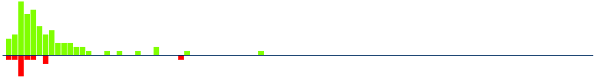
# BAPC 2021

## Solutions presentation

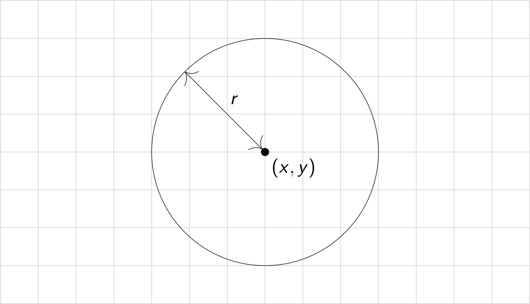
October 31, 2021

# A: Arm Coordination

Problem Author: Reinier Schmiermann



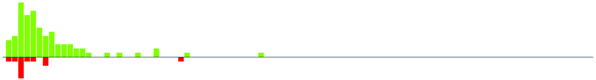
- **Problem:** Given a circle, find the smallest square which encloses this circle.



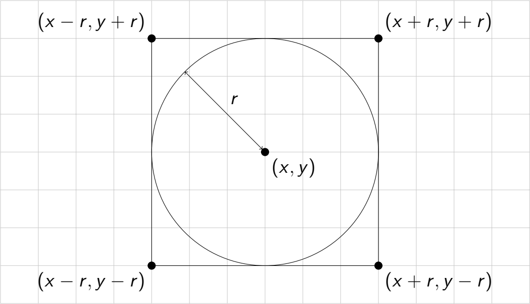

Statistics: 94 submissions, 82 accepted, 0 unknown

# A: Arm Coordination

Problem Author: Reinier Schmiermann



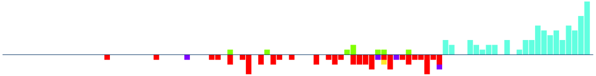
- **Problem:** Given a circle, find the smallest square which encloses this circle.
- **Solution:** simple arithmetic




Statistics: 94 submissions, 82 accepted, 0 unknown

# B: BnPC

Problem Author: Harry Smit

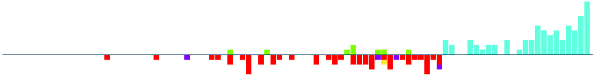


- **Problem:** increase attribute scores so that you maximize a certain score function.

Statistics: 139 submissions, 8 accepted, 78 unknown

# B: BnPC

Problem Author: Harry Smit

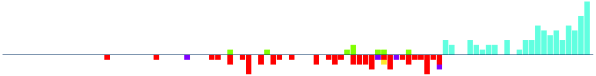


- **Problem:** increase attribute scores so that you maximize a certain score function.
- First set all attributes to the lowest value they need to be to pass all the challenges (if this is impossible, the maximum score is 0).

Statistics: 139 submissions, 8 accepted, 78 unknown

# B: BnPC

Problem Author: Harry Smit

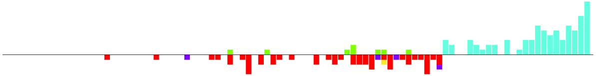


- **Problem:** increase attribute scores so that you maximize a certain score function.
- First set all attributes to the lowest value they need to be to pass all the challenges (if this is impossible, the maximum score is 0).
- Improve your score by increasing an attribute by one. There are two cases:
  - If the attribute score equals *a* of the challenge requirements, you get points equal to *a* times the new attribute score, plus the number of events that require a lower score for that attribute.
  - Otherwise, spending a point here gives additional score equal to the number of events that use this attribute.

Statistics: 139 submissions, 8 accepted, 78 unknown

# B: BnPC

Problem Author: Harry Smit

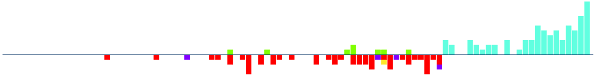


- **Problem:** increase attribute scores so that you maximize a certain score function.
- First set all attributes to the lowest value they need to be to pass all the challenges (if this is impossible, the maximum score is 0).
- Improve your score by increasing an attribute by one. There are two cases:
  - If the attribute score equals *a* of the challenge requirements, you get points equal to *a* times the new attribute score, plus the number of events that require a lower score for that attribute.
  - Otherwise, spending a point here gives additional score equal to the number of events that use this attribute.
- You can only spend one point on the first case (per attribute).

Statistics: 139 submissions, 8 accepted, 78 unknown

# B: BnPC

Problem Author: Harry Smit

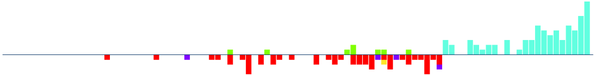


- **Problem:** increase attribute scores so that you maximize a certain score function.
- First set all attributes to the lowest value they need to be to pass all the challenges (if this is impossible, the maximum score is 0).
- Improve your score by increasing an attribute by one. There are two cases:
  - If the attribute score equals $a$ of the challenge requirements, you get points equal to $a$ times the new attribute score, plus the number of events that require a lower score for that attribute.
  - Otherwise, spending a point here gives additional score equal to the number of events that use this attribute.
- You can only spend one point on the first case (per attribute).
- Be greedy: sort these options and spend points until none are left.

Statistics: 139 submissions, 8 accepted, 78 unknown

# B: BnPC

Problem Author: Harry Smit

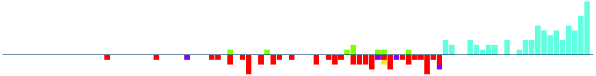


- **Problem:** increase attribute scores so that you maximize a certain score function.
- First set all attributes to the lowest value they need to be to pass all the challenges (if this is impossible, the maximum score is 0).
- Improve your score by increasing an attribute by one. There are two cases:
  - If the attribute score equals *a* of the challenge requirements, you get points equal to *a* times the new attribute score, plus the number of events that require a lower score for that attribute.
  - Otherwise, spending a point here gives additional score equal to the number of events that use this attribute.
- You can only spend one point on the first case (per attribute).
- Be greedy: sort these options and spend points until none are left.
- If you ever run into the second case, spend all of your points there.

