# Freshmen Programming Contests 2024

Solutions presentation

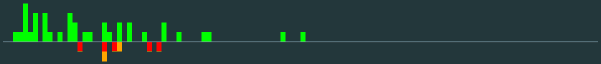
By the Freshmen Programming Contests 2024 jury for:

- AAPJE in Amsterdam
- FPC in Delft
- FYPC in Eindhoven
- GAPC in Groningen

May 4, 2024

# D: Dragged-out Duel

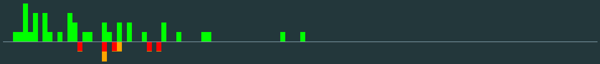
**Problem author: Wietze Koops**



- **Problem:** Read two lines, comprised of ‘R’, ‘P’, and ‘S’, and determine who wins the most games.

# D: Dragged-out Duel

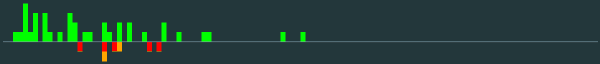
Problem author: Wietze Koops



- **Problem:** Read two lines, comprised of ‘`R`’, ‘`P`’, and ‘`S`’, and determine who wins the most games.
- **Solution:**
  - Read the two lines character by character, increment a counter if player 1 wins and decrement it if player 2 wins.
  - Finally, print “`victory`” if the counter is positive, and “`defeat`” if it is negative.

# D: Dragged-out Duel

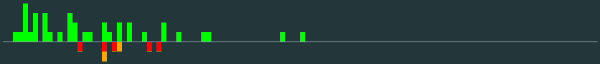
Problem author: Wietze Koops



- **Problem:** Read two lines, comprised of '`R`', '`P`', and '`S`', and determine who wins the most games.
- **Solution:**
  - Read the two lines character by character, increment a counter if player 1 wins and decrement it if player 2 wins.
  - Finally, print "`victory`" if the counter is positive, and "`defeat`" if it is negative.
- **Complexity:** $\mathcal{O}(n)$.

# D: Dragged-out Duel

Problem author: Wietze Koops



- **Problem:** Read two lines, comprised of '`R`', '`P`', and '`S`', and determine who wins the most games.
- **Solution:**
  - Read the two lines character by character, increment a counter if player 1 wins and decrement it if player 2 wins.
  - Finally, print "`victory`" if the counter is positive, and "`defeat`" if it is negative.
- **Complexity:** $\mathcal{O}(n)$.

Statistics: 45 submissions, 37 accepted

# B: Building Pyramids

Problem author: Maarten Sijm

- **Problem:** Given the edge length of a tetrahedron, calculate the number of spheres in the pyramid.

# B: Building Pyramids

**Problem author: Maarten Sijm**

- **Problem:** Given the edge length of a tetrahedron, calculate the number of spheres in the pyramid.
- **Observation:** The pyramid consists of $n$ equilateral triangles.
  The number of spheres in triangle $t$ is $T(t) = \sum_{i=1}^{t} i$.

# B: Building Pyramids

Problem author: Maarten Sijm

- **Problem:** Given the edge length of a tetrahedron, calculate the number of spheres in the pyramid.
- **Observation:** The pyramid consists of $n$ equilateral triangles.
  The number of spheres in triangle $t$ is $T(t) = \sum_{i=1}^{t} i$.
- **Slow solution:** Calculate $P(n) = \sum_{i=1}^{n} T(i)$. Runs in $\mathcal{O}(n^2)$, too slow.

# B: Building Pyramids

**Problem author: Maarten Sijm**

- **Problem:** Given the edge length of a tetrahedron, calculate the number of spheres in the pyramid.
- **Observation:** The pyramid consists of $n$ equilateral triangles.
  The number of spheres in triangle $t$ is $T(t) = \sum_{i=1}^{t} i$.
- **Slow solution:** Calculate $P(n) = \sum_{i=1}^{n} T(i)$. Runs in $\mathcal{O}(n^2)$, too slow.
- **Solution:** Simplify $T(t) = \frac{t \cdot (t+1)}{2}$. Now calculating $P(n)$ runs in $\mathcal{O}(n)$, accepted!

# B: Building Pyramids

Problem author: Maarten Sijm

- **Problem:** Given the edge length of a tetrahedron, calculate the number of spheres in the pyramid.
- **Observation:** The pyramid consists of $n$ equilateral triangles.
  The number of spheres in triangle $t$ is $T(t) = \sum_{i=1}^{t} i$.
- **Slow solution:** Calculate $P(n) = \sum_{i=1}^{n} T(i)$. Runs in $\mathcal{O}(n^2)$, too slow.
- **Solution:** Simplify $T(t) = \frac{t \cdot (t+1)}{2}$. Now calculating $P(n)$ runs in $\mathcal{O}(n)$, accepted!
- **Pitfall:** If $t$ is an `int`, $t \cdot (t + 1)$ overflows. Use 64-bit integers!

# B: Building Pyramids

Problem author: Maarten Sijm

- **Problem:** Given the edge length of a tetrahedron, calculate the number of spheres in the pyramid.
- **Observation:** The pyramid consists of $n$ equilateral triangles.
  The number of spheres in triangle $t$ is $T(t) = \sum_{i=1}^{t} i$.
- **Slow solution:** Calculate $P(n) = \sum_{i=1}^{n} T(i)$. Runs in $\mathcal{O}(n^2)$, too slow.
- **Solution:** Simplify $T(t) = \frac{t \cdot (t+1)}{2}$. Now calculating $P(n)$ runs in $\mathcal{O}(n)$, accepted!
- **Pitfall:** If $t$ is an `int`, $t \cdot (t+1)$ overflows. Use 64-bit integers!
- **Challenge:** The calculation of $P(n)$ can even be simplified to run in $\mathcal{O}(1)$.

# B: Building Pyramids

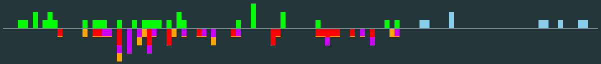
Problem author: Maarten Sijm

- **Problem:** Given the edge length of a tetrahedron, calculate the number of spheres in the pyramid.
- **Observation:** The pyramid consists of $n$ equilateral triangles.
  The number of spheres in triangle $t$ is $T(t) = \sum_{i=1}^{t} i$.
- **Slow solution:** Calculate $P(n) = \sum_{i=1}^{n} T(i)$. Runs in $\mathcal{O}(n^2)$, too slow.
- **Solution:** Simplify $T(t) = \frac{t \cdot (t+1)}{2}$. Now calculating $P(n)$ runs in $\mathcal{O}(n)$, accepted!
- **Pitfall:** If $t$ is an `int`, $t \cdot (t+1)$ overflows. Use 64-bit integers!
- **Challenge:** The calculation of $P(n)$ can even be simplified to run in $\mathcal{O}(1)$.

Statistics: 75 submissions, 36 accepted, 1 unknown

# F: Flag Rotation

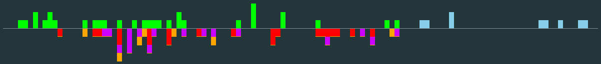
Problem author: Jeroen Op de Beek



- **Problem:** Count how many cells will change when painting the flag rotated.

# F: Flag Rotation

**Problem author: Jeroen Op de Beek**



- **Problem:** Count how many cells will change when painting the flag rotated.
- **Observation:** Since each column has to be repainted to one color, we will change $n - cnt_c$ cells in it (where $c$ is the final color).

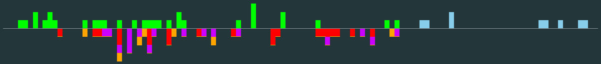
# F: Flag Rotation

Problem author: Jeroen Op de Beek

- **Problem:** Count how many cells will change when painting the flag rotated.
- **Observation:** Since each column has to be repainted to one color, we will change $n - cnt_c$ cells in it (where $c$ is the final color).
- **Solution:** Count how many cells won't change.

# F: Flag Rotation

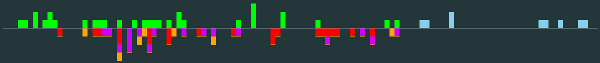
Problem author: Jeroen Op de Beek



- **Problem:** Count how many cells will change when painting the flag rotated.
- **Observation:** Since each column has to be repainted to one color, we will change $n - cnt_c$ cells in it (where $c$ is the final color).
- **Solution:** Count how many cells won't change.
  - First sort the array, then check for segments made of identical elements, this way we find the count of each cell color.

# F: Flag Rotation

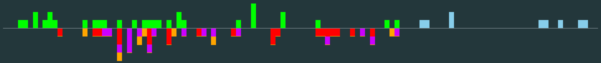
Problem author: Jeroen Op de Beek



- **Problem:** Count how many cells will change when painting the flag rotated.
- **Observation:** Since each column has to be repainted to one color, we will change $n - cnt_c$ cells in it (where $c$ is the final color).
- **Solution:** Count how many cells won't change.
    - First sort the array, then check for segments made of identical elements, this way we find the count of each cell color.
    - answer $= n^2 - \sum_c cnt_c^2$

# F: Flag Rotation

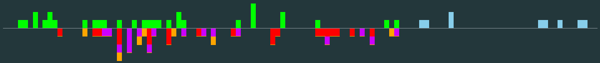
Problem author: Jeroen Op de Beek



- **Problem:** Count how many cells will change when painting the flag rotated.
- **Observation:** Since each column has to be repainted to one color, we will change $n - cnt_c$ cells in it (where $c$ is the final color).
- **Solution:** Count how many cells won't change.
  - First sort the array, then check for segments made of identical elements, this way we find the count of each cell color.
  - answer $= n^2 - \sum_c cnt_c^2$
- **Complexity:** $\mathcal{O}(n \log n)$.

# F: Flag Rotation

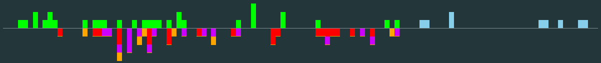
Problem author: Jeroen Op de Beek



- **Problem:** Count how many cells will change when painting the flag rotated.
- **Observation:** Since each column has to be repainted to one color, we will change $n - cnt_c$ cells in it (where $c$ is the final color).
- **Solution:** Count how many cells won't change.
  - First sort the array, then check for segments made of identical elements, this way we find the count of each cell color.
  - answer $= n^2 - \sum_c cnt_c^2$
- **Complexity:** $\mathcal{O}(n \log n)$.
- **Note:** this can also be done using a (hash) map.

