Algolympics 2023

Solution Sketches

Samsung

J: Sensor Logs

- **Solution 1:** Keep track of the room each person is in at all times.
  - Use array.
- Each corridor they go through must connect to that room.
  - Sus if this fails!
- The new room is the other endpoint of the corridor.

Samsung

J: Sensor Logs

- **Solution 2:** If you go through a corridor, you can only go back through the same corridor.
- So each person's log can only look like **a a b b c c d d …**
  - Last log can be unpaired.
  - Can be empty.
- Sus *if and only if* not of this form.

Samsung

K: Star Seeker's Socks

- Let t = total number of 'bad' socks.
  - **Important:** each pair has *two* socks!
- Let $s_1, s_2, \ldots, s_m$ be the numbers of socks of the 'good' sock types.
- If you take t + m socks, worst case you could have gotten all t bad ones and 1 of each good type. Fail!
- Thus, you need to take at least t + m + 1.

Samsung

K: Star Seeker's Socks

- On the other hand, if you take $t + m + 1$ socks, you are forced to have a duplicate good sock of same type.
  - Pigeonhole principle.
- So the answer is $t + m + 1$.

Samsung

K: Star Seeker's Socks

- Python solution (single case):

```
input()   # no need for n, throw it out
c = [2*v for v in map(int, input().split())]
print(sum(c) + 1
    - sum(c[i-1] - 1 for i in map(int, input().split())))
```

Samsung

B: Cult of Wah!

- “Just do it!”
  - Put encrypted words in a set, say E.
  - For each k = 1, 2, 3, ...
    - Shift all Wah-List words by k, then check if they’re all in E.
  - Return first k that works.
  - If no k works, answer is -1.
- (you can also do it the other way around: put the Wah-List words into a set)

Samsung

B: Cult of Wah!

- Takes **$O(\infty)$** time to finish.
- But just note that shifting by k is the same as shifting by k+26, so if none of 1, 2, ..., 26 works, none will work at all!

Samsung

B: Cult of Wah!

- Some techniques to make it easier:
  - Define a function *shift(word, k)* that shifts a word by *k*. Then just call this function many times.
  - Some languages have convenient syntax for manipulating sets.
    - E.g., in Python, `x <= y` means “`x` is a subset of `y`”.

Samsung

B: Cult of Wah!

- **Gotcha:** $k = 26$ is a possible solution, even though it is equivalent to doing nothing!
  - $k = 0$ is not a **positive** integer.

Samsung

M: TheBuzz

- We need to find pairing between names and labels [1, 2, ..., n] such that
  - (name[i], name[j]) is an edge
    iff
    (label[i], label[j]) is an edge.
  - They are of the same type (if they exist).
- This check can be done in $O(n^2)$ time.

Samsung

M: TheBuzz

- Since n is small (≤ 10), just try all possible pairings!
- There are n! pairings.
- 10! = 3628800, so this should pass.

Samsung

M: TheBuzz

- We can simplify things a bit: turn "no edge between (i, j)" into a 4th type of edge.
  - There are now 4 edge types, but the graph is complete, simplifying implementation a bit.
- Objects like this that are invented for convenience are sometimes called *dummy* or *sentinel* objects.

Cisco Ortega

H: Not Just an NP-Hard Problem

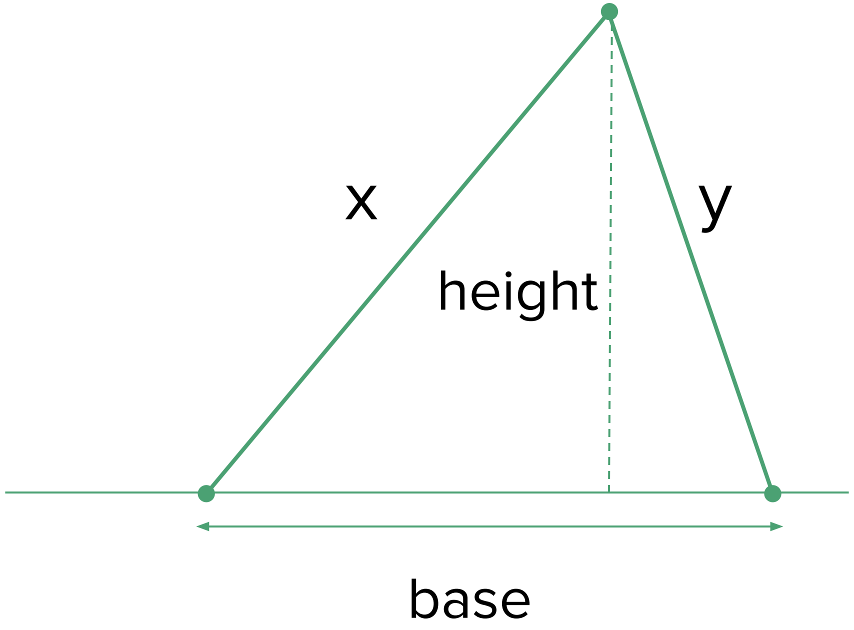
- Decompose into subproblems.
- For the no-stick-breaking version:
    a. Given two beams w/ lengths x and y, find the optimal angle to join them.
    b. Given the stick sizes, find the best way to reassemble them.

Cisco Ortega

H: Not Just an NP-Hard Problem

a. Given two beams w/ lengths x and y, find the optimal angle to join them.

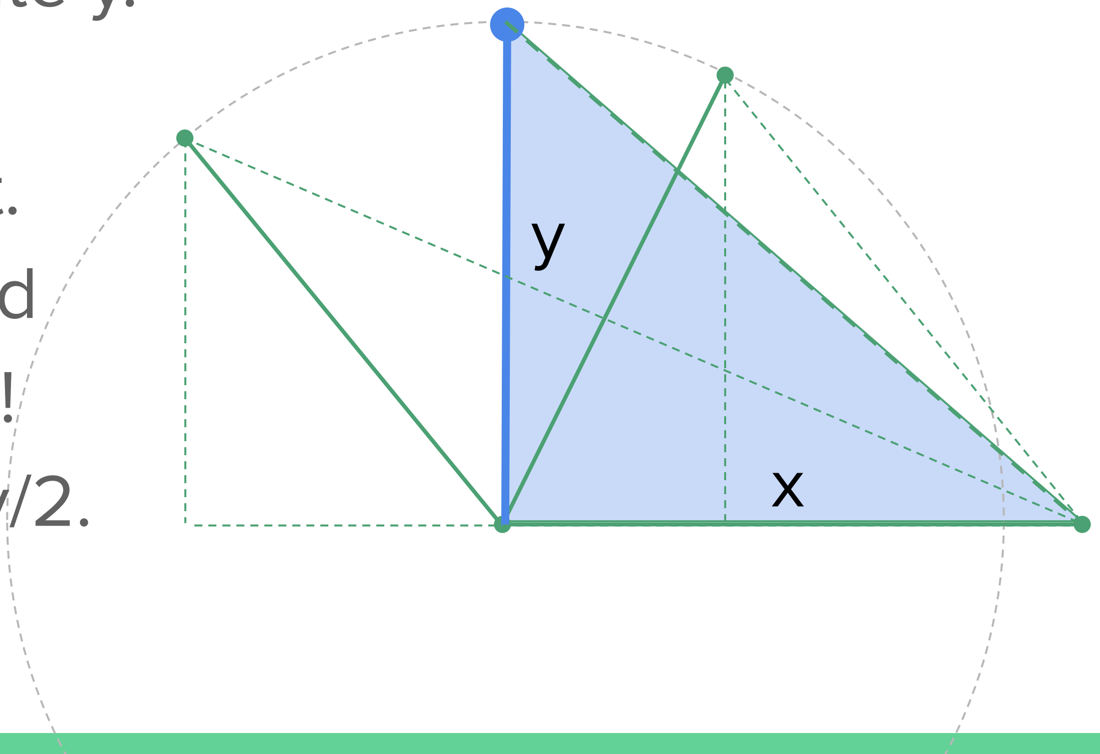
- Area = base · height / 2.



