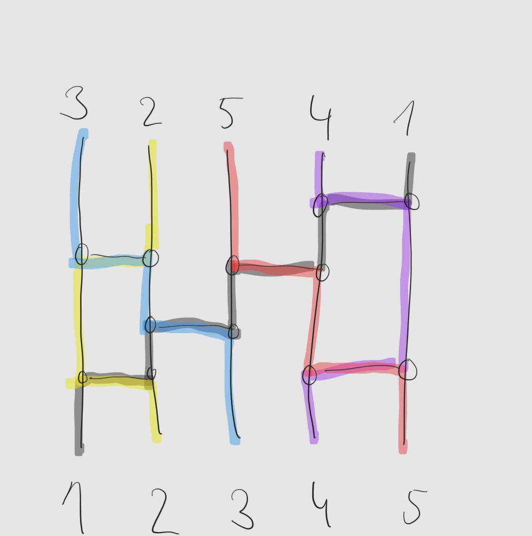
CTU Open 2025

Presentation of solutions

November 8, 2025

Grooves

Grooves



Grooves - Solution

Grooves - Solution

- Start with the initial permutation and perform the swaps for each groove in order of non-decreasing Y-coordinates.

Grooves - Solution

- Start with the initial permutation and perform the swaps for each groove in order of non-decreasing Y-coordinates.
- $O(N \log N)$ for sorting by Y-coordinates, the rest is $O(N)$.

Grooves - Solution

- Start with the initial permutation and perform the swaps for each groove in order of non-decreasing Y-coordinates.
- $O(N \log N)$ for sorting by Y-coordinates, the rest is $O(N)$.
- Alternatively: simulate it for each initial position. Each groove is considered at most two times.

Grooves - Solution

- Start with the initial permutation and perform the swaps for each groove in order of non-decreasing Y-coordinates.
- $O(N \log N)$ for sorting by Y-coordinates, the rest is $O(N)$.
- Alternatively: simulate it for each initial position. Each groove is considered at most two times. It can be implemented in $O(N \log N)$.

Grooves - Solution

- Start with the initial permutation and perform the swaps for each groove in order of non-decreasing Y-coordinates.
- $O(N \log N)$ for sorting by Y-coordinates, the rest is $O(N)$.
- Alternatively: simulate it for each initial position. Each groove is considered at most two times. It can be implemented in $O(N \log N)$.
- Fun fact: 'Amidakuji'

Constatine

Constantine - Solution

Observation: The notation encodes binary representation!

Parsing Algorithm:

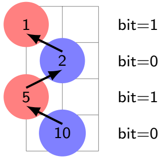
- Start from bottom row (value N)
- Track column positions upward
- Even number $\rightarrow$ move LEFT
- Odd number $\rightarrow$ move RIGHT
- Divide by 2, repeat until $= 1$

Example: $10 = 1010_2$

#.   <- 1   bit:1 2^3
.#   <- 2   bit:0 2^2
#.   <- 5   bit:1 2^1
.#   <- 10 bit:0 2^0

Each step encodes one binary bit!

Visual Representation:



Blue = even (bit=0), Red = odd (bit=1)

Binary: Read top to bottom: $1010_2 = 10$

Solution: Parse $\rightarrow$ Add $\rightarrow$ Encode

Snow

Snow

- **Input:** $N \times M$ grid where each cell is either snow or empty.
- **Output:** At given timesteps determine how many snowflakes can't fall anymore.

```
...*.
.*..*
..*..
...*.
.*.*.
*....
```

Snow

- **Input:** $N \times M$ grid where each cell is either snow or empty.
- **Output:** At given timesteps determine how many snowflakes can't fall anymore.

```
.....
...*.
.*..*
..*..
...*.
**.*.
```

- For each snowflake determine how long it can fall.

Snow

- **Input:** $N \times M$ grid where each cell is either snow or empty.
- **Output:** At given timesteps determine how many snowflakes can't fall anymore.

```
.....
.....
...*.
.*..*
..**.
**.*.
```

- For each snowflake determine how long it can fall.

Snow

- **Input:** $N \times M$ grid where each cell is either snow or empty.
- **Output:** At given timesteps determine how many snowflakes can't fall anymore.

```
.....
.....
.....
...*.
.*.**
****.
```

- For each snowflake determine how long it can fall.

Snow

- **Input:** $N \times M$ grid where each cell is either snow or empty.
- **Output:** At given timesteps determine how many snowflakes can't fall anymore.

```
.....
.....
.....
...*.
.*.*.
*****
```

- For each snowflake determine how long it can fall.

Snow

- **Input:** $N \times M$ grid where each cell is either snow or empty.
- **Output:** At given timesteps determine how many snowflakes can't fall anymore.

```
...*.
.*..*
..*..
...*.
.*.*.
*....
```

- For each snowflake determine how long it can fall.
- Consider one snowflake, a snowflake underneath shortens the fall by 1.