# F: Flag Rotation

**Problem author: Jeroen Op de Beek**

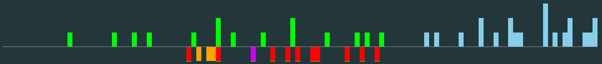


- **Problem:** Count how many cells will change when painting the flag rotated.
- **Observation:** Since each column has to be repainted to one color, we will change $n - cnt_c$ cells in it (where $c$ is the final color).
- **Solution:** Count how many cells won't change.
  - First sort the array, then check for segments made of identical elements, this way we find the count of each cell color.
  - answer $= n^2 - \sum_c cnt_c^2$
- **Complexity:** $\mathcal{O}(n \log n)$.
- **Note:** this can also be done using a (hash) map.

Statistics: 85 submissions, 31 accepted, 9 unknown

# E: European Election

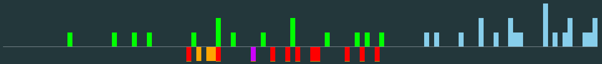
Problem author: Veselin Mitev



- **Problem:** Given ranked-choice ballots, determine the candidate who beats all other candidates.

# E: European Election

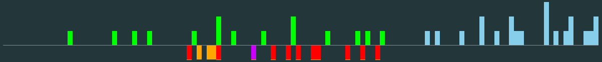
**Problem author: Veselin Mitev**



- **Problem:** Given ranked-choice ballots, determine the candidate who beats all other candidates.
- **Fun Fact:** This is also known as the *Condorcet* voting method.

# E: European Election

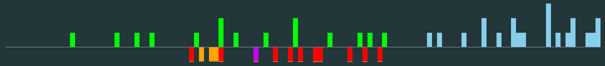
Problem author: Veselin Mitev



- **Problem:** Given ranked-choice ballots, determine the candidate who beats all other candidates.
- **Fun Fact:** This is also known as the *Condorcet* voting method.
- How to determine whether candidate A is better than B?
  - Go through all ballots – count how many times A appears before B, and vice-versa. Runs in $\mathcal{O}(n \cdot k)$.

# E: European Election

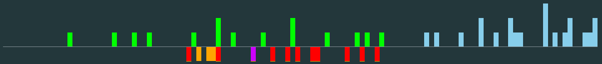
Problem author: Veselin Mitev



- **Problem:** Given ranked-choice ballots, determine the candidate who beats all other candidates.
- **Fun Fact:** This is also known as the *Condorcet* voting method.
- How to determine whether candidate A is better than B?
  - Go through all ballots – count how many times A appears before B, and vice-versa. Runs in $\mathcal{O}(n \cdot k)$.
- **Observation:** We can preprocess the ballots in $\mathcal{O}(n \cdot k)$, such that we can access the position that each candidate appears in each ballot in $\mathcal{O}(1)$. Thus, answering whether candidate A beats candidate B, now only takes $\mathcal{O}(n)$.

# E: European Election

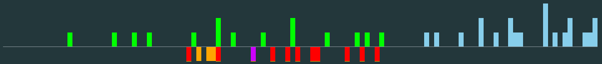
Problem author: Veselin Mitev



- **Problem:** Given ranked-choice ballots, determine the candidate who beats all other candidates.
- **Fun Fact:** This is also known as the *Condorcet* voting method.
- How to determine whether candidate A is better than B?
  - Go through all ballots – count how many times A appears before B, and vice-versa. Runs in $\mathcal{O}(n \cdot k)$.
- **Observation:** We can preprocess the ballots in $\mathcal{O}(n \cdot k)$, such that we can access the position that each candidate appears in each ballot in $\mathcal{O}(1)$. Thus, answering whether candidate A beats candidate B, now only takes $\mathcal{O}(n)$.
- **Solution:**
  - Pick a candidate $d$.

# E: European Election

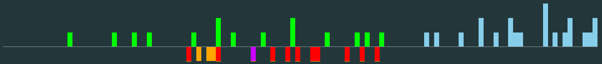
Problem author: Veselin Mitev



- **Problem:** Given ranked-choice ballots, determine the candidate who beats all other candidates.
- **Fun Fact:** This is also known as the *Condorcet* voting method.
- How to determine whether candidate A is better than B?
  - Go through all ballots – count how many times A appears before B, and vice-versa. Runs in $\mathcal{O}(n \cdot k)$.
- **Observation:** We can preprocess the ballots in $\mathcal{O}(n \cdot k)$, such that we can access the position that each candidate appears in each ballot in $\mathcal{O}(1)$. Thus, answering whether candidate A beats candidate B, now only takes $\mathcal{O}(n)$.
- **Solution:**
  - Pick a candidate $d$.
  - Go through all candidates $c_i$: Anytime $c_i$ beats $d$: $d \leftarrow c_i$. Runs in $\mathcal{O}(n \cdot k)$.

# E: European Election

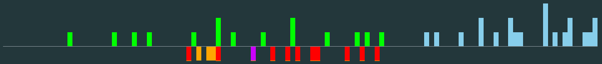
Problem author: Veselin Mitev



- **Problem:** Given ranked-choice ballots, determine the candidate who beats all other candidates.
- **Fun Fact:** This is also known as the *Condorcet* voting method.
- How to determine whether candidate A is better than B?
  - Go through all ballots – count how many times A appears before B, and vice-versa. Runs in $\mathcal{O}(n \cdot k)$.
- **Observation:** We can preprocess the ballots in $\mathcal{O}(n \cdot k)$, such that we can access the position that each candidate appears in each ballot in $\mathcal{O}(1)$. Thus, answering whether candidate A beats candidate B, now only takes $\mathcal{O}(n)$.
- **Solution:**
  - Pick a candidate $d$.
  - Go through all candidates $c_i$: Anytime $c_i$ beats $d$: $d \leftarrow c_i$. Runs in $\mathcal{O}(n \cdot k)$.
  - **Observation:** If the election has a winner, it must be $d$. (This can be proven using contradiction!)

# E: European Election

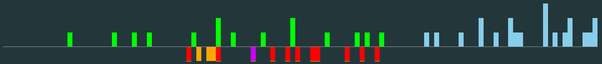
Problem author: Veselin Mitev



- **Problem:** Given ranked-choice ballots, determine the candidate who beats all other candidates.
- **Fun Fact:** This is also known as the *Condorcet* voting method.
- How to determine whether candidate A is better than B?
  - Go through all ballots – count how many times A appears before B, and vice-versa. Runs in $\mathcal{O}(n \cdot k)$.
- **Observation:** We can preprocess the ballots in $\mathcal{O}(n \cdot k)$, such that we can access the position that each candidate appears in each ballot in $\mathcal{O}(1)$. Thus, answering whether candidate A beats candidate B, now only takes $\mathcal{O}(n)$.
- **Solution:**
  - Pick a candidate $d$.
  - Go through all candidates $c_i$: Anytime $c_i$ beats $d$: $d \leftarrow c_i$. Runs in $\mathcal{O}(n \cdot k)$.
  - **Observation:** If the election has a winner, it must be $d$. (This can be proven using contradiction!)
  - Check if $d$ beats all other candidates. Runs in $\mathcal{O}(n \cdot k)$.

# E: European Election

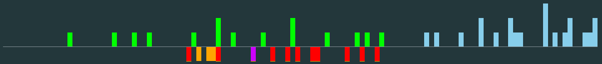
Problem author: Veselin Mitev



- **Problem:** Given ranked-choice ballots, determine the candidate who beats all other candidates.
- **Fun Fact:** This is also known as the *Condorcet* voting method.
- How to determine whether candidate A is better than B?
  - Go through all ballots – count how many times A appears before B, and vice-versa. Runs in $\mathcal{O}(n \cdot k)$.
- **Observation:** We can preprocess the ballots in $\mathcal{O}(n \cdot k)$, such that we can access the position that each candidate appears in each ballot in $\mathcal{O}(1)$. Thus, answering whether candidate A beats candidate B, now only takes $\mathcal{O}(n)$.
- **Solution:**
  - Pick a candidate $d$.
  - Go through all candidates $c_i$: Anytime $c_i$ beats $d$: $d \leftarrow c_i$. Runs in $\mathcal{O}(n \cdot k)$.
  - **Observation:** If the election has a winner, it must be $d$. (This can be proven using contradiction!)
  - Check if $d$ beats all other candidates. Runs in $\mathcal{O}(n \cdot k)$.
- **Complexity:** $\mathcal{O}(n \cdot k)$.

# E: European Election

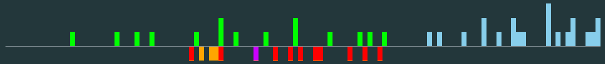
**Problem author: Veselin Mitev**



- **Problem:** Given ranked-choice ballots, determine the candidate who beats all other candidates.
- **Fun Fact:** This is also known as the *Condorcet* voting method.
- How to determine whether candidate A is better than B?
  - Go through all ballots – count how many times A appears before B, and vice-versa. Runs in $\mathcal{O}(n \cdot k)$.
- **Observation:** We can preprocess the ballots in $\mathcal{O}(n \cdot k)$, such that we can access the position that each candidate appears in each ballot in $\mathcal{O}(1)$. Thus, answering whether candidate A beats candidate B, now only takes $\mathcal{O}(n)$.
- **Solution:**
  - Pick a candidate $d$.
  - Go through all candidates $c_i$: Anytime $c_i$ beats $d$: $d \leftarrow c_i$. Runs in $\mathcal{O}(n \cdot k)$.
  - **Observation:** If the election has a winner, it must be $d$. (This can be proven using contradiction!)
  - Check if $d$ beats all other candidates. Runs in $\mathcal{O}(n \cdot k)$.
- **Complexity:** $\mathcal{O}(n \cdot k)$.
  - But we chose the time limit to also accept $\mathcal{O}(n \cdot k^2)$ or even $\mathcal{O}(n \cdot k^3)$.

