IDI Open 2025

Presentation of solutions

March 1st 2025

The Judges and Testers

Judges

- Jean Niklas L'orange (Kodemaker)
- Tobias Meyer Andersen (SINTEF Digital)
- Dilawar Mahmood (Apple)

Test Solver

- Johan Sokrates Wind (University of Oslo)

We need more judges and testers! Send an email to `jeannikl@hypirion.com`, or contact any of the judges/organizers after the presentation.

Milk Mystery, Author: Dilawar

- Find maximum sum of K consecutive deliciousness scores
- Sliding window approach avoids recalculating sums:
  - Compute initial sum of first K elements
  - For subsequent windows: subtract leaving element, add new element
  - Track maximum sum encountered
- Time complexity $\mathcal{O}(N)$ vs brute-force $\mathcal{O}(N^2)$
- Handles edge cases through proper window boundary management

Solved by 15 teams
First solution after 8 minutes

Flint Flinger, Author: Tobias

- Determine how many cities threaten each city
- Compute the length of the vectors between cities
- Avoid counting oneself!

Solved by 13 teams
First solution after 23 minutes

Ballfoot Battles, Author: Tobias

- Each time you check the score you have an equivalent problem
- Compute how many ways this last "short game" could have been played
- Equivalent to how many distinct ways to order a set of colored balls with a color for each team
- Divide the total number of orderings by the amount of orderings of each color
- $\prod_i \frac{(\sum_j g_{i,j})!}{\sum_j (g_{i,j})!}$
- Precomputing factorials can make this very quick

Solved by 2 teams
First solution after 125 minutes

Exclusive Equations, Author: Tobias

- Problem: combine subset of numbers witt XOR ($\oplus$) to obtain target number T, only 35 unique numbers
- Brute force: check all 2^{3500} subsets, too slow
- XOR is commutative, associative, and every number is its own inverse
- Multiple copies of a number does not produce any more unique results!
- Better brute force: check all 2^{35} subsets of unique numbers

Exclusive Equations

- Problem: combine subset of numbers witt XOR ($\oplus$) to obtain target number T, only 35 unique numbers
- Brute force: check all 2^{3500} subsets, too slow
- XOR is commutative, associative, and every number is its own inverse
- Multiple copies of a number does not produce any more unique results!
- Better brute force: check all 2^{35} subsets of unique numbers
- Still too slow

Exclusive Equations

- New strategy: meet in the middle
- Split the numbers in two halves
- Let A and B contain the XOR of the powersets of each half
- For each element a' in A, we want to know if a b' in B exists such that $a' \oplus b' = T$
- $T \oplus a' = b'$ by just XORing with a', so check if $T \oplus a' \in B$
- Sets have lookup time, this gives 2^{18} operations, fast enough!

Solved by 2 team
First solution after 95 minutes

Crate Chucking, Author: Tobias

- This problem is a version of the internet flash game Bloxorz

- Basically, move an rectangular box on a grid using valid squares
- This particular problem also introduces the concept of energy
- Also limitation on moving in the same direction twice in a row

Crate Chucking

- This can be solved with a shortest path algorithm, or a breadth first search
- Regular graph representation with one node per cell makes graph traversal hard to implement!
- Make one node for all three orientations the crate is in
- Produce four more nodes per cell and orientation representing the direation it came from
- You now have 12 nodes per cell and must add the weighted edges that are valid
- A regular Dijkstra implementation will then work!

Crate Chucking

- Instead of weighted edges, you can also add more nodes!
- If the distance is 3, add nodes between the nodes of the cells
- We have now turned a weighted graph to an unweighted one
- A simple BFS will now work

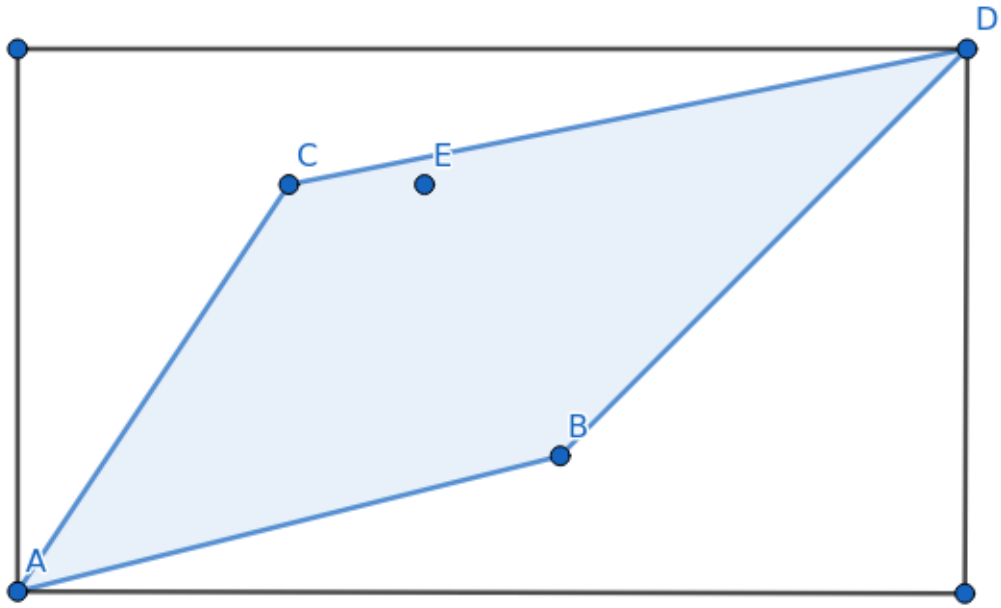
Solved by 1 team
First solution after 237 minutes

