NWERC 2023

Solutions presentation

The NWERC 2023 jury

November 26, 2023

The NWERC 2023 Jury

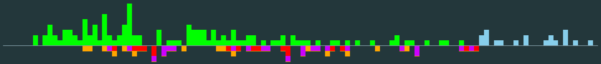
- **Doan-Dai Nguyen**
  École normale supérieure -
  Université Paris Sciences & Lettres
- **Jeroen Bransen**
  Chordify
- **Maarten Sijm**
  CHipCie (Delft University of Technology)
- **Michael Zündorf**
  Karlsruhe Institute of Technology
- **Nils Gustafsson**
  KTH Royal Institute of Technology
- **Paul Wild**
  FAU Erlangen-Nürnberg
- **Ragnar Groot Koerkamp**
  ETH Zurich
- **Reinier Schmiermann**
  Utrecht University
- **Wendy Yi**
  Karlsruhe Institute of Technology

Big thanks to our proofreaders and test solvers

- **Dany Sluijk**
  Delft University of Technology
- **Mees de Vries**
  BAPC Jury
- **Oleksandr Kulkov**
  ETH Zurich
- **Pavel Kunyavskiy**
  JetBrains, Amsterdam
- **Robin Lee**
  Google
- **Vitaly Aksenov**
  City, University of London

K: Klompendans

Problem Author: Maarten Sijm

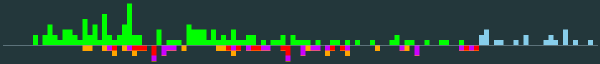


Problem

Find all reachable squares on an $n \times n$ grid that can be reached starting from the corner while alternating between knight moves of type (a, b) and (c, d).

K: Klompendans

Problem Author: Maarten Sijm



Problem

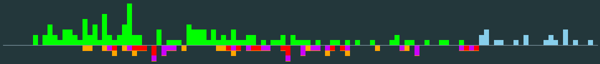
Find all reachable squares on an $n \times n$ grid that can be reached starting from the corner while alternating between knight moves of type (a, b) and (c, d).

Solution

- Create two copies of the grid, one for “the last move was of type (a, b)” and one for “the last move was of type (c, d).
- Starting from the two top left corners, run BFS or DFS to find the reachable states. After each move, transfer over to the other grid.
- Count all cells that are reachable in at least one of the grids.
- Total time: $\mathcal{O}(n^2)$.

K: Klompendans

Problem Author: Maarten Sijm



Problem

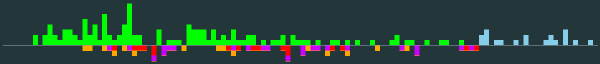
Find all reachable squares on an $n \times n$ grid that can be reached starting from the corner while alternating between knight moves of type (a, b) and (c, d).

Solution

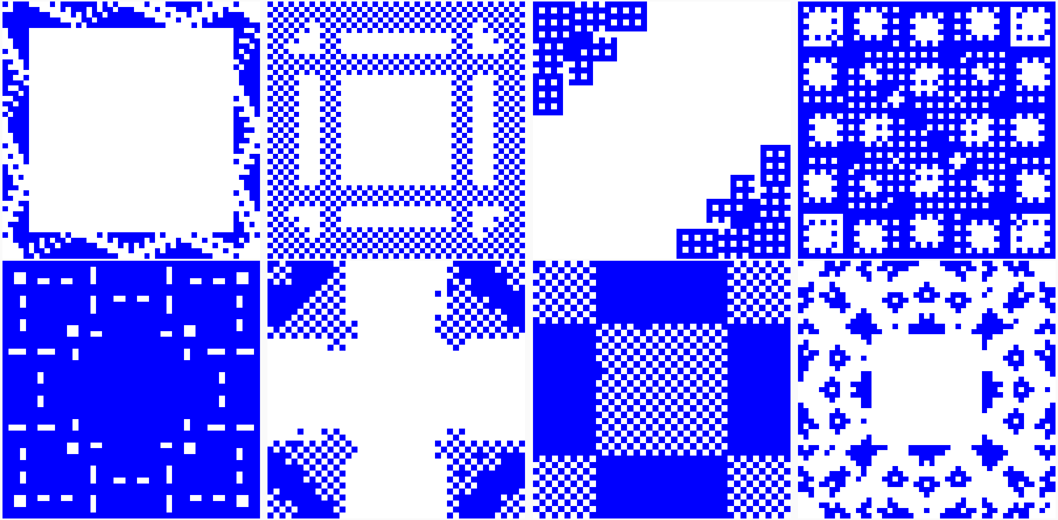
- Create two copies of the grid, one for "the last move was of type (a, b)" and one for "the last move was of type (c, d).
- Starting from the two top left corners, run BFS or DFS to find the reachable states. After each move, transfer over to the other grid.
- Count all cells that are reachable in at least one of the grids.
- Total time: $\mathcal{O}(n^2)$.

Statistics: 195 submissions, 120 accepted, 19 unknown

K: Klompendans

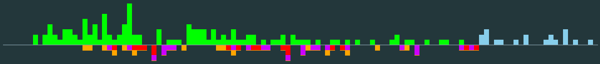


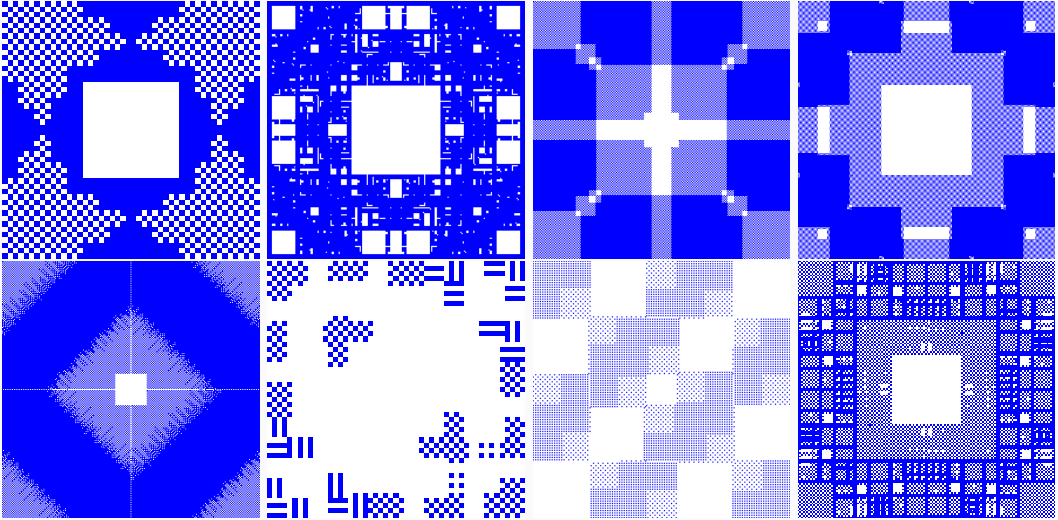
Problem Author: Maarten Sijm



K: Klompendans

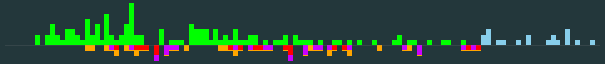
Problem Author: Maarten Sijm

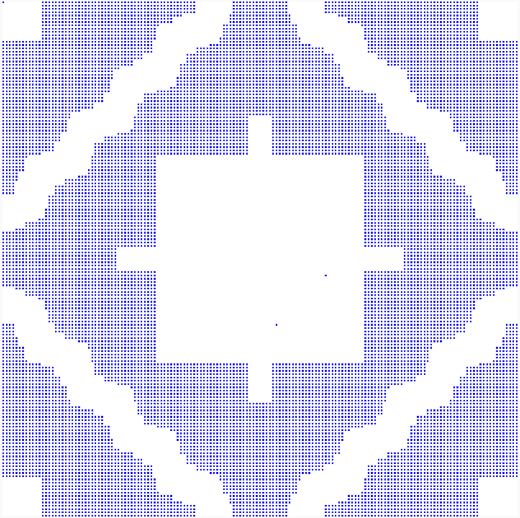


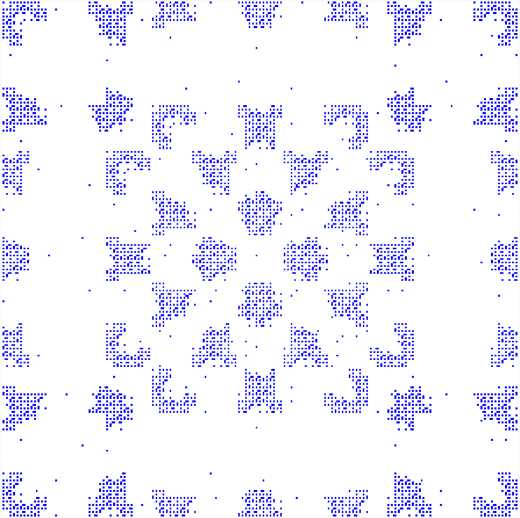


K: Klompendans

Problem Author: Maarten Sijm

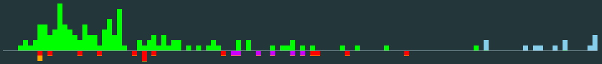






D: Date Picker

Problem Author: Jeroen Bransen



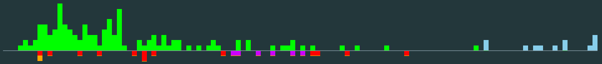
Problem

Given your availability for every hour in a week, pick at least $1 \leq d \leq 7$ days in the first poll and at least $1 \leq h \leq 24$ hours in the second poll to get the highest probability that you will be available.

