Task overview

Central Europe Regional Contest 2025

December 7, 2025

Contest statistics

Contest statistics

Number of submissions: 1135

Contest statistics

Number of submissions: 1135
Programming languages:

- C++: 1103
- Python3: 31
- Java: 1
- Kotlin: 0

Contest statistics

Number of submissions: 1135

Programming languages:

- C++: 1103
- Python3: 31
- Java: 1
- Kotlin: 0

Total test runs: 18808

Contest statistics

Number of submissions: 1135
Programming languages:

- C++: 1103
- Python3: 31
- Java: 1
- Kotlin: 0

Total test runs: 18808
Total time running tests: 33 minutes

Contest statistics

Number of submissions: 1135
Programming languages:

- C++: 1103
- Python3: 31
- Java: 1
- Kotlin: 0

Total test runs: 18808
Total time running tests: 33 minutes
Total submission size: 2.11 megabytes

Contest statistics

Number of submissions: 1135
Programming languages:

- C++: 1103
- Python3: 31
- Java: 1
- Kotlin: 0

Total test runs: 18808
Total time running tests: 33 minutes
Total submission size: 2.11 megabytes
Smallest AC source size: 387 bytes, Key Properties

Contest statistics

Number of submissions: 1135
Programming languages:

- C++: 1103
- Python3: 31
- Java: 1
- Kotlin: 0

Total test runs: 18808
Total time running tests: 33 minutes
Total submission size: 2.11 megabytes
Smallest AC source size: 387 bytes, Key Properties
Biggest AC source size: 15.79 kilobytes, Identical Fences

Contest statistics

Number of submissions: 1135
Programming languages:

- C++: 1103
- Python3: 31
- Java: 1
- Kotlin: 0

Total test runs: 18808
Total time running tests: 33 minutes
Total submission size: 2.11 megabytes
Smallest AC source size: 387 bytes, Key Properties
Biggest AC source size: 15.79 kilobytes, Identical Fences
Most submissions on a single task without ACC: 24, Key Properties

Problem L – Letters on T-shirts

Problem L – Letters on T-shirts

Accepted before freeze: 70

Teams that submitted after freeze: Not needed :)

Problem L – Letters on T-shirts

Problem

Given three short strings.

Check if it is possible to get string `cerc` by concatenation of some of them.

Problem L – Letters on T-shirts

Problem

Given three short strings.
Check if it is possible to get string `cerc` by concatenation of some of them.

Possible solutions:

- Check all 3! permutations and try to concatenate first one, two or three strings.
- String `cerc` has only 7 valid decompositions:
  `cerc`, `c|erc`, `ce|rc`, `cer|c`, `c|e|rc`, `c|er|c` and `ce|r|c`.

Problem G – Game of Darts

Problem G – Game of Darts

Accepted before freeze: 70

Teams that submitted after freeze: Not needed :)

Problem G – Game of Darts

Problem

We need to find a way of winning a game of darts in at most 3 throws. In a single throw we can score either 25, 50, a, $2a$ or $3a$ points for $a \in \{1 \ldots 20\}$. In order to win, we need to score exactly P points, but the last throw can only be for 50 or $2a$ points.

Problem G – Game of Darts

Solution 1. – Knapsack Problem

This task is an instance of a knapsack problem, so it can be solved with classic dynamic program. To retrieve exact throws needed to win, in every state of the dynamic program we need to save the number of points scored in the last throw so far.
Complexity: $\mathcal{O}(NP)$

Here N is the number of possible scores for a single throw.

Problem G – Game of Darts

Solution 1. – Knapsack Problem

This task is an instance of a knapsack problem, so it can be solved with classic dynamic program. To retrieve exact throws needed to win, in every state of the dynamic program we need to save the number of points scored in the last throw so far.
Complexity: $\mathcal{O}(NP)$

Solution 2. – Exhaustive Search

Because the number of possible scores is small (less than 200) and there are at most 3 throws, checking all the combinations also fits in the time limit.
Complexity: $\mathcal{O}(N^3)$

Here N is the number of possible scores for a single throw.

Problem D – DNA

Problem D – DNA

Accepted before freeze: 63

Teams that submitted after freeze: 4

Problem D – DNA

Problem

Given three strings consisting of digits `0` and `1`.
Find such value x, that their Longest Common Subsequence lies in interval $[x, 2 \cdot x]$ (i.e. find 2-approximation of their LCS).

Problem D – DNA

Problem

Given three strings consisting of digits 0 and 1.
Find such value x, that their Longest Common Subsequence lies in interval $[x, 2 \cdot x]$ (i.e. find 2-approximation of their LCS).

Let $a_0, a_1, b_0, b_1, c_0, c_1$ be the numbers of digits 0 and 1 in these strings.

Lets observe that:

- String of $\min\{a_0, b_0, c_0\}$ digits 0 is a proper common substring.
- The same situation holds for digits 1.
- LCS has length at most $\min\{a_0, b_0, c_0\} + \min\{a_1, b_1, c_1\}$.

Therefore good solutions are:

- $\max\{\min\{a_0, b_0, c_0\}, \min\{a_1, b_1, c_1\}\}$
- $\lceil(\min\{a_0, b_0, c_0\} + \min\{a_1, b_1, c_1\})/2\rceil$

Problem K – Key Properties

Problem K – Key Properties

Accepted before freeze: 56

Teams that submitted after freeze: 9

Problem K – Key Properties

Problem

Given a number $N \geq 42$ produce a graph with N vertices that:

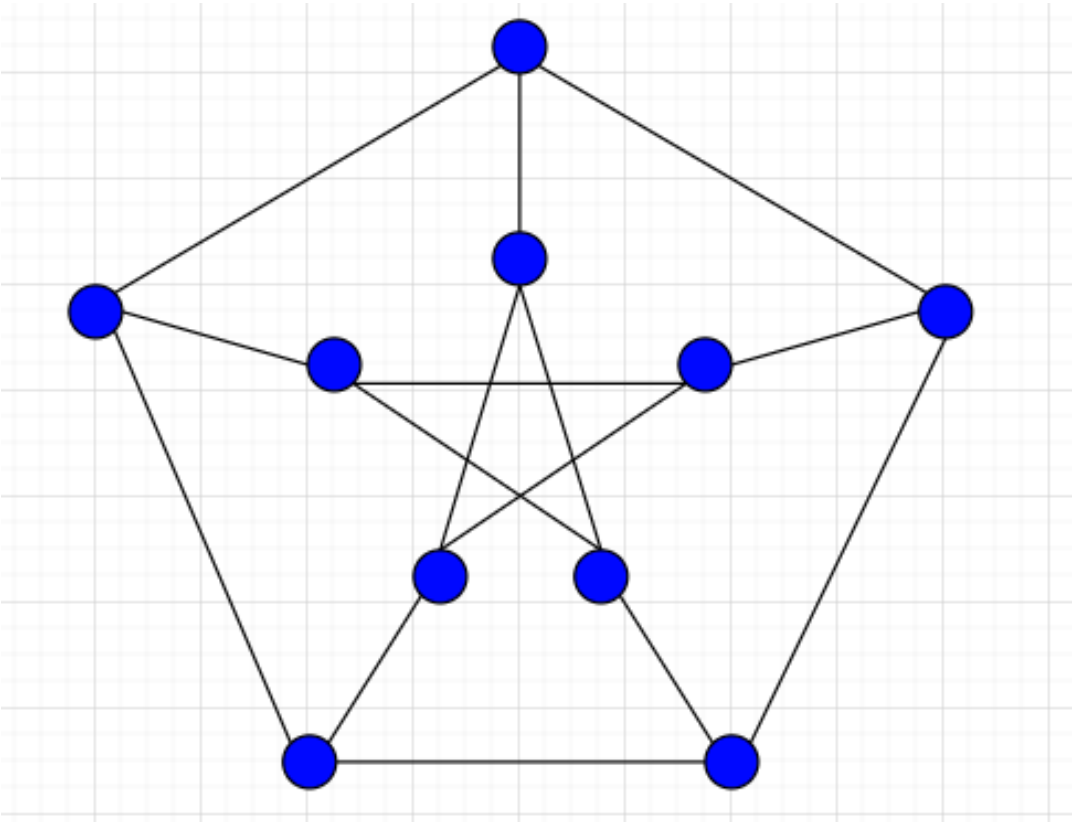
- is connected,
- is 3-regular, meaning each vertex has degree 3,
- contains no cycles of length 4.

Problem K – Key Properties

Solution 1:
If N is odd, the answer is NO (sum od degrees would have to be odd), otherwise the solution exists.

Problem K – Key Properties

Solution 1:
If N is odd, the answer is NO (sum od degrees would have to be odd), otherwise the solution exists.
If N is a multiple of 10, take the graph from sample...

