Competitive Programming and Mathematics Society

O-Week contest debrief

CPMSoc

Table of contents

Welcome

- Join our subcom!
- Mathematics workshops will (probably) run every odd-numbered week (1, 3, 5, ...)
- Programming ones are every other week
- Slides will be uploaded on website (unswcpmsoc.com)
- Competitive maths ain't so competitive!

Quick Sum

- Quick! Sum!
- If you didn't add all the numbers within the first second of the contest starting you missed out on winning a free lamborghini. L
- $\frac{2023(2023+1)}{2}$

Drawing Aces

- If 4 of the cards have hearts suits, then one of the aces must be a hearts card
- Probability that one ace is a heart is $\frac{1}{2}$
- Probability three more hearts are selected is $\frac{12}{48} \times \frac{11}{47} \times \frac{10}{45}$ (exclude all aces since we've already selected one as a heart)
- Thus total probability is $\frac{1}{2} \times \frac{12}{48} \times \frac{11}{47} \times \frac{10}{46} = \frac{55}{8648}$

Find a Function

- Substitute $x = 1$: $f(y) = f(1)y$
- Let $f(1) = A$ for any real constant A.
- Substituting back: $x \cdot A \cdot y - y \cdot A \cdot x = 0$, which always holds.
- So $f(x) = Ax$ for any real constant A covers all solutions.

Triangular Edges

- Let the number of triangles a vertex touches be its "degree".
- Call the number of vertexes with degree i n_i, where $i \in \{1, 2, 3, 4, 5, 6\}$
- The total degree counts each triangle 3 times, so if there are n triangles, we have
- $\sum_{i=1}^{6} in_i = 3n$. Taking modulo 3, we have the sum
- $n_1 + 2n_2 + n_4 + 2n_5 = (n_1 + n_4) + 2(n_2 + n_5) = 2x + y = 0 \bmod 3$
- So $2x + y$ is divisible by 3.

Horrendously Complex

$$\frac{3x^2+12x+11}{(x+1)(x+2)(x+3)} = \frac{\frac{\mathrm{d}}{\mathrm{d}x}\left((x+1)(x+2)(x+3)\right)}{(x+1)(x+2)(x+3)} = \frac{1}{x+1}+\frac{1}{x+2}+\frac{1}{x+3}.$$

So,

$$\begin{aligned}
&\sum_{i=1}^{7} \frac{3\omega_i^2+12\omega_i+11}{(\omega_i+1)(\omega_i+2)(\omega_i+3)} \\
=&\sum_{i=1}^{7}\left(\frac{1}{\omega_i+1}+\frac{1}{\omega_i+2}+\frac{1}{\omega_i+3}\right) \\
=&\sum_{i=1}^{7}\frac{1}{\omega_i+1}+\sum_{i=1}^{7}\frac{1}{\omega_i+2}+\sum_{i=1}^{7}\frac{1}{\omega_i+3} \\
=&\frac{7\times 1^6}{1^7+1}+\frac{7\times 2^6}{2^7+1}+\frac{7\times 3^6}{3^7+1} = \frac{2626393}{282252}
\end{aligned}$$

Manhattan's Quadrilateral

- Sample two points from the red edge and two from the blue
- Shape is concave, so perimeter is
  $2 \times$ width $+ 2 \times$ height $= 2 \times$ (max blue $-$ min red) $+ 2 \times$ (max all $-$ min all)
- Calculating expected values is linear, so just take average of each of these values
- Average maximum of n points is $\frac{n}{n+1}$ (find CDF, get PDF, calculate $\int_0^1 xP(x)\mathrm{d}x$)
- So answer is $2 \times (\frac{5}{3} - \frac{1}{3}) + 2 \times (\frac{4}{5} - \frac{1}{5}) = \frac{58}{15}$

Addition

- This one was also a Lamborghini if you solved it in the first second of the contest.