Snow

- **Input:** $N \times M$ grid where each cell is either snow or empty.
- **Output:** At given timesteps determine how many snowflakes can't fall anymore.

```
...*.
.*..*
..*..
...*.
.*.*.
*....
```

- For each snowflake determine how long it can fall.
- Consider one snowflake, a snowflake underneath shortens the fall by 1.
- For each column start from the bottom, for each empty space increase the time by 1, for stars assign the current time.

Snow

- **Input:** $N \times M$ grid where each cell is either snow or empty.
- **Output:** At given timesteps determine how many snowflakes can't fall anymore.

```
...*.
.*..*
..*..
...*.
.*.*.
*....
```

- For each snowflake determine how long it can fall.
- Consider one snowflake, a snowflake underneath shortens the fall by 1.
- For each column start from the bottom, for each empty space increase the time by 1, for stars assign the current time.
- Aggregate the stars by the time and use prefix sum.

Snow

- **Input:** $N \times M$ grid where each cell is either snow or empty.
- **Output:** At given timesteps determine how many snowflakes can't fall anymore.

```
...*.
.*..*
..*..
...*.
.*.*.
*....
```

- For each snowflake determine how long it can fall.
- Consider one snowflake, a snowflake underneath shortens the fall by 1.
- For each column start from the bottom, for each empty space increase the time by 1, for stars assign the current time.
- Aggregate the stars by the time and use prefix sum.
- Answer the querries.

Archery

Archery

Archery

- **Input**: Rooks on a large chessboard, we can 'shoot' from (x_1, y_1) to (x_2, y_2) if $x_1 = x_2$ or $y_1 = y_2$.

Archery

- **Input**: Rooks on a large chessboard, we can 'shoot' from (x_1, y_1) to (x_2, y_2) if $x_1 = x_2$ or $y_1 = y_2$. No matter where we place start and finish, we should be able to reach from start to finish.

Archery

- **Input**: Rooks on a large chessboard, we can 'shoot' from (x_1, y_1) to (x_2, y_2) if $x_1 = x_2$ or $y_1 = y_2$. No matter where we place start and finish, we should be able to reach from start to finish.
- **Output**: Minimum number of vertical/horizontal moves such that we can reach between any two rooks.

Archery

- **Input**: Rooks on a large chessboard, we can 'shoot' from (x_1, y_1) to (x_2, y_2) if $x_1 = x_2$ or $y_1 = y_2$. No matter where we place start and finish, we should be able to reach from start to finish.
- **Output**: Minimum number of vertical/horizontal moves such that we can reach between any two rooks.
- Transformation to a graph: $(x_1, y_1), (x_2, y_2)$ are adjacent iff $x_1 = x_2$ or $y_1 = y_2$.

Archery

- **Input**: Rooks on a large chessboard, we can 'shoot' from (x_1, y_1) to (x_2, y_2) if $x_1 = x_2$ or $y_1 = y_2$. No matter where we place start and finish, we should be able to reach from start to finish.
- **Output**: Minimum number of vertical/horizontal moves such that we can reach between any two rooks.
- Transformation to a graph: $(x_1, y_1), (x_2, y_2)$ are adjacent iff $x_1 = x_2$ or $y_1 = y_2$.
- If the graph is connected, output 0.

Archery cont.

Archery cont.

- If the graph is disconnected, we must perform some moves. Note that we can "jump" over other pieces.

Archery cont.

- If the graph is disconnected, we must perform some moves. Note that we can "jump" over other pieces.
- By moving one rook, we cannot decrease the number of connected components by more than 1.

Archery cont.

- If the graph is disconnected, we must perform some moves. Note that we can "jump" over other pieces.
- By moving one rook, we cannot decrease the number of connected components by more than 1.
- On the other hand, we can always decrease the number of components by 1 by a clever move.

Archery cont.

- If the graph is disconnected, we must perform some moves. Note that we can "jump" over other pieces.
- By moving one rook, we cannot decrease the number of connected components by more than 1.
- On the other hand, we can always decrease the number of components by 1 by a clever move.
- $\Rightarrow$ answer is number of connected components minus one.

Archery cont. 2

Archery cont. 2

- $\Rightarrow$ answer is number of connected components minus one.

Archery cont. 2

- $\Rightarrow$ answer is number of connected components minus one.
- Note that the graph can have N^2 edges, no need to store all of them. Just store edges for neighboring Y-coordinates for same X (and vice-versa for X-coords for same Y)

Archery cont. 2

- $\Rightarrow$ answer is number of connected components minus one.
- Note that the graph can have N^2 edges, no need to store all of them. Just store edges for neighboring Y-coordinates for same X (and vice-versa for X-coords for same Y)
- Such subgraph H has $O(N)$ edges and connected components of H coincide with connected components of G and it can be constructed in $O(N \log N)$.

Hussites

Hussites

The Setup:

- Given N wagons (points in a 2D plane) and a required percentage P.
- Constraints: $N \leq 1500$. Coordinates are integers between -10^6 and 10^6.

