which is closest to M.

Illustration



Walking Along the Circumference

So, to be sure, we can just look at around 100 000 integer points close to circumference in each direction.

Walking Along the Circumference

So, to be sure, we can just look at around 100 000 integer points close to circumference in each direction.

- Classy: Bresenham algorithm.

Walking Along the Circumference

So, to be sure, we can just look at around 100 000 integer points close to circumference in each direction.

- Classy: Bresenham algorithm.
- Crude: for 100 000 adjacent x, find the best y; then, for 100 000 adjacent y, find the best x.

Walking Along the Circumference

So, to be sure, we can just look at around 100 000 integer points close to circumference in each direction.

- Classy: Bresenham algorithm.
- Crude: for 100 000 adjacent x, find the best y; then, for 100 000 adjacent y, find the best x.
- More crude: when searching for the "best" y, consider a couple adjacent y, and for each of them, check the distance to the circle center. Same for the "best" x.

Walking Along the Circumference

So, to be sure, we can just look at around 100 000 integer points close to circumference in each direction.

- Classy: Bresenham algorithm.
- Crude: for 100 000 adjacent x, find the best y; then, for 100 000 adjacent y, find the best x.
- More crude: when searching for the "best" y, consider a couple adjacent y, and for each of them, check the distance to the circle center. Same for the "best" x.

So, why is the success rate for this problem so low?

Walking Along the Circumference

So, to be sure, we can just look at around 100 000 integer points close to circumference in each direction.

- Classy: Bresenham algorithm.
- Crude: for 100 000 adjacent x, find the best y; then, for 100 000 adjacent y, find the best x.
- More crude: when searching for the "best" y, consider a couple adjacent y, and for each of them, check the distance to the circle center. Same for the "best" x.

So, why is the success rate for this problem so low?

- Not trying far enough: in the tests, the optimal point can be more than 10 000 units far from M.

Walking Along the Circumference

So, to be sure, we can just look at around 100 000 integer points close to circumference in each direction.

- Classy: Bresenham algorithm.
- Crude: for 100 000 adjacent x, find the best y; then, for 100 000 adjacent y, find the best x.
- More crude: when searching for the "best" y, consider a couple adjacent y, and for each of them, check the distance to the circle center. Same for the "best" x.

So, why is the success rate for this problem so low?

- Not trying far enough: in the tests, the optimal point can be more than 10 000 units far from M.
- In a crude solution, neglect having both x and y parts.

Walking Along the Circumference

So, to be sure, we can just look at around 100 000 integer points close to circumference in each direction.

- Classy: Bresenham algorithm.
- Crude: for 100 000 adjacent x, find the best y; then, for 100 000 adjacent y, find the best x.
- More crude: when searching for the "best" y, consider a couple adjacent y, and for each of them, check the distance to the circle center. Same for the "best" x.

So, why is the success rate for this problem so low?

- Not trying far enough: in the tests, the optimal point can be more than 10 000 units far from M.
- In a crude solution, neglect having both x and y parts.
- Overflows or precision errors.

Walking Along the Circumference

So, to be sure, we can just look at around 100 000 integer points close to circumference in each direction.

- Classy: Bresenham algorithm.
- Crude: for 100 000 adjacent x, find the best y; then, for 100 000 adjacent y, find the best x.
- More crude: when searching for the "best" y, consider a couple adjacent y, and for each of them, check the distance to the circle center. Same for the "best" x.

So, why is the success rate for this problem so low?

- Not trying far enough: in the tests, the optimal point can be more than 10 000 units far from M.
- In a crude solution, neglect having both x and y parts.
- Overflows or precision errors.
- ...tell us about your case!

Statement

Problem D: Lost in Transfer

In this problem, you have to recover the numbers being transmitted, even in the unfortunate case one of them is lost.

Details

In this problem, you have to recover the numbers being transmitted, even in the unfortunate case one of them is lost.

- You are given a set of 20 to 100 distinct integers from 1 to 500.

Details

In this problem, you have to recover the numbers being transmitted, even in the unfortunate case one of them is lost.

- You are given a set of 20 to 100 distinct integers from 1 to 500.
- You have to transmit each of them exactly once.

Details

In this problem, you have to recover the numbers being transmitted, even in the unfortunate case one of them is lost.

- You are given a set of 20 to 100 distinct integers from 1 to 500.
- You have to transmit each of them exactly once.
- You control the order of transmission.

Details

In this problem, you have to recover the numbers being transmitted, even in the unfortunate case one of them is lost.

- You are given a set of 20 to 100 distinct integers from 1 to 500.
- You have to transmit each of them exactly once.
- You control the order of transmission.
- You then receive the same numbers in the same order, but one of them may be missing.

Details

In this problem, you have to recover the numbers being transmitted, even in the unfortunate case one of them is lost.

- You are given a set of 20 to 100 distinct integers from 1 to 500.
- You have to transmit each of them exactly once.
- You control the order of transmission.
- You then receive the same numbers in the same order, but one of them may be missing.
- Nevertheless, you have to recover the original set (order is irrelevant).

Order Does Matter

Naturally, you will have to use the order to transfer additional information.

Order Does Matter

Naturally, you will have to use the order to transfer additional information.

- Calculate some redundant statistic of all the numbers in the set: xor-sum, or sum modulo 501, or...

Order Does Matter

Naturally, you will have to use the order to transfer additional information.

- Calculate some redundant statistic of all the numbers in the set: xor-sum, or sum modulo 501, or...
- In the first run, encode this value using the order of elements.

Order Does Matter

Naturally, you will have to use the order to transfer additional information.

- Calculate some redundant statistic of all the numbers in the set: xor-sum, or sum modulo 501, or...
- In the first run, encode this value using the order of elements.
- In the second run, calculate the same statistic for the received set, extract the original value from the order, and compare them.

