ICPC Manila 2019

Solution Sketches

Payton Yao

Problem A: Suffix Three

- Straightforward implementation
- **`.endsWith`**`(...)`, or something similar.
- Or just implement yourself.

Payton Yao

Problem A: Suffix Three

- Taking a line, C++
  a. `getline(cin, s);`
- Taking a line, Java
  a. `s = bufferedReader.readLine();`
  b. `s = scanner.nextLine();`
- Taking a line, Python
  a. `s = input()`

Payton Yao

Problem A: Suffix Three

- Getting the last word, C++
  a. `while (cin >> s);`
  b. `while (scanf("%s", s) != EOF);`

Payton Yao

Problem A: Suffix Three

- Example:

```
if (s.endsWith(“po”)) ...
if (s.endsWith(“desu”) ||
    s.endsWith(“masu”)) ...
if (s.endsWith(“mnida”)) ...
```

Payton Yao

Problem A: Suffix Three

- **Faster** to code:

```
if (s.endsWith("o")) ...

if (s.endsWith("u")) ...

if (s.endsWith("a")) ...
```

Payton Yao

Problem A: Suffix Three

- Python one line:

```
print({'o': "FILIPINO",
       'u': "JAPANESE",
       'a': "KOREAN"}[input()[-1]])
```

Payton Yao

Problem A: Suffix Three

- No advanced machine learning algorithms needed!

Kevin Atienza

Problem I: A Case By Case Basis

- Classify the letters as uppercase or lowercase.
- An example of a **classification** problem
- Which is a **machine learning** problem
- Advanced machine learning algorithms needed
  - Just kidding!

Kevin Atienza

Problem I: A Case By Case Basis

- Encode all letters.
- Check everything.
- 52 letters to type.

Kevin Atienza

Problem I: A Case By Case Basis

- Encode upper ones only.
- Check which one matches.
- **26** letters to type.

Kevin Atienza

Problem I: A Case By Case Basis

- Compare the number of black cells.
- The larger one is uppercase... often.
  - Only need to type the exceptions.
- **3** letters to type! G, J, Y

Kevin Atienza

Problem I: A Case By Case Basis

- if $\mathbf{abs(first_1 - first_2) \geq 2}$:

  return *larger*

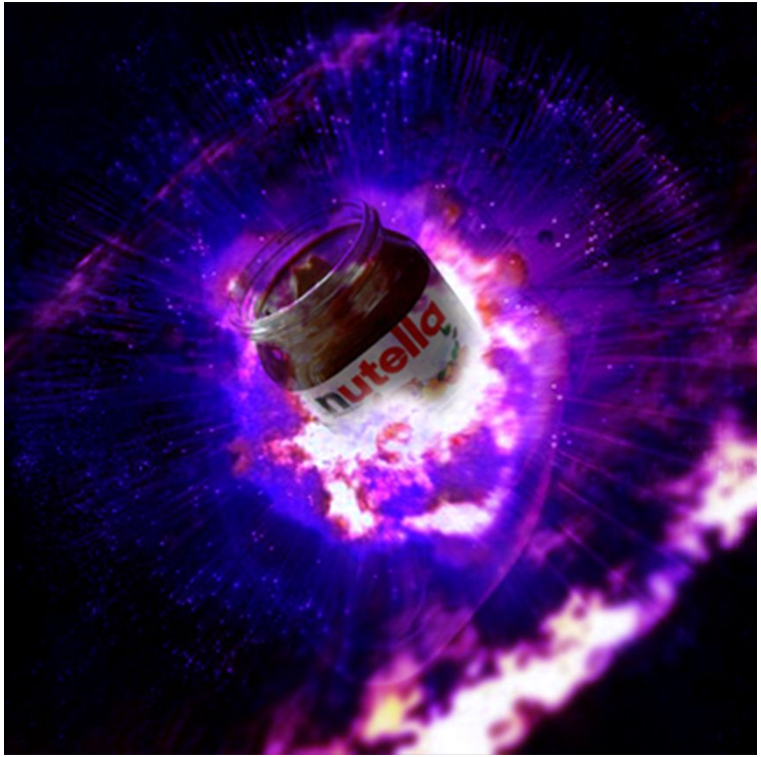
- if $\mathbf{last_1 \neq last_2}$:

  return *smaller*

- if $\mathbf{total_1 \neq total_2}$:

  return *larger*

- **0** letters to type!



Kevin Atienza

Problem M: Beingawesomeism

- If all **P**’s, impossible.
- Otherwise, can be done in **4** moves.
- Case analysis. Just need to know if 0, 1, 2, 3 or 4.

Kevin Atienza

Problem M: Beingawesomeism

- 0 if:
  - All **A**'s.
- 1 if:
  - At least one border has all **A**'s.
  - Otherwise, can be proven that > 1.

Kevin Atienza

Problem M: Beingawesomeism

- 2 if:
  - At least one corner **A**.
  - At least one full column or row of **A**’s.
  - Otherwise, can be proven > 2.
- 3 if:
  - At least one **A** in a border.
  - Otherwise, can be proven > 3.
- 4 otherwise.

Payton Yao

Problem D: Okkeika Ferry Co.

- Collapse connected components.
- Find smallest node in each component.
- Then greedy: Add the pair of components with the most requests.
    - Add the ferry between their smallest nodes.

Payton Yao

Problem D: Okkeika Ferry Co.

- Collapse connected components.
- Find smallest node in each component.
- Then greedy: Add the pair of components with the most requests.
    - Add the ferry between their smallest nodes.
- **Gotcha:** Disregard requests that are already fulfilled.

Payton Yao

Problem D: Okkeika Ferry Co.

- If all requests already OK, just find the smallest pair.
    - If no such pair (i.e., complete graph), impossible.

Payton Yao

Problem D: Okkeika Ferry Co.

- If all requests already OK, just find the smallest pair.
  - If no such pair (i.e., complete graph), impossible.
- **Gotcha:** O(n+m+k) not allowed!
  - Should be O(m+k).
  - Only consider the *mentioned* nodes.
    - Except if all requests are already fulfilled.

Jared Asuncion

Problem H: Kirchhoff's Current Loss

- Number theory?
- Sounds hard, so let's ignore "integer" restriction first.

Jared Asuncion

Problem H: Kirchhoff’s Current Loss

- If we’re not restricted to integers, then the optimal cost to obtain **r** is *linear* in **r**.
  - That is, **c*r** for some **c** > 0.
- We need to find **c** for our circuit.

Jared Asuncion

Problem H: Kirchhoff's Current Loss

- For **a single resistor**, $\mathbf{c} = 1$.
- For **series**, $\mathbf{c} = \min(\mathbf{c_1}, \mathbf{c_2})$.
- For **parallel**, $\sqrt{\mathbf{c}} = \sqrt{\mathbf{c_1}} + \sqrt{\mathbf{c_2}}$
    - Can be shown with calculus.
- This shows that $\sqrt{\mathbf{c}}$ is *always an integer.*
    - Can be proven with induction.

Jared Asuncion

Problem H: Kirchhoff's Current Loss

- √**c** is always an integer.
- Even stronger: WRT minimizing costs, **the circuit is equivalent to a parallel circuit with √c resistors**!
  - Can be proven inductively.
- Thus, we can replace exactly **√c** resistors with a resistance of **√c*r** (and the rest **0**) to achieve **r**.

Jared Asuncion

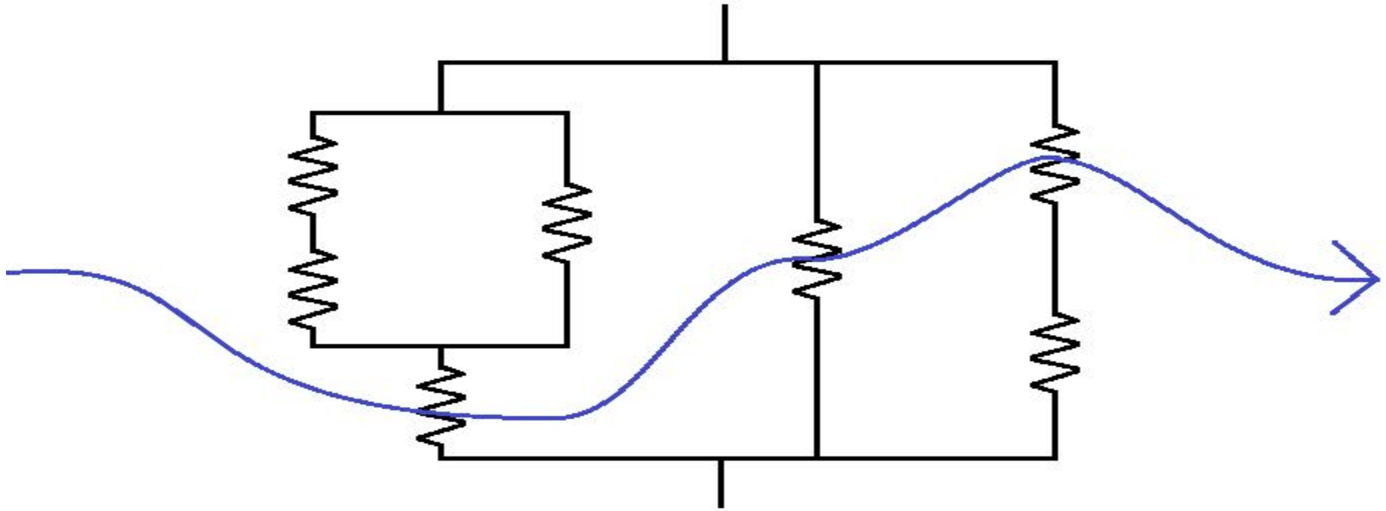
Problem H: Kirchhoff's Current Loss

- We can replace exactly **√c** resistors with a resistance of **√c*r** (and the rest **0**) to achieve **r**.
- But **√c** and **√c*r** are *integers*, so there's an optimal solution involving integers only.
- This also solves the integer case!

Jared Asuncion

Problem H: Kirchhoff's Current Loss

- **√c** can be interpreted as shortest path/min cut:
  - Draw the circuit.
  - Then the shortest path from left to right is **√c**.



