Day 5: Adam Gasienica-Samek Contest 1 Tutorial

Adam_GS

1st OCPC, Winter 2023

February 22, 2023

C - Convolution

Given two sequences $a_0, a_1, \ldots, a_{n-1}$ and $a_0, a_1, \ldots, a_{n-1}$, define a sequence $c_0, c_1, \ldots, c_{n+m-2}$ as

$$c_k = \sum_{i+j=k} a_i b_j$$

Find the sum of c.
$n, m \leq 10^5$.

C - Convolution

Consider the polynomials:

- $A(x) = a_0 x^0 + a_1 x^1 + \cdots + a_{n-1} x^{n-1}$
- $B(x) = b_0 x^0 + b_1 x^1 + \cdots + b_{m-1} x^{m-1}$
- $C(x) = c_0 x^0 + c_1 x^1 + \cdots + c_{n+m-2} x^{n+m-2}$

C - Convolution

Consider the polynomials:

- $A(x) = a_0x^0 + a_1x^1 + \cdots + a_{n-1}x^{n-1}$
- $B(x) = b_0x^0 + b_1x^1 + \cdots + b_{m-1}x^{m-1}$
- $C(x) = c_0x^0 + c_1x^1 + \cdots + c_{n+m-2}x^{n+m-2}$

Then $C(x) = A(x)B(x) \Rightarrow$ You can solve this with FFT.

C - Convolution

Easier solution:

$$\sum_{i=0}^{n+m-2} c_i = \left(\sum_{i=0}^{n-1} a_i\right)\left(\sum_{i=0}^{m-1} b_i\right)$$

Proof: Expand it, or use the fact that $C(1) = A(1)B(1)$.

G - Guessing by Divisibility

There is a secret positive integer n. You can ask questions “Is n divisible by x?”. Find n using 1500 questions.
$n \leq 10^4$.

G - Guessing by Divisibility

For each prime number p, we can find the largest a such that p^a divides n. The largest possible a is $\left\lfloor \log_p 10^4 \right\rfloor$.

G - Guessing by Divisibility

For each prime number p, we can find the largest a such that p^a divides n. The largest possible a is $\left\lfloor \log_p 10^4 \right\rfloor$.
There are 1229 primes numbers below 10000. The sum of $\left\lfloor \log_p 10^4 \right\rfloor$ over these primes is 1280.

L - Lying Faces

There are n people. Each person either always tells the truth or always lie.
There are more truth tellers than liars.
You can ask person a the question “How long is b's nose?”. Whenever a person lie, their nose length increases. Determine all the liars with 10001 questions.
$n \leq 10^4$.

L - Lying Faces

Let's ask two people x and y about z in two consecutive queries.
If they reply with the same number, then they are either both liars or both truth tellers.

L - Lying Faces

Ask people $2, 3, \ldots, n$ about person 1. There are two cases:

- If $n = 2k$, then there are at least $k + 1$ truth tellers. A strict majority of $2, 3, \ldots, n$ must give the same answer.
- If $n = 2k + 1$, then there are also at least $k + 1$ truth tellers. At least half of $2, 3, \ldots, n$ must give the same answer.

L - Lying Faces

Case 1: A majority of $2, 3, \ldots, n$ answer the same.
Then the majority are all truth tellers, and the rest are liars. To check person 1, ask 1 about anyone and then ask any truth teller about 1 to check if the nose of 1 changed.

L - Lying Faces

Case 2: No majority of $2, 3, \ldots, n$ answer the same. Then exactly half of them answer the same.
If the other half doesn't answer the same, this is the same as case 1.

L - Lying Faces

Case 2: No majority of $2, 3, \ldots, n$ answer the same. Then exactly half of them answer the same.
If the other half doesn't answer the same, this is the same as case 1.
Otherwise, one half is truth tellers and the other is liars. Then person 1 must be a truth teller. We ask both 1 and 2 about 3 to check if 2 is a liar.

L - Lying Faces

Both cases require $n + 1$ questions.
The limit is set to 10001 so that an alternative solution which requires $n + 2$ questions when n is odd can pass.

D - DAG Probability

A tournament with n vertices is generated by randomly selecting each edge: from the smaller vertex to the larger with probability $\frac{a}{b}$ and from larger to smaller otherwise. Find the probability that the generated graph is a DAG.
$n \le 10^6$.

