IV Кубок України з програмування

Official Solutions
Contest 3, div1
2025

A. Life game

- Given a matrix
  - of size up to 50

A. Life game

- Given a matrix
  - of size up to 50
- Color each element of matrix in one of two colors

A. Life game

- Given a matrix
  - of size up to 50
- Color each element of matrix in one of two colors
- You have up to 50 000 awards you can get
  - x_1, x_2, y_1, y_2, c and m
  - You get an award m if submatrix (x_1, x_2, y_1, y_2) is colored in c

A. Life game

- Given a matrix
  - of size up to 50
- Color each element of matrix in one of two colors
- You have up to 50 000 awards you can get
  - x_1, x_2, y_1, y_2, c and m
  - You get an award m if submatrix (x_1, x_2, y_1, y_2) is colored in c
- There are also awards for each cell colored
  - These one can be converted to the same awards as before
    - One cell is also submatrix

A. Life game

- Given a matrix
  - of size up to 50
- Color each element of matrix in one of two colors
- You have up to 50 000 awards you can get
  - x_1, x_2, y_1, y_2, c and m
  - You get an award m if submatrix (x_1, x_2, y_1, y_2) is colored in c
- There are also awards for each cell colored
  - These one can be converted to the same awards as before
    - One cell is also submatrix
- Earn maximum amount of money

A. Life game

Solution

- Dividing elements into two subsets reminds of $\langle S, T \rangle$-cut

A. Life game

Solution

- Dividing elements into two subsets reminds of $\langle S, T\rangle$-cut
- Build a network, so that there is a bijection between $\langle S, T\rangle$-cuts and different colorings

A. Life game

Solution

- Dividing elements into two subsets reminds of $\langle S, T \rangle$-cut
- Build a network, so that there is a bijection between $\langle S, T \rangle$-cuts and different colorings
  - Make cut value equal to corresponding amount of money

A. Life game

Solution

- Dividing elements into two subsets reminds of $\langle S, T\rangle$-cut
- Build a network, so that there is a bijection between $\langle S, T\rangle$-cuts and different colorings
  - Make cut value equal to corresponding amount of money
- We can find minimum cut and we have to maximize answer

A. Life game

Solution

- Dividing elements into two subsets reminds of $\langle S, T\rangle$-cut
- Build a network, so that there is a bijection between $\langle S, T\rangle$-cuts and different colorings
  - Make cut value equal to corresponding amount of money
- We can find minimum cut and we have to maximize answer
  - Assume that we already earned all the money and have to return the money for non-satisfied awards

A. Life game

Solution

- Dividing elements into two subsets reminds of $\langle S, T\rangle$-cut
- Build a network, so that there is a bijection between $\langle S, T\rangle$-cuts and different colorings
  - Make cut value equal to corresponding amount of money
- We can find minimum cut and we have to maximize answer
  - Assume that we already earned all the money and have to return the money for non-satisfied awards
  - Minimize amount of money we have to return

A. Life game

Solution

- Dividing elements into two subsets reminds of $\langle S, T\rangle$-cut
- Build a network, so that there is a bijection between $\langle S, T\rangle$-cuts and different colorings
  - Make cut value equal to corresponding amount of money
- We can find minimum cut and we have to maximize answer
  - Assume that we already earned all the money and have to return the money for non-satisfied awards
  - Minimize amount of money we have to return
  - Let this amount be the value of the cut

A. Life game

Solution

- Dividing elements into two subsets reminds of $\langle S, T\rangle$-cut
- Build a network, so that there is a bijection between $\langle S, T\rangle$-cuts and different colorings
  - Make cut value equal to corresponding amount of money
- We can find minimum cut and we have to maximize answer
  - Assume that we already earned all the money and have to return the money for non-satisfied awards
  - Minimize amount of money we have to return
  - Let this amount be the value of the cut
- Consider one award, let's say $c = 0$

A. Life game

Solution

- Dividing elements into two subsets reminds of $\langle S, T\rangle$-cut
- Build a network, so that there is a bijection between $\langle S, T\rangle$-cuts and different colorings
  - Make cut value equal to corresponding amount of money
- We can find minimum cut and we have to maximize answer
  - Assume that we already earned all the money and have to return the money for non-satisfied awards
  - Minimize amount of money we have to return
  - Let this amount be the value of the cut
- Consider one award, let's say $c = 0$
  - Create one vertex v for this award

A. Life game

Solution

- Dividing elements into two subsets reminds of $\langle S, T\rangle$-cut
- Build a network, so that there is a bijection between $\langle S, T\rangle$-cuts and different colorings
  - Make cut value equal to corresponding amount of money
- We can find minimum cut and we have to maximize answer
  - Assume that we already earned all the money and have to return the money for non-satisfied awards
  - Minimize amount of money we have to return
  - Let this amount be the value of the cut
- Consider one award, let's say $c = 0$
  - Create one vertex v for this award
  - Add edges of $+\infty$ capacity from v to vertices in its submatrix

A. Life game

Solution

- Dividing elements into two subsets reminds of $\langle S, T\rangle$-cut
- Build a network, so that there is a bijection between $\langle S, T\rangle$-cuts and different colorings
  - Make cut value equal to corresponding amount of money
- We can find minimum cut and we have to maximize answer
  - Assume that we already earned all the money and have to return the money for non-satisfied awards
  - Minimize amount of money we have to return
  - Let this amount be the value of the cut
- Consider one award, let's say $c = 0$
  - Create one vertex v for this award
  - Add edges of $+\infty$ capacity from v to vertices in its submatrix
  - Add edge of m capacity from source to v