Cisco Ortega

H: Not Just an NP-Hard Problem

- Fix base x, then rotate y.
- We now need to maximize the height.
- So area is maximized when perpendicular!
- Thus, max. area = xy/2.



Cisco Ortega

H: Not Just an NP-Hard Problem

b. Given the stick sizes, find the best way to reassemble them.

- NP-Hard! This is essentially the subset sum problem.
- No known polynomial-time solutions.
- So let’s find a solvable variant.

Cisco Ortega

H: Not Just an NP-Hard Problem

- Suppose you can break sticks at any time. What is the answer?
- You can basically build any beam sizes you want!
  - More precisely: If t = total length of sticks, we can produce beams (x, y) iff $x + y = t$.
    - (and of course x and y must be positive)

Cisco Ortega

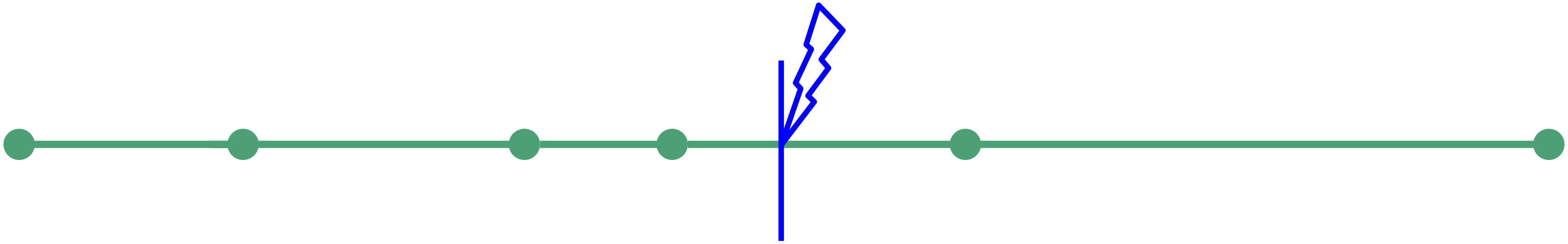
H: Not Just an NP-Hard Problem

- Maximize $xy/2$ subject to $x + y = t$.
- Basic optimization. The answer is a square: $x = y = t/2$.
  - But x and y need to be integers, so sometimes we must have $|x - y| = 1$.
- Thus, the optimal beams are roughly equal:
  - $(t/2, t/2)$ if t is even,
  - $([t/2], [t/2] + 1)$ if t is odd.
- This solves the “easy” variant.

Cisco Ortega

H: Not Just an NP-Hard Problem

- **Insight:** You can produce roughly equal beams (t/2, t/2) or ([t/2], [t/2]+1) just by breaking one stick!



Cisco Ortega

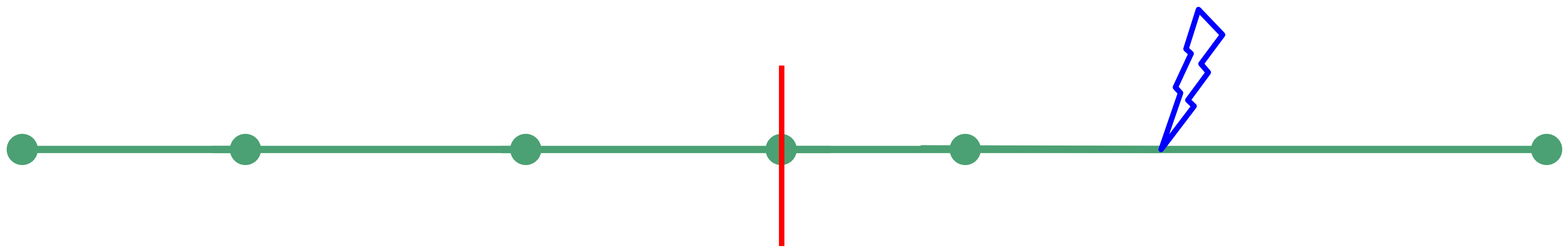
H: Not Just an NP-Hard Problem

- Thus, the optimal solution is the same!
- **O(n)**

Cisco Ortega

H: Not Just an NP-Hard Problem

- **Gotcha:** The middle point may already be cut!
- If so, just cut a random stick.
  - Always possible because x[i] ≥ 2.



Cisco Ortega

C: Dethrone Antares Now

- If staying is okay, then people can wait. So just find the “center” by doing a breadth-first search (BFS) from each of the *k* sources.
- Doesn’t work since staying is not okay.

Cisco Ortega

C: Dethrone Antares Now

- But they can just go back and forth some edge!
- Thus, if they can reach a node at time t, then they can also reach it at $t+2$, $t+4$, $t+6$, ...
- So, we just need to find earliest time to reach each node in odd time, and also even time.

Cisco Ortega

C: Dethrone Antares Now

- Trick: Build a graph with 2*n* nodes.
- For each node *i*,
  - add two nodes (*i*, *even*) and (*i*, *odd*).
- For each edge *i* – *j*,
  - Add edge (*i*, *even*) – (*j*, *odd*).
  - Add edge (*i*, *odd*) – (*j*, *even*).
- Then BFS starting at (*s*, *even*)!

Cisco Ortega

C: Dethrone Antares Now

- BFS'ing k times takes $O(k(2n + 2m))$ time.
  - $O(k(2n + 2m)) = O(k(n + m))$.
- Then just find the node (i, *parity*) that has minimum max time; i will then be the meeting point.
- Then, just have people walk to it, and have early birds go back and forth some arbitrary edge while waiting for others.

Kevin Atienza

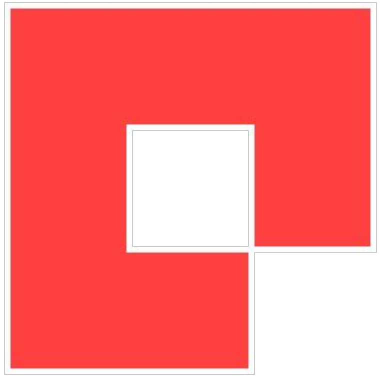
I: Ominous Acids

- Lots of *k*-ominoes even for small *k* !
- You can prove the number grows exponentially.
- Numbers up to $k = 15$:
  - 1, 1, 2, 7, 18, 60, 196, 704, 2500, 9189, 33896, 126759, 476270, 1802312, 6849777
- Last few don't even fit in 2023×2023 area!
  - This suggests there are impossible cases.

Kevin Atienza

I: Ominous Acids

- **Insight:**



- $k \geq 7$ impossible!

Kevin Atienza

I: Ominous Acids

- All that remains are $k \leq 6$.
- $k \leq 3$ trivial, can be done by hand.
- With more effort, maybe $k = 4$ too. And even $k = 5$. And even $k = 6$.
  - Can be tedious though.

Kevin Atienza

I: Ominous Acids

- **Insight 2:** You have a computer. Use it!
- Backtracking to enumerate all *k*-ominoes.
- Backtracking to find valid tiling?
  - Can be very hairy.

Kevin Atienza

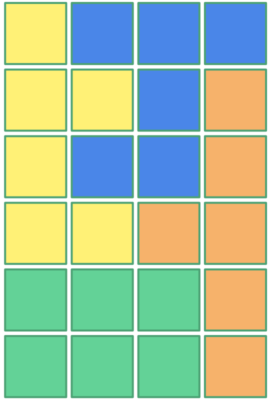
I: Ominous Acids

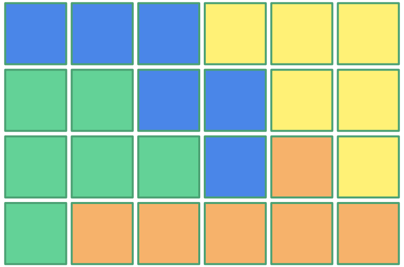
- **Insight 3:** You have paper. Use it!
- You don't need to code everything.
- Easier to find tiling by hand once you have all distinct *k*-ominoes.

Kevin Atienza

I: Ominous Acids

- **Insight 4:** Just make a neat tiling, e.g., by building small rectangular tiles and assembling them into one.
    - Assembly can be done manually or with code.





