ACM ICPC 2017 - Gwalior Regional

Presentation of solutions

1. Coupon System	Easy
2. Optimize The Slow Code	Easy
3. A Tale of Three Squares	Easy-medium
4. A Tale of Two Right Angled Triangles	Easy-medium
5. Chef and Triangles	Medium
6. Queries on Tree Again	Medium
7. Proprietary Probabilistic Problem	Medium
8. Organize The Wallet	Medium-Hard
9. Find Lighted Area	Hard

Problem 1
Coupon System

Accepted: 98+

First solved by: **Accio ACs**
Delhi Public School, Kalyanpur, Kanpur

00:03:34

Author: Praveen Dhinwa

Coupon System

Main Idea - Find the maximum discount for a particular level across all cities and break ties using the id of the cities.

- **Udit Sanghi** (Class 9 school student) solved this the earliest in Python at 3 minutes.
  - 4 minutes before any other submission!

Net Complexity - O(N)

Problem 2
Optimize The Slow Code

Accepted: 96+

First solved by: **los_pepes**
Indian Institute of Technology, Varanasi Uttar Pradesh

00:07:35

Authors: Akashdeep Nain, Praveen Dhinwa

Optimize The Slow Code

Main Idea - For each distinct X[i] you only care about the maximum Y[i] associated with it. The 3 largest Y[i]'s contribute to the answer.

- For each distinct value of X[i] only store the maximum Y[i] you get in the input.
  - Can do this with maps, or a linear scan after sorting.
- Now sum the largest 3 values out of these Y[i]s.
  - If number of distinct Y[i]s < 3, answer is 0.

Net Complexity - $O(N*\log_2 N)$

Problem 3
A Tale of Three Squares

Accepted: 87+

First solved by: **karma_**
MNNIT Allahabad

00:31:43

Author: Praveen Dhinwa

A Tale of Three Squares

Main Idea - Consider the squares as line segments. For these line segments the line segment in the middle does NOT matter because K is equal for every square.

- Each square can be considered as a horizontal line segment, since the height of all the squares is equal.
- Now the second reduction is to remove the middle line segment.
  The intuition - Since the K is equal for all the 3 squares, so if we can make the first and the third square intersect, then we could obviously move the second square in such a way that it is a superset of this intersection region.

- Let X_1, X_2, X_3 be the x^{th} coordinate of the 1^{st}, 2^{nd} and 3^{rd} square in ascending order respectively.
- Now take 2 cases -
  - $X_3 - X_1 < 2.K$, here the 2 centers can be coincided, so the answer is a^2
  - $X_3 - X_1 > 2.K$, here move X_1 += 2.K and now check what is the intersection region.

Net Complexity - O(1)

Problem 4
A Tale of Two Right Angled Triangles

Accepted: 70+

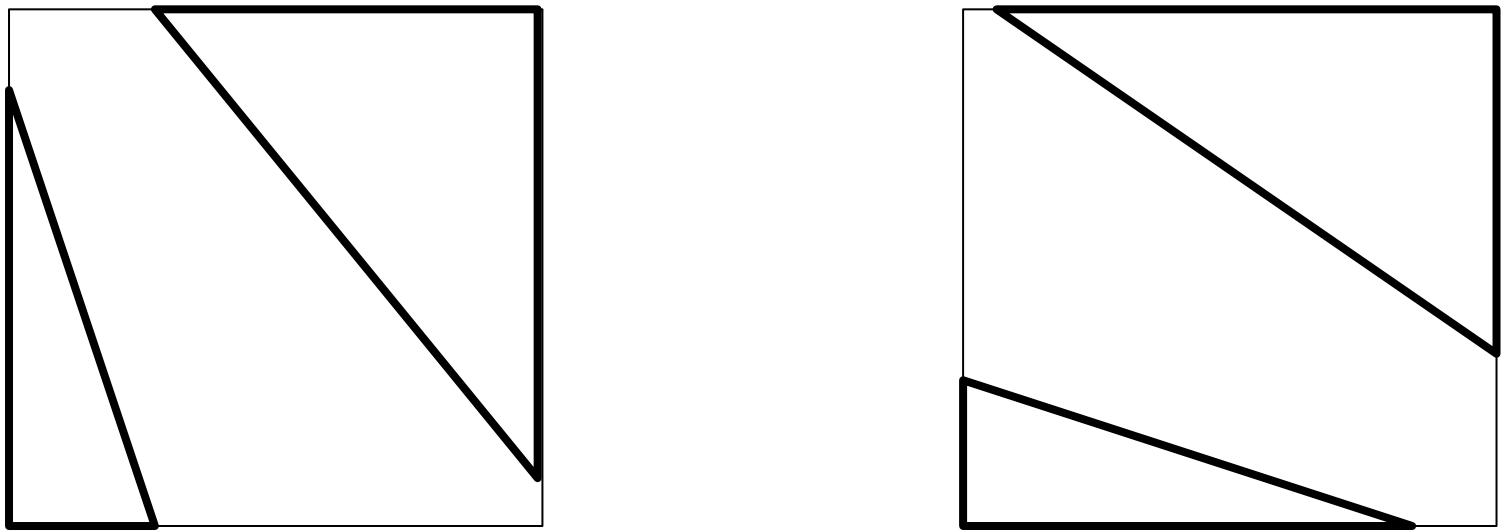
First solved by: **importedpandas**
Delhi Technological University