# E: European Election

Problem author: Veselin Mitev

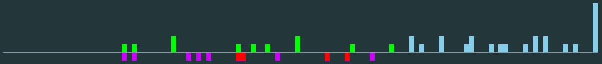


- **Problem:** Given ranked-choice ballots, determine the candidate who beats all other candidates.
- **Fun Fact:** This is also known as the *Condorcet* voting method.
- How to determine whether candidate A is better than B?
  - Go through all ballots – count how many times A appears before B, and vice-versa. Runs in $\mathcal{O}(n \cdot k)$.
- **Observation:** We can preprocess the ballots in $\mathcal{O}(n \cdot k)$, such that we can access the position that each candidate appears in each ballot in $\mathcal{O}(1)$. Thus, answering whether candidate A beats candidate B, now only takes $\mathcal{O}(n)$.
- **Solution:**
  - Pick a candidate $d$.
  - Go through all candidates $c_i$: Anytime $c_i$ beats $d$: $d \leftarrow c_i$. Runs in $\mathcal{O}(n \cdot k)$.
  - **Observation:** If the election has a winner, it must be $d$. (This can be proven using contradiction!)
  - Check if $d$ beats all other candidates. Runs in $\mathcal{O}(n \cdot k)$.
- **Complexity:** $\mathcal{O}(n \cdot k)$.
  - But we chose the time limit to also accept $\mathcal{O}(n \cdot k^2)$ or even $\mathcal{O}(n \cdot k^3)$.

Statistics: 50 submissions, 15 accepted, 21 unknown

# H: Horrendous Mistake

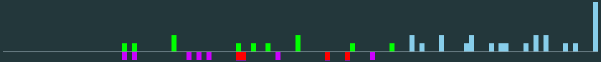
Problem author: Jeroen Op de Beek



- **Problem:** Calculate the value of the function `sum`, which uses values instead of indices.

# H: Horrendous Mistake

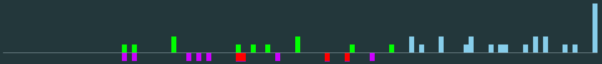
Problem author: Jeroen Op de Beek



- **Problem:** Calculate the value of the function `sum`, which uses values instead of indices.
- **Naive solution:** Simply run the function after every update. This takes $\mathcal{O}(n \cdot q)$ time, too slow!

# H: Horrendous Mistake

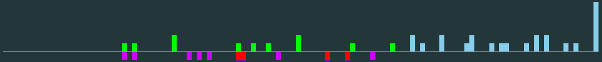
Problem author: Jeroen Op de Beek



- **Problem:** Calculate the value of the function `sum`, which uses values instead of indices.
- **Naive solution:** Simply run the function after every update. This takes $\mathcal{O}(n \cdot q)$ time, too slow!
- **Observation:** To be fast enough, every query must be processed in $\mathcal{O}(1)$.

# H: Horrendous Mistake

Problem author: Jeroen Op de Beek



- **Problem:** Calculate the value of the function `sum`, which uses values instead of indices.
- **Naive solution:** Simply run the function after every update. This takes $\mathcal{O}(n \cdot q)$ time, too slow!
- **Observation:** To be fast enough, every query must be processed in $\mathcal{O}(1)$.
- **Solution:** Do some extra bookkeeping:
  - Count how often every value occurs in the initial array ($= c_x$ for every $0 \leq x < n$).

# H: Horrendous Mistake

Problem author: Jeroen Op de Beek

- **Problem:** Calculate the value of the function `sum`, which uses values instead of indices.
- **Naive solution:** Simply run the function after every update. This takes $\mathcal{O}(n \cdot q)$ time, too slow!
- **Observation:** To be fast enough, every query must be processed in $\mathcal{O}(1)$.
- **Solution:** Do some extra bookkeeping:
  - Count how often every value occurs in the initial array ($= c_x$ for every $0 \leq x < n$).
  - Calculate the value of `sum` for the initial array and store this.

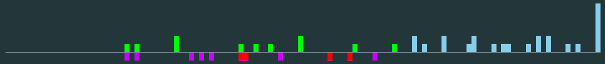
# H: Horrendous Mistake

Problem author: Jeroen Op de Beek

- **Problem:** Calculate the value of the function `sum`, which uses values instead of indices.
- **Naive solution:** Simply run the function after every update. This takes $\mathcal{O}(n \cdot q)$ time, too slow!
- **Observation:** To be fast enough, every query must be processed in $\mathcal{O}(1)$.
- **Solution:** Do some extra bookkeeping:
  - Count how often every value occurs in the initial array ($= c_x$ for every $0 \leq x < n$).
  - Calculate the value of `sum` for the initial array and store this.
  - For every update $(x, v)$ (let the old value in the array be $v_{old}$):
    - Decrement $c_{v_{old}}$.
    - Subtract $c_x \cdot v_{old} + a_{v_{old}}$ and add $c_x \cdot v + a_v$ to the stored value of `sum`.
    - Increment $c_v$.
    - Update the value in the array.

# H: Horrendous Mistake

Problem author: Jeroen Op de Beek



- **Problem:** Calculate the value of the function `sum`, which uses values instead of indices.
- **Naive solution:** Simply run the function after every update. This takes $\mathcal{O}(n \cdot q)$ time, too slow!
- **Observation:** To be fast enough, every query must be processed in $\mathcal{O}(1)$.
- **Solution:** Do some extra bookkeeping:
  - Count how often every value occurs in the initial array ($= c_x$ for every $0 \leq x < n$).
  - Calculate the value of `sum` for the initial array and store this.
  - For every update $(x, v)$ (let the old value in the array be $v_{old}$):
    - Decrement $c_{v_{old}}$.
    - Subtract $c_x \cdot v_{old} + a_{v_{old}}$ and add $c_x \cdot v + a_v$ to the stored value of `sum`.
    - Increment $c_v$.
    - Update the value in the array.
- **Complexity:** $\mathcal{O}(n + q)$.

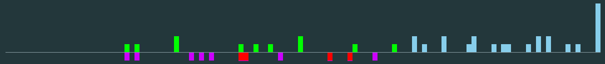
# H: Horrendous Mistake

Problem author: Jeroen Op de Beek

- **Problem:** Calculate the value of the function `sum`, which uses values instead of indices.
- **Naive solution:** Simply run the function after every update. This takes $\mathcal{O}(n \cdot q)$ time, too slow!
- **Observation:** To be fast enough, every query must be processed in $\mathcal{O}(1)$.
- **Solution:** Do some extra bookkeeping:
  - Count how often every value occurs in the initial array ($= c_x$ for every $0 \leq x < n$).
  - Calculate the value of `sum` for the initial array and store this.
  - For every update $(x, v)$ (let the old value in the array be $v_{old}$):
    - Decrement $c_{v_{old}}$.
    - Subtract $c_x \cdot v_{old} + a_{v_{old}}$ and add $c_x \cdot v + a_v$ to the stored value of `sum`.
    - Increment $c_v$.
    - Update the value in the array.
- **Complexity:** $\mathcal{O}(n + q)$.
- **Pitfall:** Beware of `int` overflow, be sure to use 64-bit integers!

# H: Horrendous Mistake

Problem author: Jeroen Op de Beek

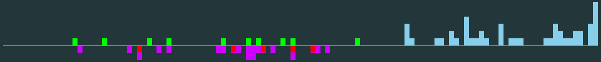


- **Problem:** Calculate the value of the function `sum`, which uses values instead of indices.
- **Naive solution:** Simply run the function after every update. This takes $\mathcal{O}(n \cdot q)$ time, too slow!
- **Observation:** To be fast enough, every query must be processed in $\mathcal{O}(1)$.
- **Solution:** Do some extra bookkeeping:
  - Count how often every value occurs in the initial array ($= c_x$ for every $0 \leq x < n$).
  - Calculate the value of `sum` for the initial array and store this.
  - For every update $(x, v)$ (let the old value in the array be $v_{old}$):
    - Decrement $c_{v_{old}}$.
    - Subtract $c_x \cdot v_{old} + a_{v_{old}}$ and add $c_x \cdot v + a_v$ to the stored value of `sum`.
    - Increment $c_v$.
    - Update the value in the array.
- **Complexity:** $\mathcal{O}(n + q)$.
- **Pitfall:** Beware of `int` overflow, be sure to use 64-bit integers!

Statistics: 46 submissions, 11 accepted, 24 unknown

# I: Intelligence Exploration

**Problem author: Makar Kuleshov**



- **Problem:** Calculate the value of the implication $a_l \to a_{l+1} \to \ldots \to a_r$ for many subarrays.

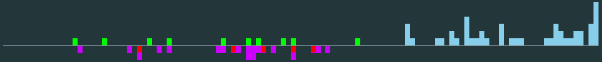
# I: Intelligence Exploration

Problem author: Makar Kuleshov

- **Problem:** Calculate the value of the implication $a_l \to a_{l+1} \to \ldots \to a_r$ for many subarrays.
- **Naive solution:** Go through the whole subarray to calculate the result. Runs in $\mathcal{O}(n \cdot q)$, too slow!

# I: Intelligence Exploration

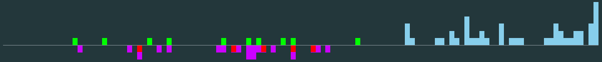
**Problem author: Makar Kuleshov**



- **Problem:** Calculate the value of the implication $a_l \to a_{l+1} \to \ldots \to a_r$ for many subarrays.
- **Naive solution:** Go through the whole subarray to calculate the result. Runs in $\mathcal{O}(n \cdot q)$, too slow!
- **Observation:** When the right argument of an implication is 1, the result is always equal to 1. So, we can look only at the last 1 in the subarray and the following zeros.