Jared Asuncion

Problem H: Kirchhoff's Current Loss

- Needs some parsing.
- Just use any standard infix-to-postfix algorithm.
- ```
  eval(s.replace('*','Resistor()')
          .replace('S','+').replace('P','*'))
  ```
- Just kidding, that won't work!
    - Why?
  ```

Kevin Atienza

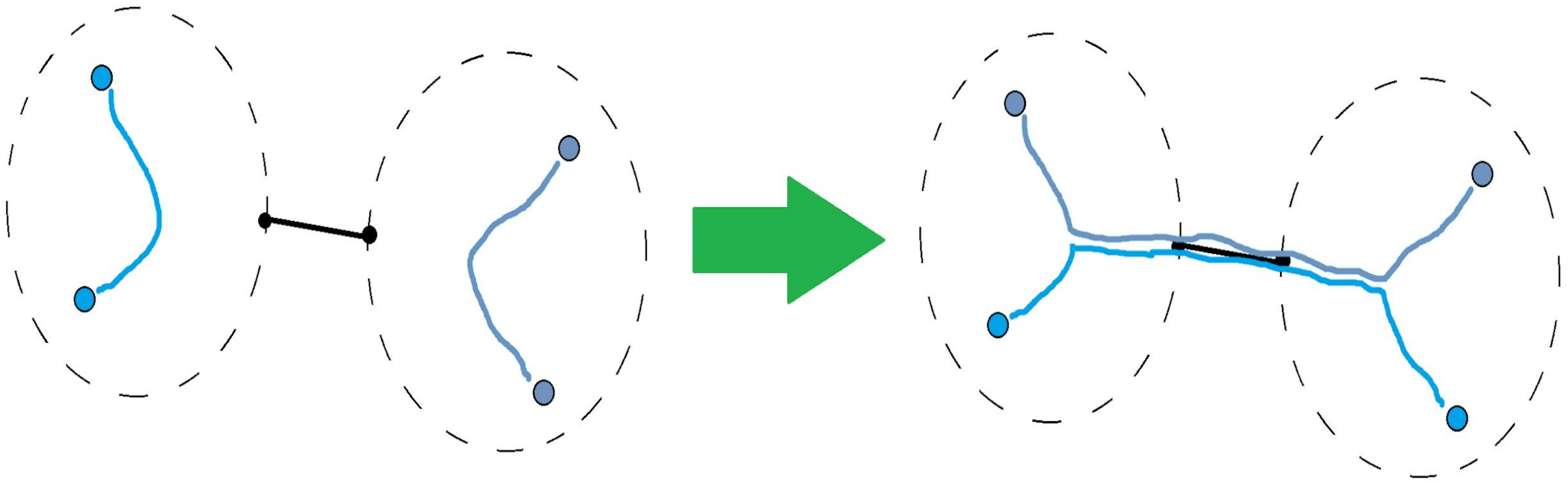
Problem L: Jeremy Bearimy

- DP?
- There’s no obvious one, I think.

Kevin Atienza

Problem L: Jeremy Bearimy

- (Maximization) Observation:



Kevin Atienza

Problem L: Jeremy Bearimy

- Thus, one component must be all matched to the other component.
- Also, this is the maximum possible contribution for an edge.
- So, in the optimal solution, an edge contributes **min(size[left], size[right]) * weight**.
- But the same is true for all edges!

Kevin Atienza

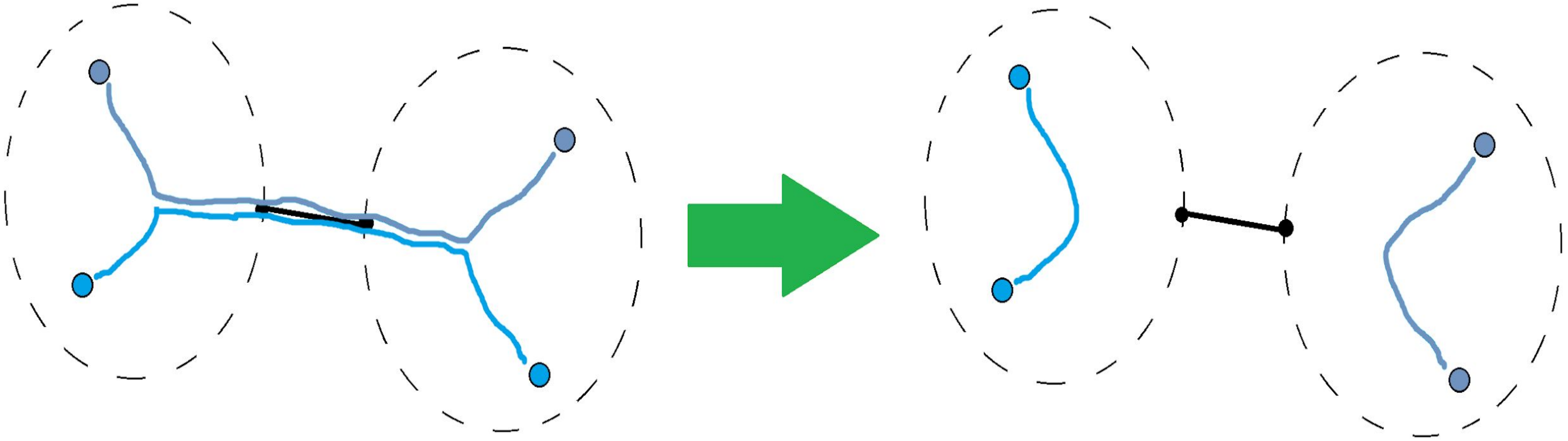
Problem L: Jeremy Bearimy

- Every edge contributes **min(size[left], size[right]) * weight** in the optimal solution.
- O(n): compute the sizes of all subtrees, then just add up all contributions!

Kevin Atienza

Problem L: Jeremy Bearimy

- (Minimization) Observation:



Kevin Atienza

Problem L: Jeremy Bearimy

- Thus, an edge contributes **at most 1** in the optimal solution.
- Also, the parity of an edge's contribution is fixed.
- So, in the optimal solution, an edge contributes **(size[left] mod 2)** * **weight**.
- But the same is true for all edges!

Kevin Atienza

Problem L: Jeremy Bearimy

- Every edge contributes **(size[left] mod 2) * weight** in the optimal solution.
- O(n): compute the **parity** of sizes of all subtrees, then just add up all contributions!

Kyle See

Problem E: Do You Wanna Build More Snowmen?

- Adjust allowed moves slightly: Only allow “animating” the *top* of the stack.
- Can be shown equivalent, but slightly easier to think about.

Kyle See

Problem E: Do You Wanna Build More Snowmen?

- It smells of DP, but what are the states?
- Typical DP approach seems to fail, since we can stack arbitrarily many prefixes of words.

Kyle See

Problem E: Do You Wanna Build More Snowmen?

- Correct state: **(substring, prefix of some word)**.
  - $O(n^3)$ states.
- Transition: The next letter in the word can be anywhere in the substring.
  - Recurse on the two substrings.