D - DAG Probability

There are exactly $n!$ tournaments that are DAGs, and each of them corresponds to a permutation of $1, 2, \ldots, n$.

D - DAG Probability

There are exactly $n!$ tournaments that are DAGs, and each of them corresponds to a permutation of $1, 2, \ldots, n$.
Let $P_1, P_2, \ldots, P_n$ be a permutation of $1, 2, \ldots, n$. The corresponding tournament graph has an edge $P_i \to P_j$ for every $1 \leq i < j \leq n$.

D - DAG Probability

There are exactly $n!$ tournaments that are DAGs, and each of them corresponds to a permutation of $1, 2, \ldots, n$.
Let $P_1, P_2, \ldots, P_n$ be a permutation of $1, 2, \ldots, n$. The corresponding tournament graph has an edge $P_i \to P_j$ for every $1 \leq i < j \leq n$.
Let k denote the number of inversions in P and $p = \frac{a}{b}$. The probability of P being the topological ordering of the graph is:

$$p^{m-k}(1-p)^k$$

where m is the number of edges $\frac{n(n-1)}{2}$.

D - DAG Probability

We want to find the sum over all permutations. Let $c(n, k)$ denote the number of permutations with n elements and k inversions.
Then the answer to the problem is:

$$\sum_{k=0}^{m} c(n, k) p^{m-k} (1-p)^k$$

D - DAG Probability

Let's find a recurrence for $c(n, k)$.

- $c(1, 0) = 1$
- $c(n, k) = \sum_{i=0}^{n-1} c(n-1, k-i)$

D - DAG Probability

Let's find a recurrence for $c(n, k)$.

- $c(1, 0) = 1$
- $c(n, k) = \sum_{i=0}^{n-1} c(n-1, k-i)$

Doing this directly takes $O(n^3)$ time.

D - DAG Probability

Consider the generating function

$$f_n(x) = \sum_{i=0}^{m} c(n,i)x^i.$$

D - DAG Probability

Consider the generating function

$$f_n(x) = \sum_{i=0}^{m} c(n, i) x^i.$$

Then

$$f_n(x) = f_{n-1}(x) \cdot (1 + x + x^2 + \cdots + x^{n-1}).$$

D - DAG Probability

Consider the generating function

$$f_n(x) = \sum_{i=0}^{m} c(n,i)x^i.$$

Then

$$f_n(x) = f_{n-1}(x) \cdot (1 + x + x^2 + \cdots + x^{n-1}).$$

Therefore $f_n(x) = 1(1+x)(1+x+x^2)\cdots(1+x+\cdots+x^{n-1})$.

D - DAG Probability

We can rewrite the answer to

$$\sum_{i=0}^{m} c(n,i)p^{m-i}(1-p)^{i} = p^{m}\sum_{i=0}^{m} c(n,i)\left(\frac{1-p}{p}\right)^{i}$$

D - DAG Probability

We can rewrite the answer to

$$\sum_{i=0}^{m} c(n,i)p^{m-i}(1-p)^i = p^m \sum_{i=0}^{m} c(n,i) \left(\frac{1-p}{p}\right)^i$$

Notice that the right side is just evaluating the polynomial f_n at $\frac{1-p}{p}$.
We may calculate it in $O(n)$.

H - Highest Median Walk

Given a directed, weighted graph, find a walk with length at least k that has the highest possible median weight.
$n \leq 10^5$, $m \leq 2 \times 10^5$, $k \leq 50$.

H - Highest Median Walk

Binary search the answer by checking if the median can be at least x.

H - Highest Median Walk

Binary search the answer by checking if the median can be at least x.
Assign each edge value 1 or -1 by comparing it with x.

H - Highest Median Walk

Binary search the answer by checking if the median can be at least x.
Assign each edge value 1 or -1 by comparing it with x.
Is there a path with length at least k that has positive sum?

H - Highest Median Walk

Claim: We only have to check walks with length at most $2k$.

H - Highest Median Walk

Claim: We only have to check walks with length at most $2k$.
Proof: Suppose there is a walk with length $> 2k$ and has positive sum. If we split the walk in half, each part has length at least k and at least one of them has positive sum.

H - Highest Median Walk

Let $dp[x][l]$ be the maximal sum of values of a walk with length l ending at vertex x.

