Competitive
Programming and
Mathematics
Society

2025 CPMSOC Debrief & Prizes

Competitive Pizza Making Situated Online Contest

CPMSoc 2025 Competitions Portfolio

Table of contents

Disclaimer

This contest was called **C**ompetitive **P**izza **M**aking **S**ituated **O**nline **C**ontest, so is abbreviated as **CPMSOC**.

The society is **C**ompetitive **P**rogramming and **M**athematics **Soc**iety, so is abbreviated as **CPMSoc**.

Please note the difference in capitalisation.

Guess how much

In 2010, CPMSoc purchased 2 pizzas for "only" $10,000$ BTC. If we assume 1 BTC is now worth $\$135,000$ AUD, and 1 pizza costs $\$20$ AUD, how much money did CPMSoc lose?

Step 1: Calculate the current value of 10,000 BTC:

$$10{,}000 \text{ BTC} \times \$135{,}000 \text{ AUD/BTC} = \$1{,}350{,}000{,}000 \text{ AUD}$$

Step 2: Calculate the actual cost of 2 pizzas:

$$2 \times \$20 = \$40 \text{ AUD}$$

Step 3: Calculate the loss:

$$\text{Loss} = \$1{,}350{,}000{,}000 - \$40 = \$1{,}349{,}999{,}960 \text{ AUD}$$

Therefore, CPMSoc lost $\boxed{\$1{,}349{,}999{,}960}$ AUD.

Pizza is my F5VE!

You eat a lotta lettuce but youre toxic—

The average angle of the slices is the sum of the angles divided by the number of non-zero slices. We know that the angle sum for a full revolution is $360°$.

The number of non-zero slices for a circle is the same as the number of unique cuts. We can find the number of unique cuts by counting the total number of cuts, and subtracting the amount that were double counted. The amount that were double counted, which can otherwise be thought of as overlapping cuts, is given by $\gcd(X, Y)$, so the number of non-zero slices is given by $X + Y - \gcd(X, Y)$.

Therefore, the average angle per slice is $\dfrac{360°}{X + Y - \gcd(X, Y)}$

We can now use this formula to calculate the solution for each part.

Pizza is my F5VE!

Part 1: $X = 5,\ Y = 8$

$$\gcd(5, 8) = 1 \Rightarrow \text{Non-zero slices} = 5 + 8 - 1 = 12$$

$$\text{Average angle} = 360^\circ \div 12 = \boxed{30^\circ}$$

Part 2: $X = 12,\ Y = 18$

$$\gcd(12, 18) = 6 \Rightarrow \text{Non-zero slices} = 12 + 18 - 6 = 24$$

$$\text{Average angle} = 360^\circ \div 24 = \boxed{15^\circ}$$

Part 3: $X = 36,\ Y = 64$

$$\gcd(36, 64) = 4 \Rightarrow \text{Non-zero slices} = 36 + 64 - 4 = 96$$

$$\text{Average angle} = 360^\circ \div 96 = 3.75^\circ \approx \boxed{4^\circ}$$

FOC SUC Sum

Who do we know comes from the Republic of Pizza?

FOC SUC Sum

Who do we know comes from the Republic of Pizza?
Fibonacci!

What sequence might they be claiming to have invented?

FOC SUC Sum

Who do we know comes from the Republic of Pizza?
Fibonacci!

What sequence might they be claiming to have invented?
Fibonacci Sequence!

Does that look ... familiar ...?

FOC SUC Sum

Who do we know comes from the Republic of Pizza?
Fibonacci!

What sequence might they be claiming to have invented?
Fibonacci Sequence!

Does that look ... familiar ...?

```
FibOnaCci SeqUenCe < Look at what letters are capitalised
1  4  7   10 13 16 < Index of the capitalised letters
1  3  13 ...    987 < Fibonacci Sequence situated number
```

Therefore, the sum of the sequence is 6, and the exponent of the third term will be 13.

If-Then-Otherwise

I wasn't particularly concerned with how you would solve it, I didn't need to solve it anyway.

I got the facts from Cindy and wrote some 5 minute code the generate the statement.

If-Then-Otherwise

```
def f(layer):
    if layer == 6:
        global n_output
        cur_output = outputs[n_output]
        n_output += 1
        print("submit the term ", end = '')
        print(cur_output, end = '')
        return cur_output
    global n_statement
    print("if ", end = '')
    cur_statement = statements[n_statement]
    n_statement += 1
    print(cur_statement, end = '')
    print(" then ", end = '')
    x = f(layer + 1)
    print(" otherwise ", end = '')
    y = f(layer + 1)
    if cur_statement in correct:
        return cur_statement + " is true -> " + x
    else:
        return cur_statement + " is false -> " + y
```

1+1=3

The judge code uses the minute of the submission as a seed for its random number generator. Random number generators, when given the same seed, will give the same output.

As such, you can write a similar judge code locally, and test various seeds to find out what minutes to make submissions at.

In the 125 hours of the contest, there were a total of 73 one minute long windows where submissions would've been accepted by the judge.

