Benelux Algorithm Programming Contest (BAPC) 2025

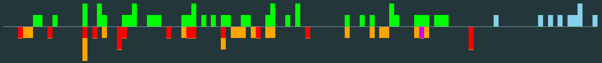
Solutions presentation

The BAPC 2025 Jury

October 26, 2025

A: Accidental Arithmetic

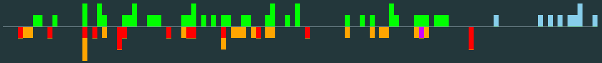
Problem author: Freek Henstra



Problem: Given a number n of $d \leq 1000$ digits. Between any two digits, insert a $+$ or a $-$ with 45% chance each. What is the expected value of the resulting expression?

A: Accidental Arithmetic

Problem author: Freek Henstra

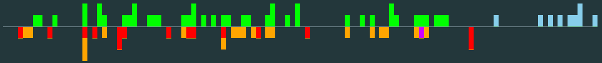


Problem: Given a number n of $d \leq 1000$ digits. Between any two digits, insert a $+$ or a $-$ with 45% chance each. What is the expected value of the resulting expression?

Observation: Since $+$ and $-$ are equally likely, anything after the first $+$ or $-$ has expected value 0.

A: Accidental Arithmetic

Problem author: Freek Henstra



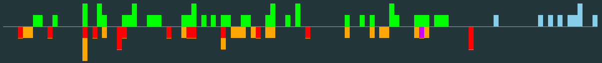
Problem: Given a number n of $d \leq 1000$ digits. Between any two digits, insert a + or a − with 45% chance each. What is the expected value of the resulting expression?

Observation: Since + and − are equally likely, anything after the first + or − has expected value 0.

Solution: The first term has k digits $(1 \leq k \leq d-1)$ with probability $0.1^{k-1} \cdot 0.9$, and d digits with probability 0.1^{d-1}. Multiplying by 0.1 just moves the decimal, so if, for example, $n = 1234$, the answer is $0.9 \cdot (1 + 1.2 + 1.23) + 1.234 = 4.321$.

A: Accidental Arithmetic

Problem author: Freek Henstra



Problem: Given a number n of $d \leq 1000$ digits. Between any two digits, insert a $\boxed{+}$ or a $\boxed{-}$ with 45% chance each. What is the expected value of the resulting expression?

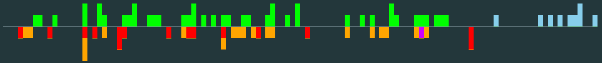
Observation: Since $\boxed{+}$ and $\boxed{-}$ are equally likely, anything after the first $\boxed{+}$ or $\boxed{-}$ has expected value 0.

Solution: The first term has k digits $(1 \leq k \leq d-1)$ with probability $0.1^{k-1} \cdot 0.9$, and d digits with probability 0.1^{d-1}. Multiplying by 0.1 just moves the decimal, so if, for example, $n = 1234$, the answer is $0.9 \cdot (1 + 1.2 + 1.23) + 1.234 = 4.321$.

Running time: $\mathcal{O}(d^2)$ naively, but can be optimized to $\mathcal{O}(d)$, though this was not necessary.

A: Accidental Arithmetic

Problem author: Freek Henstra



Problem: Given a number n of $d \leq 1000$ digits. Between any two digits, insert a $\boxed{+}$ or a $\boxed{-}$ with 45% chance each. What is the expected value of the resulting expression?

Observation: Since $\boxed{+}$ and $\boxed{-}$ are equally likely, anything after the first $\boxed{+}$ or $\boxed{-}$ has expected value 0.

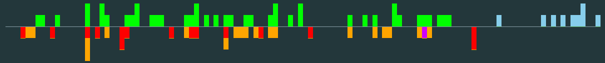
Solution: The first term has k digits $(1 \leq k \leq d-1)$ with probability $0.1^{k-1} \cdot 0.9$, and d digits with probability 0.1^{d-1}. Multiplying by 0.1 just moves the decimal, so if, for example, $n = 1234$, the answer is $0.9 \cdot (1 + 1.2 + 1.23) + 1.234 = 4.321$.

Running time: $\mathcal{O}(d^2)$ naively, but can be optimized to $\mathcal{O}(d)$, though this was not necessary.

Alternative: The kth digit is multiplied by $0.1^{k-1} \cdot (1 + 0.9(d - k + 1))$. It turns out, summing over only the first 20 digits suffice to have enough precision.

A: Accidental Arithmetic

Problem author: Freek Henstra



Problem: Given a number n of $d \leq 1000$ digits. Between any two digits, insert a $\boxed{+}$ or a $\boxed{-}$ with 45% chance each. What is the expected value of the resulting expression?

Observation: Since $\boxed{+}$ and $\boxed{-}$ are equally likely, anything after the first $\boxed{+}$ or $\boxed{-}$ has expected value 0.

Solution: The first term has k digits $(1 \leq k \leq d-1)$ with probability $0.1^{k-1} \cdot 0.9$, and d digits with probability 0.1^{d-1}. Multiplying by 0.1 just moves the decimal, so if, for example, $n = 1234$, the answer is $0.9 \cdot (1 + 1.2 + 1.23) + 1.234 = 4.321$.

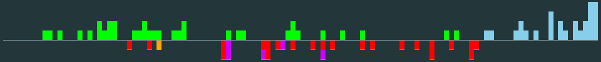
Running time: $\mathcal{O}(d^2)$ naively, but can be optimized to $\mathcal{O}(d)$, though this was not necessary.

Alternative: The kth digit is multiplied by $0.1^{k-1} \cdot (1 + 0.9(d-k+1))$. It turns out, summing over only the first 20 digits suffice to have enough precision.

Statistics: 87 submissions, 43 accepted, 9 unknown

B: Boggle Sort

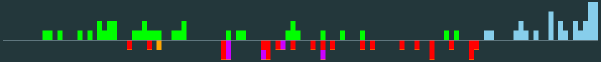
Problem author: Thore Husfeldt



Problem: Turn 16 six-sided dice in given order so that the top-facing sides are alphabetically ordered using as few turns as possible.

B: Boggle Sort

Problem author: Thore Husfeldt

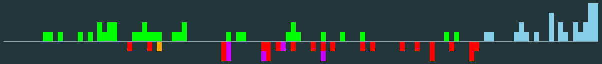


Problem: Turn 16 six-sided dice in given order so that the top-facing sides are alphabetically ordered using as few turns as possible.

Naive solution: There are $6^{16} > 2 \cdot 10^{12}$ possible turns; unoptimised brute-force will not work.

B: Boggle Sort

Problem author: Thore Husfeldt



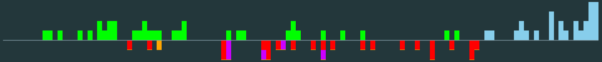
Problem: Turn 16 six-sided dice in given order so that the top-facing sides are alphabetically ordered using as few turns as possible.

Naive solution: There are $6^{16} > 2 \cdot 10^{12}$ possible turns; unoptimised brute-force will not work.

Greedy: Greedily turning the dice to the alphabetically earliest letter does not work; it (helpfully) fails on Sample 1.

B: Boggle Sort

Problem author: Thore Husfeldt



Problem: Turn 16 six-sided dice in given order so that the top-facing sides are alphabetically ordered using as few turns as possible.

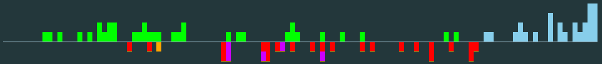
Naive solution: There are $6^{16} > 2 \cdot 10^{12}$ possible turns; unoptimised brute-force will not work.

Greedy: Greedily turning the dice to the alphabetically earliest letter does not work; it (helpfully) fails on Sample 1.

Brute-force-y sol.: Among the 4 sideways faces, it is optimal to take the alphabetically earliest face that still fits. Thus, there are really only 3 different choices per die: keep, turn on smallest side, turn bottom up. This gives $3^{16} = 43\,046\,721$ choices, which maybe can be systematically checked.

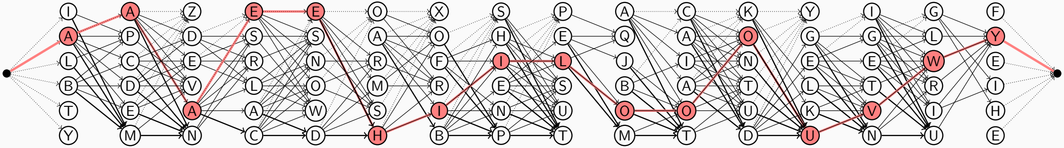
B: Boggle Sort

Problem author: Thore Husfeldt



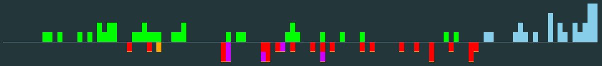
Graph-y solution: Create digraph with vertex set 6×16; connect (r, c) to $(r', c+1)$ if the character in (line l, column c) precedes the character in (line r', column $c+1$) in the alphabet. The weight is 0 if $r = 1$ (dotted), 2 if $r = 6$ (fat), and 1 otherwise. Connect s to $(r, 1)$ and $(r, 16)$ to t. Then a minimum-weight s, t-path is the solution.

Optimal solution for Sample 1:



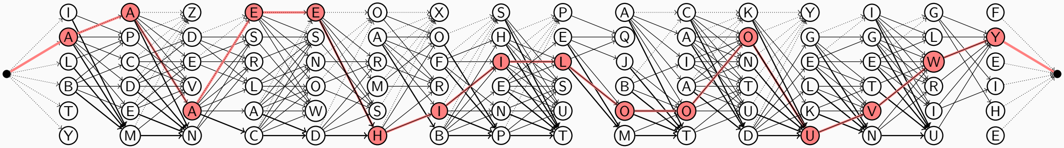
B: Boggle Sort

Problem author: Thore Husfeldt



Graph-y solution: Create digraph with vertex set 6×16; connect (r, c) to $(r', c+1)$ if the character in (line l, column c) precedes the character in (line r', column $c+1$) in the alphabet. The weight is 0 if $r = 1$ (dotted), 2 if $r = 6$ (fat), and 1 otherwise. Connect s to $(r, 1)$ and $(r, 16)$ to t. Then a minimum-weight s, t-path is the solution.

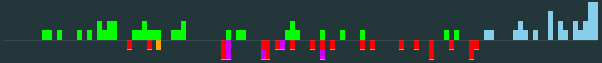
Optimal solution for Sample 1:



Pitfall: There should be no edge from `Q` to `T`, even though `Q`<`T` (because it should be treated as `QU` and `U`>`T`). Ignoring this leads (helpfully) to a wrong answer on Sample 1.

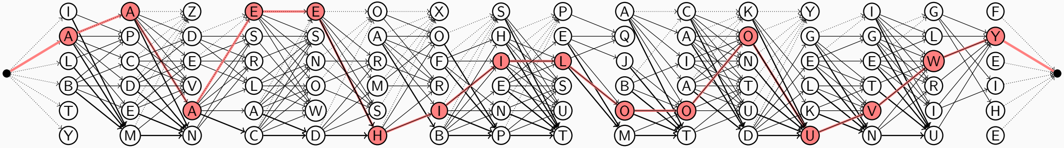
B: Boggle Sort

Problem author: Thore Husfeldt



Graph-y solution: Create digraph with vertex set 6×16; connect (r, c) to $(r', c+1)$ if the character in (line l, column c) precedes the character in (line r', column $c+1$) in the alphabet. The weight is 0 if $r = 1$ (dotted), 2 if $r = 6$ (fat), and 1 otherwise. Connect s to $(r, 1)$ and $(r, 16)$ to t. Then a minimum-weight s, t-path is the solution.

Optimal solution for Sample 1:

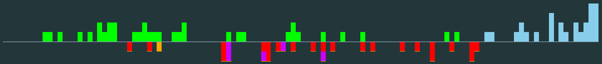


Pitfall: There should be no edge from `Q` to `T`, even though `Q<T` (because it should be treated as `QU` and `U>T`). Ignoring this leads (helpfully) to a wrong answer on Sample 1.

Running time: Using Dijkstra's algorithm, time $\mathcal{O}(rc \log rc)$ for c dice with r faces. But graph is acyclic, so actually $\mathcal{O}(rc)$.