Fun fact: based on a true story, while the jury was planning their first meeting!

D: Date Picker

Problem Author: Jeroen Bransen



Problem

Given your availability for every hour in a week, pick at least $1 \leq d \leq 7$ days in the first poll and at least $1 \leq h \leq 24$ hours in the second poll to get the highest probability that you will be available.

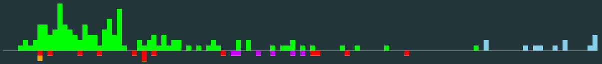
Fun fact: based on a true story, while the jury was planning their first meeting!

Observation

Selecting more than d days/h hours is never more efficient than selecting exactly d days/h hours.

D: Date Picker

Problem Author: Jeroen Bransen



Problem

Given your availability for every hour in a week, pick at least $1 \leq d \leq 7$ days in the first poll and at least $1 \leq h \leq 24$ hours in the second poll to get the highest probability that you will be available.

Fun fact: based on a true story, while the jury was planning their first meeting!

Observation

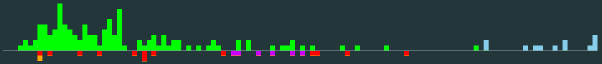
Selecting more than d days/h hours is never more efficient than selecting exactly d days/h hours.

Brute-force solution

For every combination of (a subset of d days) and (a subset of h hours), calculate the number of free timeslots, take the maximum, and divide by $d \cdot h$.

D: Date Picker

Problem Author: Jeroen Bransen



Problem

Given your availability for every hour in a week, pick at least $1 \leq d \leq 7$ days in the first poll and at least $1 \leq h \leq 24$ hours in the second poll to get the highest probability that you will be available.

Fun fact: based on a true story, while the jury was planning their first meeting!

Observation

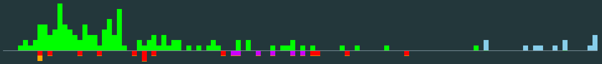
Selecting more than d days/h hours is never more efficient than selecting exactly d days/h hours.

Brute-force solution

For every combination of (a subset of d days) and (a subset of h hours), calculate the number of free timeslots, take the maximum, and divide by $d \cdot h$. **Too slow:** in the worst case where $d = 3$ and $h = 12$, this requires checking $\binom{7}{3} \cdot \binom{24}{12} \cdot 3 \cdot 12 \approx 3 \cdot 10^9$ timeslots.

D: Date Picker

Problem Author: Jeroen Bransen



Problem

Given your availability for every hour in a week, pick at least $1 \le d \le 7$ days in the first poll and at least $1 \le h \le 24$ hours in the second poll to get the highest probability that you will be available.

Fun fact: based on a true story, while the jury was planning their first meeting!

Observation

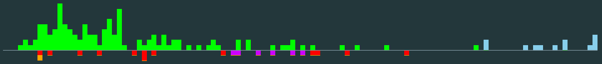
Selecting more than d days/h hours is never more efficient than selecting exactly d days/h hours.

Brute-force solution

For every combination of (a subset of d days) and (a subset of h hours), calculate the number of free timeslots, take the maximum, and divide by $d \cdot h$. **Too slow:** in the worst case where $d = 3$ and $h = 12$, this requires checking $\binom{7}{3} \cdot \binom{24}{12} \cdot 3 \cdot 12 \approx 3 \cdot 10^9$ timeslots. (Unless you write *very* efficient C++)

D: Date Picker

Problem Author: Jeroen Bransen



Problem

Given your availability for every hour in a week, pick at least $1 \leq d \leq 7$ days in the first poll and at least $1 \leq h \leq 24$ hours in the second poll to get the highest probability that you will be available.

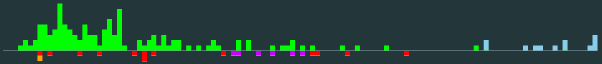
Fun fact: based on a true story, while the jury was planning their first meeting!

Greedy Solution

To avoid having to check all combinations, only check all combinations of d days.

D: Date Picker

Problem Author: Jeroen Bransen



Problem

Given your availability for every hour in a week, pick at least $1 \leq d \leq 7$ days in the first poll and at least $1 \leq h \leq 24$ hours in the second poll to get the highest probability that you will be available.

Fun fact: based on a true story, while the jury was planning their first meeting!

Greedy Solution

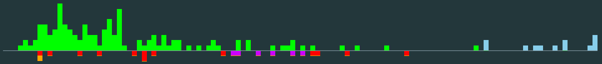
To avoid having to check all combinations, only check all combinations of d days.

For every combination of d days:

- For every hour, count the number of cells with '.'.
- Sort this list and select the h hours with the most open timeslots.
- Calculate the number of free timeslots, take the maximum, and divide by $d \cdot h$.

D: Date Picker

Problem Author: Jeroen Bransen



Problem

Given your availability for every hour in a week, pick at least $1 \leq d \leq 7$ days in the first poll and at least $1 \leq h \leq 24$ hours in the second poll to get the highest probability that you will be available.

Fun fact: based on a true story, while the jury was planning their first meeting!

Greedy Solution

To avoid having to check all combinations, only check all combinations of d days.

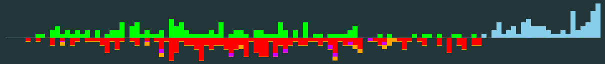
For every combination of d days:

- For every hour, count the number of cells with '.'.
- Sort this list and select the h hours with the most open timeslots.
- Calculate the number of free timeslots, take the maximum, and divide by $d \cdot h$.

Statistics: 150 submissions, 118 accepted, 12 unknown

L: Lateral Damage

Problem Author: Paul Wild

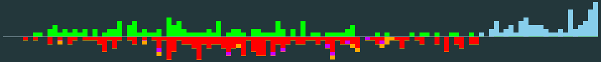


Problem

Play *Battleships* with a 100×100 grid where you need to sink up to 10 aircraft carriers in at most 2500 shots, and your opponent is potentially cheating (adaptive).

L: Lateral Damage

Problem Author: Paul Wild



Problem

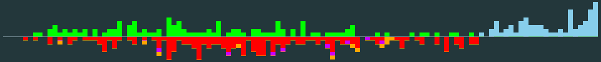
Play *Battleships* with a 100×100 grid where you need to sink up to 10 aircraft carriers in at most 2500 shots, and your opponent is potentially cheating (adaptive).

Observation

Shooting every fifth position in a straight line prevents your opponent from placing ships in between them.

L: Lateral Damage

Problem Author: Paul Wild



Problem

Play *Battleships* with a 100×100 grid where you need to sink up to 10 aircraft carriers in at most 2500 shots, and your opponent is potentially cheating (adaptive).

Observation

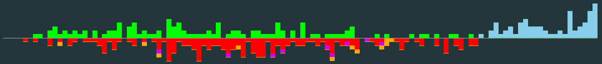
Shooting every fifth position in a straight line prevents your opponent from placing ships in between them.

Solution

- Generalizing this observation over two dimensions:
  shoot every position on every fifth diagonal line.
- For every hit, shoot the four positions left, right, above, and below to sink the full ship.

L: Lateral Damage

Problem Author: Paul Wild



Problem

Play *Battleships* with a 100×100 grid where you need to sink up to 10 aircraft carriers in at most 2500 shots, and your opponent is potentially cheating (adaptive).

Observation

Shooting every fifth position in a straight line prevents your opponent from placing ships in between them.

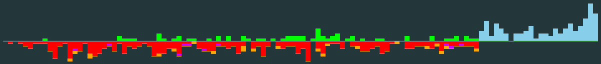
Solution

- Generalizing this observation over two dimensions:
  shoot every position on every fifth diagonal line.
- For every hit, shoot the four positions left, right, above, and below to sink the full ship.

Statistics: 363 submissions, 111 accepted, 71 unknown

H: Higher Arithmetic

Problem Author: Paul Wild

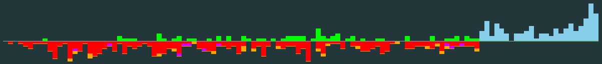


Problem

Print a valid arithmetic expression using +, *, (, and) and all given numbers exactly once such that the value is maximal.

H: Higher Arithmetic

Problem Author: Paul Wild



Problem

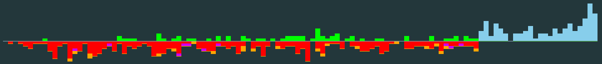
Print a valid arithmetic expression using +, *, (, and) and all given numbers exactly once such that the value is maximal.

Solution

- Idea: A maximal expression always is the product of sums.