Kevin Atienza

I: Ominous Acids

- Other notes:
  - You can also enumerate *k*-ominoes by hand, but it's **risky**: easy to make mistake and miss some *k*-ominoes.
  - You can ignore the reflected *k*-ominoes. Just take your final grid, flip, then combine.
    - (Reduces work by ≈ half)

Kevin Atienza

I: Ominous Acids

- **Moral:** You have brains, paper, and computer. Use all of them (in the right way) for best results!

Cisco Ortega

L: Starquake!

- Clearly, the naive solution is too slow.
  - nq is large!
- Clearly, we need some data structures.

Cisco Ortega

L: Starquake!

- We need to count landmasses in h[i..j] quickly.
- Assume i, i+1, ..., j are initially in separate landmasses.
- **Observation:** Every consecutive height difference of -1, 0, or +1 decreases # landmasses by 1.
- Thus, # landmasses in h[i..j] is:
  - (j - i + 1) - (# of -1s, 0s, +1s in difference array.)

Cisco Ortega

L: Starquake!

- We need to process range queries quickly on *difference array* (h[2] - h[1], h[3] - h[2], ..., h[n] - h[n-1]).
- We need to know effects of updates in difference array.

Cisco Ortega

L: Starquake!

- `FISSURE` i j: changes h[i] - h[i-1] and h[j+1] - h[j], and nothing else.
  - Middle sections unaffected; they move together.
- `EARTHQUAKE`: change looks like +1, -1, +1, -1, ...

Cisco Ortega

L: Starquake!

- **Insight:** Split difference array into odd and even parts.
- Then earthquake becomes
  +1, +1, +1, ... in one array, and
  -1, -1, -1, ... in the other!

L: Starquake!

Cisco Ortega

- So now, we have reduced to:
- `QUERY`: count {-1, 0, +1} in subarray
- `FISSURE`: Point update
- `EARTHQUAKE`: Range increase or decrease by 1

Cisco Ortega

L: Starquake!

- We can just use sqrt decomposition to be simple!!
  - I’m not actually sure if there’s a segment tree solution.
- Process $\sqrt{n}$ commands at a time, in roughly $O(n)$.
- If your solution needs some sorting of blocks, then it might be $O((n + q) \sqrt{n} \log n)$.
- It can be improved to **$O((n + q) \sqrt{n})$**.
  - (Hint: range updates are only +1 or -1)

Josh Quinto

G: Irreversible Events

- Only irreversible paths matter.
- i.e., we don't care about a path a $\rightsquigarrow$ b if there's also a path b $\rightsquigarrow$ a.
  - i.e., we don't care if scc(a) = scc(b).
- So, look at SCCs (strongly connected components).

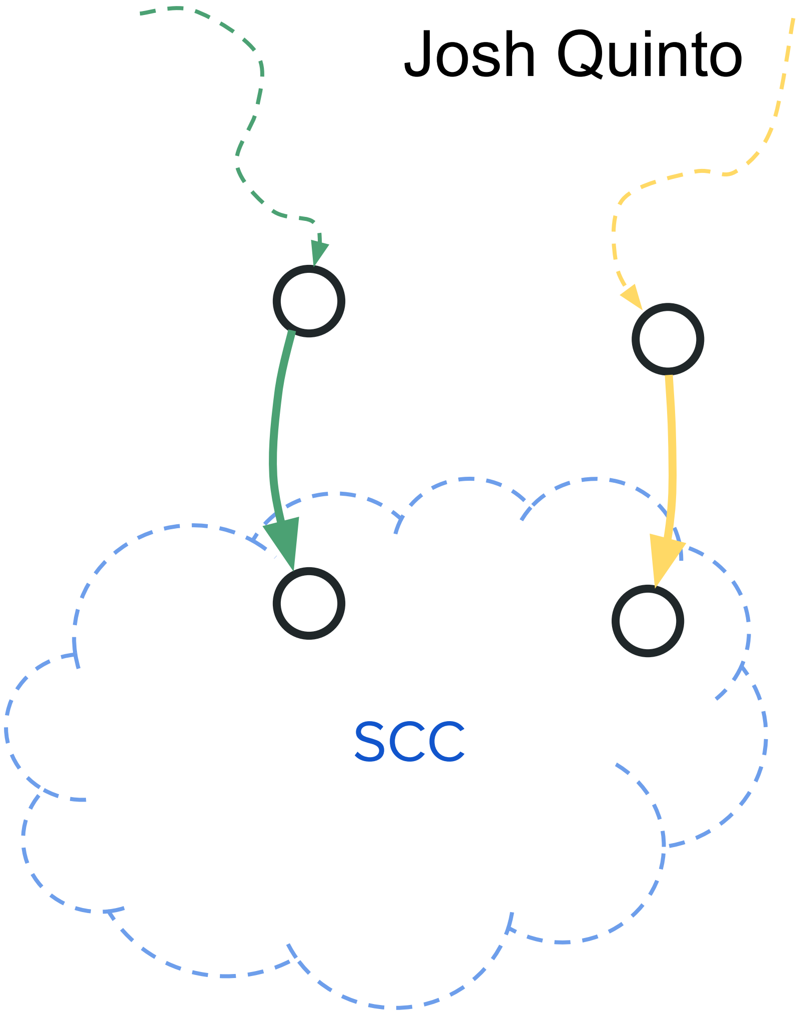
Josh Quinto

G: Irreversible Events

- Where are divergent events?
- Paths need to come from outside SCC.
- We need a pair of such paths not sharing an edge.
- In particular, the last edges to the SCC are different!

G: Irreversible Events

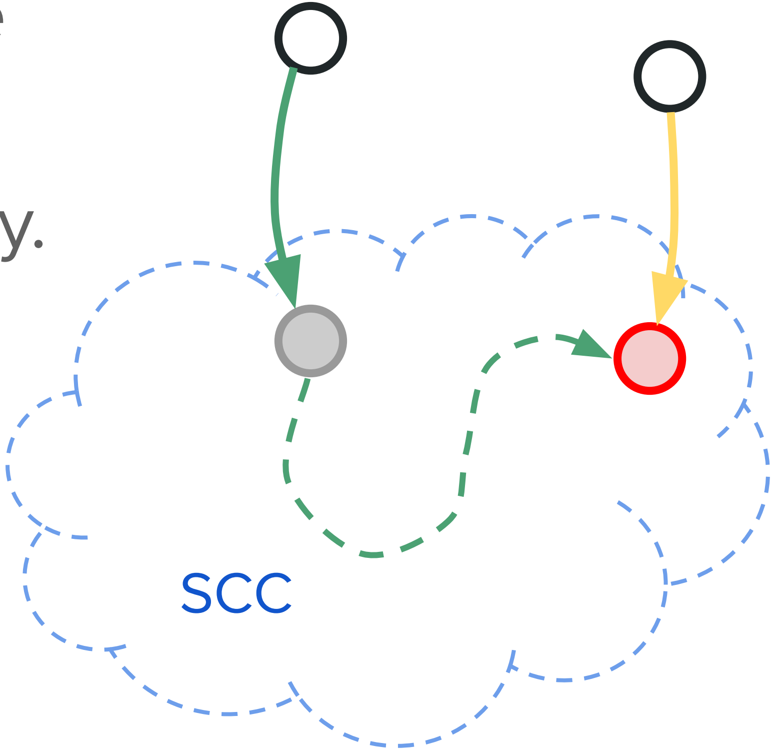
- In particular, the last edges to the SCC are different:



Josh Quinto

G: Irreversible Events

- **Claim:** If 2 edges to the SCC exist, then some event is *not* in continuity.
- So let's look at graphs with at most one edge to each SCC.



Josh Quinto

G: Irreversible Events

- **Observation:** If there's at most one edge to the SCC, then all paths to it go through that edge.
- Thus, all events in it are in continuity!
- Therefore,
  (graph is good) ⇔ (at most one edge to every SCC).

Josh Quinto

G: Irreversible Events

- Greedy? Take SCCs, then remove all but one edge to each SCC.
  - This gives the answer ∑ max(0, indeg[C]-1) for all SCCs C.
- Is it correct? **Greedy needs proof!**
- **Issue:** This doesn't rule out strategies that break some SCCs apart.

