

Freshmen Programming Contests 2026

Solutions presentation

By the Freshmen Programming Contests 2026 jury for:

- AAPJE in Amsterdam
- FPC in Delft
- FPC in Eindhoven
- GAPC in Groningen
- Contest in Mons

April 25, 2026



Please do not post the problems online

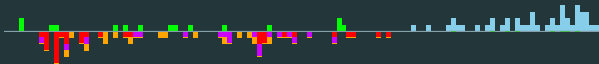
Other universities will have their contests in the coming weeks.

Please, do not post/discuss the problems online before

Saturday 9 May 2026 at 17:00

A: Animal Attire

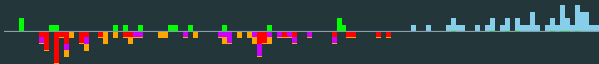
Problem author: Leon van der Waal



- **Problem:** Pick how many items in each of the k categories such that $\prod_{i=1}^k c_i \geq n$ while minimizing the total items

A: Animal Attire

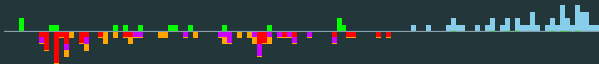
Problem author: Leon van der Waal



- **Problem:** Pick how many items in each of the k categories such that $\prod_{i=1}^k c_i \geq n$ while minimizing the total items
- **Observation:** The c_i 's shouldn't differ by more than one.

A: Animal Attire

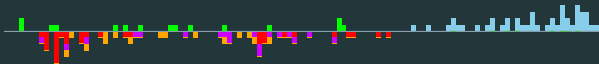
Problem author: Leon van der Waal



- **Problem:** Pick how many items in each of the k categories such that $\prod_{i=1}^k c_i \geq n$ while minimizing the total items
- **Observation:** The c_i 's shouldn't differ by more than one.
 - Otherwise, we can increase one number and decrease another: $3 \times 3 > 2 \times 4$.

A: Animal Attire

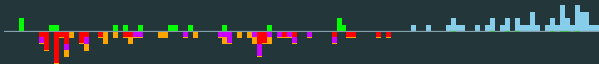
Problem author: Leon van der Waal



- **Problem:** Pick how many items in each of the k categories such that $\prod_{i=1}^k c_i \geq n$ while minimizing the total items
- **Observation:** The c_i 's shouldn't differ by more than one.
 - Otherwise, we can increase one number and decrease another: $3 \times 3 > 2 \times 4$.
- **Naive solution:** Make all c_i 's equal.

A: Animal Attire

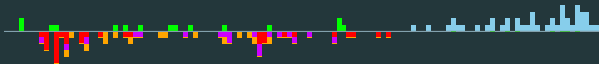
Problem author: Leon van der Waal



- **Problem:** Pick how many items in each of the k categories such that $\prod_{i=1}^k c_i \geq n$ while minimizing the total items
- **Observation:** The c_i 's shouldn't differ by more than one.
 - Otherwise, we can increase one number and decrease another: $3 \times 3 > 2 \times 4$.
- **Naive solution:** Make all c_i 's equal.
 - Solve for $c^k \geq n$.

A: Animal Attire

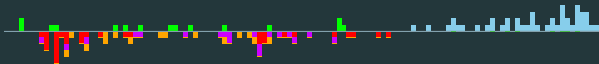
Problem author: Leon van der Waal



- **Problem:** Pick how many items in each of the k categories such that $\prod_{i=1}^k c_i \geq n$ while minimizing the total items
- **Observation:** The c_i 's shouldn't differ by more than one.
 - Otherwise, we can increase one number and decrease another: $3 \times 3 > 2 \times 4$.
- **Naive solution:** Make all c_i 's equal.
 - Solve for $c^k \geq n$.
 - This gives a solution with $k \cdot \lceil n^{1/k} \rceil$ items. We can do better.

A: Animal Attire

Problem author: Leon van der Waal



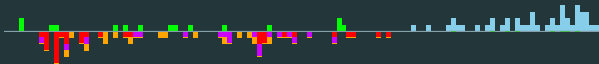
- **Problem:** Pick how many items in each of the k categories such that $\prod_{i=1}^k c_i \geq n$ while minimizing the total items
- **Observation:** The c_i 's shouldn't differ by more than one.
 - Otherwise, we can increase one number and decrease another: $3 \times 3 > 2 \times 4$.
- **Naive solution:** Make all c_i 's equal.
 - Solve for $c^k \geq n$.
 - This gives a solution with $k \cdot \lceil n^{1/k} \rceil$ items. We can do better.
- **Observation:** Not all c_i 's have to be the same.



- **Observation:** Not all c_i 's have to be the same.

A: Animal Attire

Problem author: Leon van der Waal



- **Observation:** Not all c_i 's have to be the same.
- **Solution:** Take x times $\lfloor n^{1/k} \rfloor + 1$ and $k - x$ times $\lfloor n^{1/k} \rfloor$.

A: Animal Attire

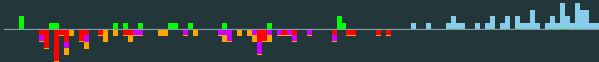
Problem author: Leon van der Waal



- **Observation:** Not all c_i 's have to be the same.
- **Solution:** Take x times $\lfloor n^{1/k} \rfloor + 1$ and $k - x$ times $\lfloor n^{1/k} \rfloor$.
- **Solution:** Loop over x to find smallest possible x for which this works.

A: Animal Attire

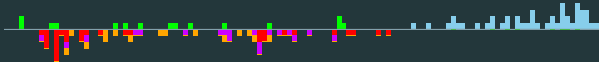
Problem author: Leon van der Waal



- **Observation:** Not all c_i 's have to be the same.
- **Solution:** Take x times $\lfloor n^{1/k} \rfloor + 1$ and $k - x$ times $\lfloor n^{1/k} \rfloor$.
- **Solution:** Loop over x to find smallest possible x for which this works.
- **Complexity:** Final complexity: $\mathcal{O}(k)$. Slower solutions also get AC.

A: Animal Attire

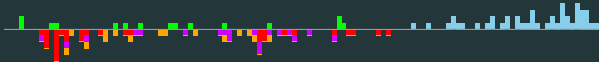
Problem author: Leon van der Waal



- **Observation:** Not all c_i 's have to be the same.
- **Solution:** Take x times $\lfloor n^{1/k} \rfloor + 1$ and $k - x$ times $\lfloor n^{1/k} \rfloor$.
- **Solution:** Loop over x to find smallest possible x for which this works.
- **Complexity:** Final complexity: $\mathcal{O}(k)$. Slower solutions also get AC.
- **Common Pitfalls:** Trying to take the logarithm on both sides but messing up logarithm rules

A: Animal Attire

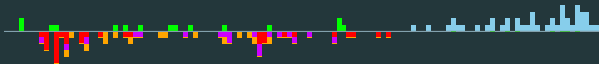
Problem author: Leon van der Waal



- **Observation:** Not all c_i 's have to be the same.
- **Solution:** Take x times $\lfloor n^{1/k} \rfloor + 1$ and $k - x$ times $\lfloor n^{1/k} \rfloor$.
- **Solution:** Loop over x to find smallest possible x for which this works.
- **Complexity:** Final complexity: $\mathcal{O}(k)$. Slower solutions also get AC.
- **Common Pitfalls:** Trying to take the logarithm on both sides but messing up logarithm rules
- $x^k \geq n \nRightarrow x \geq \lfloor \log(n)/\log(k) \rfloor$

A: Animal Attire

Problem author: Leon van der Waal



- **Observation:** Not all c_i 's have to be the same.
- **Solution:** Take x times $\lfloor n^{1/k} \rfloor + 1$ and $k - x$ times $\lfloor n^{1/k} \rfloor$.
- **Solution:** Loop over x to find smallest possible x for which this works.
- **Complexity:** Final complexity: $\mathcal{O}(k)$. Slower solutions also get AC.
- **Common Pitfalls:** Trying to take the logarithm on both sides but messing up logarithm rules
- $x^k \geq n \not\Rightarrow x \geq \lfloor \log(n)/\log(k) \rfloor$

Statistics: 123 submissions, 15 accepted, 45 unknown

B: Building Beaver

Problem author: Jeroen Op de Beek

- **Problem:** Given a number n ($1 \leq n \leq 2 \cdot 10^5$), design an ordering of the numbers $1, 2, \dots, n$ to insert elements into a binary search tree (BST), such that the BST satisfies the AVL property. Find the lexicographically smallest such ordering.

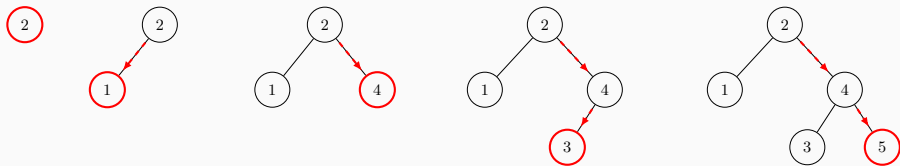


Figure 1: Inserting the values in order (2, 1, 4, 3, 5) in a BST.

B: Building Beaver

Problem author: Jeroen Op de Beek

- **Problem:** Given a number n ($1 \leq n \leq 2 \cdot 10^5$), design an ordering of the numbers $1, 2, \dots, n$ to insert elements into a binary search tree (BST), such that the BST satisfies the AVL property. Find the lexicographically smallest such ordering.