# I: Intelligence Exploration

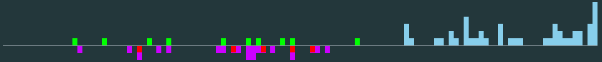
**Problem author: Makar Kuleshov**



- **Problem:** Calculate the value of the implication $a_l \to a_{l+1} \to \ldots \to a_r$ for many subarrays.
- **Naive solution:** Go through the whole subarray to calculate the result. Runs in $\mathcal{O}(n \cdot q)$, too slow!
- **Observation:** When the right argument of an implication is 1, the result is always equal to 1.
  So, we can look only at the last 1 in the subarray and the following zeros.
  $a_l \to \ldots \to 1 = 1$

# I: Intelligence Exploration

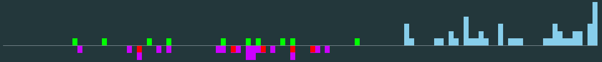
**Problem author: Makar Kuleshov**



- **Problem:** Calculate the value of the implication $a_l \to a_{l+1} \to \ldots \to a_r$ for many subarrays.
- **Naive solution:** Go through the whole subarray to calculate the result. Runs in $\mathcal{O}(n \cdot q)$, too slow!
- **Observation:** When the right argument of an implication is 1, the result is always equal to 1.
  So, we can look only at the last 1 in the subarray and the following zeros.
  $\underbrace{a_l \to \ldots \to 1}_{1} \to 0 = 0$

# I: Intelligence Exploration

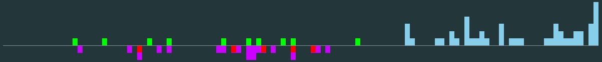
Problem author: Makar Kuleshov



- **Problem:** Calculate the value of the implication $a_l \to a_{l+1} \to \ldots \to a_r$ for many subarrays.
- **Naive solution:** Go through the whole subarray to calculate the result. Runs in $\mathcal{O}(n \cdot q)$, too slow!
- **Observation:** When the right argument of an implication is 1, the result is always equal to 1.
  So, we can look only at the last 1 in the subarray and the following zeros.
  $\underbrace{a_l \to \ldots \to 1 \to 0}_{0} \to 0 = 1$

# I: Intelligence Exploration

**Problem author: Makar Kuleshov**



- **Problem:** Calculate the value of the implication $a_l \to a_{l+1} \to \ldots \to a_r$ for many subarrays.
- **Naive solution:** Go through the whole subarray to calculate the result. Runs in $\mathcal{O}(n \cdot q)$, too slow!
- **Observation:** When the right argument of an implication is 1, the result is always equal to 1. So, we can look only at the last 1 in the subarray and the following zeros.

  $\underbrace{a_l \to \ldots \to 1 \to 0 \to 0}_{1} \to 0 = 0$

# I: Intelligence Exploration

Problem author: Makar Kuleshov

- **Problem:** Calculate the value of the implication $a_l \to a_{l+1} \to \ldots \to a_r$ for many subarrays.
- **Naive solution:** Go through the whole subarray to calculate the result. Runs in $\mathcal{O}(n \cdot q)$, too slow!
- **Observation:** When the right argument of an implication is 1, the result is always equal to 1.

  So, we can look only at the last 1 in the subarray and the following zeros.

  $a_l \to \ldots \to 1 \to \underbrace{0 \to \ldots \to 0}_{k \text{ zeros}}$

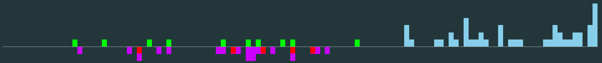
  If $k$ is even then the result equals 1.

  If $k$ is odd then the result equals 0.

- **Problem:** Calculate the value of the implication $a_l \to a_{l+1} \to \ldots \to a_r$ for many subarrays.
- **Naive solution:** Go through the whole subarray to calculate the result. Runs in $\mathcal{O}(n \cdot q)$, too slow!
- **Observation:** When the right argument of an implication is 1, the result is always equal to 1.
  So, we can look only at the last 1 in the subarray and the following zeros.
  $a_l \to \ldots \to 1 \to \underbrace{0 \to \ldots \to 0}_{k \text{ zeros}}$
  If $k$ is even then the result equals 1.
  If $k$ is odd then the result equals 0.
- **Solution:** For each position precompute the index of the last 1 appearing not after it.
  This way you can determine the number of zeros in the end of a subarray in $\mathcal{O}(1)$.

# I: Intelligence Exploration

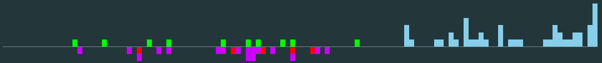
**Problem author: Makar Kuleshov**



- **Problem:** Calculate the value of the implication $a_l \to a_{l+1} \to \ldots \to a_r$ for many subarrays.
- **Naive solution:** Go through the whole subarray to calculate the result. Runs in $\mathcal{O}(n \cdot q)$, too slow!
- **Observation:** When the right argument of an implication is 1, the result is always equal to 1.
  So, we can look only at the last 1 in the subarray and the following zeros.
  $a_l \to \ldots \to 1 \to \underbrace{0 \to \ldots \to 0}_{k \text{ zeros}}$
  If $k$ is even then the result equals 1.
  If $k$ is odd then the result equals 0.
- **Solution:** For each position precompute the index of the last 1 appearing not after it.
  This way you can determine the number of zeros in the end of a subarray in $\mathcal{O}(1)$.
- **Complexity:** $\mathcal{O}(n + q)$

# I: Intelligence Exploration

Problem author: Makar Kuleshov

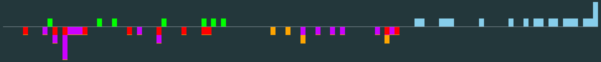


- **Problem:** Calculate the value of the implication $a_l \to a_{l+1} \to \ldots \to a_r$ for many subarrays.
- **Naive solution:** Go through the whole subarray to calculate the result. Runs in $\mathcal{O}(n \cdot q)$, too slow!
- **Observation:** When the right argument of an implication is 1, the result is always equal to 1.
  So, we can look only at the last 1 in the subarray and the following zeros.
  $a_l \to \ldots \to 1 \to \underbrace{0 \to \ldots \to 0}_{k \text{ zeros}}$
  If $k$ is even then the result equals 1.
  If $k$ is odd then the result equals 0.
- **Solution:** For each position precompute the index of the last 1 appearing not after it.
  This way you can determine the number of zeros in the end of a subarray in $\mathcal{O}(1)$.
- **Complexity:** $\mathcal{O}(n + q)$

Statistics: 78 submissions, 10 accepted, 46 unknown

# A: Annoying Alliterations

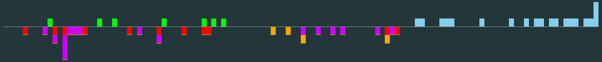
Problem author: Maciek Sidor



- **Problem:** Given $n$ words, find a pair such that after their common prefix is removed, the sum of lengths of the two resulting words is the greatest.

# A: Annoying Alliterations

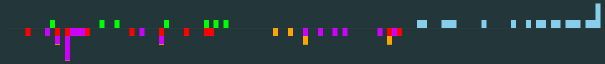
Problem author: Maciek Sidor



- **Problem:** Given $n$ words, find a pair such that after their common prefix is removed, the sum of lengths of the two resulting words is the greatest.
- **Naive solution:** Check every pair. Runs in $\mathcal{O}(n^2)$, too slow.

# A: Annoying Alliterations

Problem author: Maciek Sidor



- **Problem:** Given $n$ words, find a pair such that after their common prefix is removed, the sum of lengths of the two resulting words is the greatest.
- **Naive solution:** Check every pair. Runs in $\mathcal{O}(n^2)$, too slow.
- **Claim:** For a given pair $s, t$ and a third word $v$ such that $|v| \geq \max(|s|, |t|)$, we can always replace one of the words and the score will not decrease.

# A: Annoying Alliterations

**Problem author: Maciek Sidor**

- **Claim:** For a given pair $s, t$ and a third word $v$ such that $|v| \geq \max(|s|, |t|)$, we can always replace one of the words and the score will not decrease.

- **Claim:** For a given pair $s, t$ and a third word $v$ such that $|v| \geq \max(|s|, |t|)$, we can always replace one of the words and the score will not decrease.
- **Proof:** Denote the common prefix of $s, t$ as $p(s, t)$ and let $g(s, t) = |s| + |t| - 2|p(s, t)|$ be our goodness function.

- **Claim:** For a given pair $s, t$ and a third word $v$ such that $|v| \geq \max(|s|, |t|)$, we can always replace one of the words and the score will not decrease.
- **Proof:** Denote the common prefix of $s, t$ as $p(s, t)$ and let $g(s, t) = |s| + |t| - 2|p(s, t)|$ be our goodness function.
- Suppose $g(s, t) > g(s, v)$ and $g(s, t) > g(v, t)$. Then:

# A: Annoying Alliterations

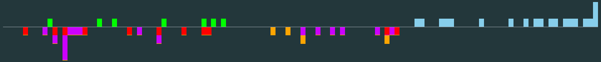
Problem author: Maciek Sidor

- **Claim:** For a given pair $s, t$ and a third word $v$ such that $|v| \geq \max(|s|, |t|)$, we can always replace one of the words and the score will not decrease.
- **Proof:** Denote the common prefix of $s, t$ as $p(s, t)$ and let $g(s, t) = |s| + |t| - 2|p(s, t)|$ be our goodness function.
- Suppose $g(s, t) > g(s, v)$ and $g(s, t) > g(v, t)$. Then:

$$|s| + |t| - 2|p(s, t)| > |s| + |v| - 2|p(s, v)|$$

# A: Annoying Alliterations

Problem author: Maciek Sidor



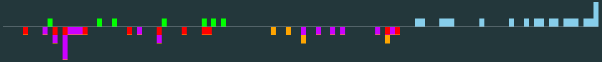
- **Claim:** For a given pair $s, t$ and a third word $v$ such that $|v| \geq \max(|s|, |t|)$, we can always replace one of the words and the score will not decrease.
- **Proof:** Denote the common prefix of $s, t$ as $p(s, t)$ and let $g(s, t) = |s| + |t| - 2|p(s, t)|$ be our goodness function.
- Suppose $g(s, t) > g(s, v)$ and $g(s, t) > g(v, t)$. Then:

$$|s| + |t| - 2|p(s, t)| > |s| + |v| - 2|p(s, v)|$$

$$|t| - 2|p(s, t)| > |v| - 2|p(s, v)|$$

# A: Annoying Alliterations

Problem author: Maciek Sidor



- **Claim:** For a given pair $s, t$ and a third word $v$ such that $|v| \geq \max(|s|, |t|)$, we can always replace one of the words and the score will not decrease.
- **Proof:** Denote the common prefix of $s, t$ as $p(s, t)$ and let $g(s, t) = |s| + |t| - 2|p(s, t)|$ be our goodness function.
- Suppose $g(s, t) > g(s, v)$ and $g(s, t) > g(v, t)$. Then:

$$|s| + |t| - 2|p(s, t)| > |s| + |v| - 2|p(s, v)|$$

$$|t| - 2|p(s, t)| > |v| - 2|p(s, v)|$$

$$|p(s, v)| > |p(s, t)|$$

- **Claim:** For a given pair $s, t$ and a third word $v$ such that $|v| \geq \max(|s|, |t|)$, we can always replace one of the words and the score will not decrease.
- **Proof:** Denote the common prefix of $s, t$ as $p(s, t)$ and let $g(s, t) = |s| + |t| - 2|p(s, t)|$ be our goodness function.
- Suppose $g(s, t) > g(s, v)$ and $g(s, t) > g(v, t)$. Then:

$$|s| + |t| - 2|p(s, t)| > |s| + |v| - 2|p(s, v)|$$
$$|t| - 2|p(s, t)| > |v| - 2|p(s, v)|$$
$$|p(s, v)| > |p(s, t)|$$

- Similarly, $|p(v, t)| > |p(s, t)|$, but these two together give us a contradiction. $\square$

- **Claim:** For a given pair $s, t$ and a third word $v$ such that $|v| \geq \max(|s|, |t|)$, we can always replace one of the words and the score will not decrease.

