Delft Algorithm Programming Contest (DAPC) 2025

Solutions presentation

The BAPC 2025 Jury

October 4, 2025

K: Koehandel

Problem author: Mike de Vries

Problem: Your opponent bid c coins, and you have n coins. You get your opponent's cow when bidding more than c coins, and lose your cow when bidding less than c coins.
What should you bid to maximize the number of cows you end up with and secondarily your number of coins?

K: Koehandel

Problem author: Mike de Vries

Problem: Your opponent bid c coins, and you have n coins. You get your opponent's cow when bidding more than c coins, and lose your cow when bidding less than c coins.
What should you bid to maximize the number of cows you end up with and secondarily your number of coins?

Case 1: $n > c$. You can obtain your opponent's cow by bidding $c + 1$, which is optimal.

K: Koehandel

Problem author: Mike de Vries

Problem: Your opponent bid c coins, and you have n coins. You get your opponent's cow when bidding more than c coins, and lose your cow when bidding less than c coins.
What should you bid to maximize the number of cows you end up with and secondarily your number of coins?

Case 1: $n > c$. You can obtain your opponent's cow by bidding $c + 1$, which is optimal.

Case 2: $n = c$. You can keep your cow by bidding c, which is optimal.

K: Koehandel

Problem author: Mike de Vries

Problem: Your opponent bid c coins, and you have n coins. You get your opponent's cow when bidding more than c coins, and lose your cow when bidding less than c coins.

What should you bid to maximize the number of cows you end up with and secondarily your number of coins?

Case 1: $n > c$. You can obtain your opponent's cow by bidding $c + 1$, which is optimal.

Case 2: $n = c$. You can keep your cow by bidding c, which is optimal.

Case 3: $n < c$. You will always lose your cow, so bidding 0 is optimal.

K: Koehandel

Problem author: Mike de Vries

Problem: Your opponent bid c coins, and you have n coins. You get your opponent's cow when bidding more than c coins, and lose your cow when bidding less than c coins.
What should you bid to maximize the number of cows you end up with and secondarily your number of coins?

Case 1: $n > c$. You can obtain your opponent's cow by bidding $c + 1$, which is optimal.

Case 2: $n = c$. You can keep your cow by bidding c, which is optimal.

Case 3: $n < c$. You will always lose your cow, so bidding 0 is optimal.

Complexity: $O(1)$.

K: Koehandel

Problem author: Mike de Vries

Problem: Your opponent bid c coins, and you have n coins. You get your opponent's cow when bidding more than c coins, and lose your cow when bidding less than c coins.
What should you bid to maximize the number of cows you end up with and secondarily your number of coins?

Case 1: $n > c$. You can obtain your opponent's cow by bidding $c + 1$, which is optimal.

Case 2: $n = c$. You can keep your cow by bidding c, which is optimal.

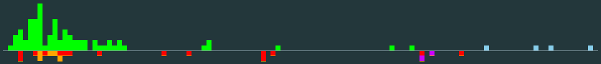
Case 3: $n < c$. You will always lose your cow, so bidding 0 is optimal.

Complexity: $O(1)$.

Statistics: 83 submissions, 75 accepted

D: Dralinpome

Problem author: Mike de Vries

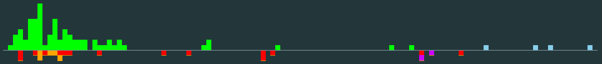


Problem: Determine whether a word is a *dralinpome*.

A word is a dralinpome if there exists a permutation of its letters that is a palindrome.

D: Dralinpome

Problem author: Mike de Vries



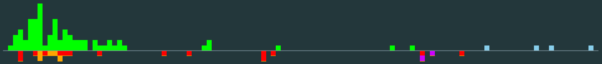
Problem: Determine whether a word is a *dralinpome*.

A word is a dralinpome if there exists a permutation of its letters that is a palindrome.

Observation: Palindromes are mirrored words, so each letter must occur an even number of times.

D: Dralinpome

Problem author: Mike de Vries



Problem: Determine whether a word is a *dralinpome*.

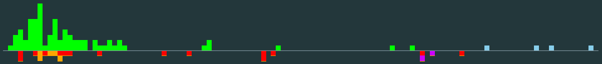
A word is a dralinpome if there exists a permutation of its letters that is a palindrome.

Observation: Palindromes are mirrored words, so each letter must occur an even number of times.

Exception: If the length of the word is odd, the middle letter can occur an odd number of times.

D: Dralinpome

Problem author: Mike de Vries



Problem: Determine whether a word is a *dralinpome*.

A word is a dralinpome if there exists a permutation of its letters that is a palindrome.

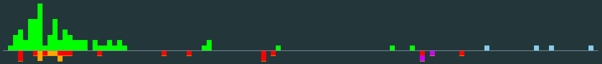
Observation: Palindromes are mirrored words, so each letter must occur an even number of times.

Exception: If the length of the word is odd, the middle letter can occur an odd number of times.

Solution: Count the number of occurrences of each letter and check whether they are all even, with at most one odd one out.

D: Dralinpome

Problem author: Mike de Vries



Problem: Determine whether a word is a *dralinpome*.
A word is a dralinpome if there exists a permutation of its letters that is a palindrome.

Observation: Palindromes are mirrored words, so each letter must occur an even number of times.

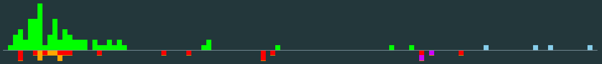
Exception: If the length of the word is odd, the middle letter can occur an odd number of times.

Solution: Count the number of occurrences of each letter and check whether they are all even, with at most one odd one out.

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

D: Dralinpome

Problem author: Mike de Vries



Problem: Determine whether a word is a *dralinpome*.
A word is a dralinpome if there exists a permutation of its letters that is a palindrome.

Observation: Palindromes are mirrored words, so each letter must occur an even number of times.

Exception: If the length of the word is odd, the middle letter can occur an odd number of times.

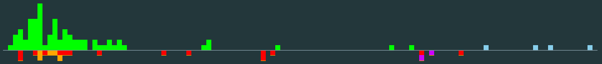
Solution: Count the number of occurrences of each letter and check whether they are all even, with at most one odd one out.

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

Easter egg: Can you find all the dralinpomes in the problem statement?

D: Dralinpome

Problem author: Mike de Vries



Problem: Determine whether a word is a *dralinpome*.
A word is a dralinpome if there exists a permutation of its letters that is a palindrome.

Observation: Palindromes are mirrored words, so each letter must occur an even number of times.

Exception: If the length of the word is odd, the middle letter can occur an odd number of times.

Solution: Count the number of occurrences of each letter and check whether they are all even, with at most one odd one out.

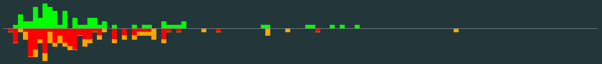
Running time: $\mathcal{O}(n)$.

Easter egg: Can you find all the dralinpomes in the problem statement?

Statistics: 98 submissions, 72 accepted, 4 unknown

B: Bottle of New Port

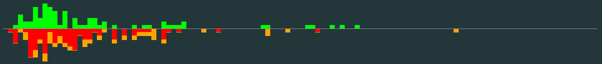
Problem author: Thore Husfeldt



Problem: Compute the amount of alcohol left in a bottle after d days.

B: Bottle of New Port

Problem author: Thore Husfeldt



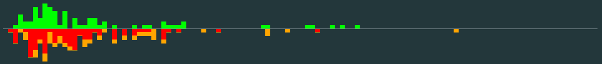
Problem: Compute the amount of alcohol left in a bottle after d days.

Solution:

- The amount of alcohol left is $x = \max(a - d \cdot \Delta_a, 0)$.

B: Bottle of New Port

Problem author: Thore Husfeldt



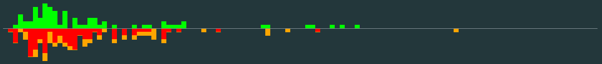
Problem: Compute the amount of alcohol left in a bottle after d days.

Solution:

- The amount of alcohol left is $x = \max(a - d \cdot \Delta_a, 0)$.
- For other liquids, it is $y = \max(o - d \cdot \Delta_o, 0)$.

B: Bottle of New Port

Problem author: Thore Husfeldt



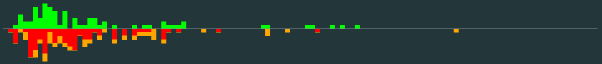
Problem: Compute the amount of alcohol left in a bottle after d days.

Solution:

- The amount of alcohol left is $x = \max(a - d \cdot \Delta_a, 0)$.
- For other liquids, it is $y = \max(o - d \cdot \Delta_o, 0)$.
- Then just compute $\frac{x}{x+y}$.

B: Bottle of New Port

Problem author: Thore Husfeldt



Problem: Compute the amount of alcohol left in a bottle after d days.

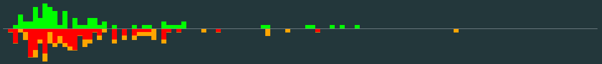
Solution:

- The amount of alcohol left is $x = \max(a - d \cdot \Delta_a, 0)$.
- For other liquids, it is $y = \max(o - d \cdot \Delta_o, 0)$.
- Then just compute $\frac{x}{x+y}$.