Binary Help

- Just check every ascending power of 2 until you find one that is larger than N
- This is only $O(\log N)$, since N was only up to $10^{18} \approx 2^{60}$
- Also could use binary, check the most significant bit -> then just set next bit to 1 and all other bits to 0
- $2^{\lceil \log_2 (N+1) \rceil}$

Counting Rectangles

- Brute force - count every possible top, left, bottom, right edge of rectangle ($O(W^2H^2)$)

Counting Rectangles

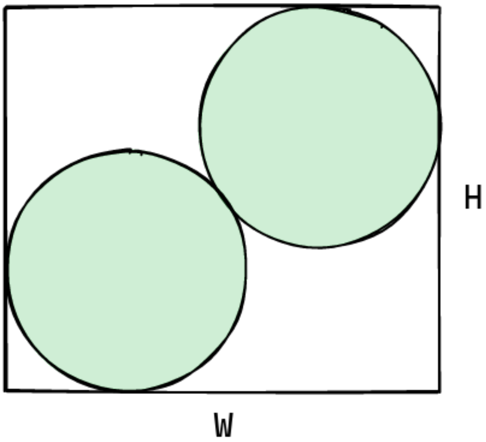
- Pick any possible left and right edge ($\frac{W(W+1)}{2}$), then any possible top and bottom edge ($\frac{H(H+1)}{2}$)
- All combinations of these $= \frac{W(W+1)H(H+1)}{4}$

Burger

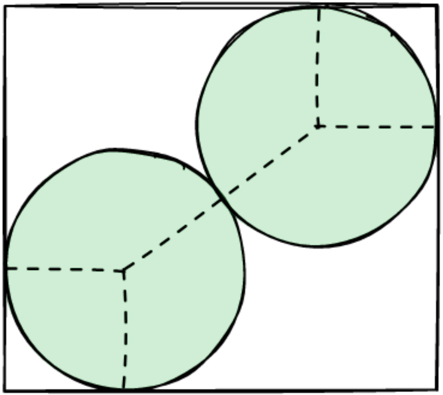
- Can be solved either mathematically or programmatically
- Both solutions require some maths