H - Highest Median Walk

Let $dp[x][l]$ be the maximal sum of values of a walk with length l ending at vertex x.
For $l = 1, 2, \ldots, 2k$ and each edge $u \to v$, we can update $dp[v][l]$ with $dp[u][l-1]$.
Time complexity: $O((n+m)k \log C)$ where C is the maximum weight.

E - Erase the Primes

There is an array $a_1, a_2, \ldots, a_n$ and there are q queries. For each query, you have to find the cost of the subarray $a_l, a_{l+1}, \ldots, a_r$.
The cost of an array is the minimum number of the operations needed to make every number divisible by two different primes ("good"), or -1 if it's impossible.
An operation is replacing two numbers with their product.
$n, q \leq 2 \times 10^5$, $a_i \leq 10^7$.

E - Erase the Primes

If every number is a power of the same prime number, then the cost is -1.

E - Erase the Primes

If every number is a power of the same prime number, then the cost is -1.
Otherwise, we can multiply them all together so the cost is positive.

E - Erase the Primes

Every number is one of:

- 1, which always contributes 1 to the answer.
- a good number, which always contributes 0 to the answer.
- a prime power.

E - Erase the Primes

Every number is one of:

- 1, which always contributes 1 to the answer.
- a good number, which always contributes 0 to the answer.
- a prime power.

We only have to consider prime powers.

E - Erase the Primes

In one operation, we can remove 1 or 2 prime powers. If two numbers are powers of different primes, we can remove them together with one operation. The strategy is to do operations on as many pairs of different primes as possible.

E - Erase the Primes

Let k be the number of prime powers in the range, and $p = [p_1, p_2, \ldots, p_k]$ be the primes.
Claim: If no prime has majority in p, then we need $\left\lceil \frac{k}{2} \right\rceil$ operations.

E - Erase the Primes

Let k be the number of prime powers in the range, and $p = [p_1, p_2, \ldots, p_k]$ be the primes.
Claim: If no prime has majority in p, then we need $\left\lceil \frac{k}{2} \right\rceil$ operations.
We can form $\left\lfloor \frac{k}{2} \right\rfloor$ pairs of different primes by greedily picking the two primes with largest frequency in each operation.

E - Erase the Primes

Claim: If a prime has majority and it appears t times, then we need t operations.

E - Erase the Primes

Claim: If a prime has majority and it appears t times, then we need t operations. We can form at most $k - t$ pairs of different primes. The remaining $2t - k$ majority elements have to be removed one by one.

E - Erase the Primes

The problem reduces to finding the frequency of the majority element in a range, which is a standard problem and can be done in $O(\log n)$, $O(\log^2 n)$ or $O(\sqrt{n})$ time per query.

J - Joining Arrays

Define the permutation scaling $P(B)$ of $[B_1, B_2, \ldots, B_m]$ as the permutation of $[1, 2, 3, \ldots, m]$ such that $P(B)_i < P(B)_j$ iff $B_i \leq B_j$.
Given n sets $S_1, S_2, \ldots, S_n$, we make a sequence $[A_1, A_2, \ldots, A_n]$ such that $A_i \in S_i$. Count the number of ways to choose A such that $P(A)$ is lexicographically largest.
$n, \sum |S_i| \leq 5 \times 10^5$

J - Joining Arrays

Let's find the lexicographically largest permutation scaling.

1. Choose the biggest element from the first set.
2. For each element we want to maximize the number of previous elements that are bigger.
3. If there are multiple values that maximize that, we take the biggest one.

J - Joining Arrays

After knowing the maximal permutation scaling, we can reorder the sets in the order of the permutation.
Now the problem becomes counting the number of ways to choose A such that it is increasing. This can be done with DP using two pointers.

I - Incomplete Information Queries

There is an unknown tournament graph with n vertices. The i-th vertex has c_i reachable vertices.
There are m bidirectional edges that you can add to the graph. Answer q queries: What is the minimum number of edge additions required to travel from a to b in the best/worst case?
$n, m, q \leq 5 \times 10^5$.

I - Incomplete Information Queries

Contract all the SCCs and sort them in topological order. Notice that the result is uniquely determined by the sequence c_i.
Suppose we can add an edge between the u-th SCC and the v-th SCC. This operation makes every SCC between u and v strongly connected.