Josh Quinto

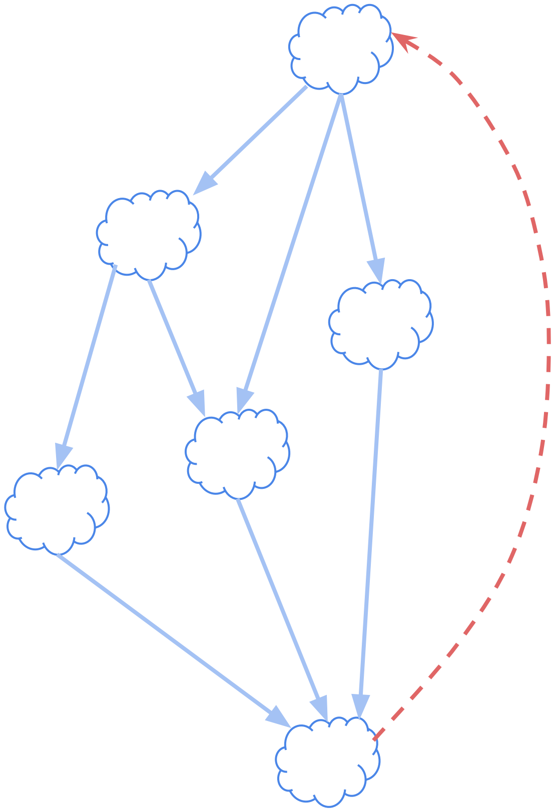
G: Irreversible Events

- **Claim:** Breaking apart an SCC doesn't help.
- Need to prove: removing an edge within SCC doesn't decrease $\sum \max(0, \text{indeg}[C]-1)$, even if the SCCs change.

Josh Quinto

G: Irreversible Events

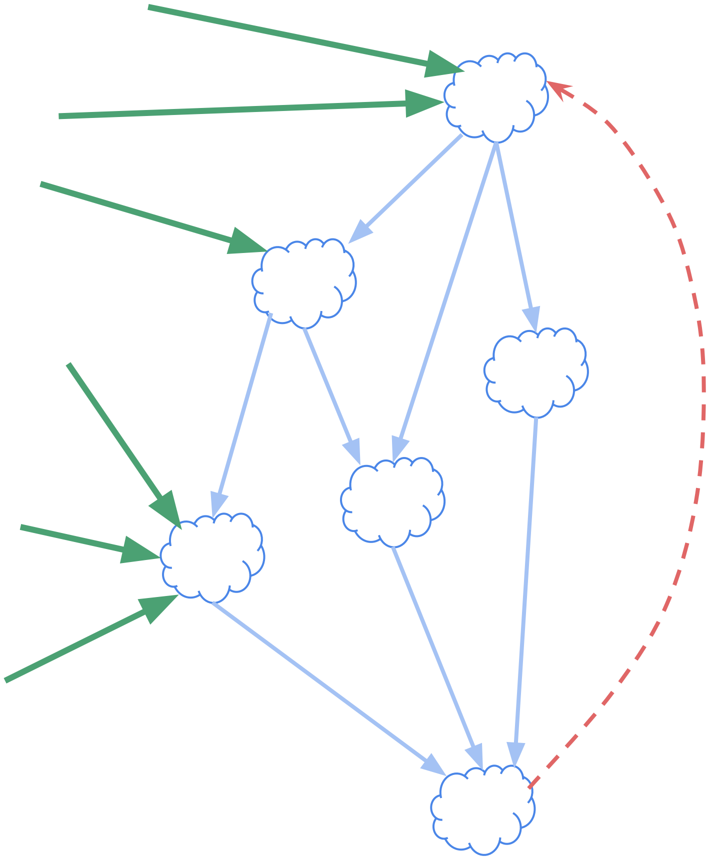
- **Key:** Removing edge in SCC X yields a DAG of SCCs with at most one source (and sink).



Josh Quinto

G: Irreversible Events

- Thus, each old edge to X still contributes 1 to ∑ max(0, indeg[C]-1), except possibly for one edge (to the source).



Josh Quinto

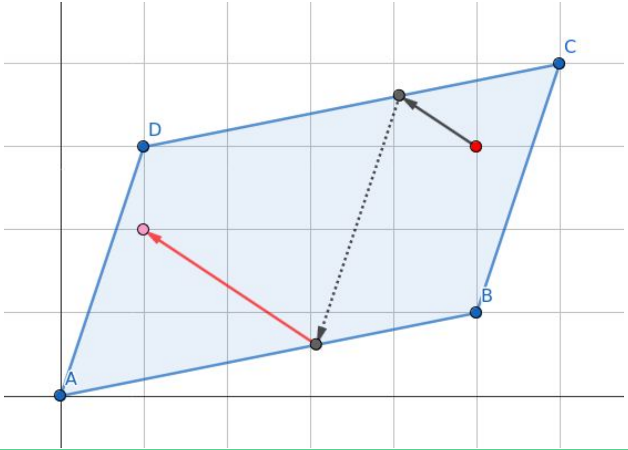
G: Irreversible Events

- Thus, $\sum \max(0, \text{indeg}[C]-1)$ stays the same.
- Thus, removing an edge in an SCC doesn’t help.
- Thus, the greedy solution is correct!
- Finding SCCs takes **linear** time.
    - Kosaraju’s or Tarjan’s algorithm.

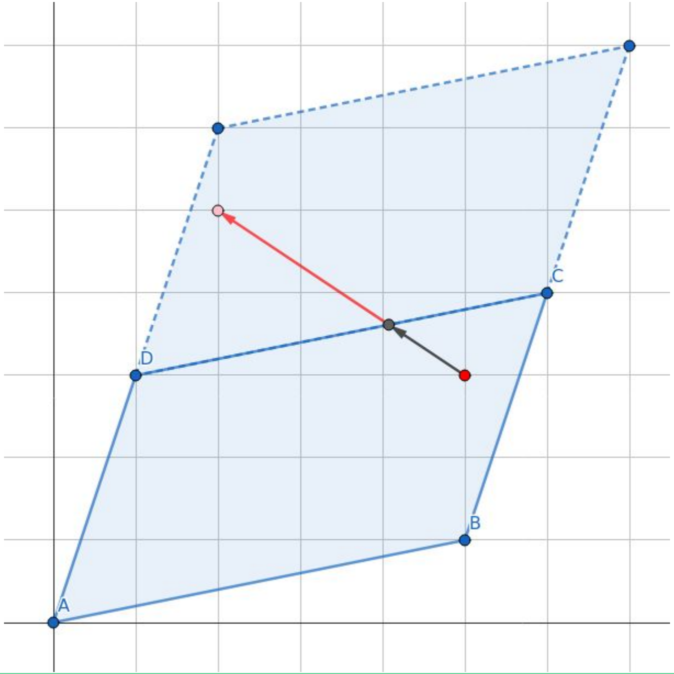
Kevin Atienza

E: Euclidean Travel with Parallel Universes

- **Insight:** Teleporting via AB ↔ DC is the same as going to a "parallel universe"