Statistics: 139 submissions, 8 accepted, 78 unknown

# B: BnPC

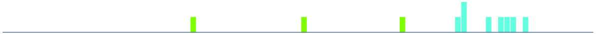
Problem Author: Harry Smit



- **Problem:** increase attribute scores so that you maximize a certain score function.
- First set all attributes to the lowest value they need to be to pass all the challenges (if this is impossible, the maximum score is 0).
- Improve your score by increasing an attribute by one. There are two cases:
  - If the attribute score equals *a* of the challenge requirements, you get points equal to *a* times the new attribute score, plus the number of events that require a lower score for that attribute.
  - Otherwise, spending a point here gives additional score equal to the number of events that use this attribute.
- You can only spend one point on the first case (per attribute).
- Be greedy: sort these options and spend points until none are left.
- If you ever run into the second case, spend all of your points there.
- Runtime: $\mathcal{O}(n \log n + l)$.

Statistics: 139 submissions, 8 accepted, 78 unknown

# C: Cangaroo



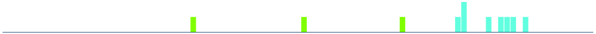
Problem Author: Abe Wits

- **Problem:** Given a $n \times m$ grid with marked locations, what is the minimum amount of $2 \times 2$ cans needed to cover all marked locations?

Statistics: 11 submissions, 3 accepted, 8 unknown

# C: Cangaroo

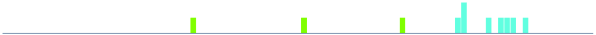
Problem Author: Abe Wits



- **Problem:** Given a $n \times m$ grid with marked locations, what is the minimum amount of $2 \times 2$ cans needed to cover all marked locations?
- **Solution:** Do DP and calculate what the minimal number of cans is needed if you fill up the last $r$ rows for a given can placement of the top row.
  For calculating the next row, iterate over all rows that support a can placement and take the best.

Statistics: 11 submissions, 3 accepted, 8 unknown

# C: Cangaroo



Problem Author: Abe Wits

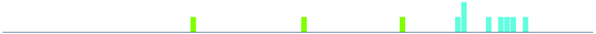
- **Problem:** Given a $n \times m$ grid with marked locations, what is the minimum amount of $2 \times 2$ cans needed to cover all marked locations?
- **Solution:** Do DP and calculate what the minimal number of cans is needed if you fill up the last $r$ rows for a given can placement of the top row.
  For calculating the next row, iterate over all rows that support a can placement and take the best.

$$\mathtt{DP}[\mathrm{row}][\mathcal{C}] = \begin{cases} |\mathcal{C}| + \min\limits_{\mathcal{D} \text{ supports } \mathcal{C}} \mathtt{DP}[\mathrm{row}-1][\mathcal{D}] & \text{if } \mathcal{C} \text{ covers locations,} \\ \infty & \text{else.} \end{cases}$$

Statistics: 11 submissions, 3 accepted, 8 unknown

# C: Cangaroo

Problem Author: Abe Wits



- **Problem:** Given a $n \times m$ grid with marked locations, what is the minimum amount of $2 \times 2$ cans needed to cover all marked locations?
- **Solution:** Do DP and calculate what the minimal number of cans is needed if you fill up the last $r$ rows for a given can placement of the top row.
  For calculating the next row, iterate over all rows that support a can placement and take the best.

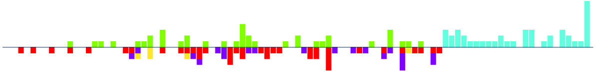
$$\mathtt{DP}[\mathrm{row}][\mathcal{C}] = \begin{cases} |\mathcal{C}| + \min\limits_{\mathcal{D} \text{ supports } \mathcal{C}} \mathtt{DP}[\mathrm{row}-1][\mathcal{D}] & \text{if } \mathcal{C} \text{ covers locations,} \\ \infty & \text{else.} \end{cases}$$

- Number of can placements is $F_{m+1}$, the $(m+1)$th Fibonacci number.
  Time complexity: $\mathcal{O}\left(n \cdot F_{m+1}^2\right) = \mathcal{O}\left(n \cdot 3.3^m\right)$ when using bitmasks.

Statistics: 11 submissions, 3 accepted, 8 unknown

# D: Decelerating Jump

Problem Author: Onno Berrevoets

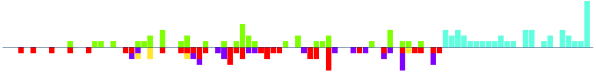


- **Problem:** Given a sequence of $n$ integers $p_1, \ldots, p_n$, find a subsequence $1 = p_{i_1} < p_{i_2} < \cdots < p_{i_k} = n$ such that the distance between consecutive elements does not increase.