The Goal:

- Determine if there exists a "false escape route" (a straight line L) such that at least $P\%$ of the wagons are "powerful".

Hussites

The Setup:

- Given N wagons (points in a 2D plane) and a required percentage P.
- Constraints: $N \leq 1500$. Coordinates are integers between -10^6 and 10^6.

The Goal:

- Determine if there exists a "false escape route" (a straight line L) such that at least $P\%$ of the wagons are "powerful".

Solution

1. Testing each line defined by two points is too slow - $O(N^3)$.

Hussites

The Setup:

- Given N wagons (points in a 2D plane) and a required percentage P.
- Constraints: $N \leq 1500$. Coordinates are integers between -10^6 and 10^6.

The Goal:

- Determine if there exists a "false escape route" (a straight line L) such that at least $P\%$ of the wagons are "powerful".

Solution

1. Testing each line defined by two points is too slow - $O(N^3)$.
2. Find canonical name of an bisector for each pair of points.

Hussites

The Setup:

- Given N wagons (points in a 2D plane) and a required percentage P.
- Constraints: $N \leq 1500$. Coordinates are integers between -10^6 and 10^6.

The Goal:

- Determine if there exists a "false escape route" (a straight line L) such that at least $P\%$ of the wagons are "powerful".

Solution

1. Testing each line defined by two points is too slow - $O(N^3)$.
2. Find canonical name of an bisector for each pair of points.
3. If there are $N * P/100$ copies of it, output `YES`.

Hussites

The Setup:

- Given N wagons (points in a 2D plane) and a required percentage P.
- Constraints: $N \leq 1500$. Coordinates are integers between -10^6 and 10^6.

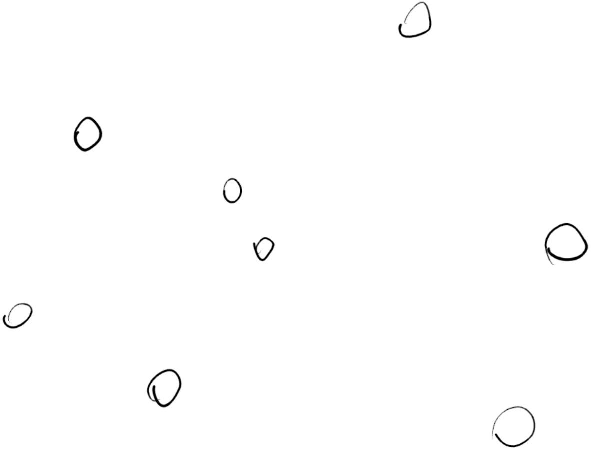
The Goal:

- Determine if there exists a "false escape route" (a straight line L) such that at least $P\%$ of the wagons are "powerful".

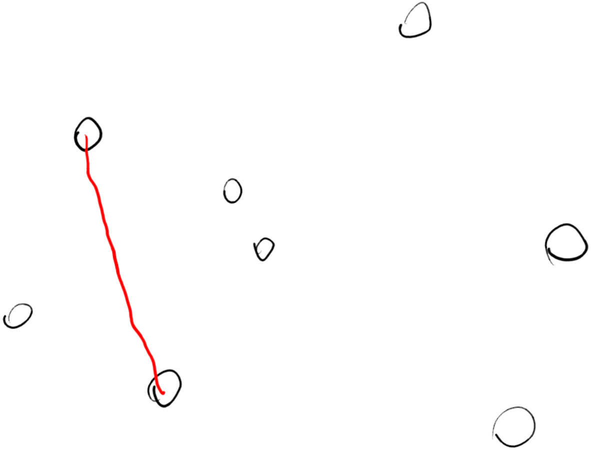
Solution

1. Testing each line defined by two points is too slow - $O(N^3)$.
2. Find canonical name of an bisector for each pair of points.
3. If there are $N * P/100$ copies of it, output `YES`.
4. Total complexity $O(N^2)$.