A. Life game

Solution

- Do for $c = 1$ in the same way
  - Create one vertex v for this award
  - Add edges of $+\infty$ capacity from vertices in its submatrix to v
  - Add edge of m capacity from v to sink

A. Life game

Solution

- Do for $c = 1$ in the same way
  - Create one vertex v for this award
  - Add edges of $+\infty$ capacity from vertices in its submatrix to v
  - Add edge of m capacity from v to sink
- One can see, that if award criterion is not satisfied, then m-valued edge is intersecting a cut

A. Life game

Solution

- Do for $c = 1$ in the same way
  - Create one vertex v for this award
  - Add edges of $+\infty$ capacity from vertices in its submatrix to v
  - Add edge of m capacity from v to sink
- One can see, that if award criterion is not satisfied, then m-valued edge is intersecting a cut
- This will give up to $50^2 \cdot 50\,000$ edges and $50 + 50\,000$ vertices
  - Pretty much for maxflow algorithms

A. Life game

Optimizing

- Make 2D sparse table structure

A. Life game

Optimizing

- Make 2D sparse table structure
  - Add vertex for every r, c, k_r, k_c
    - Submatrix $[r, r + 2^{k_r}) \times [c, c + 2^{k_c})$

A. Life game

Optimizing

- Make 2D sparse table structure
  - Add vertex for every r, c, k_r, k_c
    - Submatrix $[r, r + 2^{k_r}) \times [c, c + 2^{k_c})$
- Don't add edges from every cell to an award
  - Add 4 edges from 4 submatrices of power-of-2 sizes to an award as in 2D sparse table
  - We don't care if they overlap

A. Life game

Optimizing

- Make 2D sparse table structure
  - Add vertex for every r, c, k_r, k_c
    - Submatrix $[r, r + 2^{k_r}) \times [c, c + 2^{k_c})$
- Don't add edges from every cell to an award
  - Add 4 edges from 4 submatrices of power-of-2 sizes to an award as in 2D sparse table
  - We don't care if they overlap
- Add $+\infty$ edges between power-of-2 size submatrices
  - From big one to two times smaller ones

A. Life game

Optimizing

- Make 2 such structures
  - One for outgoing edges
  - Another for incoming edges

A. Life game

Optimizing

- Make 2 such structures
  - One for outgoing edges
  - Another for incoming edges
- As it turns out, this optimization helps to get passed TL

A. Life game

Optimizing

- Make 2 such structures
  - One for outgoing edges
  - Another for incoming edges
- As it turns out, this optimization helps to get passed TL
  - Fast algorithms like Dinitz or Preflow-push algorithms help

A. Life game

Optimizing

- Make 2 such structures
  - One for outgoing edges
  - Another for incoming edges
- As it turns out, this optimization helps to get passed TL
  - Fast algorithms like Dinitz or Preflow-push algorithms help
- Here we have about $2 \cdot 50^2 \cdot \log^2 50$ edges for sparse table
- $5 \cdot 50\,000$ edges for edges between awards and submatrices
- And $2 \cdot 50^2 \cdot \log^2 50 + 50\,000$ vertices

B. String Queries

- You are given a string s
  - Length up to 5 000

B. String Queries

- You are given a string s
  - Length up to 5 000
- You are also given queries
  - Given L and R
  - Find number of different substrings in $s(L, R)$

B. String Queries

Solution

- Solution is similar to number of different colors in subtree of rooted tree problem

B. String Queries

Solution

- Solution is similar to number of different colors in subtree of rooted tree problem
- Build an array of $f_{L,R}$ — number of substrings in $s(L, R)$

B. String Queries

Solution

- Solution is similar to number of different colors in subtree of rooted tree problem
- Build an array of $f_{L,R}$ — number of substrings in $s(L, R)$
- Initialize $f_{i,j} = 0$ for all i and j

B. String Queries

Solution

- Solution is similar to number of different colors in subtree of rooted tree problem
- Build an array of $f_{L,R}$ — number of substrings in $s(L, R)$
- Initialize $f_{i,j} = 0$ for all i and j
- Substring $s(i, j)$ is inside all (x, y) for $x \leqslant i$ and $j \leqslant y$
  - For all $i \leqslant j$ add 1 to all $f_{x,y}$ such that $x \leqslant i$ and $j \leqslant y$

B. String Queries

Solution

- Solution is similar to number of different colors in subtree of rooted tree problem
- Build an array of $f_{L,R}$ — number of substrings in $s(L, R)$
- Initialize $f_{i,j} = 0$ for all i and j
- Substring $s(i, j)$ is inside all (x, y) for $x \leqslant i$ and $j \leqslant y$
    - For all $i \leqslant j$ add 1 to all $f_{x,y}$ such that $x \leqslant i$ and $j \leqslant y$
- Get all substrings equal to some t

B. String Queries

Solution

- Solution is similar to number of different colors in subtree of rooted tree problem
- Build an array of $f_{L,R}$ — number of substrings in $s(L, R)$
- Initialize $f_{i,j} = 0$ for all i and j
- Substring $s(i, j)$ is inside all (x, y) for $x \leqslant i$ and $j \leqslant y$
  - For all $i \leqslant j$ add 1 to all $f_{x,y}$ such that $x \leqslant i$ and $j \leqslant y$
- Get all substrings equal to some t
  - $t = s(i_1, j_1) = s(i_2, j_2) = \ldots = s(i_k, j_k)$
    - $i_1 < i_2 < \cdots < i_k$
    - $j_1 - i_1 = j_2 - i_2 = \cdots = j_k - i_k$

