2021 Mini Competition 0 Editorial

2021-02-14

香港電腦奧林匹克競賽
Hong Kong Olympiad in Informatics

M2101
Social Distancing and Exam

Possible condition

Compute Total Area ***A*** = ***N*** x ***M***

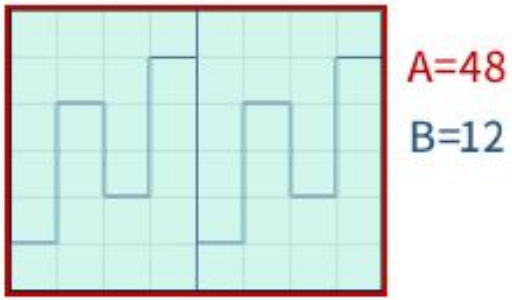
Obviously, if ***A*** is not visible by ***K***, then there is no solution.

Does that mean if ***A*** is divisible by ***K***, there would be a solution?

Yes! We will show the proof in the next slide.

M2101 Solution

- Total Area ***A*** = ***N*** x ***M***
- Target Area for each student **B** = **A** / **K**



Let's factorize

$N = p_1 \times p_2 \times p_3 \times \ldots \times p_n$ $\quad M = q_1 \times q_2 \times q_3 \times \ldots \times q_m$

$A = p_1 \times p_2 \times p_3 \times \ldots \times p_n \times q_1 \times q_2 \times q_3 \times \ldots \times q_m$

$B = r_1 \times r_2 \times r_3 \times \ldots \times r_b$

Because ***B*** divides ***A***, there always exist a one to one mapping from *r* to *p* and *q*.

For example in subtask 1, ***N*** = 2 x 3 ***M*** = 2 x 2 x 2, ***A*** = 2 x 3 x 2 x 2 x 2.

If ***K*** = 4, ***B*** = 48 / 4 = 12 = 2 x 2 x 3,

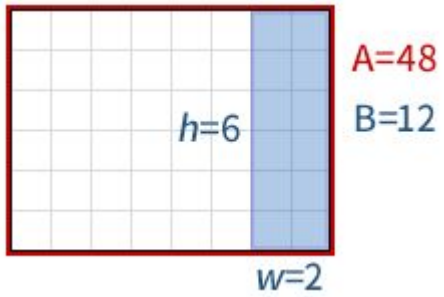
then one option is to pick prime factor 3 from ***N*** and 2 x 2 from ***M***.

M2101 Solution

We can first decide which factors to pick from ***N***.
In fact, it is always better to pick as many common factors as possible.

Therefore, we can compute ***h*** = GCD(***N***, ***B***).
Using the previous example, ***h*** = GCD(6, 12) = 6.
What does this 6 mean? It means each area will be ***h*** = 6 rows tall.

Finally, we can compute ***w*** = ***B*** / ***h*** = 12 / 6 = 2.
It means each area will be ***w*** = 2 columns wide.



M2101 Other notes

- 4 colour theorem says that it is always possible to use at most 4 colours to paint areas such that no 2 neighbouring areas share the same colour.
- Hint: In this task, it is possible to use only 2 colours.

- Using 100000 x 5000 2D array to store the colour will exceed memory limit.
- You can also happily get 1 point if you forget to output “Possible”.

- Challenge: write the checker for this task.

M2102
Social Distancing and miamia

M2102 Subtask 1

Extract the starting time(S) from 1st line.

Extract the bpm(B) from 2nd line.

Extract the note value(V) from 3rd line.

For the rest of each line, simply count the number of commas.

Answer = S + # of commas * 240 / (B * V)

M2102 Subtask 2

Extract the starting time from 1st line.

Extract the bpm(B) **OR** note value(V) from 2nd line.

Extract the note value(V) **OR** bpm(B) from 3rd line.

For the rest of each line, if it is bpm or note value, extract and update the corresponding value.

Otherwise, count the number of commas and update the answer.

M2102 Subtask 3

Extract the starting time from 1st line.