Order Does Matter

Naturally, you will have to use the order to transfer additional information.

- Calculate some redundant statistic of all the numbers in the set: xor-sum, or sum modulo 501, or...
- In the first run, encode this value using the order of elements.
- In the second run, calculate the same statistic for the received set, extract the original value from the order, and compare them.
- If they are equal, no element is missing.

Order Does Matter

Naturally, you will have to use the order to transfer additional information.

- Calculate some redundant statistic of all the numbers in the set: xor-sum, or sum modulo 501, or...
- In the first run, encode this value using the order of elements.
- In the second run, calculate the same statistic for the received set, extract the original value from the order, and compare them.
- If they are equal, no element is missing.
- Otherwise, use the two statistics to determine the missing element.

Bitwise Solution

Solution 1: encode individual bits of the statistic.

Bitwise Solution

Solution 1: encode individual bits of the statistic.

- Sort the elements of the set.

Bitwise Solution

Solution 1: encode individual bits of the statistic.

- Sort the elements of the set.
- Write each bit twice.

Bitwise Solution

Solution 1: encode individual bits of the statistic.

- Sort the elements of the set.
- Write each bit twice.
- To encode a 1 bit, print the maximal remaining element two times.

Bitwise Solution

Solution 1: encode individual bits of the statistic.

- Sort the elements of the set.
- Write each bit twice.
- To encode a `1` bit, print the maximal remaining element two times.
- To encode a `0` bit, print the minimal remaining element two times.

Bitwise Solution

Solution 1: encode individual bits of the statistic.

- Sort the elements of the set.
- Write each bit twice.
- To encode a `1` bit, print the maximal remaining element two times.
- To encode a `0` bit, print the minimal remaining element two times.
- Proceed until there are no more elements left.

Bitwise Solution

Solution 1: encode individual bits of the statistic.

- Sort the elements of the set.
- Write each bit twice.
- To encode a `1` bit, print the maximal remaining element two times.
- To encode a `0` bit, print the minimal remaining element two times.
- Proceed until there are no more elements left.
- Example: to encode bits `10110...` (followed by zeroes) with the numbers 1, 2, ..., 20, print:

$$\underbrace{20, 19}_{for 1}, \underbrace{1, 2}_{for 0}, \underbrace{18, 17}_{for 1}, \underbrace{16, 15}_{for 1}, \underbrace{3, 4}_{for 0}, \ldots, \underbrace{13, 14}_{for 0}.$$

Bitwise Solution

Solution 1: encode individual bits of the statistic.

- Decoding: consider the relations between adjacent elements.

Bitwise Solution

Solution 1: encode individual bits of the statistic.

- Decoding: consider the relations between adjacent elements.
- Here, we have:
  $20 > 19 > 1 < 2 < 18 > 17 > 16 > 15 > 3 < 4 < \ldots < 13 < 14$.

Bitwise Solution

Solution 1: encode individual bits of the statistic.

- Decoding: consider the relations between adjacent elements.
- Here, we have:
  $20 > 19 > 1 < 2 < 18 > 17 > 16 > 15 > 3 < 4 < \ldots < 13 < 14$.
- If no element is missing, inequalities come in pairs:
  >><<>>>><<<<...<<.

Bitwise Solution

Solution 1: encode individual bits of the statistic.

- Decoding: consider the relations between adjacent elements.
- Here, we have:
  $20 > 19 > 1 < 2 < 18 > 17 > 16 > 15 > 3 < 4 < \ldots < 13 < 14$.
- If no element is missing, inequalities come in pairs:
  >><<>>>><<<<...<<.
- If an element is missing, some run of equal signs will have odd length.

Bitwise Solution

Solution 1: encode individual bits of the statistic.

- Decoding: consider the relations between adjacent elements.
- Here, we have:
  $20 > 19 > 1 < 2 < 18 > 17 > 16 > 15 > 3 < 4 < \ldots < 13 < 14.$
- If no element is missing, inequalities come in pairs:
  $>><<>>>><<<<\ldots<<.$
- If an element is missing, some run of equal signs will have odd length.
- For example, if 17 is missing, we still have:
  $20 > 19 > 1 < 2 < 18 > 16 > 15 > 3 < 4 < \ldots < 13 < 14.$

Bitwise Solution

Solution 1: encode individual bits of the statistic.

- Decoding: consider the relations between adjacent elements.
- Here, we have:
  $20 > 19 > 1 < 2 < 18 > 17 > 16 > 15 > 3 < 4 < \ldots < 13 < 14.$
- If no element is missing, inequalities come in pairs:
  $>><<>>>><<<<\ldots<<.$
- If an element is missing, some run of equal signs will have odd length.
- For example, if 17 is missing, we still have:
  $20 > 19 > 1 < 2 < 18 > 16 > 15 > 3 < 4 < \ldots < 13 < 14.$
- The way to decode it: take the next sign and transform $> \to 1$ and $< \to 0$; remove the sign from consideration; if the sign after it is the same, remove it too.

Randomized Solution

Solution 2: encode each possible value with a random permutation.

Randomized Solution

Solution 2: encode each possible value with a random permutation.

- Assign a pseudorandom permutation $p(i)$ (same each run) of length 20 to each possible value i (perhaps values can range from 0 to 511).

Randomized Solution

Solution 2: encode each possible value with a random permutation.

- Assign a pseudorandom permutation $p(i)$ (same each run) of length 20 to each possible value i (perhaps values can range from 0 to 511).
- In the first run, reorder the first 20 elements of the set according to the permutation $p(x)$ corresponding to the value x you want to transfer.

Randomized Solution

Solution 2: encode each possible value with a random permutation.