B. String Queries

Solution

- Solution is similar to number of different colors in subtree of rooted tree problem
- Build an array of $f_{L,R}$ — number of substrings in $s(L, R)$
- Initialize $f_{i,j} = 0$ for all i and j
- Substring $s(i, j)$ is inside all (x, y) for $x \leqslant i$ and $j \leqslant y$
  - For all $i \leqslant j$ add 1 to all $f_{x,y}$ such that $x \leqslant i$ and $j \leqslant y$
- Get all substrings equal to some t
  - $t = s(i_1, j_1) = s(i_2, j_2) = \ldots = s(i_k, j_k)$
    - $i_1 < i_2 < \cdots < i_k$
    - $j_1 - i_1 = j_2 - i_2 = \cdots = j_k - i_k$
  - Subtract 1 of all $f_{x,y}$ such that $x \leqslant i_e$ and $j_{e+1} \leqslant y$
    - for $1 \leqslant e < k$

B. String Queries

Solution

- Solution is similar to number of different colors in subtree of rooted tree problem
- Build an array of $f_{L,R}$ — number of substrings in $s(L, R)$
- Initialize $f_{i,j} = 0$ for all i and j
- Substring $s(i, j)$ is inside all (x, y) for $x \leqslant i$ and $j \leqslant y$
    - For all $i \leqslant j$ add 1 to all $f_{x,y}$ such that $x \leqslant i$ and $j \leqslant y$
- Get all substrings equal to some t
    - $t = s(i_1, j_1) = s(i_2, j_2) = \ldots = s(i_k, j_k)$
        - $i_1 < i_2 < \cdots < i_k$
        - $j_1 - i_1 = j_2 - i_2 = \cdots = j_k - i_k$
    - Subtract 1 of all $f_{x,y}$ such that $x \leqslant i_e$ and $j_{e+1} \leqslant y$
        - for $1 \leqslant e < k$
- Then just answer all queries outputting $f_{L,R}$

B. String Queries

Why will it work?

- Consider some $s(L, R)$

B. String Queries

Why will it work?

- Consider some $s(L, R)$
- Suppose there are w copies of t in $s(L, R)$

B. String Queries

Why will it work?

- Consider some $s(L, R)$
- Suppose there are w copies of t in $s(L, R)$
- We added 1 for each of the copy, so it's $+w$

B. String Queries

Why will it work?

- Consider some $s(L, R)$
- Suppose there are w copies of t in $s(L, R)$
- We added 1 for each of the copy, so it's $+w$
- For each neighbouring copies we subtracted 1, so it's $-(w-1)$

B. String Queries

Why will it work?

- Consider some $s(L, R)$
- Suppose there are w copies of t in $s(L, R)$
- We added 1 for each of the copy, so it's $+w$
- For each neighbouring copies we subtracted 1, so it's $-(w-1)$
- So it's $w-(w-1) = +1$ for each substring that is inside

B. String Queries

Implementation

- Calculate $LCP(i,j)$
  - Longest common prefix of $s(i,|s|)$ and $s(j,|s|)$

B. String Queries

Implementation

- Calculate $LCP(i,j)$
  - Longest common prefix of $s(i,|s|)$ and $s(j,|s|)$
- Fix i and iterate over j

B. String Queries

Implementation

- Calculate $LCP(i, j)$
  - Longest common prefix of $s(i, |s|)$ and $s(j, |s|)$
- Fix i and iterate over j
- Find equal substrings to those, which start at i

B. String Queries

Implementation

- Calculate $LCP(i, j)$
  - Longest common prefix of $s(i, |s|)$ and $s(j, |s|)$
- Fix i and iterate over j
- Find equal substrings to those, which start at i
- Maintain d the longest substring found so far

B. String Queries

Implementation

- Calculate $LCP(i, j)$
  - Longest common prefix of $s(i, |s|)$ and $s(j, |s|)$
- Fix i and iterate over j
- Find equal substrings to those, which start at i
- Maintain d the longest substring found so far
- If $LCP(i, j) > d$
  - We found substrings with lengths $d + 1 \ldots LCP(i, j)$

B. String Queries

Implementation

- Calculate $LCP(i, j)$
  - Longest common prefix of $s(i, |s|)$ and $s(j, |s|)$
- Fix i and iterate over j
- Find equal substrings to those, which start at i
- Maintain d the longest substring found so far
- If $LCP(i, j) > d$
  - We found substrings with lengths $d + 1 \dots LCP(i, j)$
    - For each substring of length $d + 1 \leqslant g \leqslant LCP(i, j)$ subtract 1

B. String Queries

Implementation

- Calculate $LCP(i, j)$
  - Longest common prefix of $s(i, |s|)$ and $s(j, |s|)$
- Fix i and iterate over j
- Find equal substrings to those, which start at i
- Maintain d the longest substring found so far
- If $LCP(i, j) > d$
  - We found substrings with lengths $d + 1 \ldots LCP(i, j)$
    - For each substring of length $d + 1 \leqslant g \leqslant LCP(i, j)$ subtract 1
    - Subtract 1 from $f_{x,y}$ such that $x \leqslant i$ and $j + g - 1 \leqslant y$