H: Higher Arithmetic

Problem Author: Paul Wild



Problem

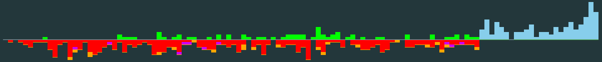
Print a valid arithmetic expression using +, *, (, and) and all given numbers exactly once such that the value is maximal.

Solution

- Idea: A maximal expression always is the product of sums.
- All numbers are > 1: Multiply all numbers.

H: Higher Arithmetic

Problem Author: Paul Wild



Problem

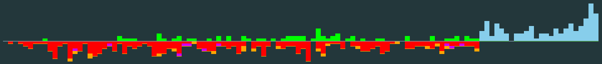
Print a valid arithmetic expression using +, *, (, and) and all given numbers exactly once such that the value is maximal.

Solution

- Idea: A maximal expression always is the product of sums.
- All numbers are > 1: Multiply all numbers.
- With 1s and 2s, some numbers need to be combined into sums.

H: Higher Arithmetic

Problem Author: Paul Wild



Problem

Print a valid arithmetic expression using +, *, (, and) and all given numbers exactly once such that the value is maximal.

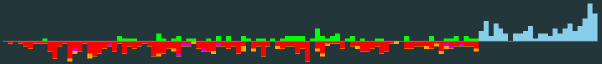
Solution

- Idea: A maximal expression always is the product of sums.
- All numbers are > 1: Multiply all numbers.
- With 1s and 2s, some numbers need to be combined into sums.

Cases:

H: Higher Arithmetic

Problem Author: Paul Wild



Problem

Print a valid arithmetic expression using +, *, (, and) and all given numbers exactly once such that the value is maximal.

Solution

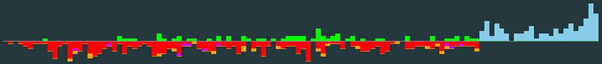
- Idea: A maximal expression always is the product of sums.
- All numbers are > 1: Multiply all numbers.
- With 1s and 2s, some numbers need to be combined into sums.

Cases:

- Only one 1: Add to second smallest number.

H: Higher Arithmetic

Problem Author: Paul Wild



Problem

Print a valid arithmetic expression using +, *, (, and) and all given numbers exactly once such that the value is maximal.

Solution

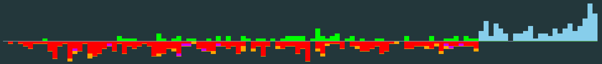
- Idea: A maximal expression always is the product of sums.
- All numbers are > 1: Multiply all numbers.
- With 1s and 2s, some numbers need to be combined into sums.

Cases:

- Only one 1: Add to second smallest number.
- No 2s: Repeatedly combine three 1s.

H: Higher Arithmetic

Problem Author: Paul Wild



Problem

Print a valid arithmetic expression using +, ∗, (, and) and all given numbers exactly once such that the value is maximal.

Solution

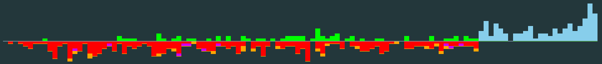
- Idea: A maximal expression always is the product of sums.
- All numbers are > 1: Multiply all numbers.
- With 1s and 2s, some numbers need to be combined into sums.

Cases:

- Only one 1: Add to second smallest number.
- No 2s: Repeatedly combine three 1s.
  - Special case: If two 1s or four 1s, combine two 1s.

H: Higher Arithmetic

Problem Author: Paul Wild



Problem

Print a valid arithmetic expression using +, *, (, and) and all given numbers exactly once such that the value is maximal.

Solution

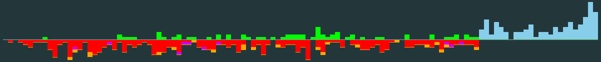
- Idea: A maximal expression always is the product of sums.
- All numbers are > 1: Multiply all numbers.
- With 1s and 2s, some numbers need to be combined into sums.

Cases:

- Only one 1: Add to second smallest number.
- No 2s: Repeatedly combine three 1s.
  - Special case: If two 1s or four 1s, combine two 1s.
- At least one 1 and one 2: Repeatedly combine one 1 and one 2.

H: Higher Arithmetic

Problem Author: Paul Wild



Problem

Print a valid arithmetic expression using +, *, (, and) and all given numbers exactly once such that the value is maximal.

Solution

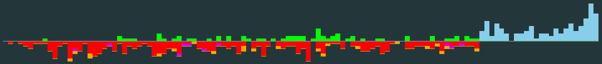
- Idea: A maximal expression always is the product of sums.
- All numbers are > 1: Multiply all numbers.
- With 1s and 2s, some numbers need to be combined into sums.

Cases:

- Only one 1: Add to second smallest number.
- No 2s: Repeatedly combine three 1s.
  - Special case: If two 1s or four 1s, combine two 1s.
- At least one 1 and one 2: Repeatedly combine one 1 and one 2.
  - Special case: If two 1s and one 2, combine those.

H: Higher Arithmetic

Problem Author: Paul Wild



Problem

Print a valid arithmetic expression using +, $*$, (, and) and all given numbers exactly once such that the value is maximal.

Solution

- Idea: A maximal expression always is the product of sums.
- All numbers are > 1: Multiply all numbers.
- With 1s and 2s, some numbers need to be combined into sums.

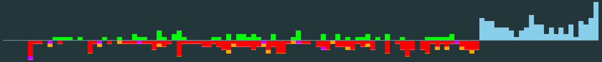
Cases:

- Only one 1: Add to second smallest number.
- No 2s: Repeatedly combine three 1s.
  - Special case: If two 1s or four 1s, combine two 1s.
- At least one 1 and one 2: Repeatedly combine one 1 and one 2.
  - Special case: If two 1s and one 2, combine those.

Statistics: 467 submissions, 67 accepted, 119 unknown

A: Arranging Adapters

Problem Author: Michael Zündorf

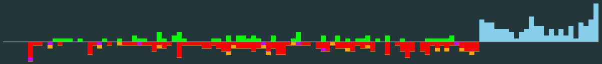


Problem

Given $1 \leq n \leq 2 \cdot 10^5$ chargers, each $3 \leq w \leq 10^9\,\mathrm{cm}$ wide, how many fit into a powerstrip comprising a row of $1 \leq s \leq 10^5$ sockets, each of width $3\,\mathrm{cm}$?

A: Arranging Adapters

Problem Author: Michael Zündorf



Problem

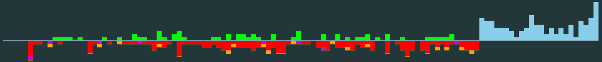
Given $1 \le n \le 2 \cdot 10^5$ chargers, each $3 \le w \le 10^9\,\mathrm{cm}$ wide, how many fit into a powerstrip comprising a row of $1 \le s \le 10^5$ sockets, each of width $3\,\mathrm{cm}$?

Solution

- First, greedily put the two largest chargers on the outside.

A: Arranging Adapters

Problem Author: Michael Zündorf



Problem

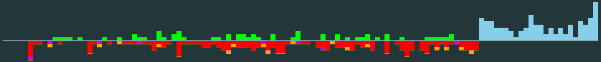
Given $1 \leq n \leq 2 \cdot 10^5$ chargers, each $3 \leq w \leq 10^9\,\mathrm{cm}$ wide, how many fit into a powerstrip comprising a row of $1 \leq s \leq 10^5$ sockets, each of width $3\,\mathrm{cm}$?

Solution

- First, greedily put the two largest chargers on the outside.
- If the answer is k, we can use the k smallest chargers.

A: Arranging Adapters

Problem Author: Michael Zündorf



Problem

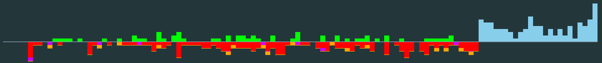
Given $1 \leq n \leq 2 \cdot 10^5$ chargers, each $3 \leq w \leq 10^9\,\mathrm{cm}$ wide, how many fit into a powerstrip comprising a row of $1 \leq s \leq 10^5$ sockets, each of width $3\,\mathrm{cm}$?

Solution

- First, greedily put the two largest chargers on the outside.
- If the answer is k, we can use the k smallest chargers.
- To test if the smallest k chargers fit:
  - Start with those of length $0 \mod 3$.
  - Then pair up $1 \mod 3$ and $2 \mod 3$ chargers, filling the gaps.
  - Then pair up remaining $1 \mod 3$, leaving a gap of 1 in between.
  - Lastly put the remaining chargers, and check the total length used.

A: Arranging Adapters

Problem Author: Michael Zündorf



Problem

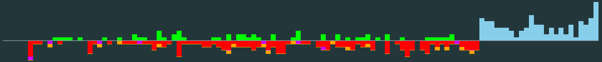
Given $1 \leq n \leq 2 \cdot 10^5$ chargers, each $3 \leq w \leq 10^9\,\mathrm{cm}$ wide, how many fit into a powerstrip comprising a row of $1 \leq s \leq 10^5$ sockets, each of width $3\,\mathrm{cm}$?