B: Boggle Sort

Problem author: Thore Husfeldt



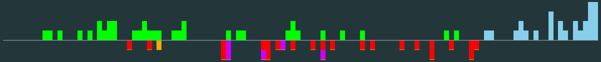
DP: Compute, for $1 \leq i \leq 16$, and each letter x, the smallest number $f(i,x)$ of turns needed to bring the first i dice into nondecreasing order such that the ith die shows x. Then, if x appears on the ith die, we have the general case

$$f(i,x) = \max_{y \leq x} f(i-1,y)$$

where y ranges over all letters appearing on die $(i-1)$. (Remember the `Q`=`QU` pitfall.)

B: Boggle Sort

Problem author: Thore Husfeldt



DP: Compute, for $1 \le i \le 16$, and each letter x, the smallest number $f(i, x)$ of turns needed to bring the first i dice into nondecreasing order such that the ith die shows x. Then, if x appears on the ith die, we have the general case

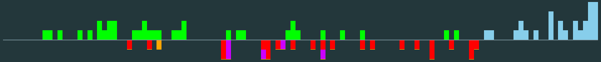
$$f(i, x) = \max_{y \le x} f(i-1, y)$$

where y ranges over all letters appearing on die $(i-1)$. (Remember the $\texttt{Q}{=}\texttt{QU}$ pitfall.)

Running time: $\mathcal{O}(rc)$ for c dice with r faces.

B: Boggle Sort

Problem author: Thore Husfeldt



DP: Compute, for $1 \leq i \leq 16$, and each letter x, the smallest number $f(i, x)$ of turns needed to bring the first i dice into nondecreasing order such that the ith die shows x. Then, if x appears on the ith die, we have the general case

$$f(i, x) = \max_{y \leq x} f(i-1, y)$$

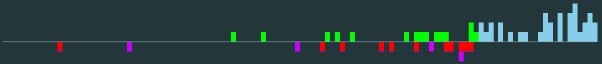
where y ranges over all letters appearing on die $(i-1)$. (Remember the `Q`=`QU` pitfall.)

Running time: $\mathcal{O}(rc)$ for c dice with r faces.

Statistics: 89 submissions, 35 accepted, 26 unknown

C: Coherency

Problem author: Thore Husfeldt



Problem: Given *n* models on a gaming board, represented as non-overlapping disks with diameter between 25 and 165 mm. Check coherency.

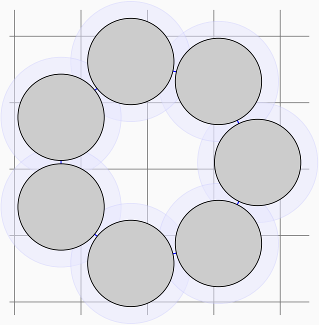
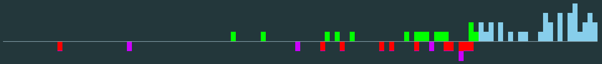


Figure 1: Example board configuration (example 4)

C: Coherency

Problem author: Thore Husfeldt



Problem: Given n models on a gaming board, represented as non-overlapping disks with diameter between 25 and 165 mm. Check coherency.

Models are *coherent* if they can reach each other. Models are adjacent when ≤ 2 inches apart. For $n \geq 7$, each model must have at least two neighbors, for coherency.

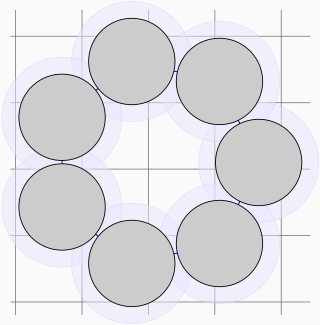


Figure 1: Example board configuration (example 4)

Problem: Given n models on a gaming board, represented as non-overlapping disks with diameter between 25 and 165 mm. Check coherency.

Models are *coherent* if they can reach each other. Models are adjacent when ≤ 2 inches apart. For $n \geq 7$, each model must have at least two neighbors, for coherency.

Naive solution: Check adjacency for all pairs of models in $\mathcal{O}(n^2)$.

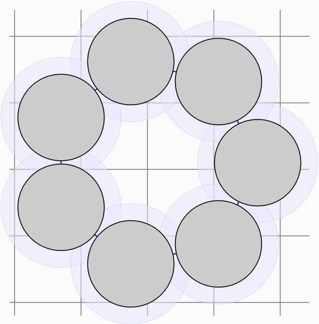


Figure 1: Example board configuration (example 4)

C: Coherency

Problem author: Thore Husfeldt

Problem: Given n models on a gaming board, represented as non-overlapping disks with diameter between 25 and 165 mm. Check coherency.

Models are *coherent* if they can reach each other. Models are adjacent when ≤ 2 inches apart. For $n \geq 7$, each model must have at least two neighbors, for coherency.

Naive solution: Check adjacency for all pairs of models in $\mathcal{O}(n^2)$.

Make a graph of n nodes, and represent adjacency as undirected edges.

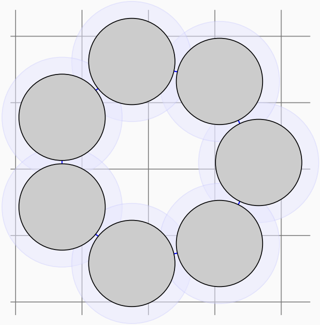
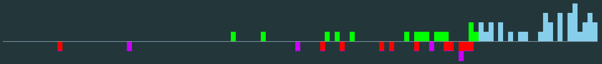


Figure 1: Example board configuration (example 4)

Problem author: Thore Husfeldt



Problem: Given n models on a gaming board, represented as non-overlapping disks with diameter between 25 and 165 mm. Check coherency.

Models are *coherent* if they can reach each other. Models are adjacent when ≤ 2 inches apart. For $n \geq 7$, each model must have at least two neighbors, for coherency.

Naive solution: Check adjacency for all pairs of models in $\mathcal{O}(n^2)$.

Make a graph of n nodes, and represent adjacency as undirected edges.

Run your favourite algorithm for finding connected components, and check degrees, for $n \geq 7$.

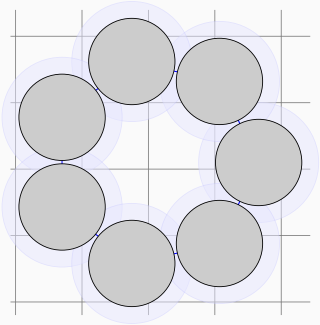
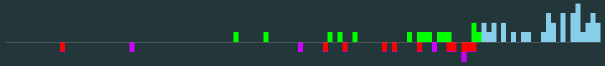


Figure 1: Example board configuration (example 4)

C: Coherency

Problem author: Thore Husfeldt



Problem: Given n models on a gaming board, represented as non-overlapping disks with diameter between 25 and 165 mm. Check coherency.

Models are *coherent* if they can reach each other. Models are adjacent when ≤ 2 inches apart. For $n \geq 7$, each model must have at least two neighbors, for coherency.

Naive solution: Check adjacency for all pairs of models in $\mathcal{O}(n^2)$.

Make a graph of n nodes, and represent adjacency as undirected edges.

Run your favourite algorithm for finding connected components, and check degrees, for $n \geq 7$.

In total, $\mathcal{O}(n^2)$. This is too slow, as $n \leq 200000$.

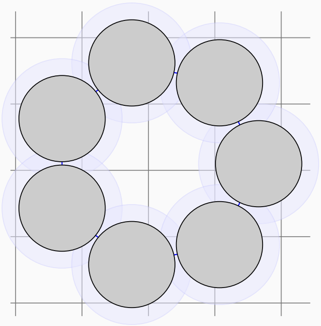


Figure 1: Example board configuration (example 4)

Idea: Use a grid of cells of 211 x 211 mm.

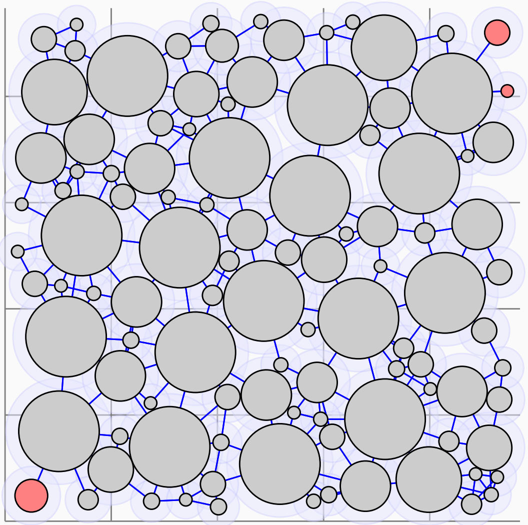


Figure 2: Secret testcase

Idea: Use a grid of cells of 211 x 211 mm.

Centres of disks are placed into corresponding cell. This can be done with a map / dictionary, in $\mathcal{O}(n \log n)$ or $\mathcal{O}(n)$.

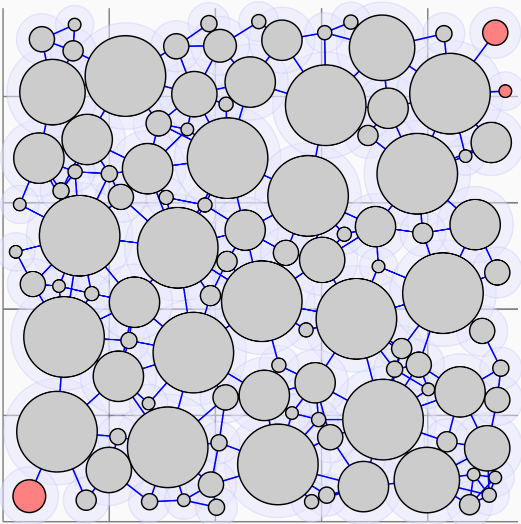
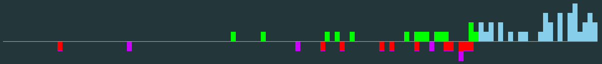


Figure 2: Secret testcase



Idea: Use a grid of cells of 211 x 211 mm.

Centres of disks are placed into corresponding cell. This can be done with a map / dictionary, in $\mathcal{O}(n \log n)$ or $\mathcal{O}(n)$.

Observation: Disks do not influence disks in non-adjacent cells (8-adjacency). (this is why 211 mm is chosen)

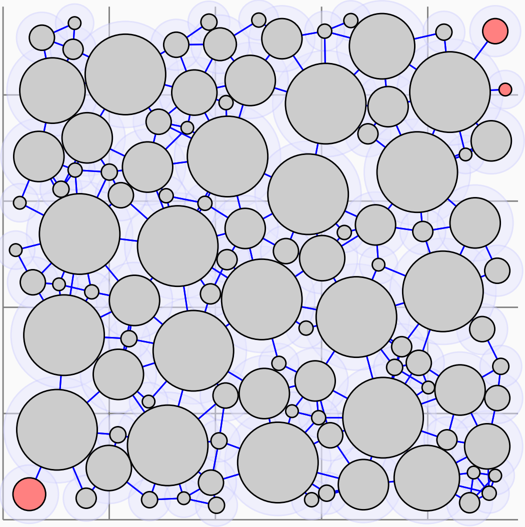
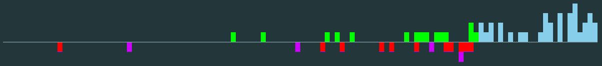


Figure 2: Secret testcase



Idea: Use a grid of cells of 211 x 211 mm.

Centres of disks are placed into corresponding cell. This can be done with a map / dictionary, in $\mathcal{O}(n \log n)$ or $\mathcal{O}(n)$.

Observation: Disks do not influence disks in non-adjacent cells (8-adjacency). (this is why 211 mm is chosen)

Algorithm: For each disk, loop through all 8 adjacent cells, and its own cell, and check all candidate disks for adjacency.
DFS, BFS or DSU can be used to find the connected components.

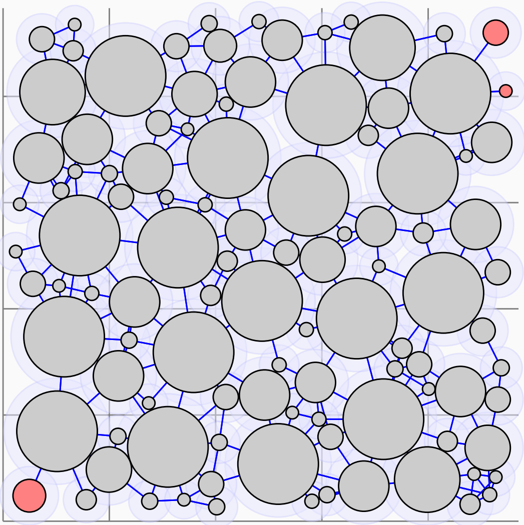


Figure 2: Secret testcase

Idea: Use a grid of cells of 211 x 211 mm.