Burger

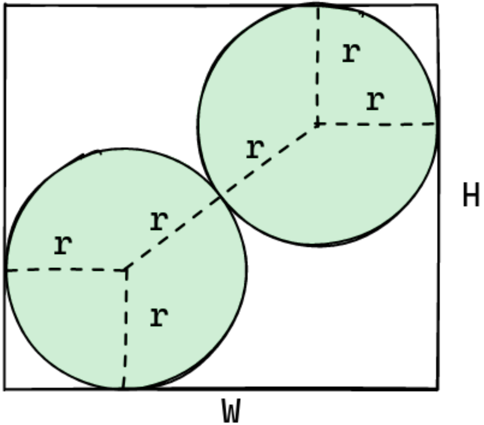
CPMSOC



Burger

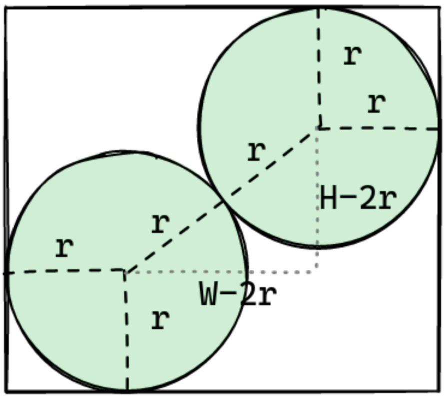


Burger

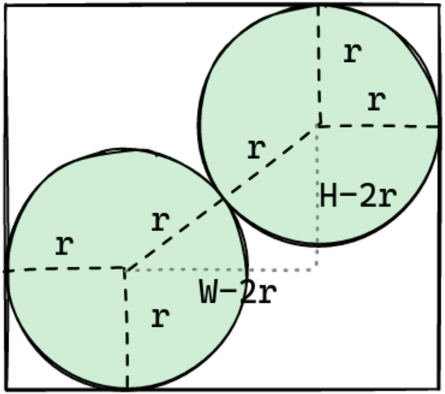


Burger

CPMSOC

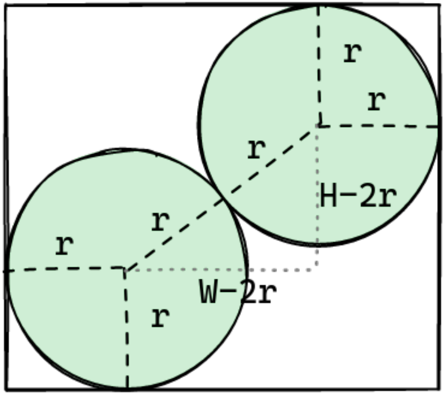


Burger



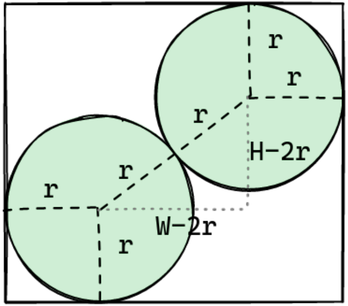
$$(2r)^2 \leq (H - 2r)^2 + (W - 2r)^2$$

Burger



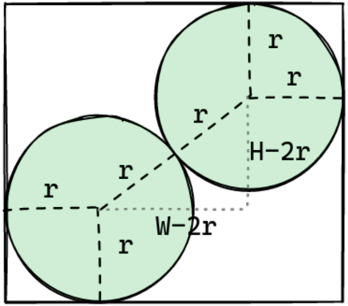
$$d^2 \le (H-d)^2 + (W-d)^2$$

Burger Maths Solution



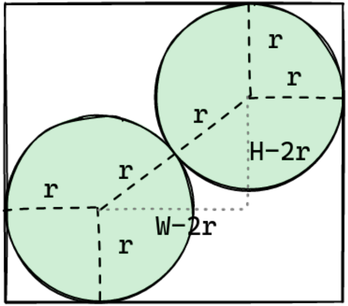
$$d^2 \leq (H-d)^2 + (W-d)^2$$

Burger Maths Solution



$$d^2 \leq (H-d)^2 + (W-d)^2$$
$$d^2 \leq H^2 - 2dH + d^2 + W^2 - 2dW + d^2$$

Burger Maths Solution



$$d^2 \le (H - d)^2 + (W - d)^2$$
$$d^2 \le H^2 - 2dH + d^2 + W^2 - 2dW + d^2$$
$$0 \le H^2 + W^2 - 2dH - 2dW + d^2$$

Burger Maths Solution

CPMSOC

$$\begin{aligned}
d^2 &\le (H-d)^2 + (W-d)^2 \\
d^2 &\le H^2 - 2dH + d^2 + W^2 - 2dW + d^2 \\
0 &\le H^2 + W^2 - 2dH - 2dW + d^2 \\
0 &\le H^2 + W^2 - 2d(H+W) + d^2
\end{aligned}$$

Burger Maths Solution

$$
\begin{aligned}
d^2 &\leq (H-d)^2 + (W-d)^2 \\
d^2 &\leq H^2 - 2dH + d^2 + W^2 - 2dW + d^2 \\
0 &\leq H^2 + W^2 - 2dH - 2dW + d^2 \\
0 &\leq H^2 + W^2 - 2d(H+W) + d^2 \\
0 &\leq (H+W)^2 - 2HW - 2d(H+W) + d^2
\end{aligned}
$$

Burger Maths Solution

CPMSOC

$$
\begin{aligned}
d^2 &\le (H-d)^2 + (W-d)^2 \\
d^2 &\le H^2 - 2dH + d^2 + W^2 - 2dW + d^2 \\
0 &\le H^2 + W^2 - 2dH - 2dW + d^2 \\
0 &\le H^2 + W^2 - 2d(H+W) + d^2 \\
0 &\le (H+W)^2 - 2HW - 2d(H+W) + d^2 \\
0 &\le (H+W)^2 - 2d(H+W) + d^2
\end{aligned}
$$