Solution

- First, greedily put the two largest chargers on the outside.
- If the answer is k, we can use the k smallest chargers.
- To test if the smallest k chargers fit:
  - Start with those of length $0 \mod 3$.
  - Then pair up $1 \mod 3$ and $2 \mod 3$ chargers, filling the gaps.
  - Then pair up remaining $1 \mod 3$, leaving a gap of 1 in between.
  - Lastly put the remaining chargers, and check the total length used.
- Binary search over k. Runtime $\mathcal{O}(n \log n)$.

A: Arranging Adapters

Problem Author: Michael Zündorf



Problem

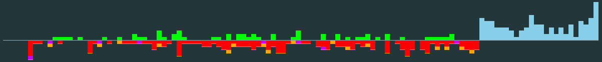
Given $1 \leq n \leq 2 \cdot 10^5$ chargers, each $3 \leq w \leq 10^9\,\mathrm{cm}$ wide, how many fit into a powerstrip comprising a row of $1 \leq s \leq 10^5$ sockets, each of width $3\,\mathrm{cm}$?

Solution

- First, greedily put the two largest chargers on the outside.
- If the answer is k, we can use the k smallest chargers.
- To test if the smallest k chargers fit:
  - Start with those of length $0 \mod 3$.
  - Then pair up $1 \mod 3$ and $2 \mod 3$ chargers, filling the gaps.
  - Then pair up remaining $1 \mod 3$, leaving a gap of 1 in between.
  - Lastly put the remaining chargers, and check the total length used.
- Binary search over k. Runtime $\mathcal{O}(n \log n)$.
- Edge case: when there is only a single socket.

A: Arranging Adapters

Problem Author: Michael Zündorf



Problem

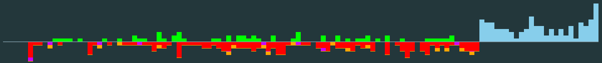
Given $1 \leq n \leq 2 \cdot 10^5$ chargers, each $3 \leq w \leq 10^9\,\mathrm{cm}$ wide, how many fit into a powerstrip comprising a row of $1 \leq s \leq 10^5$ sockets, each of width $3\,\mathrm{cm}$?

Solution

- First, greedily put the two largest chargers on the outside.
- If the answer is k, we can use the k smallest chargers.
- To test if the smallest k chargers fit:
  - Start with those of length $0 \mod 3$.
  - Then pair up $1 \mod 3$ and $2 \mod 3$ chargers, filling the gaps.
  - Then pair up remaining $1 \mod 3$, leaving a gap of 1 in between.
  - Lastly put the remaining chargers, and check the total length used.
- Binary search over k. Runtime $\mathcal{O}(n \log n)$.
- Edge case: when there is only a single socket.
- Linear time is also possible, trying to add one charger at a time.

A: Arranging Adapters

Problem Author: Michael Zündorf



Problem

Given $1 \leq n \leq 2 \cdot 10^5$ chargers, each $3 \leq w \leq 10^9\,\mathrm{cm}$ wide, how many fit into a powerstrip comprising a row of $1 \leq s \leq 10^5$ sockets, each of width $3\,\mathrm{cm}$?

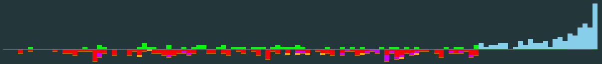
Solution

- First, greedily put the two largest chargers on the outside.
- If the answer is k, we can use the k smallest chargers.
- To test if the smallest k chargers fit:
  - Start with those of length $0 \mod 3$.
  - Then pair up $1 \mod 3$ and $2 \mod 3$ chargers, filling the gaps.
  - Then pair up remaining $1 \mod 3$, leaving a gap of 1 in between.
  - Lastly put the remaining chargers, and check the total length used.
- Binary search over k. Runtime $\mathcal{O}(n \log n)$.
- Edge case: when there is only a single socket.
- Linear time is also possible, trying to add one charger at a time.

Statistics: 333 submissions, 59 accepted, 110 unknown

F: Fixing Fractions

Problem Author: Michael Zündorf

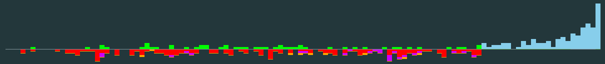


Problem

Given a fraction $\frac{a}{b}$, try to make it equal to $\frac{c}{d}$ by cancelling some digits in a and b

F: Fixing Fractions

Problem Author: Michael Zündorf



Problem

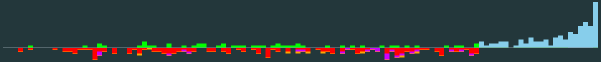
Given a fraction $\frac{a}{b}$, try to make it equal to $\frac{c}{d}$ by cancelling some digits in a and b

Solution

- Try all possible $\mathcal{O}(2^{|a|})$ subsets of a
- Given a', c and d, we know $b' = \frac{a' \cdot d}{c}$ must hold
- Check if b can be made into b' by removing the same digits

F: Fixing Fractions

Problem Author: Michael Zündorf



Problem

Given a fraction $\frac{a}{b}$, try to make it equal to $\frac{c}{d}$ by cancelling some digits in a and b

Solution

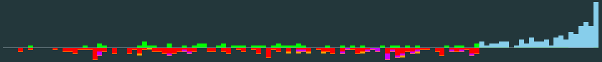
- Try all possible $\mathcal{O}(2^{|a|})$ subsets of a
- Given a', c and d, we know $b' = \frac{a' \cdot d}{c}$ must hold
- Check if b can be made into b' by removing the same digits

Pitfalls

- $a' \cdot d$ not divisible by c
- Leading zeroes
- 64-bit integer overflow: take GCD first, do operations modulo some prime, use bigger integers

F: Fixing Fractions

Problem Author: Michael Zündorf



Problem

Given a fraction $\frac{a}{b}$, try to make it equal to $\frac{c}{d}$ by cancelling some digits in a and b

Solution

- Try all possible $\mathcal{O}(2^{|a|})$ subsets of a
- Given a', c and d, we know $b' = \frac{a' \cdot d}{c}$ must hold
- Check if b can be made into b' by removing the same digits

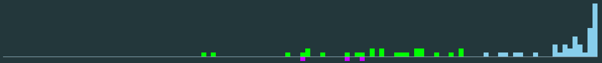
Pitfalls

- $a' \cdot d$ not divisible by c
- Leading zeroes
- 64-bit integer overflow: take GCD first, do operations modulo some prime, use bigger integers

Statistics: 347 submissions, 51 accepted, 125 unknown

J: Jogging Tour

Problem Author: Paul Wild

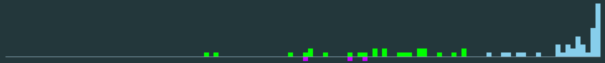


Problem

Find the optimal grid angle to make a tour through $n \leq 12$ points.

J: Jogging Tour

Problem Author: Paul Wild



Problem

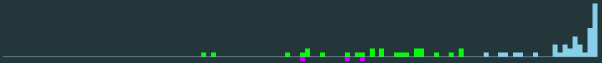
Find the optimal grid angle to make a tour through $n \leq 12$ points.

Subtask: assume we know the angle

- All possible $\mathcal{O}(n!)$ routes, too slow!
- DP with (current location, locations still todo)
- This runs in $\mathcal{O}(n^2 \cdot 2^n)$

J: Jogging Tour

Problem Author: Paul Wild



Problem

Find the optimal grid angle to make a tour through $n \leq 12$ points.

Subtask: assume we know the angle

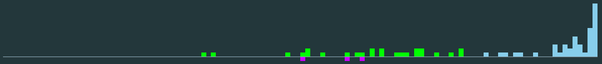
- All possible $\mathcal{O}(n!)$ routes, too slow!
- DP with (current location, locations still todo)
- This runs in $\mathcal{O}(n^2 \cdot 2^n)$

Complete solution

- Insight: in the optimal solution, there is a straight line between two consecutive locations
- Consider all n^2 angles between pairs of locations
- Total complexity $\mathcal{O}(n^4 \cdot 2^n)$

J: Jogging Tour

Problem Author: Paul Wild



Problem

Find the optimal grid angle to make a tour through $n \leq 12$ points.