- **Claim:** For a given pair $s, t$ and a third word $v$ such that $|v| \geq \max(|s|, |t|)$, we can always replace one of the words and the score will not decrease.
- **Therefore:** Any word of maximum length is part of a valid solution.

# A: Annoying Alliterations

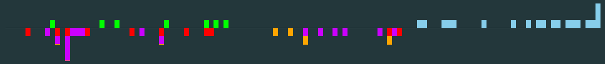
Problem author: Maciek Sidor

- **Claim:** For a given pair $s, t$ and a third word $v$ such that $|v| \geq \max(|s|, |t|)$, we can always replace one of the words and the score will not decrease.
- **Therefore:** Any word of maximum length is part of a valid solution.
- **Solution:** Pick any word of maximum length and check it with every other word, take the maximum result.

- **Claim:** For a given pair $s, t$ and a third word $v$ such that $|v| \geq \max(|s|, |t|)$, we can always replace one of the words and the score will not decrease.
- **Therefore:** Any word of maximum length is part of a valid solution.
- **Solution:** Pick any word of maximum length and check it with every other word, take the maximum result.
- **Complexity:** $\mathcal{O}(n)$.

# A: Annoying Alliterations

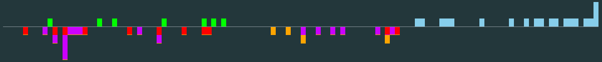
Problem author: Maciek Sidor



- **Claim:** For a given pair $s, t$ and a third word $v$ such that $|v| \geq \max(|s|, |t|)$, we can always replace one of the words and the score will not decrease.
- **Therefore:** Any word of maximum length is part of a valid solution.
- **Solution:** Pick any word of maximum length and check it with every other word, take the maximum result.
- **Complexity:** $\mathcal{O}(n)$.
- **Note:** Can also be solved using a trie (also known as a prefix tree).

# A: Annoying Alliterations

Problem author: Maciek Sidor



- **Claim:** For a given pair $s, t$ and a third word $v$ such that $|v| \geq \max(|s|, |t|)$, we can always replace one of the words and the score will not decrease.
- **Therefore:** Any word of maximum length is part of a valid solution.
- **Solution:** Pick any word of maximum length and check it with every other word, take the maximum result.
- **Complexity:** $\mathcal{O}(n)$.
- **Note:** Can also be solved using a trie (also known as a prefix tree).

Statistics: 58 submissions, 7 accepted, 20 unknown

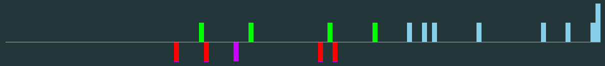
# J: Jailbreak

Problem author: Wietze Koops

- **Problem:** Escape from a $w \times h$ grid jail where you can go up only if you have a ladder. Ladders can be carried to a different place on the same storey.

# J: Jailbreak

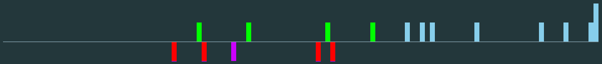
Problem author: Wietze Koops



- **Problem:** Escape from a $w \times h$ grid jail where you can go up only if you have a ladder. Ladders can be carried to a different place on the same storey.
- **Observation:** If we know to which holes a ladder can be carried, then for each cell, we know which cell we can move to.

# J: Jailbreak

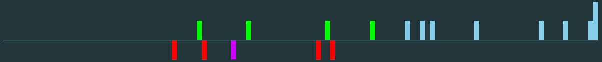
**Problem author: Wietze Koops**



- **Problem:** Escape from a $w \times h$ grid jail where you can go up only if you have a ladder. Ladders can be carried to a different place on the same storey.
- **Observation:** If we know to which holes a ladder can be carried, then for each cell, we know which cell we can move to.
- **Solution:**
  - Using a for loop in both directions, determine which cells can access a ladder.

# J: Jailbreak

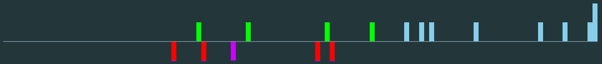
Problem author: Wietze Koops



- **Problem:** Escape from a $w \times h$ grid jail where you can go up only if you have a ladder. Ladders can be carried to a different place on the same storey.
- **Observation:** If we know to which holes a ladder can be carried, then for each cell, we know which cell we can move to.
- **Solution:**
  - Using a for loop in both directions, determine which cells can access a ladder.
  - Then we know for each cell to which cell we can move.
  - Hence, we can define a graph representing the grid.

# J: Jailbreak

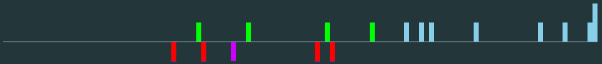
Problem author: Wietze Koops



- **Problem:** Escape from a $w \times h$ grid jail where you can go up only if you have a ladder. Ladders can be carried to a different place on the same storey.
- **Observation:** If we know to which holes a ladder can be carried, then for each cell, we know which cell we can move to.
- **Solution:**
  - Using a for loop in both directions, determine which cells can access a ladder.
  - Then we know for each cell to which cell we can move.
  - Hence, we can define a graph representing the grid.
  - Determining whether a path exists from the starting cell to an exit can be done using $\mathcal{O}(wh)$ BFS/DFS.

# J: Jailbreak

Problem author: Wietze Koops



- **Problem:** Escape from a $w \times h$ grid jail where you can go up only if you have a ladder. Ladders can be carried to a different place on the same storey.
- **Observation:** If we know to which holes a ladder can be carried, then for each cell, we know which cell we can move to.
- **Solution:**
  - Using a for loop in both directions, determine which cells can access a ladder.
  - Then we know for each cell to which cell we can move.
  - Hence, we can define a graph representing the grid.
  - Determining whether a path exists from the starting cell to an exit can be done using $\mathcal{O}(wh)$ BFS/DFS.

Statistics: 18 submissions, 4 accepted, 9 unknown

# C: Curious Jury

Problem author: Jeroen Op de Beek

- **Problem:** Given two types of penalty times for $n$ teams ($1 \leq l_i < s_i \leq n$), find out over all ways of choosing the type of penalty time for each team, how many fixed points the scoreboard contains in total.

# C: Curious Jury

Problem author: Jeroen Op de Beek

- **Problem:** Given two types of penalty times for $n$ teams ($1 \leq l_i < s_i \leq n$), find out over all ways of choosing the type of penalty time for each team, how many fixed points the scoreboard contains in total.
- **Observation 1:** Instead of finding fixed points for each out of $2^n$ options, find how many times team $i$ is a fixed point.

# C: Curious Jury

Problem author: Jeroen Op de Beek

- **Problem:** Given two types of penalty times for $n$ teams $(1 \leq l_i < s_i \leq n)$, find out over all ways of choosing the type of penalty time for each team, how many fixed points the scoreboard contains in total.
- **Observation 1:** Instead of finding fixed points for each out of $2^n$ options, find how many times team $i$ is a fixed point.
- **Observation 2:** Loop over all teams, try both options $l_i$ and $s_i$ as potential fixed point, for team $i$, call this potential fixed point $f$.

# C: Curious Jury

Problem author: Jeroen Op de Beek

- **Problem:** Given two types of penalty times for $n$ teams $(1 \leq l_i < s_i \leq n)$, find out over all ways of choosing the type of penalty time for each team, how many fixed points the scoreboard contains in total.
- **Observation 1:** Instead of finding fixed points for each out of $2^n$ options, find how many times team $i$ is a fixed point.
- **Observation 2:** Loop over all teams, try both options $l_i$ and $s_i$ as potential fixed point, for team $i$, call this potential fixed point $f$.
- **Observation 3:** Other teams form 3 groups:
  - **A** Teams with $l_j < f, s_j < f$
  - **B** Teams with $l_j < f, s_j \geq f$
  - **C** Teams with $l_j \geq f, s_j \geq f$

# C: Curious Jury

Problem author: Jeroen Op de Beek

- **Problem:** Given two types of penalty times for $n$ teams $(1 \le l_i < s_i \le n)$, find out over all ways of choosing the type of penalty time for each team, how many fixed points the scoreboard contains in total.
- **Observation 1:** Instead of finding fixed points for each out of $2^n$ options, find how many times team $i$ is a fixed point.
- **Observation 2:** Loop over all teams, try both options $l_i$ and $s_i$ as potential fixed point, for team $i$, call this potential fixed point $f$.
- **Observation 3:** Other teams form 3 groups:
  - **A** Teams with $l_j < f, s_j < f$
  - **B** Teams with $l_j < f, s_j \ge f$
  - **C** Teams with $l_j \ge f, s_j \ge f$
- **Observation 4:** The number of ways to choose the other submission times, for team $i$ to have a fixed point at rank $f$: $2^{|A|+|C|} \cdot \binom{|B|}{f-|A|}$

# C: Curious Jury

Problem author: Jeroen Op de Beek

- **Problem:** Given two types of penalty times for $n$ teams ($1 \leq l_i < s_i \leq n$), find out over all ways of choosing the type of penalty time for each team, how many fixed points the scoreboard contains in total.
- **Observation 1:** Instead of finding fixed points for each out of $2^n$ options, find how many times team $i$ is a fixed point.
- **Observation 2:** Loop over all teams, try both options $l_i$ and $s_i$ as potential fixed point, for team $i$, call this potential fixed point $f$.
- **Observation 3:** Other teams form 3 groups:
  - **A** Teams with $l_j < f, s_j < f$
  - **B** Teams with $l_j < f, s_j \geq f$
  - **C** Teams with $l_j \geq f, s_j \geq f$
- **Observation 4:** The number of ways to choose the other submission times, for team $i$ to have a fixed point at rank $f$: $2^{|A|+|C|} \cdot \binom{|B|}{f-|A|}$
- Team $j$ is in group **A** if $s_j < f$.