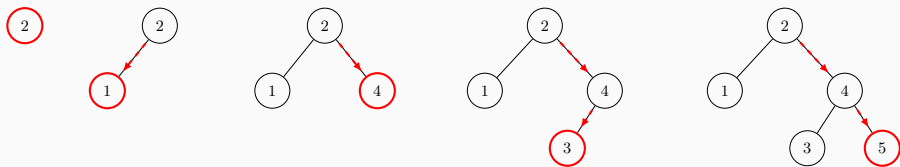


Figure 1: Inserting the values in order (2, 1, 4, 3, 5) in a BST.

- **Fun fact:** AVL rebalancing shows solution always exists. It does not help finding the smallest such ordering.

B: Building Beaver

Problem author: Jeroen Op de Beek

- **Problem:** Given a number n ($1 \leq n \leq 2 \cdot 10^5$), design an ordering of the numbers $1, 2, \dots, n$ to insert elements into a binary search tree (BST), such that the BST satisfies the AVL property. Find the lexicographically smallest such ordering.

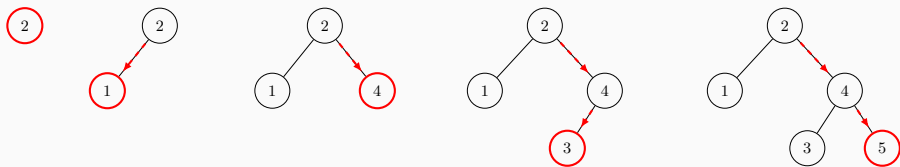


Figure 1: Inserting the values in order (2, 1, 4, 3, 5) in a BST.

- **Fun fact:** AVL rebalancing shows solution always exists. It does not help finding the smallest such ordering.
- **Observation:** The first number p_1 becomes the root of the BST.

B: Building Beaver

Problem author: Jeroen Op de Beek

- **Problem:** Given a number n ($1 \leq n \leq 2 \cdot 10^5$), design an ordering of the numbers $1, 2, \dots, n$ to insert elements into a binary search tree (BST), such that the BST satisfies the AVL property. Find the lexicographically smallest such ordering.

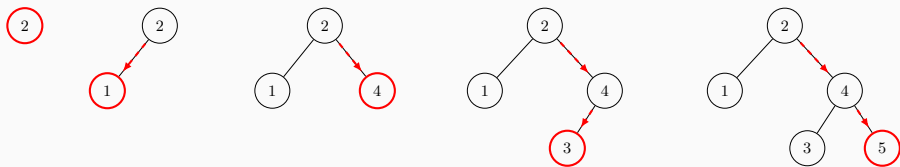


Figure 1: Inserting the values in order (2, 1, 4, 3, 5) in a BST.

- **Fun fact:** AVL rebalancing shows solution always exists. It does not help finding the smallest such ordering.
- **Observation:** The first number p_1 becomes the root of the BST.
- All $p_i < p_1$ form the left subtree. All $p_i > p_1$ form the right subtree.

- **Observation:** Optimal permutation will look like:

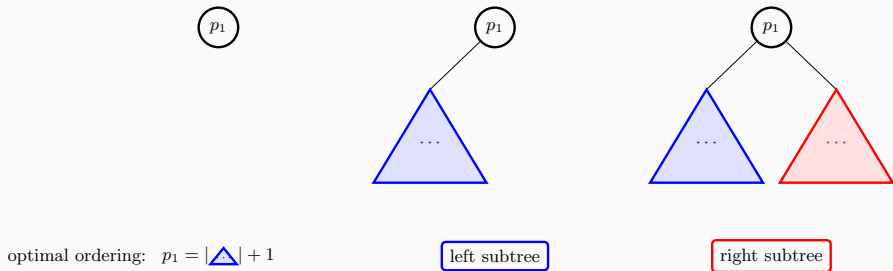


Figure 2: Lexicographically smallest AVL ordering.

- **Observation:** Optimal permutation will look like:

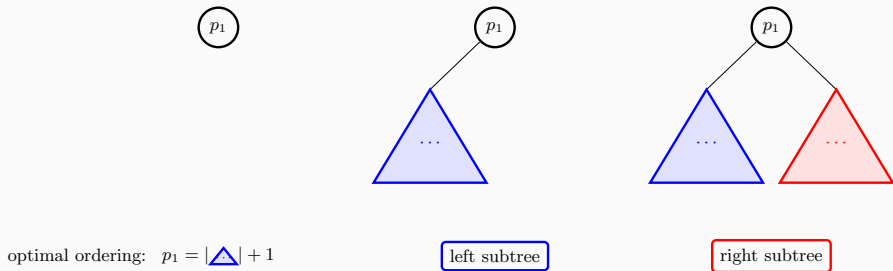


Figure 2: Lexicographically smallest AVL ordering.

- **Observation:** Left and right subtree are independent subproblems, can be recursively solved.

B: Building Beaver

Problem author: Jeroen Op de Beek

- **Observation:** Optimal permutation will look like:

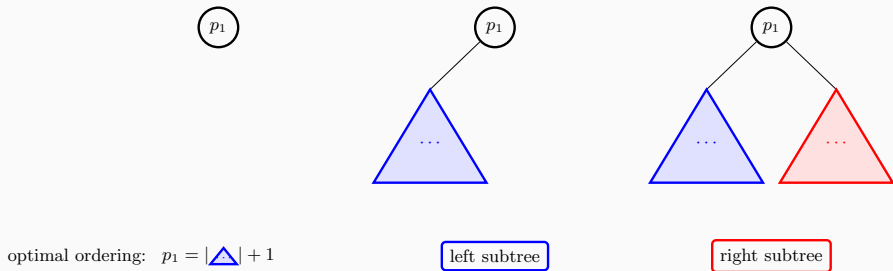


Figure 2: Lexicographically smallest AVL ordering.

- **Observation:** Left and right subtree are independent subproblems, can be recursively solved.
- **Solution:** Make a recursive function $\text{avltree}(n, h_{\text{LO}}, h_{\text{HI}})$

B: Building Beaver

Problem author: Jeroen Op de Beek

- **Observation:** Optimal permutation will look like:

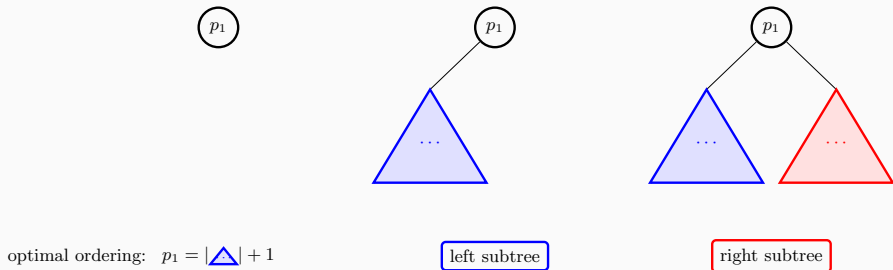


Figure 2: Lexicographically smallest AVL ordering.

- **Observation:** Left and right subtree are independent subproblems, can be recursively solved.
- **Solution:** Make a recursive function $\text{avltree}(n, h_{\text{LO}}, h_{\text{HI}})$
- Will construct lexicographically smallest ordering for an AVL tree with $h_{\text{LO}} \leq \text{height} \leq h_{\text{HI}}$.

B: Building Beaver

Problem author: Jeroen Op de Beek

- **Observation:** Optimal permutation will look like:

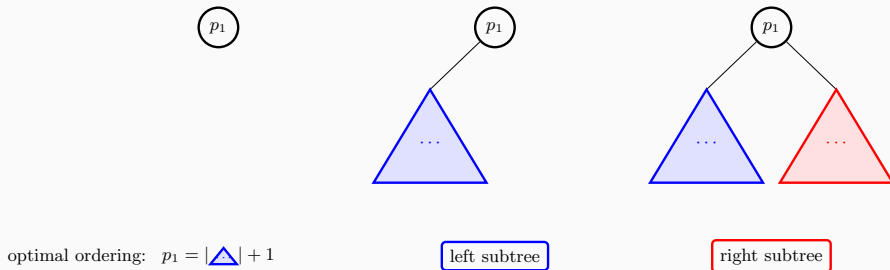


Figure 2: Lexicographically smallest AVL ordering.

- **Observation:** Left and right subtree are independent subproblems, can be recursively solved.
- **Solution:** Make a recursive function $\text{avltree}(n, h_{\text{LO}}, h_{\text{HI}})$
- Will construct lexicographically smallest ordering for an AVL tree with $h_{\text{LO}} \leq \text{height} \leq h_{\text{HI}}$.
- Want to minimize p_1 first, so want to minimize left subtree size

B: Building Beaver

Problem author: Jeroen Op de Beek



- **Observation:** Can calculate minimum and maximum subtree size of AVL BST of height h with recursion:

B: Building Beaver

Problem author: Jeroen Op de Beek



- **Observation:** Can calculate minimum and maximum subtree size of AVL BST of height h with recursion:
- $S_{\min}(h) = S_{\min}(h-2) + S_{\min}(h-1) + 1$, $S_{\min}(0) = 0$

B: Building Beaver

Problem author: Jeroen Op de Beek



- **Observation:** Can calculate minimum and maximum subtree size of AVL BST of height h with recursion:
- $S_{\min}(h) = S_{\min}(h-2) + S_{\min}(h-1) + 1$, $S_{\min}(0) = 0$
- $S_{\max}(h) = 2 \cdot S_{\max}(h-1) + 1$, $S_{\max}(0) = 0$

B: Building Beaver

Problem author: Jeroen Op de Beek



- **Observation:** Can calculate minimum and maximum subtree size of AVL BST of height h with recursion:
- $S_{\min}(h) = S_{\min}(h-2) + S_{\min}(h-1) + 1$, $S_{\min}(0) = 0$
- $S_{\max}(h) = 2 \cdot S_{\max}(h-1) + 1$, $S_{\max}(0) = 0$
- Only need $O(\log(n))$ first values, as both grow exponentially.