Extract the bpm(B) **OR** note value(V) from 2nd line.

Extract the note value(V) **OR** bpm(B) from 3rd line.

For the rest of each line, if it is bpm or note value, extract and update the corresponding value.

Otherwise, ~~count the number of commas and update the answer.~~

M2102 Subtask 3

Since we have multiple notes, we can't simply count the number of commas for each line.

- Commas inside "[ ]" don't count
- A multiple note can be splitted into 2 or more lines

M2102 Subtask 3

For each line, if it is not bpm nor note value, we combine the lines together.

When the line is bpm or value, we count the number of commas(excluding commas inside “[ ]”) in the lines we combined so far.

A easier way for computing the answer is to update the answer whenever encountering a new note(rest/single/multiple)

Remember to check one more time if we need to count the number of commas or not, after processing all the lines of the input.

M2102 General Solution

In fact, we can just combine all the lines and process the data as one line.

Time complexity: O(Total # of characters)

M2102 Other notes

In Hong Kong, you can try playing maimai in arcades if you are 16 years old or above ^_^

In some other places(e.g. Japan, Taiwan, Singapore, Malaysia...), you can play maimai even if you are below 16! If you have not been 16 years old yet, give it a try when you travel to those places.

M2103
Social Distancing and Virtual New Year Visit

M2103 Solution

It is trivial that when all the numbers are negative, the answer is -1(impossible).

M2103 Subtask 1

Since R = 0, we can rearrange the numbers without any cost.

Obviously, it is optimal to sort the numbers by descending order.

Compute the sum of the numbers, keep subtracting the smallest number until the sum > 0.

Min cost = # of cancelling visits * C

Time complexity: O(N log N)

M2103 Subtask 2

Since R is too large(1e9) and C = 1, we only need to consider cancelling the visits.

Keep accumulating the numbers. Whenever current sum < 0, we need to cancel some visits(subtract some numbers).

What kind of numbers should we subtract?

M2103 Subtask 2

It's not hard to see that subtracting the smallest number so far is optimal.

We can find the smallest number so far by simply iterating the array.

Remember to add extra cost if final sum = 0, as the problem requires final sum > 0.

Time complexity: $O(N^2)$

M2103 Subtask 3

Instead of simply iterating the array, we can use a heap(**priority_queue** in C++) to maintain and find the smallest number so far.

Time complexity: O(N log N)

M2103 General Solution

From the previous subtasks, it's not difficult to realize that we only need to consider rearranging the visits or not.

If rearranging, the solution is same as that of subtask 1.

If not rearranging, the solution is same as that of subtask 3.

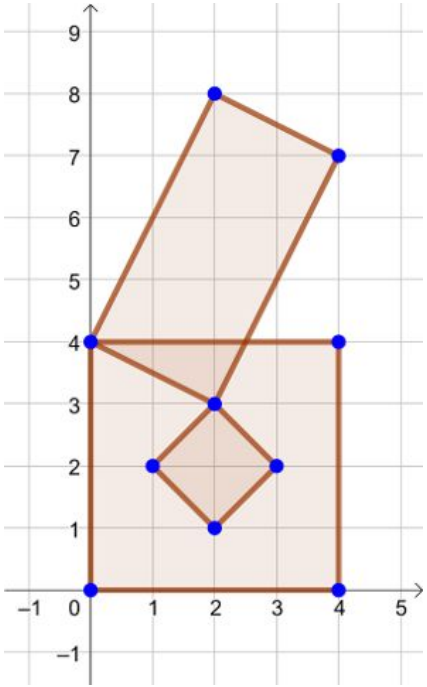
Time complexity: O(N log N)

M2104
Social Distancing and Lockdown Countdown

M2104 Problem

Simplified version: count the number of rectangles formed by 4 given points

Note: tilted rectangles also count



M2104 Subtask 1

N is pretty small (≤30), exhaust all possible combinations of 4 points in $O(N^4)$