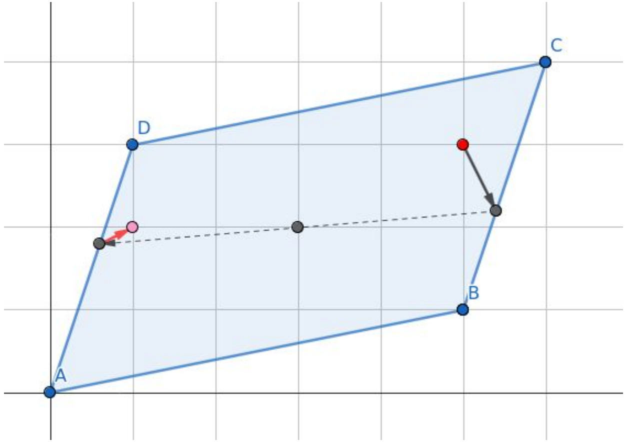
=



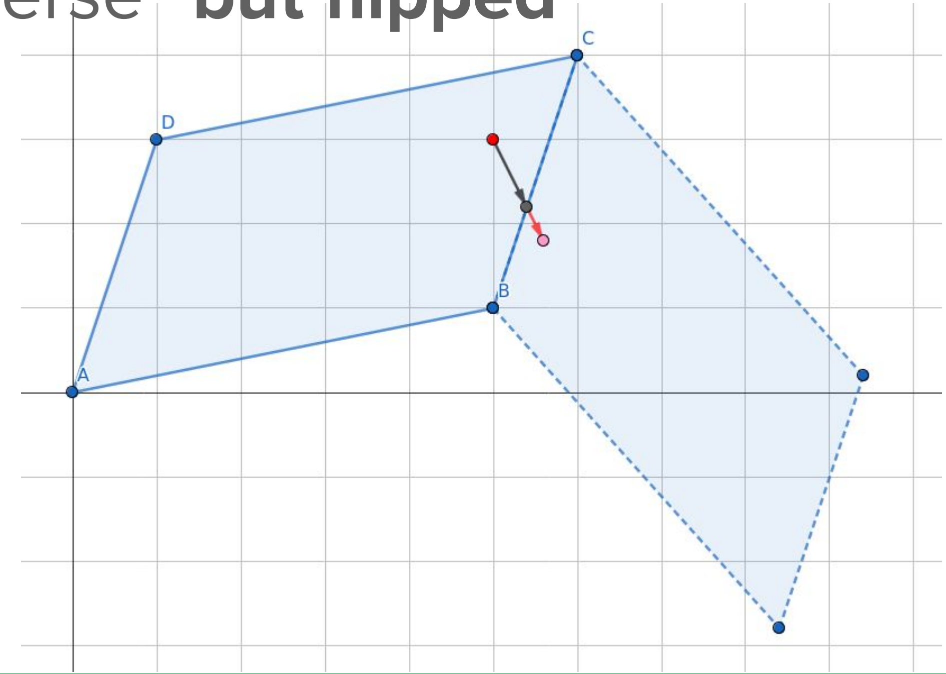
Kevin Atienza

E: Euclidean Travel with Parallel Universes

- **Insight:** Teleporting via BC ↔ DA is the same as going to a "parallel universe" **but flipped**



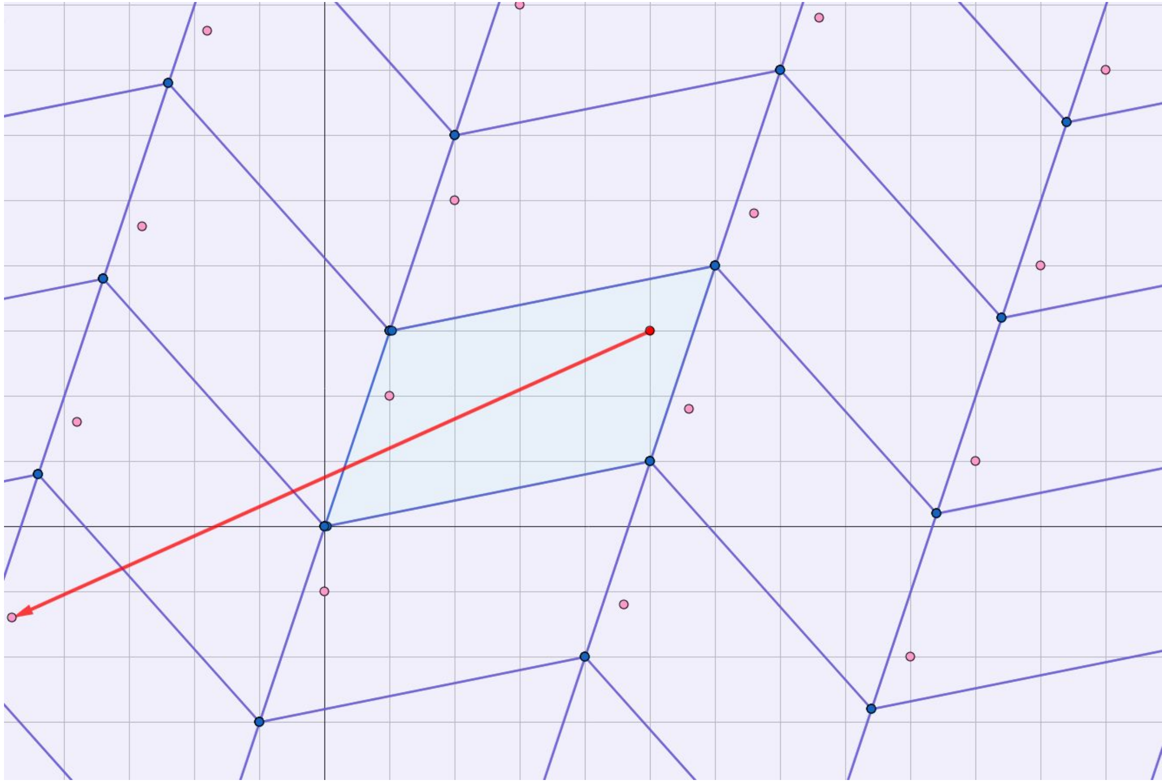
=



Kevin Atienza

E: Euclidean Travel with Parallel Universes

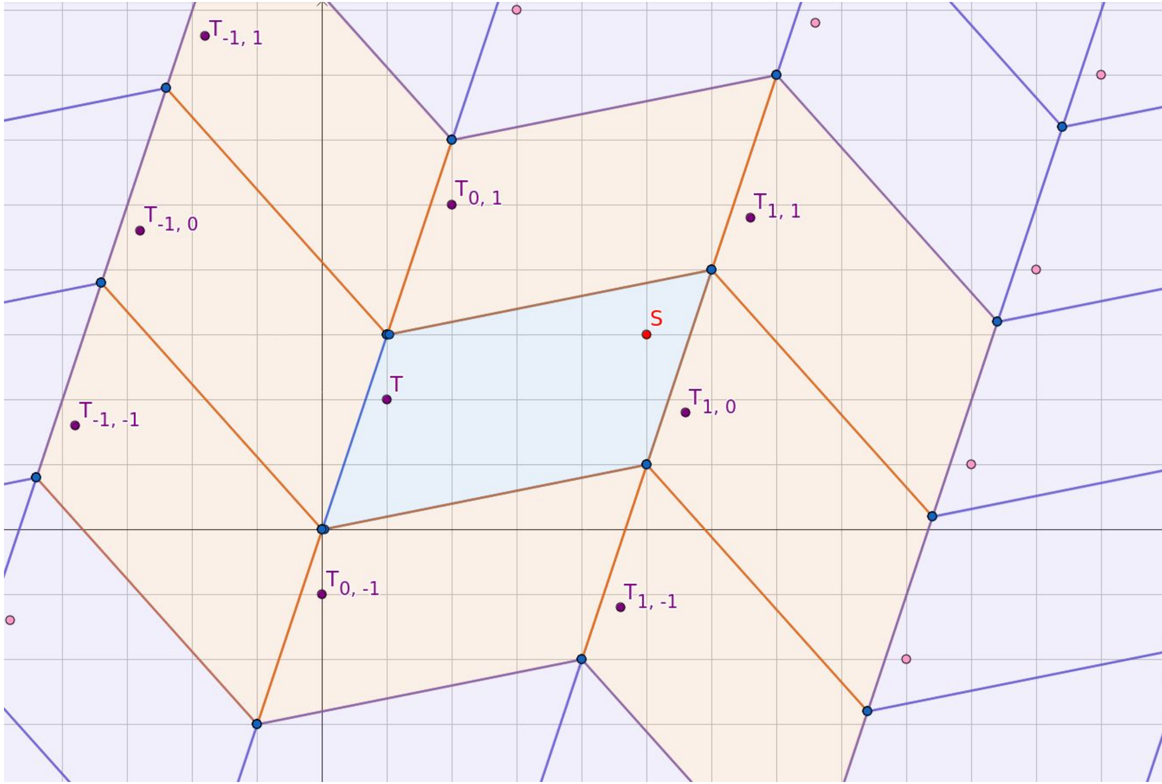
- **Insight:** Multiple teleports means travelling through multiple parallel universes in this tiling:



Kevin Atienza

E: Euclidean Travel with Parallel Universes

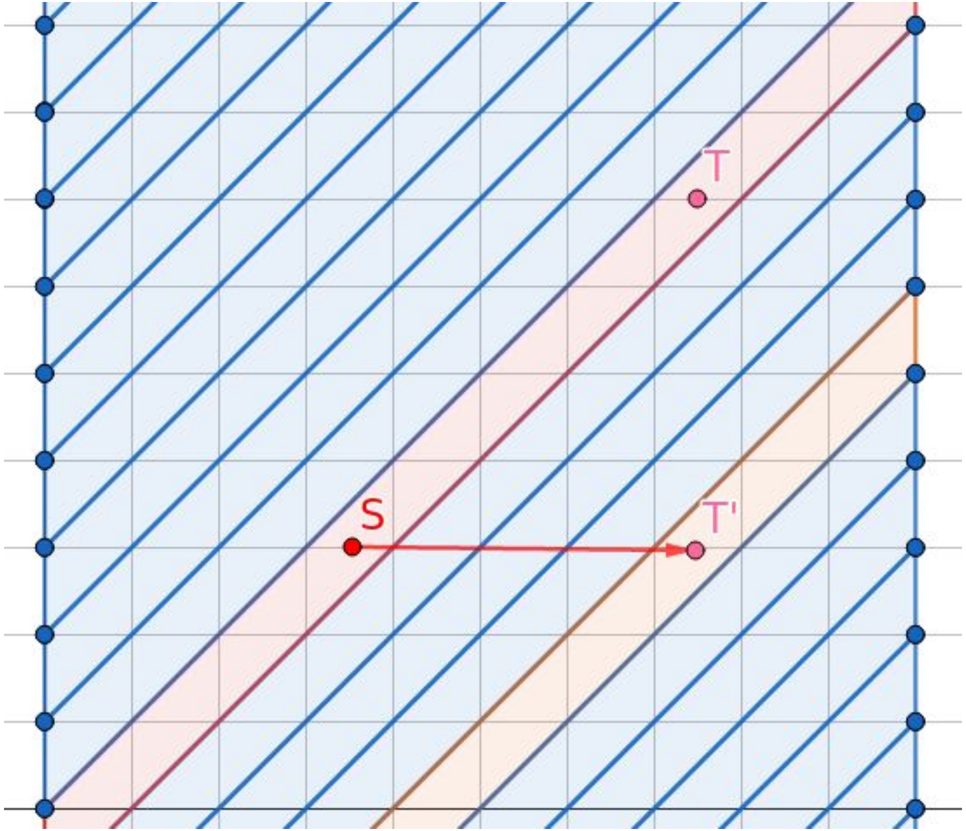
- **Solution:** produce the 8 neighboring parallel universes and find shortest path to each image of (x_t, y_t).
- **O(1)**.



Kevin Atienza

E: Euclidean Travel with Parallel Universes

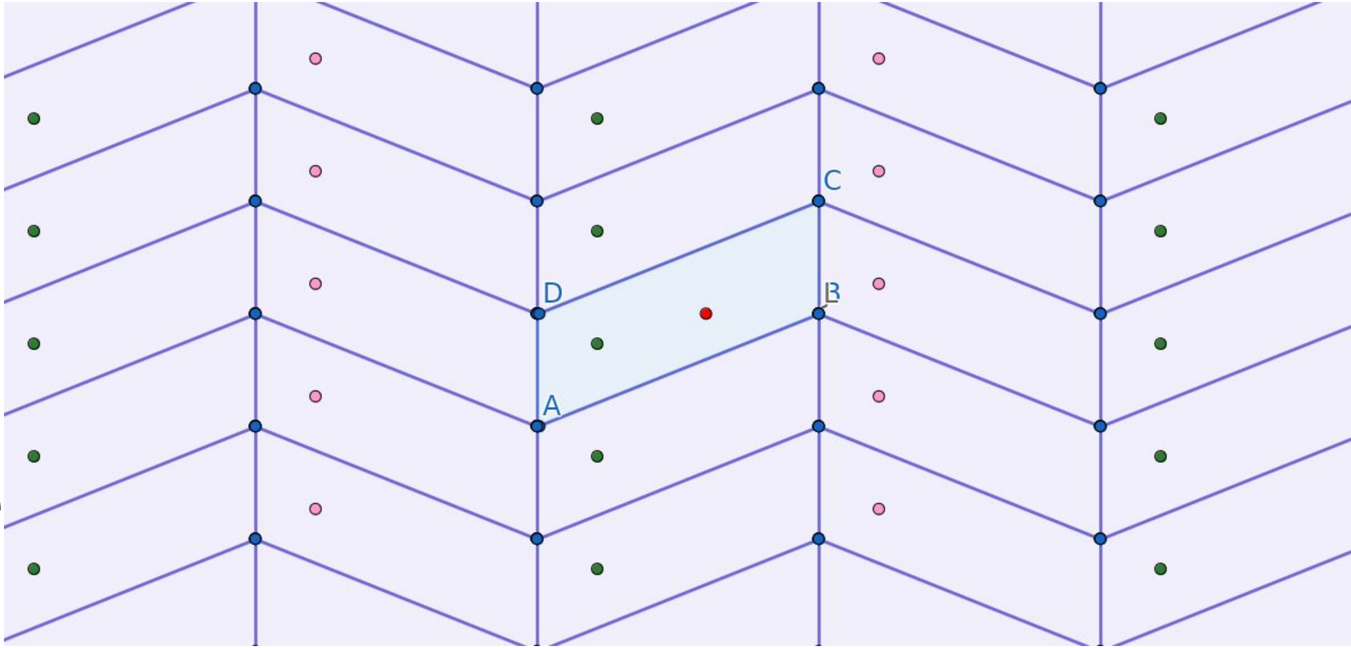
- **WRONG!**
  - It's possible to have lots of teleports!



Kevin Atienza

E: Euclidean Travel with Parallel Universes

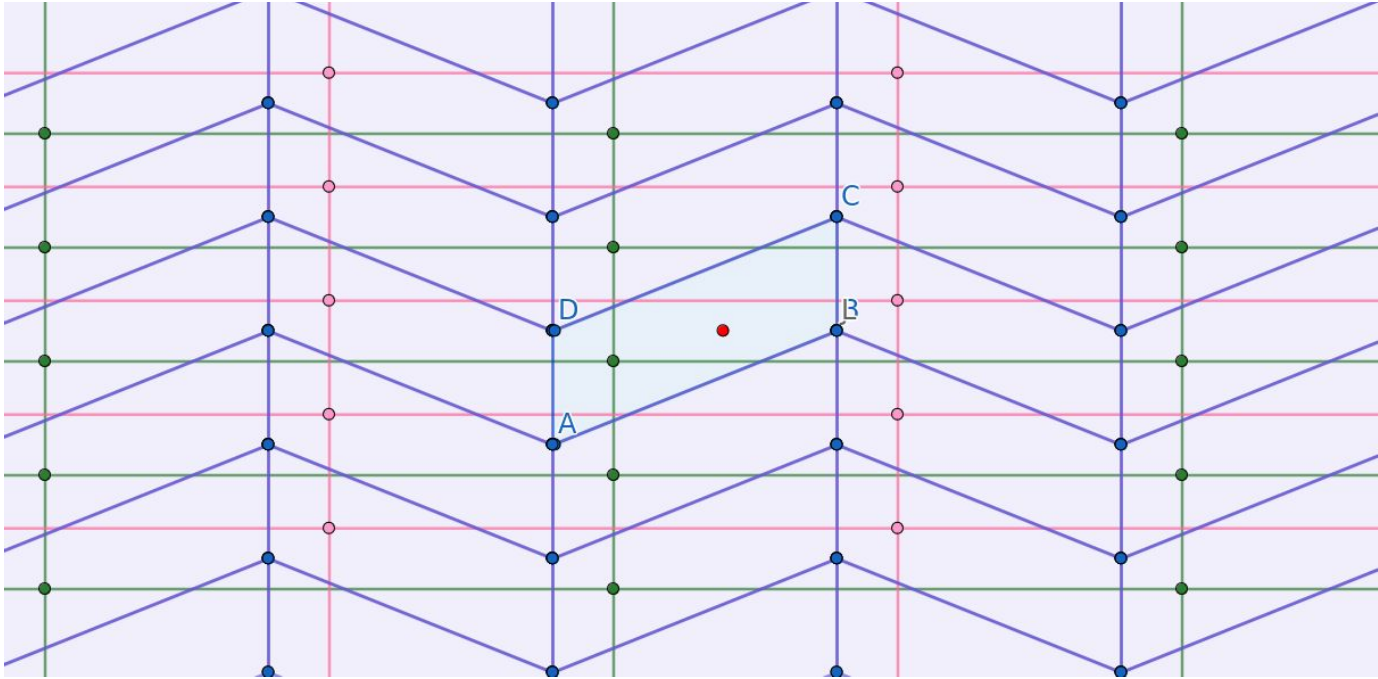
- Let's simplify by rotating the shape so BC and AD are vertical. Then produce infinite tiling.