- Assign a pseudorandom permutation $p(i)$ (same each run) of length 20 to each possible value i (perhaps values can range from 0 to 511).
- In the first run, reorder the first 20 elements of the set according to the permutation $p(x)$ corresponding to the value x you want to transfer.
- In the second run, look at the permutation q of the first 19 elements.

Randomized Solution

Solution 2: encode each possible value with a random permutation.

- Assign a pseudorandom permutation $p(i)$ (same each run) of length 20 to each possible value i (perhaps values can range from 0 to 511).
- In the first run, reorder the first 20 elements of the set according to the permutation $p(x)$ corresponding to the value x you want to transfer.
- In the second run, look at the permutation q of the first 19 elements.
- With high probability, there is only one possible y such that permuting according to $p(y)$ and then dropping one of the elements could have produced q.

Randomized Solution

Solution 2: encode each possible value with a random permutation.

- Assign a pseudorandom permutation $p(i)$ (same each run) of length 20 to each possible value i (perhaps values can range from 0 to 511).
- In the first run, reorder the first 20 elements of the set according to the permutation $p(x)$ corresponding to the value x you want to transfer.
- In the second run, look at the permutation q of the first 19 elements.
- With high probability, there is only one possible y such that permuting according to $p(y)$ and then dropping one of the elements could have produced q.
- Just check all $512 \cdot 20$ possibilities to be sure!

Randomized Solution

Solution 2: encode each possible value with a random permutation.

- Assign a pseudorandom permutation $p(i)$ (same each run) of length 20 to each possible value i (perhaps values can range from 0 to 511).
- In the first run, reorder the first 20 elements of the set according to the permutation $p(x)$ corresponding to the value x you want to transfer.
- In the second run, look at the permutation q of the first 19 elements.
- With high probability, there is only one possible y such that permuting according to $p(y)$ and then dropping one of the elements could have produced q.
- Just check all $512 \cdot 20$ possibilities to be sure!
- Then y is the value x you wanted to transfer.

Statement

Problem E: Maze with a Hint

In this problem, you have to first look at a maze and write a short hint, and then look at the hint and interactively solve the maze.

Solving the Maze

In this problem, you have to first look at a maze and write a short hint, and then look at the hint and interactively solve the maze.

- How do we solve a maze with full information?

Solving the Maze

In this problem, you have to first look at a maze and write a short hint, and then look at the hint and interactively solve the maze.

- How do we solve a maze with full information?
  - Breadth-first search.

Solving the Maze

In this problem, you have to first look at a maze and write a short hint, and then look at the hint and interactively solve the maze.

- How do we solve a maze with full information?
  - Breadth-first search.
- How do we solve a maze when we only see the current square?

Solving the Maze

In this problem, you have to first look at a maze and write a short hint, and then look at the hint and interactively solve the maze.

- How do we solve a maze with full information?
  - Breadth-first search.
- How do we solve a maze when we only see the current square?
  - Right-hand rule: move along the maze so that you always keep touching the wall with your right hand.

Solving the Maze

In this problem, you have to first look at a maze and write a short hint, and then look at the hint and interactively solve the maze.

- How do we solve a maze with full information?
  - Breadth-first search.
- How do we solve a maze when we only see the current square?
  - Right-hand rule: move along the maze so that you always keep touching the wall with your right hand.
  - Heuristic: maintain the map of visited squares, and move to the "best" unknown square, in some sense (closest to us, most probably on the right path, ...).

Bumping into Constraints

- Maze is up to 200×200 squares.
  Hint is up to 1000 bits.
  Path is up to 6000 steps.

Bumping into Constraints

- Maze is up to 200×200 squares.
  Hint is up to 1000 bits.
  Path is up to 6000 steps.

- If we do a breadth-first search, we find the shortest path.
  In the tests, it is up to 1394 steps long.
  Remember that you have half the tests, so you can check that without writing a test generator!

Bumping into Constraints

- Maze is up to 200×200 squares.
  Hint is up to 1000 bits.
  Path is up to 6000 steps.

- If we do a breadth-first search, we find the shortest path.
  In the tests, it is up to 1394 steps long.
  Remember that you have half the tests, so you can check that without writing a test generator!

- If we solve using the right-hand rule, the path will contain half the squares *on average*.

Packing the Optimal Path

Solution 1: just encode the shortest path into 1000 bits we have.

Packing the Optimal Path

Solution 1: just encode the shortest path into 1000 bits we have.

- On each step, we have 1, 2, or 3 possible paths forward.

Packing the Optimal Path

Solution 1: just encode the shortest path into 1000 bits we have.

- On each step, we have 1, 2, or 3 possible paths forward.
- If there is a single path forward, no hint is needed.

Packing the Optimal Path

Solution 1: just encode the shortest path into 1000 bits we have.

- On each step, we have 1, 2, or 3 possible paths forward.
- If there is a single path forward, no hint is needed.
- Baseline: Remember each remaining step using two bits.
  Hint length: 1530, still too long.

Packing the Optimal Path

Solution 1: just encode the shortest path into 1000 bits we have.

- On each step, we have 1, 2, or 3 possible paths forward.
- If there is a single path forward, no hint is needed.
- Baseline: Remember each remaining step using two bits.
  Hint length: 1530, still too long.
- Improved: Use one bit when there are 2 possible paths, two bits when there are 3.
  Hint length: 910, OK.

Packing the Optimal Path

Solution 1: just encode the shortest path into 1000 bits we have.

- On each step, we have 1, 2, or 3 possible paths forward.
- If there is a single path forward, no hint is needed.
- Baseline: Remember each remaining step using two bits.
  Hint length: 1530, still too long.
- Improved: Use one bit when there are 2 possible paths, two bits when there are 3.
  Hint length: 910, OK.
- Big-integer: Use one bit when there are 2 possible paths, one ternary digit when there are 3. Pack all that into a big integer, and print it in binary as a hint.
  Hint length: 850, OK.