Statistics: 146 submissions, 38 accepted, 43 unknown

# D: Decelerating Jump

Problem Author: Onno Berrevoets



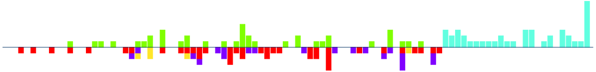
- **Problem:** Given a sequence of $n$ integers $p_1, \ldots, p_n$, find a subsequence $1 = p_{i_1} < p_{i_2} < \cdots < p_{i_k} = n$ such that the distance between consecutive elements does not increase.
- **Cubic solution:** Keep a DP table `dp[position][speed]`, which is computed as

$$\mathrm{dp}[i][s] = p_i + \max_{k \geq s} \mathrm{dp}[i-k][k]$$

Statistics: 146 submissions, 38 accepted, 43 unknown

# D: Decelerating Jump

Problem Author: Onno Berrevoets



- **Problem:** Given a sequence of $n$ integers $p_1, \ldots, p_n$, find a subsequence $1 = p_{i_1} < p_{i_2} < \cdots < p_{i_k} = n$ such that the distance between consecutive elements does not increase.
- **Cubic solution:** Keep a DP table `dp[position][speed]`, which is computed as

$$\mathrm{dp}[i][s] = p_i + \max_{k \geq s} \mathrm{dp}[i-k][k]$$

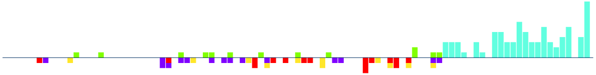
- **Quadratic solution:** Loop over speed $s$ from $n-1$ to 1, keeping track of the maximum score if you end in each cell with speed at least $s$. Then update all positions $i$ from 1 to $n$:

$$\mathrm{dp}[i] = \max(\mathrm{dp}[i], p_i + \mathrm{dp}[i-s])$$

Statistics: 146 submissions, 38 accepted, 43 unknown

# E: Evolutionary Excerpt

Problem Author: Ragnar Groot Koerkamp

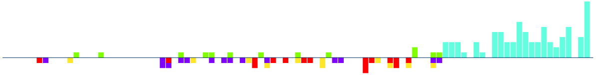


- **Problem:** Given two independent uniform random sequences over "`ACTG`" of length $n = 10^6$, find a common subsequence of length at least 500 000.

Statistics: 139 submissions, 13 accepted, 84 unknown

# E: Evolutionary Excerpt

Problem Author: Ragnar Groot Koerkamp

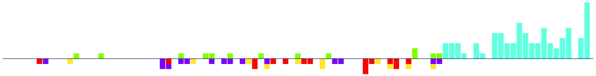


- **Problem:** Given two independent uniform random sequences over "`ACTG`" of length $n = 10^6$, find a common subsequence of length at least 500 000.
- Naive solution: run the Longest Common Subsequence algorithm. $\mathcal{O}(n^2)$ is too slow!

Statistics: 139 submissions, 13 accepted, 84 unknown

# E: Evolutionary Excerpt

Problem Author: Ragnar Groot Koerkamp

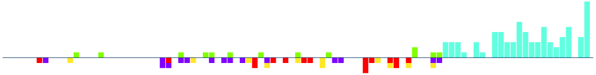


- **Problem:** Given two independent uniform random sequences over "`ACTG`" of length $n = 10^6$, find a common subsequence of length at least 500 000.
- Naive solution: run the Longest Common Subsequence algorithm. $\mathcal{O}(n^2)$ is too slow!
- Greedy: if the front two characters are the same, take it. Otherwise, remove the first character from the longer sequence. $\rightarrow$ length 400 000.

Statistics: 139 submissions, 13 accepted, 84 unknown

# E: Evolutionary Excerpt

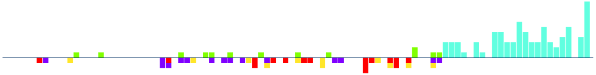
Problem Author: Ragnar Groot Koerkamp



- Greedy, second attempt: Instead of only comparing the front characters, we can compare the front character of each sequence with the first three or four characters of the other sequence, and use the first match we find. $\rightarrow$ length 531 000.

# E: Evolutionary Excerpt

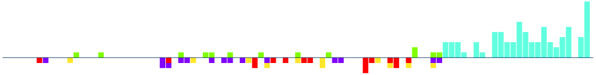
Problem Author: Ragnar Groot Koerkamp



- Greedy, second attempt: Instead of only comparing the front characters, we can compare the front character of each sequence with the first three or four characters of the other sequence, and use the first match we find. $\rightarrow$ length 531 000.
- LCS DP, but smarter: instead of computing the full $n^2$ DP table, we can only keep entries close to the diagonal. Keeping a diagonal of width $k = 10 \rightarrow$ length 624 000, $\mathcal{O}(nk)$.

# E: Evolutionary Excerpt

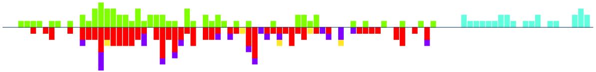
Problem Author: Ragnar Groot Koerkamp



- Greedy, second attempt: Instead of only comparing the front characters, we can compare the front character of each sequence with the first three or four characters of the other sequence, and use the first match we find. $\rightarrow$ length 531 000.
- LCS DP, but smarter: instead of computing the full $n^2$ DP table, we can only keep entries close to the diagonal. Keeping a diagonal of width $k = 10 \rightarrow$ length 624 000, $\mathcal{O}(nk)$.
- Split the input in chunks of size $k \geq 7$, and run LCS for each chunk. $\rightarrow \mathcal{O}(nk)$, length 502 000 for $k = 7$, length 530 000 for $k = 10$.
  Probability of failure is less than $10^{-16}$ for $k = 7$, and less than $10^{-1000}$ from $k = 9$ onward.

# F: Fair Play

Problem Author: Robin Lee

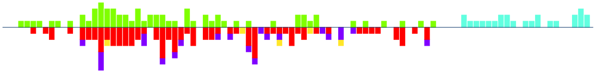


- **Problem:** decide if it is possible to pair up vectors so that the sum of each pair is the same.

Statistics: 208 submissions, 66 accepted, 25 unknown

# F: Fair Play

Problem Author: Robin Lee

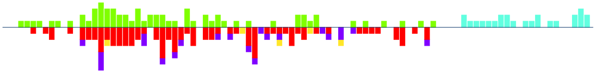


- **Problem:** decide if it is possible to pair up vectors so that the sum of each pair is the same.
- If this is possible, then the sum is equal to two times the average. Calculate this average, and check if it is integer.