Running time: $\mathcal{O}(1)$, but even $\mathcal{O}(d)$ will pass.

B: Bottle of New Port

Problem author: Thore Husfeldt



Problem: Compute the amount of alcohol left in a bottle after d days.

Solution:

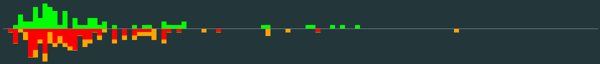
- The amount of alcohol left is $x = \max(a - d \cdot \Delta_a, 0)$.
- For other liquids, it is $y = \max(o - d \cdot \Delta_o, 0)$.
- Then just compute $\frac{x}{x+y}$.

Running time: $\mathcal{O}(1)$, but even $\mathcal{O}(d)$ will pass.

Pitfall: Values are larger than 10^9, do not use 32-bit `int`.

B: Bottle of New Port

Problem author: Thore Husfeldt



Problem: Compute the amount of alcohol left in a bottle after d days.

Solution:

- The amount of alcohol left is $x = \max(a - d \cdot \Delta_a, 0)$.
- For other liquids, it is $y = \max(o - d \cdot \Delta_o, 0)$.
- Then just compute $\frac{x}{x+y}$.

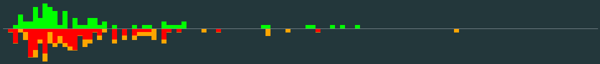
Running time: $\mathcal{O}(1)$, but even $\mathcal{O}(d)$ will pass.

Pitfall: Values are larger than 10^9, do not use 32-bit `int`.

Pitfall: Many teams forgot the max function, which fails on the fourth sample.

B: Bottle of New Port

Problem author: Thore Husfeldt



Problem: Compute the amount of alcohol left in a bottle after d days.

Solution:

- The amount of alcohol left is $x = \max(a - d \cdot \Delta_a, 0)$.
- For other liquids, it is $y = \max(o - d \cdot \Delta_o, 0)$.
- Then just compute $\frac{x}{x+y}$.

Running time: $\mathcal{O}(1)$, but even $\mathcal{O}(d)$ will pass.

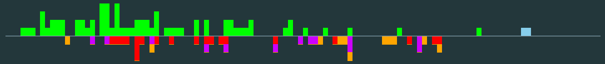
Pitfall: Values are larger than 10^9, do not use 32-bit `int`.

Pitfall: Many teams forgot the max function, which fails on the fourth sample.

Statistics: 191 submissions, 74 accepted

H: Hidden Sequence

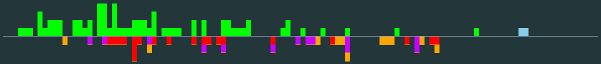
Problem author: Mike de Vries



Problem: Given three partial sequences of game winners, each missing one of the three players, determine the full sequence of winners.

H: Hidden Sequence

Problem author: Mike de Vries

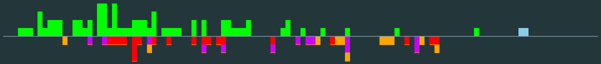


Problem: Given three partial sequences of game winners, each missing one of the three players, determine the full sequence of winners.

Observation: For each game, even though the winner does not list their name, the other two players do append the winner to their sequence. The winner of the first/final game is always the first/final name in exactly two of the sequences. Remove that name from the two sequences, and iterate.

H: Hidden Sequence

Problem author: Mike de Vries



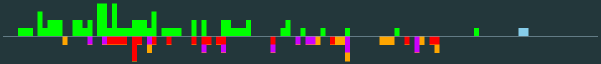
Problem: Given three partial sequences of game winners, each missing one of the three players, determine the full sequence of winners.

Observation: For each game, even though the winner does not list their name, the other two players do append the winner to their sequence. The winner of the first/final game is always the first/final name in exactly two of the sequences. Remove that name from the two sequences, and iterate.

Solution: Keep track of three indices in the three sequences, iteratively incrementing the two that agree on their value, and appending that value to the solution.

H: Hidden Sequence

Problem author: Mike de Vries



Problem: Given three partial sequences of game winners, each missing one of the three players, determine the full sequence of winners.

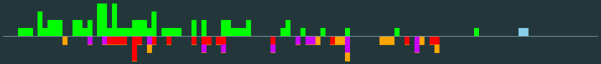
Observation: For each game, even though the winner does not list their name, the other two players do append the winner to their sequence. The winner of the first/final game is always the first/final name in exactly two of the sequences. Remove that name from the two sequences, and iterate.

Solution: Keep track of three indices in the three sequences, iteratively incrementing the two that agree on their value, and appending that value to the solution.

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

H: Hidden Sequence

Problem author: Mike de Vries



Problem: Given three partial sequences of game winners, each missing one of the three players, determine the full sequence of winners.

Observation: For each game, even though the winner does not list their name, the other two players do append the winner to their sequence. The winner of the first/final game is always the first/final name in exactly two of the sequences. Remove that name from the two sequences, and iterate.

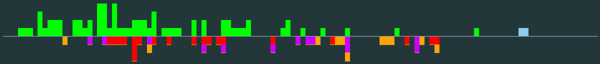
Solution: Keep track of three indices in the three sequences, iteratively incrementing the two that agree on their value, and appending that value to the solution.

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

Pitfall: Modifying the sequences themselves by removing characters from the strings can be unwantedly slow. Popping characters from the end is fine, but using `erase` is an $\mathcal{O}(n)$ operation $\implies \mathcal{O}(n^2)$ running time.

H: Hidden Sequence

Problem author: Mike de Vries



Problem: Given three partial sequences of game winners, each missing one of the three players, determine the full sequence of winners.

Observation: For each game, even though the winner does not list their name, the other two players do append the winner to their sequence. The winner of the first/final game is always the first/final name in exactly two of the sequences. Remove that name from the two sequences, and iterate.

Solution: Keep track of three indices in the three sequences, iteratively incrementing the two that agree on their value, and appending that value to the solution.

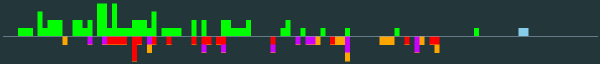
Running time: $\mathcal{O}(n)$.

Pitfall: Modifying the sequences themselves by removing characters from the strings can be unwantedly slow. Popping characters from the end is fine, but using `erase` is an $\mathcal{O}(n)$ operation $\implies \mathcal{O}(n^2)$ running time.

However: String operations are extremely optimized, so in practice this often still passes.

H: Hidden Sequence

Problem author: Mike de Vries



Problem: Given three partial sequences of game winners, each missing one of the three players, determine the full sequence of winners.

Observation: For each game, even though the winner does not list their name, the other two players do append the winner to their sequence. The winner of the first/final game is always the first/final name in exactly two of the sequences. Remove that name from the two sequences, and iterate.

Solution: Keep track of three indices in the three sequences, iteratively incrementing the two that agree on their value, and appending that value to the solution.

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

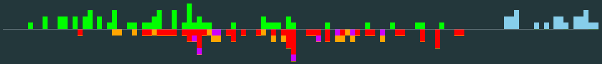
Pitfall: Modifying the sequences themselves by removing characters from the strings can be unwantedly slow. Popping characters from the end is fine, but using `erase` is an $\mathcal{O}(n)$ operation $\implies$ $\mathcal{O}(n^2)$ running time.

However: String operations are extremely optimized, so in practice this often still passes.

Statistics: 117 submissions, 71 accepted, 2 unknown

J: Journal Publication

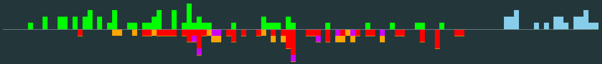
Problem author: Ragnar Groot Koerkamp



Problem: Is it possible to pick a name part for each author such that the list is sorted?

J: Journal Publication

Problem author: Ragnar Groot Koerkamp

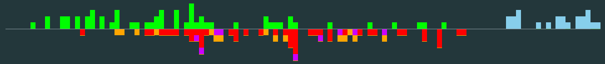


Problem: Is it possible to pick a name part for each author such that the list is sorted?

Observation: It is always optimal to pick the lexicographically smallest name part that is still allowed by the previous name part.

J: Journal Publication

Problem author: Ragnar Groot Koerkamp



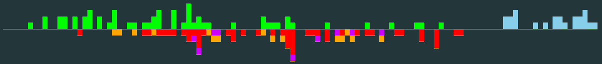
Problem: Is it possible to pick a name part for each author such that the list is sorted?

Observation: It is always optimal to pick the lexicographically smallest name part that is still allowed by the previous name part.

Solution: For each author in order, determine the appropriate name part in linear time with a running minimum.

J: Journal Publication

Problem author: Ragnar Groot Koerkamp



Problem: Is it possible to pick a name part for each author such that the list is sorted?

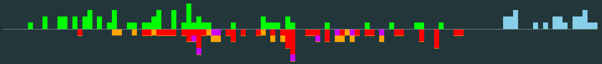
Observation: It is always optimal to pick the lexicographically smallest name part that is still allowed by the previous name part.

Solution: For each author in order, determine the appropriate name part in linear time with a running minimum.

Running time: Linear in the total size of the input.

J: Journal Publication

Problem author: Ragnar Groot Koerkamp



Problem: Is it possible to pick a name part for each author such that the list is sorted?