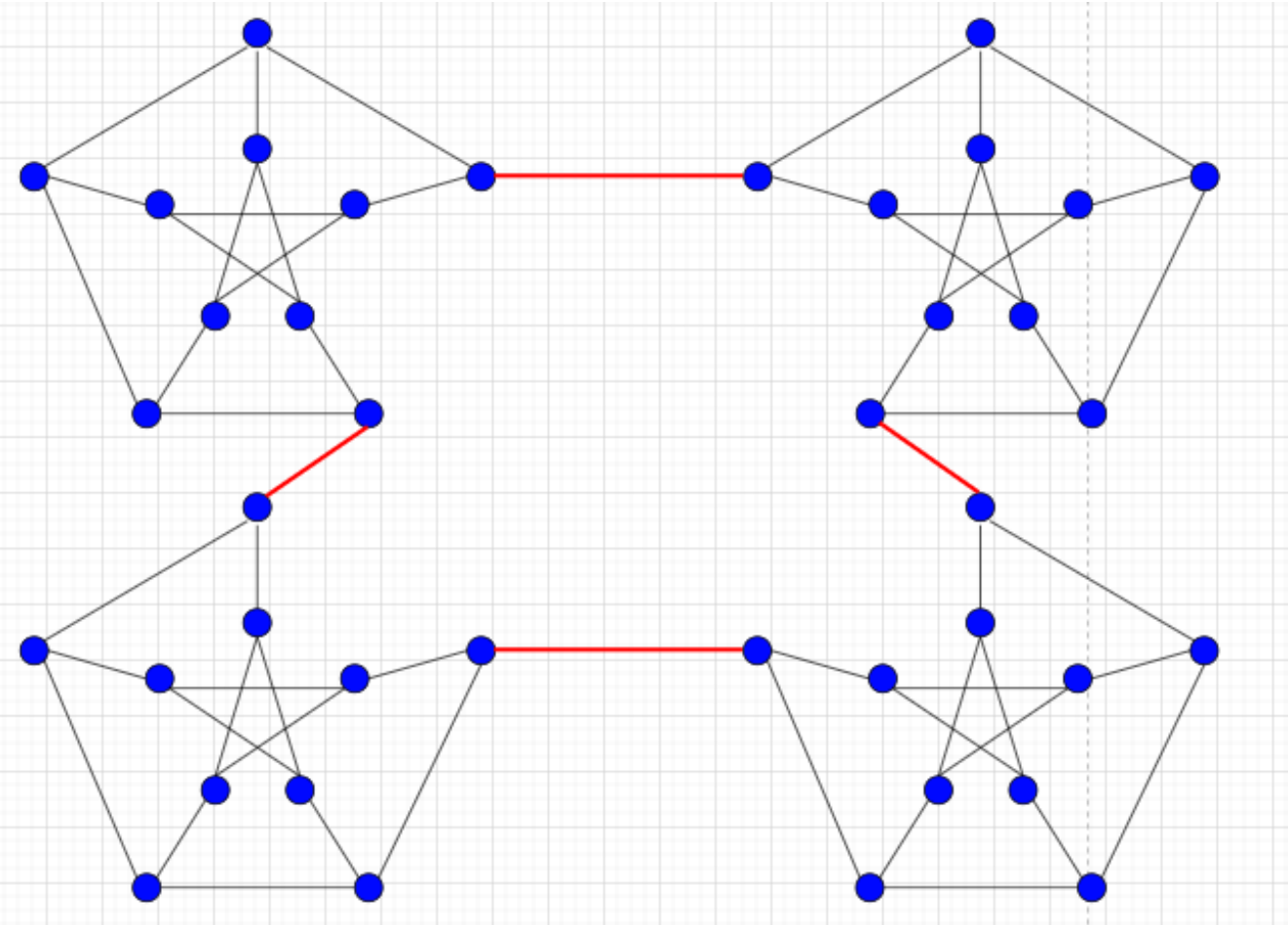


Problem K – Key Properties

Solution 1:

If N is a multiple of 10, take the graph from sample...

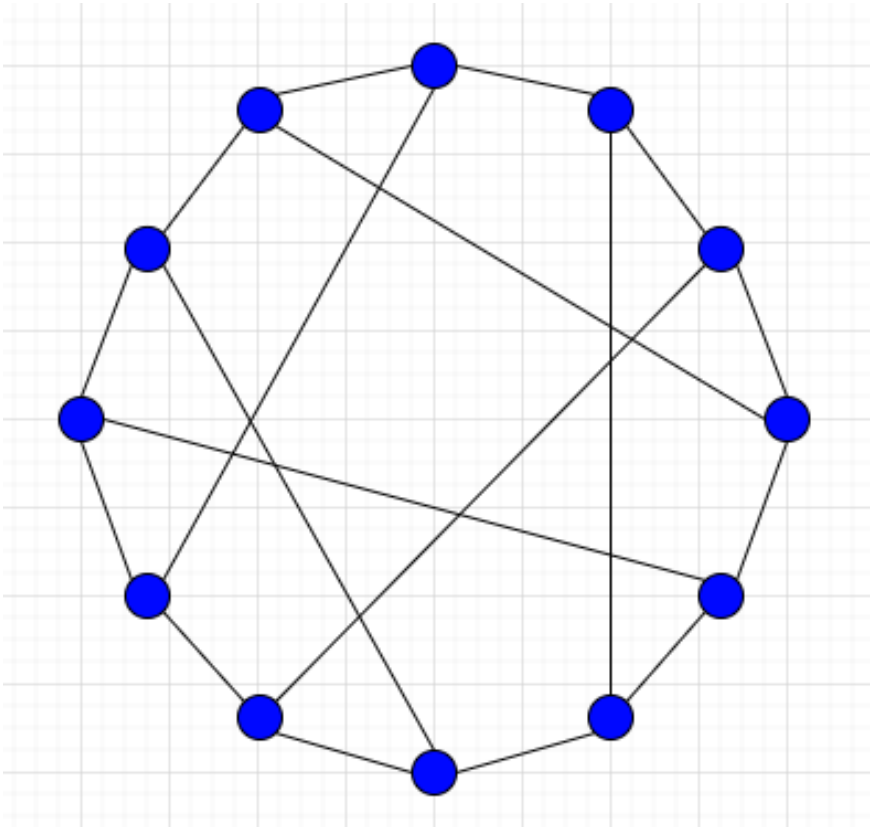
...and connect many copies of it.



Problem K – Key Properties

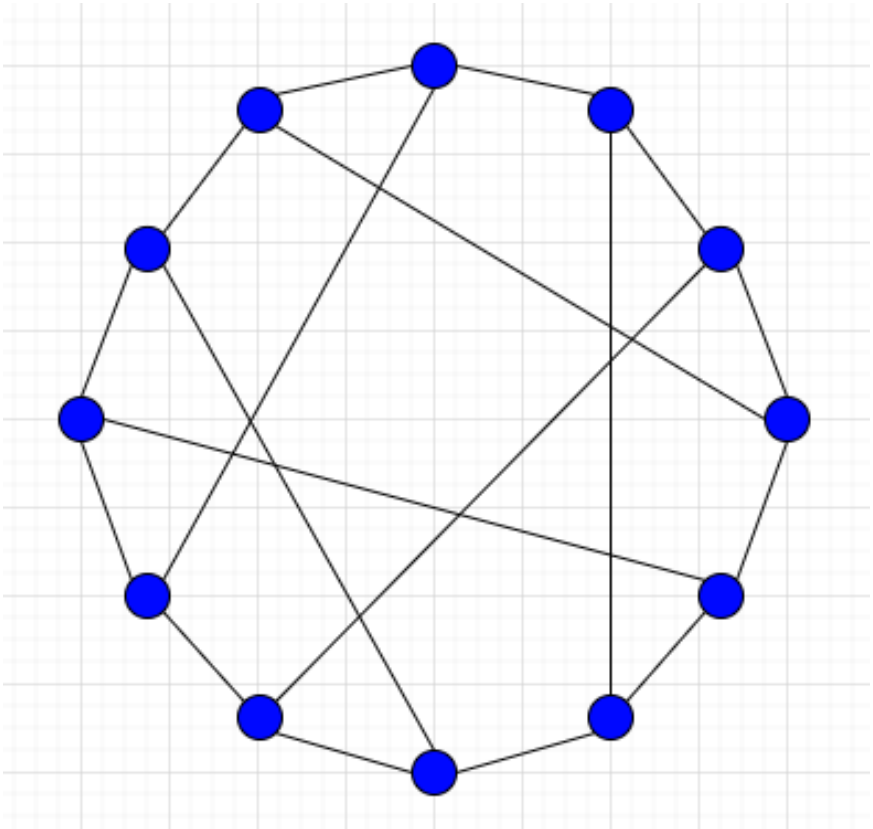
Solution 1:

A graph satisfying our properties for $N = 12$ can be easily found via brutal search.



Problem K – Key Properties

Solution 1:
A graph satisfying our properties for $N = 12$ can be easily found via brutal search.

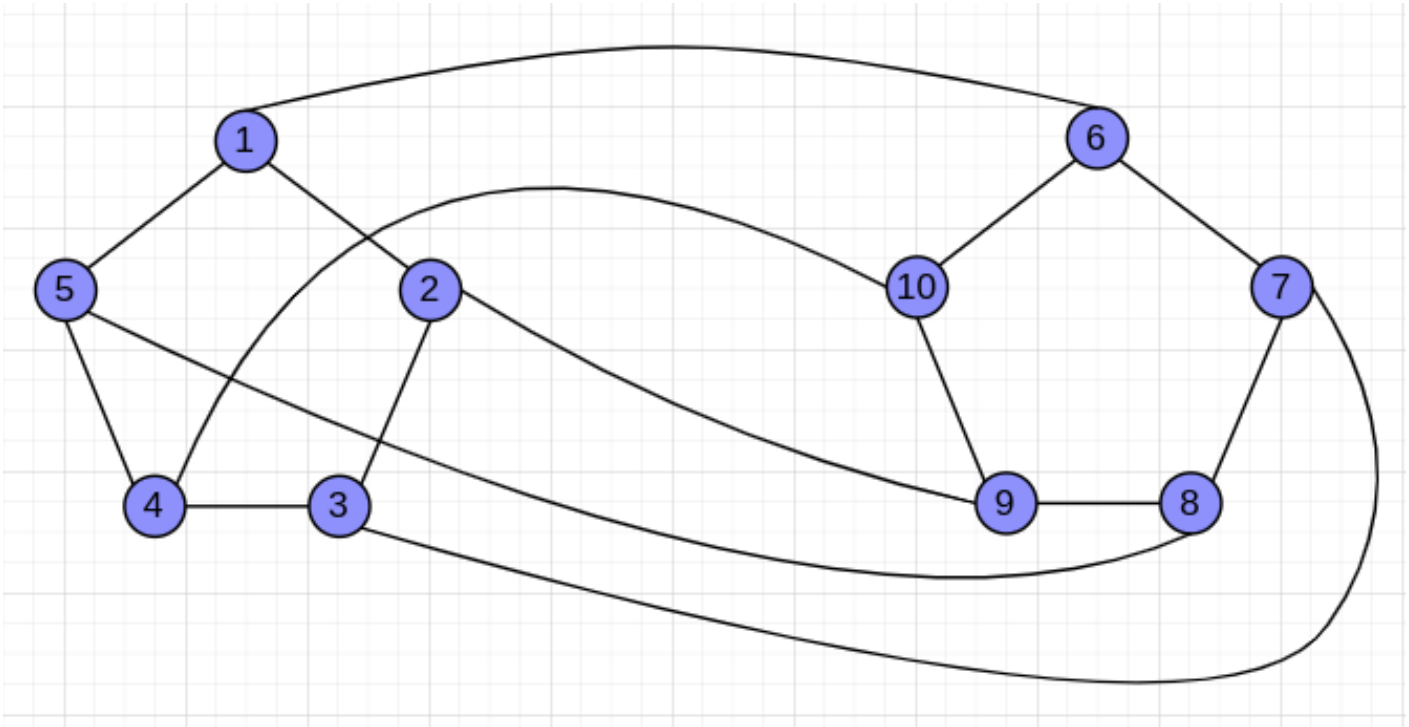


For any $N \geq 42$ a solution can be composed of graphs with 10 and 12 vertices.

Problem K – Key Properties

Solution 2:

- Create two cycles, one on vertices $\{1, \ldots, N/2\}$, and the other on $\{N/2+1, \ldots, N\}$.
- Find a number p coprime with $N/2$.
- Connect 1 with $N/2+1$ and then each subsequent $i \leq N/2$ with the vertex on the other cycle shifted by p compared to the previous vertex.