I - Incomplete Information Queries

Contract all the SCCs and sort them in topological order. Notice that the result is uniquely determined by the sequence c_i.
Suppose we can add an edge between the u-th SCC and the v-th SCC. This operation makes every SCC between u and v strongly connected.
The queries now become: Given a segment $[l, r]$, find the minimum number of segments from a given set needed to cover $[l, r]$.
The answer to each query can be found in $O(\log n)$ time using binary lifting.

F - Forest Game

Consider a forest with n vertices. k special vertices are secretly chosen, and you are allowed to start a walk from vertex x with length m. Find the maximum probability that you visit a special vertex on the walk if you choose it optimally. You should process q queries online. Each query is either adding an edge to the forest or asking the answer for some x, k and m.
$n \leq 2 \times 10^5$, $q \leq 5 \times 10^5$.

F - Forest Game

The optimal strategy is choosing a walk that visits as many different vertices as possible. Suppose a different vertices are visited. Then the answer is

$$1 - \frac{\binom{n-a}{k}}{\binom{n}{k}}.$$

F - Forest Game

The optimal strategy is choosing a walk that visits as many different vertices as possible. Suppose a different vertices are visited. Then the answer is

$$1 - \frac{\binom{n-a}{k}}{\binom{n}{k}}.$$

The queries become “How many different vertices can you visit with a walk of length m starting from x?”.

F - Forest Game

Observation: The vertices visited must form a tree.

F - Forest Game

Observation: The vertices visited must form a tree.
Suppose we visit all vertices of a tree starting at vertex x and ending at y. The minimum number of edges required is equal to $2 \times$ #edges in the tree $-$ the distance between x and y.

F - Forest Game

Observation: The vertices visited must form a tree.
Suppose we visit all vertices of a tree starting at vertex x and ending at y. The minimum number of edges required is equal to $2 \times$ #edges in the tree $-$ the distance between x and y.
The problem reduces to finding the longest path starting from x.

F - Forest Game

Lemma: The farthest vertex from any vertex in a tree is the endpoint of a diameter.

F - Forest Game

Lemma: The farthest vertex from any vertex in a tree is the endpoint of a diameter.

We want to create a data structure on a forest that supports:

- Adding an edge between two different trees.
- Maintaining a diameter of each tree.
- Querying the distance between two vertices.

F - Forest Game

Instead of merging two trees, we support adding a leaf instead.

F - Forest Game

Instead of merging two trees, we support adding a leaf instead.
Maintain the LCA binary lifting array and depth for each vertex.

F - Forest Game

Finding distance between two vertices:

- Find a' and b' such that a' is an ancestor of a, b' is an ancestor of b and $\mathrm{depth}(a') = \mathrm{depth}(b') = \max(\mathrm{depth}(a), \mathrm{depth}(b))$
- Jump with both a' and b' at the same time to see if they reach the same ancestor at the same depth.
- Return $\mathrm{depth}(a) + \mathrm{depth}(b) - 2\,\mathrm{depth}(\mathrm{lca}(a, b))$.

F - Forest Game

Finding distance between two vertices:

- Find a' and b' such that a' is an ancestor of a, b' is an ancestor of b and $\operatorname{depth}(a') = \operatorname{depth}(b') = \max(\operatorname{depth}(a), \operatorname{depth}(b))$
- Jump with both a' and b' at the same time to see if they reach the same ancestor at the same depth.
- Return $\operatorname{depth}(a) + \operatorname{depth}(b) - 2\operatorname{depth}(\operatorname{lca}(a, b))$.

Updating the depths and binary lifting array take $O(1)$ and $O(\log n)$ time.

F - Forest Game

Maintaining diameters:
Let s and t be current diameter endpoints and v be the added leaf. The diameter endpoints after the addition will be one of (s, t), (s, v) or (v, t). We can find the correct one by using the distance query.

F - Forest Game

We can use leaf addition to merge trees by adding vertices from the smaller tree to the larger tree one by one in DFS order.
The final time complexity will be $O(n \log^2 n + q \log n)$.

B - Beautiful XOR Problem

There are n numbers $a_1, a_2, \ldots, a_n$. Count the number of ways to choose a subsequence that has length divisible by k and has bitwise XOR 0.
$n \leq 10^6$, $k \leq 20$, $0 \leq a_i \leq 10^6$.