B. String Queries

Implementation

- Calculate $LCP(i, j)$
  - Longest common prefix of $s(i, |s|)$ and $s(j, |s|)$
- Fix i and iterate over j
- Find equal substrings to those, which start at i
- Maintain d the longest substring found so far
- If $LCP(i, j) > d$
  - We found substrings with lengths $d + 1 \ldots LCP(i, j)$
    - For each substring of length $d + 1 \leqslant g \leqslant LCP(i, j)$ subtract 1
    - Subtract 1 from $f_{x,y}$ such that $x \leqslant i$ and $j + g - 1 \leqslant y$
- Partial sums will help iterating over g

B. String Queries

Implementation

- Calculate $LCP(i, j)$
  - Longest common prefix of $s(i, |s|)$ and $s(j, |s|)$
- Fix i and iterate over j
- Find equal substrings to those, which start at i
- Maintain d the longest substring found so far
- If $LCP(i, j) > d$
  - We found substrings with lengths $d + 1 \ldots LCP(i, j)$
    - For each substring of length $d + 1 \leqslant g \leqslant LCP(i, j)$ subtract 1
    - Subtract 1 from $f_{x,y}$ such that $x \leqslant i$ and $j + g - 1 \leqslant y$
- Partial sums will help iterating over g
- And 2D Partial sums will help adding $f_{x,y}$ for $x \leqslant i$ and $j \leqslant y$

B. String Queries

Implementation

- Calculate $LCP(i, j)$
  - Longest common prefix of $s(i, |s|)$ and $s(j, |s|)$
- Fix i and iterate over j
- Find equal substrings to those, which start at i
- Maintain d the longest substring found so far
- If $LCP(i, j) > d$
  - We found substrings with lengths $d + 1 \ldots LCP(i, j)$
    - For each substring of length $d + 1 \leqslant g \leqslant LCP(i, j)$ subtract 1
    - Subtract 1 from $f_{x,y}$ such that $x \leqslant i$ and $j + g - 1 \leqslant y$
- Partial sums will help iterating over g
- And 2D Partial sums will help adding $f_{x,y}$ for $x \leqslant i$ and $j \leqslant y$
- Solution time and memory complexity is $O(|s|^2 + Q)$
  - Q — the number of queries

B. String Queries

Suffix tree or automaton solution

- For every suffix build suffix data structure in linear time

B. String Queries

Suffix tree or automaton solution

- For every suffix build suffix data structure in linear time
- Append single letter
  - Learn how the number of substrings changed

B. String Queries

Suffix tree or automaton solution

- For every suffix build suffix data structure in linear time
- Append single letter
  - Learn how the number of substrings changed
- For automaton it's += `len[last] - len[sufLink[last]]`

B. String Queries

Suffix tree or automaton solution

- For every suffix build suffix data structure in linear time
- Append single letter
  - Learn how the number of substrings changed
- For automaton it's += `len[last] - len[sufLink[last]]`
- Still $O(|s|^2)$ time and $O(|s|^2)$ memory solution

C. Coprimes

- Given n up to 28

C. Coprimes

- Given n up to 28
- Find number of different permutations
  - Of length n
  - Every two consecutive elements are coprime

C. Coprimes

Solution

- For each x we are interested in the subset of $[1 \dots n]$, so that x is coprime to them

C. Coprimes

Solution

- For each x we are interested in the subset of $[1 \ldots n]$, so that x is coprime to them
- Build equivalence classes on that criteria

C. Coprimes

Solution

- For each x we are interested in the subset of $[1 \ldots n]$, so that x is coprime to them
- Build equivalence classes on that criteria
- Based on these equivalence classes count number of different multisets of classes
    - There are $1\,728\,000$ of those

C. Coprimes

Solution

- For each x we are interested in the subset of $[1 \ldots n]$, so that x is coprime to them
- Build equivalence classes on that criteria
- Based on these equivalence classes count number of different multisets of classes
  - There are 1 728 000 of those
- Make dynamic programming

D. Colored Balls

- You are given n white balls

D. Colored Balls

- You are given n white balls
- In one turn you choose (l, r), so that $1 \leqslant l \leqslant r \leqslant n$ equiprobably

D. Colored Balls

- You are given n white balls
- In one turn you choose (l, r), so that $1 \leqslant l \leqslant r \leqslant n$ equiprobably
- Color each ball x, such that $l \leqslant x \leqslant r$, to black

D. Colored Balls

- You are given n white balls
- In one turn you choose (l, r), so that $1 \leqslant l \leqslant r \leqslant n$ equiprobably
- Color each ball x, such that $l \leqslant x \leqslant r$, to black
- What is the expected number of turns, so that every ball is colored?

D. Colored Balls

Solution

- Answer is $\sum_{i=0}^{+\infty} p(i)$
  - $p(x)$ is the probability, that in x moves there exists a white ball

D. Colored Balls

Solution

- Answer is $\sum_{i=0}^{+\infty} p(i)$
  - $p(x)$ is the probability, that in x moves there exists a white ball
- How to calculate $p(x)$?