Centres of disks are placed into corresponding cell. This can be done with a map / dictionary, in $\mathcal{O}(n \log n)$ or $\mathcal{O}(n)$.

Observation: Disks do not influence disks in non-adjacent cells (8-adjacency). (this is why 211 mm is chosen)

Algorithm: For each disk, loop through all 8 adjacent cells, and its own cell, and check all candidate disks for adjacency.
DFS, BFS or DSU can be used to find the connected components.

Time complexity?

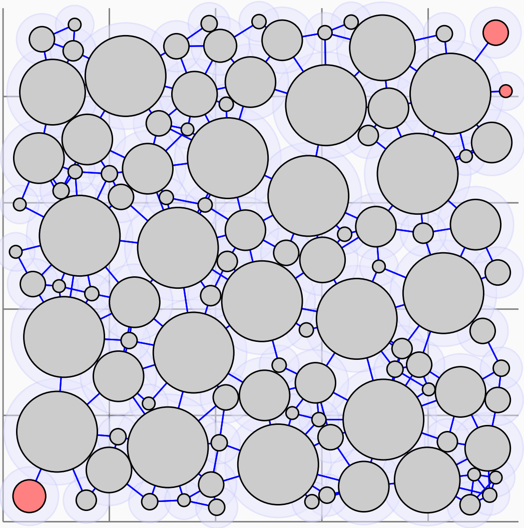
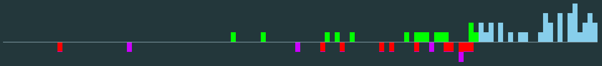


Figure 2: Secret testcase

C: Coherency

Problem author: Thore Husfeldt



Idea: Use a grid of cells of 211 x 211 mm.

Centres of disks are placed into corresponding cell. This can be done with a map / dictionary, in $\mathcal{O}(n \log n)$ or $\mathcal{O}(n)$.

Observation: Disks do not influence disks in non-adjacent cells (8-adjacency). (this is why 211 mm is chosen)

Algorithm: For each disk, loop through all 8 adjacent cells, and its own cell, and check all candidate disks for adjacency.
DFS, BFS or DSU can be used to find the connected components.

Time complexity?

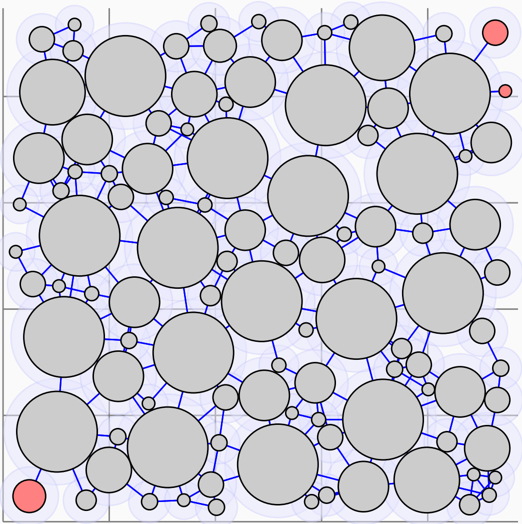


Figure 2: Secret testcase

Time complexity: Time complexity is
$\mathcal{O}(n \times 9 \times \text{Max number of disks in one cell})$

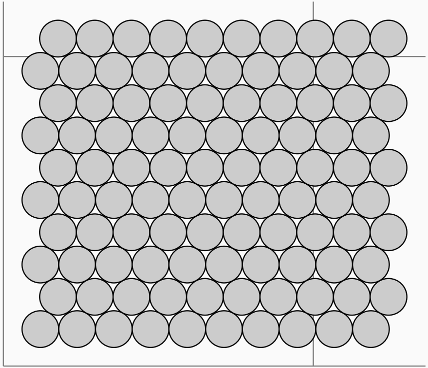


Figure 3: Worst case triangular packing with smallest diameter.

Time complexity: Time complexity is

$\mathcal{O}(n \times 9 \times \text{Max number of disks in one cell})$

Roughly $\mathcal{O}(900 \times n)$

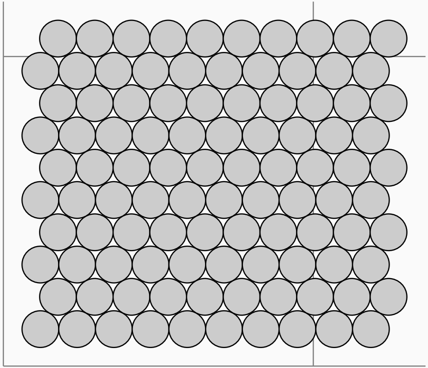
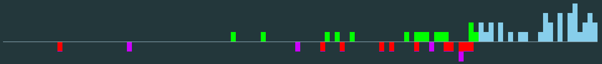


Figure 3: Worst case triangular packing with smallest diameter.

C: Coherency

Problem author: Thore Husfeldt



Time complexity: Time complexity is

$\mathcal{O}(n \times 9 \times \text{Max number of disks in one cell})$

Roughly $\mathcal{O}(900 \times n)$

More precise: Cells need to be $\Omega(D_{\text{max}})$ mm big, so $\mathcal{O}\left(\left(\frac{D_{\text{max}}}{D_{\text{min}}}\right)^2\right)$ disks of the minimum diameter fit inside one cell

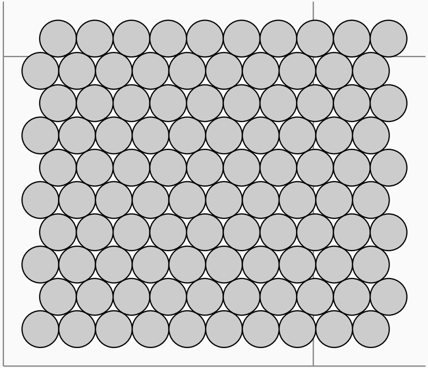
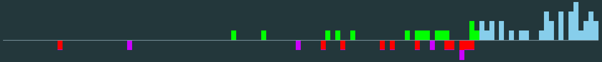


Figure 3: Worst case triangular packing with smallest diameter.

C: Coherency

Problem author: Thore Husfeldt



Time complexity: Time complexity is

$\mathcal{O}(n \times 9 \times \text{Max number of disks in one cell})$

Roughly $\mathcal{O}(900 \times n)$

More precise: Cells need to be $\Omega(D_{\text{max}})$ mm big, so $\mathcal{O}\left(\left(\frac{D_{\text{max}}}{D_{\text{min}}}\right)^2\right)$ disks of the minimum diameter fit inside one cell

Many alternative solutions possible, but based on similar ideas.

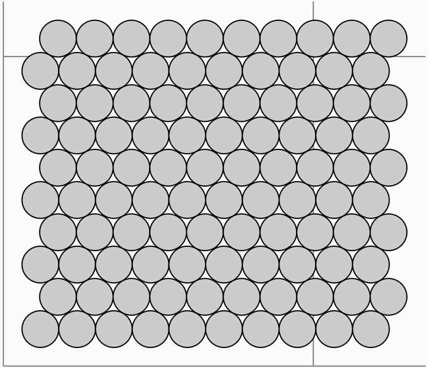


Figure 3: Worst case triangular packing with smallest diameter.

C: Coherency

Problem author: Thore Husfeldt

Time complexity: Time complexity is
$\mathcal{O}(n \times 9 \times \text{Max number of disks in one cell})$

Roughly $\mathcal{O}(900 \times n)$

More precise: Cells need to be $\Omega(D_{\text{max}})$ mm big, so $\mathcal{O}\left(\left(\frac{D_{\text{max}}}{D_{\text{min}}}\right)^2\right)$ disks of the minimum diameter fit inside one cell

Many alternative solutions possible, but based on similar ideas.

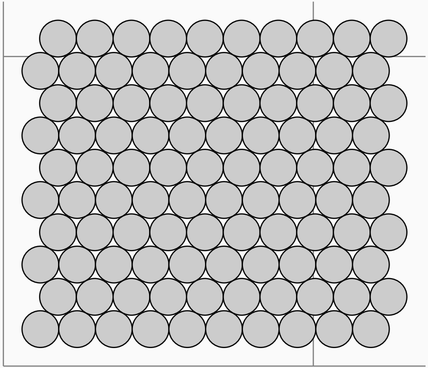
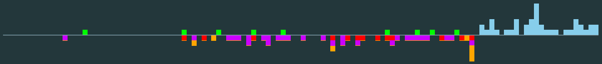


Figure 3: Worst case triangular packing with smallest diameter.

Statistics: 64 submissions, 9 accepted, 34 unknown

D: Duo Detection

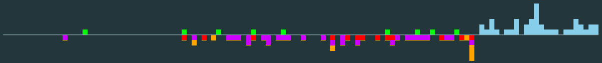
Problem author: Mike de Vries



Problem: Given n messages, M_i, find two messages which have at least 2 symbols in common. The total size of all messages is not more than $m = 100000$.

D: Duo Detection

Problem author: Mike de Vries

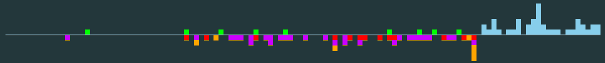


Problem: Given n messages, M_i, find two messages which have at least 2 symbols in common. The total size of all messages is not more than $m = 100000$.

Observation: Build a bipartite graph with on one side the messages, and on the other the symbols. Want to find a 4-cycle in the graph.

D: Duo Detection

Problem author: Mike de Vries



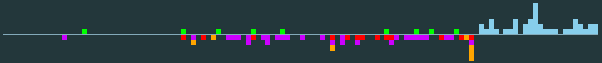
Problem: Given n messages, M_i, find two messages which have at least 2 symbols in common. The total size of all messages is not more than $m = 100000$.

Observation: Build a bipartite graph with on one side the messages, and on the other the symbols. Want to find a 4-cycle in the graph.

Naive solution 1: Loop over all pairs of messages, and calculate the intersection of their symbol sets in $\mathcal{O}(\min(|M_i|, |M_j|))$, using hash sets or boolean arrays after using coordinate compression. Time complexity, roughly: $\mathcal{O}(n \sum_{i=1}^{n} |M_i|)$

D: Duo Detection

Problem author: Mike de Vries



Problem: Given n messages, M_i, find two messages which have at least 2 symbols in common. The total size of all messages is not more than $m = 100000$.

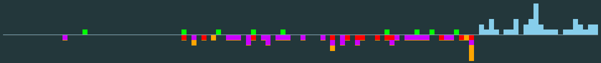
Observation: Build a bipartite graph with on one side the messages, and on the other the symbols. Want to find a 4-cycle in the graph.

Naive solution 1: Loop over all pairs of messages, and calculate the intersection of their symbol sets in $\mathcal{O}(\min(|M_i|, |M_j|))$, using hash sets or boolean arrays after using coordinate compression. Time complexity, roughly: $\mathcal{O}(n \sum_{i=1}^{n} |M_i|)$

When n is small, works well, but can be quadratic.

D: Duo Detection

Problem author: Mike de Vries



Problem: Given n messages, M_i, find two messages which have at least 2 symbols in common. The total size of all messages is not more than $m = 100000$.

Observation: Build a bipartite graph with on one side the messages, and on the other the symbols. Want to find a 4-cycle in the graph.

Naive solution 1: Loop over all pairs of messages, and calculate the intersection of their symbol sets in $\mathcal{O}(\min(|M_i|, |M_j|))$, using hash sets or boolean arrays after using coordinate compression. Time complexity, roughly: $\mathcal{O}(n \sum_{i=1}^{n} |M_i|)$

When n is small, works well, but can be quadratic.

Naive solution 2: Loop over all pairs of symbols in each message. Check if any of these pairs is the same, by using a hash map. Time complexity: $\mathcal{O}(\sum_{i=1}^{n} |M_i|^2)$ Works well when the sizes of the messages are small.