B - Beautiful XOR Problem

Let's solve for $k = 1$.
The XOR convolution $a \star b$ of sequences a and b is defined as

$$c_k = \sum_{i \oplus j = k} a_i \cdot b_j.$$

The Walsh-Hadamard transform (WHT) of a sequence a is defined as

$$\mathrm{WHT}(a)_i = \sum_{j=0}^{n} (-1)^{\mathrm{popcnt}(i \,\mathrm{AND}\, j)} \cdot a_j.$$

B - Beautiful XOR Problem

Let's solve for $k = 1$.
The XOR convolution $a \star b$ of sequences a and b is defined as

$$c_k = \sum_{i \oplus j = k} a_i \cdot b_j.$$

The Walsh-Hadamard transform (WHT) of a sequence a is defined as

$$\mathrm{WHT}(a)_i = \sum_{j=0}^{n} (-1)^{\mathrm{popcnt}(i \,\mathrm{AND}\, j)} \cdot a_j.$$

Let A_i be a sequence $A_{i,0}, A_{i,1}, \ldots, A_{i,2^{20}-1}$ s.t. $A_{i,0} = 1$, $A_{i,a_i} = 1$ and $A_{i,j} = 0$ for every other j. The solution when $k = 1$ is the 0-th element of

$$A_1 \star A_2 \star \cdots \star A_n.$$

B - Beautiful XOR Problem

With WHT, the product is equal to

$$\mathrm{WHT}^{-1}(\mathrm{WHT}(A_1) \cdot \mathrm{WHT}(A_2) \cdot \ldots \cdot \mathrm{WHT}(A_n)).$$

Since $\mathrm{popcnt}(0 \,\mathrm{AND}\, j) = 0$, each element of $\mathrm{WHT}(A_i)$ is either 0 or 2.

B - Beautiful XOR Problem

With WHT, the product is equal to

$$\mathrm{WHT}^{-1}(\mathrm{WHT}(A_1) \cdot \mathrm{WHT}(A_2) \cdot \ldots \cdot \mathrm{WHT}(A_n)).$$

Since $\mathrm{popcnt}(0 \,\mathrm{AND}\, j) = 0$, each element of $\mathrm{WHT}(A_i)$ is either 0 or 2.
Each element of $\mathrm{WHT}(A_1) \cdot \mathrm{WHT}(A_2) \cdot \ldots \cdot \mathrm{WHT}(A_n)$ is either 2^n or 0.
We can check which one it is by doing SOS DP.

B - Beautiful XOR Problem

For a different k, we treat each $A_{i,j}$ as a polynomial in x modulo $x^k - 1$.
Then $A_{i,0} = 1$ and $A_{i,a_i} = x$.

B - Beautiful XOR Problem

For a different k, we treat each $A_{i,j}$ as a polynomial in x modulo $x^k - 1$.
Then $A_{i,0} = 1$ and $A_{i,a_i} = x$.
Each element of $\mathrm{WHT}(A_1) \cdot \mathrm{WHT}(A_2) \cdot \ldots \cdot \mathrm{WHT}(A_n)$ is equal to

$$(1 + x)^m \cdot (1 - x)^{n-m}.$$

We can calculate m using SOS.

B - Beautiful XOR Problem

For a different k, we treat each $A_{i,j}$ as a polynomial in x modulo $x^k - 1$.
Then $A_{i,0} = 1$ and $A_{i,a_i} = x$.
Each element of $\mathrm{WHT}(A_1) \cdot \mathrm{WHT}(A_2) \cdot \ldots \cdot \mathrm{WHT}(A_n)$ is equal to

$$(1+x)^m \cdot (1-x)^{n-m}.$$

We can calculate m using SOS.
We can precompute $(1+x)^m$ and $(1-x)^m$ for all m in $O(nk)$ time.
For each value $(1+x)^m \cdot (1-x)^{n-m}$ we only care about the coefficient of x^0, so we can multiply two polynomials in $O(k)$.
The total time complexity is $O(nk)$.

A - Assembly

Solve the following problem in an assembly-like language with only 4 32-bit registers:
Given n integers $a_1, a_2, \ldots, a_n$, find the product of the k largest primes in it modulo 2023.
$n \leq 100$, $a_i \leq 500$, $k \leq 4$

A - Assembly

We can store three numbers in one register in base 1000, i.e., we store (x, y, z) as $x + y \times 10^3 + z \times 10^6$.
Let $a \geq b \geq c \geq d$ be the four currently biggest prime numbers in the sequence.
Initially each of them is equal to 1.
The initial registers setup will look like this:

- A: (a, b, n)
- B: $(c, d, 0)$
- C: $(0, 0, 0)$
- D: $(0, 0, 0)$

A - Assembly

Our algorithm looks like this:

1. Repeat n times:
2. Read a number and check if it is prime.
3. If it is prime, update the list of biggest primes.
4. Find their product modulo 2023.