# C: Curious Jury

Problem author: Jeroen Op de Beek

- **Problem:** Given two types of penalty times for $n$ teams $(1 \leq l_i < s_i \leq n)$, find out over all ways of choosing the type of penalty time for each team, how many fixed points the scoreboard contains in total.
- **Observation 1:** Instead of finding fixed points for each out of $2^n$ options, find how many times team $i$ is a fixed point.
- **Observation 2:** Loop over all teams, try both options $l_i$ and $s_i$ as potential fixed point, for team $i$, call this potential fixed point $f$.
- **Observation 3:** Other teams form 3 groups:
  - **A** Teams with $l_j < f, s_j < f$
  - **B** Teams with $l_j < f, s_j \geq f$
  - **C** Teams with $l_j \geq f, s_j \geq f$
- **Observation 4:** The number of ways to choose the other submission times, for team $i$ to have a fixed point at rank $f$: $2^{|A|+|C|} \cdot \binom{|B|}{f-|A|}$
- Team $j$ is in group **A** if $s_j < f$.
- Team $j$ is in group **C** if $l_j \geq f$.

# C: Curious Jury

Problem author: Jeroen Op de Beek

- **Problem:** Given two types of penalty times for $n$ teams ($1 \leq l_i < s_i \leq n$), find out over all ways of choosing the type of penalty time for each team, how many fixed points the scoreboard contains in total.
- **Observation 1:** Instead of finding fixed points for each out of $2^n$ options, find how many times team $i$ is a fixed point.
- **Observation 2:** Loop over all teams, try both options $l_i$ and $s_i$ as potential fixed point, for team $i$, call this potential fixed point $f$.
- **Observation 3:** Other teams form 3 groups:
  - **A** Teams with $l_j < f, s_j < f$
  - **B** Teams with $l_j < f, s_j \geq f$
  - **C** Teams with $l_j \geq f, s_j \geq f$
- **Observation 4:** The number of ways to choose the other submission times, for team $i$ to have a fixed point at rank $f$: $2^{|A|+|C|} \cdot \binom{|B|}{f-|A|}$
- Team $j$ is in group **A** if $s_j < f$.
- Team $j$ is in group **C** if $l_j \geq f$.
- Otherwise, team $j$ is in **B**. By sorting the $l_j$ and $s_j$ arrays, $|A|$ and $|C|$ can be found by binary search.

## C: Curious Jury

Problem author: Jeroen Op de Beek

- **Problem:** Given two types of penalty times for $n$ teams ($1 \leq l_i < s_i \leq n$), find out over all ways of choosing the type of penalty time for each team, how many fixed points the scoreboard contains in total.
- **Observation 4:** The number of ways to choose the other submission times, for team $i$ to have a fixed point at rank $f$: $2^{|A|+|C|} \cdot \binom{|B|}{f-|A|}$
- Need to calculate $\mathcal{O}(n)$ binomial coefficients $\binom{a}{b}$, with $0 \leq a, b \leq n$: $\binom{a}{b} = \frac{a!}{b!(a-b)!}$ and $2^a$ for $0 \leq a \leq n$, both $\mod (10^9 + 7)$

## C: Curious Jury

Problem author: Jeroen Op de Beek

- **Problem:** Given two types of penalty times for $n$ teams $(1 \leq l_i < s_i \leq n)$, find out over all ways of choosing the type of penalty time for each team, how many fixed points the scoreboard contains in total.
- **Observation 4:** The number of ways to choose the other submission times, for team $i$ to have a fixed point at rank $f$: $2^{|A|+|C|} \cdot \binom{|B|}{f-|A|}$
- Need to calculate $\mathcal{O}(n)$ binomial coefficients $\binom{a}{b}$, with $0 \leq a, b \leq n$: $\binom{a}{b} = \frac{a!}{b!(a-b)!}$ and $2^a$ for $0 \leq a \leq n$, both $\mod (10^9 + 7)$
- Can precalculate factorial[$k$] and twopower[$k$] in $\mathcal{O}(n)$.

# C: Curious Jury

Problem author: Jeroen Op de Beek

- **Problem:** Given two types of penalty times for $n$ teams $(1 \leq l_i < s_i \leq n)$, find out over all ways of choosing the type of penalty time for each team, how many fixed points the scoreboard contains in total.
- **Observation 4:** The number of ways to choose the other submission times, for team $i$ to have a fixed point at rank $f$: $2^{|A|+|C|} \cdot \binom{|B|}{f-|A|}$
- Need to calculate $\mathcal{O}(n)$ binomial coefficients $\binom{a}{b}$, with $0 \leq a, b \leq n$: $\binom{a}{b} = \frac{a!}{b!(a-b)!}$ and $2^a$ for $0 \leq a \leq n$, both $\mod (10^9 + 7)$
- Can precalculate factorial[$k$] and twopower[$k$] in $\mathcal{O}(n)$.
- Can find inverse of factorial[$n$] in $\mathcal{O}(\log(MOD))$ (or if you don't know how to calculate a modular inverse, you can bruteforce it on your own computer).

# C: Curious Jury

Problem author: Jeroen Op de Beek

- **Problem:** Given two types of penalty times for $n$ teams ($1 \leq l_i < s_i \leq n$), find out over all ways of choosing the type of penalty time for each team, how many fixed points the scoreboard contains in total.
- **Observation 4:** The number of ways to choose the other submission times, for team $i$ to have a fixed point at rank $f$: $2^{|A|+|C|} \cdot \binom{|B|}{f-|A|}$
- Need to calculate $\mathcal{O}(n)$ binomial coefficients $\binom{a}{b}$, with $0 \leq a, b \leq n$: $\binom{a}{b} = \frac{a!}{b!(a-b)!}$ and $2^a$ for $0 \leq a \leq n$, both $\mod (10^9 + 7)$
- Can precalculate factorial[$k$] and twopower[$k$] in $\mathcal{O}(n)$.
- Can find inverse of factorial[$n$] in $\mathcal{O}(\log(MOD))$ (or if you don't know how to calculate a modular inverse, you can bruteforce it on your own computer).
- Now fill the array invfactorial[$k$] using invfactorial[$k$] $=$ invfactorial[$k+1$] $\cdot (k+1)$ in $\mathcal{O}(n)$.

# C: Curious Jury

Problem author: Jeroen Op de Beek

- **Problem:** Given two types of penalty times for $n$ teams $(1 \leq l_i < s_i \leq n)$, find out over all ways of choosing the type of penalty time for each team, how many fixed points the scoreboard contains in total.
- **Observation 4:** The number of ways to choose the other submission times, for team $i$ to have a fixed point at rank $f$: $2^{|A|+|C|} \cdot \binom{|B|}{f-|A|}$
- Need to calculate $\mathcal{O}(n)$ binomial coefficients $\binom{a}{b}$, with $0 \leq a, b \leq n$: $\binom{a}{b} = \frac{a!}{b!(a-b)!}$ and $2^a$ for $0 \leq a \leq n$, both $\mod (10^9 + 7)$
- Can precalculate factorial$[k]$ and twopower$[k]$ in $\mathcal{O}(n)$.
- Can find inverse of factorial$[n]$ in $\mathcal{O}(\log(MOD))$ (or if you don't know how to calculate a modular inverse, you can bruteforce it on your own computer).
- Now fill the array invfactorial$[k]$ using invfactorial$[k]$ = invfactorial$[k+1] \cdot (k+1)$ in $\mathcal{O}(n)$.
- Complexity varying from $\mathcal{O}(n(\log(n) + \log(MOD)))$ to $\mathcal{O}(n + \log(MOD))$ depending on exact implementation.

# C: Curious Jury

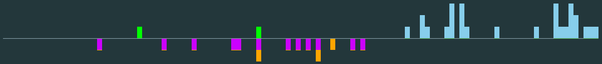
Problem author: Jeroen Op de Beek

- **Problem:** Given two types of penalty times for $n$ teams ($1 \leq l_i < s_i \leq n$), find out over all ways of choosing the type of penalty time for each team, how many fixed points the scoreboard contains in total.
- **Observation 4:** The number of ways to choose the other submission times, for team $i$ to have a fixed point at rank $f$: $2^{|A|+|C|} \cdot \binom{|B|}{f-|A|}$
- Need to calculate $\mathcal{O}(n)$ binomial coefficients $\binom{a}{b}$, with $0 \leq a, b \leq n$: $\binom{a}{b} = \frac{a!}{b!(a-b)!}$ and $2^a$ for $0 \leq a \leq n$, both $\mod (10^9 + 7)$
- Can precalculate factorial[$k$] and twopower[$k$] in $\mathcal{O}(n)$.
- Can find inverse of factorial[$n$] in $\mathcal{O}(\log(MOD))$ (or if you don't know how to calculate a modular inverse, you can bruteforce it on your own computer).
- Now fill the array invfactorial[$k$] using invfactorial[$k$] $=$ invfactorial[$k+1$] $\cdot (k+1)$ in $\mathcal{O}(n)$.
- Complexity varying from $\mathcal{O}(n(\log(n) + \log(MOD)))$ to $\mathcal{O}(n + \log(MOD))$ depending on exact implementation.

Statistics: 6 submissions, 3 accepted, 2 unknown

# K: Kangaroo Race

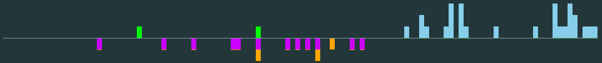
Problem author: Leon van der Waal



- **Problem:** A kangeroo jumps from $x$ to $x + x(x-1) = x^2$ in one step. How many steps until it reaches 1?

# K: Kangaroo Race

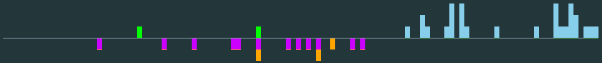
Problem author: Leon van der Waal



- **Problem:** A kangeroo jumps from $x$ to $x + x(x-1) = x^2$ in one step. How many steps until it reaches 1?
- Notice that when a kangeroo jumps over the $n$-th segment, it jumps to $x^2 \mod n$.