  - $O(n)$ transition.
- **$O(n^4)$** overall. (actually, closer to $O(n^4/6)$.)

Payton Yao

Problem C: JaBloo 11: Lord of Expansion

- Observation: “Can defeat” graph is acyclic.
- Thus, any selection of attacks can be fulfilled.
  - Lower levels attack first.
- For a fixed **a**, it’s just a **maximum matching** between attackers and defenders.

Payton Yao

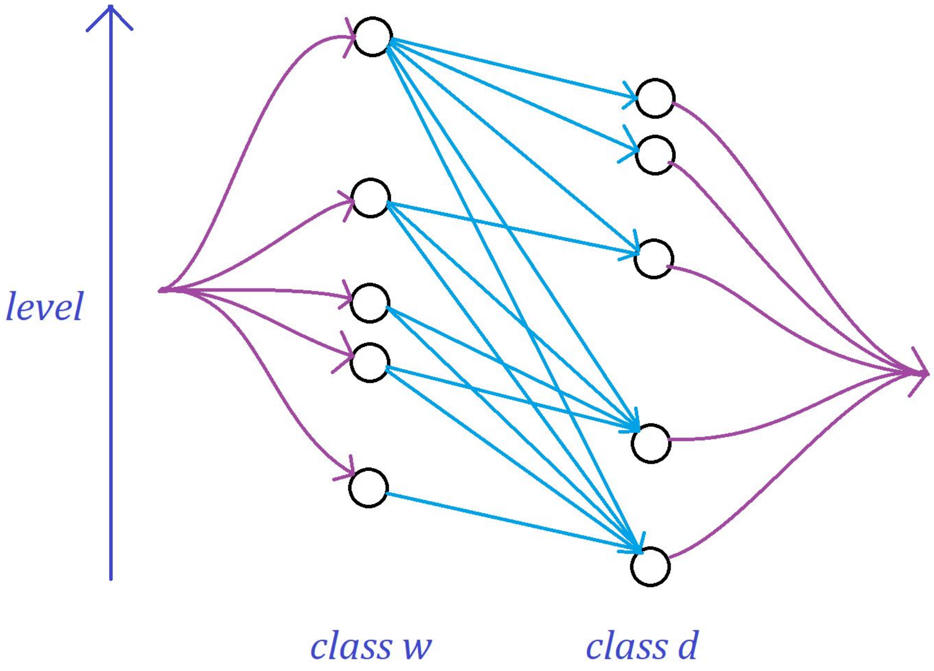
Problem C: JaBloo 11: Lord of Expansion

- Maximum matching? **Max flow**.
  - Make source **s** and sink **t**.
  - For each person **p**, make two nodes $\mathbf{p}_{att}$ and $\mathbf{p}_{def}$.
  - Add edge $\mathbf{p}_{att} \rightarrow \mathbf{q}_{def}$ if **p** can defeat **q**.
  - Add $\mathbf{s} \rightarrow \mathbf{p}_{att}$ with capacity **a**.
  - Add $\mathbf{p}_{def} \rightarrow \mathbf{t}$ with capacity **1**.

Payton Yao

Problem C: JaBloo 11: Lord of Expansion

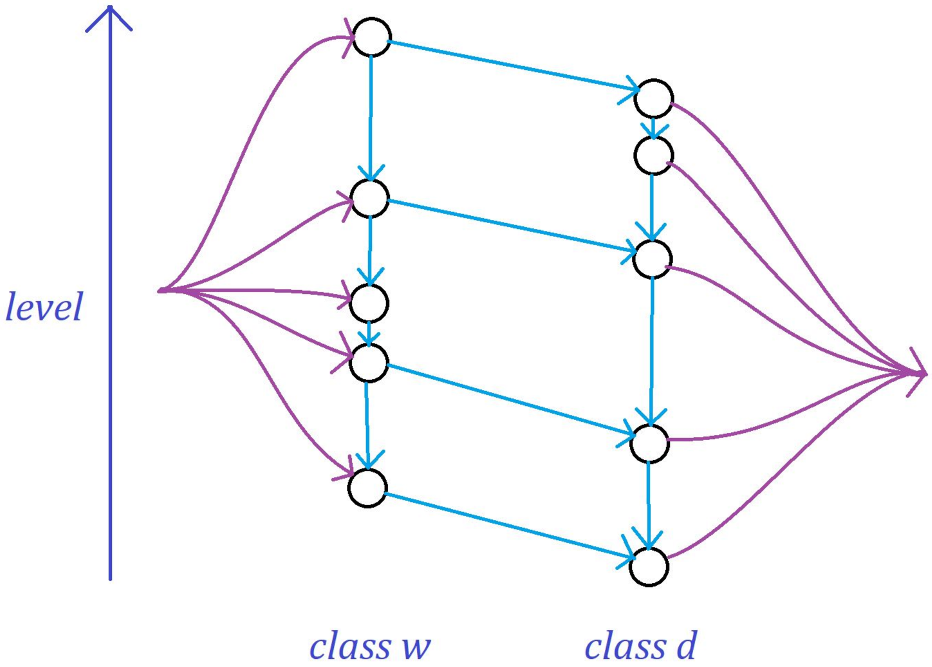
- Problem: $\Omega(V^2)$ edges.
- Dinic?
  - No, doesn't help.
- Push-Relabel?
  - No. Even worse.
- We seem forced to have $\Omega(VE)$ runtime here.



Payton Yao

Problem C: JaBloo 11: Lord of Expansion

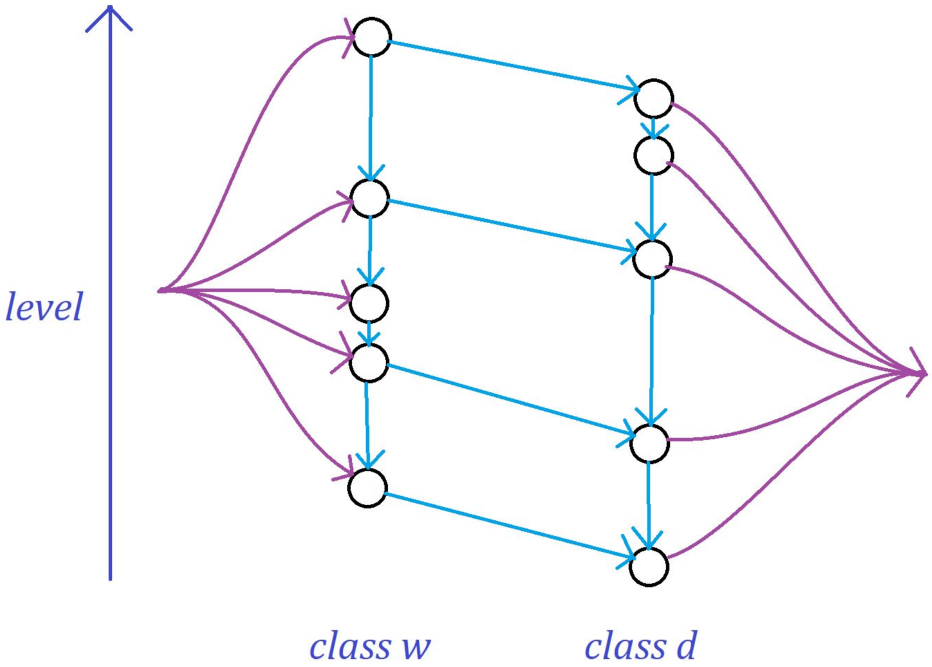
- We seem forced to have $\Omega(VE)$ runtime.
- **But we can reduce E !!**



Payton Yao

Problem C: JaBloo 11: Lord of Expansion

- Analysis shows $E = O(Vk^{1/2})$.
- So overall, $\mathbf{O(V^2k^{1/2})}$.
- TL generous enough to allow $E = O(Vk)$.
- Or overall, $O(V^2k)$.
    - i.e., I can't break it :P



Payton Yao

Problem B: Miss Punyverse

- DP? Given subtree and # of regions, find the maximum # of winning regions.
- **Problem:** Need to also “merge” with the topmost component, so its “vote advantage” at the top matters.

Payton Yao

Problem B: Miss Punyverse

- Greedy observation: It is optimal to maximize the # of winning regions *first*, then maximize the vote advantage of the root *second*.
  - Convince yourself that
    - **(x winning regions, -∞ vote advantage)**
  - is better than
    - **(x-1 winning regions, +∞ vote advantage)**

Payton Yao

Problem B: Miss Punyverse

- Convince yourself that
  - **(x winning regions, -∞ vote advantage)**
- is better than
  - **(x-1 winning regions, +∞ vote advantage)**