A - Assembly

If A stores (a, b, n), we can make a loop that runs n times.

- DECREASE A BY 10^6
- IF $A \geq 10^6$ GOTO loop ELSE GOTO break

A - Assembly

We can make a prime checking procedure.

1. Read input to C.
2. Set D to 2.
3. While $D < C$:
4. If D divides C, exit the procedure.
5. Increase D by 1.
6. Continue to the next step where C is prime.

A - Assembly

To check whether D divides C:

1. Set a new variable e to 0.
2. While $C > D$:
3.   Decrease C by D.
4.   Increase e by 1.
5. D divides C iff $C = D$.

We can store e in the most significant digit of B.

A - Assembly

To check whether D divides C:

1. Set a new variable e to 0.
2. While $C > D$:
3. Decrease C by D.
4. Increase e by 1.
5. D divides C iff $C = D$.

We can store e in the most significant digit of B.
After the procedure, we need to restore C:

1. While $e > 0$:
2. Increase C by D.
3. Decrease e by 1.

A - Assembly

Suppose we found a prime number p and want to update the values of a, b, c, d.

- If $p \geq d$, swap p, d
- If $d \geq c$, swap d, c
- If $c \geq b$, swap c, b
- If $b \geq a$, swap b, a

A - Assembly

We can move a value d from B to D:

1. DECREASE B BY 10^3
2. INCREASE D BY 1
3. IF $B \geq 10^3$ GOTO 1 ELSE GOTO 4
4. ...

A - Assembly

We can move a value d from B to D:

1. DECREASE B BY 10^3
2. INCREASE D BY 1
3. IF $B \geq 10^3$ GOTO 1 ELSE GOTO 4
4. $\ldots$

The result becomes

- $A\colon (a, b, n)$
- $B\colon (c, 0, 0)$
- $C\colon (0, 0, 0)$
- $D\colon (d, 0, 0)$

A - Assembly

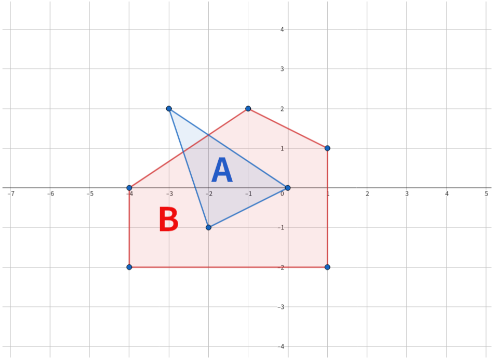
The most significant digits of other registers can be moved similarly — they behave like stacks.
We may use C and D as temporary values to swap variables.

K - Kid's First Geometry Problem

There is a convex polygon A and k convex polygons $B_1, B_2, \ldots, B_k$. Initially A intersects every of B_i.
Find the minimum value of $|dx| + |dy|$ such that after you move A by (dx, dy), A intersects none of B_i.
Total #vertices of $B_i \leq 75000$, #vertices of $A \leq 75000$, $k \leq 25000$.

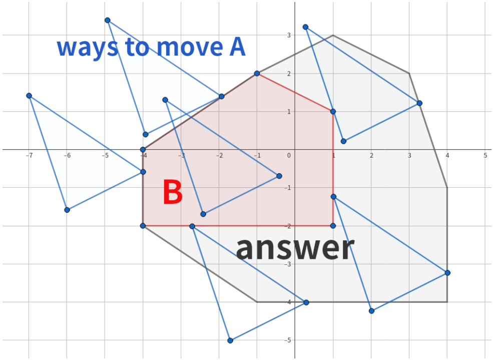
K - Kid's First Geometry Problem

Consider $k = 1$.
A and B are convex polygons. What is the set of vectors (dx, dy) such that $A + (dx, dy)$ intersects B?