Statistics: 208 submissions, 66 accepted, 25 unknown

# F: Fair Play

Problem Author: Robin Lee

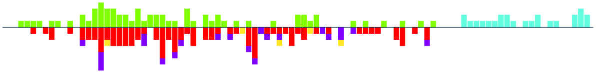


- **Problem:** decide if it is possible to pair up vectors so that the sum of each pair is the same.
- If this is possible, then the sum is equal to two times the average. Calculate this average, and check if it is integer.
- If it is, say it is $(a, b)$, pair up the vectors one by one: for every vector $(x, y)$ there needs to be a vector $(2a - x, 2b - y)$.

Statistics: 208 submissions, 66 accepted, 25 unknown

# F: Fair Play

Problem Author: Robin Lee

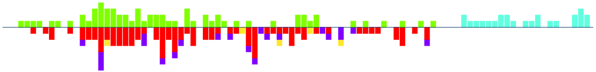


- **Problem:** decide if it is possible to pair up vectors so that the sum of each pair is the same.
- If this is possible, then the sum is equal to two times the average. Calculate this average, and check if it is integer.
- If it is, say it is $(a, b)$, pair up the vectors one by one: for every vector $(x, y)$ there needs to be a vector $(2a - x, 2b - y)$.
- Make sure to check that $(x, y)$ and $(2a - x, 2b - y)$ occur equally often!

Statistics: 208 submissions, 66 accepted, 25 unknown

# F: Fair Play

Problem Author: Robin Lee



- **Problem:** decide if it is possible to pair up vectors so that the sum of each pair is the same.
- If this is possible, then the sum is equal to two times the average. Calculate this average, and check if it is integer.
- If it is, say it is $(a, b)$, pair up the vectors one by one: for every vector $(x, y)$ there needs to be a vector $(2a - x, 2b - y)$.
- Make sure to check that $(x, y)$ and $(2a - x, 2b - y)$ occur equally often!
- Runtime: $\mathcal{O}(n)$.

Statistics: 208 submissions, 66 accepted, 25 unknown

# G: Gyrating Glyphs

Problem Author: Reinier Schmiermann

- **Problem:** Reverse engineer the $\leq 20000$ operators using $\leq 1400$ queries:

$$f_n(a_0, \dots, a_n) := (\dots(((a_0 \, \mathrm{op}_1 \, a_1) \, \mathrm{op}_2 \, a_2) \, \mathrm{op}_3 \, a_3) \dots \mathrm{op}_n \, a_n) \bmod 10^9 + 7$$

Statistics: 14 submissions, 0 accepted, 10 unknown

# G: Gyrating Glyphs

Problem Author: Reinier Schmiermann

- **Problem:** Reverse engineer the $\leq 20000$ operators using $\leq 1400$ queries:

$$f_n(a_0, \ldots, a_n) := (\ldots(((a_0 \operatorname{op}_1 a_1) \operatorname{op}_2 a_2) \operatorname{op}_3 a_3) \ldots \operatorname{op}_n a_n) \bmod 10^9 + 7$$

- First solve the problem for 15 operators with a single query $0, q_1, \ldots, q_{15}$.

Statistics: 14 submissions, 0 accepted, 10 unknown

# G: Gyrating Glyphs

Problem Author: Reinier Schmiermann

- **Problem:** Reverse engineer the $\leq 20000$ operators using $\leq 1400$ queries:

$$f_n(a_0, \ldots, a_n) := (\ldots(((a_0 \operatorname{op}_1 a_1) \operatorname{op}_2 a_2) \operatorname{op}_3 a_3) \ldots \operatorname{op}_n a_n) \bmod 10^9 + 7$$

- First solve the problem for 15 operators with a single query $0, q_1, \ldots, q_{15}$.
- Use this to find all operators in $20000/15 < 1400$ queries.

Statistics: 14 submissions, 0 accepted, 10 unknown

# G: Gyrating Glyphs

Problem Author: Reinier Schmiermann

- **Problem:** Reverse engineer the $\leq 20000$ operators using $\leq 1400$ queries:

$$f_n(a_0, \ldots, a_n) := (\ldots(((a_0 \operatorname{op}_1 a_1) \operatorname{op}_2 a_2) \operatorname{op}_3 a_3) \ldots \operatorname{op}_n a_n) \bmod 10^9 + 7$$

- First solve the problem for 15 operators with a single query $0, q_1, \ldots, q_{15}$.
- Use this to find all operators in $20000/15 < 1400$ queries.
- Example with 30 operators:

Recover last 15 operators:

| | | |
|---|---|---|
| $??? \ldots ??$ | $??? \ldots ???$ | Ops |
| $\underbrace{000 \ldots 000}_{16}$ | $q_1 q_2 \ldots q_{15}$ | Query 1 |

$+0$ and $\times 1$ do not change the query outcome.

Continue with the next 15 operators.

| | | |
|---|---|---|
| $??? \ldots ??$ | $\cancel{+ \times + \ldots + \times}$ | Ops |
| $0 q_1 \ldots q_{15}$ | $\cancel{010 \ldots 01}$ | Query 2 |

Statistics: 14 submissions, 0 accepted, 10 unknown

# G: Gyrating Glyphs

Problem Author: Reinier Schmiermann

- We consider the case with 15 operators.

# G: Gyrating Glyphs

Problem Author: Reinier Schmiermann

- We consider the case with 15 operators.
- Let $0, q_1, \ldots, q_{15}$ where $q_i$ is random in $\{1, \ldots, 10^9 + 6\}$.

# G: Gyrating Glyphs

Problem Author: Reinier Schmiermann

- We consider the case with 15 operators.
- Let $0, q_1, \dots, q_{15}$ where $q_i$ is random in $\{1, \dots, 10^9 + 6\}$.
- For all $2^{15}$ possibilities for the 15 operators compute the query outcome.