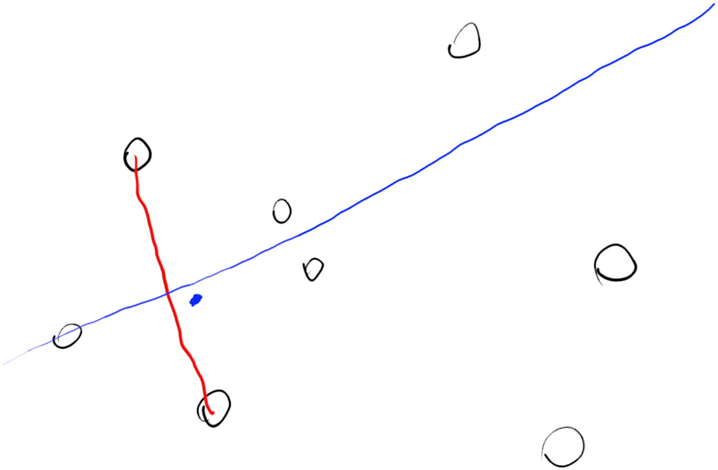
Hussites



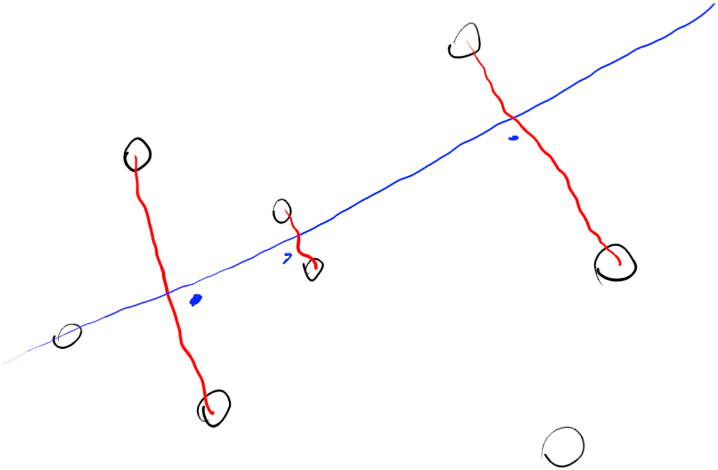
Hussites



Hussites



Hussites



Ducks

Ducks

We have N distinct grillsticks. Each grillstick holds exactly 4 animals. The order of animals on the sticks, and the order of the sticks, matters.

1. All N grillsticks are used (Total animals $= 4N$).
2. Total number of Ducks $=$ Total number of Hares.
3. At least one grillstick must contain only ducks (DDDD).

Goal: Find the total number of distinct feast configurations satisfying these rules, modulo $10^9 + 7$.

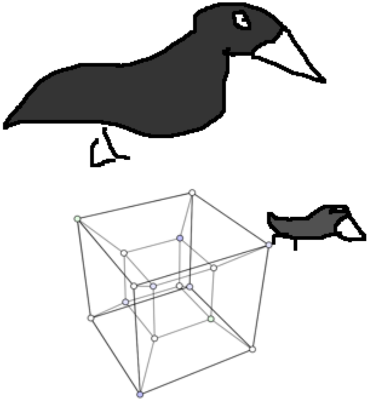
We use the PIE formula. For a fixed k, there are $\binom{N}{k}$ ways to choose which k sticks are fixed to DDDD. We multiply this by the number of ways to configure the remaining sticks.

Result

The total number of distinct feasts is:

$$\sum_{k=1}^{\lfloor N/2 \rfloor} (-1)^{k-1} \binom{N}{k} \binom{4N-4k}{2N-4k}$$

Ornithology



Ornithology

Problem Statement:

Input: N vectors with D elements
Task: Find minimum number of steps to transform vectors such that:

- All vectors have equal coordinates in all dimensions except one
- Pairwise Manhattan distances are preserved
- One step $=$ changing one coordinate of one vector by one

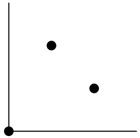
Manhattan distance

For x, y: $\sum_{i=1}^{D} |x_i - y_i|$

Ornithology

Key Observation:

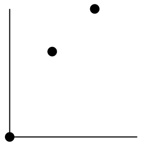
- Consider vectors in pairs of dimensions
- If any triple of vectors forms a non-monotone sequence: Solution does not exist
- Otherwise, a solution is guaranteed to exist



Ornithology

Idea of solution:

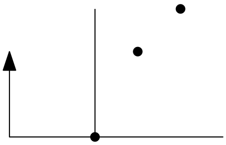
- Fix just two dimensions.
- The vectors must make a non-increasing/non-decreasing sequence.
- Grab a block of vectors and move so that they align with the next vector.



Ornithology

Idea of solution:

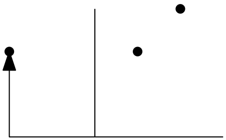
- Fix just two dimensions.
- The vectors must make a non-increasing/non-decreasing sequence.
- Grab a block of vectors and move so that they align with the next vector.



Ornithology

Idea of solution:

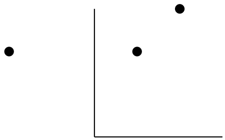
- Fix just two dimensions.
- The vectors must make a non-increasing/non-decreasing sequence.
- Grab a block of vectors and move so that they align with the next vector.



Ornithology

Idea of solution:

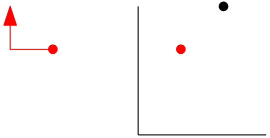
- Fix just two dimensions.
- The vectors must make a non-increasing/non-decreasing sequence.
- Grab a block of vectors and move so that they align with the next vector.



Ornithology

Idea of solution:

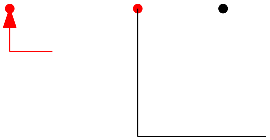
- Fix just two dimensions.
- The vectors must make a non-increasing/non-decreasing sequence.
- Grab a block of vectors and move so that they align with the next vector.