- The +∞ vote advantage, at best, can increase the winning regions by 1, but we can already achieve that even with (x, -∞).

Payton Yao

Problem B: Miss Punyverse

- So the DP now becomes: Given subtree and # of regions, find the maximum # of winning regions, and among all such possibilities, find the maximum vote advantage of the root component.
  - Some edge cases to consider, e.g., don’t construct size-0 partitions!

Payton Yao

Problem B: Miss Punyverse

- DP(node, #regions).
- Here, **#regions ≤ size(node)**
- Total transition for “node” is **O(size(left)*size(right))**.
- This is the DP pattern that looks like O(n^3) but is actually **O(n^2)**.

Kyle See

Problem F: Marking the Territory

- “2D binary search”.
- First step: Identify which cells are markable.
- Second step: Find a “fast” way to mark it.

Kyle See

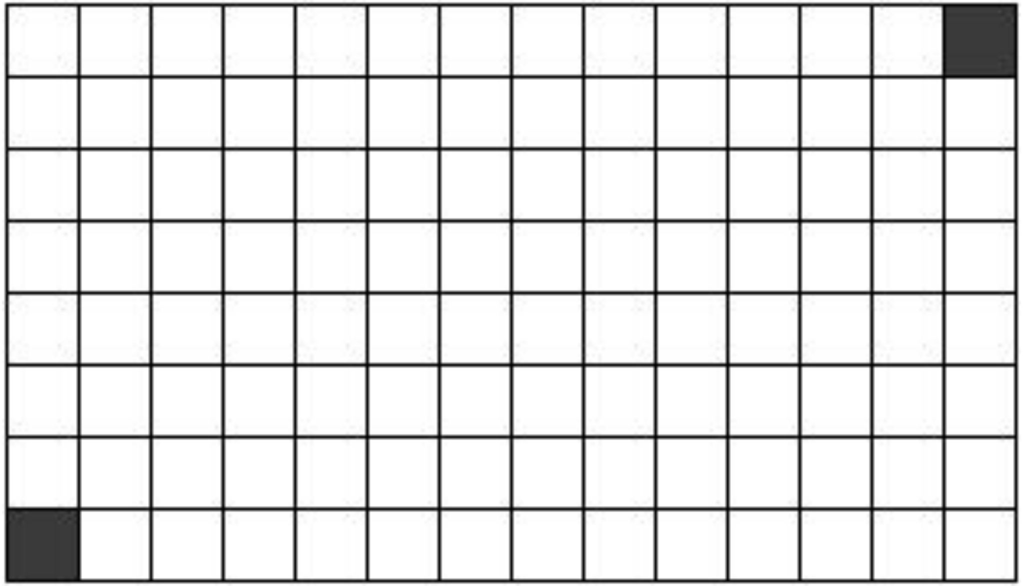
Problem F: Marking the Territory

- If bounding box is not a line:
  - If there are at least two same-parity cells, then all cells can be marked!

Kyle See

Problem F: Marking the Territory

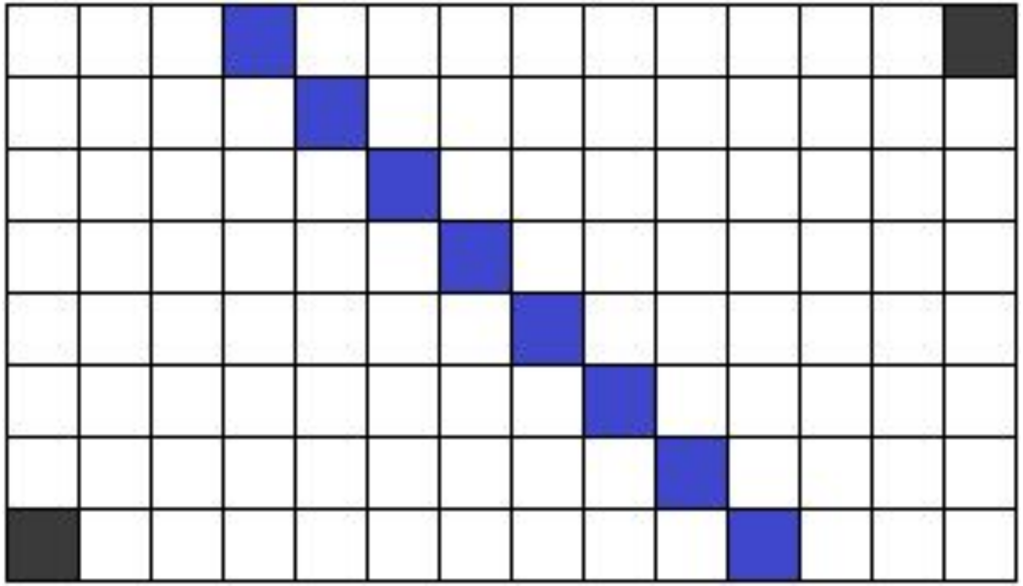
- If bounding box is not a line:
    - If there are at least two same-parity cells...



Kyle See

Problem F: Marking the Territory

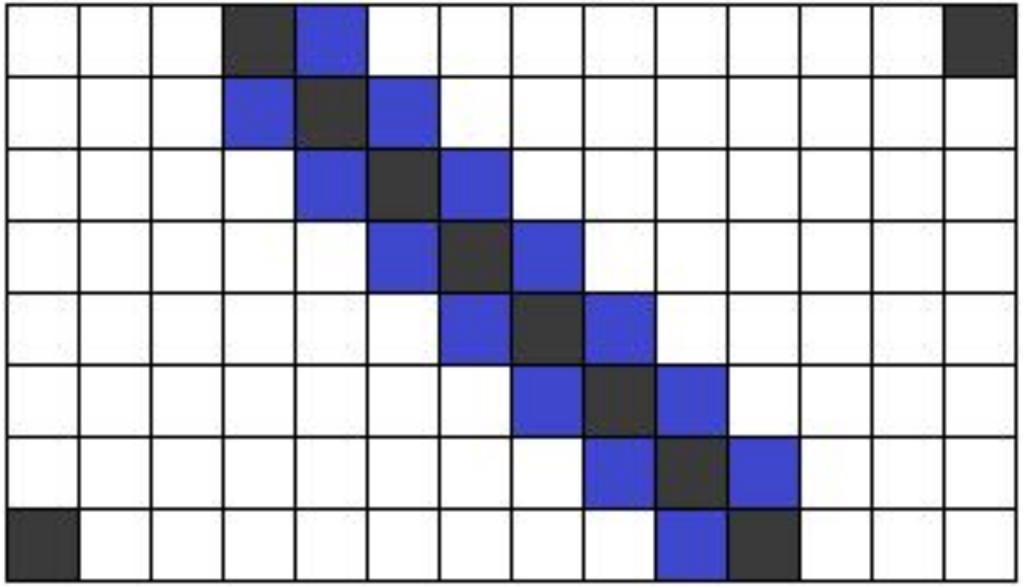
- If bounding box is not a line:
  - If there are at least two same-parity cells...



Kyle See

Problem F: Marking the Territory

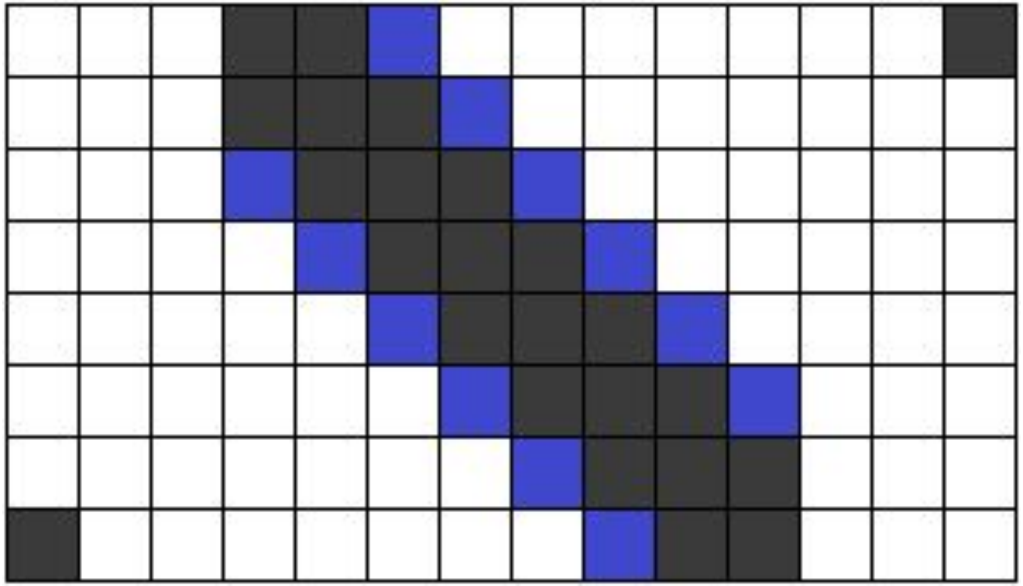
- If bounding box is not a line:
  - If there are at least two same-parity cells...



Kyle See

Problem F: Marking the Territory

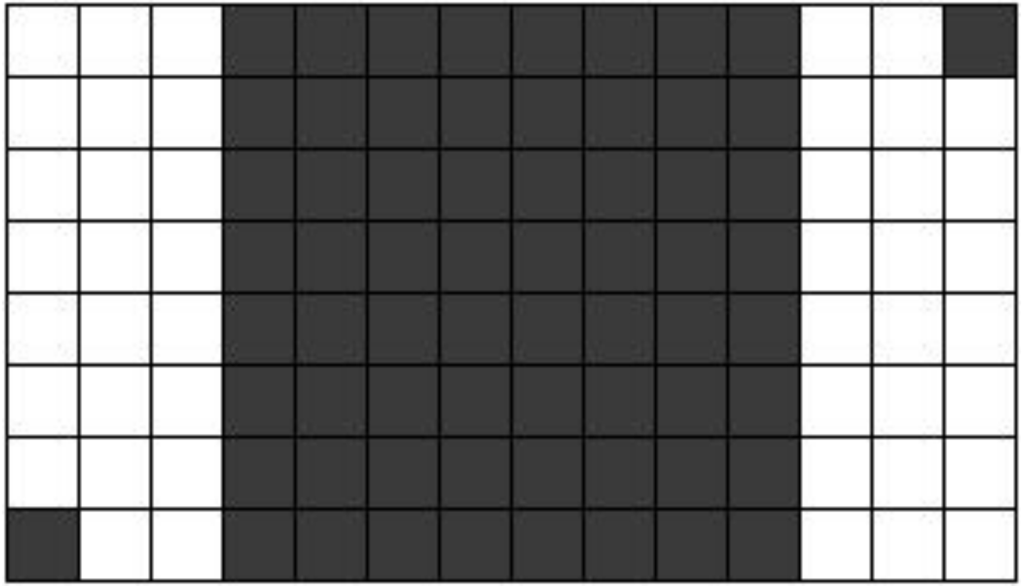
- If bounding box is not a line:
  - If there are at least two same-parity cells...



Kyle See

Problem F: Marking the Territory

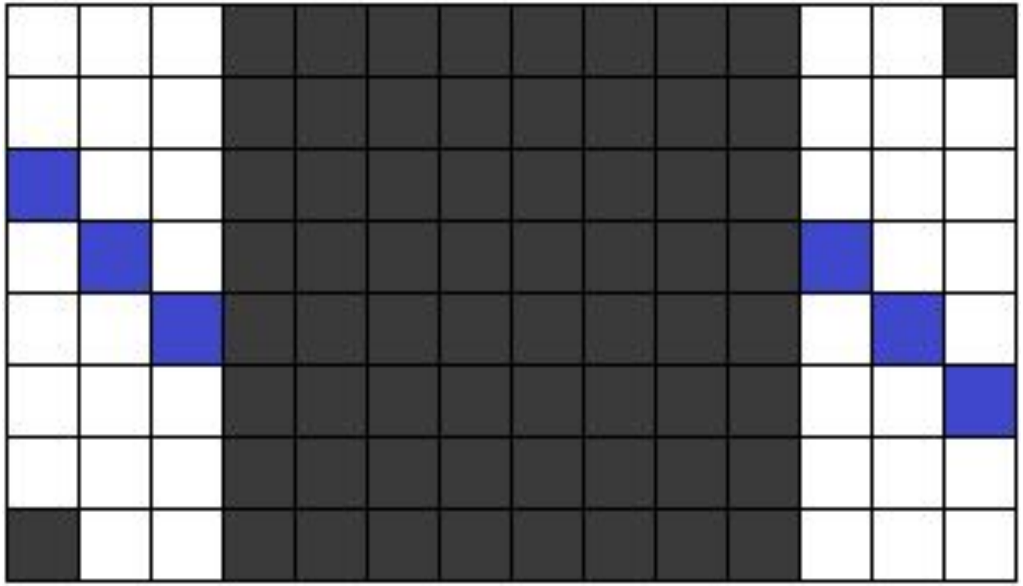
- If bounding box is not a line:
  - If there are at least two same-parity cells...



Kyle See

Problem F: Marking the Territory

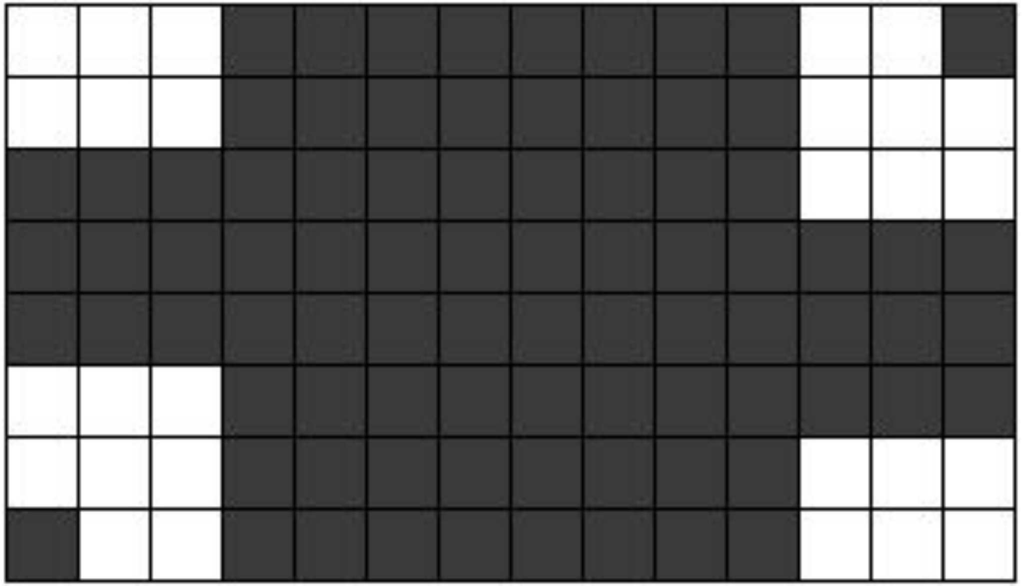
- If bounding box is not a line:
  - If there are at least two same-parity cells...



Kyle See

Problem F: Marking the Territory

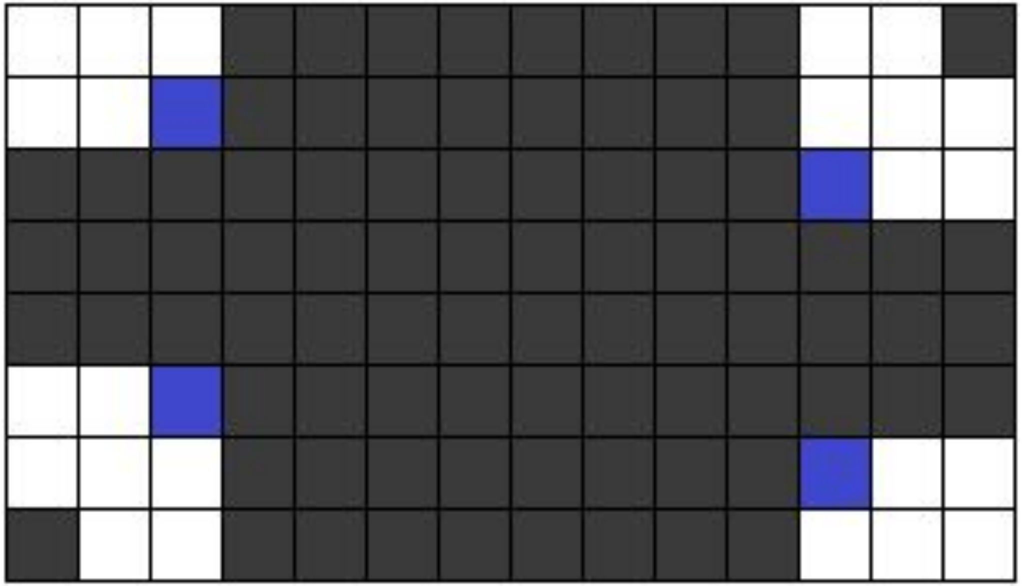
- If bounding box is not a line:
    - If there are at least two same-parity cells...



Kyle See