Reminder: Naive 1: $\mathcal{O}(\min(|M_i|, |M_j|))$ over all pairs of messages.
Naive 2: $\mathcal{O}(\sum_{i=1}^{n} |M_i|^2)$

D: Duo Detection

Problem author: Mike de Vries

Reminder: Naive 1: $\mathcal{O}(\min(|M_i|, |M_j|))$ over all pairs of messages.
Naive 2: $\mathcal{O}(\sum_{i=1}^{n} |M_i|^2)$

Solution: The time limit and constraints are generous. Combine the naive solutions in a smart way to obtain a faster algorithm in the worst case.
Divide messages in big and small messages, based on parameter B. Do casework:

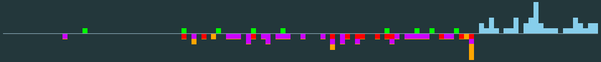
Reminder: Naive 1: $\mathcal{O}(\min(|M_i|, |M_j|))$ over all pairs of messages.
Naive 2: $\mathcal{O}(\sum_{i=1}^{n} |M_i|^2)$

Solution: The time limit and constraints are generous. Combine the naive solutions in a smart way to obtain a faster algorithm in the worst case.
Divide messages in big and small messages, based on parameter B. Do casework:

Big-Big Use naive solution 1. Works in $\mathcal{O}(\sum_{i \in \text{Bigs}} |M_i| \frac{m}{B}) = \mathcal{O}(\frac{m^2}{B})$

D: Duo Detection

Problem author: Mike de Vries



Reminder: Naive 1: $\mathcal{O}(\min(|M_i|, |M_j|))$ over all pairs of messages.
Naive 2: $\mathcal{O}(\sum_{i=1}^{n} |M_i|^2)$

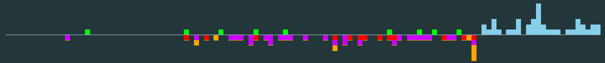
Solution: The time limit and constraints are generous. Combine the naive solutions in a smart way to obtain a faster algorithm in the worst case.
Divide messages in big and small messages, based on parameter B. Do casework:

Big-Big Use naive solution 1. Works in $\mathcal{O}(\sum_{i \in \text{Bigs}} |M_i| \frac{m}{B}) = \mathcal{O}(\frac{m^2}{B})$

Small-Big Use naive solution 1. Because of the minimum over the two message lengths, still $\mathcal{O}(\frac{m^2}{B})$

D: Duo Detection

Problem author: Mike de Vries



Reminder: Naive 1: $\mathcal{O}(\min(|M_i|, |M_j|))$ over all pairs of messages.
Naive 2: $\mathcal{O}(\sum_{i=1}^{n} |M_i|^2)$

Solution: The time limit and constraints are generous. Combine the naive solutions in a smart way to obtain a faster algorithm in the worst case.
Divide messages in big and small messages, based on parameter B. Do casework:

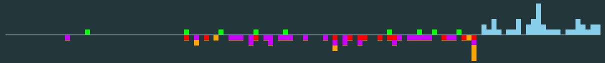
Big-Big Use naive solution 1. Works in $\mathcal{O}(\sum_{i \in \text{Bigs}} |M_i| \frac{m}{B}) = \mathcal{O}(\frac{m^2}{B})$

Small-Big Use naive solution 1. Because of the minimum over the two message lengths, still $\mathcal{O}(\frac{m^2}{B})$

Small-Small Use naive solution 2, works in $\mathcal{O}(mB)$ (worst case is all small messages are equal in size and $< B$.)
This handles all the cases, and total complexity is $\mathcal{O}(\frac{m^2}{B} + mB)$, best when $B = \Theta(\sqrt{m})$.

D: Duo Detection

Problem author: Mike de Vries



Reminder: Naive 1: $\mathcal{O}(\min(|M_i|, |M_j|))$ over all pairs of messages.
Naive 2: $\mathcal{O}(\sum_{i=1}^{n} |M_i|^2)$

Solution: The time limit and constraints are generous. Combine the naive solutions in a smart way to obtain a faster algorithm in the worst case.
Divide messages in big and small messages, based on parameter B. Do casework:

Big-Big Use naive solution 1. Works in $\mathcal{O}(\sum_{i \in \text{Bigs}} |M_i| \frac{m}{B}) = \mathcal{O}(\frac{m^2}{B})$

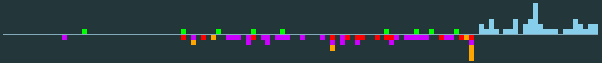
Small-Big Use naive solution 1. Because of the minimum over the two message lengths, still $\mathcal{O}(\frac{m^2}{B})$

Small-Small Use naive solution 2, works in $\mathcal{O}(mB)$ (worst case is all small messages are equal in size and $< B$.)
This handles all the cases, and total complexity is $\mathcal{O}(\frac{m^2}{B} + mB)$, best when $B = \Theta(\sqrt{m})$.

Running time: $\mathcal{O}(m\sqrt{m})$ with a hash map. The algorithm has a considerable constant factor.

D: Duo Detection

Problem author: Mike de Vries



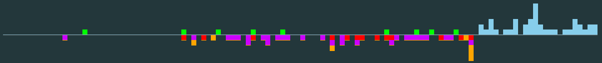
Bonus: You can get rid of the hash map. This requires in naive solution 2 to order the computations in a smart way, such that a global boolean array can be used, which is set and unset for each message. Also requires sorting and coordinate compression beforehand. Same tricks need to be used in naive solution 1.

Bonus: You can get rid of the hash map. This requires in naive solution 2 to order the computations in a smart way, such that a global boolean array can be used, which is set and unset for each message. Also requires sorting and coordinate compression beforehand. Same tricks need to be used in naive solution 1.

Bonus 2: The algorithm can be sped up to $\mathcal{O}(m\sqrt{\frac{m}{w}})$, where w is the word size of the machine (typically 32 or 64). This can be done with the use of bitsets in naive algorithm 1. The details are an exercise for the interested reader.

D: Duo Detection

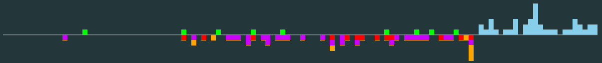
Problem author: Mike de Vries



Bonus: You can get rid of the hash map. This requires in naive solution 2 to order the computations in a smart way, such that a global boolean array can be used, which is set and unset for each message. Also requires sorting and coordinate compression beforehand. Same tricks need to be used in naive solution 1.

Bonus 2: The algorithm can be sped up to $\mathcal{O}(m\sqrt{\frac{m}{w}})$, where w is the word size of the machine (typically 32 or 64). This can be done with the use of bitsets in naive algorithm 1. The details are an exercise for the interested reader.

Alternative solution: A time complexity of $\mathcal{O}(\frac{m^2}{w})$ can also get accepted when implemented well. (This is basically doing bonus 2 without the square-root trick).



Bonus: You can get rid of the hash map. This requires in naive solution 2 to order the computations in a smart way, such that a global boolean array can be used, which is set and unset for each message. Also requires sorting and coordinate compression beforehand. Same tricks need to be used in naive solution 1.

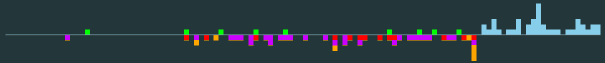
Bonus 2: The algorithm can be sped up to $\mathcal{O}(m\sqrt{\frac{m}{w}})$, where w is the word size of the machine (typically 32 or 64). This can be done with the use of bitsets in naive algorithm 1. The details are an exercise for the interested reader.

Alternative solution: A time complexity of $\mathcal{O}(\frac{m^2}{w})$ can also get accepted when implemented well. (This is basically doing bonus 2 without the square-root trick).

Fun fact: Constructions similar to problem F (Faulty Connection) are used in the testdata to make dense testcases with a impossible answer.

D: Duo Detection

Problem author: Mike de Vries



Bonus: You can get rid of the hash map. This requires in naive solution 2 to order the computations in a smart way, such that a global boolean array can be used, which is set and unset for each message. Also requires sorting and coordinate compression beforehand. Same tricks need to be used in naive solution 1.

Bonus 2: The algorithm can be sped up to $\mathcal{O}(m\sqrt{\frac{m}{w}})$, where w is the word size of the machine (typically 32 or 64). This can be done with the use of bitsets in naive algorithm 1. The details are an exercise for the interested reader.

Alternative solution: A time complexity of $\mathcal{O}(\frac{m^2}{w})$ can also get accepted when implemented well. (This is basically doing bonus 2 without the square-root trick).

Fun fact: Constructions similar to problem F (Faulty Connection) are used in the testdata to make dense testcases with a impossible answer.

Statistics: 98 submissions, 9 accepted, 40 unknown

Problem: Find the fastest possible time to visit floors $f_1, \ldots, f_n$ with optimal starting configuration of the elevators.

E: Excruciating Elevators

Problem author: Albert Eikelenboom

Problem: Find the fastest possible time to visit floors $f_1, \ldots, f_n$ with optimal starting configuration of the elevators.

Observation 1: Label the elevators A,B,C,D. We can assume that A starts on floor 0. Indeed, if no elevator starts on floor 0, we can move each elevator ahead until one does.

E: Excruciating Elevators

Problem author: Albert Eikelenboom

Problem: Find the fastest possible time to visit floors $f_1, \ldots, f_n$ with optimal starting configuration of the elevators.

Observation 1: Label the elevators A,B,C,D. We can assume that A starts on floor 0. Indeed, if no elevator starts on floor 0, we can move each elevator ahead until one does.

Observation 2: Consider a directed graph on the elevators, by drawing an edge from X to Y whenever X arrives perfectly on time at some point where Y was used most recently. Then from each used elevator, there must be a path in this graph to elevator A. Otherwise, we can move all reachable elevators ahead until a new edge emerges.

E: Excruciating Elevators

Problem author: Albert Eikelenboom

Problem: Find the fastest possible time to visit floors $f_1, \ldots, f_n$ with optimal starting configuration of the elevators.

Observation 1: Label the elevators A,B,C,D. We can assume that A starts on floor 0. Indeed, if no elevator starts on floor 0, we can move each elevator ahead until one does.

Observation 2: Consider a directed graph on the elevators, by drawing an edge from X to Y whenever X arrives perfectly on time at some point where Y was used most recently. Then from each used elevator, there must be a path in this graph to elevator A. Otherwise, we can move all reachable elevators ahead until a new edge emerges.

Conclusion: We can assume there is an edge from B to A, from C to at least one of A or B, and from D to at least one of A or B or C, giving 6 graphs to consider.

E: Excruciating Elevators

Problem author: Albert Eikelenboom

Problem: Find the fastest possible time to visit floors $f_1, \ldots, f_n$ with optimal starting configuration of the elevators.

Observation 1: Label the elevators A,B,C,D. We can assume that A starts on floor 0. Indeed, if no elevator starts on floor 0, we can move each elevator ahead until one does.

Observation 2: Consider a directed graph on the elevators, by drawing an edge from X to Y whenever X arrives perfectly on time at some point where Y was used most recently. Then from each used elevator, there must be a path in this graph to elevator A. Otherwise, we can move all reachable elevators ahead until a new edge emerges.

Conclusion: We can assume there is an edge from B to A, from C to at least one of A or B, and from D to at least one of A or B or C, giving 6 graphs to consider.

Solution: Iterate all 6 graphs and all n^3 possible floors where the perfect transitions take place. This determines the starting positions of all elevators, from which we can simulate the corresponding solution.

E: Excruciating Elevators

Problem author: Albert Eikelenboom

Problem: Find the fastest possible time to visit floors $f_1, \ldots, f_n$ with optimal starting configuration of the elevators.

Observation 1: Label the elevators A,B,C,D. We can assume that A starts on floor 0. Indeed, if no elevator starts on floor 0, we can move each elevator ahead until one does.

Observation 2: Consider a directed graph on the elevators, by drawing an edge from X to Y whenever X arrives perfectly on time at some point where Y was used most recently. Then from each used elevator, there must be a path in this graph to elevator A. Otherwise, we can move all reachable elevators ahead until a new edge emerges.

Conclusion: We can assume there is an edge from B to A, from C to at least one of A or B, and from D to at least one of A or B or C, giving 6 graphs to consider.

Solution: Iterate all 6 graphs and all n^3 possible floors where the perfect transitions take place. This determines the starting positions of all elevators, from which we can simulate the corresponding solution.