Remembering Critical Crossroads

Solution 2: walk using the right-hand rule, but take a hint at 4-crossroads.

Remembering Critical Crossroads

Solution 2: walk using the right-hand rule, but take a hint at 4-crossroads.

- A 4-crossroads is a square with all 4 paths open.

Remembering Critical Crossroads

Solution 2: walk using the right-hand rule, but take a hint at 4-crossroads.

- A 4-crossroads is a square with all 4 paths open.
- There will be only few 4-crossroads along our path.

Remembering Critical Crossroads

Solution 2: walk using the right-hand rule, but take a hint at 4-crossroads.

- A 4-crossroads is a square with all 4 paths open.
- There will be only few 4-crossroads along our path.
- On each of them, remember the right direction (found by breadth-first search) using two bits.

Remembering Critical Crossroads

Solution 2: walk using the right-hand rule, but take a hint at 4-crossroads.

- A 4-crossroads is a square with all 4 paths open.
- There will be only few 4-crossroads along our path.
- On each of them, remember the right direction (found by breadth-first search) using two bits.
- Right-hand rule: hint length 430, path length 5530.

Remembering Critical Crossroads

Solution 2: walk using the right-hand rule, but take a hint at 4-crossroads.

- A 4-crossroads is a square with all 4 paths open.
- There will be only few 4-crossroads along our path.
- On each of them, remember the right direction (found by breadth-first search) using two bits.
- Right-hand rule: hint length 430, path length 5530.
- Left-hand rule: hint length 406, path length 4858.

Other Solutions Are Possible

Solution 3: (insert your solution here).

Statement

Problem F: Maharajas Are Going Home

In this problem, you have to find a winning move for a disjunctive sum of games, or determine that there is none.

Details

In this problem, you have to find a winning move for a disjunctive sum of games, or determine that there is none.

A maharaja has all the moves available to chess queen and chess knight. In this problem, it can move only towards the origin, and multiple maharajas can occupy the same square.

Details

In this problem, you have to find a winning move for a disjunctive sum of games, or determine that there is none.

A maharaja has all the moves available to chess queen and chess knight. In this problem, it can move only towards the origin, and multiple maharajas can occupy the same square.

You are given the initial positions of maharajas on a board which is infinite in positive directions.

Details

In this problem, you have to find a winning move for a disjunctive sum of games, or determine that there is none.

A maharaja has all the moves available to chess queen and chess knight. In this problem, it can move only towards the origin, and multiple maharajas can occupy the same square.

You are given the initial positions of maharajas on a board which is infinite in positive directions.

Two players move in turns, the player who can not make a move loses.

Details

In this problem, you have to find a winning move for a disjunctive sum of games, or determine that there is none.

A maharaja has all the moves available to chess queen and chess knight. In this problem, it can move only towards the origin, and multiple maharajas can occupy the same square.

You are given the initial positions of maharajas on a board which is infinite in positive directions.

Two players move in turns, the player who can not make a move loses.

Find the lexicographically smallest winning move in the following form: number of maharaja, destination row, destination column.

Details

In this problem, you have to find a winning move for a disjunctive sum of games, or determine that there is none.

A maharaja has all the moves available to chess queen and chess knight. In this problem, it can move only towards the origin, and multiple maharajas can occupy the same square.

You are given the initial positions of maharajas on a board which is infinite in positive directions.

Two players move in turns, the player who can not make a move loses.

Find the lexicographically smallest winning move in the following form: number of maharaja, destination row, destination column.

Or determine there is no such move.

Grundy Numbers

The problem boils down to finding Grundy value for every possible position of a maharaja. Then, compute the xor-sum of the given maharajas. If it is zero, there is no winning move. Otherwise, iterate over all possible moves and find the lexicographically first move that makes the xor-sum zero.

Grundy Numbers

The problem boils down to finding Grundy value for every possible position of a maharaja. Then, compute the xor-sum of the given maharajas. If it is zero, there is no winning move. Otherwise, iterate over all possible moves and find the lexicographically first move that makes the xor-sum zero.

Straightforward solution, by definition: for every square, construct a set of Grundy values after all possible moves, and find its minimal excludant.

Bitsets

How to find Grundy values fast?

Bitsets

How to find Grundy values fast?

- For each square, the set consists of the values in its row, its column, its diagonal, and the two knight's moves.

Bitsets

How to find Grundy values fast?

- For each square, the set consists of the values in its row, its column, its diagonal, and the two knight's moves.
- Maintain the sets for each row, column, and diagonal.
  When we learn the value in a new square, update these sets.

Bitsets

How to find Grundy values fast?

- For each square, the set consists of the values in its row, its column, its diagonal, and the two knight's moves.
- Maintain the sets for each row, column, and diagonal.
  When we learn the value in a new square, update these sets.
- For each square, the slowest operation is to find the union of the respective three sets, then find its minimal excludant.

Bitsets

How to find Grundy values fast?

- For each square, the set consists of the values in its row, its column, its diagonal, and the two knight's moves.
- Maintain the sets for each row, column, and diagonal.
  When we learn the value in a new square, update these sets.
- For each square, the slowest operation is to find the union of the respective three sets, then find its minimal excludant.
- Speedup: store sets as bitsets.
  Then we effectively operate on 32 or 64 elements at once.

Optimizations

What if it is still too slow?

Optimizations

What if it is still too slow?

- Symmetry: rows have the same sets as columns, lower diagonals have the same sets as upper diagonals.
  The table of Grundy values is also symmetric.

Optimizations

What if it is still too slow?