B: Building Beaver

Problem author: Jeroen Op de Beek



- **Observation:** Can calculate minimum and maximum subtree size of AVL BST of height h with recursion:
- $S_{\min}(h) = S_{\min}(h-2) + S_{\min}(h-1) + 1$, $S_{\min}(0) = 0$
- $S_{\max}(h) = 2 \cdot S_{\max}(h-1) + 1$, $S_{\max}(0) = 0$
- Only need $O(\log(n))$ first values, as both grow exponentially.
- **Solution (cont.):** To calculate $\text{avltree}(n, h_{\text{LO}}, h_{\text{HI}})$, loop over left and right subtree heights h_L, h_R . Make sure $|h_L - h_R| \leq 1$ (AVL property), and $h_{\text{LO}} \leq 1 + \max(h_L, h_R) \leq h_{\text{HI}}$

B: Building Beaver

Problem author: Jeroen Op de Beek



- **Observation:** Can calculate minimum and maximum subtree size of AVL BST of height h with recursion:
- $S_{\min}(h) = S_{\min}(h-2) + S_{\min}(h-1) + 1$, $S_{\min}(0) = 0$
- $S_{\max}(h) = 2 \cdot S_{\max}(h-1) + 1$, $S_{\max}(0) = 0$
- Only need $O(\log(n))$ first values, as both grow exponentially.
- **Solution (cont.):** To calculate $\text{avltree}(n, h_{\text{LO}}, h_{\text{HI}})$, loop over left and right subtree heights h_L, h_R . Make sure $|h_L - h_R| \leq 1$ (AVL property), and $h_{\text{LO}} \leq 1 + \max(h_L, h_R) \leq h_{\text{HI}}$
- By using $S_{\min}(h)$ and $S_{\max}(h)$, figure out which h_L, h_R pairs give the minimum left subtree size S_L .



- **Observation:** Can calculate minimum and maximum subtree size of AVL BST of height h with recursion:
- $S_{\min}(h) = S_{\min}(h-2) + S_{\min}(h-1) + 1$, $S_{\min}(0) = 0$
- $S_{\max}(h) = 2 \cdot S_{\max}(h-1) + 1$, $S_{\max}(0) = 0$
- Only need $O(\log(n))$ first values, as both grow exponentially.
- **Solution (cont.):** To calculate $\text{avltree}(n, h_{\text{LO}}, h_{\text{HI}})$, loop over left and right subtree heights h_L, h_R . Make sure $|h_L - h_R| \leq 1$ (AVL property), and $h_{\text{LO}} \leq 1 + \max(h_L, h_R) \leq h_{\text{HI}}$
- By using $S_{\min}(h)$ and $S_{\max}(h)$, figure out which h_L, h_R pairs give the minimum left subtree size S_L .
- **Observation:** All h_L 's reaching the minimum left subtree size form an interval $[a, b]$.



- **Observation:** Can calculate minimum and maximum subtree size of AVL BST of height h with recursion:
- $S_{\min}(h) = S_{\min}(h-2) + S_{\min}(h-1) + 1$, $S_{\min}(0) = 0$
- $S_{\max}(h) = 2 \cdot S_{\max}(h-1) + 1$, $S_{\max}(0) = 0$
- Only need $O(\log(n))$ first values, as both grow exponentially.
- **Solution (cont.):** To calculate $\text{avltree}(n, h_{\text{LO}}, h_{\text{HI}})$, loop over left and right subtree heights h_L, h_R . Make sure $|h_L - h_R| \leq 1$ (AVL property), and $h_{\text{LO}} \leq 1 + \max(h_L, h_R) \leq h_{\text{HI}}$
- By using $S_{\min}(h)$ and $S_{\max}(h)$, figure out which h_L, h_R pairs give the minimum left subtree size S_L .
- **Observation:** All h_L 's reaching the minimum left subtree size form an interval $[a, b]$.
- Use $\text{avltree}(S_L, a, b)$ to find optimal left subtree ordering.

B: Building Beaver

Problem author: Jeroen Op de Beek



- **Observation:** Can calculate minimum and maximum subtree size of AVL BST of height h with recursion:
- $S_{\min}(h) = S_{\min}(h-2) + S_{\min}(h-1) + 1$, $S_{\min}(0) = 0$
- $S_{\max}(h) = 2 \cdot S_{\max}(h-1) + 1$, $S_{\max}(0) = 0$
- Only need $O(\log(n))$ first values, as both grow exponentially.
- **Solution (cont.):** To calculate $\text{avltree}(n, h_{\text{LO}}, h_{\text{HI}})$, loop over left and right subtree heights h_L, h_R . Make sure $|h_L - h_R| \leq 1$ (AVL property), and $h_{\text{LO}} \leq 1 + \max(h_L, h_R) \leq h_{\text{HI}}$
- By using $S_{\min}(h)$ and $S_{\max}(h)$, figure out which h_L, h_R pairs give the minimum left subtree size S_L .
- **Observation:** All h_L 's reaching the minimum left subtree size form an interval $[a, b]$.
- Use $\text{avltree}(S_L, a, b)$ to find optimal left subtree ordering.
- Left subtree height h_L is known, so h_R subtree heights will again form interval, call $\text{avltree}(n-1-S_L, u, v)$, with appropriate u, v , and reindex with offset.

B: Building Beaver

Problem author: Jeroen Op de Beek



- **Observation:** Can calculate minimum and maximum subtree size of AVL BST of height h with recursion:
- $S_{\min}(h) = S_{\min}(h-2) + S_{\min}(h-1) + 1$, $S_{\min}(0) = 0$
- $S_{\max}(h) = 2 \cdot S_{\max}(h-1) + 1$, $S_{\max}(0) = 0$
- Only need $O(\log(n))$ first values, as both grow exponentially.
- **Solution (cont.):** To calculate $\text{avltree}(n, h_{\text{LO}}, h_{\text{HI}})$, loop over left and right subtree heights h_L, h_R . Make sure $|h_L - h_R| \leq 1$ (AVL property), and $h_{\text{LO}} \leq 1 + \max(h_L, h_R) \leq h_{\text{HI}}$
- By using $S_{\min}(h)$ and $S_{\max}(h)$, figure out which h_L, h_R pairs give the minimum left subtree size S_L .
- **Observation:** All h_L 's reaching the minimum left subtree size form an interval $[a, b]$.
- Use $\text{avltree}(S_L, a, b)$ to find optimal left subtree ordering.
- Left subtree height h_L is known, so h_R subtree heights will again form interval, call $\text{avltree}(n-1-S_L, u, v)$, with appropriate u, v , and reindex with offset.
- Time complexity: $T(n) = O(\log(n)) + T(S_L) + T(n-1-S_L) = O(n \log(n))$.

B: Building Beaver

Problem author: Jeroen Op de Beek



- **Observation:** Can calculate minimum and maximum subtree size of AVL BST of height h with recursion:
- $S_{\min}(h) = S_{\min}(h-2) + S_{\min}(h-1) + 1$, $S_{\min}(0) = 0$
- $S_{\max}(h) = 2 \cdot S_{\max}(h-1) + 1$, $S_{\max}(0) = 0$
- Only need $O(\log(n))$ first values, as both grow exponentially.
- **Solution (cont.):** To calculate $\text{avltree}(n, h_{\text{LO}}, h_{\text{HI}})$, loop over left and right subtree heights h_L, h_R . Make sure $|h_L - h_R| \leq 1$ (AVL property), and $h_{\text{LO}} \leq 1 + \max(h_L, h_R) \leq h_{\text{HI}}$
- By using $S_{\min}(h)$ and $S_{\max}(h)$, figure out which h_L, h_R pairs give the minimum left subtree size S_L .
- **Observation:** All h_L 's reaching the minimum left subtree size form an interval $[a, b]$.
- Use $\text{avltree}(S_L, a, b)$ to find optimal left subtree ordering.
- Left subtree height h_L is known, so h_R subtree heights will again form interval, call $\text{avltree}(n-1-S_L, u, v)$, with appropriate u, v , and reindex with offset.
- Time complexity: $T(n) = O(\log(n)) + T(S_L) + T(n-1-S_L) = O(n \log(n))$.
- **Bonus:** Can prove because recursion tree is AVL tree, $T(n) = \Theta(n)$.