Observation: It is always optimal to pick the lexicographically smallest name part that is still allowed by the previous name part.

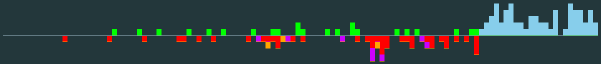
Solution: For each author in order, determine the appropriate name part in linear time with a running minimum.

Running time: Linear in the total size of the input.

Statistics: 152 submissions, 57 accepted, 23 unknown

E: Entropy Evasion

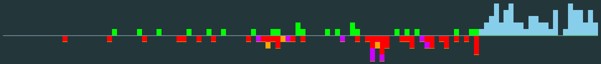
Problem author: Ragnar Groot Koerkamp



Problem: Repeatedly randomize a consecutive string of bits until at least 70% are 1.

E: Entropy Evasion

Problem author: Ragnar Groot Koerkamp

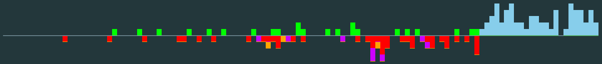


Problem: Repeatedly randomize a consecutive string of bits until at least 70% are 1.

Observation: The expected gain of a 0 is $\frac{1}{2}$ and that of a 1 is $-\frac{1}{2}$. So when we choose a string with a zeroes and b ones, the expected gain is $(a-b)/2$.

E: Entropy Evasion

Problem author: Ragnar Groot Koerkamp



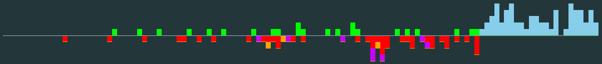
Problem: Repeatedly randomize a consecutive string of bits until at least 70% are 1.

Observation: The expected gain of a 0 is $\frac{1}{2}$ and that of a 1 is $-\frac{1}{2}$. So when we choose a string with a zeroes and b ones, the expected gain is $(a-b)/2$.

Solution: Repeatedly choose the string that maximizes expected gain.

E: Entropy Evasion

Problem author: Ragnar Groot Koerkamp



Problem: Repeatedly randomize a consecutive string of bits until at least 70% are 1.

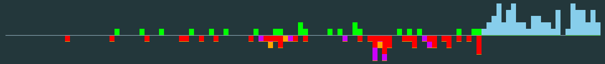
Observation: The expected gain of a 0 is $\frac{1}{2}$ and that of a 1 is $-\frac{1}{2}$. So when we choose a string with a zeroes and b ones, the expected gain is $(a-b)/2$.

Solution: Repeatedly choose the string that maximizes expected gain.

Implementation: To find this, calculate all prefix sums of expected gains, and take the largest increase.

E: Entropy Evasion

Problem author: Ragnar Groot Koerkamp



Problem: Repeatedly randomize a consecutive string of bits until at least 70% are 1.

Observation: The expected gain of a 0 is $\frac{1}{2}$ and that of a 1 is $-\frac{1}{2}$. So when we choose a string with a zeroes and b ones, the expected gain is $(a-b)/2$.

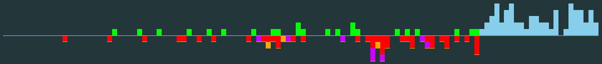
Solution: Repeatedly choose the string that maximizes expected gain.

Implementation: To find this, calculate all prefix sums of expected gains, and take the largest increase.

Pitfall: Off-by-one errors when finding the maximum subarray are hard to debug, and only fail on a few testcases.

E: Entropy Evasion

Problem author: Ragnar Groot Koerkamp



Problem: Repeatedly randomize a consecutive string of bits until at least 70% are 1.

Observation: The expected gain of a 0 is $\frac{1}{2}$ and that of a 1 is $-\frac{1}{2}$. So when we choose a string with a zeroes and b ones, the expected gain is $(a-b)/2$.

Solution: Repeatedly choose the string that maximizes expected gain.

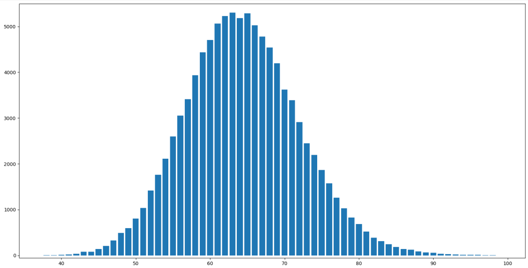
Implementation: To find this, calculate all prefix sums of expected gains, and take the largest increase.

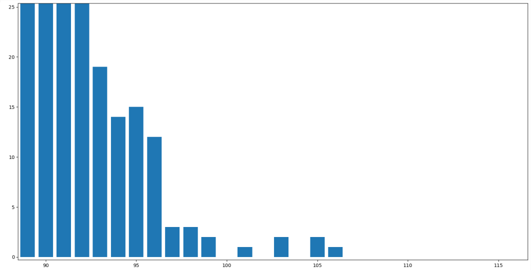
Pitfall: Off-by-one errors when finding the maximum subarray are hard to debug, and only fail on a few testcases.

Running time: $\mathcal{O}(nq)$ for q commands.

Problem author: Ragnar Groot Koerkamp

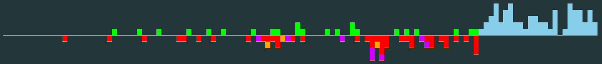
Statistical analysis: Out of 100 000 runs, the highest number of commands used is 106.
Lowest number of commands is 34.



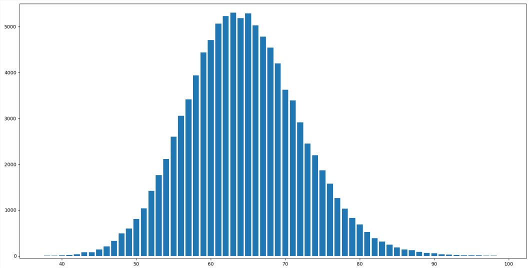


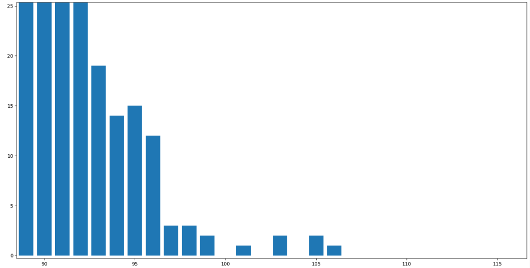
E: Entropy Evasion

Problem author: Ragnar Groot Koerkamp



Statistical analysis: Out of 100 000 runs, the highest number of commands used is 106.
Lowest number of commands is 34.

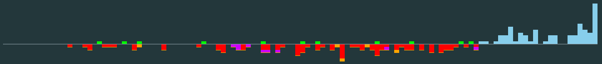




Statistics: 138 submissions, 23 accepted, 64 unknown

G: Genealogy Gumbo

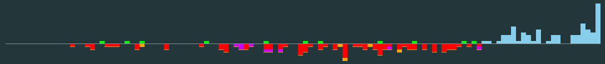
Problem author: Lammert Westerdijk



Problem: Determine whether everyone can have the same ancestor.

G: Genealogy Gumbo

Problem author: Lammert Westerdijk

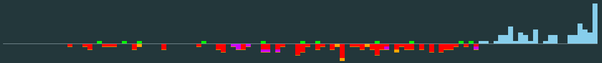


Problem: Determine whether everyone can have the same ancestor.

Initial idea: This is a graph problem.

G: Genealogy Gumbo

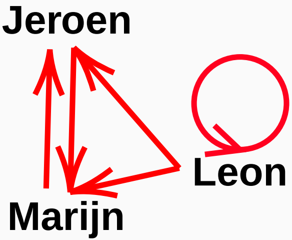
Problem author: Lammert Westerdijk

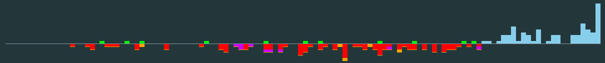


Problem: Determine whether everyone can have the same ancestor.

Initial idea: This is a graph problem.

Naive solution: Make a directed graph. If A is a son of B, add a directed edge from B to A.

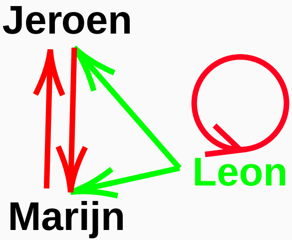




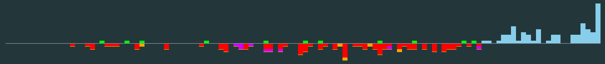
Problem: Determine whether everyone can have the same ancestor.

Initial idea: This is a graph problem.

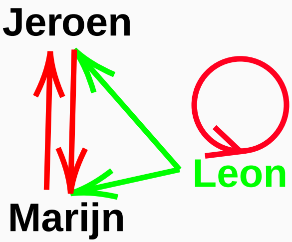
Naive solution: Make a directed graph. If A is a son of B, add a directed edge from B to A.



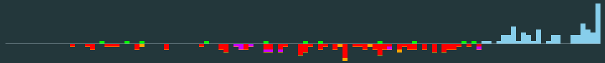
Naive solution: Output `possible` if some node reaches all other nodes, and `impossible` otherwise.



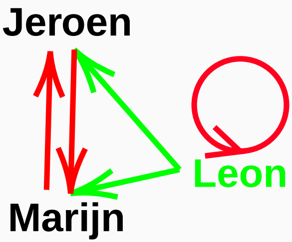
Problem: Determine whether everyone can have the same ancestor.