Problem I – Identical Fences

Problem I – Identical Fences

Accepted before freeze: 55

Teams that submitted after freeze: 5

Problem I – Identical Fences

Problem

Given a random sequence of 0 and 1 of length 100 000.
Goal is to find two disjoint subsequences that forms the same string.
At least 84% of elements need to be used.

Example:

```
0110101001011
.....01.010.1
0.101..0...1.
```

Problem I – Identical Fences

Technique 1:

- Divide input into blocks of size at least 9.
- Find optimal division into two identical strings for every block.

Exp. result: 86%

Technique 2:

- Check if prefix is one of the following:
  - 00
  - 0101
  - 01000
  - 01001
  - 01100
  - 01101
  - 01110
  - 01111
  - analogously with 1 in the beginning.
- Find optimal solution for this prefix and go on.

Exp. result: 85%

Problem I – Identical Fences

Technique 3:

- Check if prefix is a *square word* (a word copied two times) of length 2, 4, 6, 8 or 10.
- If not, skip one letter.

Exp. result: 89.5%

Technique 4:

- Keep two strings, not necessary of the same length (first one will possibly be longer).
- If their lengths are equal, put next letter into first string.
- If next letter is equal to corresponding letter in first string, put it into the second string.
- Otherwise put it into first string.

Exp. result: 99.9% ($n - O(\sqrt{n})$, the result of one dimensional random walk)

Problem H – Hiking

Problem H – Hiking

Accepted before freeze: 45

Teams that submitted after freeze: 19

Problem H – Hiking

Problem

Given a sequence of natural numbers $a_1, a_2, \ldots, a_N$ find their ordering $b_1, b_2, \ldots, b_N$ that minimizes $\sum_{i=1}^{N-1} \lfloor b_i / b_{i+1} \rfloor$. It is guaranteed that all given numbers belong to a set $\{k, \ldots, 2k\}$ for some k.

Problem H – Hiking

Observation

There are only three possible values of $\lfloor b_i / b_{i+1} \rfloor$:

- zero: 6 4 3 4 4 6 3 5 4 3 3 4
- one: 6 4 3 4 4 6 3 5 4 3 3 4
- two: 6 4 3 4 4 6 3 5 4 3 3 4

Problem H – Hiking

Solution

Calculate the number *occ* of occurrences of the most popular number.
Divide the sequence into *occ* strictly increasing subsequences.

3 4 5 6 3 4 6 3 4 3 4 4

Problem H – Hiking

Solution

Calculate the number *occ* of occurrences of the most popular number. Divide the sequence into *occ* strictly increasing subsequences.

3 4 5 6 3 4 6 3 4 3 4 4

The maximum number should be put in the latest sequences possible, and the minimum number in the earliest possible.

3 4 5 3 4 3 4 3 4 6 4 6

Problem H – Hiking

Solution

Calculate the number *occ* of occurrences of the most popular number. Divide the sequence into *occ* strictly increasing subsequences.

$$3\ 4\ 5\ 6 \quad 3\ 4\ 6 \quad 3\ 4 \quad 3\ 4 \quad 4$$

The maximum number should be put in the latest sequences possible, and the minimum number in the earliest possible.

$$3\ 4\ 5 \quad 3\ 4 \quad 3\ 4 \quad 3\ 4\ 6 \quad 4\ 6$$

Let min_{occ} = number of occurrences of k, max_{occ} = number of occurrences of $2k$, occ = number of occurrences of the most popular number. Then this algorithm has total cost

$$max(min_{occ} + max_{occ} - 2, \quad occ - 1).$$

It is easy to verify that this cost is optimal.

Problem A – Automatized Mineral Classification

Problem A – Automatized Mineral Classification

Accepted before freeze: 19

Teams that submitted after freeze: 22

Problem A – Automatized Mineral Classification

Statement

There is a hidden sequence a_i – the color of i of length n and a queue(initially empty). We can do the following two operations:

Problem A – Automatized Mineral Classification

Statement

There is a hidden sequence a_i – the color of i of length n and a queue(initially empty). We can do the following two operations:

- push a chosen element from the sequence to the end of the queue

Problem A – Automatized Mineral Classification

Statement

There is a hidden sequence a_i – the color of i of length n and a queue(initially empty). We can do the following two operations:

- push a chosen element from the sequence to the end of the queue
- pop the element from the beginning of the queue

Problem A – Automatized Mineral Classification

Statement

There is a hidden sequence a_i – the color of i of length n and a queue(initially empty). We can do the following two operations:

- push a chosen element from the sequence to the end of the queue
- pop the element from the beginning of the queue

After each operation, we learn how many different elements are currently in the queue.

Problem A – Automatized Mineral Classification

Statement

There is a hidden sequence a_i – the color of i of length n and a queue(initially empty). We can do the following two operations:

- push a chosen element from the sequence to the end of the queue
- pop the element from the beginning of the queue

After each operation, we learn how many different elements are currently in the queue.

The aim is to recover it up to equivalence. More precisely: we have to find any sequence b_i such that for any pair of indices $1 \leq i, j \leq n$ it holds that $a_i = a_j \iff b_i = b_j$.

We can use subquadratically many operations.

Problem A – Automatized Mineral Classification

Statement

There is a hidden sequence a_i – the color of i of length n and a queue(initially empty). We can do the following two operations:

- push a chosen element from the sequence to the end of the queue
- pop the element from the beginning of the queue

After each operation, we learn how many different elements are currently in the queue.

The aim is to recover it up to equivalence. More precisely: we have to find any sequence b_i such that for any pair of indices $1 \leq i, j \leq n$ it holds that $a_i = a_j \iff b_i = b_j$.

We can use subquadratically many operations.

There are many alternative solutions with complexities varying from $\mathcal{O}(n \log n)$ through $\mathcal{O}(n \log^2 n)$ to $\mathcal{O}(n\sqrt{n})$.

Problem A – Automatized Mineral Classification

Randomized $\mathcal{O}(n\sqrt{n})$ solution

Add the elements of the sequence a to the queue one by one until the number of different elements in the queue is different than its size.

Problem A – Automatized Mineral Classification

Randomized $\mathcal{O}(n\sqrt{n})$ solution

Add the elements of the sequence a to the queue one by one until the number of different elements in the queue is different than its size. Then, the last added element x is equal to some other element in the queue y. While emptying the queue, we can find x, y.

Problem A – Automatized Mineral Classification

Randomized $\mathcal{O}(n\sqrt{n})$ solution

Add the elements of the sequence a to the queue one by one until the number of different elements in the queue is different than its size. Then, the last added element x is equal to some other element in the queue y. While emptying the queue, we can find x, y. Then we can delete x from a and repeat the above steps.

Problem A – Automatized Mineral Classification

Randomized $\mathcal{O}(n\sqrt{n})$ solution

Add the elements of the sequence a to the queue one by one until the number of different elements in the queue is different than its size. Then, the last added element x is equal to some other element in the queue y. While emptying the queue, we can find x, y. Then we can delete x from a and repeat the above steps.

We have to argue that the expected number of queries is low if the order of elements is random.

Problem A – Automatized Mineral Classification

Claim (main step of proof)

Fix $c < 1$. Assume the order of elements is random and the number of different colors is at most $c \cdot n$. Then the expected number of operations in one iteration of the procedure is $\mathcal{O}(\sqrt{n})$.

Problem A – Automatized Mineral Classification

Claim (main step of proof)

Fix $c < 1$. Assume the order of elements is random and the number of different colors is at most $c \cdot n$. Then the expected number of operations in one iteration of the procedure is $\mathcal{O}(\sqrt{n})$.

It is related to the following similar problem:

Problem A – Automatized Mineral Classification

Claim (main step of proof)

Fix $c < 1$. Assume the order of elements is random and the number of different colors is at most $c \cdot n$. Then the expected number of operations in one iteration of the procedure is $\mathcal{O}(\sqrt{n})$.

It is related to the following similar problem:

Birthday paradox

The probability that among a group of 23 random people, at least two share the same birthday is around $\frac{1}{2}$.

Problem A – Automatized Mineral Classification

Claim (main step of proof)

Fix $c < 1$. Assume the order of elements is random and the number of different colors is at most $c \cdot n$. Then the expected number of operations in one iteration of the procedure is $\mathcal{O}(\sqrt{n})$.

It is related to the following similar problem:

Birthday paradox

The probability that among a group of 23 random people, at least two share the same birthday is around $\frac{1}{2}$.

Birthday paradox – our case

If we have n elements of $k \leq \frac{n}{2}$ colors, then the sample of $\sqrt{k}$ elements has a constant probability of containing a duplicated color. This holds even if the distribution is biased.

Problem A – Automatized Mineral Classification

Claim (main step of proof)

Fix $c < 1$. Assume the order of elements is random and the number of different colors is at most $c \cdot n$. Then the expected number of operations in one iteration of the procedure is $\mathcal{O}(\sqrt{n})$.

It is related to the following similar problem:

Birthday paradox

The probability that among a group of 23 random people, at least two share the same birthday is around $\frac{1}{2}$.

Birthday paradox – our case

If we have n elements of $k \leq \frac{n}{2}$ colors, then the sample of $\sqrt{k}$ elements has a constant probability of containing a duplicated color. This holds even if the distribution is biased.

To prove the claim, a bit of calculus is necessary.

Problem A – Automatized Mineral Classification

$\mathcal{O}(n \log n)$ solution

We can find one representative for each color. This can be done by just adding all the elements to the queue, one by one.

Problem A – Automatized Mineral Classification

$\mathcal{O}(n \log n)$ solution

We can find one representative for each color. This can be done by just adding all the elements to the queue, one by one.
We have to determine, for each element, what is its representative.

Problem A – Automatized Mineral Classification

$\mathcal{O}(n \log n)$ solution

We can find one representative for each color. This can be done by just adding all the elements to the queue, one by one.
We have to determine, for each element, what is its representative.

Divide and conquer

Consider a recursive function which takes a subset S of elements and a subset R of representatives and returns a representative for each element.

Problem A – Automatized Mineral Classification

$\mathcal{O}(n \log n)$ solution

We can find one representative for each color. This can be done by just adding all the elements to the queue, one by one.
We have to determine, for each element, what is its representative.

Divide and conquer

Consider a recursive function which takes a subset S of elements and a subset R of representatives and returns a representative for each element.

We can divide elements of S into two halves H_1, H_2. Then, for each half H_i calculate what is the set of representatives of its elements. This can be done by first putting the whole half on the queue and then inserting each representative from R. Representatives which do not increase the number of colors are the ones we are looking for.

Problem A – Automatized Mineral Classification

$\mathcal{O}(n \log n)$ solution

We can find one representative for each color. This can be done by just adding all the elements to the queue, one by one.
We have to determine, for each element, what is its representative.