# G: Gyrating Glyphs

Problem Author: Reinier Schmiermann

- We consider the case with 15 operators.
- Let $0, q_1, \ldots, q_{15}$ where $q_i$ is random in $\{1, \ldots, 10^9 + 6\}$.
- For all $2^{15}$ possibilities for the 15 operators compute the query outcome.
- If all outcomes are distinct $(\bmod 10^9 + 7)$ we have a lookup table.

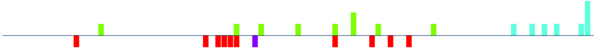
# G: Gyrating Glyphs

Problem Author: Reinier Schmiermann

- We consider the case with 15 operators.
- Let $0, q_1, \ldots, q_{15}$ where $q_i$ is random in $\{1, \ldots, 10^9 + 6\}$.
- For all $2^{15}$ possibilities for the 15 operators compute the query outcome.
- If all outcomes are distinct $(\bmod 10^9 + 7)$ we have a lookup table.
- If not, repeat with a new random query.

# H: Hamiltooonian Hike

Problem Author: Jorke de Vlas

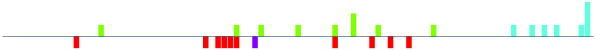


- **Problem:** Find a hiking path that visits all cabins, walking at most three trails every day.

Statistics: 28 submissions, 9 accepted, 8 unknown

# H: Hamiltooonian Hike

Problem Author: Jorke de Vlas

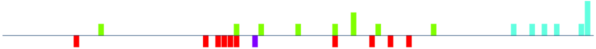


- **Problem:** Find a hiking path that visits all cabins,
  walking at most three trails every day.
- **Solution:** A modified DFS, starting from an arbitrary cabin $s$.

Statistics: 28 submissions, 9 accepted, 8 unknown

# H: Hamiltooonian Hike

Problem Author: Jorke de Vlas

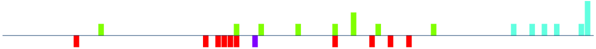


- **Problem:** Find a hiking path that visits all cabins,
  walking at most three trails every day.
- **Solution:** A modified DFS, starting from an arbitrary cabin $s$.
- **Variant 1:**
  - While *descending*, only stop at cabins that have *odd* distance to $s$.
  - While *ascending*, only stop at cabins that have *even* distance to $s$.

Statistics: 28 submissions, 9 accepted, 8 unknown

# H: Hamiltooonian Hike

Problem Author: Jorke de Vlas

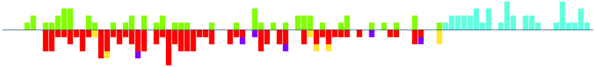


- **Problem:** Find a hiking path that visits all cabins,
  walking at most three trails every day.
- **Solution:** A modified DFS, starting from an arbitrary cabin $s$.
- **Variant 1:**
  - While *descending*, only stop at cabins that have *odd* distance to $s$.
  - While *ascending*, only stop at cabins that have *even* distance to $s$.
- **Variant 2:**
  While *descending*, only stop at a cabin $c$ if either:
  - you have walked three trails since the last cabin you stopped at, or
  - you have already walked past all neighbouring cabins of $c$ and need to *ascend* again.

Statistics: 28 submissions, 9 accepted, 8 unknown

# I: Implementation Irregularities

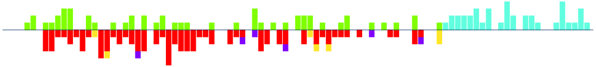
Problem Author: Ragnar Groot Koerkamp



- **Problem:** Given a list of $n \leq 10^5$ problems, the computer time $t_i$ needed to solve each of them, and the time $s_i$ each was solved, find the minimal number of computers used.

Statistics: 190 submissions, 52 accepted, 39 unknown

# I: Implementation Irregularities

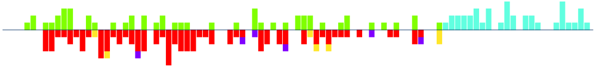


Problem Author: Ragnar Groot Koerkamp

- **Problem:** Given a list of $n \leq 10^5$ problems, the computer time $t_i$ needed to solve each of them, and the time $s_i$ each was solved, find the minimal number of computers used.
- Insight: If there are $C$ computers and the team solves their $k$th problem after $s$ minutes, the total computer time available for the first $k$ problems is $C \cdot s$.

Statistics: 190 submissions, 52 accepted, 39 unknown

# I: Implementation Irregularities

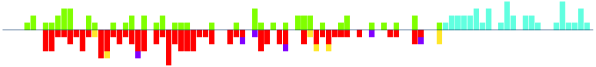


Problem Author: Ragnar Groot Koerkamp

- **Problem:** Given a list of $n \leq 10^5$ problems, the computer time $t_i$ needed to solve each of them, and the time $s_i$ each was solved, find the minimal number of computers used.
- Insight: If there are $C$ computers and the team solves their $k$th problem after $s$ minutes, the total computer time available for the first $k$ problems is $C \cdot s$.
- Sort the problems by solve time, discarding any unsolved problems.

Statistics: 190 submissions, 52 accepted, 39 unknown

# I: Implementation Irregularities

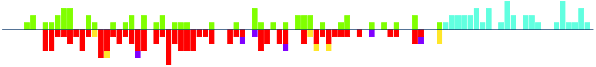


Problem Author: Ragnar Groot Koerkamp

- **Problem:** Given a list of $n \leq 10^5$ problems, the computer time $t_i$ needed to solve each of them, and the time $s_i$ each was solved, find the minimal number of computers used.
- Insight: If there are $C$ computers and the team solves their $k$th problem after $s$ minutes, the total computer time available for the first $k$ problems is $C \cdot s$.
- Sort the problems by solve time, discarding any unsolved problems.
- For each $k$ we must have $\sum_{i=1}^{k} t_i \leq C \cdot s_k$.