# K: Kangaroo Race

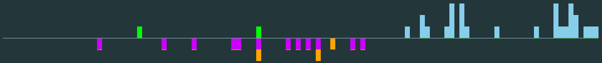
Problem author: Leon van der Waal



- **Problem:** A kangeroo jumps from $x$ to $x + x(x-1) = x^2$ in one step. How many steps until it reaches 1?
- Notice that when a kangeroo jumps over the $n$-th segment, it jumps to $x^2 \mod n$.
- So after $i$ jumps, the kangeroo is in segment $x^{2^i} \mod n$.

# K: Kangaroo Race

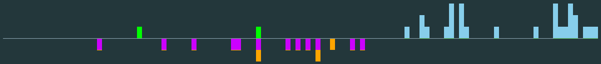
Problem author: Leon van der Waal



- **Problem:** A kangeroo jumps from $x$ to $x + x(x-1) = x^2$ in one step. How many steps until it reaches 1?
- Notice that when a kangeroo jumps over the $n$-th segment, it jumps to $x^2 \mod n$.
- So after $i$ jumps, the kangeroo is in segment $x^{2^i} \mod n$.
- Therefore we need to determine the first $i$ such that $x^{2^i} \equiv 1 \mod n$.

# K: Kangaroo Race

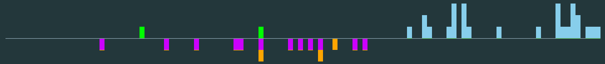
Problem author: Leon van der Waal



- **Problem:** Determine the first $i$ such that $x^{2^i} \equiv 1 \mod n$.

# K: Kangaroo Race

Problem author: Leon van der Waal

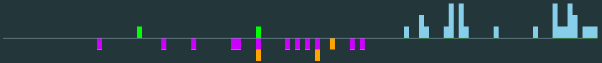


- **Problem:** Determine the first $i$ such that $x^{2^i} \equiv 1 \mod n$.
- **Observation:** If $\gcd(x, n) \neq 1$, then $\gcd(x, n) \mid x^{2^i} \mod n$, so the kangeroo will never reach 1.

- **Problem:** Determine the first $i$ such that $x^{2^i} \equiv 1 \mod n$.
- **Observation:** If $\gcd(x, n) \neq 1$, then $\gcd(x, n) \mid x^{2^i} \mod n$, so the kangeroo will never reach 1.
- Otherwise, $x^r \equiv 1 \mod n$ for some $r$. We call the smallest such $r$ the *order* of $x$ mod $n$.

# K: Kangaroo Race

**Problem author: Leon van der Waal**

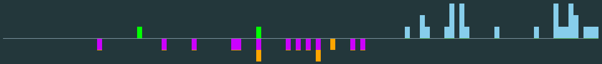


- **Problem:** Determine the first $i$ such that $x^{2^i} \equiv 1 \mod n$.
- **Observation:** If $\gcd(x, n) \neq 1$, then $\gcd(x, n) \mid x^{2^i} \mod n$, so the kangeroo will never reach 1.
- Otherwise, $x^r \equiv 1 \mod n$ for some $r$. We call the smallest such $r$ the *order* of $x$ mod $n$.
- Notice that the powers of $x$ repeat every $r$-th power:

$$1, x, x^2, x^3, \dots, x^{r-1}, x^r = 1, x^{r+1} = x, x^2, x^3, \dots$$

# K: Kangaroo Race

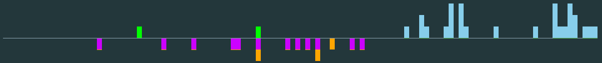
Problem author: Leon van der Waal



- **Problem:** What is the first $i$ such that $x^{2^i} \equiv 1 \mod n$.
- Therefore, any $i$ that satisfies $x^{2^i} \equiv 1 \mod n$ also satisfies $2^i \equiv 0 \mod r$.

# K: Kangaroo Race

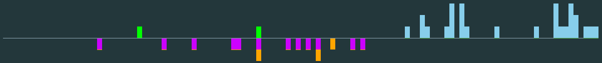
Problem author: Leon van der Waal



- **Problem:** What is the first $i$ such that $x^{2^i} \equiv 1 \mod n$.
- Therefore, any $i$ that satisfies $x^{2^i} \equiv 1 \mod n$ also satisfies $2^i \equiv 0 \mod r$.
- **Observation:** This means that $r$ is a divisor of $2^i$, and thus $r = 2^k$ for some $k$.

# K: Kangaroo Race

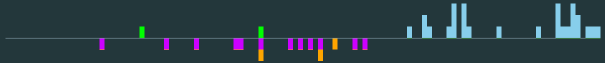
**Problem author: Leon van der Waal**



- **Problem:** What is the first $i$ such that $x^{2^i} \equiv 1 \mod n$.
- Therefore, any $i$ that satisfies $x^{2^i} \equiv 1 \mod n$ also satisfies $2^i \equiv 0 \mod r$.
- **Observation:** This means that $r$ is a divisor of $2^i$, and thus $r = 2^k$ for some $k$.
- Therefore, the answer to the problem is $k$.

# K: Kangaroo Race

Problem author: Leon van der Waal



- **Problem:** What is the first $i$ such that $x^{2^i} \equiv 1 \mod n$.
- Therefore, any $i$ that satisfies $x^{2^i} \equiv 1 \mod n$ also satisfies $2^i \equiv 0 \mod r$.
- **Observation:** This means that $r$ is a divisor of $2^i$, and thus $r = 2^k$ for some $k$.
- Therefore, the answer to the problem is $k$.
- **Observation:** $r \leq n$, so $k \leq \log_2(n)$

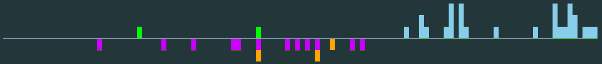
# K: Kangaroo Race

Problem author: Leon van der Waal

- **Problem:** What is the first $i$ such that $x^{2^i} \equiv 1 \mod n$.
- **Observation:** $r \leq n$, so $k \leq \log_2(n)$

# K: Kangaroo Race

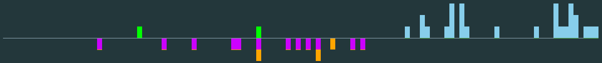
Problem author: Leon van der Waal



- **Problem:** What is the first $i$ such that $x^{2^i} \equiv 1 \mod n$.
- **Observation:** $r \leq n$, so $k \leq \log_2(n)$
- **Solution:** It therefore suffices to check the first $\log_2(n) < 60$ jumps. If the kangeroo has not reached segment 1 by then, it never will.

# K: Kangaroo Race

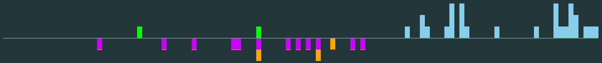
Problem author: Leon van der Waal



- **Problem:** What is the first $i$ such that $x^{2^i} \equiv 1 \mod n$.
- **Observation:** $r \leq n$, so $k \leq \log_2(n)$
- **Solution:** It therefore suffices to check the first $\log_2(n) < 60$ jumps. If the kangeroo has not reached segment 1 by then, it never will.
- **Complexity:** $\mathcal{O}(q \log n)$

# K: Kangaroo Race

Problem author: Leon van der Waal



- **Problem:** What is the first $i$ such that $x^{2^i} \equiv 1 \mod n$.
- **Observation:** $r \leq n$, so $k \leq \log_2(n)$
- **Solution:** It therefore suffices to check the first $\log_2(n) < 60$ jumps. If the kangeroo has not reached segment 1 by then, it never will.
- **Complexity:** $\mathcal{O}(q \log n)$

Statistics: 44 submissions, 2 accepted, 27 unknown

# G: Galactic Expedition

**Problem author: Veselin Mitev**

- **Problem:** Navigate between wormholes to find the ancient relic, without running out of fuel.

# G: Galactic Expedition

Problem author: Veselin Mitev

- **Problem:** Navigate between wormholes to find the ancient relic, without running out of fuel.
- **Observation:** You can refuel more than enough times to simply explore all wormholes, until you find a way to reach the relic.

# G: Galactic Expedition

Problem author: Veselin Mitev

- **Problem:** Navigate between wormholes to find the ancient relic, without running out of fuel.
- **Observation:** You can refuel more than enough times to simply explore all wormholes, until you find a way to reach the relic.
- **Solution:** Perform a “live” search – explore the wormholes while always keeping enough fuel ($\frac{d}{2}$) to go back to home base:

# G: Galactic Expedition

Problem author: Veselin Mitev

- **Problem:** Navigate between wormholes to find the ancient relic, without running out of fuel.
- **Observation:** You can refuel more than enough times to simply explore all wormholes, until you find a way to reach the relic.
- **Solution:** Perform a “live” search – explore the wormholes while always keeping enough fuel ($\frac{d}{2}$) to go back to home base:
  - If you can reach the relic within the fuel limit, do that.

# G: Galactic Expedition

Problem author: Veselin Mitev

- **Problem:** Navigate between wormholes to find the ancient relic, without running out of fuel.
- **Observation:** You can refuel more than enough times to simply explore all wormholes, until you find a way to reach the relic.
- **Solution:** Perform a "live" search – explore the wormholes while always keeping enough fuel ($\frac{d}{2}$) to go back to home base:
  - If you can reach the relic within the fuel limit, do that.
  - Find the closest unexplored wormhole:
    - You can do that using **Dijkstra**, or **Floyd-Warshall**.

# G: Galactic Expedition

Problem author: Veselin Mitev

- **Problem:** Navigate between wormholes to find the ancient relic, without running out of fuel.
- **Observation:** You can refuel more than enough times to simply explore all wormholes, until you find a way to reach the relic.
- **Solution:** Perform a "live" search – explore the wormholes while always keeping enough fuel ($\frac{d}{2}$) to go back to home base:
  - If you can reach the relic within the fuel limit, do that.
  - Find the closest unexplored wormhole:
    - You can do that using **Dijkstra**, or **Floyd-Warshall**.
  - Can we reach it while still having enough fuel to go back to home base?

# G: Galactic Expedition

Problem author: Veselin Mitev

- **Problem:** Navigate between wormholes to find the ancient relic, without running out of fuel.
- **Observation:** You can refuel more than enough times to simply explore all wormholes, until you find a way to reach the relic.
- **Solution:** Perform a "live" search – explore the wormholes while always keeping enough fuel ($\frac{d}{2}$) to go back to home base:
  - If you can reach the relic within the fuel limit, do that.
  - Find the closest unexplored wormhole:
    - You can do that using **Dijkstra**, or **Floyd-Warshall**.
  - Can we reach it while still having enough fuel to go back to home base?
    - If yes: Go to that wormhole and update the distances between the points.