Ornithology

Idea of solution:

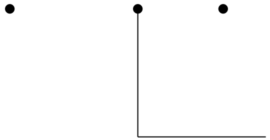
- Fix just two dimensions.
- The vectors must make a non-increasing/non-decreasing sequence.
- Grab a block of vectors and move so that they align with the next vector.



Ornithology

Idea of solution:

- Fix just two dimensions.
- The vectors must make a non-increasing/non-decreasing sequence.
- Grab a block of vectors and move so that they align with the next vector.



Ornithology

Efficient solution:

- Order vectors along first dimension.
- Determine which dimensions should be non-increasing/non-decreasing.
- Sort again.
- Check whether all dimensions are non-increasing/non-decreasing.
- The resulting order of vectors will be preserved in the solution.

Ornithology

Minimum number of operations:

- Assume middle vector m (or one of two middle vectors) stays fixed.
- If vector v has distance d from m along dimension D, it needs to move by
  - distance d to reach the same value in dimension D,
  - then distanace d to preserve the Manhatten distance from m.
- Precompute needed steps for all dimensions.
- Quickly compute total number of steps for each candidate dimension.

JJ

JJ the Almighty

Problem Setup

- **Input:**
  - Grid graph G
  - Two edge-labelings S and T (labels $\in \{0, 1\}$)
- **Question:** Can we transform S into T using:
  1. Edge flips along horizontal/vertical lines
  2. Edge flips along cycles

Key Observations

- Represent labelings as $0/1$ vectors
- Operations (O_i): vectors with 1s on flipped edges
- Let $X = O_1 \oplus O_2 \oplus \cdots \oplus O_k$ be operation sum
- $S \oplus X = T \iff S \oplus T = X$
- **Goal**: Check if $S \oplus T$ can be generated by operations

JJ the Almighty

Vector space generated by

H_i: Vectors for horizontal line operations (N vectors)

V_i: Vectors for vertical line operations (M vectors)

C_i: Vectors for cycle operations (very high number)

$C'_{x,y}$: Single tile cycles on x, y, **equivalent to** C_i ($N \times M$ vectors)

Solution

- Note $rank(\langle H_i, V_i, C_i \rangle) = rank(\langle H_i, V_i, C'_{x,y} \rangle)$
- Solution exists if and only if: $rank(\langle H_i, V_i, C'_{x,y} \rangle) = rank(\langle H_i, V_i, C'_{x,y}, S \oplus T \rangle)$
- Use Gaussian elimination to compute ranks.
- The matrix has size roughly $(N \times M) \times (N \times M)$.

JJ the Almighty - alternative approach

Transform grid to toroid

1. Identify top border vertices with bottom border vertices
2. Identify left border vertices with right border vertices
3. Result: Grid G becomes toroid G'

Key Insights

- $\langle H_i, V_i, C_i \rangle$ on G = cycle space on G'
- Cycle space = set of all Eulerian circuits
- Solution exists $\iff$ X is Eulerian on G'

Verification

Check that all vertex degrees in X on G' are even

Book Burning

Book burning

- **Input:** Large string S and many small strings strings $s_1, s_2, \dots s_Q$.
- **Output:** For each s_i count the number of occurences in S and then delete them from S.

```
yabbadabbadoo
0123456789012
```

Book burning

- **Input:** Large string S and many small strings strings $s_1, s_2, \dots s_Q$.
- **Output:** For each s_i count the number of occurences in S and then delete them from S.

```
yabbadabbadoo
0123456789012
```

- Just simulating it is too slow. We need to use that $|s_i| \leq 5$.

Book burning

- **Input:** Large string S and many small strings strings $s_1, s_2, \ldots s_Q$.
- **Output:** For each s_i count the number of occurences in S and then delete them from S.

```
yabbadabbadoo
0123456789012
```

- Just simulating it is too slow. We need to use that $|s_i| \leq 5$.
- Each substring of size ≤ 5 can intersect only $2 \cdot \binom{5}{2}$ other substrings of size ≤ 5.

Book burning

- **Input:** Large string S and many small strings strings $s_1, s_2, \ldots s_Q$.
- **Output:** For each s_i count the number of occurences in S and then delete them from S.

```
yabbadabbadoo
0123456789012
```

- yab $\rightarrow [0]$
- abb $\rightarrow [1, 6]$
- bba $\rightarrow [1, 6]$
- $\ldots$

$s_0 =$dad$\rightarrow 0$
$s_1 =$bba$\rightarrow 2$
$s_2 =$dad$\rightarrow 1$

- Just simulating it is too slow. We need to use that $|s_i| \leq 5$.
- Each substring of size ≤ 5 can intersect only $2 \cdot \binom{5}{2}$ other substrings of size ≤ 5.
- For each substring of S of size ≤ 5 note its position.