Running time: For k elevators, with $\mathcal{O}(kn)$ time per simulation, the running time is $\mathcal{O}(k!n^k)$.

E: Excruciating Elevators

Problem author: Albert Eikelenboom

Problem: Find the fastest possible time to visit floors $f_1, \ldots, f_n$ with optimal starting configuration of the elevators.

Observation 1: Label the elevators A,B,C,D. We can assume that A starts on floor 0. Indeed, if no elevator starts on floor 0, we can move each elevator ahead until one does.

Observation 2: Consider a directed graph on the elevators, by drawing an edge from X to Y whenever X arrives perfectly on time at some point where Y was used most recently. Then from each used elevator, there must be a path in this graph to elevator A. Otherwise, we can move all reachable elevators ahead until a new edge emerges.

Conclusion: We can assume there is an edge from B to A, from C to at least one of A or B, and from D to at least one of A or B or C, giving 6 graphs to consider.

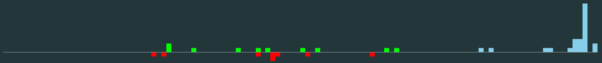
Solution: Iterate all 6 graphs and all n^3 possible floors where the perfect transitions take place. This determines the starting positions of all elevators, from which we can simulate the corresponding solution.

Running time: For k elevators, with $\mathcal{O}(kn)$ time per simulation, the running time is $\mathcal{O}(k!n^k)$.

Statistics: 16 submissions, 1 accepted, 7 unknown

F: Faulty Connection

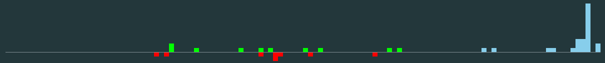
Problem author: Mike de Vries



Problem: Find 600 sets of 30 integers from 1 to 1000 such that no two sets intersect in two or more integers.

F: Faulty Connection

Problem author: Mike de Vries

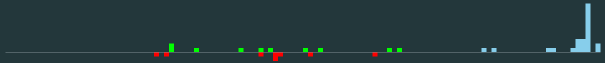


Problem: Find 600 sets of 30 integers from 1 to 1000 such that no two sets intersect in two or more integers.

Idea: Any two points on the plane define a unique line, and any two lines have at most one intersection point. We can assign lines to messages, and points on those lines to numbers. Since two different lines have at most one intersection point, two messages have at most one number in common.

F: Faulty Connection

Problem author: Mike de Vries



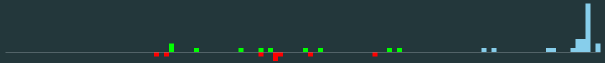
Problem: Find 600 sets of 30 integers from 1 to 1000 such that no two sets intersect in two or more integers.

Idea: Any two points on the plane define a unique line, and any two lines have at most one intersection point. We can assign lines to messages, and points on those lines to numbers. Since two different lines have at most one intersection point, two messages have at most one number in common.

Solution: Use the lines in $\mathbb{F}_{31}^2$.

F: Faulty Connection

Problem author: Mike de Vries

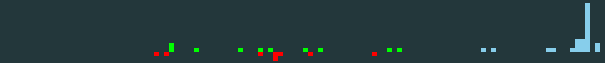


In English: Take a 31×31 grid and perform all arithmetic modulo 31 (make it wrap around the boundaries like in the game of Asteroids). Use lines of the form $y = ax + b$ (with $a, b \in \{0, 1, ..., 30\}$).

Assign a line to each message, and a number to each point.

F: Faulty Connection

Problem author: Mike de Vries

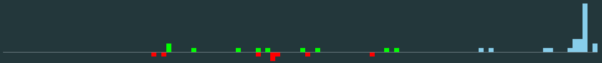


In English: Take a 31×31 grid and perform all arithmetic modulo 31 (make it wrap around the boundaries like in the game of Asteroids). Use lines of the form $y = ax + b$ (with $a, b \in \{0, 1, ..., 30\}$).
Assign a line to each message, and a number to each point.

Note: While lines may intersect multiple times when wrapping around, the fact that 31 is prime means that two different lines will intersect in at most one integer coordinate.

F: Faulty Connection

Problem author: Mike de Vries



In English: Take a 31×31 grid and perform all arithmetic modulo 31 (make it wrap around the boundaries like in the game of Asteroids). Use lines of the form $y = ax + b$ (with $a, b \in \{0, 1, ..., 30\}$).

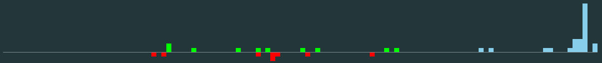
Assign a line to each message, and a number to each point.

Note: While lines may intersect multiple times when wrapping around, the fact that 31 is prime means that two different lines will intersect in at most one integer coordinate.

Analysis: There are $31 \times 31 \leq 1000$ points, and $31 \times 31 \geq 600$ such lines, with $31 \geq 30$ points each.

F: Faulty Connection

Problem author: Mike de Vries



In English: Take a 31×31 grid and perform all arithmetic modulo 31 (make it wrap around the boundaries like in the game of Asteroids). Use lines of the form $y = ax + b$ (with $a, b \in \{0, 1, ..., 30\}$).
Assign a line to each message, and a number to each point.

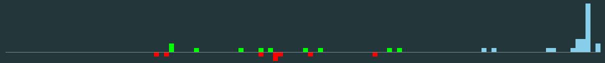
Note: While lines may intersect multiple times when wrapping around, the fact that 31 is prime means that two different lines will intersect in at most one integer coordinate.

Analysis: There are $31 \times 31 \leq 1000$ points, and $31 \times 31 \geq 600$ such lines, with $31 \geq 30$ points each.

Bonus: It is possible to solve the problem for $31^2 + 31 + 1 = 993$ messages of 32 numbers using only numbers up to 993.

F: Faulty Connection

Problem author: Mike de Vries



Example: One message may correspond to the red line, and another to blue. Receiving the points $(2, 28)$ and $(16, 8)$, the red line can be reconstructed using modular arithmetic.
The red and blue line ($y = 1x + 5$) intersect in exactly one integer point: $(7, 12)$.

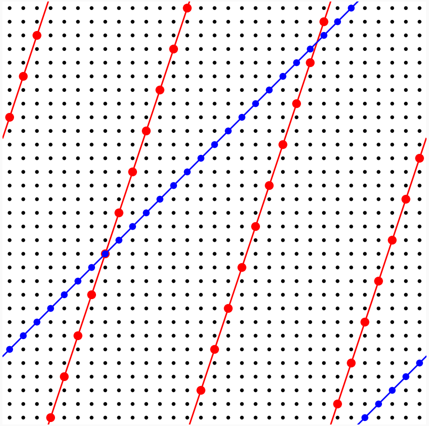
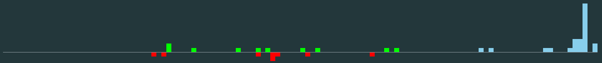


Figure 4: Lines corresponding to $y = 3x + 22$ (red) and $y = 1x + 5$ (blue).

F: Faulty Connection

Problem author: Mike de Vries



Example: One message may correspond to the red line, and another to blue. Receiving the points $(2, 28)$ and $(16, 8)$, the red line can be reconstructed using modular arithmetic.
The red and blue line ($y = 1x + 5$) intersect in exactly one integer point: $(7, 12)$.

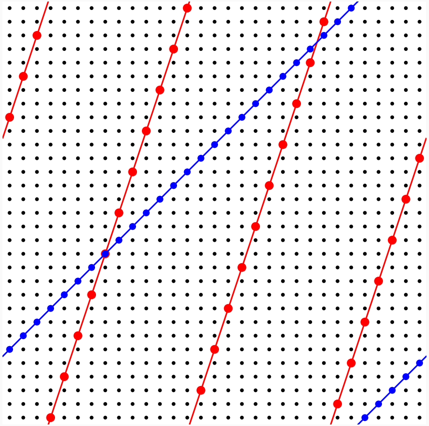
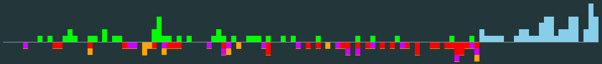


Figure 4: Lines corresponding to $y = 3x + 22$ (red) and $y = 1x + 5$ (blue).

Statistics: 42 submissions, 10 accepted, 24 unknown

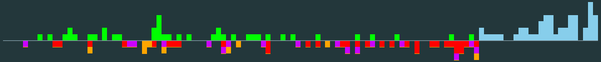
G: Garbage In, Garbage Out

Problem author: Jeroen Op de Beek



Problem: Determine whether a text is human-made or LLM-generated.

Problem author: Jeroen Op de Beek

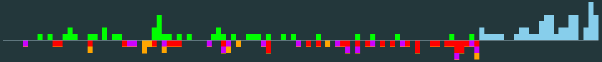


Problem: Determine whether a text is human-made or LLM-generated.

Human: The text is some concatenation of words from a list.

G: Garbage In, Garbage Out

Problem author: Jeroen Op de Beek



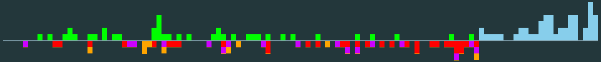
Problem: Determine whether a text is human-made or LLM-generated.

Human: The text is some concatenation of words from a list.

LLM: The text is randomly generated.

G: Garbage In, Garbage Out

Problem author: Jeroen Op de Beek



Problem: Determine whether a text is human-made or LLM-generated.

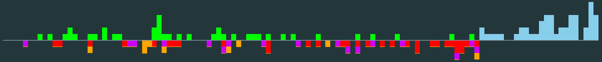
Human: The text is some concatenation of words from a list.

LLM: The text is randomly generated.

DP Solution: The text is human-made if the first 6, 7, 8, 9, or 10 letters occur in the word list, and the remainder of the text is also human-made, recursively.

G: Garbage In, Garbage Out

Problem author: Jeroen Op de Beek



Problem: Determine whether a text is human-made or LLM-generated.

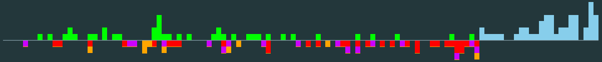
Human: The text is some concatenation of words from a list.

LLM: The text is randomly generated.

DP Solution: The text is human-made if the first 6, 7, 8, 9, or 10 letters occur in the word list, and the remainder of the text is also human-made, recursively.

Better: Only check if any word is a prefix: fails with probability at most $5000/26^6 < 0.002\%$ per case.

Problem author: Jeroen Op de Beek

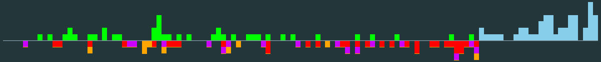


Problem: Determine whether a text is human-made or LLM-generated.

Observation: In a random text, nearly all length-6 substrings (6-mers) are different, since $26^6 \approx 300\,000\,000 \gg 300\,000$.

G: Garbage In, Garbage Out

Problem author: Jeroen Op de Beek



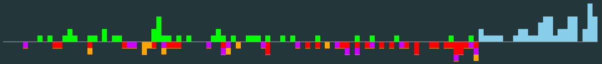
Problem: Determine whether a text is human-made or LLM-generated.

Observation: In a random text, nearly all length-6 substrings (6-mers) are different, since $26^6 \approx 300\,000\,000 \gg 300\,000$.

Solution: The text is LLM-generated if all 6-mers occur at most twice, and human otherwise.

G: Garbage In, Garbage Out

Problem author: Jeroen Op de Beek



Problem: Determine whether a text is human-made or LLM-generated.

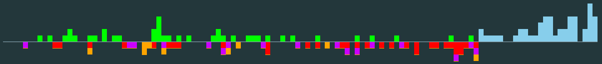
Observation: In a random text, nearly all length-6 substrings (6-mers) are different, since $26^6 \approx 300\,000\,000 \gg 300\,000$.

Solution: The text is LLM-generated if all 6-mers occur at most twice, and human otherwise.

Alternative: Count the number of distinct 6-mers instead.

G: Garbage In, Garbage Out

Problem author: Jeroen Op de Beek



Problem: Determine whether a text is human-made or LLM-generated.

Observation: In a random text, nearly all length-6 substrings (6-mers) are different, since $26^6 \approx 300\,000\,000 \gg 300\,000$.