Statistics: 190 submissions, 52 accepted, 39 unknown

# I: Implementation Irregularities

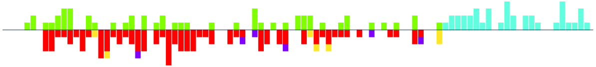


Problem Author: Ragnar Groot Koerkamp

- **Problem:** Given a list of $n \leq 10^5$ problems, the computer time $t_i$ needed to solve each of them, and the time $s_i$ each was solved, find the minimal number of computers used.
- Insight: If there are $C$ computers and the team solves their $k$th problem after $s$ minutes, the total computer time available for the first $k$ problems is $C \cdot s$.
- Sort the problems by solve time, discarding any unsolved problems.
- For each $k$ we must have $\sum_{i=1}^{k} t_i \leq C \cdot s_k$.
- The answer $C$ is the maximum of $\left(\sum_{i=1}^{k} t_i\right) \Big/ s_k$, rounded up.

Statistics: 190 submissions, 52 accepted, 39 unknown

# I: Implementation Irregularities



Problem Author: Ragnar Groot Koerkamp

- **Problem:** Given a list of $n \leq 10^5$ problems, the computer time $t_i$ needed to solve each of them, and the time $s_i$ each was solved, find the minimal number of computers used.
- Insight: If there are $C$ computers and the team solves their $k$th problem after $s$ minutes, the total computer time available for the first $k$ problems is $C \cdot s$.
- Sort the problems by solve time, discarding any unsolved problems.
- For each $k$ we must have $\sum_{i=1}^{k} t_i \leq C \cdot s_k$.
- The answer $C$ is the maximum of $\left(\sum_{i=1}^{k} t_i\right) \Big/ s_k$, rounded up.
- Alternatively, you can binary search.

Statistics: 190 submissions, 52 accepted, 39 unknown

# J: Jail or Joyride

Problem Author: Reinier Schmiermann

- **Problem:** find the least distance that a police car needs to travel to catch a group of teenagers on a graph, given that the teenagers flee as far away as possible on every approach.

Statistics: 5 submissions, 0 accepted, 2 unknown

# J: Jail or Joyride

Problem Author: Reinier Schmiermann

- **Problem:** find the least distance that a police car needs to travel to catch a group of teenagers on a graph, given that the teenagers flee as far away as possible on every approach.
- Observation 1: If the police can approach the teenagers via multiple edges, then the teenagers can always reach every vertex in the graph.
  - In particular: the approach direction of the police does not matter.
  - The police should always take the shortest path.

Statistics: 5 submissions, 0 accepted, 2 unknown

# J: Jail or Joyride

Problem Author: Reinier Schmiermann

- **Problem:** find the least distance that a police car needs to travel to catch a group of teenagers on a graph, given that the teenagers flee as far away as possible on every approach.
- Observation 1: If the police can approach the teenagers via multiple edges, then the teenagers can always reach every vertex in the graph.
  - In particular: the approach direction of the police does not matter.
  - The police should always take the shortest path.
- Observation 2: If the police can approach the teenagers via only one edge, then either the teenagers are in a leaf, or they are not as far away as possible from the police.
  - Second case only happens at the start.
  - After this, the teenagers can always either reach the whole graph, or nothing at all.

Statistics: 5 submissions, 0 accepted, 2 unknown

# J: Jail or Joyride

Problem Author: Reinier Schmiermann

- The police always takes the shortest path to the teenagers.
- After the first approach of the police, the teenagers can always either reach the whole graph, or nothing.

# J: Jail or Joyride

Problem Author: Reinier Schmiermann

- The police always takes the shortest path to the teenagers.
- After the first approach of the police, the teenagers can always either reach the whole graph, or nothing.
- Simulate the first approach of the police separately.

# J: Jail or Joyride

Problem Author: Reinier Schmiermann

- The police always takes the shortest path to the teenagers.
- After the first approach of the police, the teenagers can always either reach the whole graph, or nothing.
- Simulate the first approach of the police separately.
- For every vertex which is not a leaf: find all vertices which are as far away as possible (use APSP).

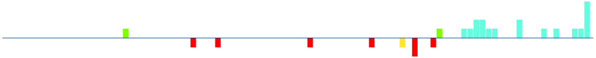
# J: Jail or Joyride

Problem Author: Reinier Schmiermann

- The police always takes the shortest path to the teenagers.
- After the first approach of the police, the teenagers can always either reach the whole graph, or nothing.
- Simulate the first approach of the police separately.
- For every vertex which is not a leaf: find all vertices which are as far away as possible (use APSP).
- Use DFS on this new directed graph to compute for every vertex $v$ the maximal distance the police needs to travel after approaching the teenagers in $v$.
  - If there is a reachable cycle in this new graph, the police cannot catch the teenagers.

# K: Kinking Cables

Problem Author: Boas Kluiving

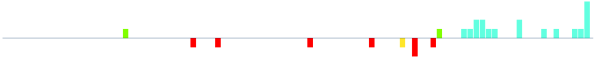


- **Problem:** Connect opposite corners of a rectangle with a cable of length $\ell$ such that
  - Line segments do not intersect.
  - The coordinate points of the path should not be too close ($< 1$) to each other.