Book burning

- **Input:** Large string S and many small strings strings $s_1, s_2, \ldots s_Q$.
- **Output:** For each s_i count the number of occurences in S and then delete them from S.

```
yabbadabbadoo
0123456789012
```

- yab $\rightarrow [0]$
- abb $\rightarrow [1, 6]$
- bba $\rightarrow [1, 6]$
- $\ldots$

$s_0 =$dad$\rightarrow 0$
$s_1 =$bba$\rightarrow 2$
$s_2 =$dad$\rightarrow 1$

- Just simulating it is too slow. We need to use that $|s_i| \leq 5$.
- Each substring of size ≤ 5 can intersect only $2 \cdot \binom{5}{2}$ other substrings of size ≤ 5.
- For each substring of S of size ≤ 5 note its position.
- For one querry $|s_i|$ go through each of the occurences, invalidate all the intersecting substrings and calculate new strings.

Book burning

- **Input:** Large string S and many small strings strings $s_1, s_2, \ldots s_Q$.
- **Output:** For each s_i count the number of occurences in S and then delete them from S.

```
yabbadabbadoo
0123456789012
```

- yab $\rightarrow [0]$
- abb $\rightarrow [1, 6]$
- bba $\rightarrow [1, 6]$
- $\ldots$

$s_0 = \mathrm{dad} \rightarrow 0$
$s_1 = \mathrm{bba} \rightarrow 2$
$s_2 = \mathrm{dad} \rightarrow 1$

- Just simulating it is too slow. We need to use that $|s_i| \leq 5$.
- Each substring of size ≤ 5 can intersect only $2 \cdot \binom{5}{2}$ other substrings of size ≤ 5.
- For each substring of S of size ≤ 5 note its position.
- For one querry $|s_i|$ go through each of the occurences, invalidate all the intersecting substrings and calculate new strings.

Book burning

- **Input:** Large string S and many small strings strings $s_1, s_2, \ldots s_Q$.
- **Output:** For each s_i count the number of occurences in S and then delete them from S.

yabbadabbadoo
0123456789012

- yab $\rightarrow [0]$
- abb $\rightarrow [1, 6]$
- bba $\rightarrow [1, 6]$
- ...

s_0 =dad$\rightarrow$ 0
s_1 =bba$\rightarrow$ 2
s_2 =dad$\rightarrow$ 1

- Just simulating it is too slow. We need to use that $|s_i| \leq 5$.
- Each substring of size ≤ 5 can intersect only $2 \cdot \binom{5}{2}$ other substrings of size ≤ 5.
- For each substring of S of size ≤ 5 note its position.
- For one querry $|s_i|$ go through each of the occurences, invalidate all the intersecting substrings and calculate new strings.

Book burning

- **Input:** Large string S and many small strings strings $s_1, s_2, \dots s_Q$.
- **Output:** For each s_i count the number of occurences in S and then delete them from S.

`yabbadabbadoo`
`0123456789012`

- yab $\rightarrow [0]$
- abb $\rightarrow [1, 6]$
- dad $\rightarrow [5]$
- $\dots$

  $s_0 =$dad$\rightarrow 0$
  $s_1 =$bba$\rightarrow 2$
  $s_2 =$dad$\rightarrow 1$

- Just simulating it is too slow. We need to use that $|s_i| \leq 5$.
- Each substring of size ≤ 5 can intersect only $2 \cdot \binom{5}{2}$ other substrings of size ≤ 5.
- For each substring of S of size ≤ 5 note its position.
- For one querry $|s_i|$ go through each of the occurences, invalidate all the intersecting substrings and calculate new strings.

Meetings

Meetings

Meetings

- **Input:** N intervals, C 'daggers', Q queries moving daggers to different places.

Meetings

- **Input:** N intervals, C 'daggers', Q queries moving daggers to different places.
- **Output:** For each query: 'How many intervals are not stabbed by any dagger?'

Meetings - Solution

Meetings - Solution

- Split the query ‘move from f to t’ into two queries: ‘remove from f’ and ‘add to t’.

Meetings - Solution

- Split the query ‘move from f to t’ into two queries: ‘remove from f’ and ‘add to t’.
- Keep set of active daggers in a set. Adding a dagger c: find two neighboring daggers c_p and c_n.

Meetings - Solution

- Split the query 'move from f to t' into two queries: 'remove from f' and 'add to t'.
- Keep set of active daggers in a set. Adding a dagger c: find two neighboring daggers c_p and c_n.
- Intervals inside $[c_p + 1, c - 1]$ and $[c + 1, c_n - 1]$ are uncovered, but those in $[c_p + 1, c_n - 1]$ were also uncovered before.