Subtask: assume we know the angle

- All possible $\mathcal{O}(n!)$ routes, too slow!
- DP with (current location, locations still todo)
- This runs in $\mathcal{O}(n^2 \cdot 2^n)$

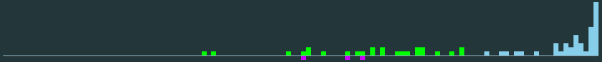
Complete solution

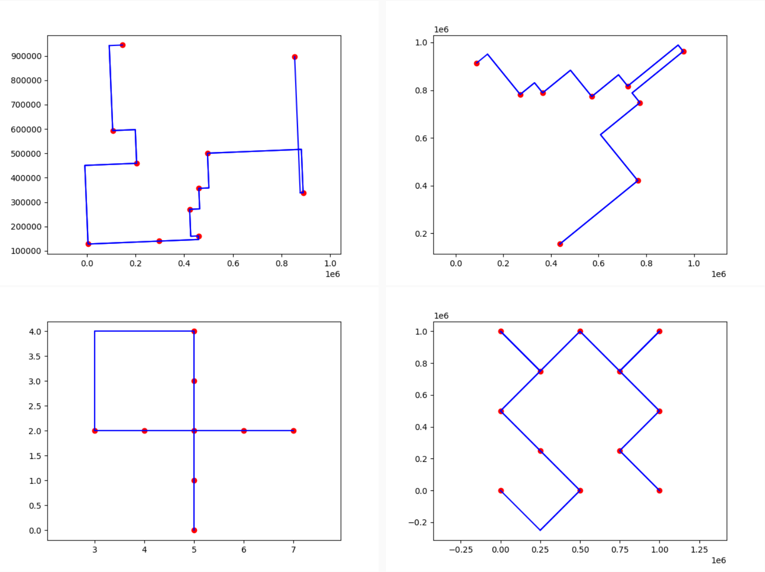
- Insight: in the optimal solution, there is a straight line between two consecutive locations
- Consider all n^2 angles between pairs of locations
- Total complexity $\mathcal{O}(n^4 \cdot 2^n)$

Statistics: 72 submissions, 25 accepted, 44 unknown

J: Jogging Tour

Problem Author: Paul Wild





C: Chair Dance

Problem Author: Michael Zündorf

Problem

Given are $n \leq 10^5$ players playing a deterministic version of *musical chairs*. Player i starts on chair i. Apply up to 10^5 commands:

- Rotate by $+r$: the person on chair i moves clockwise to chair $i + r$.
- Multiply by $*m$, the person on chair i moves to $i \cdot m$, where the person walking the least gets it.
- On $?q$, print who sits on chair q.

C: Chair Dance

Problem Author: Michael Zündorf

Problem

Given are $n \leq 10^5$ players playing a deterministic version of *musical chairs*. Player i starts on chair i. Apply up to 10^5 commands:

- Rotate by $+r$: the person on chair i moves clockwise to chair $i + r$.
- Multiply by $*m$, the person on chair i moves to $i \cdot m$, where the person walking the least gets it.
- On $?q$, print who sits on chair q.

Naive solution

Store who sits on each chair, and apply each command. $\mathcal{O}(n^2)$

C: Chair Dance

Problem Author: Michael Zündorf

Solution

Be *lazy*! Initialize $p[i] = i$, the person on chair i.

- Instead of rotating by $+r$, increment the *total rotation* R. $p[i]$ is now at $i + R$, so query $p[q - R]$.

C: Chair Dance

Problem Author: Michael Zündorf

Solution

Be *lazy*! Initialize $p[i] = i$, the person on chair i.

- Instead of rotating by $+r$, increment the *total rotation* R. $p[i]$ is now at $i + R$, so query $p[q - R]$.
- For *collision-free* multiplications: store total multiplication M, so $p[i]$ is now at $M \cdot i + R$. When multiplying by m, update $M \leftarrow m \cdot M$ and $R \leftarrow m \cdot R$. Query $p[(q - R) \cdot M^{-1}]$.

C: Chair Dance

Problem Author: Michael Zündorf

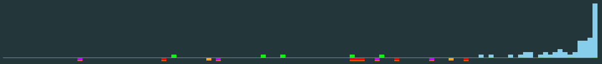
Solution

Be *lazy*! Initialize $p[i] = i$, the person on chair i.

- Instead of rotating by $+r$, increment the *total rotation* R. $p[i]$ is now at $i + R$, so query $p[q - R]$.
- For *collision-free* multiplications: store total multiplication M, so $p[i]$ is now at $M \cdot i + R$. When multiplying by m, update $M \leftarrow m \cdot M$ and $R \leftarrow m \cdot R$. Query $p[(q - R) \cdot M^{-1}]$.
- Collisions occur when $\gcd(m, k) > 1$ ($k = \#$leftover people).
  Simulate these fully, set $k \leftarrow k / \gcd(m, k)$, and reset R and M.

C: Chair Dance

Problem Author: Michael Zündorf



Solution

Be *lazy*! Initialize $p[i] = i$, the person on chair i.

- Instead of rotating by $+r$, increment the *total rotation* R. $p[i]$ is now at $i + R$, so query $p[q - R]$.
- For *collision-free* multiplications: store total multiplication M, so $p[i]$ is now at $M \cdot i + R$. When multiplying by m, update $M \leftarrow m \cdot M$ and $R \leftarrow m \cdot R$. Query $p[(q - R) \cdot M^{-1}]$.
- Collisions occur when $\gcd(m, k) > 1$ ($k = \#$leftover people).
  Simulate these fully, set $k \leftarrow k / \gcd(m, k)$, and reset R and M.
- Be careful about queries to empty chairs.

C: Chair Dance

Problem Author: Michael Zündorf

Solution

Be *lazy*! Initialize $p[i] = i$, the person on chair i.

- Instead of rotating by $+r$, increment the *total rotation* R. $p[i]$ is now at $i + R$, so query $p[q - R]$.
- For *collision-free* multiplications: store total multiplication M, so $p[i]$ is now at $M \cdot i + R$. When multiplying by m, update $M \leftarrow m \cdot M$ and $R \leftarrow m \cdot R$. Query $p[(q - R) \cdot M^{-1}]$.
- Collisions occur when $\gcd(m, k) > 1$ ($k = \#\text{leftover people}$). Simulate these fully, set $k \leftarrow k / \gcd(m, k)$, and reset R and M.
- Be careful about queries to empty chairs.
- Each collision at least halves k, so at most $\lg n$ collisions.

C: Chair Dance

Problem Author: Michael Zündorf

Solution

Be *lazy*! Initialize $p[i] = i$, the person on chair i.

- Instead of rotating by $+r$, increment the *total rotation* R. $p[i]$ is now at $i + R$, so query $p[q - R]$.
- For *collision-free* multiplications: store total multiplication M, so $p[i]$ is now at $M \cdot i + R$. When multiplying by m, update $M \leftarrow m \cdot M$ and $R \leftarrow m \cdot R$. Query $p[(q - R) \cdot M^{-1}]$.
- Collisions occur when $\gcd(m, k) > 1$ ($k = \#$leftover people).
  Simulate these fully, set $k \leftarrow k / \gcd(m, k)$, and reset R and M.
- Be careful about queries to empty chairs.
- Each collision at least halves k, so at most $\lg n$ collisions.
- Runtime: $\mathcal{O}(n \log n)$.

C: Chair Dance

Problem Author: Michael Zündorf

Solution

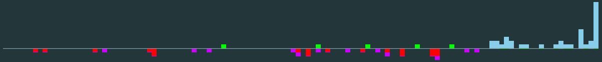
Be *lazy*! Initialize $p[i] = i$, the person on chair i.

- Instead of rotating by $+r$, increment the *total rotation* R. $p[i]$ is now at $i + R$, so query $p[q - R]$.
- For *collision-free* multiplications: store total multiplication M, so $p[i]$ is now at $M \cdot i + R$. When multiplying by m, update $M \leftarrow m \cdot M$ and $R \leftarrow m \cdot R$. Query $p[(q - R) \cdot M^{-1}]$.
- Collisions occur when $\gcd(m, k) > 1$ ($k = \#$leftover people).
  Simulate these fully, set $k \leftarrow k / \gcd(m, k)$, and reset R and M.
- Be careful about queries to empty chairs.
- Each collision at least halves k, so at most $\lg n$ collisions.
- Runtime: $\mathcal{O}(n \log n)$.

Statistics: 77 submissions, 5 accepted, 60 unknown

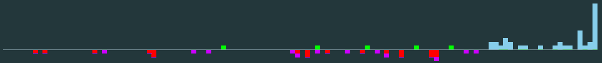
E: Exponentiation

Problem Author: Reinier Schmiermann



Problem

There are n variables $x_1, x_2, \dots, x_n$, initially set to 2023. You are given m queries that either assigns x_i to $x_i^{x_j}$, or asks you to compare x_i and x_j.



Problem

There are n variables $x_1, x_2, \ldots, x_n$, initially set to 2023. You are given m queries that either assigns x_i to $x_i^{x_j}$, or asks you to compare x_i and x_j.

Observation

- To make the numbers slightly less huge, take the logarithm twice. Let $y_i = \log\log(x_i)$.

Problem

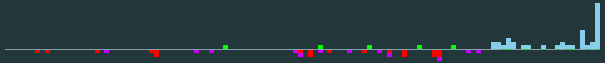
There are n variables $x_1, x_2, \ldots, x_n$, initially set to 2023. You are given m queries that either assigns x_i to $x_i^{x_j}$, or asks you to compare x_i and x_j.

Observation

- To make the numbers slightly less huge, take the logarithm twice. Let $y_i = \log\log(x_i)$.
- $x_i = x_i^{x_j} \iff y_i = y_i + 2023^{y_j}$.

E: Exponentiation

Problem Author: Reinier Schmiermann



Problem

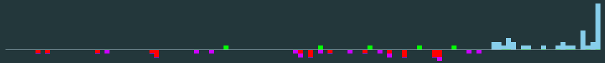
There are n variables $x_1, x_2, \ldots, x_n$, initially set to 2023. You are given m queries that either assigns x_i to $x_i^{x_j}$, or asks you to compare x_i and x_j.