- Symmetry: rows have the same sets as columns, lower diagonals have the same sets as upper diagonals.
  The table of Grundy values is also symmetric.
- When looking for minimal excludant in a row, start with the previous value for that row, instead of starting from 0 every time.

Optimizations

What if it is still too slow?

- Symmetry: rows have the same sets as columns, lower diagonals have the same sets as upper diagonals.
  The table of Grundy values is also symmetric.
- When looking for minimal excludant in a row, start with the previous value for that row, instead of starting from 0 every time.
- When looking for the best move, consider only possible moves, instead of iterating over the whole board.

Statement

Problem G: Ook

In this problem, you have to cut pieces from a string, match them to a pattern, and maximize the sum of results.

Details

In this problem, you have to cut pieces from a string, match them to a pattern, and maximize the sum of results.

Example: $o = 3$, $k = 5$, $S =$ `ookkoko`, $P =$ `ok?`.

Details

In this problem, you have to cut pieces from a string, match them to a pattern, and maximize the sum of results.

Example: $o = 3$, $k = 5$, $S =$ `ookkoko`, $P =$ `ok?`.

```
ookkoko
ok?        miss 1, value 3 + 3 + 5 = 11, result 11/2^1 = 5.
 ok?       miss 0, value 3 + 5 + 5 = 13, result 13/2^0 = 13.
  ok?      miss 1, value 5 + 5 + 3 = 13, result 13/2^1 = 6.
   ok?     miss 2, value 5 + 3 + 5 = 13, result 13/2^2 = 3.
    ok?    miss 0, value 3 + 5 + 3 = 11, result 11/2^0 = 11.
```

Details

In this problem, you have to cut pieces from a string, match them to a pattern, and maximize the sum of results.

Example: $o = 3$, $k = 5$, $S =$ `ookkoko`, $P =$ `ok?`.

```
ookkoko
ok?
 ok?
  ok?
   ok?
    ok?
```

`ok?`	miss 1, value $3 + 3 + 5 = 11$, result $11/2^1 = 5$.
` ok?`	miss 0, value $3 + 5 + 5 = 13$, result $13/2^0 = 13$.
`  ok?`	miss 1, value $5 + 5 + 3 = 13$, result $13/2^1 = 6$.
`   ok?`	miss 2, value $5 + 3 + 5 = 13$, result $13/2^2 = 3$.
`    ok?`	miss 0, value $3 + 5 + 3 = 11$, result $11/2^0 = 11$.

The best solution is to cut like this: $\texttt{o}\underbrace{\texttt{okk}}\underbrace{\texttt{oko}}$.
This gives a total score of $13 + 11 = 24$.

Convolution

For each starting position i for the pattern in string S, the number of mismatches is the sum of two values, $A_i + B_i$:

- A_i is the number of `k` in the pattern when there is `o` in the string.
- B_i is the number of `o` in the pattern when there is `k` in the string.

Convolution

For each starting position i for the pattern in string S, the number of mismatches is the sum of two values, $A_i + B_i$:

- A_i is the number of `k` in the pattern when there is `o` in the string.
- B_i is the number of `o` in the pattern when there is `k` in the string.

Let us calculate all numbers A_i (and then B_i similarly).

Convolution

For each starting position i for the pattern in string S, the number of mismatches is the sum of two values, $A_i + B_i$:

- A_i is the number of `k` in the pattern when there is `o` in the string.
- B_i is the number of `o` in the pattern when there is `k` in the string.

Let us calculate all numbers A_i (and then B_i similarly).

Consider a binary sequence "there is `o` in this position of S" and a binary sequence "there is `k` in this position of P".

Find their convolution using Fast Fourier Transform: reverse the latter, FFT both, multiply, inverse FFT the result.

The resulting values are the numbers A_i.

Dynamic Programming

Let $f(i)$ be the best result for string $S_1 S_2 \ldots S_i$.

Dynamic Programming

Let $f(i)$ be the best result for string $S_1 S_2 \ldots S_i$.

Then $f(i) = \max\{f(i-1), f(i-|P|) + v(i-|P|)\}$.
Here, $v(j)$ is the value obtained by matching the pattern starting at position j of the string.

Statement

Problem H: Pi Approximation

In this problem, you have to find the best approximation for π *obtained by calculating the number of Pythagorean triples from integers* $1, 2, \ldots, n$.

General Outline

In this problem, you have to find the best approximation for π obtained by calculating the number of Pythagorean triples from integers $1, 2, \ldots, n$.

Turns out we can just generate primitive Pythagorean triples fast enough.

General Outline

In this problem, you have to find the best approximation for π obtained by calculating the number of Pythagorean triples from integers $1, 2, \ldots, n$.

Turns out we can just generate primitive Pythagorean triples fast enough.

For example, there's Euclid's formula: for $0 < n < m$ where n and m are coprime and of different parity, $a = m^2 - n^2$, $b = 2mn$, and $c = m^2 + n^2$ are a unique primitive Pythagorean triple.

General Outline

In this problem, you have to find the best approximation for π obtained by calculating the number of Pythagorean triples from integers $1, 2, \ldots, n$.

Turns out we can just generate primitive Pythagorean triples fast enough.

For example, there's Euclid's formula: for $0 < n < m$ where n and m are coprime and of different parity, $a = m^2 - n^2$, $b = 2mn$, and $c = m^2 + n^2$ are a unique primitive Pythagorean triple.

It turns out that the number of such triples with each possible sum is small enough (as hinted by the example), so we can store these quantities in a straightforward way, in a 200MB array of bytes.

General Outline

In this problem, you have to find the best approximation for π obtained by calculating the number of Pythagorean triples from integers $1, 2, \ldots, n$.

Turns out we can just generate primitive Pythagorean triples fast enough.