Problem F: Marking the Territory

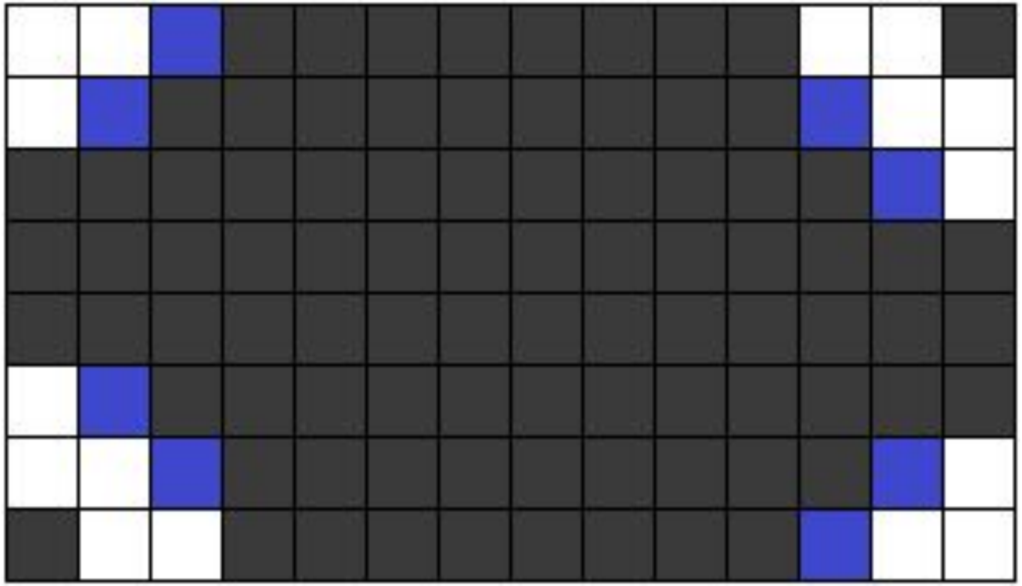
- If bounding box is not a line:
  - If there are at least two same-parity cells…



Kyle See

Problem F: Marking the Territory

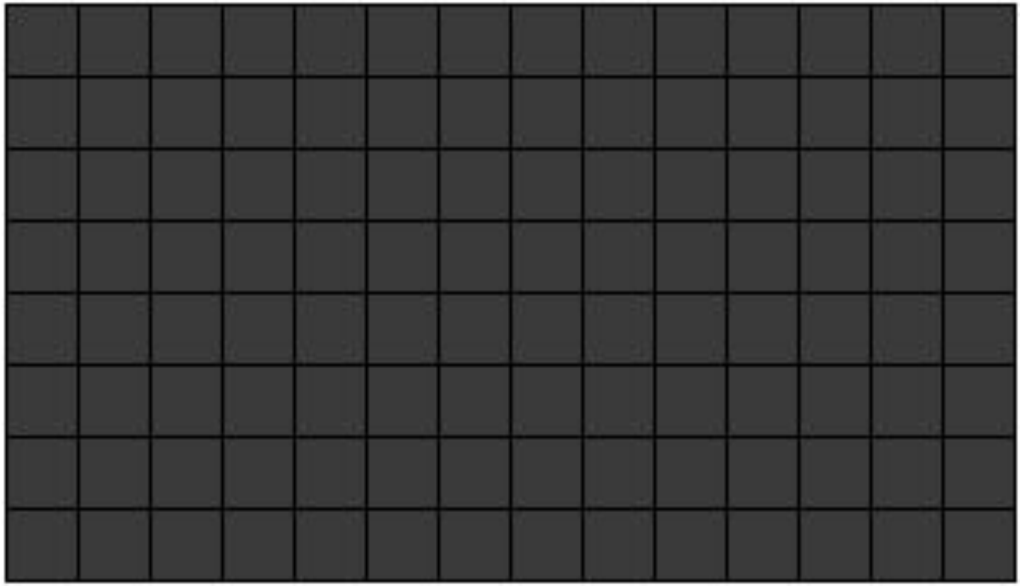
- If bounding box is not a line:
  - If there are at least two same-parity cells...



Kyle See

Problem F: Marking the Territory

- If bounding box is not a line:
  - If there are at least two same-parity cells...



Kyle See

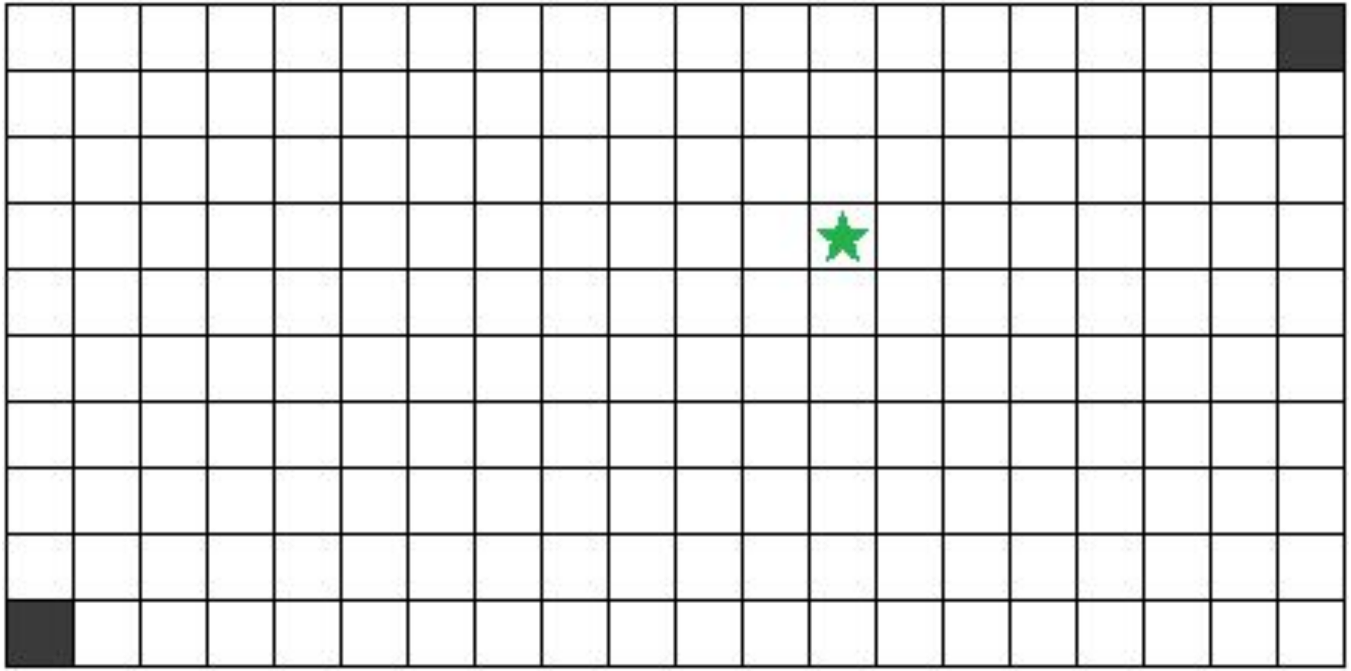
Problem F: Marking the Territory

- If bounding box is not a line:
  - If there are at least two same-parity cells, then all cells can be marked!
- But how to do it “fast”?
  - Use some sort of “2D” binary search.
  - Several edge cases. We won’t discuss all of them here.

Kyle See

Problem F: Marking the Territory

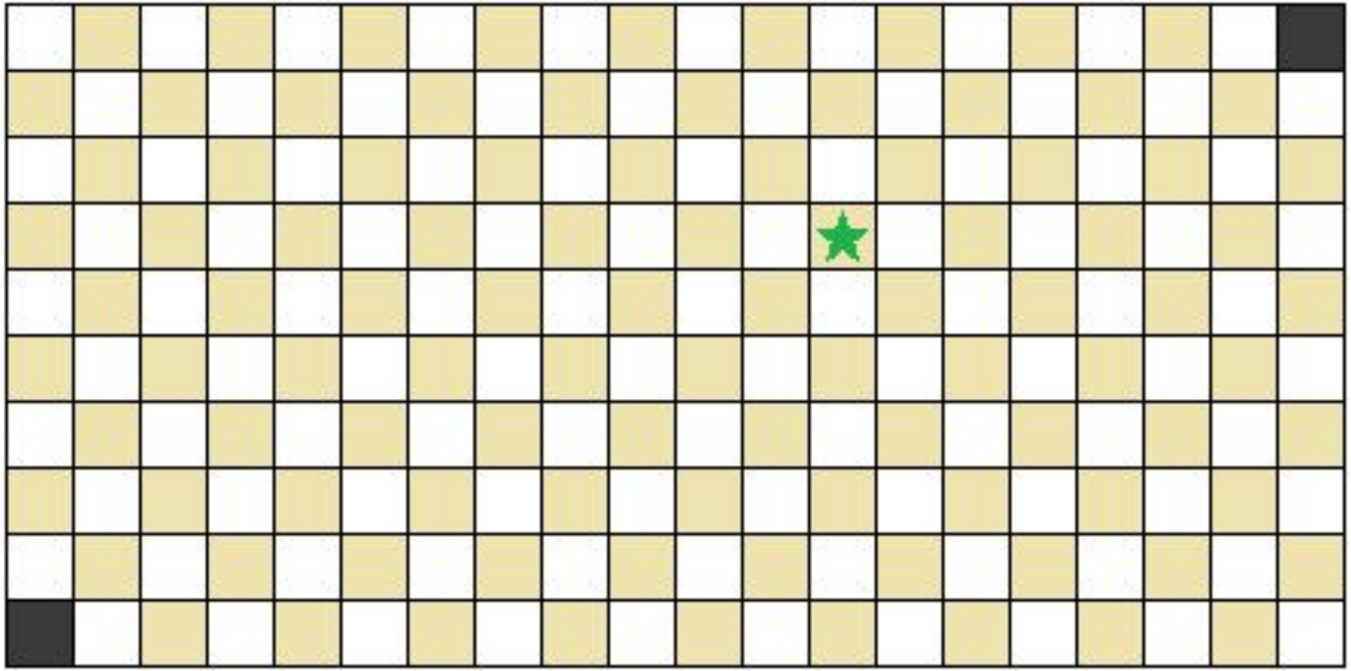
- If bounding box is not a line:



Kyle See

Problem F: Marking the Territory

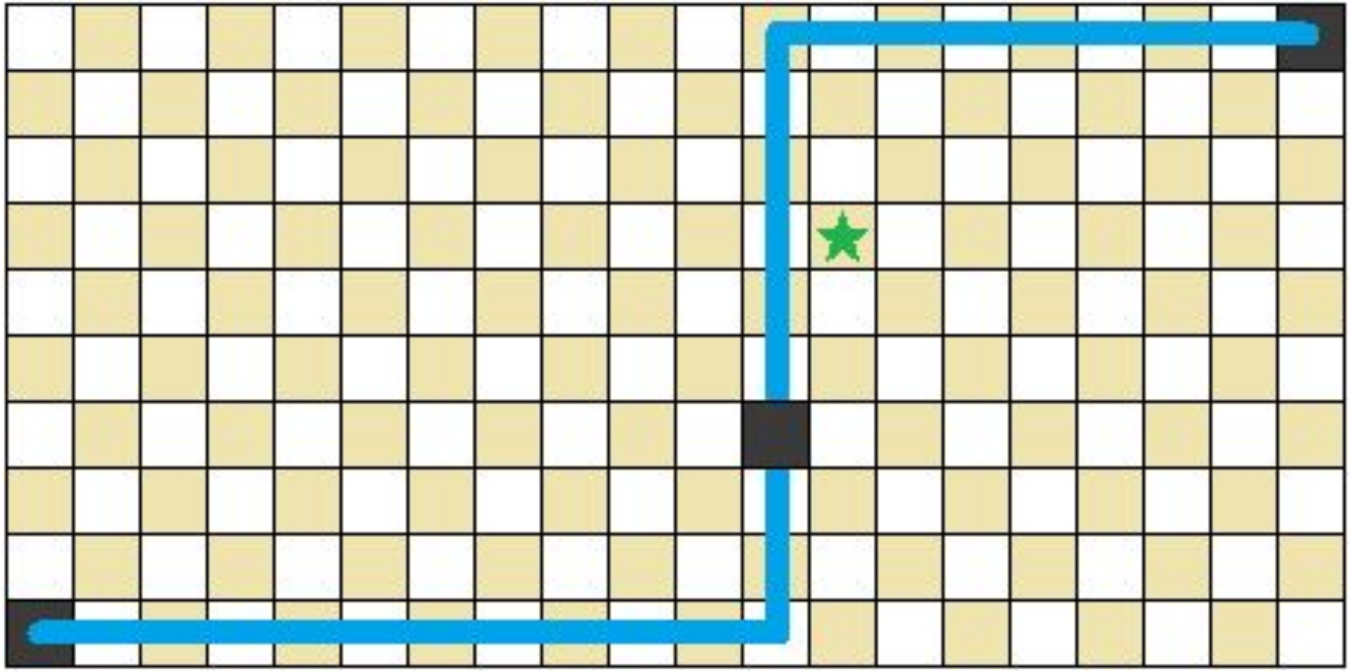
- If bounding box is not a line:



Kyle See

Problem F: Marking the Territory

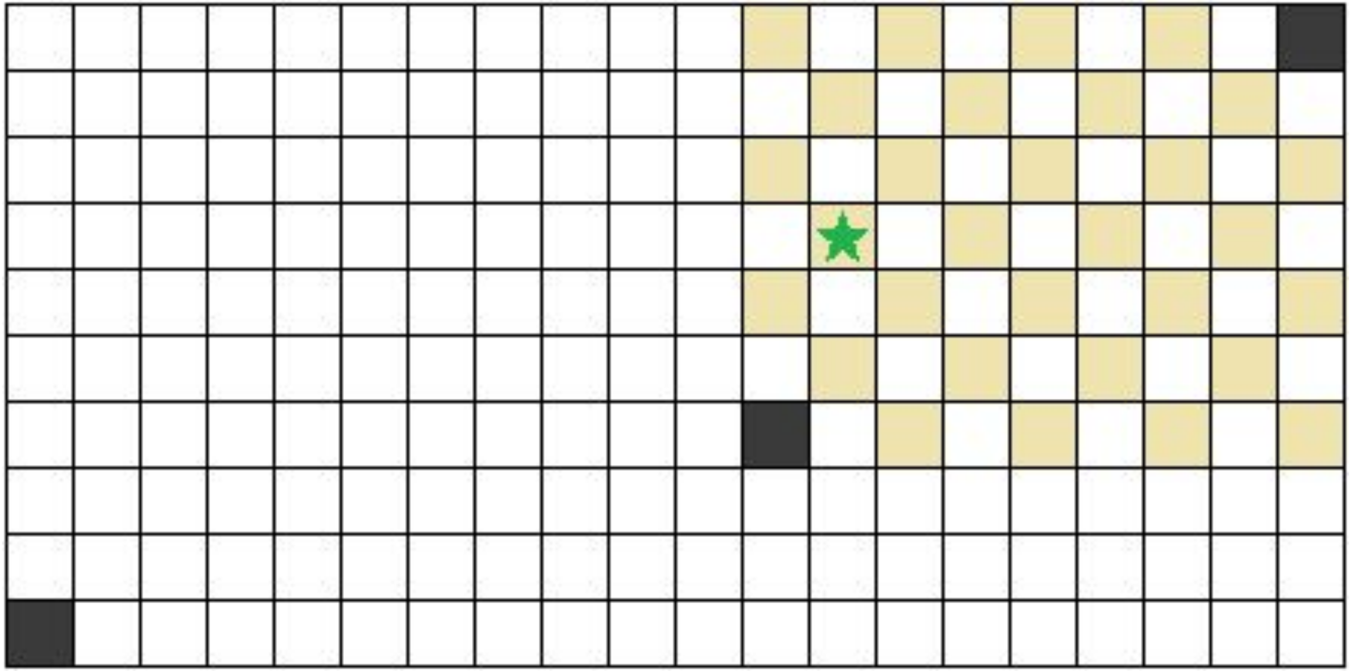
- If bounding box is not a line:



Kyle See

Problem F: Marking the Territory

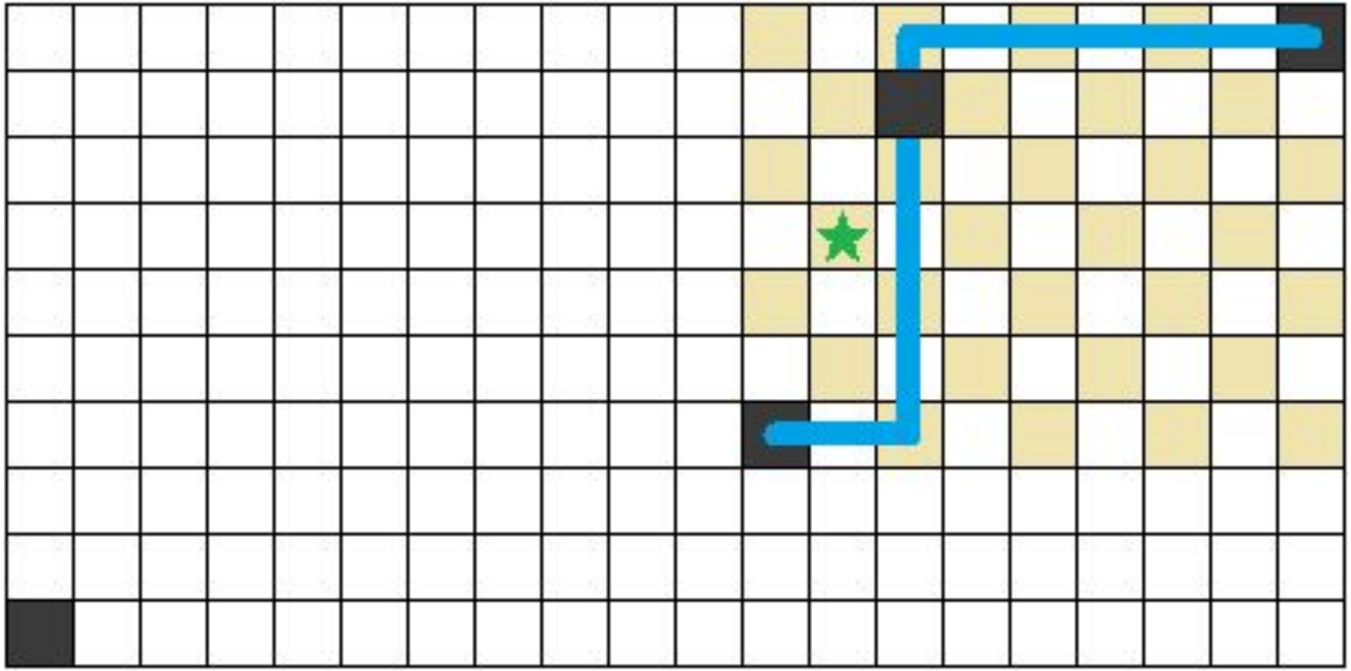
- If bounding box is not a line:



Kyle See

Problem F: Marking the Territory

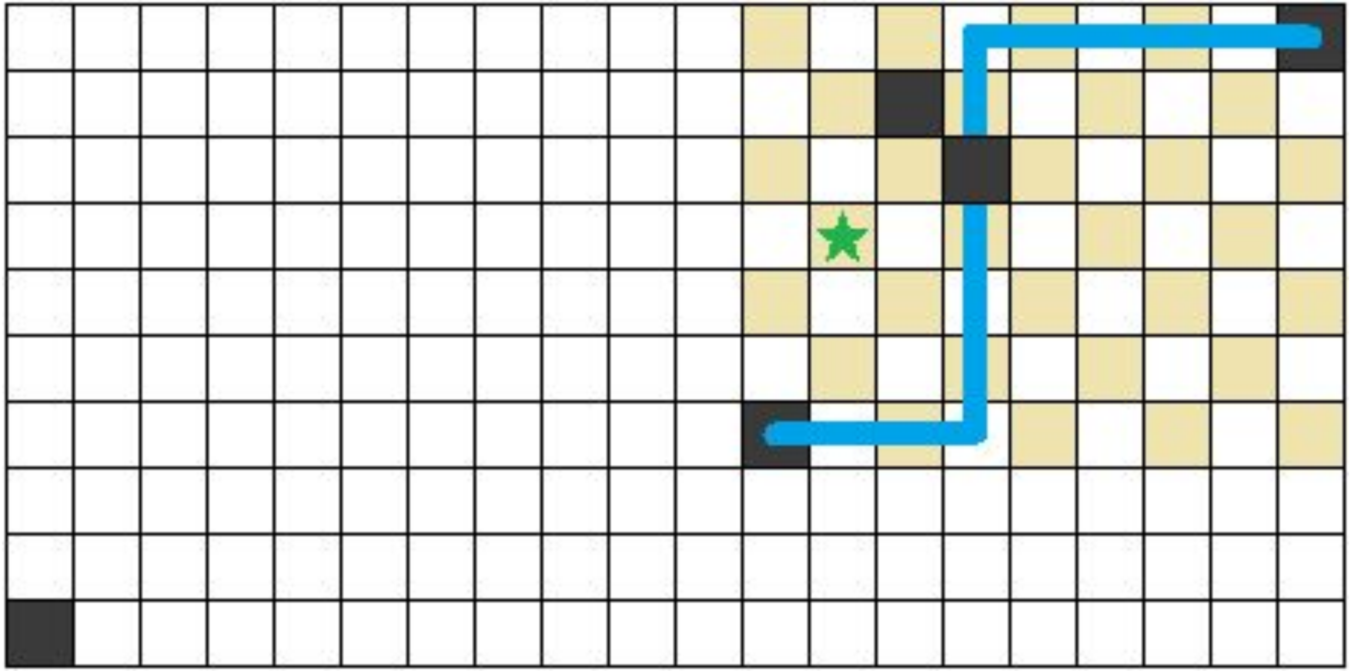
- If bounding box is not a line:



Kyle See

Problem F: Marking the Territory

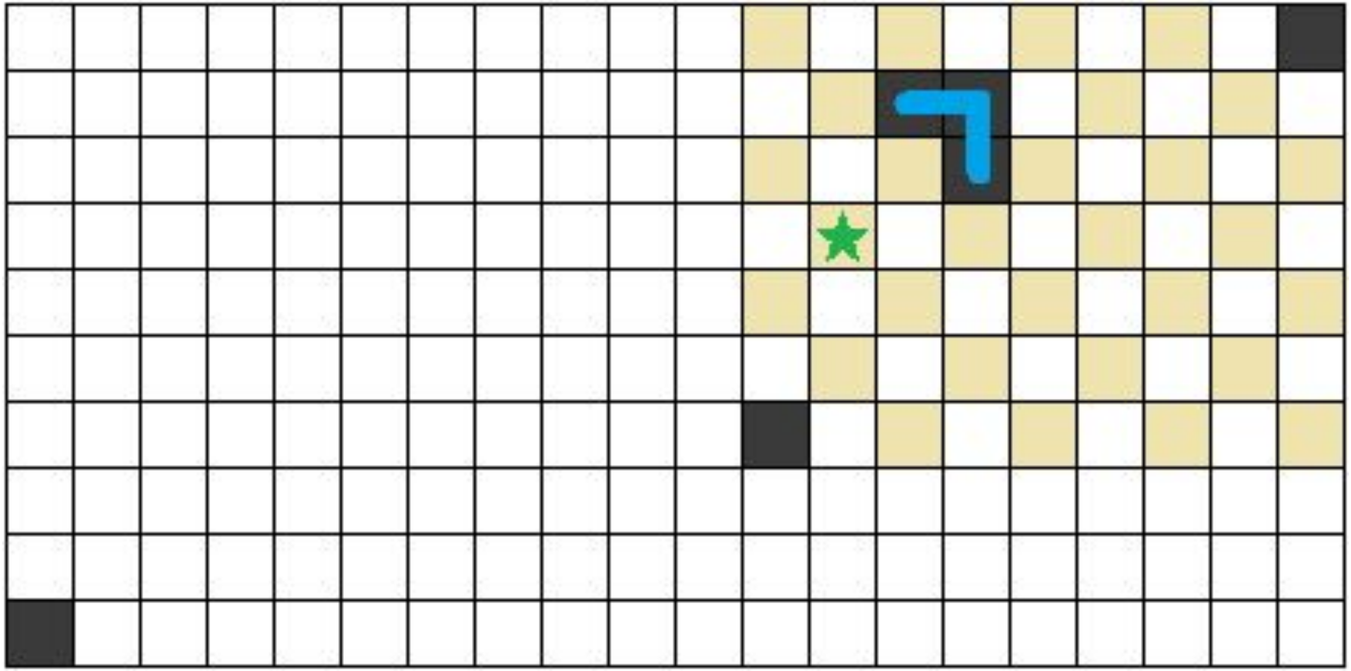
- If bounding box is not a line:



Kyle See

Problem F: Marking the Territory

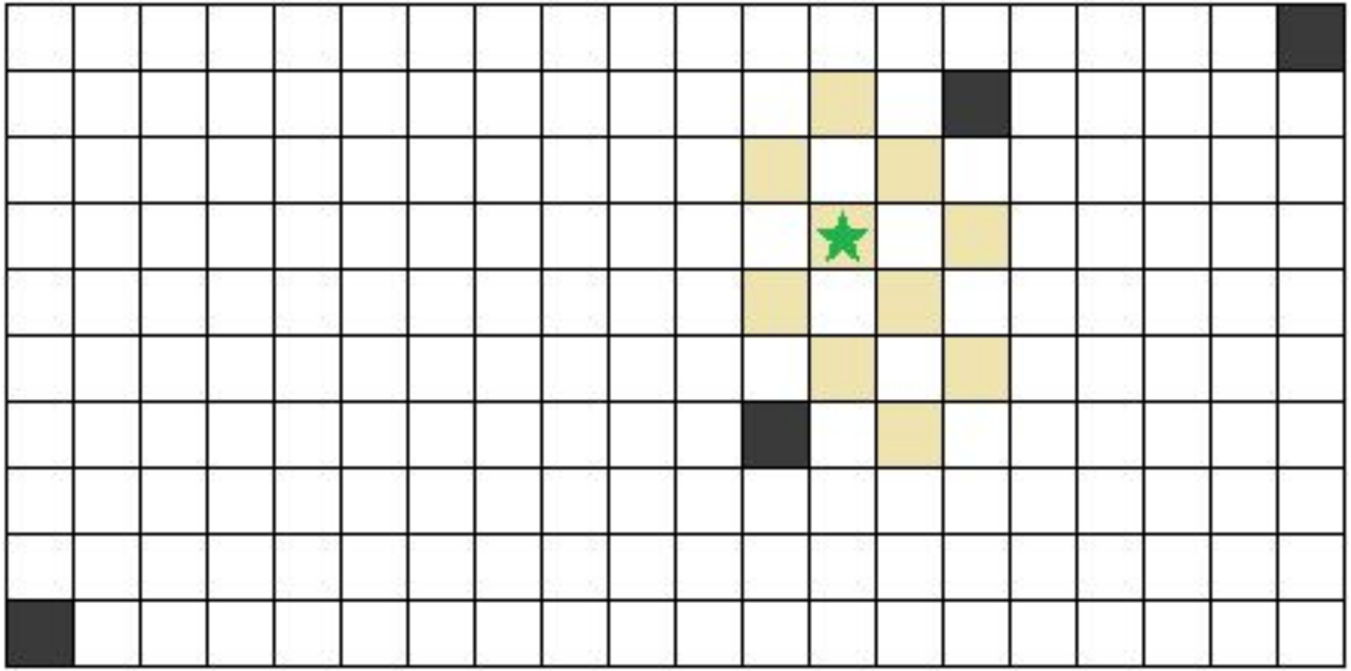
- If bounding box is not a line:



Kyle See

Problem F: Marking the Territory

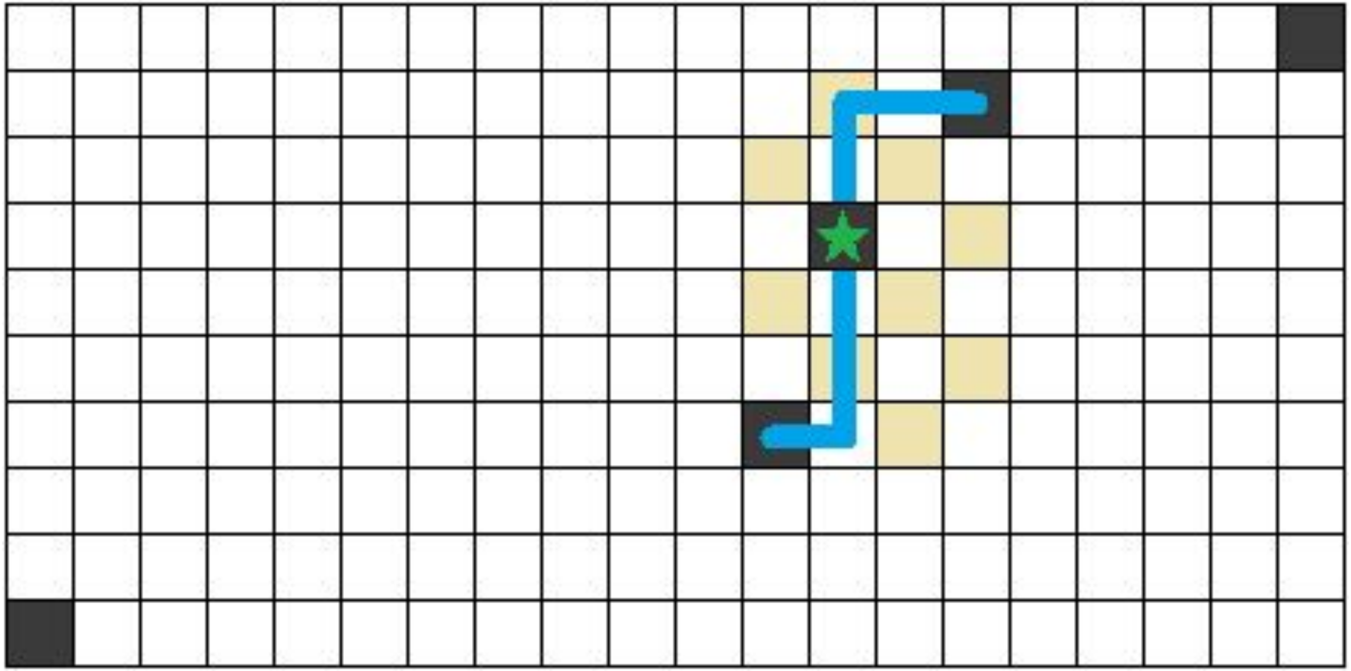
- If bounding box is not a line:



Kyle See

Problem F: Marking the Territory

- If bounding box is not a line:



Kyle See

Problem F: Marking the Territory

- If bounding box is a line:
  - Take the gcd of all distances, and remove all factors 2.
  - Then it's clear that only multiples of this distance can be marked.

Kyle See

Problem F: Marking the Territory

- If bounding box is a line:
  - Take the gcd of all distances, and remove all factors 2.
  - Then it’s clear that only multiples of this distance can be marked.