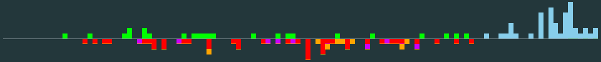
Solution: The text is LLM-generated if all 6-mers occur at most twice, and human otherwise.

Alternative: Count the number of distinct 6-mers instead.

Statistics: 146 submissions, 39 accepted, 49 unknown

H: Homesick

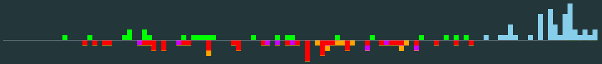
Problem author: Jonas van der Schaaf



Problem: In an undirected unweighted simple graph, find a shortest walk from v_1 back to v_1 using at least one edge and without a *digon* (i.e., without a subwalk of the form u, v, u).

H: Homesick

Problem author: Jonas van der Schaaf

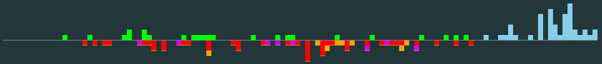


Problem: In an undirected unweighted simple graph, find a shortest walk from v_1 back to v_1 using at least one edge and without a *digon* (i.e., without a subwalk of the form u, v, u).

Idea: Assume there is a valid solution, and look at a node w on the path that is *furthest* from v_1 (i.e. the node with the greatest depth from v_1).

H: Homesick

Problem author: Jonas van der Schaaf



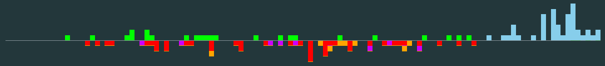
Problem: In an undirected unweighted simple graph, find a shortest walk from v_1 back to v_1 using at least one edge and without a *digon* (i.e., without a subwalk of the form u, v, u).

Idea: Assume there is a valid solution, and look at a node w on the path that is *furthest* from v_1 (i.e. the node with the greatest depth from v_1).

Observation: From this node, two paths to v_1 must exist that start out towards different neighbours of w.

H: Homesick

Problem author: Jonas van der Schaaf



Problem: In an undirected unweighted simple graph, find a shortest walk from v_1 back to v_1 using at least one edge and without a *digon* (i.e., without a subwalk of the form u, v, u).

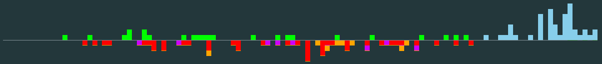
Idea: Assume there is a valid solution, and look at a node w on the path that is *furthest* from v_1 (i.e. the node with the greatest depth from v_1).

Observation: From this node, two paths to v_1 must exist that start out towards different neighbours of w.

Observation 2: In the optimal solution, the sum of the lengths of these paths must be minimal.

H: Homesick

Problem author: Jonas van der Schaaf



Problem: In an undirected unweighted simple graph, find a shortest walk from v_1 back to v_1 using at least one edge and without a *digon* (i.e., without a subwalk of the form u, v, u).

Idea: Assume there is a valid solution, and look at a node w on the path that is *furthest* from v_1 (i.e. the node with the greatest depth from v_1).

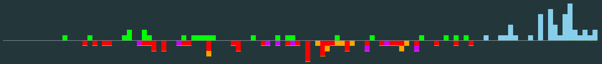
Observation: From this node, two paths to v_1 must exist that start out towards different neighbours of w.

Observation 2: In the optimal solution, the sum of the lengths of these paths must be minimal.

Observation 3: The length of such a path is 1 plus the depth of the neighbour of w the path goes through.

H: Homesick

Problem author: Jonas van der Schaaf



Problem: In an undirected unweighted simple graph, find a shortest walk from v_1 back to v_1 using at least one edge and without a *digon* (i.e., without a subwalk of the form u, v, u).

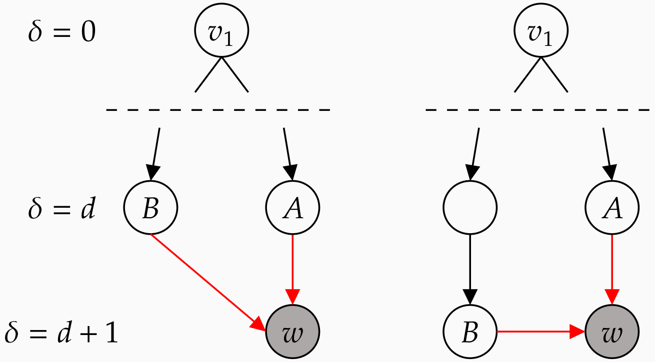
Idea: Assume there is a valid solution, and look at a node w on the path that is *furthest* from v_1 (i.e. the node with the greatest depth from v_1).

Observation: From this node, two paths to v_1 must exist that start out towards different neighbours of w.

Observation 2: In the optimal solution, the sum of the lengths of these paths must be minimal.

Observation 3: The length of such a path is 1 plus the depth of the neighbour of w the path goes through.

Observation 4: These paths go to the two neighbours of w that are closest to v_1.

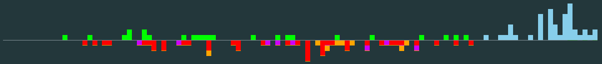


$$l = \delta_A + \delta_B + 2$$

Figure 5: The two cases for optimal solutions given a furthest node.

H: Homesick

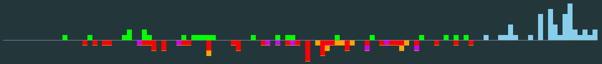
Problem author: Jonas van der Schaaf



Solution: Run a BFS and check the optimal path length for all possible deepest nodes:

H: Homesick

Problem author: Jonas van der Schaaf

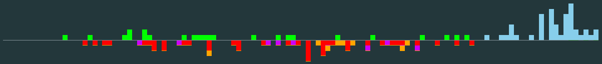


Solution: Run a BFS and check the optimal path length for all possible deepest nodes:

Mark the depth of each node.

H: Homesick

Problem author: Jonas van der Schaaf



Solution: Run a BFS and check the optimal path length for all possible deepest nodes:

Mark the depth of each node.

When encounting a node for the second time, the path length is the depth of the node plus the path length of the second path to this node.

H: Homesick

Problem author: Jonas van der Schaaf

Solution: Run a BFS and check the optimal path length for all possible deepest nodes:

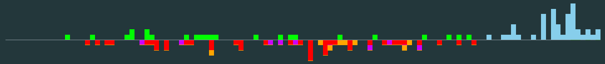
Mark the depth of each node.

When encounting a node for the second time, the path length is the depth of the node plus the path length of the second path to this node.

Pick the optimal deepest node (if one exists) and reconstruct the path.

H: Homesick

Problem author: Jonas van der Schaaf



Solution: Run a BFS and check the optimal path length for all possible deepest nodes:

Mark the depth of each node.

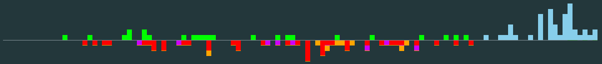
When encounting a node for the second time, the path length is the depth of the node plus the path length of the second path to this node.

Pick the optimal deepest node (if one exists) and reconstruct the path.

Running time: $\mathcal{O}(n + m)$

H: Homesick

Problem author: Jonas van der Schaaf



Solution: Run a BFS and check the optimal path length for all possible deepest nodes:

Mark the depth of each node.

When encounting a node for the second time, the path length is the depth of the node plus the path length of the second path to this node.

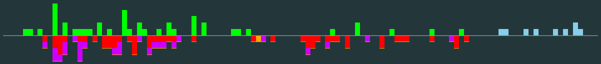
Pick the optimal deepest node (if one exists) and reconstruct the path.

Running time: $\mathcal{O}(n + m)$

Statistics: 123 submissions, 24 accepted, 43 unknown

I: Intermill Logistics

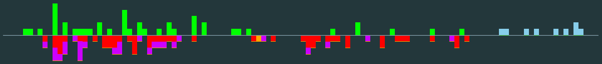
Problem author: Maarten Sijm



Problem: For $i \in \{1, \ldots, n\}$, the ith windmill can process p_i kg wheat in 1 hour; it takes t_i hours to travel to (and from). How fast can you process w kg wheat and get it back?

I: Intermill Logistics

Problem author: Maarten Sijm

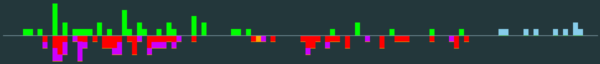


Problem: For $i \in \{1, \ldots, n\}$, the ith windmill can process p_i kg wheat in 1 hour; it takes t_i hours to travel to (and from). How fast can you process w kg wheat and get it back?

Easier problem: Given time r, can the wheat be processed in time r or less?

I: Intermill Logistics

Problem author: Maarten Sijm



Problem: For $i \in \{1, \ldots, n\}$, the ith windmill can process p_i kg wheat in 1 hour; it takes t_i hours to travel to (and from). How fast can you process w kg wheat and get it back?

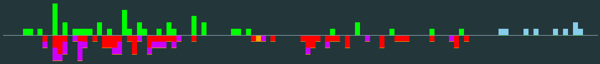
Easier problem: Given time r, can the wheat be processed in time r or less?

Observation: If you can reach ith windmill in time r (and get back), *i.e.*, if $r \geq 2t_i$, you can process $p_i(r - 2t_i)$ kg wheat there. In total,

$$\sum_{i=1}^{n} \max\big(0, p_i(r - 2t_i)\big) \; .$$

I: Intermill Logistics

Problem author: Maarten Sijm



Problem: For $i \in \{1, \ldots, n\}$, the ith windmill can process p_i kg wheat in 1 hour; it takes t_i hours to travel to (and from). How fast can you process w kg wheat and get it back?

Easier problem: Given time r, can the wheat be processed in time r or less?

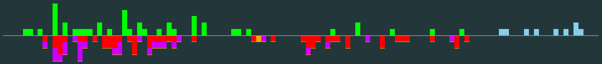
Observation: If you can reach ith windmill in time r (and get back), *i.e.*, if $r \geq 2t_i$, you can process $p_i(r - 2t_i)$ kg wheat there. In total,

$$\sum_{i=1}^{n} \max\big(0, p_i(r - 2t_i)\big) .$$

Solution: We can *binary-search for the answer*. Travel times are positive integers, so $l = 2$ is a lower bound. For an upper bound, processing *all* wheat on the first machine takes $h = 2t_1 + w/p_1 \leq 2 \cdot 10^9$. Can now binary search for the correct value of r with $l \leq r \leq h$.

I: Intermill Logistics

Problem author: Maarten Sijm



Problem: For $i \in \{1, \dots, n\}$, the ith windmill can process p_i kg wheat in 1 hour; it takes t_i hours to travel to (and from). How fast can you process w kg wheat and get it back?

Easier problem: Given time r, can the wheat be processed in time r or less?

Observation: If you can reach ith windmill in time r (and get back), *i.e.*, if $r \geq 2t_i$, you can process $p_i(r - 2t_i)$ kg wheat there. In total,

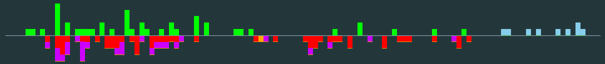
$$\sum_{i=1}^{n} \max\big(0, p_i(r - 2t_i)\big) \,.$$

Solution: We can *binary-search for the answer*. Travel times are positive integers, so $l = 2$ is a lower bound. For an upper bound, processing *all* wheat on the first machine takes $h = 2t_1 + w/p_1 \leq 2 \cdot 10^9$. Can now binary search for the correct value of r with $l \leq r \leq h$.

Running time: $\mathcal{O}(n \log h)$.

I: Intermill Logistics

Problem author: Maarten Sijm



Problem: For $i \in \{1, \dots, n\}$, the ith windmill can process p_i kg wheat in 1 hour; it takes t_i hours to travel to (and from). How fast can you process w kg wheat and get it back?

Easier problem: Given time r, can the wheat be processed in time r or less?

Observation: If you can reach ith windmill in time r (and get back), *i.e.*, if $r \geq 2t_i$, you can process $p_i(r - 2t_i)$ kg wheat there. In total,

$$\sum_{i=1}^{n} \max\bigl(0, p_i(r - 2t_i)\bigr) .$$

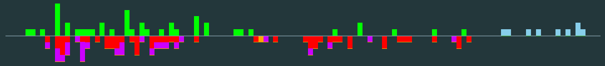
Solution: We can *binary-search for the answer*. Travel times are positive integers, so $l = 2$ is a lower bound. For an upper bound, processing *all* wheat on the first machine takes $h = 2t_1 + w/p_1 \leq 2 \cdot 10^9$. Can now binary search for the correct value of r with $l \leq r \leq h$.