For example, there's Euclid's formula: for $0 < n < m$ where n and m are coprime and of different parity, $a = m^2 - n^2$, $b = 2mn$, and $c = m^2 + n^2$ are a unique primitive Pythagorean triple.

It turns out that the number of such triples with each possible sum is small enough (as hinted by the example), so we can store these quantities in a straightforward way, in a 200MB array of bytes.

To speed things up, process all queries at once.

Statement

Problem I: Partition of Queries

In this problem, you have to decide when to rebuild a data structure so that the total time for processing the given queries is minimized.

Details

In this problem, you have to decide when to rebuild a data structure so that the total time for processing the given queries is minimized.

Example: queries are "++??+?", rebuild cost is $y = 5$.

Details

In this problem, you have to decide when to rebuild a data structure so that the total time for processing the given queries is minimized.

Example: queries are “++??+?”, rebuild cost is $y = 5$.

- Without interfering, we get the total cost $2 + 2 + 3 = 7$.

Details

In this problem, you have to decide when to rebuild a data structure so that the total time for processing the given queries is minimized.

Example: queries are "++??+?", rebuild cost is $y = 5$.

- Without interfering, we get the total cost $2 + 2 + 3 = 7$.
- We put a rebuild after two adds: "++|??+?", then the cost is $5 + 0 + 0 + 1 = 6$.

Dynamic Programming 1

Compute the prefix sums:
$p(i)$ is the number of "+" up to position i,
$r(i)$ is the number of "?" up to position i.
Positions are numbered from 1 to n, prefix sums from 0 to n.

Dynamic Programming 1

Compute the prefix sums:
$p(i)$ is the number of "+" up to position i,
$r(i)$ is the number of "?" up to position i.
Positions are numbered from 1 to n, prefix sums from 0 to n.

Let $c(i)$ be the cost to process the first i queries without rebuilds:
The $c(i) = c(i-1)$, plus $p(i)$ if the i-th query is a "?".

Dynamic Programming 1

Compute the prefix sums:
$p(i)$ is the number of "+" up to position i,
$r(i)$ is the number of "?" up to position i.
Positions are numbered from 1 to n, prefix sums from 0 to n.

Let $c(i)$ be the cost to process the first i queries without rebuilds:
The $c(i) = c(i-1)$, plus $p(i)$ if the i-th query is a "?".

Let $s(j, i)$ be the cost to process all queries from the segment from $j+1$ to i inclusive without rebuilds.
Then $s(j, i) = c(i) - c(j) - p(i) \cdot (r(j) - r(i))$. Indeed, we consider only the $(r(j) - r(i))$ "?" on the respective positions, and for each of them, we have to subtract $p(i)$ "+" before that position.

Dynamic Programming 2

Finally, let $f(i)$ be the minimum possible cost to process the first i queries:

- we either process all of them, or
- insert a rebuild after position $j < i$, take $f(j)$ into account, and process all queries from the segment from $j + 1$ to i inclusive without rebuilds.

Dynamic Programming 2

Finally, let $f(i)$ be the minimum possible cost to process the first i queries:

- we either process all of them, or
- insert a rebuild after position $j < i$, take $f(j)$ into account, and process all queries from the segment from $j + 1$ to i inclusive without rebuilds.

So, $f(i) = \min\{c(i), c(j) + s(j, i) + y \text{ for all } j < i\}$.

The answer we want is $f(n)$.

Dynamic Programming 3

Let us make some adjustments to the formula:
$f(i) = \min\{c(i), b(j,i) \text{ for all } j < i\}$, where $b(j,i) = c(j) + s(j,i) + y$.
Naturally,
$b(j,i) = c(j) + c(i) - c(j) - p(i) \cdot (r(j) - r(i)) = c(i) + p(i) \cdot r(i) - p(i) \cdot r(j)$.

Dynamic Programming 3

Let us make some adjustments to the formula:
$f(i) = \min\{c(i), b(j,i) \text{ for all } j < i\}$, where $b(j,i) = c(j) + s(j,i) + y$.
Naturally,
$b(j,i) = c(j) + c(i) - c(j) - p(i) \cdot (r(j) - r(i)) = c(i) + p(i) \cdot r(i) - p(i) \cdot r(j)$.

Now consider everything depending only on i as a separate function:
$g(i) = c(i) + p(i) \cdot r(i)$.
Then $g(i) - b(j,i)$ is just $p(i) \cdot r(j)$.

Note that $p(i)$ and $r(j)$ are nondecreasing sequences.
So it turns out we can use some "convex hull" optimization to compute the minimums efficiently: do a binary search for the respective j, or do two pointers, as when i increases, j can not decrease.

Statement

Problem J: Random Chess Game

In this problem, you have to win 150 *chess games against an opponent who picks random moves, and all possible moves are listed for you each time you have to make a move.*

Details

In this problem, you have to win 150 *chess games against an opponent who picks random moves, and all possible moves are listed for you each time you have to make a move.*

- Implementing all chess rules from scratch is very hard to do in a contest.

Details

In this problem, you have to win 150 *chess games against an opponent who picks random moves, and all possible moves are listed for you each time you have to make a move.*

- Implementing all chess rules from scratch is very hard to do in a contest.
- Thus a more realistic approach is to use the provided move generator, consider only a part of game state, and use heuristics for the rest.

Details

In this problem, you have to win 150 *chess games against an opponent who picks random moves, and all possible moves are listed for you each time you have to make a move.*

- Implementing all chess rules from scratch is very hard to do in a contest.
- Thus a more realistic approach is to use the provided move generator, consider only a part of game state, and use heuristics for the rest.
- There are two sources for heuristics:
  - We know chess rules.
  - We know the opponent picks moves uniformly at random.