Divide and conquer

Consider a recursive function which takes a subset S of elements and a subset R of representatives and returns a representative for each element.

We can divide elements of S into two halves H_1, H_2. Then, for each half H_i calculate what is the set of representatives of its elements. This can be done by first putting the whole half on the queue and then inserting each representative from R. Representatives which do not increase the number of colors are the ones we are looking for.
We then recurse on the two halves.

Problem C – Connecting Railway Stations

Problem C – Connecting Railway Stations

Accepted before freeze: 9

Teams that submitted after freeze: 17

Problem C – Connecting Railway Stations

Statement

We are given a tree with an even number of leaves and a sequence a_e written on its edges. Consider a pairing of leaves. Each pair induces a path between them. For each edge, we denote the number of induced paths that contain this edge by b_e.

The coloring is *valid* if $b_e \leq a_e$ for each edge.

We have to find the maximal value of $\sum b_e$ attained by a valid pairing.

Problem C – Connecting Railway Stations

First, let us characterize when the sequence b_e results from a pairing:

Problem C – Connecting Railway Stations

First, let us characterize when the sequence b_e results from a pairing:

1. Each edge to a leaf has $b_e = 1$.

Problem C – Connecting Railway Stations

First, let us characterize when the sequence b_e results from a pairing:

1. Each edge to a leaf has $b_e = 1$.
2. For every internal vertex v, the sum of b_e over the edges incident to v is even.

Problem C – Connecting Railway Stations

First, let us characterize when the sequence b_e results from a pairing:

1. Each edge to a leaf has $b_e = 1$.
2. For every internal vertex v, the sum of b_e over the edges incident to v is even.
3. For every vertex v, the values b_e of the edges incident to v satisfy the triangle inequality: $\sum_{e \ni v} b_e \geq 2 \max(b_e)$.

Problem C – Connecting Railway Stations

First, let us characterize when the sequence b_e results from a pairing:

1. Each edge to a leaf has $b_e = 1$.
2. For every internal vertex v, the sum of b_e over the edges incident to v is even.
3. For every vertex v, the values b_e of the edges incident to v satisfy the triangle inequality: $\sum_{e \ni v} b_e \geq 2 \max(b_e)$.

The second condition is equivalent to a different one: that parity of b_e is the same as the parity of the number of leaves in the subtree rooted at $v \in e$.

Problem C – Connecting Railway Stations

First, let us characterize when the sequence b_e results from a pairing:

1. Each edge to a leaf has $b_e = 1$.
2. For every internal vertex v, the sum of b_e over the edges incident to v is even.
3. For every vertex v, the values b_e of the edges incident to v satisfy the triangle inequality: $\sum_{e \ni v} b_e \geq 2 \max(b_e)$.

The second condition is equivalent to a different one: that parity of b_e is the same as the parity of the number of leaves in the subtree rooted at $v \in e$.

Sufficiency

Can be argued by induction or constructively, by providing an appropriate algorithm. Inductive argument could be the following: take any leaf and subsequently walk to the adjacent edge with largest b_e. After some number of steps, we reach another leaf. We subtract 1 on this path, remove those leaves, and induct.

Problem C – Connecting Railway Stations

We want to impose the former requirements on a sequence a_e by decreasing some of its elements.

Problem C – Connecting Railway Stations

We want to impose the former requirements on a sequence a_e by decreasing some of its elements.
Assume that a_e has the proper parity (the same as any possible b_e). If not, decrease it by 1. Decrease the values of a_e on leaves to 1 (if $a_e < 1$, there is no valid pairing).

Problem C – Connecting Railway Stations

We want to impose the former requirements on a sequence a_e by decreasing some of its elements.
Assume that a_e has the proper parity (the same as any possible b_e). If not, decrease it by 1. Decrease the values of a_e on leaves to 1 (if $a_e < 1$, there is no valid pairing).
We have ensured that the first two demanded properties are satisfied.

Problem C – Connecting Railway Stations

We want to impose the former requirements on a sequence a_e by decreasing some of its elements.
Assume that a_e has the proper parity (the same as any possible b_e). If not, decrease it by 1. Decrease the values of a_e on leaves to 1 (if $a_e < 1$, there is no valid pairing).
We have ensured that the first two demanded properties are satisfied.

Idea

If a node v does not satisfy the triangle inequality, we can find the incident with largest value a_e and decrease it to the sum of the a values of the rest incident edges. Parity requirement still holds. The operation can be repeated until the sequence a_e is valid. All changes are "forced", i.e. any valid $b \leq a$ remains valid after the decrease.

Problem C – Connecting Railway Stations

We want to impose the former requirements on a sequence a_e by decreasing some of its elements.
Assume that a_e has the proper parity (the same as any possible b_e). If not, decrease it by 1. Decrease the values of a_e on leaves to 1 (if $a_e < 1$, there is no valid pairing).
We have ensured that the first two demanded properties are satisfied.

Idea

If a node v does not satisfy the triangle inequality, we can find the incident with largest value a_e and decrease it to the sum of the a values of the rest incident edges. Parity requirement still holds. The operation can be repeated until the sequence a_e is valid. All changes are "forced", i.e. any valid $b \leq a$ remains valid after the decrease.

We have to make it faster (in some structured way).

Problem C – Connecting Railway Stations

There are two possible approaches for spreading the limits on a_e:

Problem C – Connecting Railway Stations

There are two possible approaches for spreading the limits on a_e:

- dp with rerooting

Problem C – Connecting Railway Stations

There are two possible approaches for spreading the limits on a_e:

- dp with rerooting
- Dijkstra-like approach

Problem C – Connecting Railway Stations

There are two possible approaches for spreading the limits on a_e:

- dp with rerooting
- Dijkstra-like approach

The former one is very standard (and quite uninteresting). Let us focus on the latter.

Problem C – Connecting Railway Stations

There are two possible approaches for spreading the limits on a_e:

- dp with rerooting
- Dijkstra-like approach

The former one is very standard (and quite uninteresting). Let us focus on the latter.

Dijkstra-like approach

In short, we can take the edge that does not satisfy the triangle inequality and have the least value of a_e. Then, correct the value a_e and repeat.

Problem C – Connecting Railway Stations

There are two possible approaches for spreading the limits on a_e:

- dp with rerooting
- Dijkstra-like approach

The former one is very standard (and quite uninteresting). Let us focus on the latter.

Dijkstra-like approach

In short, we can take the edge that does not satisfy the triangle inequality and have the least value of a_e. Then, correct the value a_e and repeat.

The following slides will contain a justification of the approach. They are meant to show a broader perspective. Hence, they may look advanced, while they are not.

Problem C – Connecting Railway Stations

Dijkstra as an algorithm solving a (dual) linear program

Given a graph with w_{uv} – weight on edge $u \to v$ and a source vertex s, we want to find distances d_v such that:

- $d_s = 0$
- $d_u + w_{uv} \geq d_v$
- the sum of d_v is maximized

Problem C – Connecting Railway Stations

Dijkstra as an algorithm solving a (dual) linear program

Given a graph with w_{uv} – weight on edge $u \to v$ and a source vertex s, we want to find distances d_v such that:

- $d_s = 0$
- $d_u + w_{uv} \geq d_v$
- the sum of d_v is maximized

Our aim

We want to find values b_e such that:

- $b_e = 1$ for edges to leaves
- $b_e \leq a_e$
- $\sum_{e \ni v, e \neq f} b_e \geq b_f$ where e, f are incident to a internal vertex v
- the sum of b_e is maximized

Problem C – Connecting Railway Stations

Dijkstra works as follows:

1. Starts with all vertices unmarked
2. Sets d_s to 0 and d_v to ∞
3. Finds an unmarked vertex u with the least value of d_u
4. Marks this vertex, checks if all inequalities of the form $d_u + \ldots \geq d_v$. If not, it decreases d_v.

Problem C – Connecting Railway Stations

Dijkstra works as follows:

1. Starts with all vertices unmarked
2. Sets d_s to 0 and d_v to ∞
3. Finds an unmarked vertex u with the least value of d_u
4. Marks this vertex, checks if all inequalities of the form $d_u + \ldots \geq d_v$. If not, it decreases d_v.

We can use the same pseudocode in our case.

Problem C – Connecting Railway Stations

Dijkstra works as follows:

1. Starts with all vertices unmarked
2. Sets d_s to 0 and d_v to ∞
3. Finds an unmarked vertex u with the least value of d_u
4. Marks this vertex, checks if all inequalities of the form $d_u + \ldots \geq d_v$. If not, it decreases d_v.

We can use the same pseudocode in our case.

1. Start with all edges unmarked
2. Set b_e to a_e
3. Find an unmarked edge f with the least value of b_f
4. Mark this edge, check if all inequalities of the form $b_f + \ldots \geq b_e$. If not, decrease b_e. This only makes sense when there is exactly one unmarked edge, since otherwise the LHS is undefined. Hence, this step will take linear time overall.

Problem C – Connecting Railway Stations

Dijkstra works as follows:

1. Starts with all vertices unmarked
2. Sets d_s to 0 and d_v to ∞
3. Finds an unmarked vertex u with the least value of d_u
4. Marks this vertex, checks if all inequalities of the form $d_u + \ldots \geq d_v$. If not, it decreases d_v.

We can use the same pseudocode in our case.

1. Start with all edges unmarked
2. Set b_e to a_e
3. Find an unmarked edge f with the least value of b_f
4. Mark this edge, check if all inequalities of the form $b_f + \ldots \geq b_e$. If not, decrease b_e. This only makes sense when there is exactly one unmarked edge, since otherwise the LHS is undefined. Hence, this step will take linear time overall.

We did not enforced parity condition explicitly, but this can be shown to remain correct.