Naive solution: Output `possible` if some node reaches all other nodes, and `impossible` otherwise.



Problem: Determine whether everyone can have the same ancestor.



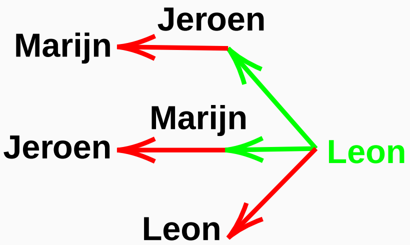
Naive solution: Output `possible` if some node reaches all other nodes, and `impossible` otherwise.

Issue: Take some spanning tree. What about unused edges?

G: Genealogy Gumbo

Problem author: Lammert Westerdijk

Problem: Determine whether everyone can have the same ancestor.



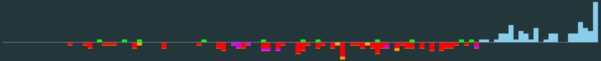
Naive solution: Output `possible` if some node reaches all other nodes, and `impossible` otherwise.

Issue: Take some spanning tree. What about unused edges?

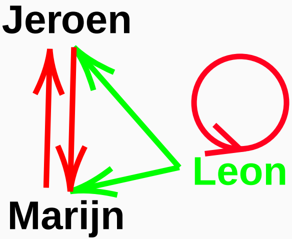
Fix: Make valid tree by adding an extra person for each unused edge.

G: Genealogy Gumbo

Problem author: Lammert Westerdijk



Problem: Determine whether everyone can have the same ancestor.



Naive solution: Output `possible` if some node reaches all other nodes, and `impossible` otherwise.

Issue: Take some spanning tree. What about unused edges?

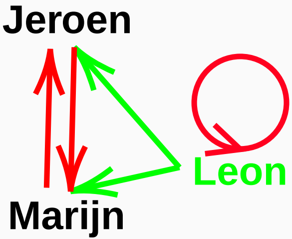
Fix: Make valid tree by adding an extra person for each unused edge.

Solution: Therefore, we need to find a node that can reach all other nodes.

G: Genealogy Gumbo

Problem author: Lammert Westerdijk

Problem: Determine whether everyone can have the same ancestor.



Naive solution: Output `possible` if some node reaches all other nodes, and `impossible` otherwise.

Issue: Take some spanning tree. What about unused edges?

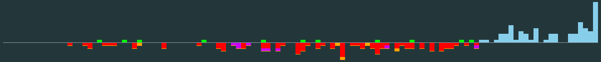
Fix: Make valid tree by adding an extra person for each unused edge.

Solution: Therefore, we need to find a node that can reach all other nodes.

Issue: $n = 10^5$, so we need an $\mathcal{O}(n)$-time solution.

G: Genealogy Gumbo

Problem author: Lammert Westerdijk

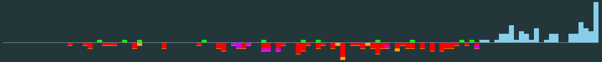


Solution:

- Loop through all nodes, and run a DFS if not yet visited.

G: Genealogy Gumbo

Problem author: Lammert Westerdijk

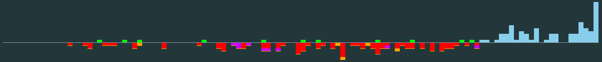


Solution:

- Loop through all nodes, and run a DFS if not yet visited.
- Once we reach a “good” node, all nodes are marked as visited.

G: Genealogy Gumbo

Problem author: Lammert Westerdijk



Solution:

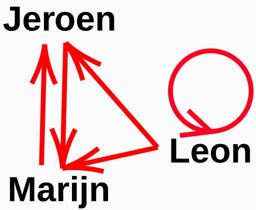
- Loop through all nodes, and run a DFS if not yet visited.
- Once we reach a “good” node, all nodes are marked as visited.
- If a “good” node exists, the last node where we started a DFS is “good”.

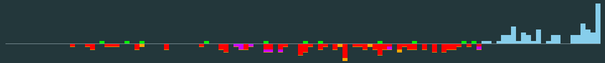
G: Genealogy Gumbo

Problem author: Lammert Westerdijk

Solution:

- Loop through all nodes, and run a DFS if not yet visited.
- Once we reach a "good" node, all nodes are marked as visited.
- If a "good" node exists, the last node where we started a DFS is "good".

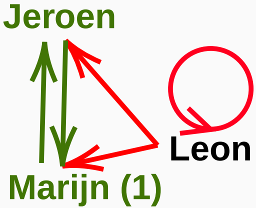




Problem author: Lammert Westerdijk

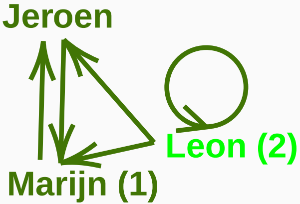
Solution:

- Loop through all nodes, and run a DFS if not yet visited.
- Once we reach a “good” node, all nodes are marked as visited.
- If a “good” node exists, the last node where we started a DFS is “good”.



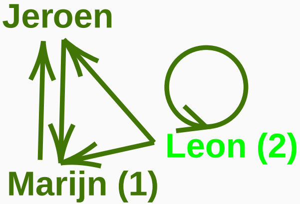
Solution:

- Loop through all nodes, and run a DFS if not yet visited.
- Once we reach a “good” node, all nodes are marked as visited.
- If a “good” node exists, the last node where we started a DFS is “good”.



Solution:

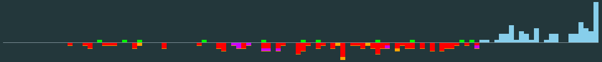
- Loop through all nodes, and run a DFS if not yet visited.
- Once we reach a "good" node, all nodes are marked as visited.
- If a "good" node exists, the last node where we started a DFS is "good".



- Check whether this is true using a DFS, starting from this last node.

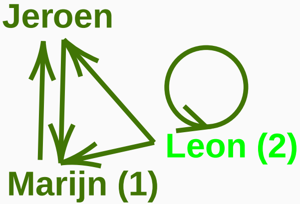
G: Genealogy Gumbo

Problem author: Lammert Westerdijk



Solution:

- Loop through all nodes, and run a DFS if not yet visited.
- Once we reach a "good" node, all nodes are marked as visited.
- If a "good" node exists, the last node where we started a DFS is "good".

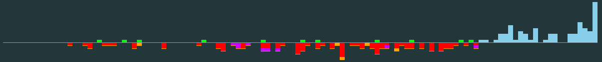


- Check whether this is true using a DFS, starting from this last node.

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

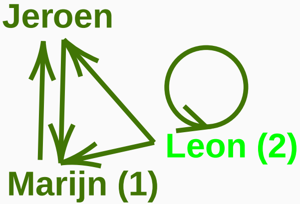
G: Genealogy Gumbo

Problem author: Lammert Westerdijk



Solution:

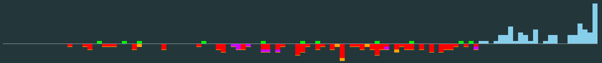
- Loop through all nodes, and run a DFS if not yet visited.
- Once we reach a "good" node, all nodes are marked as visited.
- If a "good" node exists, the last node where we started a DFS is "good".



- Check whether this is true using a DFS, starting from this last node.

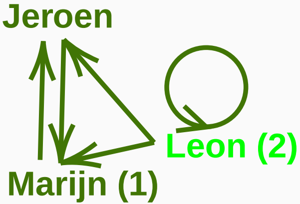
Running time: $\mathcal{O}(n)$.

Alternative solution: Using strongly connected components.



Solution:

- Loop through all nodes, and run a DFS if not yet visited.
- Once we reach a "good" node, all nodes are marked as visited.
- If a "good" node exists, the last node where we started a DFS is "good".



- Check whether this is true using a DFS, starting from this last node.

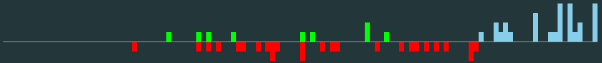
Running time: $\mathcal{O}(n)$.

Alternative solution: Using strongly connected components.