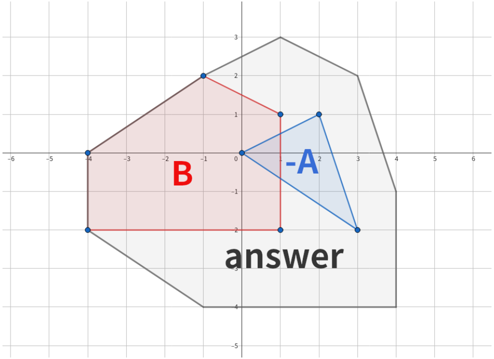
K - Kid's First Geometry Problem

Intuitively, it should look like this:



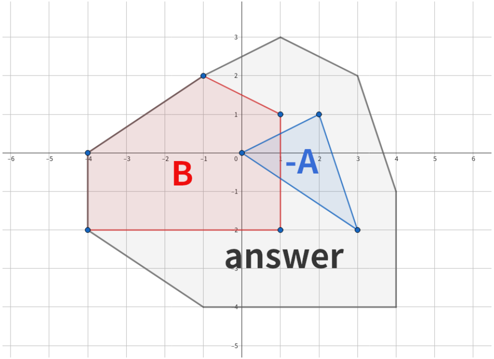
K - Kid's First Geometry Problem

Theorem: It is the Minkowski sum of $-A$ and B.



K - Kid's First Geometry Problem

Theorem: It is the Minkowski sum of $-A$ and B.



The sum is $\{b - a \mid a \in A, b \in B\}$. A vector $d = (dx, dy)$ is in the set if and only if $a + d = b$ for some points $a \in A$ and $b \in B$.

K - Kid's First Geometry Problem

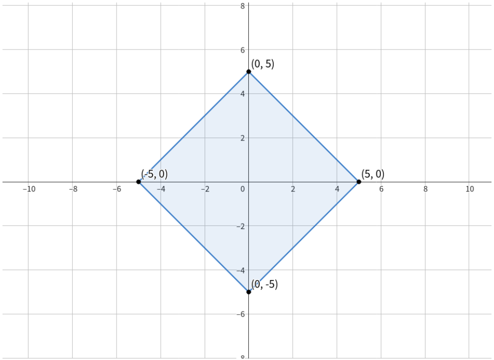
Rewrite the problem:
There is a convex polygon $-A$ and k convex polygons $B_1, B_2, \ldots, B_k$. Find the minimum value of $|x| + |y|$ of a point (x, y) that is not inside the Minkowski sum of A and any B_i. The point $(0, 0)$ is contained in all these sums.

K - Kid's First Geometry Problem

Try binary search: For a value c, we want to check whether the region $|x| + |y| \leq c$ contains a point outside all the sums.

K - Kid's First Geometry Problem

Try binary search: For a value c, we want to check whether the region $|x| + |y| \leq c$ contains a point outside all the sums.
Fact: The region $|x| + |y| \leq c$ is a rotated square centered at $(0, 0)$.



K - Kid's First Geometry Problem

Claim: We only have to check the boundary of the square.

K - Kid's First Geometry Problem

Claim: We only have to check the boundary of the square.
Reason: If a point is inside one of the sums, every point between it and $(0, 0)$ is also inside that sum, since the sum is a convex set and $(0, 0)$ belongs to it.

K - Kid's First Geometry Problem

Claim: We only have to check the boundary of the square.
Reason: If a point is inside one of the sums, every point between it and $(0, 0)$ is also inside that sum, since the sum is a convex set and $(0, 0)$ belongs to it.
Therefore, if the boundary doesn't contain a point outside the sums, then neither does its interior.

K - Kid's First Geometry Problem

Let's make life easier:

- Check one edge at a time (e.g., assuming $dx, dy \geq 0$).
- Rotate the problem by 45 degrees, so the squares become axis-aligned.

K - Kid's First Geometry Problem

Let's make life easier:

- Check one edge at a time (e.g., assuming $dx, dy \geq 0$).
- Rotate the problem by 45 degrees, so the squares become axis-aligned.

Now we're interested about an edge $(-c, -c) - (c, -c)$ of the square.

K - Kid's First Geometry Problem

Suppose we want to check whether the segment $(-c, -c) - (c, -c)$ satisfies the constraint.

K - Kid's First Geometry Problem

Suppose we want to check whether the segment $(-c, -c) - (c, -c)$ satisfies the constraint.
Each Minkowski sum is a convex polygon, which covers a contiguous interval on the segment. It is sufficient to find the intersections of each sum with the line $y = -c$ and check whether the entire segment is covered.