B: Building Beaver

Problem author: Jeroen Op de Beek

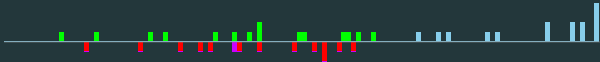


- **Observation:** Can calculate minimum and maximum subtree size of AVL BST of height h with recursion:
- $S_{\min}(h) = S_{\min}(h-2) + S_{\min}(h-1) + 1$, $S_{\min}(0) = 0$
- $S_{\max}(h) = 2 \cdot S_{\max}(h-1) + 1$, $S_{\max}(0) = 0$
- Only need $O(\log(n))$ first values, as both grow exponentially.
- **Solution (cont.):** To calculate $\text{avltree}(n, h_{\text{LO}}, h_{\text{HI}})$, loop over left and right subtree heights h_L, h_R . Make sure $|h_L - h_R| \leq 1$ (AVL property), and $h_{\text{LO}} \leq 1 + \max(h_L, h_R) \leq h_{\text{HI}}$
- By using $S_{\min}(h)$ and $S_{\max}(h)$, figure out which h_L, h_R pairs give the minimum left subtree size S_L .
- **Observation:** All h_L 's reaching the minimum left subtree size form an interval $[a, b]$.
- Use $\text{avltree}(S_L, a, b)$ to find optimal left subtree ordering.
- Left subtree height h_L is known, so h_R subtree heights will again form interval, call $\text{avltree}(n-1-S_L, u, v)$, with appropriate u, v , and reindex with offset.
- Time complexity: $T(n) = O(\log(n)) + T(S_L) + T(n-1-S_L) = O(n \log(n))$.
- **Bonus:** Can prove because recursion tree is AVL tree, $T(n) = \Theta(n)$.

Statistics: 54 submissions, 0 accepted, 40 unknown

C: Coven Complications

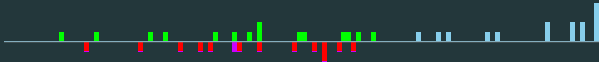
Problem author: Andrei Voicu



- **Problem:** Minimize total witch losses while sealing one community per day.

C: Coven Complications

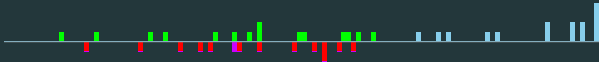
Problem author: Andrei Voicu



- **Problem:** Minimize total witch losses while sealing one community per day.
- Precompute daily risk r_i for each owned community.

C: Coven Complications

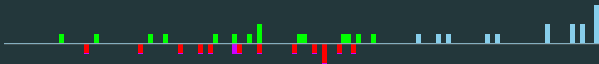
Problem author: Andrei Voicu



- **Problem:** Minimize total witch losses while sealing one community per day.
- Precompute daily risk r_i for each owned community.
- Total loss = \sum_k (remaining total risk after sealing k -th community).

C: Coven Complications

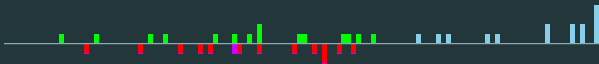
Problem author: Andrei Voicu



- **Problem:** Minimize total witch losses while sealing one community per day.
- Precompute daily risk r_i for each owned community.
- Total loss = \sum_k (remaining total risk after sealing k -th community).
- Greedy: seal highest-risk first. Sort descending, accumulate using suffix sums. $\mathcal{O}(n \log n)$.

C: Coven Complications

Problem author: Andrei Voicu



- **Problem:** Minimize total witch losses while sealing one community per day.
- Precompute daily risk r_i for each owned community.
- Total loss = \sum_k (remaining total risk after sealing k -th community).
- Greedy: seal highest-risk first. Sort descending, accumulate using suffix sums. $\mathcal{O}(n \log n)$.

Statistics: 44 submissions, 15 accepted, 15 unknown

D: Digit Display

Problem author: Moham Balfakeih



- **Problem:** Given n segments, display the largest possible number on 7-segment digits.

D: Digit Display

Problem author: Moham Balfakeih



- **Problem:** Given n segments, display the largest possible number on 7-segment digits.
- If $n < 2$: no digit can be displayed, output `impossible`.

D: Digit Display

Problem author: Moham Balfakeih



- **Problem:** Given n segments, display the largest possible number on 7-segment digits.
- If $n < 2$: no digit can be displayed, output `impossible`.
- **Key insight:** A larger number of digits always beats fewer digits (e.g. $11 > 9$).

D: Digit Display

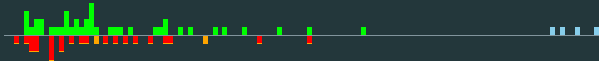
Problem author: Moham Balfakeih



- **Problem:** Given n segments, display the largest possible number on 7-segment digits.
- If $n < 2$: no digit can be displayed, output *impossible*.
- **Key insight:** A larger number of digits always beats fewer digits (e.g. $11 > 9$).
- So we want to *maximize the number of digits* first.

D: Digit Display

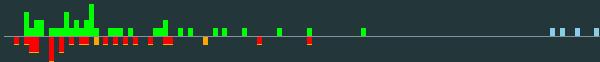
Problem author: Moham Balfakeih



- **Problem:** Given n segments, display the largest possible number on 7-segment digits.
- If $n < 2$: no digit can be displayed, output *impossible*.
- **Key insight:** A larger number of digits always beats fewer digits (e.g. $11 > 9$).
- So we want to *maximize the number of digits* first.
- The cheapest digit is 1 at 2 segments. So $\lfloor n/2 \rfloor$ digits is optimal.

D: Digit Display

Problem author: Moham Balfakeih



- **Problem:** Given n segments, display the largest possible number on 7-segment digits.
- If $n < 2$: no digit can be displayed, output *impossible*.
- **Key insight:** A larger number of digits always beats fewer digits (e.g. $11 > 9$).
- So we want to *maximize the number of digits* first.
- The cheapest digit is 1 at 2 segments. So $\lfloor n/2 \rfloor$ digits is optimal.
- If n is even: all 1s $\Rightarrow 111 \dots 1$ ($n/2$ digits).

D: Digit Display

Problem author: Moham Balfakeih



- **Problem:** Given n segments, display the largest possible number on 7-segment digits.
- If $n < 2$: no digit can be displayed, output impossible.
- **Key insight:** A larger number of digits always beats fewer digits (e.g. $11 > 9$).
- So we want to *maximize the number of digits* first.
- The cheapest digit is 1 at 2 segments. So $\lfloor n/2 \rfloor$ digits is optimal.
- If n is even: all 1s $\Rightarrow 111 \dots 1$ ($n/2$ digits).
- If n is odd: one leftover segment. Replace the leading 1 (2 segments) with 7 (3 segments) $\Rightarrow 711 \dots 1$.

D: Digit Display

Problem author: Moham Balfakeih



- **Problem:** Given n segments, display the largest possible number on 7-segment digits.
- If $n < 2$: no digit can be displayed, output impossible.
- **Key insight:** A larger number of digits always beats fewer digits (e.g. $11 > 9$).
- So we want to *maximize the number of digits* first.
- The cheapest digit is 1 at 2 segments. So $\lfloor n/2 \rfloor$ digits is optimal.
- If n is even: all 1s $\Rightarrow 111 \dots 1$ ($n/2$ digits).
- If n is odd: one leftover segment. Replace the leading 1 (2 segments) with 7 (3 segments) $\Rightarrow 711 \dots 1$.
- This is optimal because 7 is the largest single digit costing exactly 3 segments, and placing it first maximizes the value.

D: Digit Display

Problem author: Moham Balfakeih



- **Problem:** Given n segments, display the largest possible number on 7-segment digits.
- If $n < 2$: no digit can be displayed, output impossible.
- **Key insight:** A larger number of digits always beats fewer digits (e.g. $11 > 9$).
- So we want to *maximize the number of digits* first.
- The cheapest digit is 1 at 2 segments. So $\lfloor n/2 \rfloor$ digits is optimal.
- If n is even: all 1s $\Rightarrow 111 \dots 1$ ($n/2$ digits).
- If n is odd: one leftover segment. Replace the leading 1 (2 segments) with 7 (3 segments) $\Rightarrow 711 \dots 1$.
- This is optimal because 7 is the largest single digit costing exactly 3 segments, and placing it first maximizes the value.
- $\mathcal{O}(n)$ to print the output (or $\mathcal{O}(1)$ to compute what to print).

D: Digit Display

Problem author: Moham Balfakeih

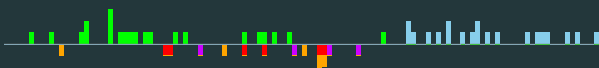


- **Problem:** Given n segments, display the largest possible number on 7-segment digits.
- If $n < 2$: no digit can be displayed, output impossible.
- **Key insight:** A larger number of digits always beats fewer digits (e.g. $11 > 9$).
- So we want to *maximize the number of digits* first.
- The cheapest digit is 1 at 2 segments. So $\lfloor n/2 \rfloor$ digits is optimal.
- If n is even: all 1s $\Rightarrow 111 \dots 1$ ($n/2$ digits).
- If n is odd: one leftover segment. Replace the leading 1 (2 segments) with 7 (3 segments) $\Rightarrow 711 \dots 1$.
- This is optimal because 7 is the largest single digit costing exactly 3 segments, and placing it first maximizes the value.
- $\mathcal{O}(n)$ to print the output (or $\mathcal{O}(1)$ to compute what to print).