Burger Maths Solution

$$
\begin{aligned}
d^2 &\le (H-d)^2 + (W-d)^2 \\
d^2 &\le H^2 - 2dH + d^2 + W^2 - 2dW + d^2 \\
0 &\le H^2 + W^2 - 2dH - 2dW + d^2 \\
0 &\le H^2 + W^2 - 2d(H+W) + d^2 \\
0 &\le (H+W)^2 - 2HW - 2d(H+W) + d^2 \\
0 &\le (H+W)^2 - 2d(H+W) + d^2 \\
2HW &\le (H+W-d)^2
\end{aligned}
$$

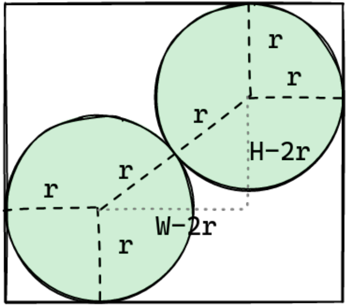
Burger Maths Solution

$$
\begin{aligned}
d^2 &\le (H-d)^2 + (W-d)^2 \\
d^2 &\le H^2 - 2dH + d^2 + W^2 - 2dW + d^2 \\
0 &\le H^2 + W^2 - 2dH - 2dW + d^2 \\
0 &\le H^2 + W^2 - 2d(H+W) + d^2 \\
0 &\le (H+W)^2 - 2HW - 2d(H+W) + d^2 \\
0 &\le (H+W)^2 - 2d(H+W) + d^2 \\
2HW &\le (H+W-d)^2 \\
\sqrt{2HW} &\le H + W - d
\end{aligned}
$$

Burger Maths Solution

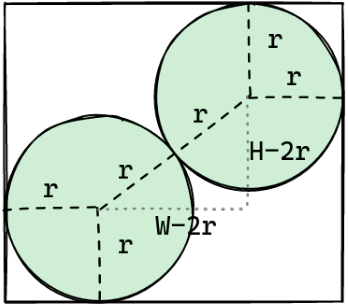
$$
\begin{aligned}
d^2 &\leq (H-d)^2 + (W-d)^2 \\
d^2 &\leq H^2 - 2dH + d^2 + W^2 - 2dW + d^2 \\
0 &\leq H^2 + W^2 - 2dH - 2dW + d^2 \\
0 &\leq H^2 + W^2 - 2d(H+W) + d^2 \\
0 &\leq (H+W)^2 - 2HW - 2d(H+W) + d^2 \\
0 &\leq (H+W)^2 - 2d(H+W) + d^2 \\
2HW &\leq (H+W-d)^2 \\
\sqrt{2HW} &\leq H+W-d \\
d &\leq H+W-\sqrt{2HW}
\end{aligned}
$$

Burger Programming Solution



$$(H - d)^2 + (W - d)^2 \geq d^2$$

Burger Programming Solution



$$(H-d)^2 + (W-d)^2 \geq d^2$$

Binary Search!

Burger Programming Solution

```
def can_fit(d, W, H):
    if d > W or d > H:
        return False
    elif d**2 > (W-d)**2 + (H-d)**2:
        return False
    else:
        return True

l, r = 1, max(W, H) + 1
while r - l > 1:
    m = (l+r) // 2
    if can_fit(m, W, H): l = m
    else: r = m

print(l)
```

Gerrymandering

- A modified version of maximum subarray sum, also called largest sum contiguous subarray
- Observation: when checking A, use a subarray sum by setting all voters for A to be 1 and all voters for B to be -1
- Two pointer technique or Kadane's algorithm to find largest sum for A and B

Gerrymandering

```
def best_margin(array, cand):
  best, curr_sum = 0, 0
  for vote in array:
    if vote == cand:
      curr_sum += 1
    else:
      curr_sum -= 1
    best = max(best, curr_sum)
    curr_sum = max(curr_sum, 0)
  return best

a = best_margin(array, 'A')
b = best_margin(array, 'B')
if a > b: print('A')
elif a < b: print('B')
else: print('BOTH')
print(max(a, b))
```