Running time: $\mathcal{O}(n \log h)$.

Statistics: 130 submissions, 43 accepted, 9 unknown

I: Intermill Logistics

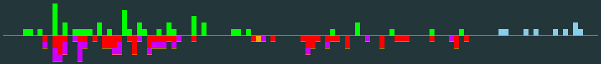
Problem author: Maarten Sijm



Problem: For $i \in \{1, \ldots, n\}$, the ith windmill can process p_i kg wheat in 1 hour; it takes t_i hours to travel to (and from). How fast can you process w kg wheat and get it back?

I: Intermill Logistics

Problem author: Maarten Sijm

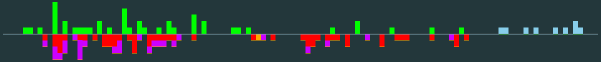


Problem: For $i \in \{1, \ldots, n\}$, the ith windmill can process p_i kg wheat in 1 hour; it takes t_i hours to travel to (and from). How fast can you process w kg wheat and get it back?

Greedy solution: Sort the mills such that $t_1 \leq \cdots \leq t_n$.

I: Intermill Logistics

Problem author: Maarten Sijm



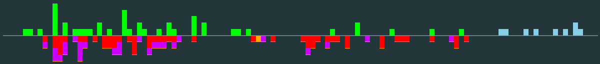
Problem: For $i \in \{1, \ldots, n\}$, the ith windmill can process p_i kg wheat in 1 hour; it takes t_i hours to travel to (and from). How fast can you process w kg wheat and get it back?

Greedy solution: Sort the mills such that $t_1 \leq \cdots \leq t_n$.

Claim: There is an optimal solution using exactly mills $1, \ldots, i$ for some i, with the mills grinding for nonzero time, and either grinding or transporting during the entire makespan.

I: Intermill Logistics

Problem author: Maarten Sijm



Problem: For $i \in \{1, \ldots, n\}$, the ith windmill can process p_i kg wheat in 1 hour; it takes t_i hours to travel to (and from). How fast can you process w kg wheat and get it back?

Greedy solution: Sort the mills such that $t_1 \leq \cdots \leq t_n$.

Claim: There is an optimal solution using exactly mills $1, \ldots, i$ for some i, with the mills grinding for nonzero time, and either grinding or transporting during the entire makespan.

Implementation: Getting to/from mill i takes time t_i, so during that time the other mills can grind

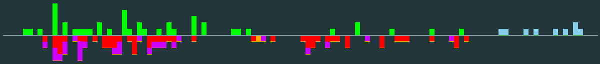
$$w' = \sum_{j=1}^{i-1} p_j \cdot (2t_i - 2t_j) \text{ kg wheat.}$$

To grind the remaining wheat (if any), all i machines can work, so the makespan is

$$2t_i + (w - w')/(p_1 + \cdots + p_i)\,.$$

I: Intermill Logistics

Problem author: Maarten Sijm



Problem: For $i \in \{1, \ldots, n\}$, the ith windmill can process p_i kg wheat in 1 hour; it takes t_i hours to travel to (and from). How fast can you process w kg wheat and get it back?

Greedy solution: Sort the mills such that $t_1 \leq \cdots \leq t_n$.

Claim: There is an optimal solution using exactly mills $1, \ldots, i$ for some i, with the mills grinding for nonzero time, and either grinding or transporting during the entire makespan.

Implementation: Getting to/from mill i takes time t_i, so during that time the other mills can grind

$$w' = \sum_{j=1}^{i-1} p_j \cdot (2t_i - 2t_j) \text{ kg wheat.}$$

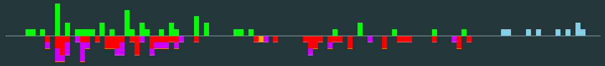
To grind the remaining wheat (if any), all i machines can work, so the makespan is

$$2t_i + (w - w')/(p_1 + \cdots + p_i)\,.$$

Running time: Naïve implementation requires $\mathcal{O}(n^2)$. The sums can be computed cumulatively to achieve time $\mathcal{O}(n \log n)$ for $n \geq 2$.

I: Intermill Logistics

Problem author: Maarten Sijm



Problem: For $i \in \{1, \ldots, n\}$, the ith windmill can process p_i kg wheat in 1 hour; it takes t_i hours to travel to (and from). How fast can you process w kg wheat and get it back?

Greedy solution: Sort the mills such that $t_1 \leq \cdots \leq t_n$.

Claim: There is an optimal solution using exactly mills $1, \ldots, i$ for some i, with the mills grinding for nonzero time, and either grinding or transporting during the entire makespan.

Implementation: Getting to/from mill i takes time t_i, so during that time the other mills can grind

$$w' = \sum_{j=1}^{i-1} p_j \cdot (2t_i - 2t_j) \text{ kg wheat.}$$

To grind the remaining wheat (if any), all i machines can work, so the makespan is

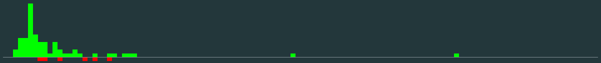
$$2t_i + (w - w')/(p_1 + \cdots + p_i)\,.$$

Running time: Naïve implementation requires $\mathcal{O}(n^2)$. The sums can be computed cumulatively to achieve time $\mathcal{O}(n \log n)$ for $n \geq 2$.

Statistics: 130 submissions, 43 accepted, 9 unknown

J: Jacobi Numbers

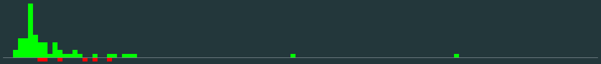
Problem author: Reinier Schmiermann



Problem: Write n as a sum of cubes.

J: Jacobi Numbers

Problem author: Reinier Schmiermann

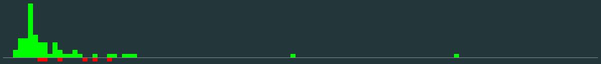


Problem: Write n as a sum of cubes.

Observation: Note that $1 = 1^3$.

J: Jacobi Numbers

Problem author: Reinier Schmiermann



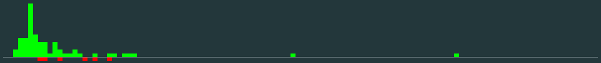
Problem: Write n as a sum of cubes.

Observation: Note that $1 = 1^3$.

Solution: With the given output bounds, it is possible to simply print 1, repeated n times.

J: Jacobi Numbers

Problem author: Reinier Schmiermann



Problem: Write n as a sum of cubes.

Observation: Note that $1 = 1^3$.

Solution: With the given output bounds, it is possible to simply print 1, repeated n times.

Running time: $\mathcal{O}(n)$.

J: Jacobi Numbers

Problem author: Reinier Schmiermann

Problem: Write n as a sum of cubes.

Observation: Note that $1 = 1^3$.

Solution: With the given output bounds, it is possible to simply print 1, repeated n times.

Running time: $\mathcal{O}(n)$.

Decompose: Decompositions of the numbers between 1 and 9241 into sums of cubes using as few terms as possible:

singular cubes,	like $27 = 3^3$	20
sums of two cubes,	like $9241 = 56^3 + (-55)^3$	453
sums of three cubes,	like $9240 = 56^3 + (-55)^3 + (-1)^3$	5761
sums of four cubes,	like $9239 = 32^3 + (-25)^3 + (-22)^3 + 14^3$	3007

Note that the terms can be much larger than the sum, *e.g.*,

$$311 = -9529^3 - 8185^3 + 8228^3 + 9497^3 .$$

Careful C++ implementation computes this within time bounds.

J: Jacobi Numbers

Problem author: Reinier Schmiermann

Problem: Write n as a sum of cubes.

Observation: Note that $1 = 1^3$.

Solution: With the given output bounds, it is possible to simply print 1, repeated n times.

Running time: $\mathcal{O}(n)$.

Decompose: Decompositions of the numbers between 1 and 9241 into sums of cubes using as few terms as possible:

singular cubes,	like $27 = 3^3$	20
sums of two cubes,	like $9241 = 56^3 + (-55)^3$	453
sums of three cubes,	like $9240 = 56^3 + (-55)^3 + (-1)^3$	5761
sums of four cubes,	like $9239 = 32^3 + (-25)^3 + (-22)^3 + 14^3$	3007

Note that the terms can be much larger than the sum, *e.g.*,

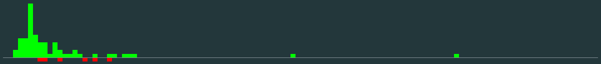
$$311 = -9529^3 - 8185^3 + 8228^3 + 9497^3 \,.$$

Careful C++ implementation computes this within time bounds.

Constant time: Determine (optimal) decomposition off-line for all possible inputs; the submission then looks up input in table with 10 000 entries.

J: Jacobi Numbers

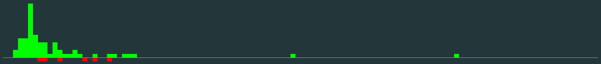
Problem author: Reinier Schmiermann



Known: Every integer is the sum of five cubes.

J: Jacobi Numbers

Problem author: Reinier Schmiermann



Known: Every integer is the sum of five cubes.

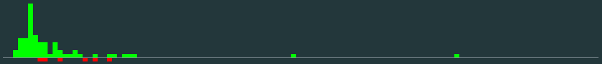
Proof: When $n = 6r$ for integer r we have

$$n = (r+1)^3 + (r-1)^3 + 2(-r)^3 .$$

Possibly adding $(-1)^3$ on the right hand side solves the problem for $6r, 6r-1, 6r+1$. The remaining cases $(6r+2, 6r+3, 6r+4)$ are handled similarly. There are many ways of doing this.

J: Jacobi Numbers

Problem author: Reinier Schmiermann



Known: Every integer is the sum of five cubes.

Proof: When $n = 6r$ for integer r we have

$$n = (r+1)^3 + (r-1)^3 + 2(-r)^3 \,.$$

Possibly adding $(-1)^3$ on the right hand side solves the problem for $6r, 6r-1, 6r+1$. The remaining cases $(6r+2, 6r+3, 6r+4)$ are handled similarly. There are many ways of doing this.

Open problem: Can every integer be decomposed into four cubes?

J: Jacobi Numbers

Problem author: Reinier Schmiermann

Known: Every integer is the sum of five cubes.

Proof: When $n = 6r$ for integer r we have

$$n = (r+1)^3 + (r-1)^3 + 2(-r)^3 .$$

Possibly adding $(-1)^3$ on the right hand side solves the problem for $6r, 6r-1, 6r+1$. The remaining cases $(6r+2, 6r+3, 6r+4)$ are handled similarly. There are many ways of doing this.

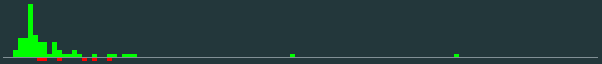
Open problem: Can every integer be decomposed into four cubes?

Open problem: Can every integer $n \neq 4, 5 \mod 9$ be decomposed into three cubes?

$$33 = 8866128975287528^3 + (-8778405442862239)^3 + (-2736111468807040)^3$$

J: Jacobi Numbers

Problem author: Reinier Schmiermann



Known: Every integer is the sum of five cubes.

Proof: When $n = 6r$ for integer r we have

$$n = (r+1)^3 + (r-1)^3 + 2(-r)^3 .$$

Possibly adding $(-1)^3$ on the right hand side solves the problem for $6r, 6r-1, 6r+1$. The remaining cases $(6r+2, 6r+3, 6r+4)$ are handled similarly. There are many ways of doing this.

Open problem: Can every integer be decomposed into four cubes?

Open problem: Can every integer $n \neq 4, 5 \mod 9$ be decomposed into three cubes?

$$33 = 8866128975287528^3 + (-8778405442862239)^3 + (-2736111468807040)^3$$

Statistics: 66 submissions, 60 accepted

K: Knowing the Clock

Problem author: Wietze Koops

Problem: Given the angles of the hands of a watch, check whether they correspond to a real time.

K: Knowing the Clock

Problem author: Wietze Koops

Problem: Given the angles of the hands of a watch, check whether they correspond to a real time.