Details

In this problem, you have to win 150 *chess games against an opponent who picks random moves, and all possible moves are listed for you each time you have to make a move.*

- Implementing all chess rules from scratch is very hard to do in a contest.
- Thus a more realistic approach is to use the provided move generator, consider only a part of game state, and use heuristics for the rest.
- There are two sources for heuristics:
  - We know chess rules.
  - We know the opponent picks moves uniformly at random.
- Be ready to iterate, and likely not have your solution accepted on the first try.

Heuristics

- If there is a checkmate move, do it!

Heuristics

- If there is a checkmate move, do it!
- We strive to take Black's pieces...

Heuristics

- If there is a checkmate move, do it!
- We strive to take Black's pieces... but we don't like to put the Black King in check: too little responses, too high the probability of a good response.

Heuristics

- If there is a checkmate move, do it!
- We strive to take Black's pieces... but we don't like to put the Black King in check: too little responses, too high the probability of a good response.
- Prefer moving pawns to moving heavier pieces...

Heuristics

- If there is a checkmate move, do it!
- We strive to take Black's pieces... but we don't like to put the Black King in check: too little responses, too high the probability of a good response.
- Prefer moving pawns to moving heavier pieces... but avoid moving pawns adjacent to the White King, just to keep the King safe.

Heuristics

- If there is a checkmate move, do it!
- We strive to take Black's pieces... but we don't like to put the Black King in check: too little responses, too high the probability of a good response.
- Prefer moving pawns to moving heavier pieces... but avoid moving pawns adjacent to the White King, just to keep the King safe.
- Strive for promoting your pawns.

Heuristics

- If there is a checkmate move, do it!
- We strive to take Black's pieces... but we don't like to put the Black King in check: too little responses, too high the probability of a good response.
- Prefer moving pawns to moving heavier pieces... but avoid moving pawns adjacent to the White King, just to keep the King safe.
- Strive for promoting your pawns.
- Avoid placing heavy pieces adjacent to the Black King.

Heuristics

- If there is a checkmate move, do it!
- We strive to take Black's pieces... but we don't like to put the Black King in check: too little responses, too high the probability of a good response.
- Prefer moving pawns to moving heavier pieces... but avoid moving pawns adjacent to the White King, just to keep the King safe.
- Strive for promoting your pawns.
- Avoid placing heavy pieces adjacent to the Black King.
- Avoid repeated positions, which can be indirectly tracked as their sets of possible moves.

Heuristics

- If there is a checkmate move, do it!
- We strive to take Black's pieces… but we don't like to put the Black King in check: too little responses, too high the probability of a good response.
- Prefer moving pawns to moving heavier pieces… but avoid moving pawns adjacent to the White King, just to keep the King safe.
- Strive for promoting your pawns.
- Avoid placing heavy pieces adjacent to the Black King.
- Avoid repeated positions, which can be indirectly tracked as their sets of possible moves.
- In the endgame, minimize distance between Kings to avoid draws by 50-move rule…

Heuristics

- If there is a checkmate move, do it!
- We strive to take Black's pieces... but we don't like to put the Black King in check: too little responses, too high the probability of a good response.
- Prefer moving pawns to moving heavier pieces... but avoid moving pawns adjacent to the White King, just to keep the King safe.
- Strive for promoting your pawns.
- Avoid placing heavy pieces adjacent to the Black King.
- Avoid repeated positions, which can be indirectly tracked as their sets of possible moves.
- In the endgame, minimize distance between Kings to avoid draws by 50-move rule... but be careful not to produce a stalemate.

Scholar's Mate

- Try Scholar's Mate first!

Scholar's Mate

- Try Scholar's Mate first!
- For example: `1.e4, 2.Nf3, 3.Bc4, 4.Ne5, 5.Qf3`.

Scholar's Mate

- Try Scholar's Mate first!
- For example: `1.e4, 2.Nf3, 3.Bc4, 4.Ne5, 5.Qf3`.
- If this does not work, consider moving the Queen back, then proceed normally with heuristics.

Statement

Problem K: What? Subtasks? Again?

In this problem, you have to pick a subset of contest features so that the number of satisfied contestants is the maximum possible integer $\leqslant m$.

Details

In this problem, you have to pick a subset of contest features so that the number of satisfied contestants is the maximum possible integer $\leqslant m$.

There are exactly five features!

Details

In this problem, you have to pick a subset of contest features so that the number of satisfied contestants is the maximum possible integer $\leqslant m$.

There are exactly five features!

For each contestant, compute the mask of features they don't like.

Details

In this problem, you have to pick a subset of contest features so that the number of satisfied contestants is the maximum possible integer $\leqslant m$.

There are exactly five features!

For each contestant, compute the mask of features they don't like.

For each of the $2^5 = 32$ possible sets of features, count the contestants satisfied by this set.

Details

In this problem, you have to pick a subset of contest features so that the number of satisfied contestants is the maximum possible integer $\leqslant m$.

There are exactly five features!

For each contestant, compute the mask of features they don't like.

For each of the $2^5 = 32$ possible sets of features, count the contestants satisfied by this set.

Pick the greatest number of contestants $\leqslant m$.

Details

In this problem, you have to pick a subset of contest features so that the number of satisfied contestants is the maximum possible integer $\leqslant m$.

There are exactly five features!

For each contestant, compute the mask of features they don't like.

For each of the $2^5 = 32$ possible sets of features, count the contestants satisfied by this set.

Pick the greatest number of contestants $\leqslant m$.

If there is none, print the required line instead of the number.

Statement

Problem L: The Five Bishops

In this problem, you have to checkmate or stalemate a lone black King with five white bishops on an infinite chessboard.

Details

In this problem, you have to checkmate or stalemate a lone black King with five white bishops on an infinite chessboard.