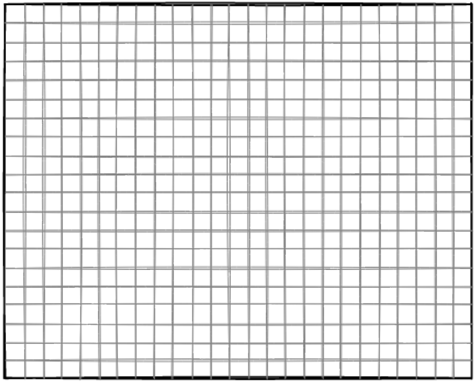
Shapes

CPMSOC

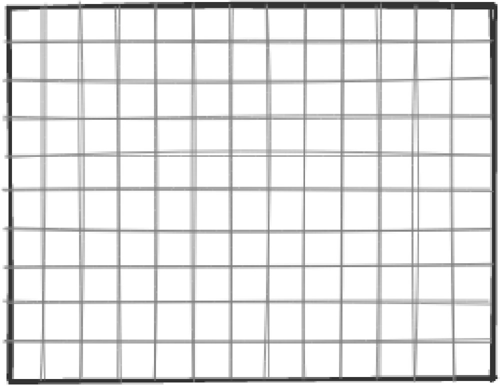
The key to this question is making observations.
First here are the rules:

- The shape must be convex. This means that everything between two filled squares are also filled. (1)
- The shape is vertically and horizontally symmetric (2)
- The shape must be exactly a height of H and a width of W. (3)