00:29:03

Author: Praveen Dhinwa

A Tale of Two Right Angled Triangles

Main Idea - Consider all 4 cases of putting the 2 triangles in different orientation.



Example of 2 different orientations.

- The broad idea is to put the first triangle at bottom left most position and the second triangle at top right triangle.
- Let ***a, b, c*** be the *base, height and hypotenuse* of the first triangle.
- Let ***d, e, f*** be the *base, height and hypotenuse* of the second triangle.
- Keep in mind that it is an invalid scenario if the *min(a, d)* is greater than the base of the rectangle and similarly for the height of the rectangle.

Net Complexity - O(1)

Problem 5
Chef and Triangles

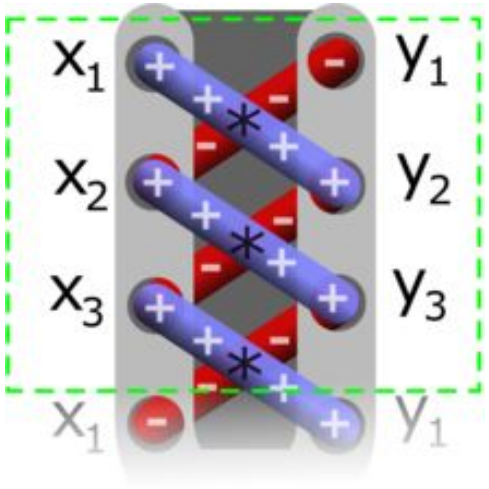
Accepted: 36+

First solved by: **TDCs**
International Institute of Information Technology, Hyderabad

00:50:32

Author: Utkarsh Saxena

Chef and Triangles



Main Idea - Divide the problem into 2 cases concerning the shoelace formula and work out the exact details.

Shoelace formula:

Area = ½ * $|X_1.Y_2 + X_2.Y_3 + X_3.Y_1 - Y_1.X_2 - Y_2.X_3 - Y_3.X_1|$

Note that $1 \leq X_i \leq 3$. Thus, we have only 2 types of triangles:

Case 1: 2 vertices of the triangle have the same x-coordinate.

- Area reduces to ½ * $|(Y_2 - Y_1) * (X_1 - X_3)|$
- Pick all possible pairs of points (X_1, Y_1) and (X_1, Y_2) to compute the sum.
- Clearly, this case is $O(N^2)$

Case 2 : All 3 vertices of the triangle have different x coordinate.

- Area reduces to ½ * $|Y_1 + Y_3 - 2.Y_2|$
- We can remove the absolute sign by considering two cases:
  - $Y_1 + Y_3 > 2.Y_2$ - Fix Y_1 and Y_3 and find sum of all valid Y_2 satisfying this inequality. This can be done efficiently by pre-computing partial sums.
  - $Y_1 + Y_3 < 2.Y_2$ - Very similar to the case above.

Net Complexity - $O(N^2)$

Problem 6
Queries on tree again

Accepted: 13+

First solved by: **LongTimeNoC**
Indian Institute of Technology Kanpur

01:34:27

Authors: Anudeep Nekkanti, Praveen Dhinwa

Queries on tree again

Main Idea - While making an update only make the update on the ancestors, and for the subtree lazily keep the update on the current node. During query you can add this lazy for all the ancestors that are not too far away from the queried node.

- For **update** on node *u* -
  - Easy Part: Updating Ancestors - Iterate through all ancestors of *u* within distance 60 and update its value.
  - Tricky Part: Updating Subtree - Brute Force is too slow.
  - Define $L_{u,\,d}$ as the contribution of node *u* for all the nodes which are at distance *d* from *u* within its subtree.
  - Lazy Updates - Make the update by adding 2^d in $L_{u,\,d}$ for the node *u* for all $1 \le d \le y$.