Observation

- To make the numbers slightly less huge, take the logarithm twice. Let $y_i = \log\log(x_i)$.
- $x_i = x_i^{x_j} \iff y_i = y_i + 2023^{y_j}$.
- Consider these numbers in base 2023. Each operation, one of the digits will increase by one. But no carry will ever happen since there are fewer than 2023 operations.

E: Exponentiation

Problem Author: Reinier Schmiermann



Problem

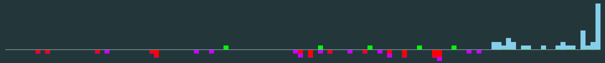
There are n variables $x_1, x_2, \ldots, x_n$, initially set to 2023. You are given m queries that either assigns x_i to $x_i^{x_j}$, or asks you to compare x_i and x_j.

Observation

- To make the numbers slightly less huge, take the logarithm twice. Let $y_i = \log\log(x_i)$.
- $x_i = x_i^{x_j} \iff y_i = y_i + 2023^{y_j}$.
- Consider these numbers in base 2023. Each operation, one of the digits will increase by one. But no carry will ever happen since there are fewer than 2023 operations.
- When a variable gets updated, it is much easier to create a new variable $y' = y_i + 2023^{y_j}$.

E: Exponentiation

Problem Author: Reinier Schmiermann

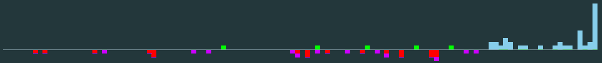


Solution

- Keep all variables ordered by size at all times. Answering queries becomes easy. But how to maintain the order?

E: Exponentiation

Problem Author: Reinier Schmiermann

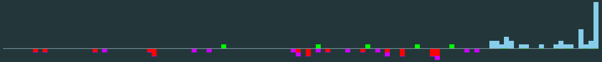


Solution

- Keep all variables ordered by size at all times. Answering queries becomes easy. But how to maintain the order?
- For every variable y, let $d(y)$ be a list containing the positions of its non-zero digits (in base 2023). These positions will be other variables, that we know the order of. Two variables can be compared by lexicographically comparing their lists.

E: Exponentiation

Problem Author: Reinier Schmiermann

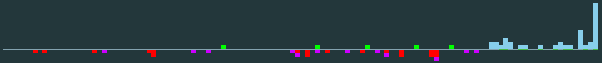


Solution

- Keep all variables ordered by size at all times. Answering queries becomes easy. But how to maintain the order?
- For every variable y, let $d(y)$ be a list containing the positions of its non-zero digits (in base 2023). These positions will be other variables, that we know the order of. Two variables can be compared by lexicographically comparing their lists.
- When a new variable $y' = y_i + 2023^{y_j}$ is created, let $d(y') = d(y_i) \cup \{y_j\}$. Insert this new variable y' into the ordering.

E: Exponentiation

Problem Author: Reinier Schmiermann

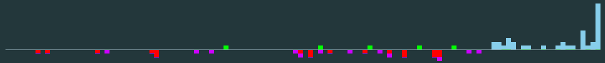


Solution

- Keep all variables ordered by size at all times. Answering queries becomes easy. But how to maintain the order?
- For every variable y, let $d(y)$ be a list containing the positions of its non-zero digits (in base 2023). These positions will be other variables, that we know the order of. Two variables can be compared by lexicographically comparing their lists.
- When a new variable $y' = y_i + 2023^{y_j}$ is created, let $d(y') = d(y_i) \cup \{y_j\}$. Insert this new variable y' into the ordering.
- To keep track of the order of variables, a trie or a sorted list can be used. This can be done in $\mathcal{O}(n^2)$ or $\mathcal{O}(n^2 \log(n))$.

E: Exponentiation

Problem Author: Reinier Schmiermann

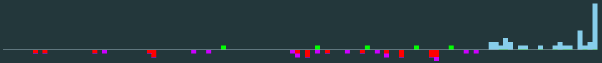


Solution

- Keep all variables ordered by size at all times. Answering queries becomes easy. But how to maintain the order?
- For every variable y, let $d(y)$ be a list containing the positions of its non-zero digits (in base 2023). These positions will be other variables, that we know the order of. Two variables can be compared by lexicographically comparing their lists.
- When a new variable $y' = y_i + 2023^{y_j}$ is created, let $d(y') = d(y_i) \cup \{y_j\}$. Insert this new variable y' into the ordering.
- To keep track of the order of variables, a trie or a sorted list can be used. This can be done in $\mathcal{O}(n^2)$ or $\mathcal{O}(n^2 \log(n))$.
- **Challenge**: Can you solve the problem faster than quadratic time?

E: Exponentiation

Problem Author: Reinier Schmiermann



Solution

- Keep all variables ordered by size at all times. Answering queries becomes easy. But how to maintain the order?
- For every variable y, let $d(y)$ be a list containing the positions of its non-zero digits (in base 2023). These positions will be other variables, that we know the order of. Two variables can be compared by lexicographically comparing their lists.
- When a new variable $y' = y_i + 2023^{y_j}$ is created, let $d(y') = d(y_i) \cup \{y_j\}$. Insert this new variable y' into the ordering.
- To keep track of the order of variables, a trie or a sorted list can be used. This can be done in $\mathcal{O}(n^2)$ or $\mathcal{O}(n^2 \log(n))$.
- **Challenge**: Can you solve the problem faster than quadratic time?

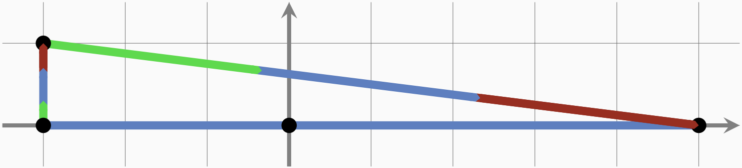
Statistics: 74 submissions, 5 accepted, 38 unknown

G: Galaxy Quest

Problem Author: Mike de Vries

Problem

You are given a graph consisting of line segments in 3D space. You travel on a ship with constant acceleration and constant fuel consumption for the time spent accelerating. You need to come to a standstill at each vertex. Given a target location and a time limit, find the minimum amount of fuel needed to get there. You need to answer multiple queries, all from the same starting location.



Solution for fixed path

- Consider a path consisting of multiple line segments.

Solution for fixed path

- Consider a path consisting of multiple line segments.
- Suppose the ith segment is d_i metres long and we accelerate/decelerate for x_i seconds along it.
- Then it takes $x_i + \frac{d_i}{x_i}$ seconds to traverse the ith segment.
- New problem: minimize $\sum 2x_i$ subject to $\sum x_i + \frac{d_i}{x_i} \leq t$.
- **Key insight**: optimum is reached when $x_i = c \cdot \sqrt{d_i}$ for some constant c.
- We can compute c by solving $c + \frac{1}{c} = t / \sum \sqrt{d_i}$. When the RHS is < 2, no solution exists.

Solution for fixed path

- Consider a path consisting of multiple line segments.
- Suppose the ith segment is d_i metres long and we accelerate/decelerate for x_i seconds along it.
- Then it takes $x_i + \frac{d_i}{x_i}$ seconds to traverse the ith segment.
- New problem: minimize $\sum 2x_i$ subject to $\sum x_i + \frac{d_i}{x_i} \leq t$.
- **Key insight**: optimum is reached when $x_i = c \cdot \sqrt{d_i}$ for some constant c.
- We can compute c by solving $c + \frac{1}{c} = t / \sum \sqrt{d_i}$. When the RHS is < 2, no solution exists.

Solution

- To keep the time limit and save fuel, find a path that minimizes $\sum \sqrt{d_i}$.

G: Galaxy Quest

Problem Author: Mike de Vries

Solution for fixed path

- Consider a path consisting of multiple line segments.
- Suppose the ith segment is d_i metres long and we accelerate/decelerate for x_i seconds along it.
- Then it takes $x_i + \frac{d_i}{x_i}$ seconds to traverse the ith segment.
- New problem: minimize $\sum 2x_i$ subject to $\sum x_i + \frac{d_i}{x_i} \leq t$.
- **Key insight**: optimum is reached when $x_i = c \cdot \sqrt{d_i}$ for some constant c.
- We can compute c by solving $c + \frac{1}{c} = t / \sum \sqrt{d_i}$. When the RHS is < 2, no solution exists.

Solution

- To keep the time limit and save fuel, find a path that minimizes $\sum \sqrt{d_i}$.
- Use Dijkstra's algorithm for this, where edges have length $\sqrt{d_i}$.

G: Galaxy Quest

Problem Author: Mike de Vries

Solution for fixed path

- Consider a path consisting of multiple line segments.
- Suppose the ith segment is d_i metres long and we accelerate/decelerate for x_i seconds along it.
- Then it takes $x_i + \frac{d_i}{x_i}$ seconds to traverse the ith segment.
- New problem: minimize $\sum 2x_i$ subject to $\sum x_i + \frac{d_i}{x_i} \leq t$.
- **Key insight**: optimum is reached when $x_i = c \cdot \sqrt{d_i}$ for some constant c.
- We can compute c by solving $c + \frac{1}{c} = t / \sum \sqrt{d_i}$. When the RHS is < 2, no solution exists.