  - Surprisingly, **all** multiples of this distance can be marked!

Kyle See

Problem F: Marking the Territory

- For x > 3, the case with “x” marked cells is reducible to “x-1” marked cells:
    - Construct the gcd of the first x-1, then replace the middle x-2 cells with the gcd.
- Base cases:
    - 2 marked cells. Just binary search.
    - 3 marked cells. Also “binary search”, but a bit more complicated.

Kyle See

Problem F: Marking the Territory

- Worst case number of moves:

$$\approx \frac{1}{2} \cdot (n-2) \cdot \lg^2 \max(r, c) \approx 2000$$

Tim Dumol

Problem G: Guizzmo and the CSS

- You might think you need to construct complicated regions and find some kind of shortest path through it.
- Too complex! Also, too slow!

Tim Dumol

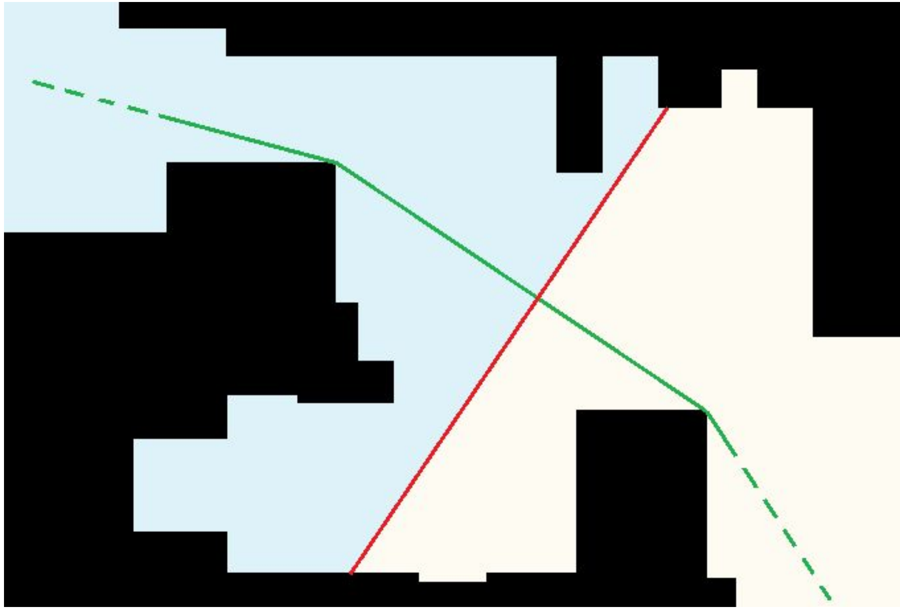
Problem G: Guizzmo and the CSS

- Crucial insight: If a laser blocks the **shortest path**, then it blocks *all* paths.

Tim Dumol

Problem G: Guizzmo and the CSS

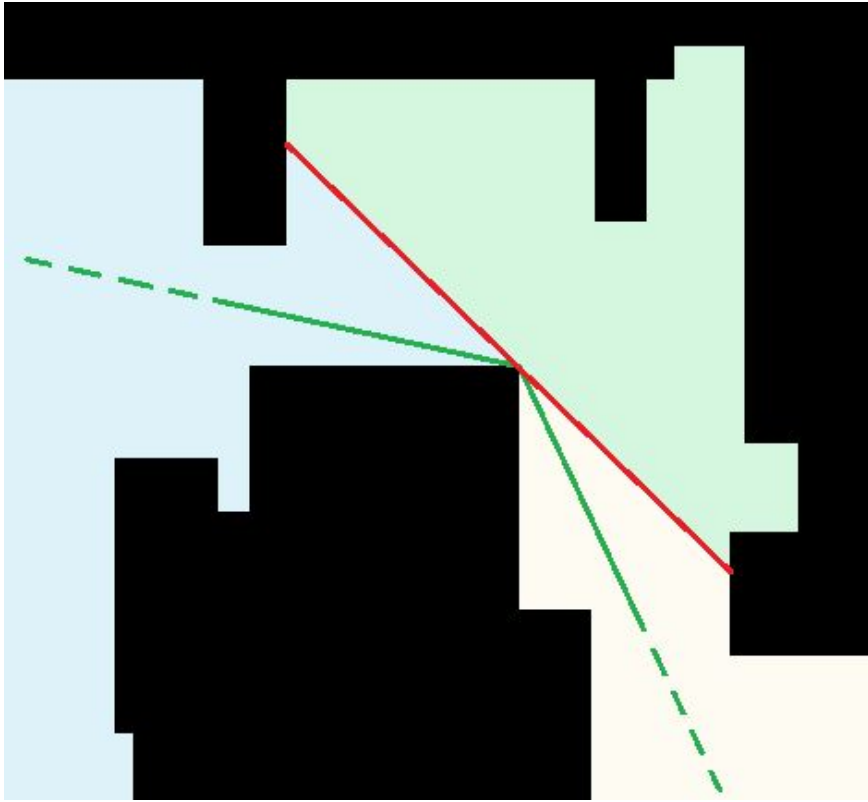
- Reason: Each straight laser section only intersects the shortest path in at most 1 location, so it separates the endpoints into 2 regions.



Tim Dumol

Problem G: Guizzmo and the CSS

- Even on “bends”. If a laser hits a bend, the endpoints are still in distinct regions, and even worse, there are at least 3 regions!



Tim Dumol

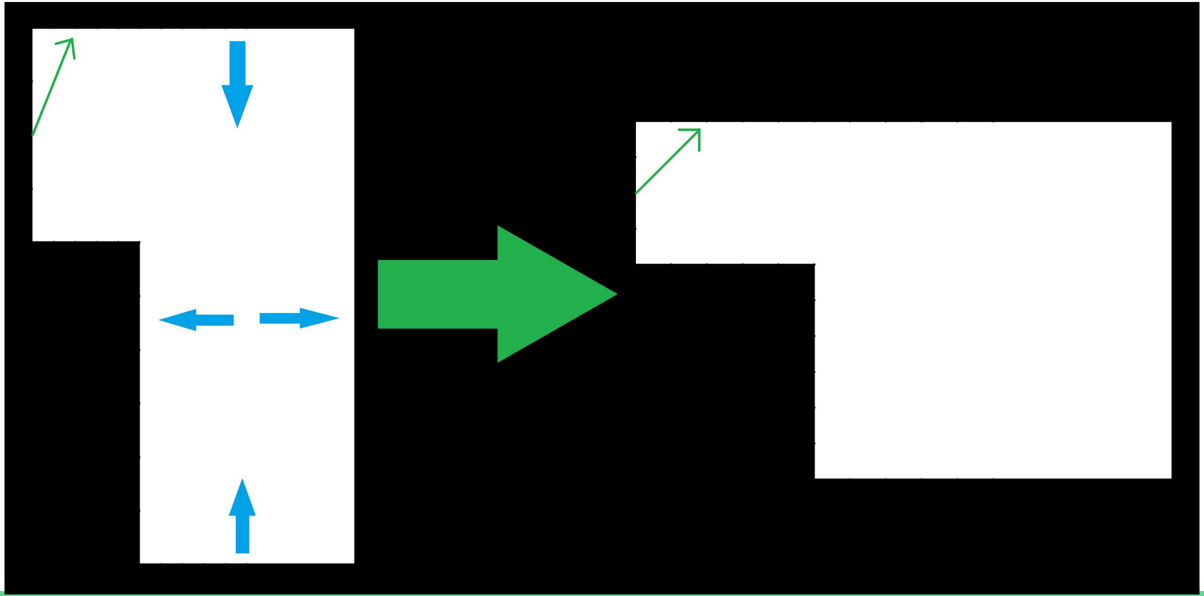
Problem G: Guizzmo and the CSS

- So the solution is simply **all lasers that intersect the shortest path**!
- The lasers can now be considered **independently**!
  - It is crucial that laser sources are not on corners.
- The problem is reduced to just computing the laser paths and intersecting with the shortest path.
- Shortest path is just Dijkstra + Geom. $O(n^3)$ is OK.

Tim Dumol

Problem G: Guizzmo and the CSS

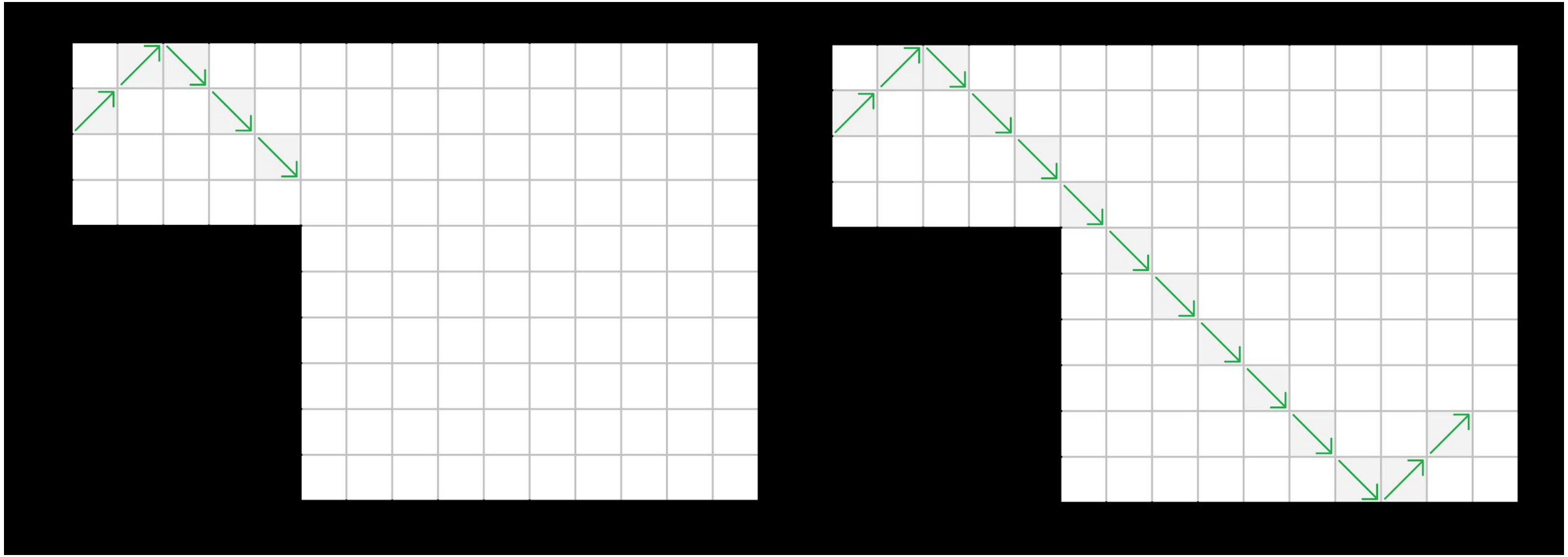
- Computing laser paths, the “cheap” way: Scale to make laser direction 45 degrees.



Tim Dumol

Problem G: Guizzmo and the CSS

- Then just walk.



- I call this the “**poor man’s ray tracing algorithm**”

Tim Dumol

Problem G: Guizzmo and the CSS

- The final step involves intersecting lasers and shortest path.
  - **O(laser_path_sections * shortest_path_sections)**