# G: Galactic Expedition

Problem author: Veselin Mitev

- **Problem:** Navigate between wormholes to find the ancient relic, without running out of fuel.
- **Observation:** You can refuel more than enough times to simply explore all wormholes, until you find a way to reach the relic.
- **Solution:** Perform a “live” search – explore the wormholes while always keeping enough fuel ($\frac{d}{2}$) to go back to home base:
  - If you can reach the relic within the fuel limit, do that.
  - Find the closest unexplored wormhole:
    - You can do that using **Dijkstra**, or **Floyd-Warshall**.
  - Can we reach it while still having enough fuel to go back to home base?
    - If yes: Go to that wormhole and update the distances between the points.
    - If no: Go back to home base and refuel.

# G: Galactic Expedition

Problem author: Veselin Mitev

- **Problem:** Navigate between wormholes to find the ancient relic, without running out of fuel.
- **Observation:** You can refuel more than enough times to simply explore all wormholes, until you find a way to reach the relic.
- **Solution:** Perform a "live" search – explore the wormholes while always keeping enough fuel ($\frac{d}{2}$) to go back to home base:
  - If you can reach the relic within the fuel limit, do that.
  - Find the closest unexplored wormhole:
    - You can do that using **Dijkstra**, or **Floyd-Warshall**.
  - Can we reach it while still having enough fuel to go back to home base?
    - If yes: Go to that wormhole and update the distances between the points.
    - If no: Go back to home base and refuel.
  - Repeat.

# G: Galactic Expedition

Problem author: Veselin Mitev

- **Problem:** Navigate between wormholes to find the ancient relic, without running out of fuel.
- **Observation:** You can refuel more than enough times to simply explore all wormholes, until you find a way to reach the relic.
- **Solution:** Perform a “live” search – explore the wormholes while always keeping enough fuel ($\frac{d}{2}$) to go back to home base:
  - If you can reach the relic within the fuel limit, do that.
  - Find the closest unexplored wormhole:
    - You can do that using **Dijkstra**, or **Floyd-Warshall**.
  - Can we reach it while still having enough fuel to go back to home base?
    - If yes: Go to that wormhole and update the distances between the points.
    - If no: Go back to home base and refuel.
  - Repeat.
- Worst case: We can explore all wormholes in $\frac{n}{2}$ runs.

# G: Galactic Expedition

Problem author: Veselin Mitev

- **Problem:** Navigate between wormholes to find the ancient relic, without running out of fuel.
- **Observation:** You can refuel more than enough times to simply explore all wormholes, until you find a way to reach the relic.
- **Solution:** Perform a "live" search – explore the wormholes while always keeping enough fuel ($\frac{d}{2}$) to go back to home base:
  - If you can reach the relic within the fuel limit, do that.
  - Find the closest unexplored wormhole:
    - You can do that using **Dijkstra**, or **Floyd-Warshall**.
  - Can we reach it while still having enough fuel to go back to home base?
    - If yes: Go to that wormhole and update the distances between the points.
    - If no: Go back to home base and refuel.
  - Repeat.
- Worst case: We can explore all wormholes in $\frac{n}{2}$ runs.
- If we know all wormholes, it is guaranteed that we can reach the relic, if we follow an optimal path.

# G: Galactic Expedition

Problem author: Veselin Mitev

- **Problem:** Navigate between wormholes to find the ancient relic, without running out of fuel.
- **Observation:** You can refuel more than enough times to simply explore all wormholes, until you find a way to reach the relic.
- **Solution:** Perform a "live" search – explore the wormholes while always keeping enough fuel ($\frac{d}{2}$) to go back to home base:
  - If you can reach the relic within the fuel limit, do that.
  - Find the closest unexplored wormhole:
    - You can do that using **Dijkstra**, or **Floyd-Warshall**.
  - Can we reach it while still having enough fuel to go back to home base?
    - If yes: Go to that wormhole and update the distances between the points.
    - If no: Go back to home base and refuel.
  - Repeat.
- Worst case: We can explore all wormholes in $\frac{n}{2}$ runs.
- If we know all wormholes, it is guaranteed that we can reach the relic, if we follow an optimal path.
- **Time Complexity:** $\mathcal{O}(n^3)$ or $\mathcal{O}(n^3 \log n)$. Or if you're clever about how you cache the results from the Dijkstra search algorithm you can do it in $\mathcal{O}(n^2)$ or $\mathcal{O}(n^2 \log n)$.

# G: Galactic Expedition

Problem author: Veselin Mitev

- **Problem:** Navigate between wormholes to find the ancient relic, without running out of fuel.
- **Observation:** You can refuel more than enough times to simply explore all wormholes, until you find a way to reach the relic.
- **Solution:** Perform a "live" search – explore the wormholes while always keeping enough fuel ($\frac{d}{2}$) to go back to home base:
  - If you can reach the relic within the fuel limit, do that.
  - Find the closest unexplored wormhole:
    - You can do that using **Dijkstra**, or **Floyd-Warshall**.
  - Can we reach it while still having enough fuel to go back to home base?
    - If yes: Go to that wormhole and update the distances between the points.
    - If no: Go back to home base and refuel.
  - Repeat.
- Worst case: We can explore all wormholes in $\frac{n}{2}$ runs.
- If we know all wormholes, it is guaranteed that we can reach the relic, if we follow an optimal path.
- **Time Complexity:** $\mathcal{O}(n^3)$ or $\mathcal{O}(n^3 \log n)$. Or if you're clever about how you cache the results from the Dijkstra search algorithm you can do it in $\mathcal{O}(n^2)$ or $\mathcal{O}(n^2 \log n)$.

Statistics: 3 submissions, 0 accepted, 3 unknown

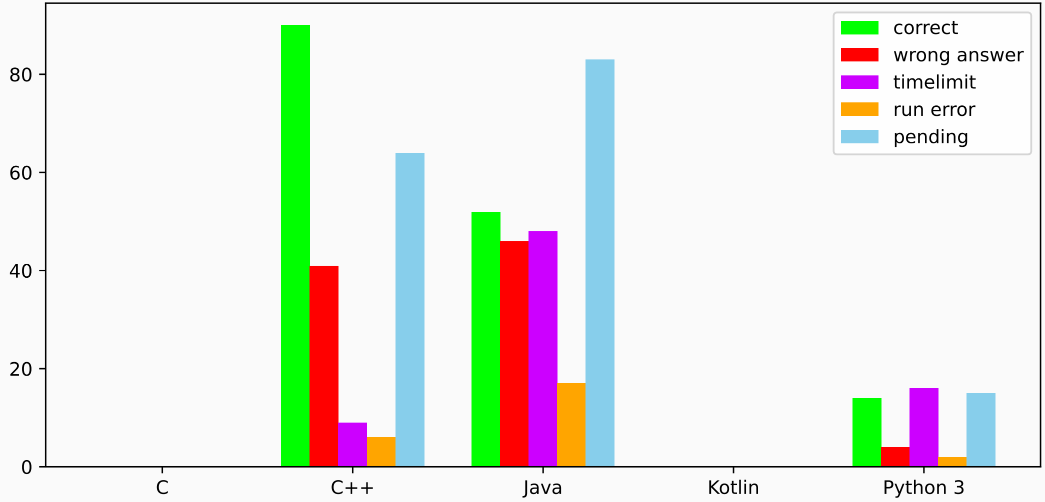
# Open online contest

Want to solve the problems you could not finish?
Or have friends that like to solve algorithmic problems?

`https://fpcs2024.bapc.eu/`

Friday 10 May 2024 13:00–17:00

## Language stats



# Random facts

## Jury work

- 448 commits (last year: 361)

[1] *After* codegolfing

# Random facts

## Jury work

- 448 commits (last year: 361)
- 357 secret test cases (last year: 339)

[1] *After* codegolfing

# Random facts

## Jury work

- 448 commits (last year: 361)
- 357 secret test cases (last year: 339)
- 120 accepted jury/proofreader solutions (last year: 96)

[1] *After* codegolfing

# Random facts

## Jury work

- 448 commits (last year: 361)
- 357 secret test cases (last year: 339)
- 120 accepted jury/proofreader solutions (last year: 96)
- The minimum[1] number of lines the jury needed to solve all problems is

$$2+1+11+1+5+1+22+5+3+11+4=66$$

On average 6.0 lines per problem, down from 6.4 last year

[1] *After* codegolfing

## Thanks to the proofreaders:

- Arnoud van der Leer (TU Delft)
- Daniel Cortild (RU Groningen)
- Davina van Meer (Delft)
- Henk van der Laan (TU Eindhoven)
- Matei Tinca (VU Amsterdam, 🎈)
- Michael Vasseur
  (VU Amsterdam / DOMjudge)
- Mylène Martodihardjo (VU Amsterdam)
- Nicky Gerritsen
  (TU Eindhoven / DOMjudge)
- Pavel Kunyavskiy
  (JetBrains Amsterdam, Kotlin Hero 🎈)
- Ragnar Groot Koerkamp
  (ETH Zürich / NWERC jury)
- Rick Wouters (TU Eindhoven)
- Siëna van Schaick (Radboud Nijmegen)
- Thomas Verwoerd
  (TU Delft, Kotlin Hero 🎈)
- Yoshi van den Akker (TU Delft)

# Thanks to the Jury for the Freshmen Programming Contests:

- Angel Karchev (TU Delft)
- Ivan Bliznets (RU Groningen)
- Jeroen Op de Beek (TU Delft)
- Leon van der Waal (TU Delft)
- Maarten Sijm (TU Delft)
- Maciek Sidor (VU Amsterdam)
- Makar Kuleshov (TU Delft)
- Mansur Nurmukhambetov (RU Groningen)
- Tymon Cichocki (TU Delft)
- Veselin Mitev (TU Delft)
- Vitor Greati (RU Groningen)
- Wietze Koops (Radboud Nijmegen / RU Groningen)
- Wiktor Cupiał (TU Delft)

## Want to help out with future contests?

https://wisv.ch/chipcie-interest

https://wisv.ch/runner