Solution

- To keep the time limit and save fuel, find a path that minimizes $\sum \sqrt{d_i}$.
- Use Dijkstra's algorithm for this, where edges have length $\sqrt{d_i}$.
- The starting location is fixed, so queries can be answered in constant time.

G: Galaxy Quest

Problem Author: Mike de Vries

Solution for fixed path

- Consider a path consisting of multiple line segments.
- Suppose the ith segment is d_i metres long and we accelerate/decelerate for x_i seconds along it.
- Then it takes $x_i + \frac{d_i}{x_i}$ seconds to traverse the ith segment.
- New problem: minimize $\sum 2x_i$ subject to $\sum x_i + \frac{d_i}{x_i} \leq t$.
- **Key insight**: optimum is reached when $x_i = c \cdot \sqrt{d_i}$ for some constant c.
- We can compute c by solving $c + \frac{1}{c} = t / \sum \sqrt{d_i}$. When the RHS is < 2, no solution exists.

Solution

- To keep the time limit and save fuel, find a path that minimizes $\sum \sqrt{d_i}$.
- Use Dijkstra's algorithm for this, where edges have length $\sqrt{d_i}$.
- The starting location is fixed, so queries can be answered in constant time.

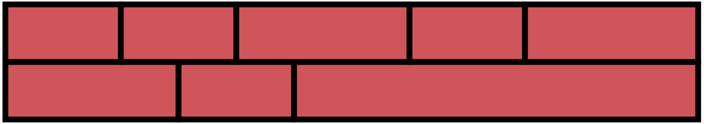
Statistics: 12 submissions, 1 accepted, 9 unknown

B: Brickwork

Problem Author: Michael Zündorf

Problem

Given n types of bricks $b_1, \ldots, b_n$, can you build a wall of width w where no two gaps appear above each other?



Subtask

Can at least one row be built?

Subtask

Can at least one row be built?

Solution

This is known as the *coin change problem* and can be solved like this:

- $\mathcal{O}(\frac{w^2}{64})$ with dp + bitsets
- $\mathcal{O}(w \log(w)^2)$ with fft (faster is possible)

B: Brickwork

Problem Author: Michael Zündorf

Subtask

Can at least one row be built?

Solution

This is known as the *coin change problem* and can be solved like this:

- $\mathcal{O}(\frac{w^2}{64})$ with dp + bitsets
- $\mathcal{O}(w\log(w)^2)$ with fft (faster is possible)
- Bitsets are much faster

B: Brickwork

Problem Author: Michael Zündorf

Case 1

- $w \in \{b_1, \ldots, b_n\}$

B: Brickwork

Problem Author: Michael Zündorf

Case 1

- $w \in \{b_1, \ldots, b_n\}$

Case 2

- There is a row that uses two bricks b_x, b_y

B: Brickwork

Problem Author: Michael Zündorf

Case 1

- $w \in \{b_1, \dots, b_n\}$

Case 2

- There is a row that uses two bricks b_x, b_y
- WLOG:
  - Let b_x be the shortest
  - Let b_y be the second shortest
  - there are as few b_x as possible
    (still at least one)

B: Brickwork

Problem Author: Michael Zündorf

Case 1

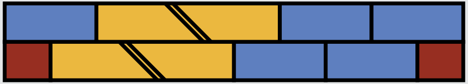
- $w \in \{b_1, \ldots, b_n\}$

Case 2

- There is a row that uses two bricks b_x,b_y
- WLOG:
  - Let b_x be the shortest
  - Let b_y be the second shortest
  - there are as few b_x as possible (still at least one)

Case 2.1

- Sum of b_x can be replace by some b_y



B: Brickwork

Problem Author: Michael Zündorf

Case 1

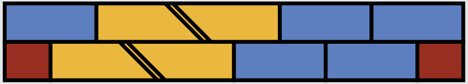
- $w \in \{b_1, \ldots, b_n\}$

Case 2

- There is a row that uses two bricks b_x, b_y
- WLOG:
  - Let b_x be the shortest
  - Let b_y be the second shortest
  - there are as few b_x as possible (still at least one)

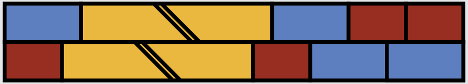
Case 2.1

- Sum of b_x can be replace by some b_y



Case 2.2

- Else



Case 3

- There are two bricks b_x, b_y that divide w

Case 3

- There are two bricks b_x,b_y that divide w
- Case 2 implies that $lcm(b_x, b_y) = w$

Case 3

- There are two bricks b_x,b_y that divide w
- Case 2 implies that $lcm(b_x, b_y) = w$

Case 4

- Impossible

Conclusion

The solution exists in two cases:

- Trivial: $w \in \{b_1, \ldots, b_n\}$
- There exist two bricks that both can be part of a solution

Case 3

- There are two bricks b_x, b_y that divide w
- Case 2 implies that $lcm(b_x, b_y) = w$

Case 4

- Impossible

Conclusion

The solution exists in two cases:

- Trivial: $w \in \{b_1, \ldots, b_n\}$
- There exist two bricks that both can be part of a solution

Statistics: 14 submissions, 0 accepted, 11 unknown

I: Isolated Island

Problem Author: Michael Zündorf

Problem

Given $2n$ points, is there a point that occurs an odd number of times?

I: Isolated Island

Problem Author: Michael Zündorf

Problem

Given $2n$ points, is there a point that occurs an odd number of times?

Solutions

- Sort the points, check whether point $2i-1$ equals point $2i$ in $\mathcal{O}(n \log n)$
- XOR hashes of all points in $\mathcal{O}(n)$

I: Isolated Island

Problem Author: Michael Zündorf

Problem

Given $n \leq 1000$ line segments that partition the plane in small regions. Are there two regions the same *distance* from the ocean?

I: Isolated Island

Problem Author: Michael Zündorf

Problem

Given $n \leq 1000$ line segments that partition the plane in small regions. Are there two regions the same *distance* from the ocean?

Geometry solution

Find all intersections and construct the dual graph on faces:
Costs $\mathcal{O}\left(n^2 \log n\right)$ and your sanity (256 lines of C++).

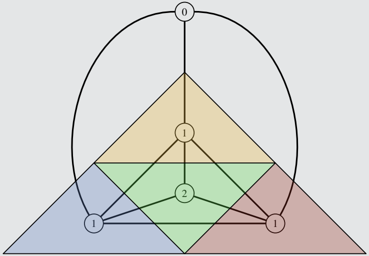
Problem Author: Michael Zündorf

Problem

Given $n \leq 1000$ line segments that partition the plane in small regions. Are there two regions the same *distance* from the ocean?

Intended solution

- Consider the dual graph, with one vertex per region.



I: Isolated Island

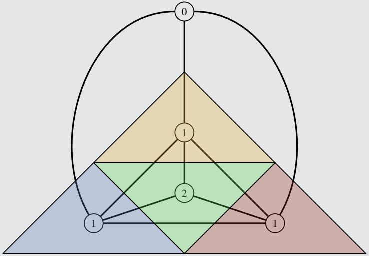
Problem Author: Michael Zündorf

Problem

Given $n \leq 1000$ line segments that partition the plane in small regions. Are there two regions the same *distance* from the ocean?

Intended solution

- Consider the dual graph, with one vertex per region.
- The answer is yes if there are adjacent regions with equal distance to the ocean.

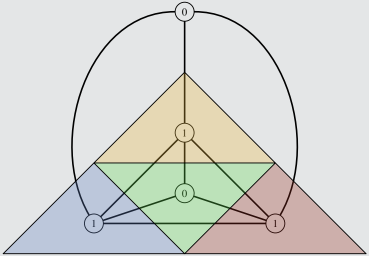


Problem

Given $n \leq 1000$ line segments that partition the plane in small regions. Are there two regions the same *distance* from the ocean?

Intended solution

- Consider the dual graph, with one vertex per region.
- The answer is yes if there are adjacent regions with equal distance to the ocean.
- The difference between adjacent distances is at most 1, so we can work modulo 2 instead.

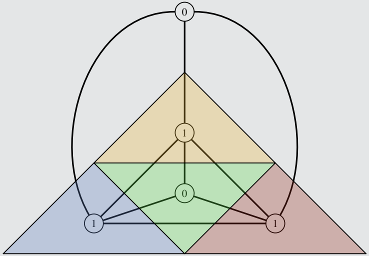


Problem

Given $n \leq 1000$ line segments that partition the plane in small regions. Are there two regions the same *distance* from the ocean?

Intended solution

- Consider the dual graph, with one vertex per region.
- The answer is yes if there are adjacent regions with equal distance to the ocean.
- The difference between adjacent distances is at most 1, so we can work modulo 2 instead.
- The answer is no iff all pairs of adjacent faces have opposite values.

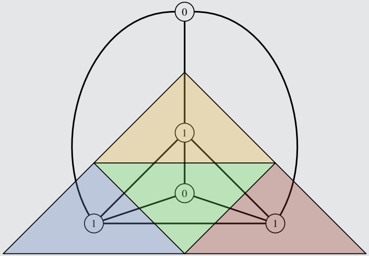


Problem

Given $n \leq 1000$ line segments that partition the plane in small regions. Are there two regions the same *distance* from the ocean?