Statistics: 176 submissions, 11 accepted, 72 unknown

I: Ingredient Intervals

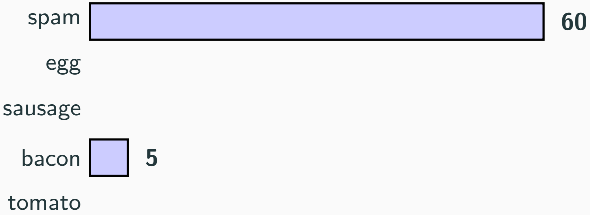
Problem author: Mike de Vries



Problem: There is a sequence of real numbers $a = (a_1, \ldots, a_n)$ with $a_1 + \cdots + a_n = 100$ and $a_1 \geq \cdots \geq a_n \geq 0$. Given a subset $\{a_i\}_{i \in S}$ of these numbers, determine maximal lower bounds $l_1, \ldots, l_n$ and minimal upper bounds $r_1, \ldots, r_n$ such that

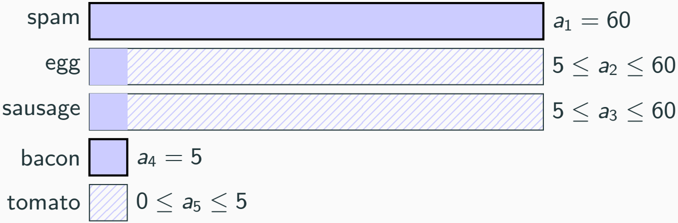
$$l_i \leq a_i \leq r_i \quad \text{for } 1 \leq i \leq n\,.$$

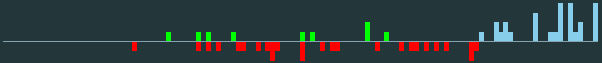
E.g., with $a = (60, ?, ?, 5, ?)$:



First observation: Since a is nonincreasing, it is immediate that a_i satisfies $l'_i \le a \le r'_i$ with

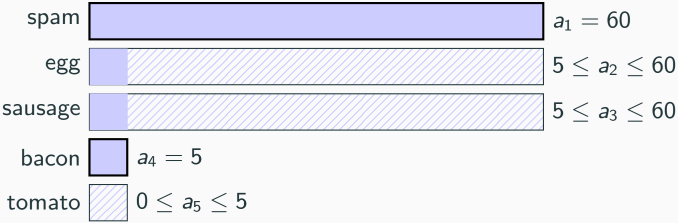
$$l'_i = \max\{\, a_j : j \ge i, j \in S \,\} \qquad r'_i = \min\{\, a_j : j \le i, j \in S \,\}\,.$$





First observation: Since a is nonincreasing, it is immediate that a_i satisfies $l_i' \le a \le r_i'$ with

$$l_i' = \max\{\, a_j : j \ge i, j \in S \,\} \qquad r_i' = \min\{\, a_j : j \le i, j \in S \,\}\,.$$



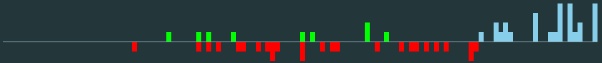
Implementation: These values can be computed in linear time by first computing

$$\mathrm{prev}(i) = \max\{\, j \in S : j < i \,\}, \qquad \mathrm{next}(i) = \max\{\, j \in S : j > i \,\}\,,$$

preferably with useful boundary values, for instance by introducing $a_0 = 100$ and $a_{n+1} = 0$. Then, for $i \notin S$ we have $l_i' = a_j$ with $j = \mathrm{next}(i)$ because a is decreasing.

I: Ingredient Intervals

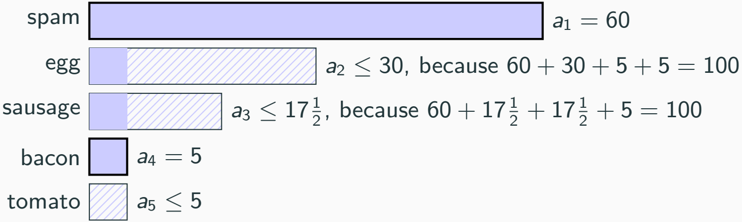
Problem author: Mike de Vries



Fix upper bounds: For $i \notin S$, the (unknown) amount a_i satisfies $l_i' \leq a_i \leq r_i'$, but we know more: Assuming all other amounts attain their *lower* bound, then their sum cannot exceed 100, so we have the constraint

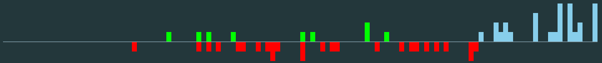
$$\left(\sum_{j<i} \max(a_i, l_j')\right) + a_i + \sum_{j>i} l_j' \leq 100\,,$$

Can solve for the largest a_i.



I: Ingredient Intervals

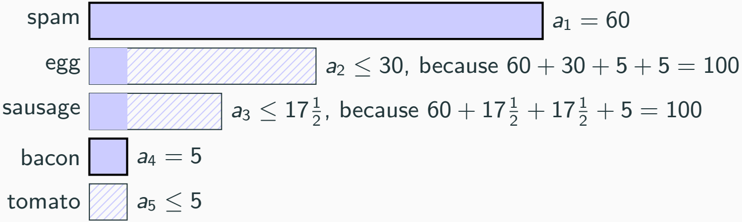
Problem author: Mike de Vries



Fix upper bounds: For $i \notin S$, the (unknown) amount a_i satisfies $l'_i \leq a_i \leq r'_i$, but we know more: Assuming all other amounts attain their *lower* bound, then their sum cannot exceed 100, so we have the constraint

$$\left(\sum_{j<i} \max(a_i, l'_j)\right) + a_i + \sum_{j>i} l'_j \leq 100\,,$$

Can solve for the largest a_i.



Implementation: To determine $\max a_i$, use binary search in the interval $[l'_i, r'_i]$ or rewrite the constraint for closed formula.

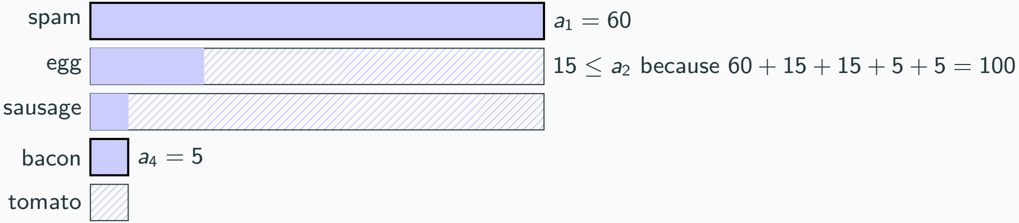
I: Ingredient Intervals

Problem author: Mike de Vries

Fix lower bounds: Symmetrically, assuming all other amounts attain their *maximum* bound, then their sum must be at least 100, so we have the constraint

$$\left(\sum_{j<i} r'_j\right) + a_i + \sum_{j>i} \min(a_i, r'_j) \geq 100\,,$$

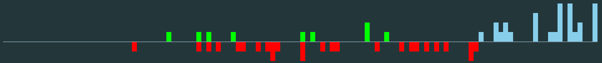
Can solve for the smallest a_i.



Tightness: Observe that the improved bounds are tight because the constraints describe valid values for a.

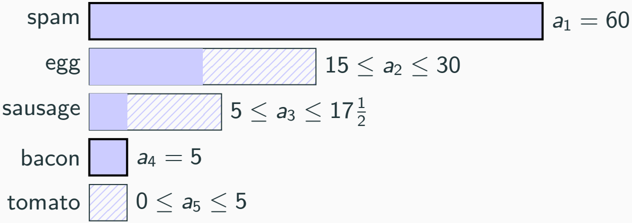
I: Ingredient Intervals

Problem author: Mike de Vries



Tightness: Observe that the improved bounds are tight because the constraints describe valid values for a.

Combine bounds:

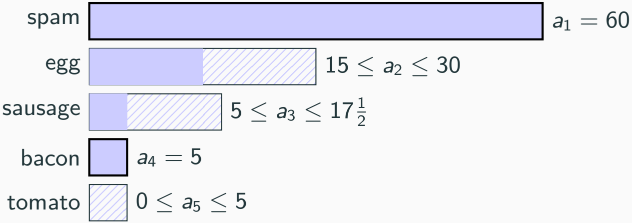


I: Ingredient Intervals

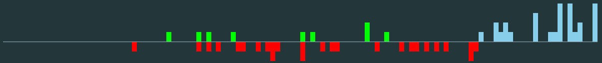
Problem author: Mike de Vries

Tightness: Observe that the improved bounds are tight because the constraints describe valid values for a.

Combine bounds:

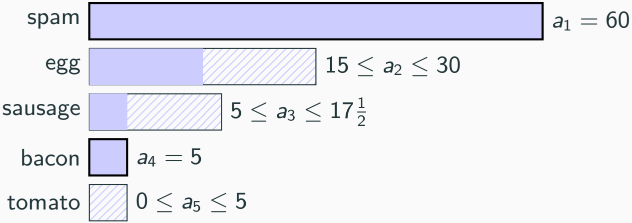


Running time: $\mathcal{O}\left(n^2 \log(100/\varepsilon)\right)$ using binary search.
With precomputing next and prev and solving the constraints: $\mathcal{O}(n)$.



Tightness: Observe that the improved bounds are tight because the constraints describe valid values for a.

Combine bounds:

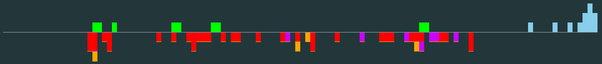


Running time: $\mathcal{O}\left(n^2 \log(100/\varepsilon)\right)$ using binary search.
With precomputing next and prev and solving the constraints: $\mathcal{O}(n)$.

Statistics: 62 submissions, 9 accepted, 27 unknown

A: Alto Adaptation

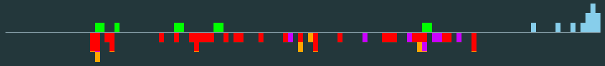
Problem author: Arnoud van der Leer, Maarten Sijm



Problem: By transposing notes to fit your vocal range, maximize the shortest interval of equally transposed notes.

A: Alto Adaptation

Problem author: Arnoud van der Leer, Maarten Sijm

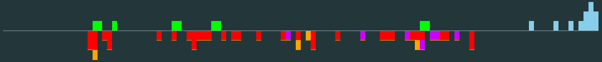


Problem: By transposing notes to fit your vocal range, maximize the shortest interval of equally transposed notes.