- If we don't have two bishops of each color (two on light squares and two on dark squares), we can't do it.

Details

In this problem, you have to checkmate or stalemate a lone black King with five white bishops on an infinite chessboard.

- If we don't have two bishops of each color (two on light squares and two on dark squares), we can't do it.
- Four bishops are sufficient.

Getting Far

Step 1: Get far from the King.

Getting Far

Step 1: Get far from the King.

- Be careful not to lose any important bishop.

Getting Far

Step 1: Get far from the King.

- Be careful not to lose any important bishop.
- The King can't attack both colors simultaneously.

Getting Far

Step 1: Get far from the King.

- Be careful not to lose any important bishop.
- The King can't attack both colors simultaneously.
- Defend one bishop of the attacked color with another.

Getting Far

Step 1: Get far from the King.

- Be careful not to lose any important bishop.
- The King can't attack both colors simultaneously.
- Defend one bishop of the attacked color with another.
- We can lose the third bishop of some color, or just move it away.

Getting Far

Step 1: Get far from the King.

- Be careful not to lose any important bishop.
- The King can't attack both colors simultaneously.
- Defend one bishop of the attacked color with another.
- We can lose the third bishop of some color, or just move it away.
- Bearing that in mind, move the bishops away (say, 10^6 squares) from the King.

Controlling the Diagonals

Step 2: Bound the King to one diagonal line.

Controlling the Diagonals

Step 2: Bound the King to one diagonal line.

- Consider one of the two types diagonals of the board.

Controlling the Diagonals

Step 2: Bound the King to one diagonal line.

- Consider one of the two types diagonals of the board.
- Enumerate them by integers, for example, $c = x - y$.

Controlling the Diagonals

Step 2: Bound the King to one diagonal line.

- Consider one of the two types diagonals of the board.
- Enumerate them by integers, for example, $c = x - y$.
- Let the King be on diagonal k.

Controlling the Diagonals

Step 2: Bound the King to one diagonal line.

- Consider one of the two types diagonals of the board.
- Enumerate them by integers, for example, $c = x - y$.
- Let the King be on diagonal k.
- Then place the bishops on diagonals $k - 10$, $k - 9$, $k + 9$, and $k + 10$.
- As a result, the King can't get to diagonals $k + x$ where $|x| \geqslant 9$.

Controlling the Diagonals

Step 2: Bound the King to one diagonal line.

- Consider one of the two types diagonals of the board.
- Enumerate them by integers, for example, $c = x - y$.
- Let the King be on diagonal k.
- Then place the bishops on diagonals $k - 10$, $k - 9$, $k + 9$, and $k + 10$.
- As a result, the King can't get to diagonals $k + x$ where $|x| \geqslant 9$.
- As long as there is more than one diagonal left for the King, pick the bishop on the diagonal farthest from the King, and move it one step closer.

Controlling the Diagonals

Step 2: Bound the King to one diagonal line.

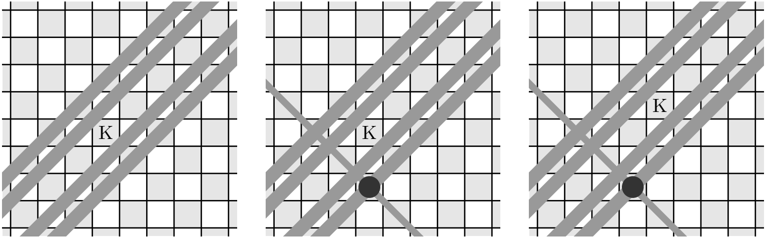
- Consider one of the two types diagonals of the board.
- Enumerate them by integers, for example, $c = x - y$.
- Let the King be on diagonal k.
- Then place the bishops on diagonals $k - 10$, $k - 9$, $k + 9$, and $k + 10$.
- As a result, the King can't get to diagonals $k + x$ where $|x| \geqslant 9$.
- As long as there is more than one diagonal left for the King, pick the bishop on the diagonal farthest from the King, and move it one step closer.
- The King won't be attacked by this move, but the number of diagonals he can visit will decrease by one.

Stalemate

Step 3: Produce a stalemate.

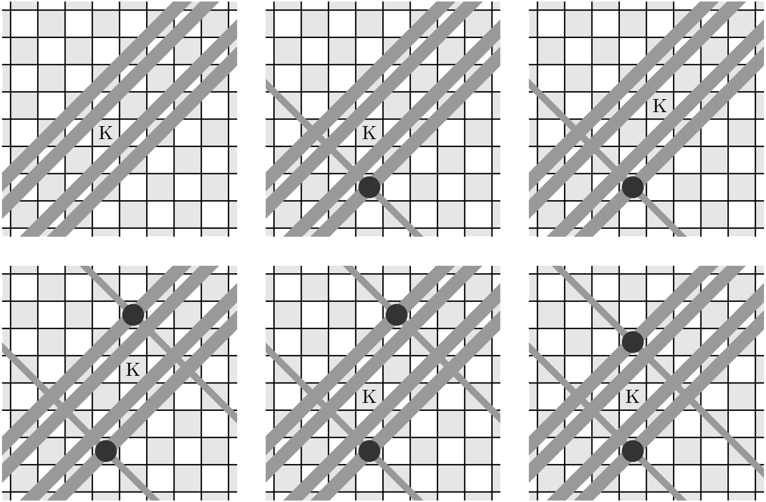
Stalemate

Step 3: Produce a stalemate.



Stalemate

Step 3: Produce a stalemate.



Based on LV St. Petersburg State University Championship

Contest Developers:

- Ivan Kazmenko
- Aleksandr Logunov
- Andrei Lopatin
- Anton Maydell
- Artur Riazanov

Questions?