Problem J – Joyful Guided Tour

Problem J – Joyful Guided Tour

Accepted before freeze: 4

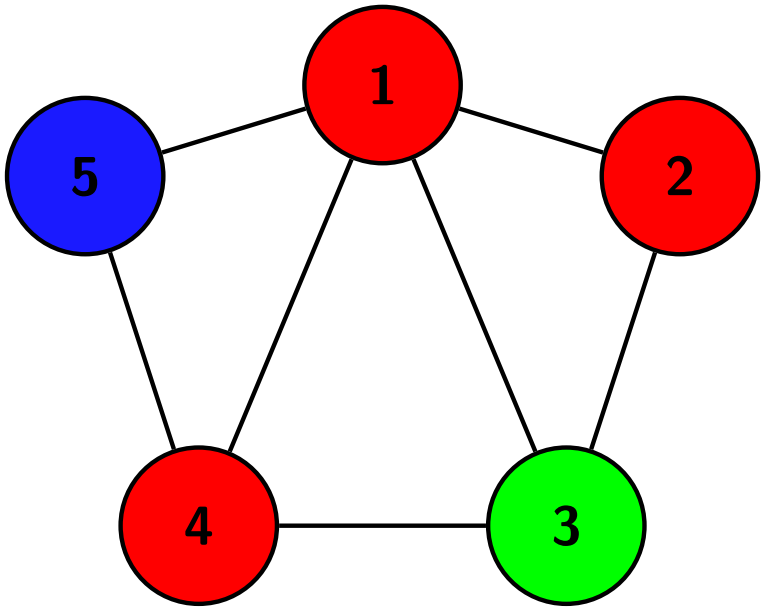
Teams that submitted after freeze: 10

Problem J – Joyful Guided Tour

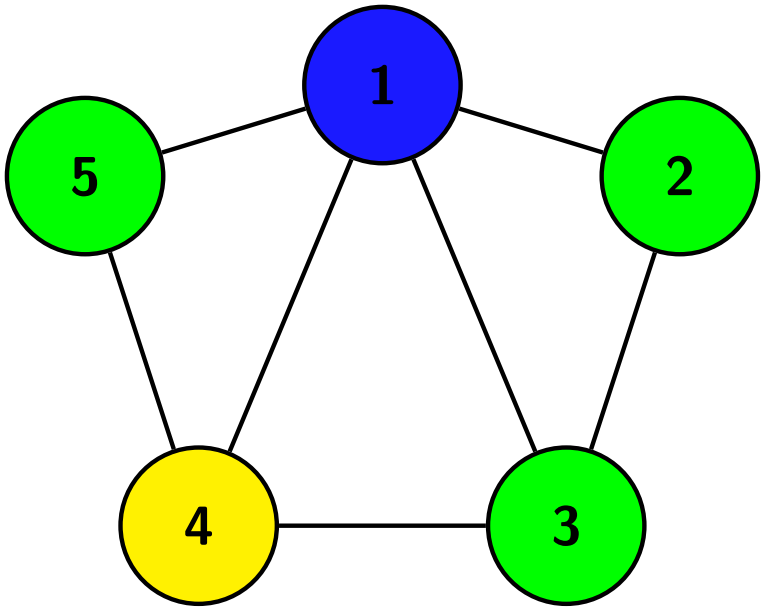
Problem

Given a graph with maximal node degree not bigger than 7.
Find coloring of nodes with at most 4 colors in a way, that there will be no simple path of length 3 with nodes of one color.

Bad:



Good:

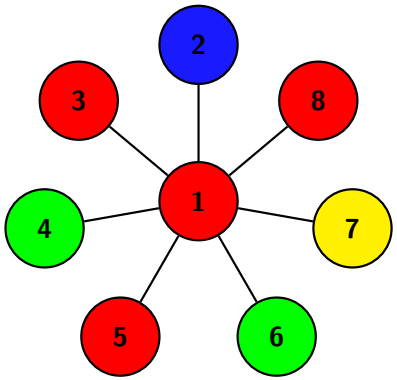


Problem J – Joyful Guided Tour

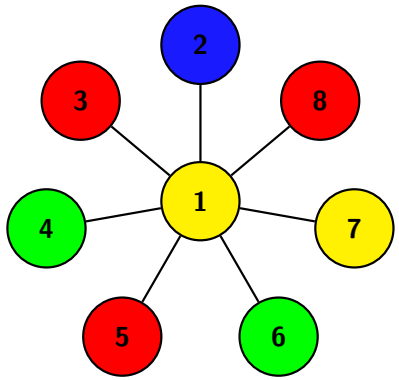
Solution:

- Find any coloring.
- While there is any path of 3 vertices with one color, repaint the central one with the color, that appears minimal number of times among his neighbours.

Before:



After:



After every repaint, number of edges with the same color at endpoints decreases. Therefore after at most $3.5 \cdot N$ iterations our coloring will be feasible! If you start with random coloring, this number is about 4 times smaller.

Problem F – Foxes

Problem F – Foxes

Accepted before freeze: 2

Teams that submitted after freeze: 5

Problem F – Foxes

Problem

Given sequence of numbers, a pointer (pointing to the first number) and a sequence of queries of following types:

- move pointer one position to the left,
- move pointer one position to the right,
- exchange value under the pointer with another one.

After every query we need to answer the length of *Longest Increasing Subsequence* (LIS) of the whole sequence.

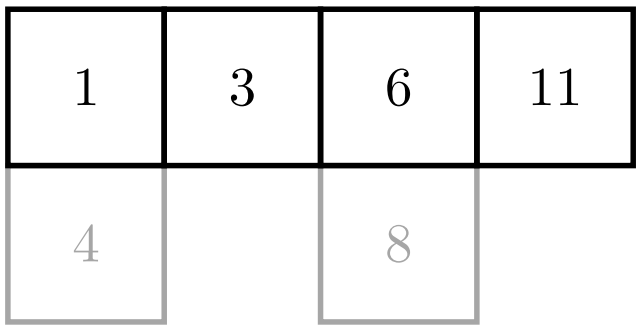
Problem F – Foxes

Observation

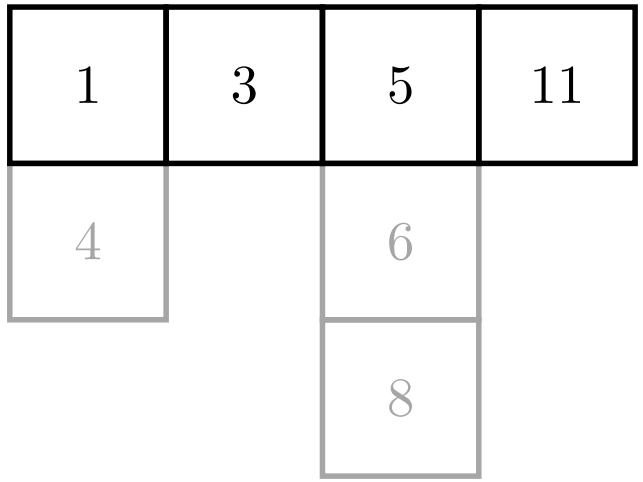
Using classical *Longest Increasing Subsequence* algorithm it is easy to add or remove element at the end of a sequence.

Example sequence: `4 1 3 8 6 11 (5)`

Before append of element 5:



After append of element 5:



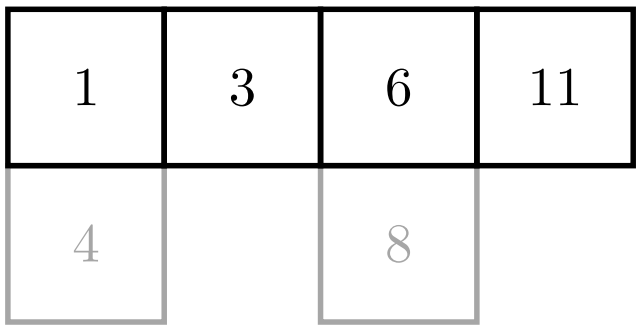
Problem F – Foxes

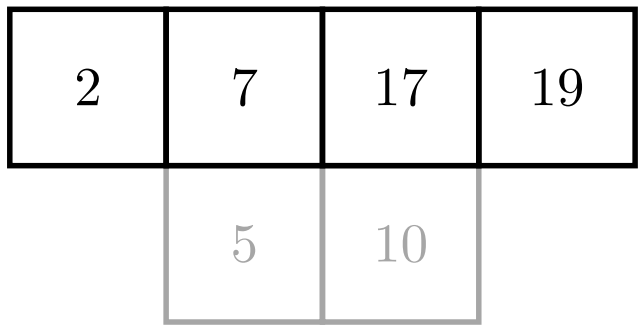
Observation 2

We can keep track of two LIS structures: one on the left side of the pointer and one on the right side.

Right structure will be *reversed*, i.e. it will find decreasing sequence, going from back to front.

Example sequence: `4 1 3 8 6 (11) 17 7 2 5 10 19`





Problem F – Foxes

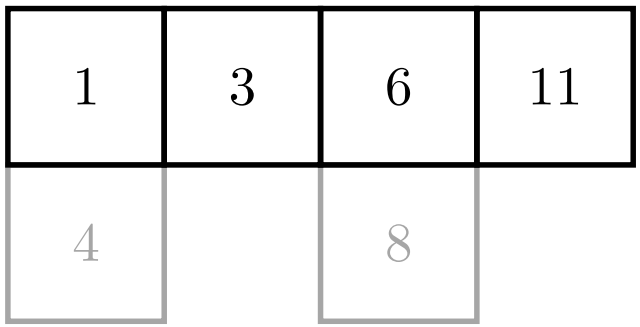
Now, operation of moving pointer left/right and exchange of some value are equivalent to addition and removal of some values on the structures.

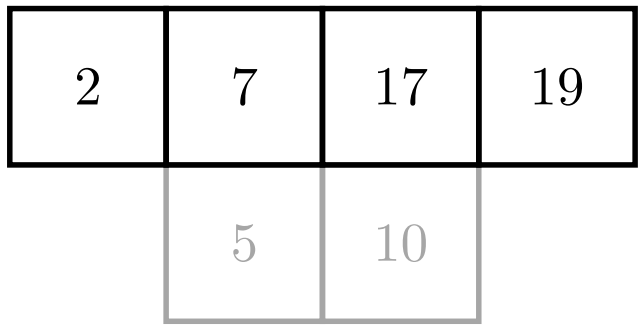
To keep track of the LIS of the whole sequence, we need a simple segment tree.

For each point x in this tree we keep the information *what is the LIS of the whole sequence if we take elements lower than x from left structure and bigger than x form the right one*.