Observation: For every full circle of the hour hand, the minute hand completes 12 circles.

K: Knowing the Clock

Problem author: Wietze Koops

Problem: Given the angles of the hands of a watch, check whether they correspond to a real time.

Observation: For every full circle of the hour hand, the minute hand completes 12 circles.

Solution: Output “`yes`” if $m = h * 12 \mod 360$, and “`no`” otherwise.

K: Knowing the Clock

Problem author: Wietze Koops

Problem: Given the angles of the hands of a watch, check whether they correspond to a real time.

Observation: For every full circle of the hour hand, the minute hand completes 12 circles.

Solution: Output "`yes`" if $m = h * 12 \mod 360$, and "`no`" otherwise.

Running time: $\mathcal{O}(1)$.

K: Knowing the Clock

Problem author: Wietze Koops

Problem: Given the angles of the hands of a watch, check whether they correspond to a real time.

Observation: For every full circle of the hour hand, the minute hand completes 12 circles.

Solution: Output "`yes`" if $m = h * 12 \mod 360$, and "`no`" otherwise.

Running time: $\mathcal{O}(1)$.

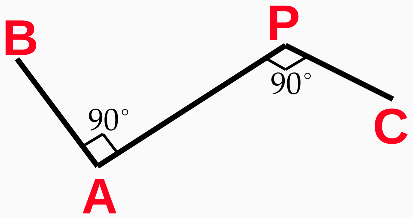
Statistics: 73 submissions, 60 accepted, 1 unknown

L: Linguistic Labyrinth

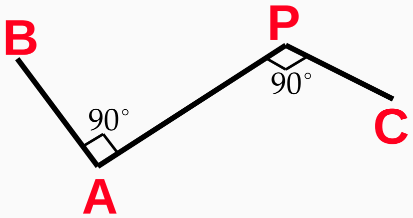
Problem author: Jeroen Op de Beek

Problem: Count number of quadruples $BAPC$ such that $\angle BAP = 90^\circ$ and $\angle APC = 90^\circ$.

Problem: Count number of quadruples $BAPC$ such that $\angle BAP = 90^\circ$ and $\angle APC = 90^\circ$.

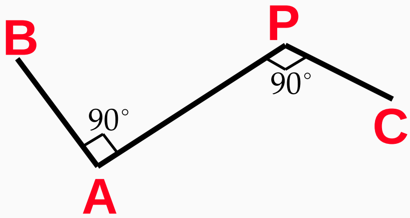


Problem: Count number of quadruples $BAPC$ such that $\angle BAP = 90^\circ$ and $\angle APC = 90^\circ$.



Naive solution: Loop over all quadruples, and check whether the angles are correct.

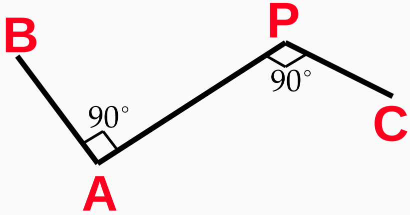
Problem: Count number of quadruples $BAPC$ such that $\angle BAP = 90^\circ$ and $\angle APC = 90^\circ$.



Naive solution: Loop over all quadruples, and check whether the angles are correct.

Running time: However, there are n^3 points, so this is $\mathcal{O}(n^{12})$, too slow!

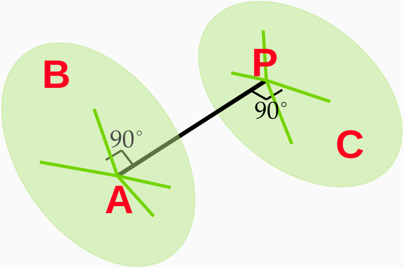
Problem: Count number of quadruples $BAPC$ such that $\angle BAP = 90^\circ$ and $\angle APC = 90^\circ$.



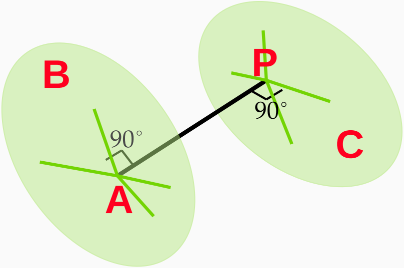
Naive solution: Loop over all quadruples, and check whether the angles are correct.

Running time: However, there are n^3 points, so this is $\mathcal{O}(n^{12})$, too slow!

Better solution: Fix A and P. Now the choice of B and C are independent.

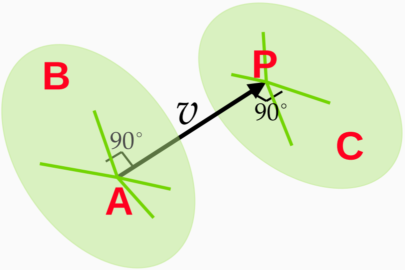


Better solution: Loop over all AP pairs, count the number of possible B's and C's, and multiply these counts.

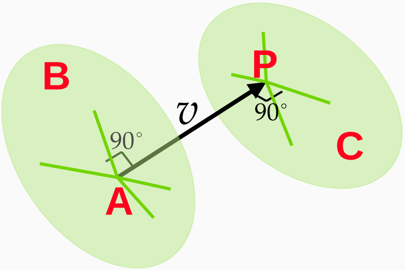


Better solution: Loop over all AP pairs, count the number of possible B's and C's, and multiply these counts.

Running time: Counting B's and C's takes $\mathcal{O}(n^3)$ per AP pair, so the runtime is $\mathcal{O}(n^9)$ in total, still too slow.

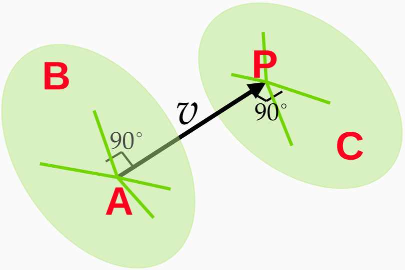


Best solution: Let v be the vector $P - A$. Then a point B is good if and only if $v \cdot B = v \cdot A$, and likewise C is good if and only if $v \cdot C = v \cdot P$.



Best solution: Let v be the vector $P - A$. Then a point B is good if and only if $v \cdot B = v \cdot A$, and likewise C is good if and only if $v \cdot C = v \cdot P$.

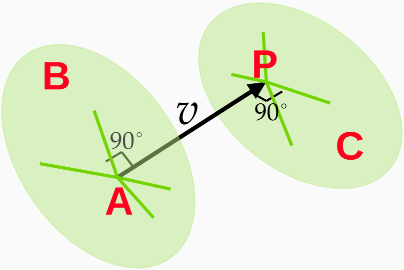
Best solution: For each possible vector v, precompute the number of B's and C's that have a certain inner product. These counts can be stored in an array of size $\mathcal{O}(n^5)$.



Best solution: Let v be the vector $P - A$. Then a point B is good if and only if $v \cdot B = v \cdot A$, and likewise C is good if and only if $v \cdot C = v \cdot P$.

Best solution: For each possible vector v, precompute the number of B's and C's that have a certain inner product. These counts can be stored in an array of size $\mathcal{O}(n^5)$.

Running time: There are $\mathcal{O}(n^3)$ different v's, so precomputation is $\mathcal{O}(n^6)$. Final complexity: $\mathcal{O}(n^6)$.

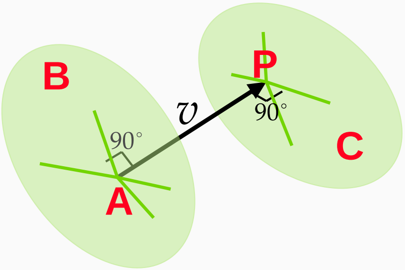


Best solution: Let v be the vector $P - A$. Then a point B is good if and only if $v \cdot B = v \cdot A$, and likewise C is good if and only if $v \cdot C = v \cdot P$.

Best solution: For each possible vector v, precompute the number of B's and C's that have a certain inner product. These counts can be stored in an array of size $\mathcal{O}(n^5)$.

Running time: There are $\mathcal{O}(n^3)$ different v's, so precomputation is $\mathcal{O}(n^6)$. Final complexity: $\mathcal{O}(n^6)$.

Pitfall: Using (unordered) maps might result in TLE.



Best solution: Let v be the vector $P - A$. Then a point B is good if and only if $v \cdot B = v \cdot A$, and likewise C is good if and only if $v \cdot C = v \cdot P$.

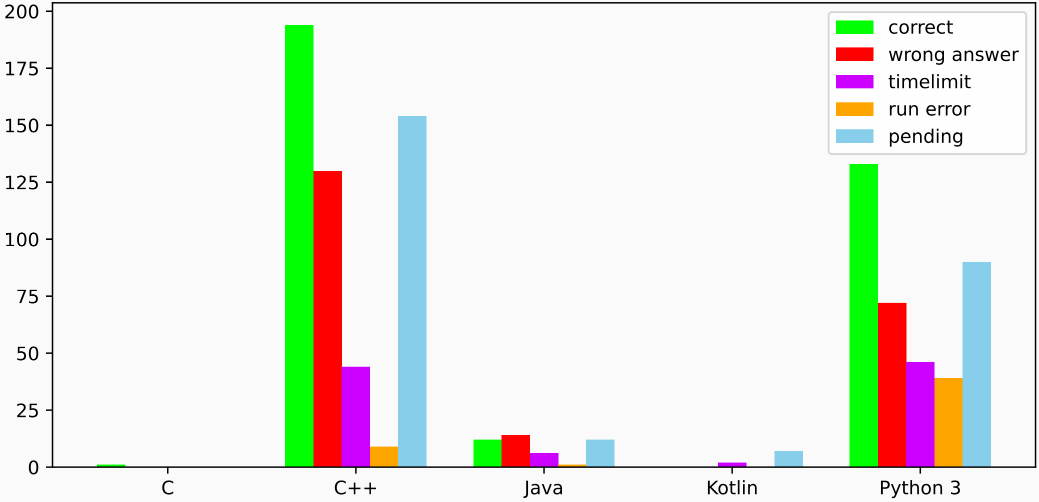
Best solution: For each possible vector v, precompute the number of B's and C's that have a certain inner product. These counts can be stored in an array of size $\mathcal{O}(n^5)$.

Running time: There are $\mathcal{O}(n^3)$ different v's, so precomputation is $\mathcal{O}(n^6)$. Final complexity: $\mathcal{O}(n^6)$.

Pitfall: Using (unordered) maps might result in TLE.

Statistics: 32 submissions, 0 accepted, 21 unknown

Language stats



Random facts

Jury work

- 945 commits, of which 525 for the main contest (last year: 1138/630)

Random facts

Jury work

- 945 commits, of which 525 for the main contest (last year: 1138/630)
- 1035 secret test cases (last year: 1192) ($86\frac{1}{4}$ per problem!)

Random facts

Jury work

- 945 commits, of which 525 for the main contest (last year: 1138/630)
- 1035 secret test cases (last year: 1192) ($86\frac{1}{4}$ per problem!)
- 293 jury + proofreader solutions (last year: 273)

Random facts

Jury work

- 945 commits, of which 525 for the main contest (last year: 1138/630)
- 1035 secret test cases (last year: 1192) ($86\frac{1}{4}$ per problem!)
- 293 jury + proofreader solutions (last year: 273)
- The minimum[1] number of lines the jury needed to solve all problems is

$$1+3+7+7+9+2+1+8+3+1+1+9=52$$

  On average, $4\frac{1}{3}$ lines per problem (8.8 in BAPC 2024, 4 in preliminaries 2024)

[1] With PEP 8 compliant code golfing

Thanks to:

The proofreaders

Arnoud van der Leer

Jaap Eldering

Jeroen Bransen Java Hero 🎈

Pavel Kunyavskiy

Tobias Roehr 🎈

Wendy Yi 🎈

Special thanks to:

Freek Henstra, for Accidental Arithmetic

The jury

Ivan Fefer

Jeroen Op de Beek

Jonas van der Schaaf

Lammert Westerdijk

Leon van der Waal

Maarten Sijm

Marijn Adriaanse

Mike de Vries

Ragnar Groot Koerkamp

Reinier Schmiermann

Thore Husfeldt

Wietze Koops

Want to join the jury? Submit to the Call for Problems of BAPC 2026 at:

`https://jury.bapc.eu/`