During the initial five hour window before penalty begins to be applied, there are only two instances which would be accepted. The first occurs at 1:05 pm AEDT, and the second occurs at 4:54 pm AEDT.

Too Greedy

As we all know, this is the most difficult question in the entire contest. Achieving a score of 0 requires not only careful calculations, but also a bit of fate and Heavens will.

Too Greedy

As we all know, this is the most difficult question in the entire contest. Achieving a score of 0 requires not only careful calculations, but also a bit of fate and Heavens will.

It's so complicated that even I can't explain it so let's move on to the next question! :p

Wires

Note: Only one wire (or cheese string) is contaminated at a time, apologies if the wording of the question was ambiguous.

We thought it'd be funny to begin a contest with comp geom.

Implementation is left as an exercise for the reader.

Pizza Printing

Wires

Just kidding, there exists a much simpler solution, let's prove it first.

If we start at $(0, x)$, then we'll have $x - 1$ wires underneath us initially. Track how this amount changes as we move from $(0, x)$ to $(100, y)$.

There are three main cases:

1. The contaminated wire does not intersect with anything.
   - The amount of wires underneath the contaminated wire is unchanged.
2. A wire intersects the contaminated one from a lower y-value to a higher y-value.
   - The contamination jumps to the higher wire, meaning the amount of wires underneath remains constant.
3. A wire intersects the contaminated one from a higher y-value to a lower y-value.
   - The contamination jumps to the lower wire, meaning the amount of wires underneath remains constant.

Wires

Since it is guaranteed that there are no intersection points with three or more wires, we see that in all possible cases, the amount of wires under the contaminated wire remain constant, which as we discussed, was initially $x - 1$.

This means that we end up with $x - 1$ wires still underneath the contaminated wire at $(100, y)$.

Therefore, $\boxed{y = x}$, making for a very easy implementation.

Pizza Printing

Its basically just ASCII manipulation.

First, we will use the number 1 to derive the forbidden ASCII codes: p, i, z, and a.

```
l = bytes([111 + 1]).decode()            # 111 + 1 = 112  'p'
ll = bytes([101 + 1+1+1+1]).decode()     # 101 + 4 = 105  'i'
lll = bytes([111 + 11]).decode()         # 111 + 11 = 122  'z'
llll = bytes([100 - 1-1-1]).decode()     # 100 - 3 = 97  'a'
```

Pizza Printing

Next, we hardcode the string `print(x)`.

```
msg0 = l + "r" + ll + "nt(x)"  # 'p' + 'r' + 'i' + 'nt(x)'  "print(x)"
```

Then, we can construct the string `"pizza"`.

```
msg1 = l + ll + lll + lll + llll  # 'p' + 'i' + 'z' + 'z' + 'a'  "pizza"
```

Pizza Printing

Finally, we use the `exec()` function to execute the Python code.

```
exec(msg0, {"x": msg1})  # Executes "print(x)" with `x = "pizza"`
```

Tadaa! (pretend this is a pizza emoji)

Circular Logic

Let's analyse some examples in the form `INPUT` → `OUTPUT`

`YES` → `NO`

`NO` → `YES`

`LEGIT` → `NO`

`UNTRUSTWORTHY` → `YES`

`GENUINE` → `NO`

`DISINGENUOUS` → `YES`

It seems that the rule is to return `NO` for all 'positive' words and to return `YES` for all 'negative' words. With a title like *Circular Logic* this makes sense.

Clearly for this problem we must use sentiment analysis AI.

Circular Logic

- jk

Circular Logic

Let's look at more examples.

$$\texttt{truth} \rightarrow \texttt{NO}$$
$$\texttt{falsehood} \rightarrow \texttt{YES}$$
$$\texttt{TrUE} \rightarrow \texttt{NO}$$
$$\texttt{FALSE} \rightarrow \texttt{YES}$$
$$\texttt{2 + 3 = 5} \rightarrow \texttt{NO}$$
$$\texttt{2 + 3 = 4} \rightarrow \texttt{YES}$$

For the first 2 examples, why did CPMSoc opt for the clunky `truth` and `falsehood` over simply `true` and `false`? In the next 2 examples, why when they finally used `TRUE` and `FALSE` is the `r` randomly lowercase? Was there a reason they opted for `2 + 3 = 5` over just `1 + 1 = 2`?

Circular Logic

Circular Logic. Return `YES` if the input contains a circle, else return `NO`.

```
int main() {
    string s;
    getline(cin, s);
    string os = "ABDOPQRabdegopq04689";
    for (char c : s) {
        for (char d : os) {
            if (c == d) {
                cout << "YES" << endl;
                return 0;
            }
        }
    }
    cout << "NO" << endl;
}
```

Zero

The problem asks you to consider a grid of random numbers between 1 and $2^{40} - 1$ and find a path with a XOR of zero.

The random generation is quite important in this problem. It suggests that you don't want an algorithm that solves the problem deterministically, but rather an algorithm that works with high probability.