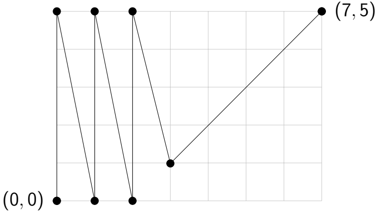

Statistics: 28 submissions, 2 accepted, 18 unknown

# K: Kinking Cables

Problem Author: Boas Kluiving



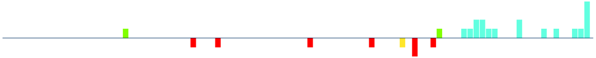
- **Problem:** Connect opposite corners of a rectangle with a cable of length $\ell$ such that
  - Line segments do not intersect.
  - The coordinate points of the path should not be too close ($< 1$) to each other.
- **Solution:** Zig-zag and use binary search for the last point:



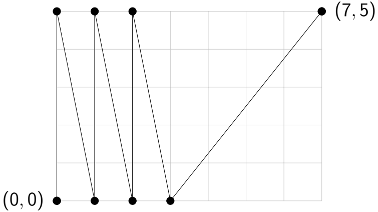

Statistics: 28 submissions, 2 accepted, 18 unknown

# K: Kinking Cables

Problem Author: Boas Kluiving

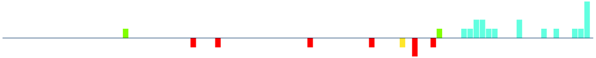


- **Problem:** Connect opposite corners of a rectangle with a cable of length $\ell$ such that
  - Line segments do not intersect.
  - The coordinate points of the path should not be too close ($< 1$) to each other.
- **Solution:** Zig-zag and use binary search for the last point:



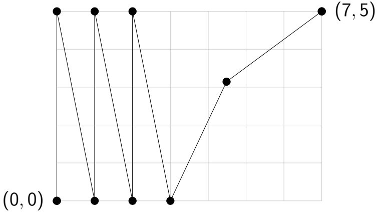

Statistics: 28 submissions, 2 accepted, 18 unknown

# K: Kinking Cables



Problem Author: Boas Kluiving

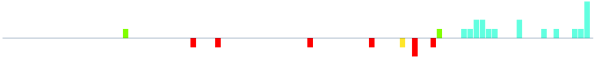
- **Problem:** Connect opposite corners of a rectangle with a cable of length $\ell$ such that
  - Line segments do not intersect.
  - The coordinate points of the path should not be too close ($< 1$) to each other.
- **Solution:** Zig-zag and use binary search for the last point:



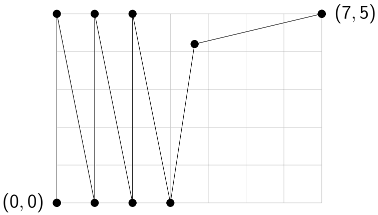

Statistics: 28 submissions, 2 accepted, 18 unknown

# K: Kinking Cables

Problem Author: Boas Kluiving



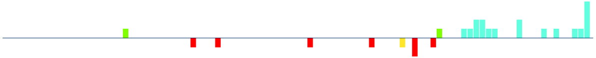
- **Problem:** Connect opposite corners of a rectangle with a cable of length $\ell$ such that
  - Line segments do not intersect.
  - The coordinate points of the path should not be too close ($< 1$) to each other.
- **Solution:** Zig-zag and use binary search for the last point:



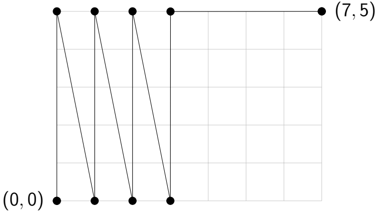

Statistics: 28 submissions, 2 accepted, 18 unknown

# K: Kinking Cables

Problem Author: Boas Kluiving



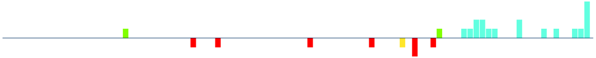
- **Problem:** Connect opposite corners of a rectangle with a cable of length $\ell$ such that
  - Line segments do not intersect.
  - The coordinate points of the path should not be too close ($< 1$) to each other.
- **Solution:** Zig-zag and use binary search for the last point:



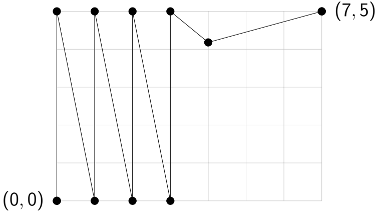

Statistics: 28 submissions, 2 accepted, 18 unknown

# K: Kinking Cables

Problem Author: Boas Kluiving



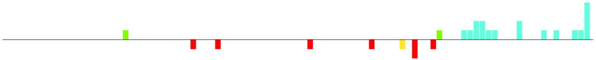
- **Problem:** Connect opposite corners of a rectangle with a cable of length $\ell$ such that
  - Line segments do not intersect.
  - The coordinate points of the path should not be too close ($< 1$) to each other.
- **Solution:** Zig-zag and use binary search for the last point:



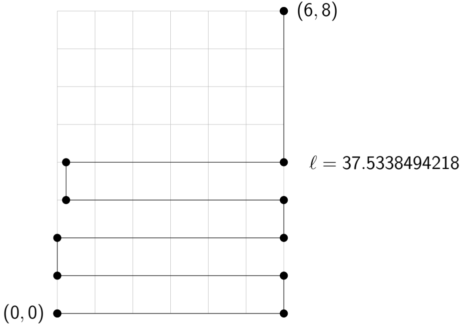

Statistics: 28 submissions, 2 accepted, 18 unknown

# K: Kinking Cables

Problem Author: Boas Kluiving



**Bonus slide:** Honourable mention for team "print(math.tan(float(input())))", for creating a solution without any diagonal lines and just simple arithmetic (which none of the jury members had thought of):



# L: Lopsided Lineup

Problem Author: Jorke de Vlas

- **Problem:** Split a group of people in two equally sized teams that are as unequally matched as possible.

Statistics: 12 submissions, 7 accepted, 3 unknown

# L: Lopsided Lineup

Problem Author: Jorke de Vlas

- **Problem:** Split a group of people in two equally sized teams that are as unequally matched as possible.
- Although the question is about synergy, the solution is actually to take *strong players* for the winning team.