Statistics: 70 submissions, 41 accepted, 4 unknown

E: Ernest's Endeavour

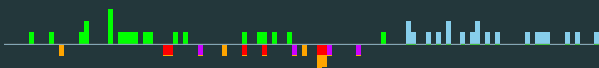
Problem author: David Ghiberdic, Rares Rauta



- **Problem:** Given an undirected graph with n labelled nodes and m edges, check if any two nodes with different labels are connected. Some nodes have no label.

E: Ernest's Endeavour

Problem author: David Ghiberdic, Rares Rauta



- **Problem:** Given an undirected graph with n labelled nodes and m edges, check if any two nodes with different labels are connected. Some nodes have no label.
- **Naive solution:** For each labelled node, traverse the graph until you reach a node with a different label.

E: Ernest's Endeavour

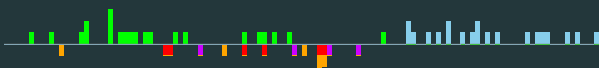
Problem author: David Ghiberdic, Rares Rauta



- **Problem:** Given an undirected graph with n labelled nodes and m edges, check if any two nodes with different labels are connected. Some nodes have no label.
- **Naive solution:** For each labelled node, traverse the graph until you reach a node with a different label.
- **Time complexity:** $\mathcal{O}(n \cdot (n + m))$ – too slow!

E: Ernest's Endeavour

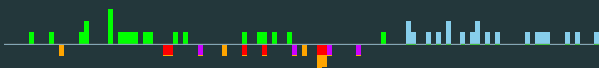
Problem author: David Ghiberdic, Rares Rauta



- **Problem:** Given an undirected graph with n labelled nodes and m edges, check if any two nodes with different labels are connected. Some nodes have no label.
- **Naive solution:** For each labelled node, traverse the graph until you reach a node with a different label.
- **Time complexity:** $\mathcal{O}(n \cdot (n + m))$ – too slow!
- **Faster solution:** For each connected component, traverse it, and remember the labels you find.

E: Ernest's Endeavour

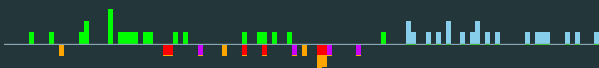
Problem author: David Ghiberdic, Rares Rauta



- **Problem:** Given an undirected graph with n labelled nodes and m edges, check if any two nodes with different labels are connected. Some nodes have no label.
- **Naive solution:** For each labelled node, traverse the graph until you reach a node with a different label.
- **Time complexity:** $\mathcal{O}(n \cdot (n + m))$ – too slow!
- **Faster solution:** For each connected component, traverse it, and remember the labels you find.
- If, while traversing, a node with different label is found, Jack is in danger: output any index of a previously labelled node and the index of the new node.

E: Ernest's Endeavour

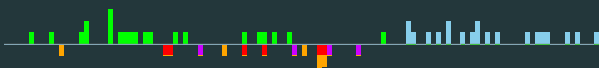
Problem author: David Ghiberdic, Rares Rauta



- **Problem:** Given an undirected graph with n labelled nodes and m edges, check if any two nodes with different labels are connected. Some nodes have no label.
- **Naive solution:** For each labelled node, traverse the graph until you reach a node with a different label.
- **Time complexity:** $\mathcal{O}(n \cdot (n + m))$ – too slow!
- **Faster solution:** For each connected component, traverse it, and remember the labels you find.
- If, while traversing, a node with different label is found, Jack is in danger: output any index of a previously labelled node and the index of the new node.
- If all connected components have been checked and no different labels were found, output “safe”.

E: Ernest's Endeavour

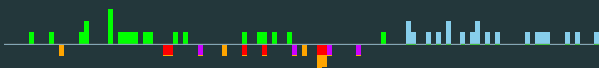
Problem author: David Ghiberdic, Rares Rauta



- **Problem:** Given an undirected graph with n labelled nodes and m edges, check if any two nodes with different labels are connected. Some nodes have no label.
- **Naive solution:** For each labelled node, traverse the graph until you reach a node with a different label.
- **Time complexity:** $\mathcal{O}(n \cdot (n + m))$ – too slow!
- **Faster solution:** For each connected component, traverse it, and remember the labels you find.
- If, while traversing, a node with different label is found, Jack is in danger: output any index of a previously labelled node and the index of the new node.
- If all connected components have been checked and no different labels were found, output “safe”.
- **Time complexity:** $\mathcal{O}(n + m)$.

E: Ernest's Endeavour

Problem author: David Ghiberdic, Rares Rauta



- **Problem:** Given an undirected graph with n labelled nodes and m edges, check if any two nodes with different labels are connected. Some nodes have no label.
- **Naive solution:** For each labelled node, traverse the graph until you reach a node with a different label.
- **Time complexity:** $\mathcal{O}(n \cdot (n + m))$ – too slow!
- **Faster solution:** For each connected component, traverse it, and remember the labels you find.
- If, while traversing, a node with different label is found, Jack is in danger: output any index of a previously labelled node and the index of the new node.
- If all connected components have been checked and no different labels were found, output “safe”.
- **Time complexity:** $\mathcal{O}(n + m)$.

Statistics: 57 submissions, 22 accepted, 20 unknown

F: Ford's Funny Fields

Problem author: David Ghiberdic, Rares Rauta



- **Problem:** Choose a subset of fields such that their total length is at least s , minimizing the total cost.

F: Ford's Funny Fields

Problem author: David Ghiberdic, Rares Rauta



- **Problem:** Choose a subset of fields such that their total length is at least s , minimizing the total cost.
- This is a variation of the classic 0/1 Knapsack Problem.

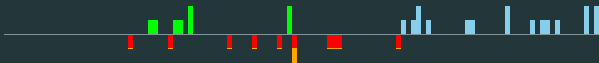
Bar chart showing the distribution of the number of children per family in 1990. The x-axis represents the number of children (0 to 6), and the y-axis represents the percentage of families. The distribution is skewed to the right, with a peak at 1 child (approximately 35%).

Problem author: David Ghiberdic, Rares Rauta

- **Problem:** Choose a subset of fields such that their total length is at least s , minimizing the total cost.
- This is a variation of the classic 0/1 Knapsack Problem.
- Iterate over all individual fields. Let $DP[j]$ be the minimum cost to reach a covered length j . We cap j at s .

F: Ford's Funny Fields

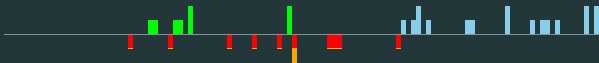
Problem author: David Ghiberdic, Rares Rauta



- **Problem:** Choose a subset of fields such that their total length is at least s , minimizing the total cost.
- This is a variation of the classic 0/1 Knapsack Problem.
- Iterate over all individual fields. Let $DP[j]$ be the minimum cost to reach a covered length j . We cap j at s .
- Since there are at most $n \cdot x \leq 2000$ fields and $s \leq 2000$, DP updates take $\mathcal{O}(\sum x \cdot s)$ time, which securely passes bounds.

F: Ford's Funny Fields

Problem author: David Ghiberdic, Rares Rauta



- **Problem:** Choose a subset of fields such that their total length is at least s , minimizing the total cost.
- This is a variation of the classic 0/1 Knapsack Problem.
- Iterate over all individual fields. Let $DP[j]$ be the minimum cost to reach a covered length j . We cap j at s .
- Since there are at most $n \cdot x \leq 2000$ fields and $s \leq 2000$, DP updates take $\mathcal{O}(\sum x \cdot s)$ time, which securely passes bounds.

Statistics: 36 submissions, 8 accepted, 17 unknown

Problem author: Arnoud van der Leer

Problem author: Arnoud van der Leer

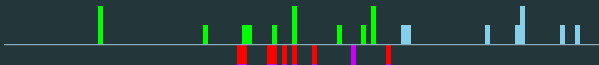
- **Problem:** Find first amount of people that cannot be written as the sum of some numbers between x and y .
- **Observation:** If we have k groups/numbers, anything between kx and ky can be formed.

Problem author: Arnoud van der Leer

- **Problem:** Find first amount of people that cannot be written as the sum of some numbers between x and y .
- **Observation:** If we have k groups/numbers, anything between kx and ky can be formed.
- **Observation:** If n can not be written, it must be larger than ky and smaller than $(k + 1)x$.