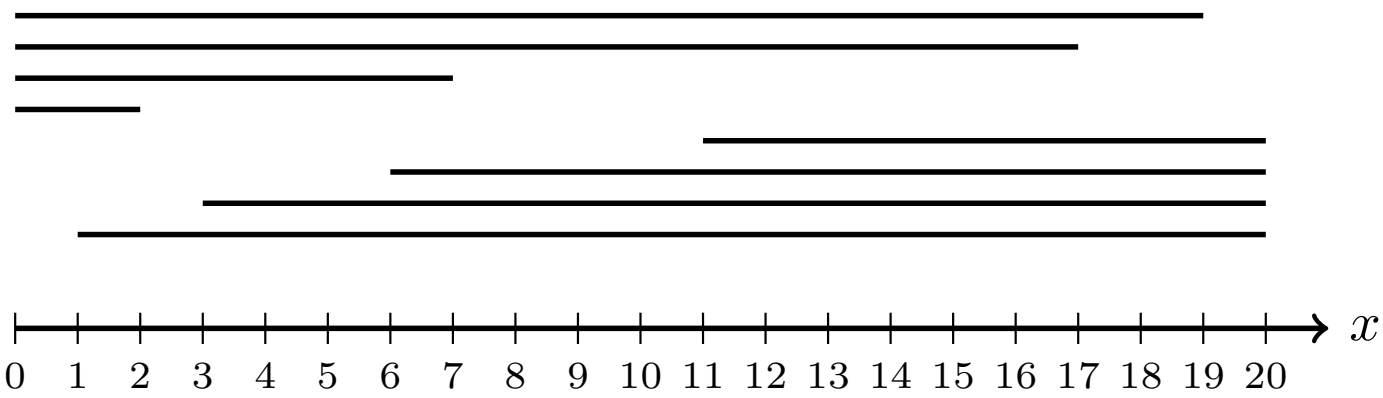
Problem F – Foxes

Example sequence: `4 1 3 8 6 (11) 17 7 2 5 10 19`





Ranges inserted on the segment tree:



Problem B – Bit Recovery

Problem B – Bit Recovery

Accepted before freeze: 0

Teams that submitted after freeze: 4

Problem B – Bit Recovery

1. Denote the xor operation by $\oplus$ and let $V = [0, 2^n - 1]$. Note that $(V, \oplus)$ is a linear space.

Problem B – Bit Recovery

1. Denote the xor operation by $\oplus$ and let $V = [0, 2^n - 1]$. Note that $(V, \oplus)$ is a linear space.
2. A subspace $U \subseteq V$ is a subset of V, closed under $\oplus$.

Problem B – Bit Recovery

1. Denote the xor operation by $\oplus$ and let $V = [0, 2^n - 1]$. Note that $(V, \oplus)$ is a linear space.
2. A subspace $U \subseteq V$ is a subset of V, closed under $\oplus$.
3. Vectors $a_1, \ldots, a_k$ span a subspace U if $a_i \in U$ and U is a minimal subspace with this property.

Problem B – Bit Recovery

1. Denote the xor operation by $\oplus$ and let $V = [0, 2^n - 1]$. Note that $(V, \oplus)$ is a linear space.
2. A subspace $U \subseteq V$ is a subset of V, closed under $\oplus$.
3. Vectors $a_1, \ldots, a_k$ span a subspace U if $a_i \in U$ and U is a minimal subspace with this property.
4. The rank/dimension of U is the size of a basis of U (minimal subset spanning U). Moreover, $|U| = 2^{\text{rank}(U)}$.

Problem B – Bit Recovery

1. Denote the xor operation by $\oplus$ and let $V = [0, 2^n - 1]$. Note that $(V, \oplus)$ is a linear space.
2. A subspace $U \subseteq V$ is a subset of V, closed under $\oplus$.
3. Vectors $a_1, \ldots, a_k$ span a subspace U if $a_i \in U$ and U is a minimal subspace with this property.
4. The rank/dimension of U is the size of a basis of U (minimal subset spanning U). Moreover, $|U| = 2^{\text{rank}(U)}$.
5. The rank of a set of vectors is the rank of the subspace spanned by them.

Problem B – Bit Recovery

1. Denote the xor operation by $\oplus$ and let $V = [0, 2^n - 1]$. Note that $(V, \oplus)$ is a linear space.
2. A subspace $U \subseteq V$ is a subset of V, closed under $\oplus$.
3. Vectors $a_1, \ldots, a_k$ span a subspace U if $a_i \in U$ and U is a minimal subspace with this property.
4. The rank/dimension of U is the size of a basis of U (minimal subset spanning U). Moreover, $|U| = 2^{\text{rank}(U)}$.
5. The rank of a set of vectors is the rank of the subspace spanned by them.

Statement

There is a hidden sequence $v_1, \ldots v_n \subseteq V$. We have to find the sequence, using queries of the following type:

- We can provide a sequence $u_1, \ldots u_n$ and, in response, learn what is the rank of $u_1 \oplus v_1, \ldots u_n \oplus v_n$

We have to use at most $n^2 + \mathcal{O}(1)$ queries.

Problem B – Bit Recovery

Claim

Random set of n elements of V has a constant probability of being linearly independent (having rank n).

Problem B – Bit Recovery

Claim

Random set of n elements of V has a constant probability of being linearly independent (having rank n).

This probability is exactly $\left(1-\frac{1}{2^n}\right)\cdot\left(1-\frac{2}{2^n}\right)\cdot\left(1-\frac{4}{2^n}\right)\cdot\ldots\cdot\left(1-\frac{2^{n-1}}{2^n}\right)$, which is $\left(1-\frac{1}{2}\right)\cdot\left(1-\frac{1}{4}\right)\cdot\ldots\cdot\left(1-2^{-n}\right)$ and tends to constant for $n\to\infty$.

Problem B – Bit Recovery

Claim

Random set of n elements of V has a constant probability of being linearly independent (having rank n).

This probability is exactly $\left(1-\frac{1}{2^n}\right)\cdot\left(1-\frac{2}{2^n}\right)\cdot\left(1-\frac{4}{2^n}\right)\cdot\ldots\cdot\left(1-\frac{2^{n-1}}{2^n}\right)$, which is $\left(1-\frac{1}{2}\right)\cdot\left(1-\frac{1}{4}\right)\cdot\ldots\cdot(1-2^{-n})$ and tends to constant for $n\to\infty$. We can choose the sequence u_i randomly, until we find $v_i'=v_i\oplus u_i$ that are independent. It takes $\mathcal{O}(1)$ queries in expectation.

Problem B – Bit Recovery

Claim

Random set of n elements of V has a constant probability of being linearly independent (having rank n).

This probability is exactly $\left(1-\frac{1}{2^n}\right)\cdot\left(1-\frac{2}{2^n}\right)\cdot\left(1-\frac{4}{2^n}\right)\cdot\ldots\cdot\left(1-\frac{2^{n-1}}{2^n}\right)$, which is $\left(1-\frac{1}{2}\right)\cdot\left(1-\frac{1}{4}\right)\cdot\ldots\cdot(1-2^{-n})$ and tends to constant for $n\to\infty$.
We can choose the sequence u_i randomly, until we find $v_i' = v_i \oplus u_i$ that are independent. It takes $\mathcal{O}(1)$ queries in expectation.
We can assume without loss of generality that $v_i = v_i'$ and adjust later queries. From now on, v_i will be linearly independent.

Problem B – Bit Recovery

Denote $V_i = \text{span}_{j \neq i} v_j$. By the assumption on independence of v_i, $\dim V_i = n - 1$ and $v_i \notin V_i$. The solution can be split into two phases: finding the subspaces V_i and using them to recover the sequence v_i.

Problem B – Bit Recovery

Denote $V_i = \text{span}_{j \neq i} v_j$. By the assumption on independence of v_i, $\dim V_i = n - 1$ and $v_i \notin V_i$. The solution can be split into two phases: finding the subspaces V_i and using them to recover the sequence v_i.

Crucial observation

If U is a subspace of dimension (rank) $n - 1$ and $x, y \notin U$, then $x \oplus y \in U$.

Problem B – Bit Recovery

Denote $V_i = \text{span}_{j \neq i} v_j$. By the assumption on independence of v_i, $\dim V_i = n - 1$ and $v_i \notin V_i$. The solution can be split into two phases: finding the subspaces V_i and using them to recover the sequence v_i.

Crucial observation

If U is a subspace of dimension (rank) $n - 1$ and $x, y \notin U$, then $x \oplus y \in U$.

Observation

Take $u = (0, 0, \ldots, 0, x, 0, \ldots, 0)$, where x is on i-th position. Then the result of query on u is:

- n if $x \in V_i$
- $n - 1$ if $x \notin V_i$

Problem B – Bit Recovery

Denote $V_i = \text{span}_{j \neq i} v_j$. By the assumption on independence of v_i, $\dim V_i = n - 1$ and $v_i \notin V_i$. The solution can be split into two phases: finding the subspaces V_i and using them to recover the sequence v_i.

Crucial observation

If U is a subspace of dimension (rank) $n - 1$ and $x, y \notin U$, then $x \oplus y \in U$.

Observation

Take $u = (0, 0, \ldots, 0, x, 0, \ldots, 0)$, where x is on i-th position. Then the result of query on u is:

- n if $x \in V_i$
- $n - 1$ if $x \notin V_i$

Testing $x \in \{1, 2, 4, \ldots, 2^{n-1}\}$, we can recover V_i (using the observations).

Problem B – Bit Recovery

What is left is to recover v_i using V_i (we identify a subspace with its basis). Notice that $\{0, v_i\} = \bigcap_{j \neq i} V_j$.

Problem B – Bit Recovery

What is left is to recover v_i using V_i (we identify a subspace with its basis). Notice that $\{0, v_i\} = \bigcap_{j \neq i} V_j$.
Hence, it suffices to calculate the intersection of subspaces. There is a general algorithm (named after Zassenhaus) but our case is easier. What we have to do is: given a subspace W with basis $b_1, \ldots b_k$, intersect it with U of dimension $n - 1$.

Problem B – Bit Recovery

What is left is to recover v_i using V_i (we identify a subspace with its basis).
Notice that $\{0, v_i\} = \bigcap_{j \neq i} V_j$.
Hence, it suffices to calculate the intersection of subspaces. There is a general algorithm (named after Zassenhaus) but our case is easier. What we have to do is: given a subspace W with basis $b_1, \ldots b_k$, intersect it with U of dimension $n - 1$.
Assume w.l.o.g. that $b_1, \ldots, b_l \notin U$ and $b_{l+1}, \ldots b_k \in U$.

Problem B – Bit Recovery

What is left is to recover v_i using V_i (we identify a subspace with its basis). Notice that $\{0, v_i\} = \bigcap_{j \neq i} V_j$.
Hence, it suffices to calculate the intersection of subspaces. There is a general algorithm (named after Zassenhaus) but our case is easier. What we have to do is: given a subspace W with basis $b_1, \ldots b_k$, intersect it with U of dimension $n - 1$.
Assume w.l.o.g. that $b_1, \ldots, b_l \notin U$ and $b_{l+1}, \ldots b_k \in U$.

1. If $l = 0$, we are done ($W \subseteq U$)

Problem B – Bit Recovery

What is left is to recover v_i using V_i (we identify a subspace with its basis).
Notice that $\{0, v_i\} = \bigcap_{j \neq i} V_j$.
Hence, it suffices to calculate the intersection of subspaces. There is a general algorithm (named after Zassenhaus) but our case is easier. What we have to do is: given a subspace W with basis $b_1, \ldots b_k$, intersect it with U of dimension $n - 1$.
Assume w.l.o.g. that $b_1, \ldots, b_l \notin U$ and $b_{l+1}, \ldots b_k \in U$.

1. If $l = 0$, we are done ($W \subseteq U$)
2. From the main observation, we can deduce that $b_1 \oplus b_2, b_1 \oplus b_3, \ldots, b_1 \oplus b_l \in U$.