Ant Attack, Author: Tobias

- Find smallest rotated rectangle containing all points
- "Oriented Minimum Bounding Box" (OMBB)
- Simplify problem by computing convex hull
- One OMBB must have an edge along the convex hull
- Computing minimum bounding box for each edge is too slow
- Solution is to use the rotating calipers technique

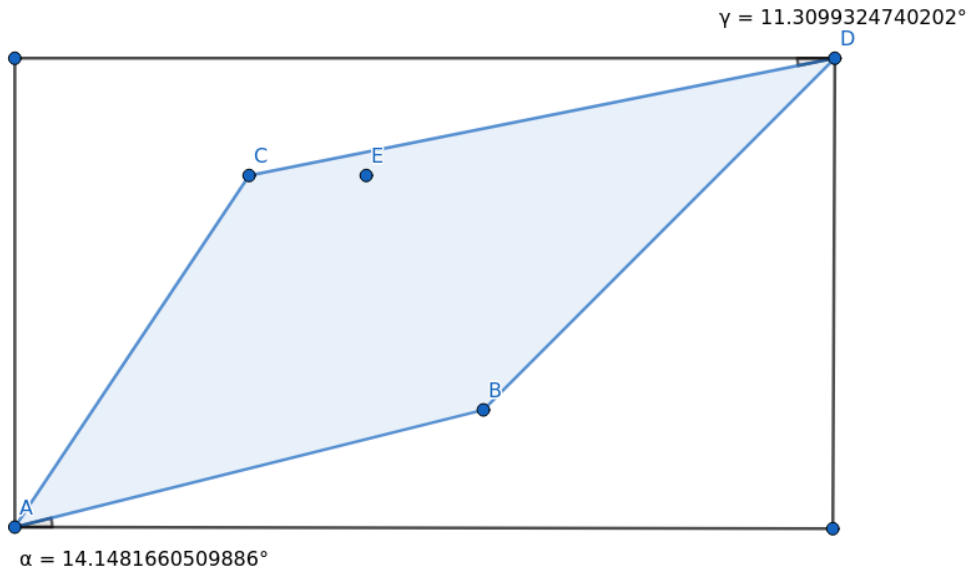
Ant Attack

- Start by making an axis aligned bounding box



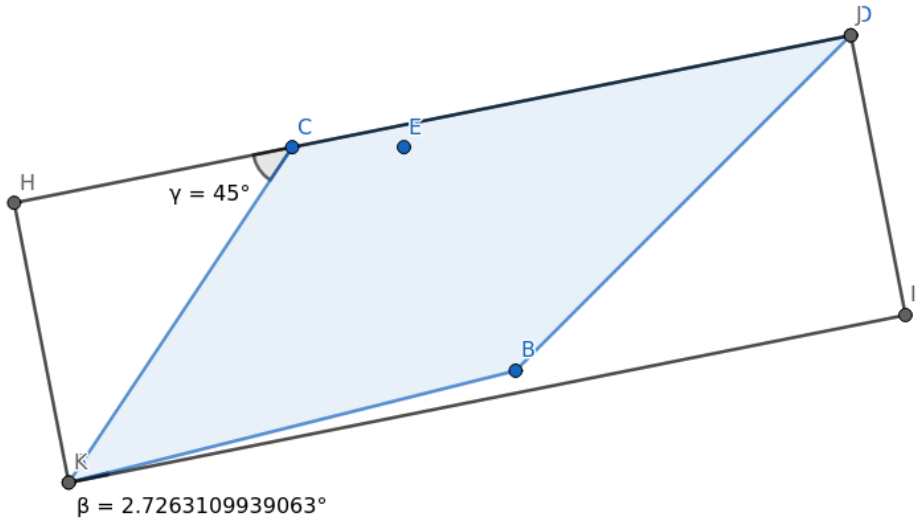
Ant Attack

- Figure out which edge will align first with the box if you gradually rotate it



Ant Attack

- Rotate it that minimal amount



- This rotation is done in constant time
- One rotation per edge in the convex hull
- Keep track of smallest area seen so far
- Linear time complexity in computing the OMBB
- Given that you have computed the convex hull first, which is NlogN

Solved by no teams

Divisor Detective, Author: Dilawar

- Sum divisor counts for all numbers 1 to N using Dirichlet's formula:

$$\sum_{k=1}^{N} d(k) = \sum_{i=1}^{N} \left\lfloor \frac{N}{i} \right\rfloor$$

- Naive approach: Check each number's divisors directly
  - Too slow: $\mathcal{O}(N\sqrt{N})$ complexity
  - Even using the formula directly is $\mathcal{O}(N)$, still too slow
- Key insight: $\left\lfloor \frac{N}{i} \right\rfloor$ remains constant for ranges of i
- These ranges change roughly $\sqrt{N}$ times