- For **query** on the node *u* -
  - All updates made to the descendants of *u* have already been successfully propagated to *u*.
  - Only need to consider the lazy updates i.e. we add $\sum L_{p(k,\ u),k}$ for $1 \le k \le 60$, where $p(k, u)$ denotes the k^{th} ancestor of *u.*

Net Complexity - O(N + 60 * Q)

Problem 7
Proprietary Probabilistic Problem

Accepted: 5+

First solved by: **Supercalifragilistic**
Indian Institute of Technology Madras

00:16:31

Author: Raaghav Passi

Proprietary Probabilistic Problem

Main Idea - Expected rank of a ball = 1 + Expected number of balls taken out before it. The latter quantity is much easier to compute.

- Expected rank of i^{th} ball = A(i) = 1 + Expected number of balls taken before it.
- Let the Expected number of balls taken out before i^{th} ball be E(i).
- Let X_j be the event that the j^{th} ball is picked before the i^{th} ball.
- Here $E(i) = E[X_1 + X_2 + \ldots + X_{i-1} + X_{i+1} + \ldots + X_n] = E[X_1] + E[X_2] + \ldots + E[X_n]$ using linearity of expectation.
- Define P(j, i) = The probability that j^{th} ball is picked before i^{th} ball = $R_j / (R_j + R_i)$
- The formula for $E[X_i] = 1 * P(j, i) + 0 * (1 - P(j, i)) = 1 + P(j, i)$
- Then, $A(i) = 1 + E(i) = 1 + \sum(1 * P(j, i))$

Net Complexity - $O(N^2)$

C++ Implementation

```
for(int i = 0; i < n; i++) {
    double A = 0, E = 0;
    for(int j = 0; j < n; j++) {
        if (i == j) continue;
        E += ((double)R[j])/(R[i] + R[j]);
    }
    A = 1 + E;
    printf("%.7lf ", A);
}
```

Problem 8
Organize The Wallet

Accepted: 3+

First solved by: **JustAnotherTeam**
DA IICT, Gandhinagar

01:24:31

Author: Kazi Hasan Zubair
Modified By: Praveen Dhinwa, Animesh Fatehpuria

Organize The Wallet

Main Idea - To approach the dual of this problem which is easier to compute.

- Let's define a sequence to be "good" if every distinct denomination in this sequence appears as a contiguous segment.
  - Example - [1, 1, 2, 3, 3] is good but [1, 2, 1, 3, 3] is not.
- Consider the indices that we don't touch in any (not necessarily optimal) solution. What can we say about those indices?
  - The relative order of those indices remain the same in the final state.
  - Thus, these untouched indices must form a "good" sequence!
- Minimizing number of moves ⇔ Maximizing the untouched subsequence.
  - We proved one direction above. The other direction has a similar argument, so we'll leave it as an exercise.

- New Goal: Find the longest “good” subsequence!
- Let **A** denote the (normalized) input array.
- Observe that there are only 7 distinct denominations. Thus a solution that is exponential in the number of denominations is acceptable.
- We use a relatively standard Bitmask DP to solve it.
- ***f(pos, current, mask)***: Maximum length of any **good** subsequence which starts at the index **pos** where the *leftmost denomination selected* is **current** and **mask** denotes the set of all the denominations used till now.
    - Example: **A = [1, 2, 3, 1, 2, 0, 0]**.
    - ***f(2, 2, {0, 2}) = 4*** as it refers to the subsequence **[2, 2, 0, 0]** where only elements from 2^{nd} position onwards are taken into consideration and the leftmost element is used is 2 and {0, 2} are the 2 denominations used.

- We present the transitions in a *top-down* implementation of this DP:
- Transitions from ***f(pos, current, mask):***
  - Case 1: We don't take the value at this position, i.e we call ***f(pos + 1, current, mask)***
  - Case 2: If ***A[pos] = current***, try to extend the current window i.e. we consider ***1 + f(pos + 1, current, mask)***
  - Case 3 - If ***A[pos] is NOT in the mask***, we can start a new denomination from here i.e ***1 + f(pos + 1, A[pos], mask $\cup$ A[pos])***
  - Take maximum over all valid cases to compute each state.
- Caveat: Naive recursive implementations consume a lot of memory and may time out even with the generous 7 second time limit. It is safer to implement it bottom-up!