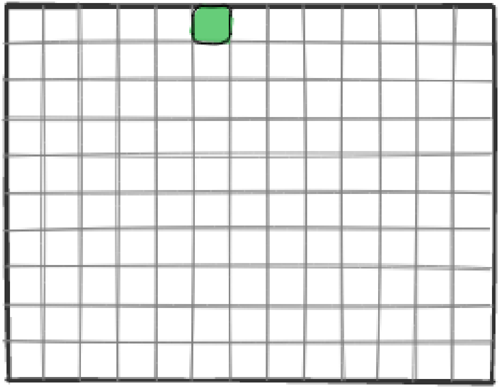
Shapes



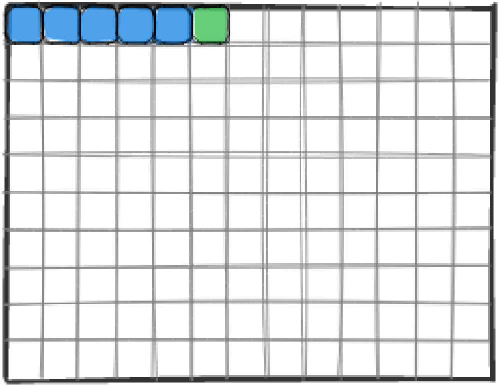
Shapes



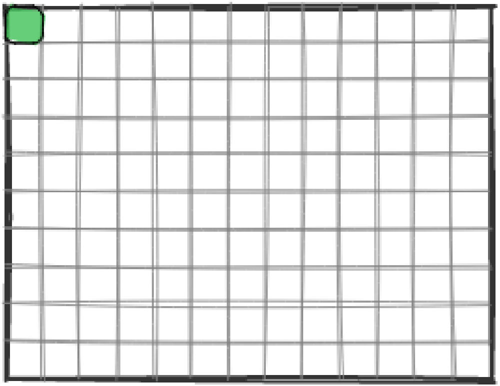
Shapes



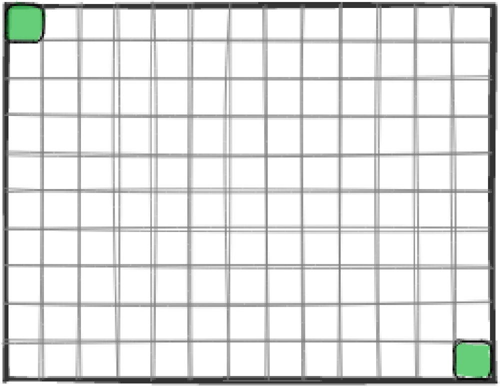
Shapes



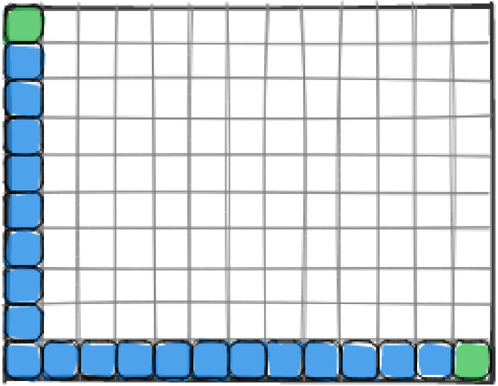
Shapes



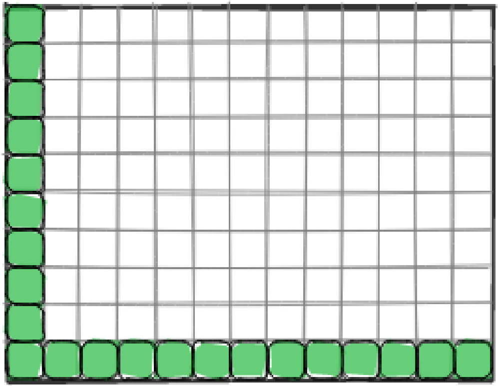
Shapes



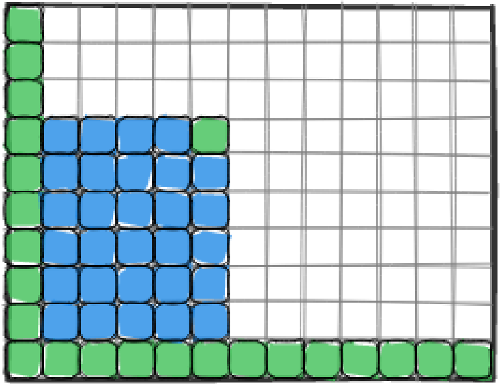
Shapes



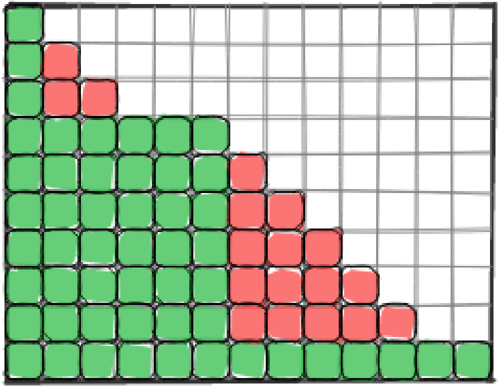
Shapes



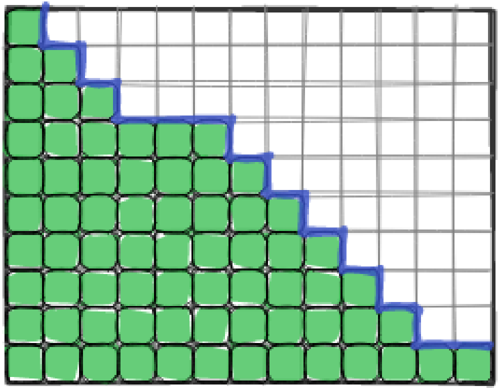
Shapes



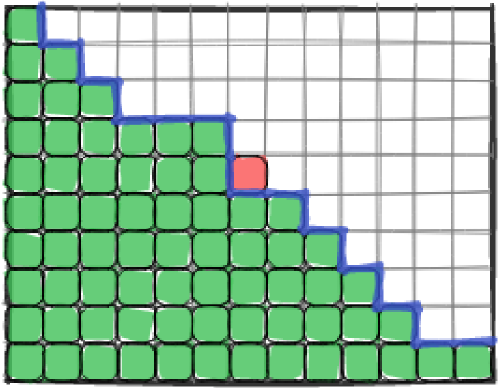
Shapes



Shapes



Shapes



Shapes

```
a_right[0][0] = 1

for i in range(R+1):
  for j in range(C+1):
    if j > 0:
      a_right[i][j] = (a_right[i][j-1] + a_down[i][j-1]) % mod
      b_right[i][j] = (b_down[i][j-1]) % mod
    if i > 0:
      a_down[i][j] = (a_right[i-1][j]) % mod
      b_down[i][j] = (a_down[i-1][j] + b_right[i-1][j] + b_down[i-1][j]) %

def solve(r, c):
  return (a_right[r][c] + a_down[r][c] + b_right[r][c] + b_down[r][c]) % m

print(solve(R, C))
```

Isaiah's Unsolved

For simplicity, number the nodes of the DAG according to their toplogical ordering so that the adjacency matrix is an upper triangular matrix. Let A be the adjacency matrix of the DAG, e be a column vector of n 1's and e^T be its transpose. Then, the sum of matrix entries is $e^T * A * e$. Note that A is nilpotent iff graph is acyclic, so let k be an integer such that $A^t = 0$.
Then $e(G) - o(G) = -e^T * A^0 * e + e^T * A^1 * e - ... + (-1)^t * e^T * A^(t-1) * e$. Since matrix multiplication is distributive, this equals: $e^T * (-A^0 + A - A^2 + ... + (-1)^t * A^(t-1)) * e$.
The middle part is a geometric series with matrices, which we can derive a formula for as long as $A + I$ is invertible. Note that, because our construction, $A + I$ is also upper triangular, and has 1's along its diagonals, so it's invertible (though this applies generally for any $A + I$ where A is nilpotent). Thus, $e(G) - o(G) = e^T * (-(A + I)^{-1}) * e$.
For simplicity, let's try and maximise/minimise the sum of entries of $(A + I)^{-1}$ (so now we are maximising/minimising $o(G) - e(G)$).