Kevin Atienza

E: Euclidean Travel with Parallel Universes

- **Insight:** Images of (x_t, y_t) form a union of two rectangular lattices:



Kevin Atienza

E: Euclidean Travel with Parallel Universes

- Finding the shortest path from a point to a rectangular lattice can be done in **O(1)**.
    - Only four candidates to check.

Kevin Atienza

E: Euclidean Travel with Parallel Universes

- **Morals:**
  - Do simplifying transforms!
  - Draw a lot!
  - Practice geometry!

Cisco Ortega

F: Flow Maximal

- Given (a, c), the max. # of chains in flow is clearly $f = \min(2a, 2(c - a))$.
  - If $a \leq c/2$, each A-bead must be between two non-A-beads.
  - If $a \geq c/2$, each non-A-bead must be between two A-beads.
  - We can just assume $a \leq c/2$ since A-beads and non-A-beads are symmetric; we can replace $a := \min(a, c - a)$.

Cisco Ortega

F: Flow Maximal

- Thus, the number of chains “should be”: choose(c-a, a).
    - (assuming $a \leq c/2$)
- BUT this double counts… a lot.
    - (because of rotations and reflections)

Cisco Ortega

F: Flow Maximal

- We can count the distinct chains using a powerful hammer known as **Burnside's lemma**
  - from group theory
  - (# orbits) × (# symmetries)
    = ∑ (# fixed points of symmetry S)
    the sum runs across all symmetries S
  - generalized by *Pólya enumeration*
  - Exposition (and proof) here:
    https://brilliant.org/wiki/burnsides-lemma/

Cisco Ortega

F: Flow Maximal

- **Burnside's lemma**
  - (# orbits) × (# symmetries)
    = ∑ (# fixed points of symmetry S)
- # orbits = # distinct objects, i.e., what we need
- For size-n beads, there are 2n symmetries:
  - n rotations and n reflections.
  - a.k.a., "dihedral group"
- Finding # fixed points is usually easy.

Cisco Ortega

F: Flow Maximal

- Working it out, you get a sum of a couple of binomial coefficients (proof left to reader):

```python
def choosepal(n, r):
    return choosepal(n-1, r) + choosepal(n-1, r-1) if n%2 else 0 if r%2 else choose(n//2, r//2)

def f(a, c):
    a, c = sorted((a, c - a))
    if a == 0: return 1
    g = gcd(a, c)
    ans = sum(choose(c//d, a//d) * phi(d) for d in divisors(g))
    ans += (c - c//2) * choosepal(c, a)
    ans += (c//2) * choosepal(c - 1, a - 1)
    ans += (c//2) * choosepal(c - 1, a)
    return ans//(2*c)
```

Cisco Ortega

F: Flow Maximal

```
def choosepal(n, r):
    return choosepal(n-1, r) + choosepal(n-1, r-1) if n%2 else 0 if r%2 else choose(n//2, r//2)

def f(a, c):
    a, c = sorted((a, c - a))
    if a == 0: return 1
    g = gcd(a, c)
    ans = sum(choose(c//d, a//d) * phi(d) for d in divisors(g))
    ans += (c - c//2) * choosepal(c, a)
    ans += (c//2) * choosepal(c - 1, a - 1)
    ans += (c//2) * choosepal(c - 1, a)
    return ans//(2*c)
```

- Proof left to reader. Some details:
    - choosepal(n, r) = number of palindromic ways to choose.
    - First sum corresponds to # fixed points of rotations.
    - Last three summands correspond to # fixed points of reflections.

Cisco Ortega

F: Flow Maximal

- choose(n, r) can be computed in O(r) time:
  (n) (n-1) (n-2) ... (n-r+1) / ((r) (r-1) (r-2) ... (1))
- Thus, the previous formula can be computed in **O(a log log g)** where g = gcd(a, c).
  - Needs the result that σ(n) = O(n log log n).
    - (Grönwall's theorem)

Cisco Ortega

F: Flow Maximal

- Now, we need to sum f(a, c) for $a^2 + b^2 = c^2$ and $0 \le a \le c$.
- $b^2 = c^2 - a^2 = (c - a)(c + a)$.
- Thus, we want to find all factorizations of b^2.
  - We can easily do it after prime-factorizing b (which is fast).
- But a and c can be large, so doing O(a log log g) for each (a, c) across some range may not cut it!

Cisco Ortega

F: Flow Maximal

- So we have $b^2 = (c - a)(c + a)$.
- **Insight 1:** $c - a \leq c + a$, so $c - a \leq \sqrt{b^2} = b$.
- **Insight 2:** A-beads and non-A-beads are symmetric. There are $c - a$ non-A-beads, therefore $f(a, c) = f(c - a, c)$!

Cisco Ortega

F: Flow Maximal

- **Insight 1:** $c - a \le b$
- **Insight 2:** $f(a, c) = f(c - a, c)$
- Combining both insights, $c - a \le b$ is small, and $f(a, c) = f(c - a, c)$ can be computed in $O((c - a) \log \log (c - a)) = O(b \log \log b)$ time.
  - This is now reasonable!
- Thus, the naive summing works after all!

Cisco Ortega

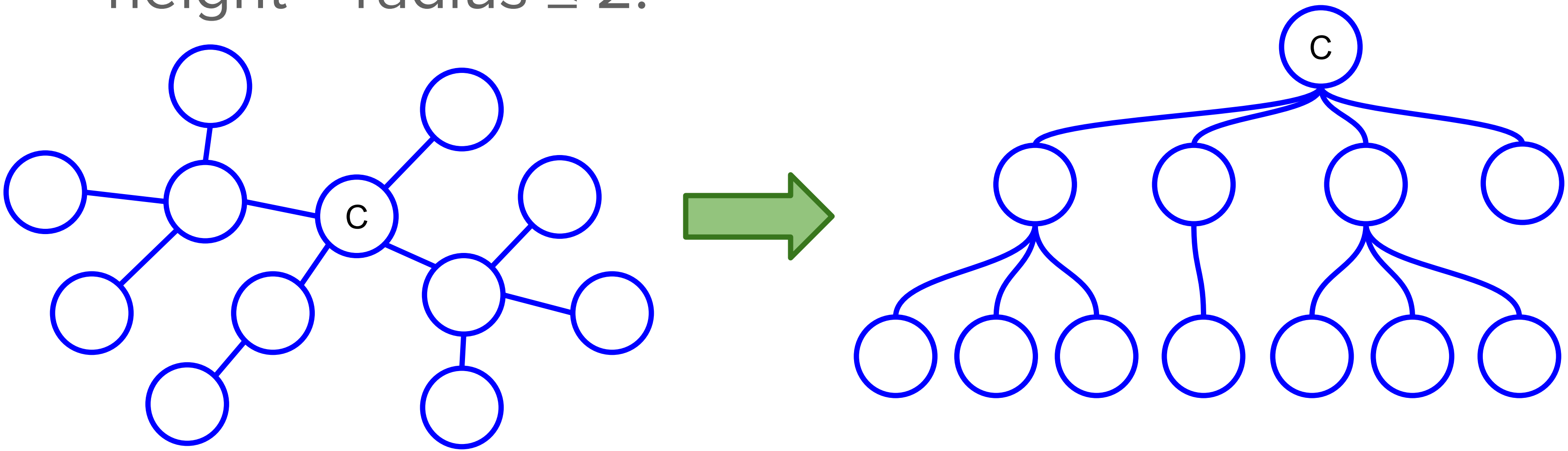
F: Flow Maximal

- **Gotcha:** b = 0 !!
    - What is the answer?