Problem B – Bit Recovery

What is left is to recover v_i using V_i (we identify a subspace with its basis). Notice that $\{0, v_i\} = \bigcap_{j \neq i} V_j$.
Hence, it suffices to calculate the intersection of subspaces. There is a general algorithm (named after Zassenhaus) but our case is easier. What we have to do is: given a subspace W with basis $b_1, \ldots b_k$, intersect it with U of dimension $n - 1$.
Assume w.l.o.g. that $b_1, \ldots, b_l \notin U$ and $b_{l+1}, \ldots b_k \in U$.

1. If $l = 0$, we are done ($W \subseteq U$)
2. From the main observation, we can deduce that $b_1 \oplus b_2, b_1 \oplus b_3, \ldots, b_1 \oplus b_l \in U$.
3. Hence, $b_1 \oplus b_2, \ldots, b_1 \oplus b_l$ together with $b_{l+1}, \ldots b_k$ form a basis of $W \cap U$

Problem E – Easy Tetris

Problem E – Easy Tetris

Accepted before freeze: 0

Teams that submitted after freeze: 7

Problem E – Easy Tetris

Problem

Easy Tetris is a simplified variation of the standard *Tetris* game. There are only I–pieces and O–pieces. Also, we can't rotate pieces nor move them midair. We define *tetris* as the event when after dropping a piece, four rows are cleared simultaneously.

Given the sequence of pieces to be dropped, calculate the maximum number of *tetrises* we can score and give the sequence of moves to achieve it.

Problem E – Easy Tetris

Remark

We will present the jury's solution to the problem. We believe though there are other approaches to solve it as well.

Problem E – Easy Tetris

Binary search

We will use binary search over the answer. By k we will later denote the number of tetrises we want to check if it is possible to achieve.

Problem E – Easy Tetris

Key intuitions and observations

Problem E – Easy Tetris

Key intuitions and observations

- We need at least $2k$ I-pieces.

Problem E – Easy Tetris

Key intuitions and observations

- We need at least $2k$ I-pieces.
- We want 2 columns where only I-pieces will be dropped (column 9 and 10 for simplicity).

Problem E – Easy Tetris

Key intuitions and observations

- We need at least $2k$ I-pieces.
- We want 2 columns where only I-pieces will be dropped (column 9 and 10 for simplicity).
- Before dropping anything into column 10, fill (make it reach height at least $4k$) column 9.

Problem E – Easy Tetris

Key intuitions and observations

- We need at least $2k$ I-pieces.
- We want 2 columns where only I-pieces will be dropped (column 9 and 10 for simplicity).
- Before dropping anything into column 10, fill (make it reach height at least $4k$) column 9.
- Drop the O-pieces only in the columns 1, 3, 5 or 7.

Problem E – Easy Tetris

Key intuitions and observations

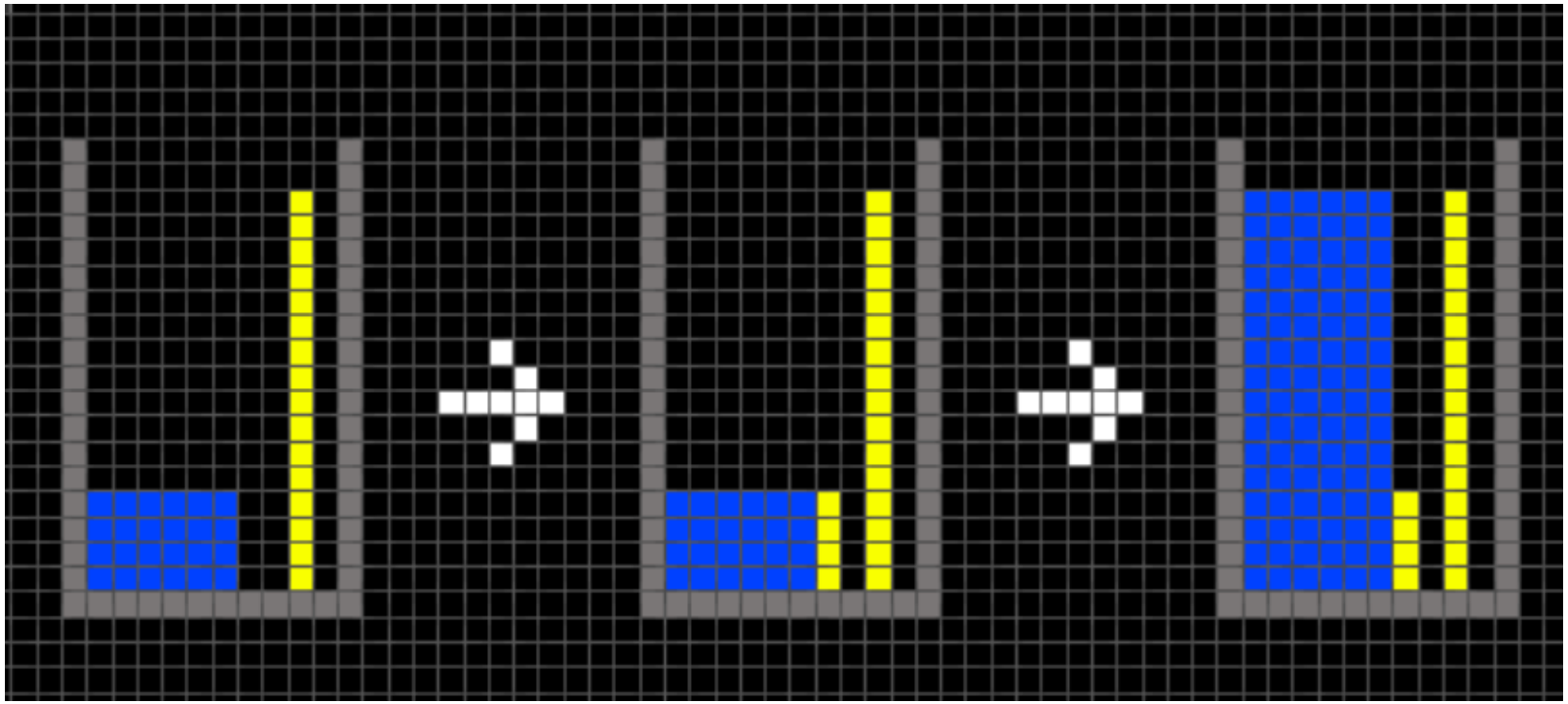
- We need at least $2k$ I-pieces.
- We want 2 columns where only I-pieces will be dropped (column 9 and 10 for simplicity).
- Before dropping anything into column 10, fill (make it reach height at least $4k$) column 9.
- Drop the O-pieces only in the columns 1, 3, 5 or 7.
- Keep the area in columns 1 to 8 as flat as possible (do not start a new level until the previous one is not ready for a *tetris*).

Problem E – Easy Tetris

Key intuitions and observations

- We need at least $2k$ I-pieces.
- We want 2 columns where only I-pieces will be dropped (column 9 and 10 for simplicity).
- Before dropping anything into column 10, fill (make it reach height at least $4k$) column 9.
- Drop the O-pieces only in the columns 1, 3, 5 or 7.
- Keep the area in columns 1 to 8 as flat as possible (do not start a new level until the previous one is not ready for a *tetris*).
- Keep the area in columns 1 to 8 higher than column 9 if possible.

Problem E – Easy Tetris



Problem E – Easy Tetris

For simplicity, we will denote paired colums (1 with 2, 3 with 4 etc.) as wide columns.

Problem E – Easy Tetris

For simplicity, we will denote paired colums (1 with 2, 3 with 4 etc.) as wide columns.

O–piece strategy

Problem E – Easy Tetris

For simplicity, we will denote paired colums (1 with 2, 3 with 4 etc.) as wide columns.

O–piece strategy

Consider only wide columns without single I-piece on top.

Problem E – Easy Tetris

For simplicity, we will denote paired colums (1 with 2, 3 with 4 etc.) as wide columns.

O–piece strategy

Consider only wide columns without single I-piece on top.

- If any of them has height 2 $\mod 4$, drop the O–piece into it.

Problem E – Easy Tetris

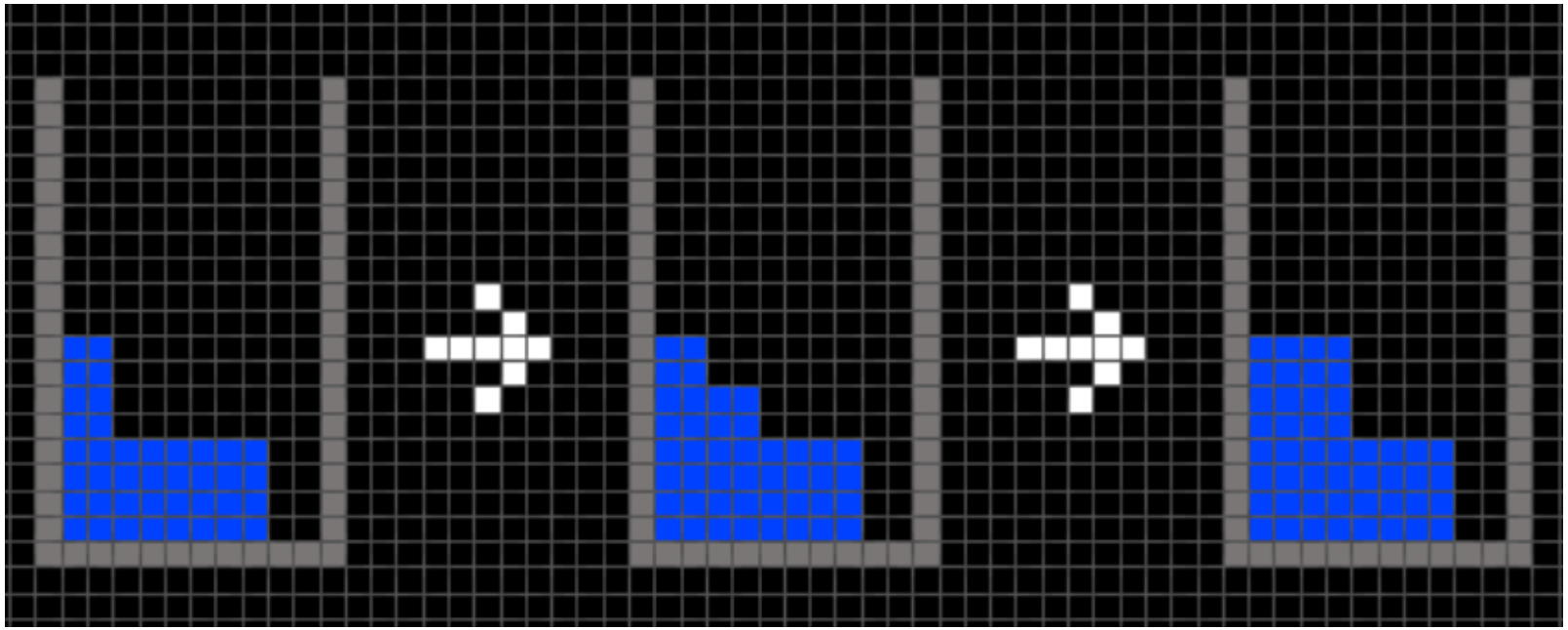
For simplicity, we will denote paired colums (1 with 2, 3 with 4 etc.) as wide columns.