To check if it forms a rectangle, we need to check the lengths and angles:

- https://assets.hkoi.org/training2019/geom.pdf (length: slide 7, angle: slide 20, precision: slide 52)

Remember to remove duplicated counts (consider how many combinations we considered for one rectangle)

- If we exhaust 4 indices (i, j, k, l) and:
  - only consider (i, j, k), (j, k, l), (k, l, i), (l, i, j) form right angles, the duplicate factor is 8 (or 4 if it's only clockwise right angle)

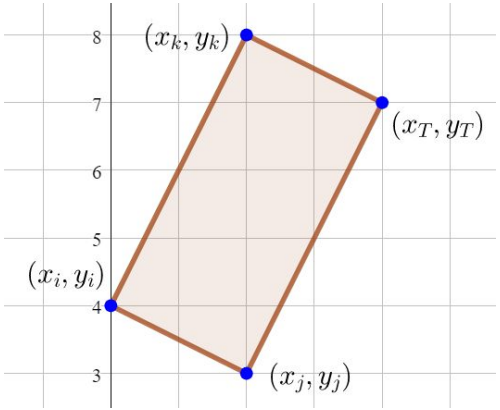
Expected score: 5/20

M2104 Subtask 1.5

Exhaust first 3 points in $O(N^3)$, and check if the remaining points exist

Suppose we have exhausted (i, j, k):

- First we have to confirm the points j, i, k forms a right angle
- If so, the target point coordinates can be calculated by:
  - $x_T = x_k - x_i + x_j$ (similar for y)
- And we can check if (x_T, y_T) exists if:
  - we first insert the N given points into a std::set (or similar)
  - Check in O(lgN) (or O(1) if some other data structures)



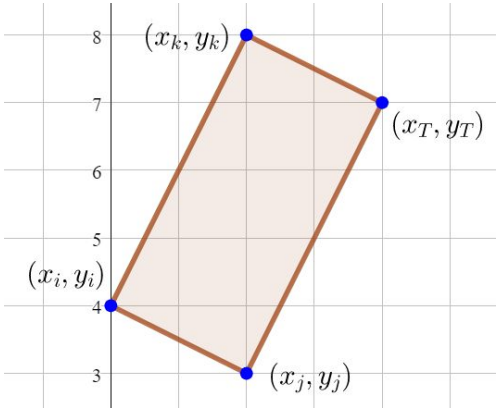
Expected score: 8/20

M2104 Subtask 2-3

From the previous solution, we know that when considering point i, only pairs (j, k) that form a right angle by point i is meaningful

For every point i exhausted, sort the N-1 points by its angle from point i

- Maintain two pointers to keep angle between points = 90 degrees
- Exhaust all pairs of points (j, k) along these two lines
  - Check if the target point T exist (same as previous slide)



Expected score: 12/20 ~ 20/20

The maximum number of triples that form a right angle is in fact $O(N^2 \lg N)$:
[János Pach, Micha Sharir, Repeated angles in the plane and related problems]

M2104 Subtask 2-3

From subtask 2 to 3, we're expecting $O(N^2 \lg N)$ or $O(N^2)$ solutions, the actual score you get highly depends on optimizations of your implementation :)

In the following slides,

we will discuss a different solution that could be implemented easier, also easier to pass the tight TL.

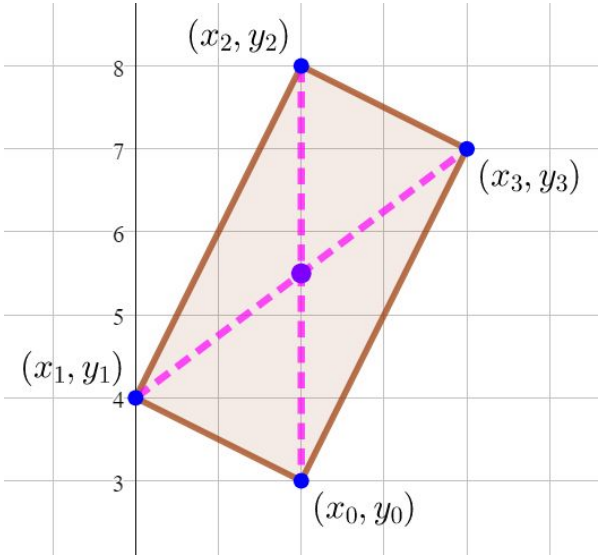
M2104 Subtask 2-3

The key observation is that, to form a rectangle, the 4 points must come from 2 pairs of points that:

- Share the same midpoint
- Have the same length

e.g. 1st pair: (x_0, y_0), (x_2, y_2); 2nd pair (x_1, y_1), (x_3, y_3)

(also if the above conditions are satisfied, they must form a rectangle)



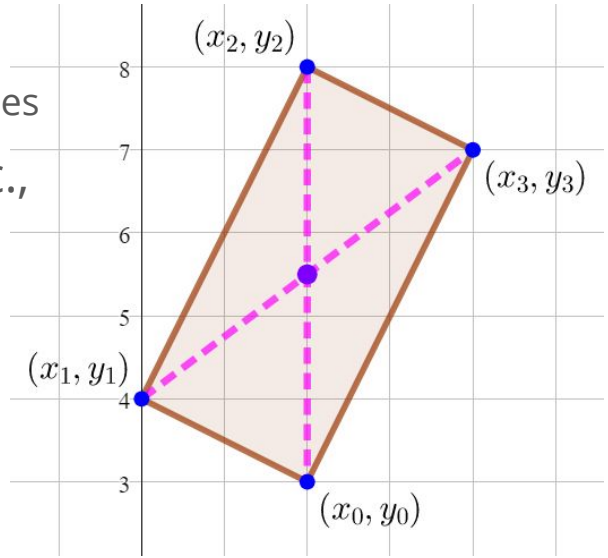
M2104 Subtask 2-3

So the idea of full solution is:

- exhaust all pairs of points
- Record their (midpoint, distance) e.g. ((2, 5.5), 5)
- Count the number of such (midpoint, distance)
  - If the count is C, there are C*(C-1)/2 ways to form rectangles

We can record them with std::map / std::multiset / etc., or we can store them in an array and sort to count

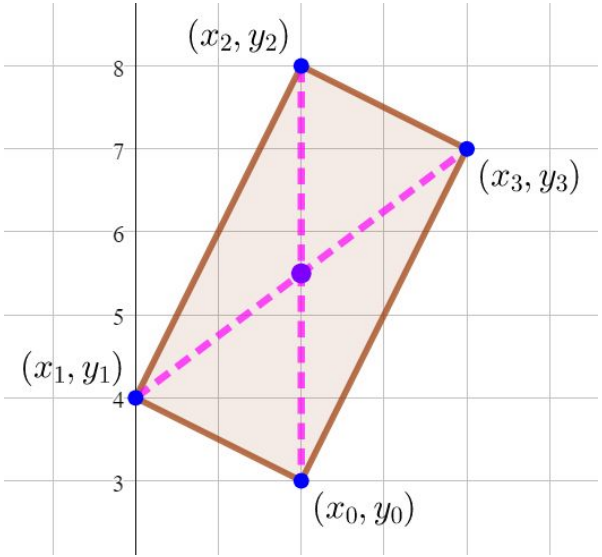
Expected score: 14/20 ~ 20/20



M2104 Other notes

To avoid precision errors, the best practice is to store everything in integers:

- Instead of storing midpoints, store midpoint*2, i.e. sum of two points
- Do not take sqrt of the distance



M2105
Social Distancing and Compulsory Testing

M2105 Problem

Given an array of N integers and an integer K, find the K-th largest average subarray.

```
3 3
1 2 3

2.00000000
```

For all cases:

$1 \leq N \leq 2 \times 10^5$

$1 \leq P_i \leq 10^9$

$1 \leq K \leq \frac{N(N+1)}{2}$

	Points	Constraints
1	30	$N \leq 5000, K \leq 10$
2	30	$K \leq 10$
3	40	No additional constraints

M2105 Subtask 1

N <= 5000

Naively calculate the average of all subarray, sort by descending order and output the K-th largest.