Net Complexity: $O(7 * 2^7 * N)$ states and $O(1)$ transition $\rightarrow$ $O(7 * 2^7 * N)$

Problem 9
Find Lighted Area

Accepted: 0+

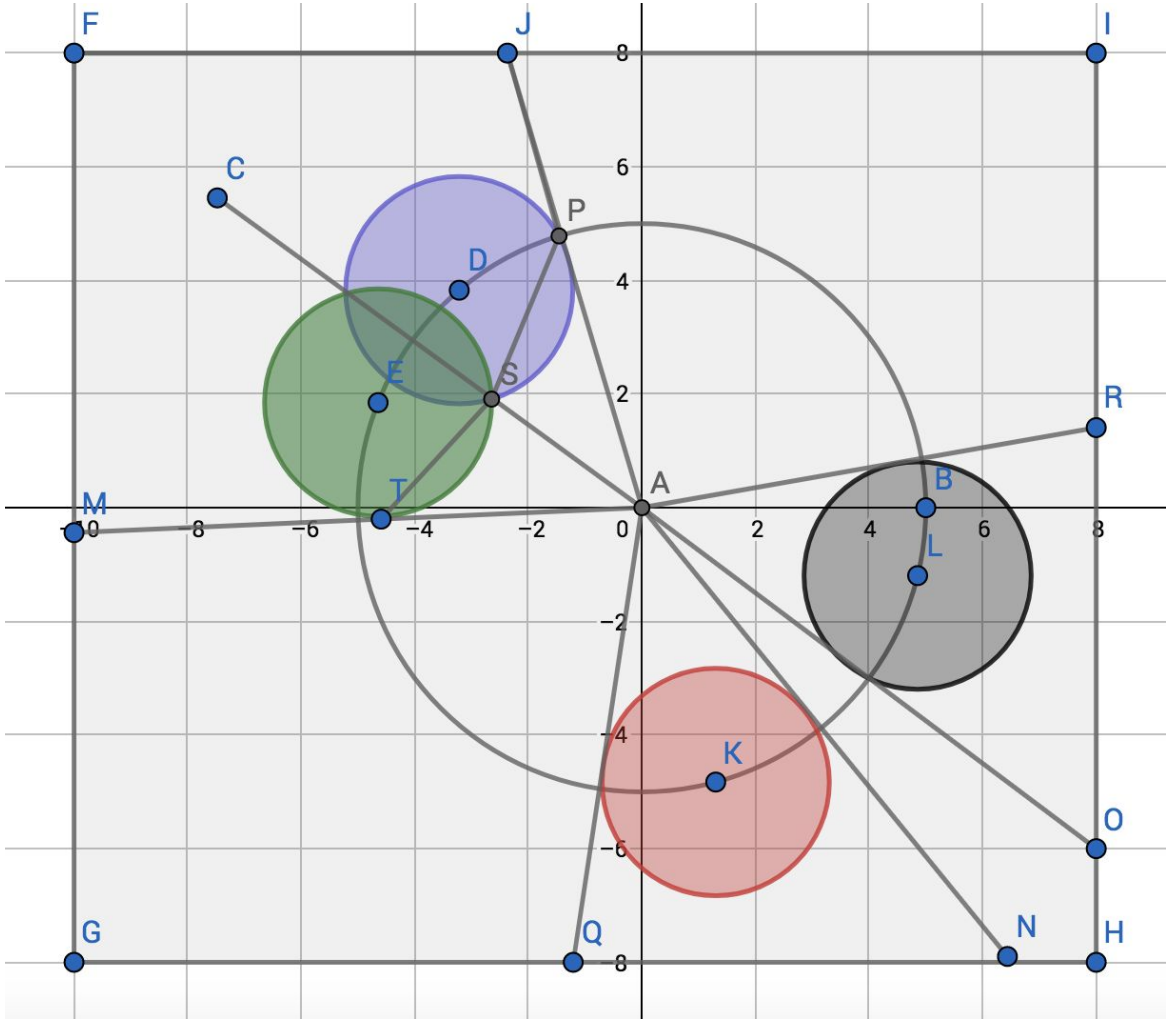
First solved by: ?
?

NA

Author: Praveen Dhinwa,Tester: Triveni Mahatha

Find Lighted Area

- The problem can be solved by doing a radial sweep along the “ring” of discs. Here, we use the fact that the centers of all discs are equidistant from origin.
- Sort the given circles according to polar angle.
- For every two adjacent circles c_1 and c_2 (in the sorted order) do -
  - If they intersect, draw a ray from origin to point of intersection
  - If they don’t draw two rays which which are tangent to the circles c_1 and c_2 respectively. The tangent on c_1 will be the one which is *radially* closer to c_2. And similarly for c_2.


F
J
I
C
P
D
E
S
R
M
T
A
B
L
K
O
G
Q
N
H
8
6
4
2
−2
−4
−6
−8
−10
−8
−6
−4
−2
0
2
4
6
8

- Now the plane is divided into segments about the origin. Each segment is two rays from origin and either blocked by a circle / quadrilateral / triangle.
- You need to compute the area of those segments and add up to give the final answer.
- Time complexity for sorting is $O(N*\log_2 N)$.
- Careful implementation is needed. Corner cases are - Only one circle, almost touching two circles.

Net complexity - $O(N*\log_2 N + N*c)$

Here O(c) is a significant constant due to calculations involving sin/cos/atan, etc.