Statistics: 12 submissions, 7 accepted, 3 unknown

# L: Lopsided Lineup

Problem Author: Jorke de Vlas

- **Problem:** Split a group of people in two equally sized teams that are as unequally matched as possible.
- Although the question is about synergy, the solution is actually to take *strong players* for the winning team.
- Take the first $n/2$ players as the strong team. Then what is the difference in scores?

Statistics: 12 submissions, 7 accepted, 3 unknown

# L: Lopsided Lineup

Problem Author: Jorke de Vlas

- **Problem:** Split a group of people in two equally sized teams that are as unequally matched as possible.
- Although the question is about synergy, the solution is actually to take *strong players* for the winning team.
- Take the first $n/2$ players as the strong team. Then what is the difference in scores?

$$c = \begin{pmatrix} S & \\ & W \end{pmatrix}$$

$$\text{score} = \frac{1}{2}(S - W)$$

# L: Lopsided Lineup

Problem Author: Jorke de Vlas

- **Problem:** Split a group of people in two equally sized teams that are as unequally matched as possible.
- Although the question is about synergy, the solution is actually to take *strong players* for the winning team.
- Take the first $n/2$ players as the strong team. Then what is the difference in scores?

$$c = \begin{pmatrix} S & X \\ X & W \end{pmatrix}$$

$$\text{score} = \frac{1}{2}(S - W)$$

# L: Lopsided Lineup

Problem Author: Jorke de Vlas

- **Problem:** Split a group of people in two equally sized teams that are as unequally matched as possible.
- Although the question is about synergy, the solution is actually to take *strong players* for the winning team.
- Take the first $n/2$ players as the strong team. Then what is the difference in scores?

$$c = \begin{pmatrix} S & X \\ X & W \end{pmatrix}$$

$$\text{score} = \frac{1}{2}((S+X)-(W+X))$$

# L: Lopsided Lineup

Problem Author: Jorke de Vlas

- **Problem:** Split a group of people in two equally sized teams that are as unequally matched as possible.
- Although the question is about synergy, the solution is actually to take *strong players* for the winning team.
- Take the first $n/2$ players as the strong team. Then what is the difference in scores?

$$c = \begin{pmatrix} \boxed{S+X} \\ \boxed{W+X} \end{pmatrix}$$

$$\text{score} = \frac{1}{2}((S+X)-(W+X))$$

# L: Lopsided Lineup

Problem Author: Jorke de Vlas

$$c = \begin{pmatrix} \boxed{S + X} \\ \boxed{W + X} \end{pmatrix} \qquad \text{score} = \frac{1}{2}((S+X)-(W+X))$$

- The score of each team is the sum of its players' row sums.

# L: Lopsided Lineup

Problem Author: Jorke de Vlas

$$c = \begin{pmatrix} \boxed{S + X} \\ \boxed{W + X} \end{pmatrix} \qquad \text{score} = \frac{1}{2}((S+X)-(W+X))$$

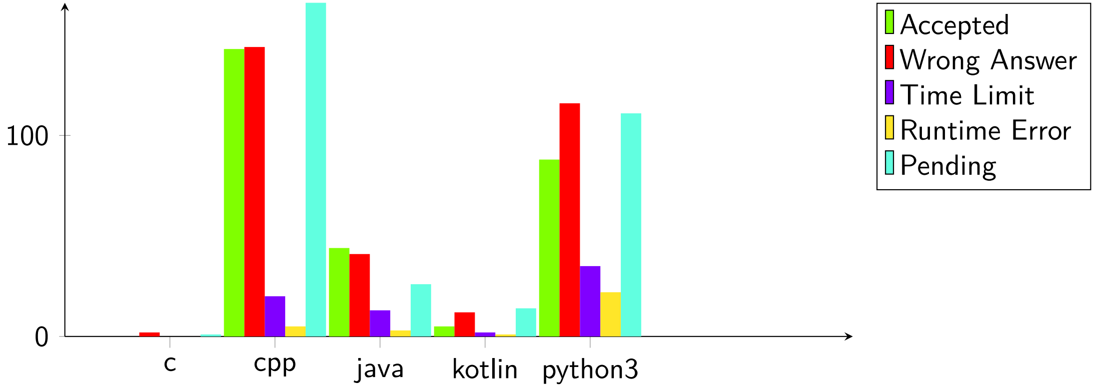
- The score of each team is the sum of its players' row sums.
- If you take any *other* strong team, you can reorder the matrix $c$ so that your chosen team is the first $n/2$. That does not change the row sums!

# L: Lopsided Lineup

Problem Author: Jorke de Vlas

- **Solution:** for each player compute its strength (i.e. the sum of its row). Take the $n/2$ strongest players for the strong team, and the others for the weak team.
- Complexity: $\mathcal{O}(n^2)$.

# Language stats



# Some stats

- 917 commits, of which 507 for the main contest
- 693 secret test cases (last year: 425) ($\approx$ 58 per problem!)
- 177 jury solutions (last year: 204)
- The minimum[1] number of lines the jury needed to solve all problems is

$$2 + 15 + 15 + 6 + 5 + 5 + 14 + 12 + 5 + 26 + 8 + 2 = 115$$

On average 9.6 lines per problem, up from 7.5 in the preliminaries

[1] Most jury members do enjoy a good code golfing competition!

# Thanks to the Proofreaders!

Jaap Eldering
Nicky Gerritsen
Mart Pluijmaekers
Michael Vasseur
Kevin Verbeek

# The Jury

Boas Kluiving
Erik Baalhuis
Freek Henstra
Harry Smit
Joey Haas

Jorke de Vlas
Ludo Pulles
Maarten Sijm
Mees de Vries
Ragnar Groot Koerkamp

Reinier Schmiermann
Robin Lee
Ruben Brokkelkamp
Timon Knigge
Wessel van Woerden

# Thanks to the Sponsors!