Solution: Dynamic Programming!

A: Alto Adaptation

Problem author: Arnoud van der Leer, Maarten Sijm



Problem: By transposing notes to fit your vocal range, maximize the shortest interval of equally transposed notes.

Solution: Dynamic Programming!

- Define $d[i]$ as the optimal value if the musical piece ended after i notes.

A: Alto Adaptation

Problem author: Arnoud van der Leer, Maarten Sijm

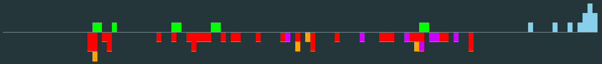
Problem: By transposing notes to fit your vocal range, maximize the shortest interval of equally transposed notes.

Solution: Dynamic Programming!

- Define $d[i]$ as the optimal value if the musical piece ended after i notes.
- **Base Case:** Let $d[0] = \infty$.

A: Alto Adaptation

Problem author: Arnoud van der Leer, Maarten Sijm



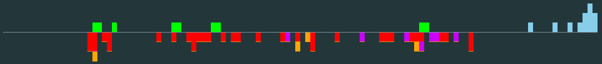
Problem: By transposing notes to fit your vocal range, maximize the shortest interval of equally transposed notes.

Solution: Dynamic Programming!

- Define $d[i]$ as the optimal value if the musical piece ended after i notes.
- **Base Case:** Let $d[0] = \infty$.
- **Recursion:** To calculate $d[i]$: for $j = i, \ldots, 1$, keep track of the range of possible transposals for the notes in $[j, i]$. (The range of possible transposals becomes smaller the further you go back in in the song.)
  Then, $d[i]$ is the maximal value of $\min\{d[j-1], i-j+1\}$ of each j with non-empty transposal range.

A: Alto Adaptation

Problem author: Arnoud van der Leer, Maarten Sijm



Problem: By transposing notes to fit your vocal range, maximize the shortest interval of equally transposed notes.

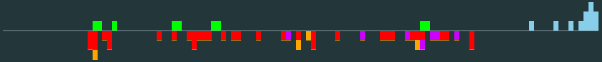
Solution: Dynamic Programming!

- Define $d[i]$ as the optimal value if the musical piece ended after i notes.
- **Base Case:** Let $d[0] = \infty$.
- **Recursion:** To calculate $d[i]$: for $j = i, \ldots, 1$, keep track of the range of possible transposals for the notes in $[j, i]$. (The range of possible transposals becomes smaller the further you go back in in the song.)
  Then, $d[i]$ is the maximal value of $\min\{d[j-1], i-j+1\}$ of each j with non-empty transposal range.

Running time: $\mathcal{O}(n^2)$.

A: Alto Adaptation

Problem author: Arnoud van der Leer, Maarten Sijm



Problem: By transposing notes to fit your vocal range, maximize the shortest interval of equally transposed notes.

Solution: Dynamic Programming!

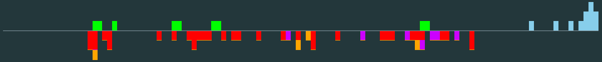
- Define $d[i]$ as the optimal value if the musical piece ended after i notes.
- **Base Case:** Let $d[0] = \infty$.
- **Recursion:** To calculate $d[i]$: for $j = i, \ldots, 1$, keep track of the range of possible transposals for the notes in $[j, i]$. (The range of possible transposals becomes smaller the further you go back in in the song.)
  Then, $d[i]$ is the maximal value of $\min\{d[j-1], i-j+1\}$ of each j with non-empty transposal range.

Running time: $\mathcal{O}(n^2)$.

Fun fact: $\mathcal{O}(n \log n)$ is possible with segment trees, or with binary search on the answer.

A: Alto Adaptation

Problem author: Arnoud van der Leer, Maarten Sijm



Problem: By transposing notes to fit your vocal range, maximize the shortest interval of equally transposed notes.

Solution: Dynamic Programming!

- Define $d[i]$ as the optimal value if the musical piece ended after i notes.
- **Base Case:** Let $d[0] = \infty$.
- **Recursion:** To calculate $d[i]$: for $j = i, \ldots, 1$, keep track of the range of possible transposals for the notes in $[j, i]$. (The range of possible transposals becomes smaller the further you go back in in the song.)
  Then, $d[i]$ is the maximal value of $\min\{d[j-1], i-j+1\}$ of each j with non-empty transposal range.

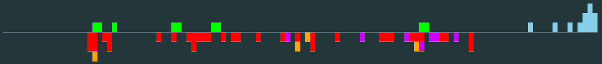
Running time: $\mathcal{O}(n^2)$.

Fun fact: $\mathcal{O}(n \log n)$ is possible with segment trees, or with binary search on the answer.

Extra fun fact: Linear is possible! This was discovered after the contest. See the jury solutions.

A: Alto Adaptation

Problem author: Arnoud van der Leer, Maarten Sijm



Problem: By transposing notes to fit your vocal range, maximize the shortest interval of equally transposed notes.

Solution: Dynamic Programming!

- Define $d[i]$ as the optimal value if the musical piece ended after i notes.
- **Base Case:** Let $d[0] = \infty$.
- **Recursion:** To calculate $d[i]$: for $j = i, \dots, 1$, keep track of the range of possible transposals for the notes in $[j, i]$. (The range of possible transposals becomes smaller the further you go back in in the song.)
  Then, $d[i]$ is the maximal value of $\min\{d[j-1], i-j+1\}$ of each j with non-empty transposal range.

Running time: $\mathcal{O}(n^2)$.

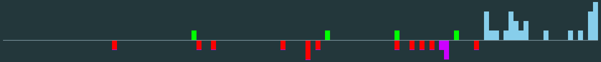
Fun fact: $\mathcal{O}(n \log n)$ is possible with segment trees, or with binary search on the answer.

Extra fun fact: Linear is possible! This was discovered after the contest. See the jury solutions.

Statistics: 65 submissions, 9 accepted, 11 unknown

F: Friendly Formation

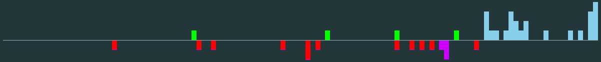
Problem author: Tobias Roehr



Problem: Partition a graph into two equally sized cliques.

F: Friendly Formation

Problem author: Tobias Roehr

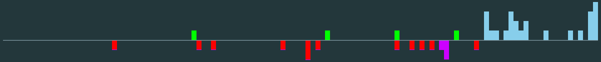


Problem: Partition a graph into two equally sized cliques.

Condition: Need $n = 2k$ and need $m \geq k(k-1)$, which gives $n \leq 2(\sqrt{m} + 1) \leq 2002$.

F: Friendly Formation

Problem author: Tobias Roehr



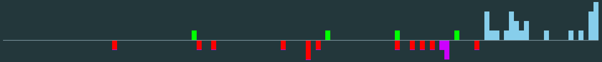
Problem: Partition a graph into two equally sized cliques.

Condition: Need $n = 2k$ and need $m \geq k(k-1)$, which gives $n \leq 2(\sqrt{m}+1) \leq 2002$.

Reduction: Take the edge complement, we need to partition the graph into two independent sets of size k.

F: Friendly Formation

Problem author: Tobias Roehr



Problem: Partition a graph into two equally sized cliques.

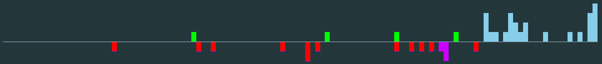
Condition: Need $n = 2k$ and need $m \geq k(k-1)$, which gives $n \leq 2(\sqrt{m} + 1) \leq 2002$.

Reduction: Take the edge complement, we need to partition the graph into two independent sets of size k.

Condition: Each connected component needs to be bipartite.

F: Friendly Formation

Problem author: Tobias Roehr



Problem: Partition a graph into two equally sized cliques.

Condition: Need $n = 2k$ and need $m \geq k(k-1)$, which gives $n \leq 2(\sqrt{m}+1) \leq 2002$.

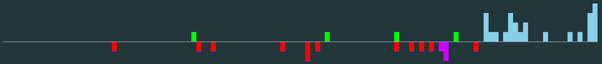
Reduction: Take the edge complement, we need to partition the graph into two independent sets of size k.

Condition: Each connected component needs to be bipartite.

Reduction: Let component i have parts of size a_i and b_i. We need to pick one of each tuple and end up with a total of exactly k.

F: Friendly Formation

Problem author: Tobias Roehr



Problem: Partition a graph into two equally sized cliques.

Condition: Need $n = 2k$ and need $m \geq k(k-1)$, which gives $n \leq 2(\sqrt{m}+1) \leq 2002$.

Reduction: Take the edge complement, we need to partition the graph into two independent sets of size k.

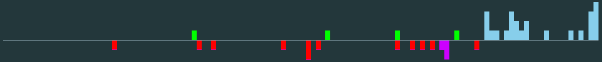
Condition: Each connected component needs to be bipartite.

Reduction: Let component i have parts of size a_i and b_i. We need to pick one of each tuple and end up with a total of exactly k.

Solution: Knapsack dynamic programming!

F: Friendly Formation

Problem author: Tobias Roehr



Problem: Partition a graph into two equally sized cliques.