G: Game Night Groups

Problem author: Arnoud van der Leer

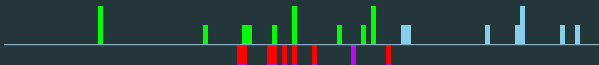


- **Problem:** Find first amount of people that cannot be written as the sum of some numbers between x and y .
- **Observation:** If we have k groups/numbers, anything between kx and ky can be formed.
- **Observation:** If n can not be written, it must be larger than ky and smaller than $(k+1)x$.
- **Solution:** Thus, $ky + 1 < (k+1)x$. Solve this equation:

$$k(y - x) < x - 1 \implies k < \frac{x - 1}{y - x}.$$

G: Game Night Groups

Problem author: Arnoud van der Leer



- **Problem:** Find first amount of people that cannot be written as the sum of some numbers between x and y .
- **Observation:** If we have k groups/numbers, anything between kx and ky can be formed.
- **Observation:** If n can not be written, it must be larger than ky and smaller than $(k+1)x$.
- **Solution:** Thus, $ky + 1 < (k+1)x$. Solve this equation:

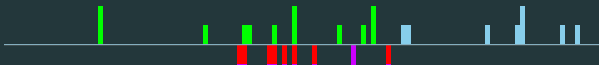
$$k(y - x) < x - 1 \implies k < \frac{x - 1}{y - x}.$$

- **Solution:** Therefore, the highest possible k is

$$k = \left\lfloor \frac{x - 1}{y - x} \right\rfloor.$$

G: Game Night Groups

Problem author: Arnoud van der Leer



- **Problem:** Find first amount of people that cannot be written as the sum of some numbers between x and y .
- **Observation:** If we have k groups/numbers, anything between kx and ky can be formed.
- **Observation:** If n can not be written, it must be larger than ky and smaller than $(k+1)x$.
- **Solution:** Thus, $ky + 1 < (k+1)x$. Solve this equation:

$$k(y - x) < x - 1 \implies k < \frac{x - 1}{y - x}.$$

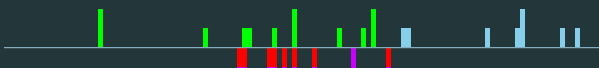
- **Solution:** Therefore, the highest possible k is

$$k = \left\lfloor \frac{x - 1}{y - x} \right\rfloor.$$

- **Solution:** The answer now is $(k+1)x - 1$, since this is the last value in between ky and $(k+1)x$.

G: Game Night Groups

Problem author: Arnoud van der Leer



- **Problem:** Find first amount of people that cannot be written as the sum of some numbers between x and y .
- **Observation:** If we have k groups/numbers, anything between kx and ky can be formed.
- **Observation:** If n can not be written, it must be larger than ky and smaller than $(k+1)x$.
- **Solution:** Thus, $ky + 1 < (k+1)x$. Solve this equation:

$$k(y - x) < x - 1 \implies k < \frac{x - 1}{y - x}.$$

- **Solution:** Therefore, the highest possible k is

$$k = \left\lfloor \frac{x - 1}{y - x} \right\rfloor.$$

- **Solution:** The answer now is $(k+1)x - 1$, since this is the last value in between ky and $(k+1)x$.

Statistics: 29 submissions, 12 accepted, 8 unknown

H: Hasty Hiker

Problem author: Jeroen Op de Beek



- **Problem:** Count the number of beautiful trails.

H: Hasty Hiker

Problem author: Jeroen Op de Beek



- **Problem:** Count the number of beautiful trails.
- **Naive solution:** Loop over all 4-tuples and check if they are beautiful trails.

H: Hasty Hiker

Problem author: Jeroen Op de Beek



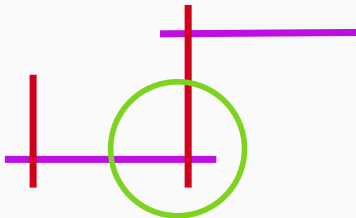
- **Problem:** Count the number of beautiful trails.
- **Naive solution:** Loop over all 4-tuples and check if they are beautiful trails.
- **Complexity:** $\mathcal{O}(n^4)$, too slow!

H: Hasty Hiker

Problem author: Jeroen Op de Beek



- **Problem:** Count the number of beautiful trails.



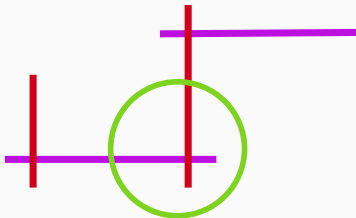
- **Faster solution:**
 - Let's loop over the two roads in the middle of the trail.

H: Hasty Hiker

Problem author: Jeroen Op de Beek



- **Problem:** Count the number of beautiful trails.



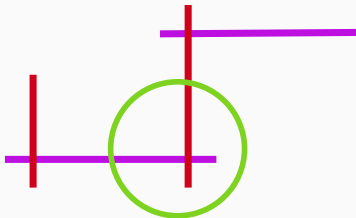
- **Faster solution:**
 - Let's loop over the two roads in the middle of the trail.
 - First check whether they intersect.

H: Hasty Hiker

Problem author: Jeroen Op de Beek



- **Problem:** Count the number of beautiful trails.



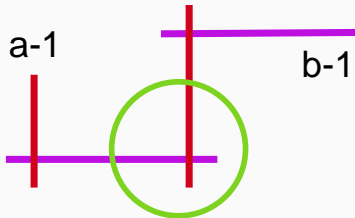
- **Faster solution:**
 - Let's loop over the two roads in the middle of the trail.
 - First check whether they intersect.
 - Count how many roads intersect these two roads. Let's call these counts a and b , respectively.

H: Hasty Hiker

Problem author: Jeroen Op de Beek



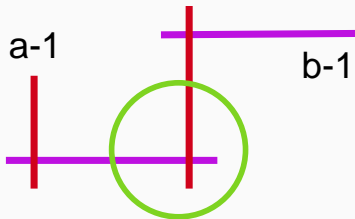
- **Problem:** Count the number of beautiful trails.



- **Faster solution:**
 - Let's loop over the two roads in the middle of the trail.
 - First check whether they intersect.
 - Count how many roads intersect these two roads. Let's call these counts a and b , respectively.
 - Then add $(a - 1) \times (b - 1)$ to the answer.



- **Problem:** Count the number of beautiful trails.



- **Faster solution:**
 - Let's loop over the two roads in the middle of the trail.
 - First check whether they intersect.
 - Count how many roads intersect these two roads. Let's call these counts a and b , respectively.
 - Then add $(a - 1) \times (b - 1)$ to the answer.
- **Complexity:** $\mathcal{O}(n)$ per intersection, so $\mathcal{O}(n^3)$ in total. Still too slow.

H: Hasty Hiker

Problem author: Jeroen Op de Beek



- **Problem:** Count the number of beautiful trails.
- **Optimization:** Precompute the number of other roads each road intersects.

H: Hasty Hiker

Problem author: Jeroen Op de Beek



- **Problem:** Count the number of beautiful trails.
- **Optimization:** Precompute the number of other roads each road intersects.
- **Complexity:** $\mathcal{O}(n^2)$, fast enough!

H: Hasty Hiker

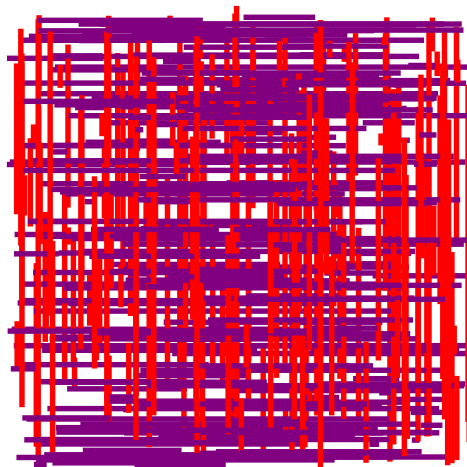
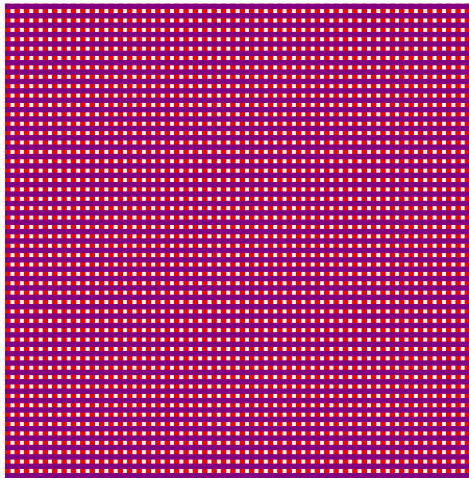
Problem author: Jeroen Op de Beek



- **Problem:** Count the number of beautiful trails.
- **Optimization:** Precompute the number of other roads each road intersects.
- **Complexity:** $\mathcal{O}(n^2)$, fast enough!
- **Bonus:** It is possible to solve this problem in $\mathcal{O}(n \log(n))$ using Fenwick tree + sweepline.

H: Hasty Hiker

Problem author: Jeroen Op de Beek



Statistics: 12 submissions, 4 accepted, 7 unknown

I: Intricate Idioms

Problem author: Jeroen Op de Beek



- **Problem:** Check how many changes a string needs to go through to respect three properties.

I: Intricate Idioms

Problem author: Jeroen Op de Beek



- **Problem:** Check how many changes a string needs to go through to respect three properties.
- We first need to initialize a counter, along with an array keeping track of the number of 'B's in each word, as well as the maximum number of 'B's.
 1. The leftmost character of each string is a 'B'.
 2. In each string, there can be no two 'R' characters adjacent to one another.

I: Intricate Idioms

Problem author: Jeroen Op de Beek



- **Problem:** Check how many changes a string needs to go through to respect three properties.
- We first need to initialize a counter, along with an array keeping track of the number of 'B's in each word, as well as the maximum number of 'B's.
 1. The leftmost character of each string is a 'B'.
 2. In each string, there can be no two 'R' characters adjacent to one another.
- For the first two properties, we will go through of each word. We first check if character is a 'B'.

I: Intricate Idioms

Problem author: Jeroen Op de Beek



- **Problem:** Check how many changes a string needs to go through to respect three properties.
- We first need to initialize a counter, along with an array keeping track of the number of 'B's in each word, as well as the maximum number of 'B's'.
 1. The leftmost character of each string is a 'B'.
 2. In each string, there can be no two 'R' characters adjacent to one another.
- For the first two properties, we will go through of each word. We first check if character is a 'B'.
- After the first check, we go letter by letter, keeping track of the last 'B' instance, which starts at 0.