O–piece strategy

Consider only wide columns without single I-piece on top.

- If any of them has height 2 $\mod 4$, drop the O–piece into it.
- Otherwise, drop it into the shortest one.

Problem E – Easy Tetris



Problem E – Easy Tetris

I–piece strategy

Problem E – Easy Tetris

I–piece strategy

- If the piece is necessary to fill column 9 or 10, we drop it there (9 if it is not filled, 10 otherwise).

Problem E – Easy Tetris

I–piece strategy

- If the piece is necessary to fill column 9 or 10, we drop it there (9 if it is not filled, 10 otherwise).
- Otherwise:

Problem E – Easy Tetris

I–piece strategy

- If the piece is necessary to fill column 9 or 10, we drop it there (9 if it is not filled, 10 otherwise).
- Otherwise:
  - If there are at least 8 O-pieces after the I-piece, we drop it to column 9.

Problem E – Easy Tetris

I–piece strategy

- If the piece is necessary to fill column 9 or 10, we drop it there (9 if it is not filled, 10 otherwise).
- Otherwise:
  - If there are at least 8 O-pieces after the I-piece, we drop it to column 9.
  - Otherwise, we pair it with the next I–piece and drop it somewhere into the wide columns, so that after dropping the O–pieces in between, we start a new level only when the previous one is finished.

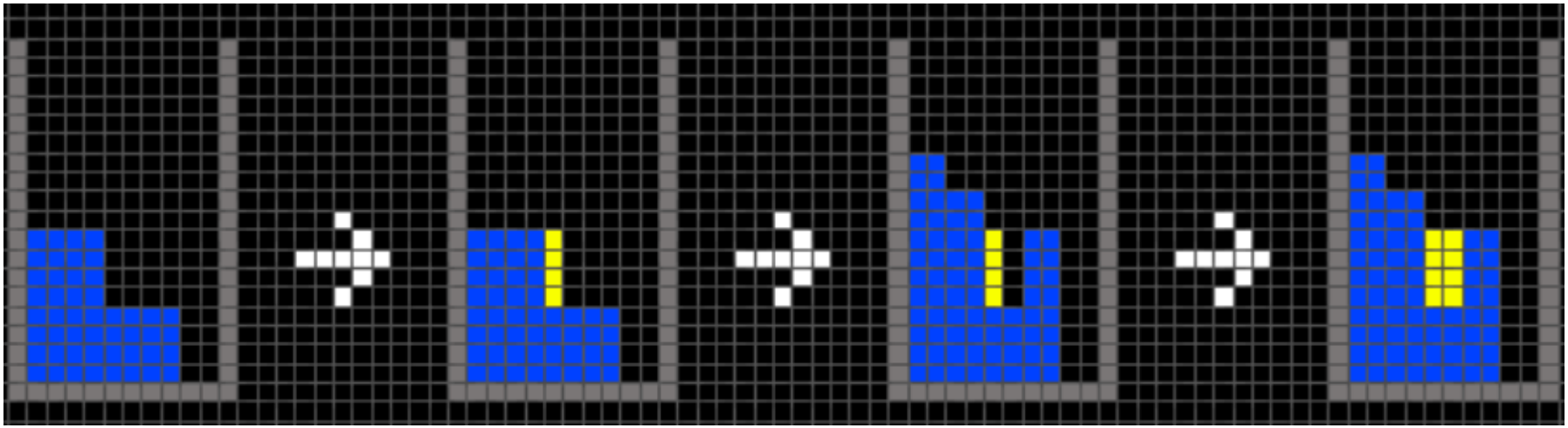
Problem E – Easy Tetris

I–piece strategy

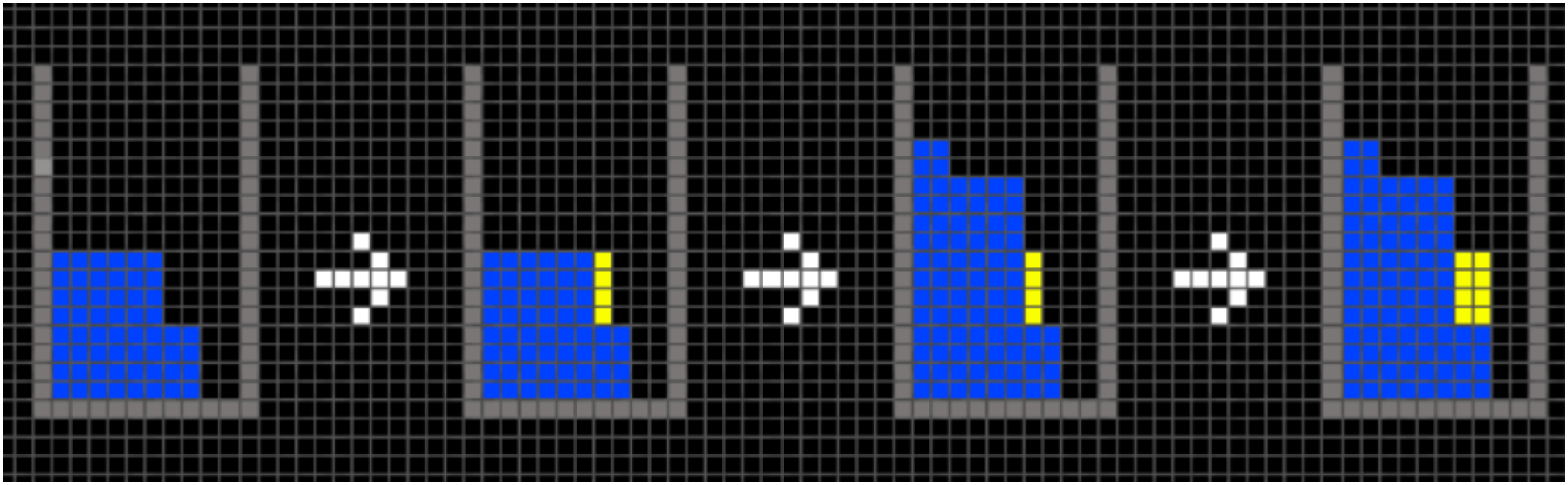
- If the piece is necessary to fill column 9 or 10, we drop it there (9 if it is not filled, 10 otherwise).
- Otherwise:
  - If there are at least 8 O-pieces after the I-piece, we drop it to column 9.
  - Otherwise, we pair it with the next I–piece and drop it somewhere into the wide columns, so that after dropping the O–pieces in between, we start a new level only when the previous one is finished.

Last case can be done by a few last conditions. If there is more than one wide column not filled up to the next level, or the one wide column can't be filled by dropping there the upcoming O-pieces, we can drop the I-pieces in the shortest wide column. Otherwise, we can pick any other one.

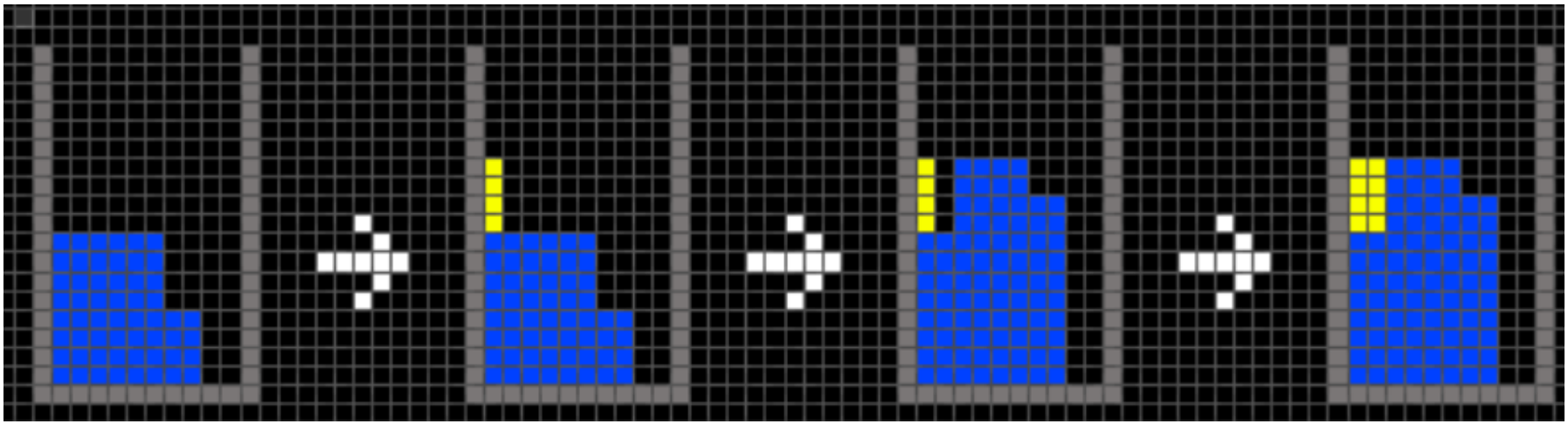
Problem E – Easy Tetris



Problem E – Easy Tetris



Problem E – Easy Tetris



Problem E – Easy Tetris

Proof of strategy

Problem E – Easy Tetris

Proof of strategy

Let's look at the moment when we try to score i-th tetris. If we do not succeed, that means the wide columns are lower than column 10.

Problem E – Easy Tetris

Proof of strategy

Let's look at the moment when we try to score i-th tetris. If we do not succeed, that means the wide columns are lower than column 10.

But we dropped the I-piece as late as we could, meaning that even if we could score the tetris now by dropping some of the earlier I–pieces into the columns 1 to 8, there would not be enough I–pieces left to fill the columns 9 and 10.

Problem E – Easy Tetris

Proof of strategy

Let's look at the moment when we try to score i-th tetris. If we do not succeed, that means the wide columns are lower than column 10.

But we dropped the I-piece as late as we could, meaning that even if we could score the tetris now by dropping some of the earlier I–pieces into the columns 1 to 8, there would not be enough I–pieces left to fill the columns 9 and 10.

Final complexity: $\mathcal{O}(n \log n)$, which can be reduced to $\mathcal{O}(n)$ by avoiding binary search.

Thank you