Divisor Detective

- **Single-loop grouping**:
  - For each quotient q, find range $[i, high]$ where $\lfloor \frac{N}{i} \rfloor = q$
  - Add $q \times (high - i + 1)$ to result
  - Jump directly to $high + 1$ for next range
  - Careful with perfect squares to avoid off-by-one errors
- **Two-phase approach** (Johan's solution):
  - Phase 1: Direct sum for $i \leq \sqrt{N}$ (small divisors)
  - Phase 2: For $j \leq \sqrt{N}$, sum contributions from large divisors
  - Avoids duplicate counting at $\sqrt{N}$ boundary

Divisor Detective

- Both methods achieve $\mathcal{O}(\sqrt{N})$ complexity
- Handle large inputs efficiently ($N \leq 10^{10}$)
- Common pitfalls:
  - Off-by-one errors near perfect squares
  - Integer overflow in intermediate calculations
  - Stopping iteration too early

Solved by 3 teams
First solution after 77 minutes

Timely Treatments

- Problem: Schedule maximum patients before their deadlines
- Greedy approach is optimal:
  - Sort patients by deadline (earliest first)
  - If total time exceeds deadline, remove longest treatment
  - This preserves maximum possible patients treated
- Key insight: Removing longest treatment minimizes impact on future capacity

Timely Treatments, Author: Dilawar

- **Max-heap strategy**:
  - Sort patients by deadline: $\mathcal{O}(N \log N)$
  - For each patient:
    - Add treatment time to heap: $\mathcal{O}(\log N)$
    - If total exceeds deadline, remove max: $\mathcal{O}(\log N)$
  - Simple to implement, efficient in practice
- Total complexity: $\mathcal{O}(N \log N)$

Timely Treatments

- **Segment tree approach**:
  - Build tree to track available time slots
  - Binary search for latest feasible treatment window
  - Maintain cumulative treatment times in nodes
  - Useful for variant problems with:
    - Range updates
    - Dynamic scheduling changes
    - Multiple resource constraints

Solved by 3 teams
First solution after 121 minutes

Incinerating Incantations, Author: Jean Niklas

- Simulation problem: Follow the rules **in correct order**
- Few enough spells that you can try all $\mathcal{O}(n!)$ combinations
  - `permutations` from the `itertools` package makes this easy
- Ensure mana and fire power don't go below 0
  - NB: Zeroing must be done after every step, and cannot be applied "in bulk"

Solved by 2 teams
First solution after 214 minutes

Hopping Haven, Author: Jean Niklas

- Problem: Find all sequence of moves a rectangle on a grid can do to avoid the most moving squares
- DP problem with a lot of details
- First make a graph $G = (V, E)$ for all valid kingdom locations and edges between them
- Then make a hash map $DP : (V, S) \to (\texttt{hits}, n)$ where S represents the set of active squares, and n represents the number of states that reaches the state $(v, s, \texttt{hits})$
- Initial state is $DP^0 = \{(v_{init}, \{\}) : (0, 1)\}$
- Also keep track of all active squares (A)

Hopping Haven

For even seconds i:

- First add new active squares appearing to A
- Make a new empty map $DP^{i+1} = \{\}$
- Then, for all $(v, s) \to (\text{hits}, n)$ in DP^i:
  - For all neighbours v_j of v:
    - Let $s_j = s \cup \{\text{squares intersecting } v_j\}$
    - If $DP^{i+1}[(v_j, s_j)].\text{hits} = \text{hits}$, add n to it
    - If $DP^{i+1}[(v_j, s_j)].\text{hits} > \text{hits}$ or does not exist, replace it with (hits, n)
    - (In English: For every (v, s), keep track of all move strategies that has the lowest number of hits)

Hopping Haven

For odd seconds i:

- Move all active squares A by their delta
- Remove squares that are outside the grid from A, if any
- Make a new empty map $DP^{i+1} = \{\}$
- For all $(v, s) \rightarrow (\text{hits}, n)$ in DP^i:
  - Let $s_j = (s \cup \{\text{squares intersecting } v\}) \cap A$
    (Important to remove the vanished squares)
  - Insert into DP^{i+1} according to the rules of last slide

Hopping Haven

- When last active square is gone from the map, first find minimal hits in the DP table, then sum up all states that has that many number of hits.
- Running time with nested maps is $\mathcal{O}(V \text{ argmax}|S|T_{max})$
- Since there are at most 6 active squares, the number of permutations of S is limited to approximately $\max\{\binom{N}{6} : 0 < N \leq 6\} = 20$, which is manageable
- Approximates the running time to $\mathcal{O}(VT_{max}) = \mathcal{O}(WH(200 + \max(W, H))) = \mathcal{O}(WH \max(W, H))$ (with a somewhat high constant factor)

Solved by no teams