I: Intricate Idioms

Problem author: Jeroen Op de Beek



- **Problem:** Check how many changes a string needs to go through to respect three properties.
- We first need to initialize a counter, along with an array keeping track of the number of 'B's in each word, as well as the maximum number of 'B's'.
 1. The leftmost character of each string is a 'B'.
 2. In each string, there can be no two 'R' characters adjacent to one another.
- For the first two properties, we will go through of each word. We first check if character is a 'B'.
- After the first check, we go letter by letter, keeping track of the last 'B' instance, which starts at 0.
- When we encounter a 'R' character and the last 'B' is not right behind it, we increment the counter and update the last 'B' to the current location

I: Intricate Idioms

Problem author: Jeroen Op de Beek



- **Problem:** Check how many changes a string needs to go through to respect three properties.
- We first need to initialize a counter, along with an array keeping track of the number of 'B's in each word, as well as the maximum number of 'B's'.
 1. The leftmost character of each string is a 'B'.
 2. In each string, there can be no two 'R' characters adjacent to one another.
- For the first two properties, we will go through of each word. We first check if character is a 'B'.
- After the first check, we go letter by letter, keeping track of the last 'B' instance, which starts at 0.
- When we encounter a 'R' character and the last 'B' is not right behind it, we increment the counter and update the last 'B' to the current location
- Beware of repeated expensive string operations (e.g., substring) to avoid $\mathcal{O}(n^2)$ time complexity.
 3. The number of 'B' characters in each string needs to be the same as the number of 'B' characters in every other string.

I: Intricate Idioms

Problem author: Jeroen Op de Beek



- **Problem:** Check how many changes a string needs to go through to respect three properties.
- We first need to initialize a counter, along with an array keeping track of the number of 'B's in each word, as well as the maximum number of 'B's.
 1. The leftmost character of each string is a 'B'.
 2. In each string, there can be no two 'R' characters adjacent to one another.
- For the first two properties, we will go through of each word. We first check if character is a 'B'.
- After the first check, we go letter by letter, keeping track of the last 'B' instance, which starts at 0.
- When we encounter a 'R' character and the last 'B' is not right behind it, we increment the counter and update the last 'B' to the current location
- Beware of repeated expensive string operations (e.g., substring) to avoid $\mathcal{O}(n^2)$ time complexity.
 3. The number of 'B' characters in each string needs to be the same as the number of 'B' characters in every other string.
- For the final requirement, for each word in the list: If the length of the word is smaller than the amount of 'B's, then it is impossible.

I: Intricate Idioms

Problem author: Jeroen Op de Beek



- **Problem:** Check how many changes a string needs to go through to respect three properties.
- We first need to initialize a counter, along with an array keeping track of the number of 'B's in each word, as well as the maximum number of 'B's'.
 1. The leftmost character of each string is a 'B'.
 2. In each string, there can be no two 'R' characters adjacent to one another.
- For the first two properties, we will go through of each word. We first check if character is a 'B'.
- After the first check, we go letter by letter, keeping track of the last 'B' instance, which starts at 0.
- When we encounter a 'R' character and the last 'B' is not right behind it, we increment the counter and update the last 'B' to the current location
- Beware of repeated expensive string operations (e.g., substring) to avoid $\mathcal{O}(n^2)$ time complexity.
 3. The number of 'B' characters in each string needs to be the same as the number of 'B' characters in every other string.
- For the final requirement, for each word in the list: If the length of the word is smaller than the amount of 'B's, then it is impossible.
- Otherwise, we increment the counter with the difference of the max 'B's minus the 'B's in the word.

I: Intricate Idioms

Problem author: Jeroen Op de Beek



- **Problem:** Check how many changes a string needs to go through to respect three properties.
- We first need to initialize a counter, along with an array keeping track of the number of 'B's in each word, as well as the maximum number of 'B's.
 1. The leftmost character of each string is a 'B'.
 2. In each string, there can be no two 'R' characters adjacent to one another.
- For the first two properties, we will go through of each word. We first check if character is a 'B'.
- After the first check, we go letter by letter, keeping track of the last 'B' instance, which starts at 0.
- When we encounter a 'R' character and the last 'B' is not right behind it, we increment the counter and update the last 'B' to the current location
- Beware of repeated expensive string operations (e.g., substring) to avoid $\mathcal{O}(n^2)$ time complexity.
 3. The number of 'B' characters in each string needs to be the same as the number of 'B' characters in every other string.
- For the final requirement, for each word in the list: If the length of the word is smaller than the amount of 'B's, then it is impossible.
- Otherwise, we increment the counter with the difference of the max 'B's minus the 'B's in the word.
- Lastly, return the counter. The overall complexity should be around $\mathcal{O}(n)$

I: Intricate Idioms

Problem author: Jeroen Op de Beek

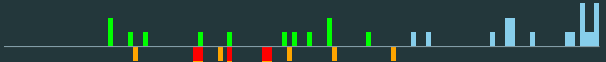


- **Problem:** Check how many changes a string needs to go through to respect three properties.
- We first need to initialize a counter, along with an array keeping track of the number of 'B's in each word, as well as the maximum number of 'B's.
 1. The leftmost character of each string is a 'B'.
 2. In each string, there can be no two 'R' characters adjacent to one another.
- For the first two properties, we will go through of each word. We first check if character is a 'B'.
- After the first check, we go letter by letter, keeping track of the last 'B' instance, which starts at 0.
- When we encounter a 'R' character and the last 'B' is not right behind it, we increment the counter and update the last 'B' to the current location
- Beware of repeated expensive string operations (e.g., substring) to avoid $\mathcal{O}(n^2)$ time complexity.
 3. The number of 'B' characters in each string needs to be the same as the number of 'B' characters in every other string.
- For the final requirement, for each word in the list: If the length of the word is smaller than the amount of 'B's, then it is impossible.
- Otherwise, we increment the counter with the difference of the max 'B's minus the 'B's in the word.
- Lastly, return the counter. The overall complexity should be around $\mathcal{O}(n)$

Statistics: 30 submissions, 13 accepted, 14 unknown

J: Joker Juggling

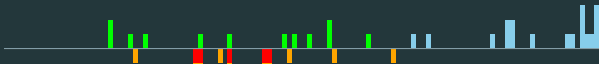
Problem author: Jeroen Op de Beek



- **Problem:** Check whether a pattern containing letters and jokers ('*') matches a word.

J: Joker Juggling

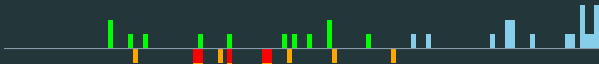
Problem author: Jeroen Op de Beek



- **Problem:** Check whether a pattern containing letters and jokers ('*') matches a word.
- First check: can every joker represent the same number of letters?
 - For example, "a**" can never match "abcd" because a joker cannot represent $1\frac{1}{2}$ letters.

J: Joker Juggling

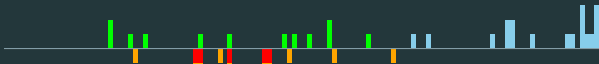
Problem author: Jeroen Op de Beek



- **Problem:** Check whether a pattern containing letters and jokers ('*') matches a word.
- First check: can every joker represent the same number of letters?
 - For example, "a**" can never match "abcd" because a joker cannot represent $1\frac{1}{2}$ letters.
- Knowing how many letters each joker represents, eagerly match the first joker.
 - For example, for "ba**" matching "banana", each joker represents 2 letters, so this must be "na".

J: Joker Juggling

Problem author: Jeroen Op de Beek



- **Problem:** Check whether a pattern containing letters and jokers ('*') matches a word.
- First check: can every joker represent the same number of letters?
 - For example, "a**" can never match "abcd" because a joker cannot represent $1\frac{1}{2}$ letters.
- Knowing how many letters each joker represents, eagerly match the first joker.
 - For example, for "ba**" matching "banana", each joker represents 2 letters, so this must be "na".
- Replace every joker with its replacement and compare it with the given word.
 - Beware of repeated string concatenation, avoid $\mathcal{O}(n^2)$ running time.

J: Joker Juggling

Problem author: Jeroen Op de Beek



- **Problem:** Check whether a pattern containing letters and jokers ('*') matches a word.
- First check: can every joker represent the same number of letters?
 - For example, "a**" can never match "abcd" because a joker cannot represent $1\frac{1}{2}$ letters.
- Knowing how many letters each joker represents, eagerly match the first joker.
 - For example, for "ba**" matching "banana", each joker represents 2 letters, so this must be "na".
- Replace every joker with its replacement and compare it with the given word.
 - Beware of repeated string concatenation, avoid $\mathcal{O}(n^2)$ running time.

Statistics: 40 submissions, 12 accepted, 18 unknown

K: Kracked Enkoder

Problem author: Leon van der Waal



- **Multi-pass Problem:** Encode an integer $1 \leq x \leq 2^{30} - 2$ into 29 bits, and decode it back.

K: Kracked Enkoder

Problem author: Leon van der Waal



- **Multi-pass Problem:** Encode an integer $1 \leq x \leq 2^{30} - 2$ into 29 bits, and decode it back.
- Normally, you need 30 bits to encode integers up to 2^{30} .