  - Can be improved to **O(laser_path_sections + shortest_path_sections)** with another insight, involving the topology of the room's boundary (i.e., a circle), i.e., "interleaving".

Kevin Atienza

Problem K: Cut and Paste

- Represent S with a **binary tree**. Initially, completely balanced.

Kevin Atienza

Problem K: Cut and Paste

- Represent S with a **binary tree**. Initially, completely balanced.
- Then **just simulate**, until the tree becomes at least x in size.

Kevin Atienza

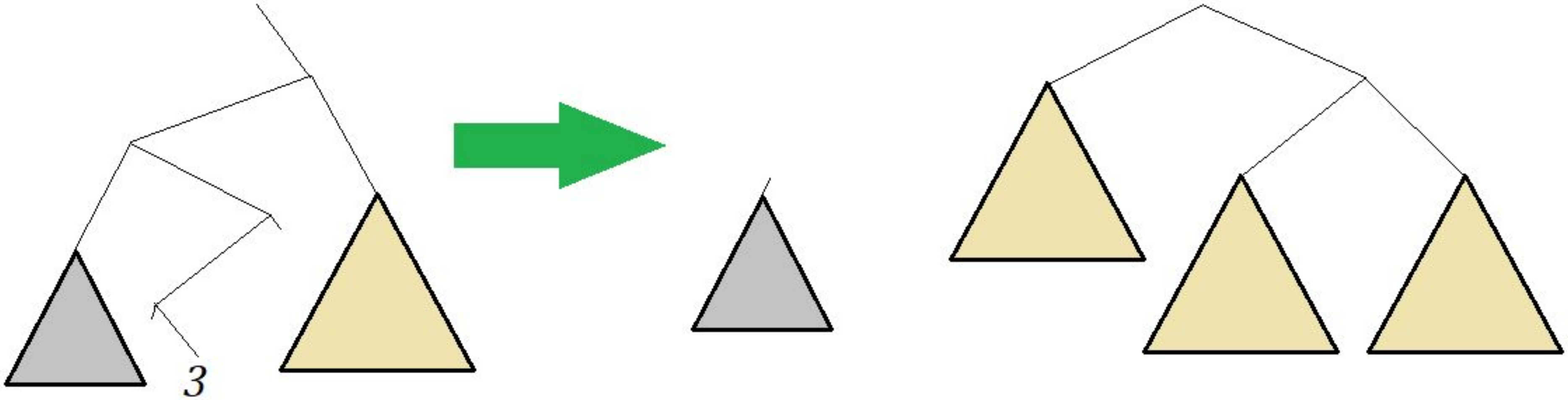
Problem K: Cut and Paste

- For each node, store its size and number of characters ≥ 2.
- Also, need to quickly be able to find the first character ≥ 2.
  - O(height) operation, given the stored values.

Kevin Atienza

Problem K: Cut and Paste

- After finding the first character ≥ 2, split the tree there, then duplicate/triplicate the right subtree.



Kevin Atienza

Problem K: Cut and Paste

- The height of the tree is always O(log x).
- Therefore, overall **O(n log x)** .....
- Any questions? Violent reactions?

Kevin Atienza

Problem K: Cut and Paste

- Well, maybe more like **O(n (log x + D))**
- where **D** = cost of duplicating a tree.

Kevin Atienza

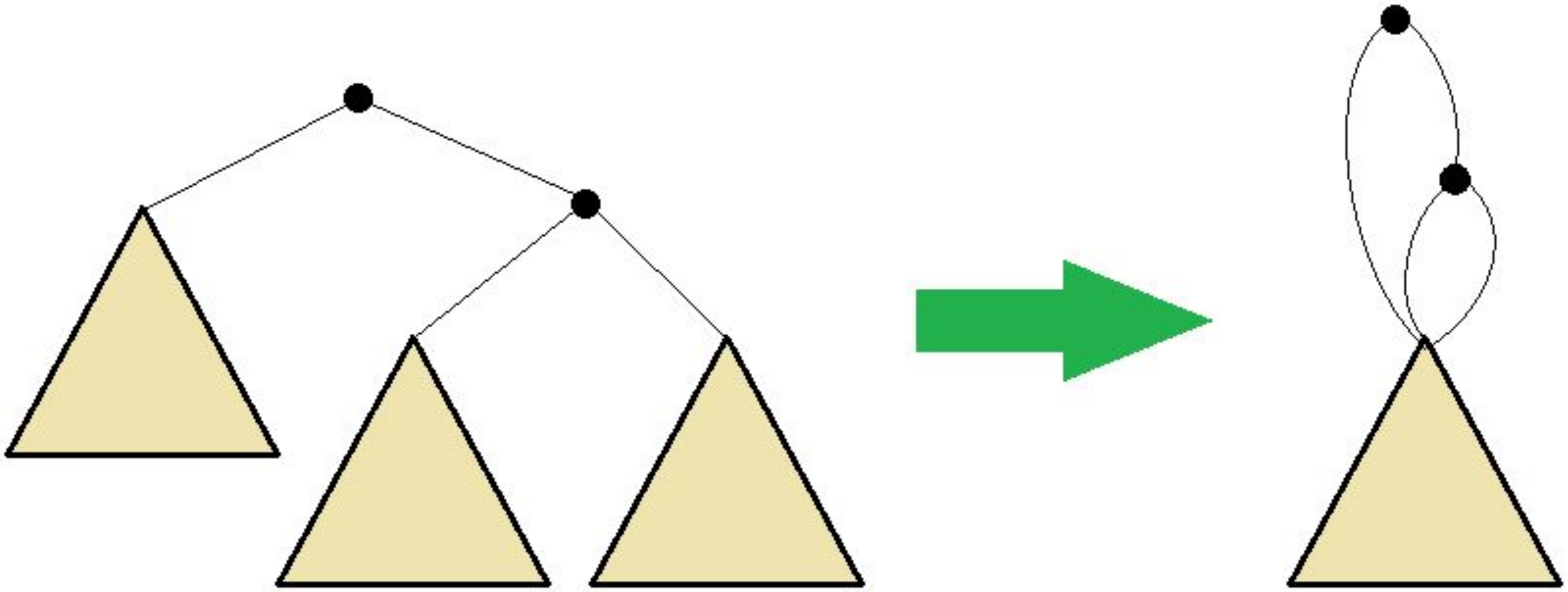
Problem K: Cut and Paste

- Well, maybe more like **O(n (log x + D))**
- where **D** = cost of duplicating a tree.
- But if we use **persistent trees**, **D** becomes **O(1)** !!
- So overall, **O(n log x)**.

Kevin Atienza

Problem K: Cut and Paste

- What happens when duplicating a persistent tree?



Kevin Atienza

Problem K: Cut and Paste

- Special case: count(“2”) = 2, count(“3”) = 0.
- Need to handle separately, string only grows quadratically, not exponentially.
- There is a pattern.

Kevin Atienza

Problem J: Intergalactic Sliding Puzzle

- “Shortcuts” are basically **subroutines**.
- Create simpler operations, then combine into more complex operations.

Kevin Atienza

Problem J: Intergalactic Sliding Puzzle

- Consider the “circular permutation”.
- Without the centermost column, you can rotate it, but you can’t change it beyond that.
- Thus, the only significant operation is **moving across the center**.

Kevin Atienza

Problem J: Intergalactic Sliding Puzzle

- This is its effect:

```
 1  2  3  4  5  6  7          1  2  3  E  5  6  7
                        →
13 12 11  E 10  9  8         13 12 11  4 10  9  8
```

Kevin Atienza

Problem J: Intergalactic Sliding Puzzle

- This is its effect:

1	2	3	4	5	6	7
13	12	11	E	10	9	8

1	2	3	E	5	6	7
13	12	11	4	10	9	8

- Or: 1 2 3 4 5 6 7 8 9 10 11 12 13

  1 2 3 5 6 7 8 9 10 4 11 12 13

Kevin Atienza

Problem J: Intergalactic Sliding Puzzle

- This is its effect:

1	2	3	4	5	6	7
13	12	11	E	10	9	8

1	2	3	E	5	6	7
13	12	11	4	10	9	8

- Or: 1 2 3 4 5 6 7 8 9 10 11 12 13

  1 2 3 5 6 7 8 9 10 4 11 12 13

Kevin Atienza

Problem J: Intergalactic Sliding Puzzle

- In other words, a **2k+1 rotation**.
- This can be done *anywhere*.
- This, along with a full **4k+1 rotation**, are the only allowed operations.
- Both are **even permutations**, so odd permutations are unsolvable.

Kevin Atienza

Problem J: Intergalactic Sliding Puzzle

- It turns out that **even** permutations are solvable! One can construct a 3-cycle using *six 2k+1 rotations*.
  - Exercise left to the reader.
  - With full rotations, this 3-cycle can be done anywhere!
- It is a standard fact that even permutations are decomposable into 3-cycles.

Thank you!

- **A. Suffix Three** - Yao
- **B. Miss Punyverse** - Yao
- **C. JaBloo 11: Lord of Expansion** - Yao
- **D. Okkeika Ferry Co.** - Yao
- **E. Do You Wanna Build More Snowmen?** - See
- **F. Marking the Territory** - See
- **G. Guizzmo and the CSS** - Dumol
- **H. Kirchhoff's Current Loss** - Asuncion
- **I. A Case By Case Basis** - Atienza
- **J. Intergalactic Sliding Puzzle** - Atienza
- **K. Cut and Paste** - Atienza
- **L. Jeremy Bearimy** - Atienza
- **M. Beingawesomeism** - Atienza

- Kevin Charles Atienza
  - Chief judge
- Kyle Stephen See
  - Chief tester
- Payton Robin Yao
- Jared Guissmo Asuncion
- Tim Joseph Dumol
- Marte Raphael Soliza
- Codeforces
  - Parallel round
  - Additional testing