Condition: Need $n = 2k$ and need $m \geq k(k-1)$, which gives $n \leq 2(\sqrt{m}+1) \leq 2002$.

Reduction: Take the edge complement, we need to partition the graph into two independent sets of size k.

Condition: Each connected component needs to be bipartite.

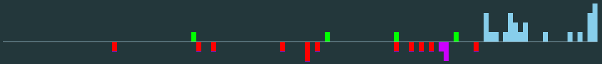
Reduction: Let component i have parts of size a_i and b_i. We need to pick one of each tuple and end up with a total of exactly k.

Solution: Knapsack dynamic programming!

Variables: Let $T[j][s]$ be 1 if we can pick one of each tuple $1, \ldots, j$ and end up with a total of exactly s, and 0 otherwise.

F: Friendly Formation

Problem author: Tobias Roehr



Problem: Partition a graph into two equally sized cliques.

Condition: Need $n = 2k$ and need $m \geq k(k-1)$, which gives $n \leq 2(\sqrt{m}+1) \leq 2002$.

Reduction: Take the edge complement, we need to partition the graph into two independent sets of size k.

Condition: Each connected component needs to be bipartite.

Reduction: Let component i have parts of size a_i and b_i. We need to pick one of each tuple and end up with a total of exactly k.

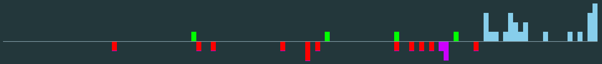
Solution: Knapsack dynamic programming!

Variables: Let $T[j][s]$ be 1 if we can pick one of each tuple $1, \ldots, j$ and end up with a total of exactly s, and 0 otherwise.

Recursion: We have $T[j][s] = 1$ if either $T[j-1][s-a_j] = 1$ or $T[j-1][s-b_j] = 1$.

F: Friendly Formation

Problem author: Tobias Roehr



Problem: Partition a graph into two equally sized cliques.

Condition: Need $n = 2k$ and need $m \geq k(k-1)$, which gives $n \leq 2(\sqrt{m}+1) \leq 2002$.

Reduction: Take the edge complement, we need to partition the graph into two independent sets of size k.

Condition: Each connected component needs to be bipartite.

Reduction: Let component i have parts of size a_i and b_i. We need to pick one of each tuple and end up with a total of exactly k.

Solution: Knapsack dynamic programming!

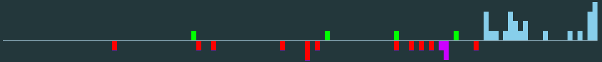
Variables: Let $T[j][s]$ be 1 if we can pick one of each tuple $1, \ldots, j$ and end up with a total of exactly s, and 0 otherwise.

Recursion: We have $T[j][s] = 1$ if either $T[j-1][s-a_j] = 1$ or $T[j-1][s-b_j] = 1$.

Running time: $\mathcal{O}(n^2) = \mathcal{O}(m)$.

F: Friendly Formation

Problem author: Tobias Roehr



Problem: Partition a graph into two equally sized cliques.

Condition: Need $n = 2k$ and need $m \geq k(k-1)$, which gives $n \leq 2(\sqrt{m}+1) \leq 2002$.

Reduction: Take the edge complement, we need to partition the graph into two independent sets of size k.

Condition: Each connected component needs to be bipartite.

Reduction: Let component i have parts of size a_i and b_i. We need to pick one of each tuple and end up with a total of exactly k.

Solution: Knapsack dynamic programming!

Variables: Let $T[j][s]$ be 1 if we can pick one of each tuple $1, \ldots, j$ and end up with a total of exactly s, and 0 otherwise.

Recursion: We have $T[j][s] = 1$ if either $T[j-1][s-a_j] = 1$ or $T[j-1][s-b_j] = 1$.

Running time: $\mathcal{O}(n^2) = \mathcal{O}(m)$.

Statistics: 43 submissions, 4 accepted, 24 unknown

Problem: Connect some of the given points in a cycle to create a polygon with all interior angles at least 90 degrees.

L: Landgrave

Problem author: Lammert Westerdijk

Problem: Connect some of the given points in a cycle to create a polygon with all interior angles at least 90 degrees.

Idea: Start with the convex hull, and simply verify whether this is a solution.

L: Landgrave

Problem author: Lammert Westerdijk

Problem: Connect some of the given points in a cycle to create a polygon with all interior angles at least 90 degrees.

Idea: Start with the convex hull, and simply verify whether this is a solution.

Observation: If not, it has a sharp interior angle at some point P. But then all points lie within a sharp angle with respect to P, so P can never be part of a solution!

L: Landgrave

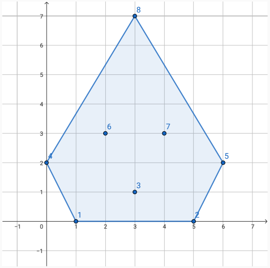
Problem author: Lammert Westerdijk

Problem: Connect some of the given points in a cycle to create a polygon with all interior angles at least 90 degrees.

Idea: Start with the convex hull, and simply verify whether this is a solution.

Observation: If not, it has a sharp interior angle at some point P. But then all points lie within a sharp angle with respect to P, so P can never be part of a solution!

Solution: Repeatedly calculate the convex hull and remove the points that make a sharp angle from the point set, until we have a solution, or there are less than 4 points remaining.



L: Landgrave

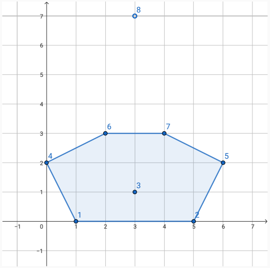
Problem author: Lammert Westerdijk

Problem: Connect some of the given points in a cycle to create a polygon with all interior angles at least 90 degrees.

Idea: Start with the convex hull, and simply verify whether this is a solution.

Observation: If not, it has a sharp interior angle at some point P. But then all points lie within a sharp angle with respect to P, so P can never be part of a solution!

Solution: Repeatedly calculate the convex hull and remove the points that make a sharp angle from the point set, until we have a solution, or there are less than 4 points remaining.



L: Landgrave

Problem author: Lammert Westerdijk

Problem: Connect some of the given points in a cycle to create a polygon with all interior angles at least 90 degrees.

Idea: Start with the convex hull, and simply verify whether this is a solution.

Observation: If not, it has a sharp interior angle at some point P. But then all points lie within a sharp angle with respect to P, so P can never be part of a solution!

Solution: Repeatedly calculate the convex hull and remove the points that make a sharp angle from the point set, until we have a solution, or there are less than 4 points remaining.

Running time: Need $\mathcal{O}(n \log n)$ convex hull algorithm to get $\mathcal{O}(n^2 \log n)$. Cubic is too slow.

L: Landgrave

Problem author: Lammert Westerdijk

Problem: Connect some of the given points in a cycle to create a polygon with all interior angles at least 90 degrees.

Idea: Start with the convex hull, and simply verify whether this is a solution.

Observation: If not, it has a sharp interior angle at some point P. But then all points lie within a sharp angle with respect to P, so P can never be part of a solution!

Solution: Repeatedly calculate the convex hull and remove the points that make a sharp angle from the point set, until we have a solution, or there are less than 4 points remaining.

Running time: Need $\mathcal{O}(n \log n)$ convex hull algorithm to get $\mathcal{O}(n^2 \log n)$. Cubic is too slow.

Bonus: With monotone chain convex hull algorithm, after $\mathcal{O}(n \log n)$ precomputation sorting the points, we can recalculate the hull in linear time, leading to $\mathcal{O}(n^2)$ total time.

L: Landgrave

Problem author: Lammert Westerdijk

Problem: Connect some of the given points in a cycle to create a polygon with all interior angles at least 90 degrees.

Idea: Start with the convex hull, and simply verify whether this is a solution.

Observation: If not, it has a sharp interior angle at some point P. But then all points lie within a sharp angle with respect to P, so P can never be part of a solution!

Solution: Repeatedly calculate the convex hull and remove the points that make a sharp angle from the point set, until we have a solution, or there are less than 4 points remaining.

Running time: Need $\mathcal{O}(n \log n)$ convex hull algorithm to get $\mathcal{O}(n^2 \log n)$. Cubic is too slow.

Bonus: With monotone chain convex hull algorithm, after $\mathcal{O}(n \log n)$ precomputation sorting the points, we can recalculate the hull in linear time, leading to $\mathcal{O}(n^2)$ total time.

Bonus 2: With a dynamic convex hull data structure, $\mathcal{O}(n \log n)$ time is possible.

L: Landgrave

Problem author: Lammert Westerdijk

Problem: Connect some of the given points in a cycle to create a polygon with all interior angles at least 90 degrees.

Idea: Start with the convex hull, and simply verify whether this is a solution.

Observation: If not, it has a sharp interior angle at some point P. But then all points lie within a sharp angle with respect to P, so P can never be part of a solution!

Solution: Repeatedly calculate the convex hull and remove the points that make a sharp angle from the point set, until we have a solution, or there are less than 4 points remaining.

Running time: Need $\mathcal{O}(n \log n)$ convex hull algorithm to get $\mathcal{O}(n^2 \log n)$. Cubic is too slow.