D. Colored Balls

Solution

- Answer is $\sum_{i=0}^{+\infty} p(i)$
  - $p(x)$ is the probability, that in x moves there exists a white ball
- How to calculate $p(x)$?
  - d_A is the probability to leave A white in x moves

D. Colored Balls

Solution

- Answer is $\sum_{i=0}^{+\infty} p(i)$
  - $p(x)$ is the probability, that in x moves there exists a white ball
- How to calculate $p(x)$?
  - d_A is the probability to leave A white in x moves
    - Choosing these intervals will color some subset of $\overline{A}$

D. Colored Balls

Solution

- Answer is $\sum_{i=0}^{+\infty} p(i)$
  - $p(x)$ is the probability, that in x moves there exists a white ball
- How to calculate $p(x)$?
  - d_A is the probability to leave A white in x moves
    - Choosing these intervals will color some subset of $\overline{A}$
  - Use inclusion-exclusion formula

D. Colored Balls

Solution

- Answer is $\sum_{i=0}^{+\infty} p(i)$
  - $p(x)$ is the probability, that in x moves there exists a white ball
- How to calculate $p(x)$?
  - d_A is the probability to leave A white in x moves
    - Choosing these intervals will color some subset of $\overline{A}$
  - Use inclusion-exclusion formula
    - Probability to color all balls in x moves $\sum_A d_A \cdot (-1)^{|A|}$

D. Colored Balls

Solution

- Answer is $\sum_{i=0}^{+\infty} p(i)$
  - $p(x)$ is the probability, that in x moves there exists a white ball
- How to calculate $p(x)$?
  - d_A is the probability to leave A white in x moves
    - Choosing these intervals will color some subset of $\overline{A}$
  - Use inclusion-exclusion formula
    - Probability to color all balls in x moves $\sum_A d_A \cdot (-1)^{|A|}$
  - c_A is the number of intervals that cover only balls from $\overline{A}$

D. Colored Balls

Solution

- Answer is $\sum_{i=0}^{+\infty} p(i)$
  - $p(x)$ is the probability, that in x moves there exists a white ball
- How to calculate $p(x)$?
  - d_A is the probability to leave A white in x moves
    - Choosing these intervals will color some subset of $\overline{A}$
  - Use inclusion-exclusion formula
    - Probability to color all balls in x moves $\sum_{A} d_A \cdot (-1)^{|A|}$
  - c_A is the number of intervals that cover only balls from $\overline{A}$
  - Then $d_A = \left(\frac{c_A}{\binom{n+1}{2}}\right)^x$

D. Colored Balls

Solution

- Answer is $\sum_{i=0}^{+\infty} p(i)$
  - $p(x)$ is the probability, that in x moves there exists a white ball
- How to calculate $p(x)$?
  - d_A is the probability to leave A white in x moves
    - Choosing these intervals will color some subset of $\overline{A}$
  - Use inclusion-exclusion formula
    - Probability to color all balls in x moves $\sum_A d_A \cdot (-1)^{|A|}$
  - c_A is the number of intervals that cover only balls from $\overline{A}$
  - Then $d_A = \left(\frac{c_A}{\binom{n+1}{2}}\right)^x$
- So answer is $\sum_{i=0}^{+\infty} \left(1 - \sum_A (-1)^{|A|} \left(\frac{c_A}{\binom{n+1}{2}}\right)^i\right)$

D. Colored Balls

Solution

- We know that $d_\varnothing = 1$, so:
  - Answer is: $\sum_{i=0}^{+\infty} \sum_{A \neq \varnothing} (-1)^{|A|} \left(\frac{c_A}{\binom{n+1}{2}} \right)^i$
- Change the order $\sum_{A \neq \varnothing} (-1)^{|A|} \sum_{i=0}^{+\infty} \left(\frac{c_A}{\binom{n+1}{2}} \right)^i$

D. Colored Balls

Solution

- We know that $d_{\varnothing} = 1$, so:
  - Answer is: $\sum_{i=0}^{+\infty} \sum_{A \neq \varnothing} (-1)^{|A|} \left(\frac{c_A}{\binom{n+1}{2}} \right)^i$
- Change the order $\sum_{A \neq \varnothing} (-1)^{|A|} \sum_{i=0}^{+\infty} \left(\frac{c_A}{\binom{n+1}{2}} \right)^i$
  - It's 2^n geometric series sums

D. Colored Balls

Solution

- We know that $d_\varnothing = 1$, so:
  - Answer is: $\sum_{i=0}^{+\infty} \sum_{A \neq \varnothing} (-1)^{|A|} \left(\frac{c_A}{\binom{n+1}{2}} \right)^i$
- Change the order $\sum_{A \neq \varnothing} (-1)^{|A|} \sum_{i=0}^{+\infty} \left(\frac{c_A}{\binom{n+1}{2}} \right)^i$
  - It's 2^n geometric series sums
- For every c_A and $(|A| \bmod 2)$ calculate the number of such A

D. Colored Balls

Solution

- We know that $d_{\varnothing} = 1$, so:
  - Answer is: $\sum_{i=0}^{+\infty} \sum_{A \neq \varnothing} (-1)^{|A|} \left(\frac{c_A}{\binom{n+1}{2}} \right)^i$
- Change the order $\sum_{A \neq \varnothing} (-1)^{|A|} \sum_{i=0}^{+\infty} \left(\frac{c_A}{\binom{n+1}{2}} \right)^i$
  - It's 2^n geometric series sums
- For every c_A and ($|A|$ mod 2) calculate the number of such A
- As an exercise come up with dynamic programming polynomial solution to do that

E. Another Tree Problem

- You are given a tree

E. Another Tree Problem

- You are given a tree
  - Number of vertices is at most 50 000

E. Another Tree Problem

- You are given a tree
  - Number of vertices is at most 50 000
- Calculate for each vertex v:

E. Another Tree Problem

- You are given a tree
  - Number of vertices is at most 50 000
- Calculate for each vertex v:
  - $\sum\limits_{u} d_{v,u}^{k}$

E. Another Tree Problem

- You are given a tree
  - Number of vertices is at most 50 000
- Calculate for each vertex v:
  - $\sum_u d_{v,u}^k$
  - $d_{x,y}$ is the distance from v to u

E. Another Tree Problem

- You are given a tree
  - Number of vertices is at most 50 000
- Calculate for each vertex v:
  - $\sum_u d_{v,u}^k$
  - $d_{x,y}$ is the distance from v to u
  - k is up to 50

E. Another Tree Problem

Solution

- Formula using Stirling numbers of second kind:
  - $d^k = \sum_{i=0}^{k} S(k,i) \cdot d \cdot (d-1) \cdot \dots \cdot (d-i+1)$
  - $d^k = \sum_{i=0}^{k} S(k,i) \cdot \binom{d}{i} \cdot i!$
  - $S(k,i)$ is the Stirling number of second kind
    - The number of ways to color k element set into i colors

E. Another Tree Problem

Solution

- Formula using Stirling numbers of second kind:
  - $d^k = \sum_{i=0}^{k} S(k,i) \cdot d \cdot (d-1) \cdot \dots \cdot (d-i+1)$
  - $d^k = \sum_{i=0}^{k} S(k,i) \cdot \binom{d}{i} \cdot i!$
  - $S(k,i)$ is the Stirling number of second kind
    - The number of ways to color k element set into i colors
- So for every vertex v calculate array a:
  - $a_i = \sum_{u} \binom{d_{v,u}}{i}$

E. Another Tree Problem

Solution

- Formula using Stirling numbers of second kind:
  - $d^k = \sum_{i=0}^{k} S(k,i) \cdot d \cdot (d-1) \cdot \dots \cdot (d-i+1)$
  - $d^k = \sum_{i=0}^{k} S(k,i) \cdot \binom{d}{i} \cdot i!$
  - $S(k,i)$ is the Stirling number of second kind
    - The number of ways to color k element set into i colors
- So for every vertex v calculate array a:
  - $a_i = \sum_{u} \binom{d_{v,u}}{i}$
- To add one edge, one has to increase every $d_{v,u}$ by one
  - $\binom{d+1}{i} = \binom{d}{i} + \binom{d}{i-1}$
  - $a_i^{\mathtt{new}} = a_i + a_{i-1}$

E. Another Tree Problem

Solution

- To calculate *a* first make tree rooted
  - Sum up all $\binom{d}{i}$ over all descendants first
  - Then sum up all $\binom{d}{i}$ for not descendants by second DFS

E. Another Tree Problem

Solution

- To calculate a first make tree rooted
  - Sum up all $\binom{d}{i}$ over all descendants first
  - Then sum up all $\binom{d}{i}$ for not descendants by second DFS
- Calculate a for all vertices in $O(nk)$ time

E. Another Tree Problem

Solution

- To calculate a first make tree rooted
  - Sum up all $\binom{d}{i}$ over all descendants first
  - Then sum up all $\binom{d}{i}$ for not descendants by second DFS
- Calculate a for all vertices in $O(nk)$ time
- Calculate $S(i, j)$ and $i!$

E. Another Tree Problem

Solution

- To calculate a first make tree rooted
  - Sum up all $\binom{d}{i}$ over all descendants first
  - Then sum up all $\binom{d}{i}$ for not descendants by second DFS
- Calculate a for all vertices in $O(nk)$ time
- Calculate $S(i, j)$ and $i!$
- Use formula to get answer for every vertex

F. String and Queries-2

- You are given a string s of length up to 10^5

F. String and Queries-2

- You are given a string s of length up to 10^5
- String consists of first 20 letters of alphabet

F. String and Queries-2

- You are given a string s of length up to 10^5
- String consists of first 20 letters of alphabet
- Answer queries:
  - Given $c_1, c_2, \dots c_k$ — letters
    - $k \leqslant 5$

F. String and Queries-2

- You are given a string s of length up to 10^5
- String consists of first 20 letters of alphabet
- Answer queries:
  - Given $c_1, c_2, \ldots c_k$ — letters
    - $k \leqslant 5$
  - Find number of pairs (i, j), so that $s(i, j)$ contains even number of each of these k letters

F. String and Queries-2

Solution

- For each prefix $0 \leqslant i \leqslant |s|$ find subset p_i:
  - which letters enter odd number of times

F. String and Queries-2

Solution

- For each prefix $0 \leqslant i \leqslant |s|$ find subset p_i:
    - which letters enter odd number of times
- For substring $s(i+1, j)$ we have to calculate $p_i \oplus p_j$

F. String and Queries-2

Solution

- For each prefix $0 \leqslant i \leqslant |s|$ find subset p_i:
  - which letters enter odd number of times
- For substring $s(i+1, j)$ we have to calculate $p_i \oplus p_j$
- Calculate f_A — number of i such that $p_i = A$

F. String and Queries-2

Solution

- For each prefix $0 \leqslant i \leqslant |s|$ find subset p_i:
  - which letters enter odd number of times
- For substring $s(i+1, j)$ we have to calculate $p_i \oplus p_j$
- Calculate f_A — number of i such that $p_i = A$
- Calculate $g_A = \sum\limits_{A \subset B} f_B$
  - It's just partial sums on $2 \times 2 \times \ldots \times 2$ array
  - Calculated in $O(2^{|\Sigma|}|\Sigma|)$