K: Kracked Enkoder

Problem author: Leon van der Waal



- **Multi-pass Problem:** Encode an integer $1 \leq x \leq 2^{30} - 2$ into 29 bits, and decode it back.
- Normally, you need 30 bits to encode integers up to 2^{30} .
- **Observation:** We can choose the length of the string we send.

K: Kracked Enkoder

Problem author: Leon van der Waal



- **Multi-pass Problem:** Encode an integer $1 \leq x \leq 2^{30} - 2$ into 29 bits, and decode it back.
- Normally, you need 30 bits to encode integers up to 2^{30} .
- **Observation:** We can choose the length of the string we send.
- Therefore, the total number of strings we can send is

$$2 + 4 + 8 + \dots + 2^{29} = 2^{30} - 2.$$

K: Kracked Enkoder

Problem author: Leon van der Waal



- **Multi-pass Problem:** Encode an integer $1 \leq x \leq 2^{30} - 2$ into 29 bits, and decode it back.
- Normally, you need 30 bits to encode integers up to 2^{30} .
- **Observation:** We can choose the length of the string we send.
- Therefore, the total number of strings we can send is

$$2 + 4 + 8 + \dots + 2^{29} = 2^{30} - 2.$$

- **Solution:** Send $n + 1$ in binary, but remove the leading 1. That way you send 1 less character.

K: Kracked Enkoder

Problem author: Leon van der Waal



- **Multi-pass Problem:** Encode an integer $1 \leq x \leq 2^{30} - 2$ into 29 bits, and decode it back.
- Normally, you need 30 bits to encode integers up to 2^{30} .
- **Observation:** We can choose the length of the string we send.
- Therefore, the total number of strings we can send is

$$2 + 4 + 8 + \dots + 2^{29} = 2^{30} - 2.$$

- **Solution:** Send $n + 1$ in binary, but remove the leading 1. That way you send 1 less character.
 - When decoding, add back the leading 1.

K: Kracked Enkoder

Problem author: Leon van der Waal



- **Multi-pass Problem:** Encode an integer $1 \leq x \leq 2^{30} - 2$ into 29 bits, and decode it back.
- Normally, you need 30 bits to encode integers up to 2^{30} .
- **Observation:** We can choose the length of the string we send.
- Therefore, the total number of strings we can send is

$$2 + 4 + 8 + \dots + 2^{29} = 2^{30} - 2.$$

- **Solution:** Send $n + 1$ in binary, but remove the leading 1. That way you send 1 less character.
 - When decoding, add back the leading 1.
- **Solution:** There are many other solutions.

K: Cracked Encoder

Problem author: Leon van der Waal



- **Multi-pass Problem:** Encode an integer $1 \leq x \leq 2^{30} - 2$ into 29 bits, and decode it back.
- Normally, you need 30 bits to encode integers up to 2^{30} .
- **Observation:** We can choose the length of the string we send.
- Therefore, the total number of strings we can send is

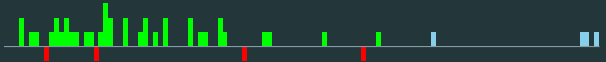
$$2 + 4 + 8 + \dots + 2^{29} = 2^{30} - 2.$$

- **Solution:** Send $n + 1$ in binary, but remove the leading 1. That way you send 1 less character.
 - When decoding, add back the leading 1.
- **Solution:** There are many other solutions.

Statistics: 44 submissions, 9 accepted, 26 unknown

L: Labyrinth Obstacle Layout

Problem author: Jeroen Op de Beek



- **Problem:** Build a grid with exactly two paths from top-left to bottom-right.

L: Labyrinth Obstacle Layout

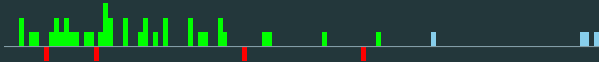
Problem author: Jeroen Op de Beek



- **Problem:** Build a grid with exactly two paths from top-left to bottom-right.
- From the top-left, you have two options: walk to the right and to the bottom.

L: Labyrinth Obstacle Layout

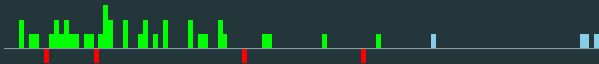
Problem author: Jeroen Op de Beek



- **Problem:** Build a grid with exactly two paths from top-left to bottom-right.
- From the top-left, you have two options: walk to the right and to the bottom.
- **Simplest solution:** One path walks all the way to the top-right and then to the bottom-right, the other path all the way to the bottom-left and then to the bottom-right.

L: Labyrinth Obstacle Layout

Problem author: Jeroen Op de Beek



- **Problem:** Build a grid with exactly two paths from top-left to bottom-right.
- From the top-left, you have two options: walk to the right and to the bottom.
- **Simplest solution:** One path walks all the way to the top-right and then to the bottom-right, the other path all the way to the bottom-left and then to the bottom-right.
- Of course, there are many other solutions possible.

L: Labyrinth Obstacle Layout

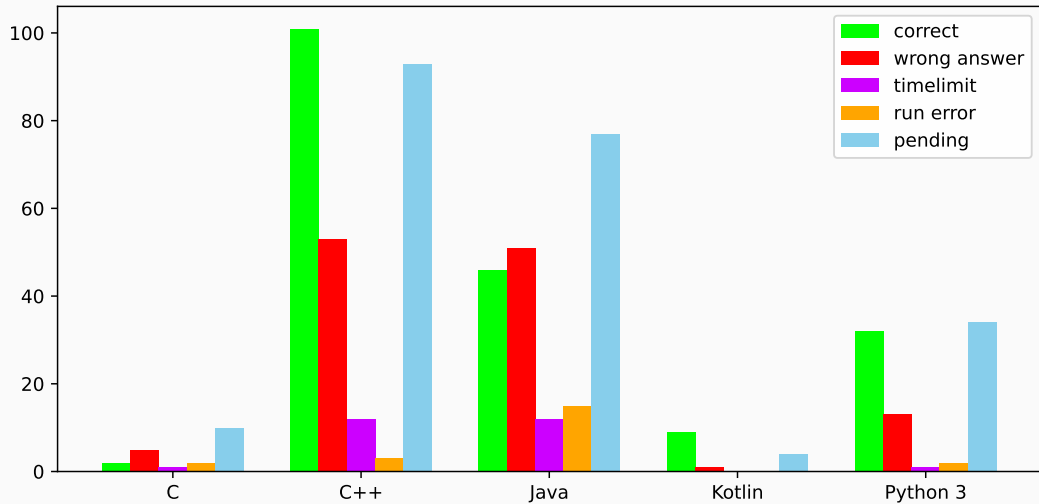
Problem author: Jeroen Op de Beek



- **Problem:** Build a grid with exactly two paths from top-left to bottom-right.
- From the top-left, you have two options: walk to the right and to the bottom.
- **Simplest solution:** One path walks all the way to the top-right and then to the bottom-right, the other path all the way to the bottom-left and then to the bottom-right.
- Of course, there are many other solutions possible.

Statistics: 47 submissions, 39 accepted, 4 unknown

Language stats



Jury work

- 387 commits (last year: 426)

Random facts

Jury work

- 387 commits (last year: 426)
- 988 secret test cases (last year: 625)

Random facts

Jury work

- 387 commits (last year: 426)
- 988 secret test cases (last year: 625)
- 237 jury/proofreader solutions (last year: 160)

Random facts

Jury work

- 387 commits (last year: 426)
- 988 secret test cases (last year: 625)
- 237 jury/proofreader solutions (last year: 160)
- The minimum¹ number of lines the jury needed to solve all problems is

$$2 + 8 + 4 + 1 + 8 + 5 + 1 + 6 + 7 + 2 + 1 + 1 = 46$$

On average $3\frac{5}{6}$ lines per problem, up from 3.5 last year

¹After code golfing, (mostly) PEP 8 compliant

Thanks to the proofreaders:

- Arnoud van der Leer (Delft / Leiden)
- Bartjan Henkemans (TU Eindhoven)
- Boaz Pat-El (Delft)
- Davina van Meer (Delft)
- Christophe Grandmont (UMONS, 📍)
- Kévin Dubrulle (UMONS, 📍)
- Moham Balfakeih (TU Delft)
- Thomas Verwoerd (TU Delft, 📍 Kotlin Hero 📍)
- Thore Husfeldt (ITU Copenhagen)
- Wietze Koops (Lund Uni.../...versity of Copenhagen, 📍)



= solved most of the problems

Thanks to the Jury for the Freshmen Programming Contests:

- Andrei Voicu (TU Delft)
- David Ghiberdic (TU Eindhoven)
- Gianmaria Piergianni (TU Delft)
- Jeroen Op de Beek (TU Delft)
- Leon van der Waal (TU Delft)
- Liudas Staniulis (VU Amsterdam)
- Maarten Sijm (TU Delft)
- Mattia Marziali (RU Groningen)
- Rares Rauta (TU Eindhoven)



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

<https://fpcs2026.bapc.eu/>

Saturday 9 May 2026 13:00–17:00

Please, do not post/discuss the problems online before this time!

Excited to participate in the next contest?

Register for the BAPC Preliminaries in September!

Want to organize these contests?

Join the organizing committee!

Want to create programming problems for FPCs next year?

Either join the committee, or contact Maarten Sijm