Intended solution

- The answer is `no` iff all pairs of adjacent faces have opposite values.

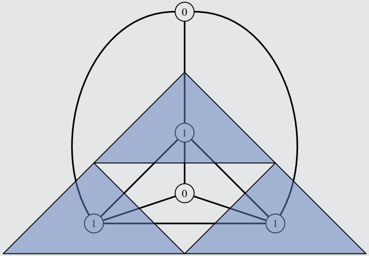


Problem

Given $n \leq 1000$ line segments that partition the plane in small regions. Are there two regions the same *distance* from the ocean?

Intended solution

- The answer is `no` iff all pairs of adjacent faces have opposite values.
- I.e.: the dual graph must be bipartite.



I: Isolated Island

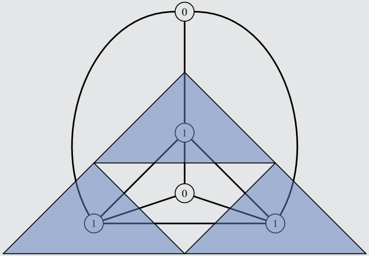
Problem Author: Michael Zündorf

Problem

Given $n \leq 1000$ line segments that partition the plane in small regions. Are there two regions the same *distance* from the ocean?

Intended solution

- The answer is `no` iff all pairs of adjacent faces have opposite values.
- I.e.: the dual graph must be bipartite.
- That's true iff in each intersection point an even number of lines meet.

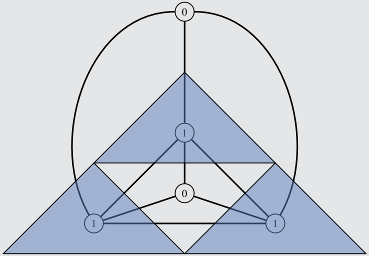


Problem

Given $n \leq 1000$ line segments that partition the plane in small regions. Are there two regions the same *distance* from the ocean?

Intended solution

- The answer is `no` iff all pairs of adjacent faces have opposite values.
- I.e.: the dual graph must be bipartite.
- That's true iff in each intersection point an even number of lines meet.
- Solution: check if each segment endpoint appears an even number of times in the input.



I: Isolated Island

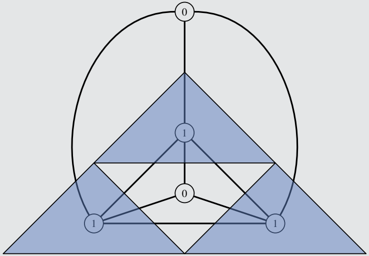
Problem Author: Michael Zündorf

Problem

Given $n \leq 1000$ line segments that partition the plane in small regions. Are there two regions the same *distance* from the ocean?

Intended solution

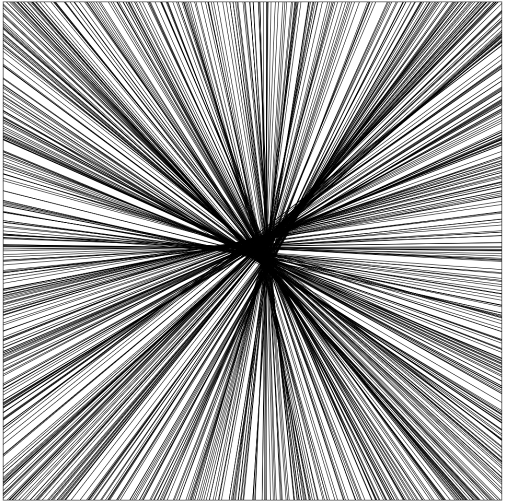
- The answer is `no` iff all pairs of adjacent faces have opposite values.
- I.e.: the dual graph must be bipartite.
- That's true iff in each intersection point an even number of lines meet.
- Solution: check if each segment endpoint appears an even number of times in the input.



Statistics: 25 submissions, 0 accepted, 21 unknown

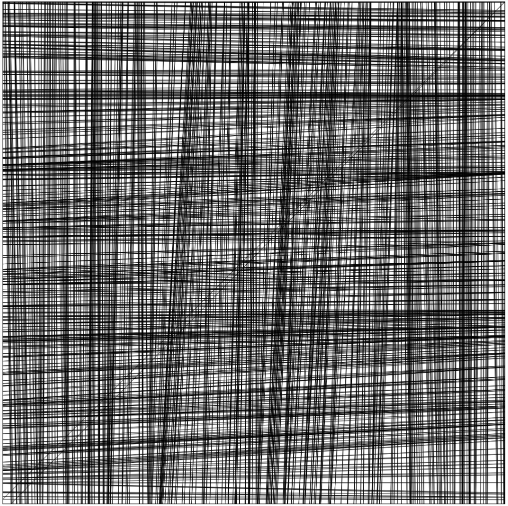
I: Isolated Island

Problem Author: Michael Zündorf



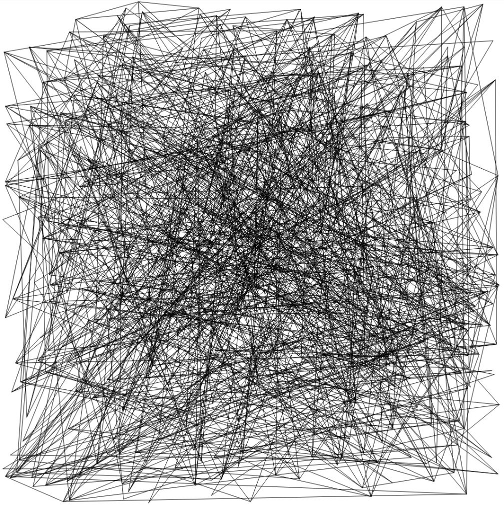
I: Isolated Island

Problem Author: Michael Zündorf

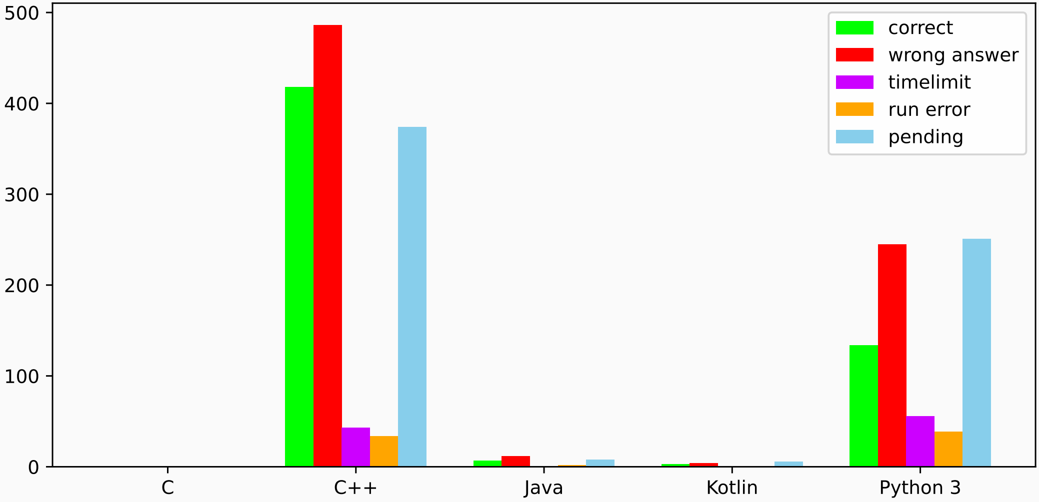


I: Isolated Island

Problem Author: Michael Zündorf



Language stats



Random facts

Jury work

- 723 commits (including test session) (last year: 720)

Random facts

Jury work

- 723 commits (including test session) (last year: 720)
- 1148 secret test cases (last year: 1424) ($95\frac{2}{3}$ per problem!)

Random facts

Jury work

- 723 commits (including test session) (last year: 720)
- 1148 secret test cases (last year: 1424) ($95\frac{2}{3}$ per problem!)
- 284 jury solutions (last year: 239)

Random facts

Jury work

- 723 commits (including test session) (last year: 720)
- 1148 secret test cases (last year: 1424) ($95\frac{2}{3}$ per problem!)
- 284 jury solutions (last year: 239)
- The minimum number of lines the jury needed to solve all problems is[1]

$$18 + 83 + 41 + 3 + 43 + 23 + 32 + 21 + 1 + 29 + 17 + 5 = 316$$

On average 26.3 lines per problem, up from 13.6 last year

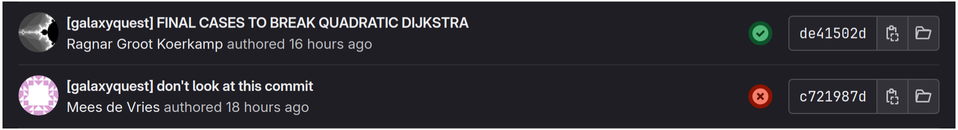
[1] But last year, we did more code golfing

Our final commits

[galaxyquest] don't look at this commit
Mees de Vries authored 18 hours ago

c721987d

Our final commits



Our final commits