K - Kid's First Geometry Problem

Recall how the Minkowski sum of two convex polygons is calculated:

- Find the lowest point of each polygon. Their sum is the lowest point of the result.
- Merge the lists of edge vectors of both polygons sorted by angle. These are the edges of the result.

K - Kid's First Geometry Problem

Recall how the Minkowski sum of two convex polygons is calculated:

- Find the lowest point of each polygon. Their sum is the lowest point of the result.
- Merge the lists of edge vectors of both polygons sorted by angle. These are the edges of the result.

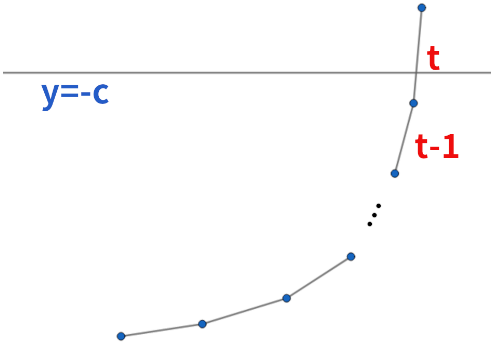
Doing this naively for every B_i takes $O(n + m_1) + O(n + m_2) + \cdots + O(n + m_k) = O(kn + \sum m_i)$ time which is too slow.

K - Kid's First Geometry Problem

Observation: To find an intersection, we don't need the entire Minkowski sum.
We only need the edge that intersects $y = -c$.

K - Kid's First Geometry Problem

Observation: To find an intersection, we don't need the entire Minkowski sum.
We only need the edge that intersects $y = -c$.
Sort the edges by angle. We want the minimum t such that the sum of the first t edges have $y \geq -c$.



K - Kid's First Geometry Problem

The edges of the i-th Minkowski sum contain the edges of A and B_i.
We have two sorted list of edges, and we want to find a target prefix sum of their union.

K - Kid's First Geometry Problem

The edges of the i-th Minkowski sum contain the edges of A and B_i.
We have two sorted list of edges, and we want to find a target prefix sum of their union.
We could use binary search again to find the slope of the target edge, but doing this directly uses two additional $\log$ factors.

K - Kid's First Geometry Problem

To optimize, let's use a data structure that supports:

- Maintaining a list of vectors sorted by angle.
- Adding and deleting vectors.
- Querying the smallest prefix of vectors with total $y \geq$ some value.

K - Kid's First Geometry Problem

To optimize, let's use a data structure that supports:

- Maintaining a list of vectors sorted by angle.
- Adding and deleting vectors.
- Querying the smallest prefix of vectors with total $y \geq$ some value.

A segment tree or Fenwick tree can do each operation in $O(\log n)$ time.

K - Kid's First Geometry Problem

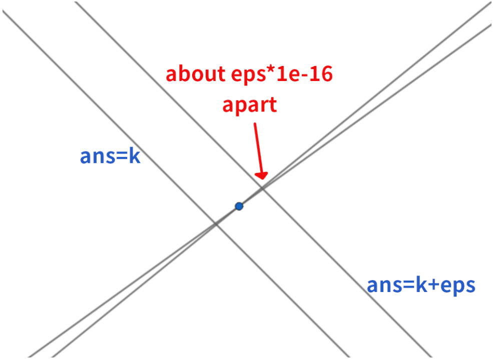
Alternative solution: For each i, binary search on the number of edges in A below $y = -c$. Then, we can find the exact number of edges in B_i below it with another binary search. It's easy to check whether we took too many or too less edges from A.
This requires trickier implementation details and a lot of edge case handling but also uses only one $\log$ factor.

K - Kid's First Geometry Problem

Caveat: Using floating points probably doesn't pass.

K - Kid's First Geometry Problem

Caveat: Using floating points probably doesn't pass.
Reason: Coordinates can be up to 10^8, and the angle between two lines can be about 10^{-16}. The answer must have a relative error below 10^{-8}, so about 10^{-24} precision is required.



K - Kid's First Geometry Problem

The total time complexity:

$$O(N \log N \log C)$$

where $N = n + \sum m_i$ and C is the coordinate range.

K - Kid's First Geometry Problem

The total time complexity:

$$O(N \log N \log C)$$

where $N = n + \sum m_i$ and C is the coordinate range.

Fun fact: The original (easier) version is the validator for this problem, and its intended time complexity is $O(N \log N)$.