Meetings - Solution

- Split the query ‘move from f to t’ into two queries: ‘remove from f’ and ‘add to t’.
- Keep set of active daggers in a set. Adding a dagger c: find two neighboring daggers c_p and c_n.
- Intervals inside $[c_p + 1, c - 1]$ and $[c + 1, c_n - 1]$ are uncovered, but those in $[c_p + 1, c_n - 1]$ were also uncovered before.
- Update current uncovered number of intervals by the value $X = \mathrm{cnt}([c_p + 1, c - 1]) + \mathrm{cnt}([c + 1, c_n - 1]) - \mathrm{cnt}([c_p + 1, c_n - 1])$, where $\mathrm{cnt}(I)$ is the number of intervals completely inside the interval I.

Meetings - Solution

- Split the query ‘move from f to t’ into two queries: ‘remove from f’ and ‘add to t’.
- Keep set of active daggers in a set. Adding a dagger c: find two neighboring daggers c_p and c_n.
- Intervals inside $[c_p + 1, c - 1]$ and $[c + 1, c_n - 1]$ are uncovered, but those in $[c_p + 1, c_n - 1]$ were also uncovered before.
- Update current uncovered number of intervals by the value $X = \mathrm{cnt}([c_p + 1, c - 1]) + \mathrm{cnt}([c + 1, c_n - 1]) - \mathrm{cnt}([c_p + 1, c_n - 1])$, where $\mathrm{cnt}(I)$ is the number of intervals completely inside the interval I.
- By similar logic, removing a dagger at c updates the total count by $-X$.

Meetings - Online solution to queries

Meetings - Online solution to queries

- Hence, we need to answer queries:

Meetings - Online solution to queries

- Hence, we need to answer queries: Given a set of intervals N (endpoints in $[1, \ldots, 10^5]$),

Meetings - Online solution to queries

- Hence, we need to answer queries: Given a set of intervals N (endpoints in $[1, \ldots, 10^5]$), for given ℓ, r find the number of intervals contained in $[\ell, r]$. That is, number of $[x, y]$ such that $\ell \leq x, y \leq r$.

Meetings - Online solution to queries

- Hence, we need to answer queries: Given a set of intervals N (endpoints in $[1, \ldots, 10^5]$), for given ℓ, r find the number of intervals contained in $[\ell, r]$. That is, number of $[x, y]$ such that $\ell \leq x, y \leq r$.
- Represent intervals as points in $2D$-plane. Number of intervals contained in $[x, y]$ is equal to the number of points in the rectangle with corners $(x, 1), (x, y), (N, y), (N, 1)$.

Meetings - Online solution to queries

- Hence, we need to answer queries: Given a set of intervals N (endpoints in $[1, \ldots, 10^5]$), for given ℓ, r find the number of intervals contained in $[\ell, r]$. That is, number of $[x, y]$ such that $\ell \leq x, y \leq r$.
- Represent intervals as points in $2D$-plane. Number of intervals contained in $[x, y]$ is equal to the number of points in the rectangle with corners $(x, 1), (x, y), (N, y), (N, 1)$.
- Use your favourite data structure for storing points that allows queries for number of points in rectangles, e.g., sparse $2D$ segment/fenwick tree

Meetings - Online solution to queries

- Hence, we need to answer queries: Given a set of intervals N (endpoints in $[1, \ldots, 10^5]$), for given ℓ, r find the number of intervals contained in $[\ell, r]$. That is, number of $[x, y]$ such that $\ell \leq x, y \leq r$.
- Represent intervals as points in $2D$-plane. Number of intervals contained in $[x, y]$ is equal to the number of points in the rectangle with corners $(x, 1), (x, y), (N, y), (N, 1)$.
- Use your favourite data structure for storing points that allows queries for number of points in rectangles, e.g., sparse $2D$ segment/fenwick tree
- Complexity $O(N + Q \log T \log N)$, where $T \leq 10^5$ is the maximum time.

Meetings - Offline solution to queries

Meetings - Offline solution to queries

- Alternatively, we can solve the queries offline.

Meetings - Offline solution to queries

- Alternatively, we can solve the queries offline.
- We have two types of intervals: the original intervals and the query intervals.

Meetings - Offline solution to queries

- Alternatively, we can solve the queries offline.
- We have two types of intervals: the original intervals and the query intervals.
- Build a segment tree over the timeline.

Meetings - Offline solution to queries

- Alternatively, we can solve the queries offline.
- We have two types of intervals: the original intervals and the query intervals.
- Build a segment tree over the timeline.
- When processing right endpoint of an original interval, update $+1$ to its left point.