Time complexity:

$O(N^2 \log(N^2))$ with partial sum

$O(N^3)$ without partial sum (You can still get 20% of overall points)

M2105 Subtask 2

K <= 10

It is impossible to calculate the average one by one.

M2105 Subtask 2

In fact we only have to consider the average of subarray [i, i+K-1] that is all subarray of size K.

Suppose the K-th largest average subarray have average value X has size K+1 and is at [i, i+K].

For i <= j < i+K, let the average value of [i, j] be avg0 and [j+1, i+K] be avg1.

X * (K + 1) = avg0 * (j - i + 1) + avg1 * (i + K - j)

Either avg0 >= X or avg1 >= X, therefore for all j, there exist at least one subarray with average value >= X, and there are K of them, which contradicts the statement that the K-th largest average subarray have average value X.

M2105 Subtask 2

In fact we only have to consider the average of subarray [i, i+K-1] that is all subarray of size K.

Time complexity: O(NKlog(K))

M2105 Subtask 2

Alternative solution:

Given average X, can we calculate if there exist at least K subarrays with average value larger than X?

Let psum[i] = $P_1+P_2+...+P_i$

Consider the subarray [i, j], its sum is psum[j] - psum[i-1].

We want (psum[j] - psum[i-1]) / (j - i + 1) >= **X**

Which is equivalent to psum[j] - psum[i-1] - **X** * (j - i + 1) >= 0

M2105 Subtask 2

Therefore we can modify the prefix sum array to be calculated by

psum[i] = psum[i - 1] + P[i] - X

From this point on any sum we talk about are calculated by the modified prefix sum above.

If we maintain the K-th smallest prefix sum and the K-th largest subarray sum for all i.

We can determine if there exist at least K subarrays with average >= X

M2105 Subtask 2

If we maintain the K-th smallest prefix sum and the K-th largest subarray sum for all i.

The K-th smallest prefix sum can be maintained by using a max heap of size K (kmin).

The K-th largest subarray sum can be maintained by using a min heap of size K (kmax).

```
priority_queue<double, vector<double>, greater<double>> kmax; // min heap
priority_queue<double> kmin;                                  // max heap
kmin.push(0);
for (int i = 1; i <= n; i++) {
  priority_queue<double> tmp = kmin;
  while (!tmp.empty()) {
    if (kmax.size() < k)
      kmax.push(psum[i] - tmp.top());
    else if (psum[i] - tmp.top() >= kmax.top())
      kmax.push(psum[i] - tmp.top()), kmax.pop();
    tmp.pop();
  }
  if (kmin.size() < k)
    kmin.push(psum[i]);
  else if (psum[i] <= kmin.top())
    kmin.push(psum[i]), kmin.pop();
}
```

M2105 Subtask 2

Since we can determine if there exist at least K subarray with average X, we can do binary search on answer to find X.

Time complexity: O(NKlog(K)log(1/eps))

M2105 Subtask 3

No additional constraints.

The previous solution will TLE since K can get very large.

M2105 Subtask 3

Instead of maintaining the K-th smallest prefix sum and largest subarray sum, can we calculate no. of i for all j such that psum[j] - psum[i-1] >= 0?

This can be modified to no. of pair (i,j) such that j > i and psum[j] >= psum[i-1], which is similar to **no. of inversion**.

We can directly count this number or calculate by n*(n+1)/2 - no. of inversion.

M2105 Subtask 3

Counting number of inversion for a size N array can be solved by modified Merge Sort or Binary Index Tree or Segment Tree.

Time complexity: O(Nlog(N)log(1/eps))

Note that because of large constant, pbds will get TLE even it have the same time complexity.