Kevin Atienza

A: Alien Gordon Ramsey

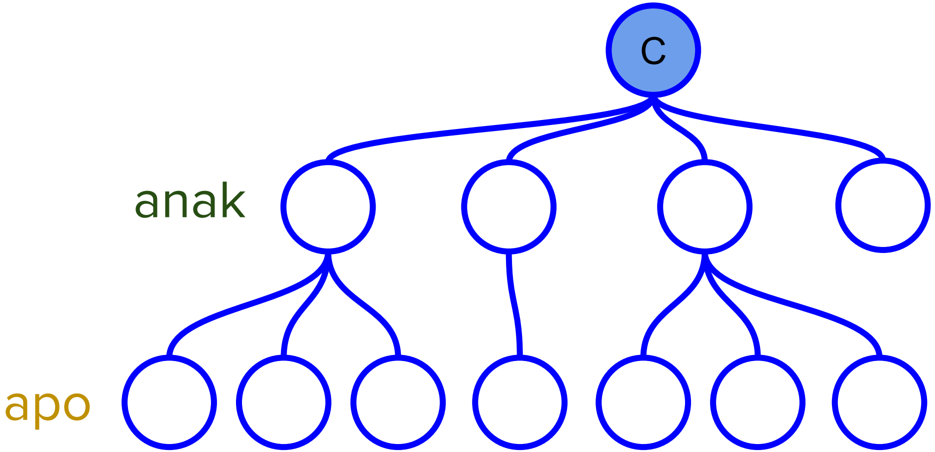
- Diameter ≤ 4 means radius ≤ 2.
- Let C be a center. Root at C. Then height = radius ≤ 2.



Kevin Atienza

A: Alien Gordon Ramsey

- Color C. Then the color of C cannot appear anywhere else.
  - even at 3rd level.



Kevin Atienza

A: Alien Gordon Ramsey

- The children of C must have distinct colors.
- Grandchildren of C may now use these colors.
- But sometimes they’re not enough.
- We need to know how many extra colors are needed.
- There’s a relatively straightforward max-flow/ matching approach
    - but of course it’s too slow

Kevin Atienza

A: Alien Gordon Ramsey

- A few greedy approaches work.
    - I’ll describe one such.
- **Insight:** Let X be the child of C with fewest children, say x. Then it’s optimal to use the colors of x siblings with fewest children.
    - If not enough colors, need extra ones, then repeat!
        - (or binary search it)
- THEN use X’s color under non-chosen siblings.

Kevin Atienza

A: Alien Gordon Ramsey

- Why does it work?
  - Can be intuitively true, but also tricky to rigorously prove.
- A possible proof is to reduce to *tournament score sequences* (**Landau's theorem**):
  - $(s_1, s_2, \ldots, s_n)$ with $0 \le s_1 \le \ldots \le s_n$ and $s_1 + \ldots + s_n = n(n - 1)/2$ is a tournament score sequence iff $s_1 + \ldots + s_i \ge i(i - 1)/2$ for all i.
  - Proof details omitted. (Left to reader 😊)

Kevin Atienza

A: Alien Gordon Ramsey

- **O(n)** with the right implementation.
- Maybe O(n log n) if you use a priority queue to find the fewest-children nodes.
  - May not pass though. Optimization may be needed

Kevin Atienza

D: Eliens Slurs

- Start with a known string matching solution: via *Fast Fourier Transform*!
- Two strings X[0..k-1] and Y[0..k-1] match iff $\sum (X[i] - Y[i])^2 = 0$.
- So the number of substring matches of S in T is the number of j such that $\sum (S[i] - T[i+j])^2 = 0$.

Kevin Atienza

D: Eliens Slurs

- $\sum (S[i] - T[i+j])^2$
  $= \sum S[i]^2 - 2 \sum S[i]\, T[i+j] + \sum T[i+j]^2$
- $\sum S[i]^2$ are just subarray sums in array $(S[i]^2)$.
- $\sum T[i+j]^2$ are just subarray sums in array $(T[i]^2)$.
- $\sum S[i]\, T[i+j]$ is a convolution of S and T.
  - Reverse S to see how it is a convolution.
- Convolution can be computed with FFT!

Kevin Atienza

D: Eliens Slurs

- Technique somewhat extensible. We just need to find an arithmetic expression that’s 0 iff it’s a match.
  - e.g., if S has wildcards (matches any character), then can use $\sum S[i]\,(S[i] - T[i+j])^2$.
    - $= \sum S[i]^3 - 2 \sum S[i]^2\, T[i+j] + \sum S[i]\, T[i+j]^2$
    - So this needs two convolutions.
      (the latter two; the first one is just subarray sums)

Kevin Atienza

D: Eliens Slurs

- **Insight:** “Difference at most 1” has one such simple arithmetic expression:
  $\sum ((X[i] - Y[i])(X[i] - Y[i] + 1)(X[i] - Y[i] - 1))^2$
- Expand this into a gigantic expression, and you’ll end up with several FFTs, and some subarray sums.
- There may be too many FFTs? Let’s try reducing.

Kevin Atienza

D: Eliens Slurs

- **Trick:**
  - Do the FFTs for all arrays $(S[i])$, $(S[i]^2)$, ..., $(S[i]^5)$, $(T[i])$, $(T[i]^2)$, ..., $(T[i]^5)$ first,
  - then do the pointwise products (linear) in freq. space
  - then do a single inverse FFT.
- Reduces # of FFTs from $O(d^2)$ to $O(d)$, where d = degree of expression.
  - Ours has $d = 6$.

Kevin Atienza

D: Eliens Slurs

- **Trick 2:** Lower the degree d.
- Note that we cannot use something like $\sum (X[i] - Y[i])(X[i] - Y[i] + 1)(X[i] - Y[i] - 1)$ because it has false positives
  - e.g., “CC” is matched with “AE” this way.
- *Crucial property:* we want the summand to be nonnegative.

Kevin Atienza

D: Eliens Slurs

- But we can use something like $\sum (X[i] - Y[i])^2 (X[i] - Y[i] + 1)(X[i] - Y[i] - 1)$.
- The summand is 0 if difference in {-1, 0, +1}, and positive otherwise.
  - We used the fact that X[i] - Y[i] is an integer.
- Degree is now d = 4 !

Kevin Atienza

D: Eliens Slurs

- Combining both techniques, we only need 6 FFTs (and 1 inverse FFT).
- O(7 (t + s) log (t + s)) = **O((t + s) log (t + s))**.

Kevin Atienza

D: Eliens Slurs

- **Gotcha:** FFT mod a prime $m2^k + 1$ may fail!
- We need two such primes $\approx 10^9$ to be completely sure $\sum (X[i] - Y[i])^2 (X[i] - Y[i] + 1) (X[i] - Y[i] - 1)$ is nonzero.
  - More primes may be needed for our $d = 6$ expression.
- Doubles the # of FFTs. Should still pass though
  - if implemented well

Thank you!

- **Gerard Francis Ortega**
- **Kevin Charles Atienza**
- **Rene Josiah Quinto**
- **Samsung R&D Institute Philippines**

- **A: Alien Gordon Ramsey** - Atienza
- **B: Cult of Wah!** - Samsung
- **C: Dethrone Antares Now** - Ortega
- **D: Eliens Slurs** - Atienza
- **E: ... Parallel Universes** - Atienza
- **F: Flow Maximal** - Ortega
- **G: Irreversible Events** - Quinto
- **H: Not Just an NP-Hard Problem** - Ortega
- **I: Ominous Acids** - Atienza
- **J: Sensor Logs** - Samsung
- **K: Star Seeker's Socks** - Samsung
- **L: Starquake!** - Ortega
- **M: TheBuzz** - Samsung