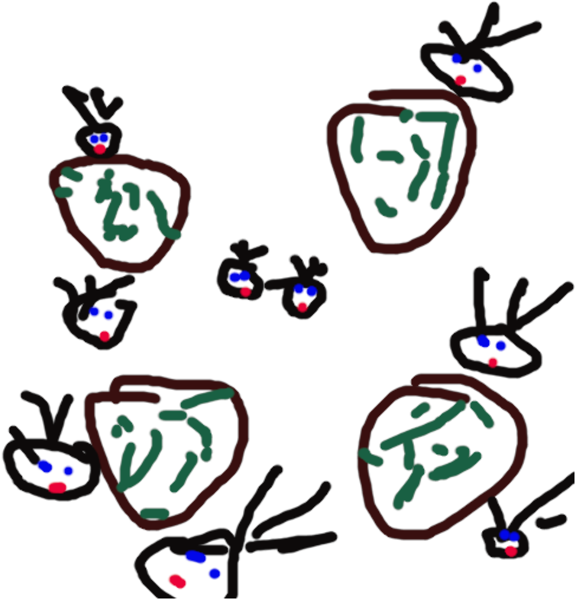
Meetings - Offline solution to queries

- Alternatively, we can solve the queries offline.
- We have two types of intervals: the original intervals and the query intervals.
- Build a segment tree over the timeline.
- When processing right endpoint of an original interval, update $+1$ to its left point.
- When processing right endpoint of a query interval, query the segment tree for the interval – this gives the answer to the query interval.

Meetings - Offline solution to queries

- Alternatively, we can solve the queries offline.
- We have two types of intervals: the original intervals and the query intervals.
- Build a segment tree over the timeline.
- When processing right endpoint of an original interval, update $+1$ to its left point.
- When processing right endpoint of a query interval, query the segment tree for the interval – this gives the answer to the query interval.
- Complexity $O((Q+N)\log T)$, where $T \leq 10^5$ is the maximum time.

Emigrants



Problem Statement

Given: Array $A[1..N]$, threshold K. Compatible pair: $\gcd(A_i, A_j) > 1$. Smart group: contiguous subseq. with $\geq K$ compatible pairs.

Key Insights:

1. $\gcd(A_i, A_j) > 1 \iff$ they share at least one prime divisor
2. Use **two-pointers** to count valid subsequences
3. Use **inclusion-exclusion** to count compatible pairs efficiently

Example: $A = [2, 3, 4, 5, 6]$, $K = 1$
Compatible pairs: $(2, 4)$, $(2, 6)$, $(3, 6)$, $(4, 6)$ $\rightarrow$ 4 pairs
Smart groups: $[2, 3, 4]$, $[2, 3, 4, 5]$, $[2, 3, 4, 5, 6]$, $[3, 4, 5, 6]$, $[4, 5, 6]$ $\rightarrow$ 5 groups

Solution Approach

Algorithm:

1. **Preprocess:** Prime Sieve
2. **Two Pointers:** Window $[L, R]$
   - Extend R while quality $< K$
   - If quality $\geq K$: add $N - R + 1$
   - Move L, update quality
3. **Inclusion-Exclusion:**
   - Extract prime divisors
   - Count pairs via subsets

Incl-Excl: For primes $\{p_1, \ldots, p_k\}$:

$$\text{pairs} = \sum_{\emptyset \neq S \subseteq \{1..k\}} (-1)^{|S|+1} \text{cnt}\left(\prod_{i \in S} p_i\right)$$

Data Structures:

- `counts[d]`: # elements with divisor d
- `quality`: # compatible pairs

Example - Adding $6 = 2 \times 3$:

Subsets: $\{2\}$, $\{3\}$, $\{2, 3\}$

$+$cnt[2] (sharing 2)
$+$cnt[3] (sharing 3)
$-$cnt[6] (avoid dbl-cnt)

Complexity:

- Sieve: $O(M \log\log M)$, $M = 5 \times 10^5$
- Per element: $O(2^{\omega(A_i)})$ where $\omega(A_i) \leq 7$
- Two pointers: $O(N)$
- **Total:** $O(M \log\log M + N \cdot 2^{\omega(A)})$

Theatre

Theatre

Task

Given N regions in a painting and M forbidden pairs. We need to find the number of ways to color the regions such that no forbidden pair shares the same color, for various palette sizes (k).

Graph Modeling

We model this as a graph coloring problem $G = (V, E)$:

- Regions $\rightarrow$ Vertices (V).
- Forbidden pairs $\rightarrow$ Edges (E).
- We seek the number of **proper colorings** of G using k colors.

Theatre continue

Definition

The **Chromatic Polynomial** $P(G, k)$ counts the number of proper vertex colorings of a graph G using at most k colors.

Strategy

Since we have many queries (T) and the palette size (k) can be large, we must compute the polynomial $P(G, k)$ explicitly beforehand. Once we have the polynomial, we can evaluate $P(G, k)$ quickly for any k (modulo $10^9 + 7$).

Base Case

If G has N vertices and no edges, $P(G, k) = k^N$.

Theatre continue

Theorem

For any edge $e = (u, v)$ in G:

$$P(G, k) = P(G - e, k) - P(G/e, k)$$

$G - e$ (Deletion) The graph G with the edge e removed.

G/e (Contraction) The graph G where vertices u and v are merged into a single vertex, maintaining connectivity to neighbors.

Complexity

The recursion depth is M, leading to $O(2^M \cdot poly(N))$ time complexity for computing the polynomial.