F. String and Queries-2

Solution

- To answer the queries:
  - p_i and p_j have to have equal parity for letters in query

F. String and Queries-2

Solution

- To answer the queries:
  - p_i and p_j have to have equal parity for letters in query
  - For each X of 2^k parities of given k letters get g_A
    - A contains only letters from query
    - Calculate $d_X = g_A$

F. String and Queries-2

Solution

- To answer the queries:
  - p_i and p_j have to have equal parity for letters in query
  - For each X of 2^k parities of given k letters get g_A
    - A contains only letters from query
    - Calculate $d_X = g_A$
  - Use inclusion-exclusion formula
  - `for` $X = (2^k - 1) \ldots 0$:

    `for` $Y \supset X$:

    $d_X := d_X - d_Y$
  - d_X is number of p_i, so that given letters' parity is X and the other letters' parity is either odd or even

F. String and Queries-2

Solution

- To answer the queries:
  - p_i and p_j have to have equal parity for letters in query
  - For each X of 2^k parities of given k letters get g_A
    - A contains only letters from query
    - Calculate $d_X = g_A$
  - Use inclusion-exclusion formula
  - for $X = (2^k - 1) \dots 0$:

    for $Y \supset X$:

    $d_X := d_X - d_Y$
  - d_X is number of p_i, so that given letters' parity is X and the other letters' parity is either odd or even
  - Answer is $\sum\limits_{X} \frac{d_X(d_X-1)}{2}$

Problem G. LCM

- Given n up to 10^9
- Find positive a and b such that
  1. $a + b = n$
  2. $\mathrm{lcm}(a, b)$ is maximum possible

Problem G. LCM

Solution

- If $x > y$ and $d > 0$, then $xy \geqslant (x + d)(y - d)$

Problem G. LCM

Solution

- If $x > y$ and $d > 0$, then $xy \geqslant (x+d)(y-d)$
- p equal to smallest prime greater than $\frac{n}{2}$
  - It's coprime to $n-p$, since $n-p < p$

Problem G. LCM

Solution

- If $x > y$ and $d > 0$, then $xy \geqslant (x + d)(y - d)$
- p equal to smallest prime greater than $\frac{n}{2}$
  - It's coprime to $n - p$, since $n - p < p$
  - So answer is not less than $(n - p)p$

Problem G. LCM

Solution

- If $x > y$ and $d > 0$, then $xy \geqslant (x+d)(y-d)$
- p equal to smallest prime greater than $\frac{n}{2}$
  - It's coprime to $n-p$, since $n-p < p$
  - So answer is not less than $(n-p)p$
  - You don't have to look to $x > p$

Problem G. LCM

Solution

- If $x > y$ and $d > 0$, then $xy \geqslant (x+d)(y-d)$
- p equal to smallest prime greater than $\frac{n}{2}$
  - It's coprime to $n - p$, since $n - p < p$
  - So answer is not less than $(n-p)p$
  - You don't have to look to $x > p$
  - Gap between prime numbers is small enough to try every $\frac{n}{2} < x \leqslant p$

H. Erase the String

- Given a string of length not greater than 16
- In one move you can erase any subsequence, that is palindrome
- Find minimum number of moves to erase all string

H. Erase the String

Solution

- For every of $2^n - 1$ subsequences calculate if it's palindrome

H. Erase the String

Solution

- For every of $2^n - 1$ subsequences calculate if it's palindrome
  - Let P be the set of all palindrome subsequences

H. Erase the String

Solution

- For every of $2^n - 1$ subsequences calculate if it's palindrome
  - Let P be the set of all palindrome subsequences
- $f[A]$ — minimum number of moves to erase subset A

H. Erase the String

Solution

- For every of $2^n - 1$ subsequences calculate if it's palindrome
  - Let P be the set of all palindrome subsequences
- $f[A]$ — minimum number of moves to erase subset A
- $f[A] = \min_{B \in P \wedge B \subset A} f[B] + 1$

H. Erase the String

Solution

- For every of $2^n - 1$ subsequences calculate if it's palindrome
  - Let P be the set of all palindrome subsequences
- $f[A]$ — minimum number of moves to erase subset A
- $f[A] = \min_{B \in P \wedge B \subset A} f[B] + 1$
- Calculated in $O(3^n)$

I. Thickness

- You are given triangles
- For every k find the area of a plane covered by exactly k triangles

I. Thickness

Solution

- Intersect all pairs of sides of all triangles

I. Thickness

Solution

- Intersect all pairs of sides of all triangles
- Get all x-coordinates of all intersection and all vertices
  - $x_1 < x_2 < \cdots < x_k$ be those coordinates

I. Thickness

Solution

- Intersect all pairs of sides of all triangles
- Get all x-coordinates of all intersection and all vertices
  - $x_1 < x_2 < \cdots < x_k$ be those coordinates
- Consider the part of plane with points (x, y) such that $x_i < x < x_{i+1}$

I. Thickness

Solution

- Intersect all pairs of sides of all triangles
- Get all x-coordinates of all intersection and all vertices
  - $x_1 < x_2 < \cdots < x_k$ be those coordinates
- Consider the part of plane with points (x, y) such that $x_i < x < x_{i+1}$
- Intersection of this part of plane with every triangle is either empty set or trapezoid