We can find the inverse matrix by performing row operations to transform

Isaiah's Unsolved

(A+ I | I) into (I | (A + I)$^{-1}$)$.Realisethat, because A +$
$Iisaninvertibleuppertriangularmatrixof0'sand1'sonly, wecandescribethisbythefollowingal$
$foreachrow, startingfromthebottomandgoingtothetop, checkallcolumnscontaininga1exclud$
$rowc_1, rowr - rowc_2, ..., rowr - rowc_k).$
Since all we care about is the sum of matrix entries, we can reduce this to only thinking about the sums of values on each row:
Starting with an array V of n 1's ([1, 1, ..., 1]), from i = 1 to n (1-indexing), we may choose to perform V[i] -= V[j] for unique values of j, where 1 <= j < i.
If we subtract by a net positive value, we might as well subtract the maximum possible amount we can for the sake of adding more value to our maximisation/minimisation later on, which will be the sum of all positive values, and similarly with a net negative value, which will be the sum of all negative values.
Therefore, we can reduce this problem further to considering two variables x, y (which start of as 0), where x is the sum of positive terms and y is the sum of negative terms, and in each of n turns, we can choose to add y + 1 to x or x - 1 to y. for n <= 4, programming this, the (min, max) seem to be (for o(G) - e(G), not e(G) - o(G)) (1, 1), (1, 2), (1

Isaiah’s unsolved

, 3), (0, 4) and I’m guessing the proof for the last step (where you show the maximum/minimum value of the x, y recurrence relation) involves saying that, after a certain point, the best way to "grow" x and y is by alternating which one you add to (maybe up until a number of steps?), since, excluding the constant factors of adding 1 and -1, this is just the fibonacci sequence, and when x - y is calculated, you get something along those lines? idk anyway plugging in $n \geq 5$ into oeis shows the values are (probably):

$$(-F_{n-1} + 2, F_{n-1} + 2),$$

where F_n is the nth Fibonacci number (where the first 5 are $1, 1, 2, 3, 5$)

Attendance form :D

Further events

Please join us for:

- Social session tomorrow
- Programming workshop next week
- Maths workshop in two weeks