It seems impossible to get a XOR of 0 with a probabilistic approach, each new square you try will move the current XOR to a practically random number between 0 and $2^{40} - 1$. However, note that $x \oplus x = 0$ for all x. Therefore, if you find two paths with an equal XOR and with adjacent endpoints, you can merge them to form a path with XOR 0.

This allows us to use an algorithm inspired by the birthday attack.

Zero

When $N = 2000$, we send $2e6$ paths to the left from $(0, 1e6)$ which travel left then down and end on unique squares. Similarly, we send $2e6$ paths to the right from $(0, 1e6)$. If a path to the left and to the right have equal XOR, then we have a solution.

The reason this works is related to the birthday paradox:

- If we send $1e6$ paths to the left, each path on the right has a $\frac{1e6}{2^{40}} \approx \frac{1}{1e6}$ chance of having the same XOR on the left.
- On average, we need to send $1e6$ paths to the right for this to happen. Therefore, it is likely that we find a collision if we send $1e6$ paths both ways.
- If we do this with $2e6$ paths both ways, the chances of this working increases to $1 - \left(\frac{2^{40} - 2e6}{2^{40}}\right)^{2e6} \approx 0.973695$.

Zero

To get this working for full, we can just send out random paths towards the left and the right. If we send out $2e6$ in both directions, it should pretty much work the same way.

Technically, these XOR of these random paths are related to each other. But it worked when I implemented it so whatever.

We don't have time for a program to run in $O(2e6 \cdot \text{path\_length})$. Instead, we generate these paths recursively, and keep a log of how each path differs from the last. We then store the index of the path with it's XOR.

When we find two indices with the same XOR, we look back through the logs to reconstruct the path. This now works in roughly $O(\text{num\_paths})$.

Zero

```
void dfs1(int x, int y) {
    done[x][y] = true;
    cval ^= arr[x][y];
    lg1.push_back({x, y});

    vals.push_back({cval, -lg1.size()});
    vector<pair<int, int>> adj = {{x + 1, y}, {x, y + 1}, {x - 1, y}, {x, y - 1}};
    for (auto [nx, ny] : adj) {
        if (lg1.size() >= lim) break;
        if (nx < 0 || nx >= n / 2 || ny < 0 || ny >= n) continue;
        if (done[nx][ny]) continue;
        dfs1(nx, ny);
    }

    done[x][y] = false;
    cval ^= arr[x][y];
    lg1.push_back({-1, -1});
}
```

Matching

The question asks to find the highest value of K such that we can pair values so that the sum of each pair is divisible by K.

We can test whether an certain K works in $O(N)$ time, we just take the reminder of each number modulo K and we match them up. However, there seems to be too many values of K to try, if we try all numbers between 1 and 200000 it will TLE.

However, we can notice that the value of K must divide the sum of the numbers. It turns out that the number of factors of a number is often not that high, so we can just try all values of K which divides the sum.

Ruh Roh

The problem reduces to the following problem.

You have an array, `A` containing n positive integers. Available to you are r jumps. To jump from index i to $i + \rho$, decrease `A[i]` by ρ. ρ cannot exceed `A[i]`. Begin at index 0 and end at index n (the index after the final element in `A`) in at most r jumps. Complete your journey while minimising the max value of `A`.

Ruh Roh

If we know that all `A[i]` are <= k, then all `A[i]` are also <= k + 1. Thus the function "is it possible to complete a journey such that all `A[i]` are at most k?" is monotonic in k and hence the minimum k that returns `TRUE` is binary searchable.

To compute the above function we can utilise a sweep line algorithm to find the minimum number of jumps that connect `A[i]` that are > k and reach the end. If we perform a jump of size ρ from each of these `A[i]`, then as long as ρ is greater then or equal to the difference of the `A[i]` to k then all `A[i]` will end up being less than or equal to k. However if our jumps jump over a `A[i]` such that `A[i]` > k or the total jumps exceed r, then the route is invalid and we return false.

This algorithm is $O(n)$. Using binary search we require at most $O(log(n))$ queries. Hence the algorithm is $O(nlog(n))$, which fits within our time constraint.

Top 7

1. caterpillow
   - 9.18 points
2. awu
   - 14.62 points
3. SirLemonMeringue
   - 15.68 points
4. lmkae
   - 24.13 points
5. programmer123
   - 32.75 points
6. cosintheta
   - 41.25 points
7. poko
   - 42.15 points

Other Prizes

Women and Gender Minorities

- iwannalearncode
  - Official rank 8; 64.29 points
- (Username redacted)
- chimken_wingz
  - Official rank 16; 83.77 points

First years

- pigeonhole
  - Official rank 14; 81.42 points
- hsk7
  - Official rank 20; 94.7 points

Randomised Prizes

Note: In order to remain eligible for the raffle prizes, you must not have already won any of the previous prizes.

Highest non-10 score for Too Greedy

- Ixbixbam (9.81/10)

Raffle prizes

- tlgeotau
- Tselmeg_Tslmg
- thegoodstuffshak
- rizzler

Attendance form

Further events

Please join us for:

- This is the last proper event of the term :<
- Term 2 Week 1 will have a Launch Week Contest!
- Pizza time!