Bonus: With monotone chain convex hull algorithm, after $\mathcal{O}(n \log n)$ precomputation sorting the points, we can recalculate the hull in linear time, leading to $\mathcal{O}(n^2)$ total time.

Bonus 2: With a dynamic convex hull data structure, $\mathcal{O}(n \log n)$ time is possible.

Statistics: 13 submissions, 0 accepted, 11 unknown

C: Calculation Obfuscation

Problem author: Marijn Adriaanse

Problem: Reorder a string to form a syntactically valid expression without redundant parentheses.

C: Calculation Obfuscation

Problem author: Marijn Adriaanse

Problem: Reorder a string to form a syntactically valid expression without redundant parentheses.

Subproblem 1: Create variables and numbers out of letters and digits.

C: Calculation Obfuscation

Problem author: Marijn Adriaanse

Problem: Reorder a string to form a syntactically valid expression without redundant parentheses.

Subproblem 1: Create variables and numbers out of letters and digits.

Subproblem 2: Join these with operators and parentheses such that no parentheses are redundant.

Observation: If we have k operators, we need $k+1$ numbers/variable tokens regardless of structure.

C: Calculation Obfuscation

Problem author: Marijn Adriaanse

Problem: Reorder a string to form a syntactically valid expression without redundant parentheses.

Subproblem 1: Create variables and numbers out of letters and digits.

Subproblem 2: Join these with operators and parentheses such that no parentheses are redundant.

Observation: If we have k operators, we need $k+1$ numbers/variable tokens regardless of structure.

Observation: We need at least $k+1$ letters/digits.

C: Calculation Obfuscation

Problem author: Marijn Adriaanse

Problem: Reorder a string to form a syntactically valid expression without redundant parentheses.

Subproblem 1: Create variables and numbers out of letters and digits.

Subproblem 2: Join these with operators and parentheses such that no parentheses are redundant.

Observation: If we have k operators, we need $k+1$ numbers/variable tokens regardless of structure.

Observation: We need at least $k+1$ letters/digits.

Edge case: If we have only zeros, we need exactly $k+1$ of them.

C: Calculation Obfuscation

Problem author: Marijn Adriaanse

Problem: Reorder a string to form a syntactically valid expression without redundant parentheses.

Subproblem 1: Create variables and numbers out of letters and digits.

Subproblem 2: Join these with operators and parentheses such that no parentheses are redundant.

Observation: If we have k operators, we need $k + 1$ numbers/variable tokens regardless of structure.

Observation: We need at least $k + 1$ letters/digits.

Edge case: If we have only zeros, we need exactly $k + 1$ of them.

Possible solution: Create k numbers/variables using single characters, starting with zeros. Put the remaining characters in the last token, making sure to start with letters, then non-zero digits. Sorting makes it easier, but is not strictly necessary.

`00123abcd` $\rightarrow$ `0, 0, 1, 2, dcba3`

`000000001` $\rightarrow$ `0, 0, 0, 0, 10000`

C: Calculation Obfuscation

Problem author: Marijn Adriaanse

Observation: Parentheses are only allowed when multiplying a sum (e.g. `(a+b)*c`).

C: Calculation Obfuscation

Problem author: Marijn Adriaanse

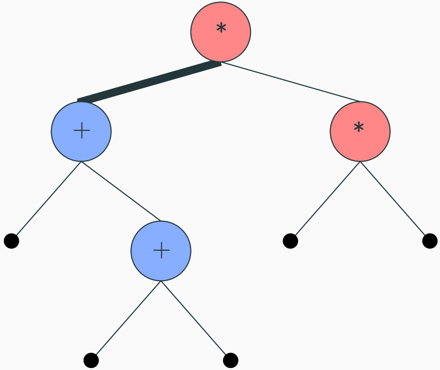
Observation: Parentheses are only allowed when multiplying a sum (e.g. `(a+b)*c`).

Observation: We can look at the expression as a binary tree, and not care about longer sums/products. Parentheses are allowed when a (+) is the child of a (∗).

Problem author: Marijn Adriaanse

Observation: Parentheses are only allowed when multiplying a sum (e.g. `(a+b)*c`).

Observation: We can look at the expression as a binary tree, and not care about longer sums/products. Parentheses are allowed when a (+) is the child of a (*).



$(\; \ldots \; + \; \ldots \; + \; \ldots \;) \; * \; \ldots \; * \; \ldots$

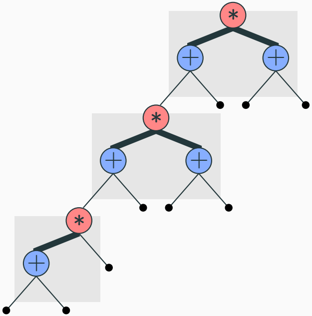
Observation: Since each operator can have at most one parent, and has two children, if there are M multiplication operators and P plus operators, we can place no more than $2M$ or A pairs of non-redundant parentheses.

C: Calculation Obfuscation

Problem author: Marijn Adriaanse

Observation: Since each operator can have at most one parent, and has two children, if there are M multiplication operators and P plus operators, we can place no more than $2M$ or A pairs of non-redundant parentheses.

Solution: Greedily construct as many (_+_)*(_+_) patterns as possible, use (_+_)*_ to get rid of a single pair of parentheses, and put all remaining operators at the end.



(((_+_)*_)*(_+_))*(_+_) +_+_+_*_*_

C: Calculation Obfuscation

Problem author: Marijn Adriaanse

Complete solution: Count the parentheses, operators, zeros, and other digits/letters, and check all edge conditions. Construct tokens. Then greedily construct the expression's structure, and fill in the tokens.

C: Calculation Obfuscation

Problem author: Marijn Adriaanse

Complete solution: Count the parentheses, operators, zeros, and other digits/letters, and check all edge conditions. Construct tokens. Then greedily construct the expression's structure, and fill in the tokens.

Running time: $\mathcal{O}(n)$, or $\mathcal{O}(n \log n)$ when sorting the characters.

C: Calculation Obfuscation

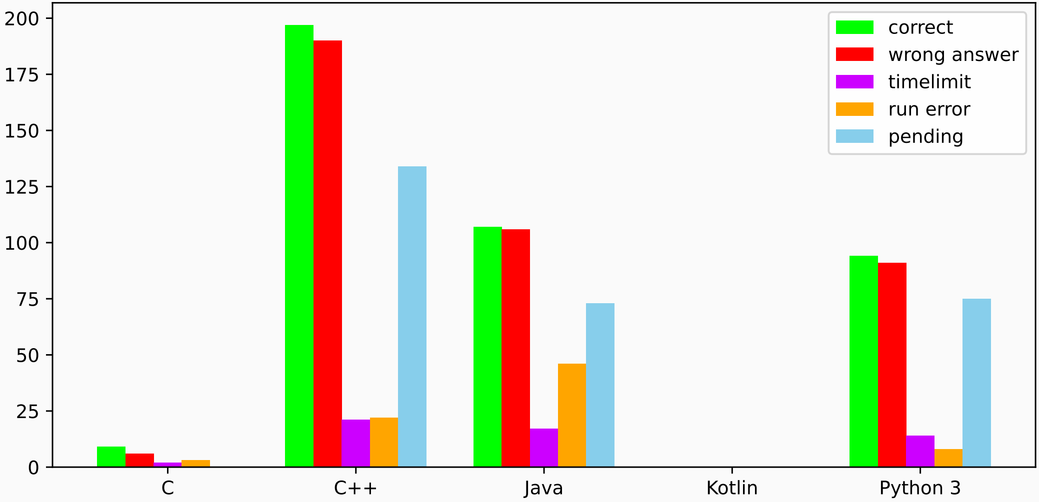
Problem author: Marijn Adriaanse

Complete solution: Count the parentheses, operators, zeros, and other digits/letters, and check all edge conditions. Construct tokens. Then greedily construct the expression's structure, and fill in the tokens.

Running time: $\mathcal{O}(n)$, or $\mathcal{O}(n \log n)$ when sorting the characters.

Statistics: 83 submissions, 2 accepted, 44 unknown

Language stats



Random facts

Jury work

- 440 commits (last year: 505)

Random facts

Jury work

- 440 commits (last year: 505)
- 802 secret test cases (last year: 1228) ($\approx$ 67 per problem!)

Random facts

Jury work

- 440 commits (last year: 505)
- 802 secret test cases (last year: 1228) ($\approx$ 67 per problem!)
- 221 jury + proofreader solutions (last year: 236)

Random facts

Jury work

- 440 commits (last year: 505)
- 802 secret test cases (last year: 1228) ($\approx$ 67 per problem!)
- 221 jury + proofreader solutions (last year: 236)
- The minimum[1] number of lines the jury needed to solve all problems is

$$4 + 2 + 6 + 1 + 4 + 11 + 3 + 2 + 5 + 2 + 1 + 7 = 48$$

On average 4 lines per problem, down from $16\frac{1}{4}$ in last year's preliminaries[2]

[1] With mostly™ PEP 8-compliant code golfing
[2] But we did way less golfing last year

Thanks to:

The proofreaders

Arnoud van der Leer
Dany Sluijk
Geertje Ulijn
Jaap Eldering
Kevin Verbeek
Michael Zündorf
Pavel Kunyavskiy Kotlin Hero
Tobias Roehr
Thomas Verwoerd
Wendy Yi

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/`