I. Thickness

Solution

- Intersect all pairs of sides of all triangles
- Get all x-coordinates of all intersection and all vertices
  - $x_1 < x_2 < \cdots < x_k$ be those coordinates
- Consider the part of plane with points (x, y) such that $x_i < x < x_{i+1}$
- Intersection of this part of plane with every triangle is either empty set or trapezoid
- No two non-vertical trapezoid sides intersect

I. Thickness

Solution

- Intersect every side of triangle with this part of the plane

I. Thickness

Solution

- Intersect every side of triangle with this part of the plane
- Get middle point of intersection

I. Thickness

Solution

- Intersect every side of triangle with this part of the plane
- Get middle point of intersection
- Sort all segments by y-coordinate of this middle point

I. Thickness

Solution

- Intersect every side of triangle with this part of the plane
- Get middle point of intersection
- Sort all segments by y-coordinate of this middle point
- Do another sweepline iterating over segments

I. Thickness

Solution

- Intersect every side of triangle with this part of the plane
- Get middle point of intersection
- Sort all segments by y-coordinate of this middle point
- Do another sweepline iterating over segments
- Each segment is either the start of a triangle or the end
    - Keep track of k — number of triangles covering

I. Thickness

Solution

- Intersect every side of triangle with this part of the plane
- Get middle point of intersection
- Sort all segments by y-coordinate of this middle point
- Do another sweepline iterating over segments
- Each segment is either the start of a triangle or the end
    - Keep track of k — number of triangles covering
- Calculate trapezoid area and add it to corresponding answer

J. GCD

- You are given n natural numbers ($n \leqslant 50\,000$)

J. GCD

- You are given n natural numbers ($n \leqslant 50\,000$)
  - Each number is not greater than $50\,000$

J. GCD

- You are given n natural numbers ($n \leqslant 50\,000$)
  - Each number is not greater than $50\,000$
- You are also given queries:
  - Each consists of L and R

J. GCD

- You are given n natural numbers ($n \leqslant 50\,000$)
  - Each number is not greater than $50\,000$
- You are also given queries:
  - Each consists of L and R
  - Find pair (i, j) such that $i \neq j$ and $L \leqslant i, j \leqslant R$

J. GCD

- You are given n natural numbers ($n \leqslant 50\,000$)
  - Each number is not greater than $50\,000$
- You are also given queries:
  - Each consists of L and R
  - Find pair (i, j) such that $i \neq j$ and $L \leqslant i, j \leqslant R$
    - And $\gcd(a_i, a_j)$ is maximum possible

J. GCD

Solution

- Let's answer all queries in order of increasing R

J. GCD

Solution

- Let's answer all queries in order of increasing R
  - Consider gcd is equal to v

J. GCD

Solution

- Let's answer all queries in order of increasing R
  - Consider gcd is equal to v
  - Consider rightmost two positions $i < j \leqslant R$:
    - so that $v \mid a_i$ and $v \mid a_j$

J. GCD

Solution

- Let's answer all queries in order of increasing R
  - Consider gcd is equal to v
  - Consider rightmost two positions $i < j \leqslant R$:
    - so that $v \mid a_i$ and $v \mid a_j$
  - For every $L \leqslant i$ answer is at least v

J. GCD

Solution

- Let's answer all queries in order of increasing R
  - Consider gcd is equal to v
  - Consider rightmost two positions $i < j \leqslant R$:
    - so that $v \mid a_i$ and $v \mid a_j$
  - For every $L \leqslant i$ answer is at least v
  - For every $1 \leqslant i \leqslant R$ store the maximum possible divisor of a_i

J. GCD

Solution

- Let's answer all queries in order of increasing R
  - Consider gcd is equal to v
  - Consider rightmost two positions $i < j \leqslant R$:
    - so that $v \mid a_i$ and $v \mid a_j$
  - For every $L \leqslant i$ answer is at least v
  - For every $1 \leqslant i \leqslant R$ store the maximum possible divisor of a_i
    - Such v, so that there is $j > i$ and $j \leqslant R$
    - And $v \mid a_j$

J. GCD

Solution

- Let's answer all queries in order of increasing R
  - Consider gcd is equal to v
  - Consider rightmost two positions $i < j \leqslant R$:
    - so that $v \mid a_i$ and $v \mid a_j$
  - For every $L \leqslant i$ answer is at least v
  - For every $1 \leqslant i \leqslant R$ store the maximum possible divisor of a_i
    - Such v, so that there is $j > i$ and $j \leqslant R$
    - And $v \mid a_j$
  - Keep track of interval tree or binary indexed tree, say t

J. GCD

Solution

- Let's answer all queries in order of increasing R
  - Consider gcd is equal to v
  - Consider rightmost two positions $i < j \leqslant R$:
    - so that $v \mid a_i$ and $v \mid a_j$
  - For every $L \leqslant i$ answer is at least v
  - For every $1 \leqslant i \leqslant R$ store the maximum possible divisor of a_i
    - Such v, so that there is $j > i$ and $j \leqslant R$
    - And $v \mid a_j$
  - Keep track of interval tree or binary indexed tree, say t
  - Keep track of last number, that is divisible by each v

J. GCD

Solution

- Let's answer all queries in order of increasing R
  - Consider gcd is equal to v
  - Consider rightmost two positions $i < j \leqslant R$:
    - so that $v \mid a_i$ and $v \mid a_j$
  - For every $L \leqslant i$ answer is at least v
  - For every $1 \leqslant i \leqslant R$ store the maximum possible divisor of a_i
    - Such v, so that there is $j > i$ and $j \leqslant R$
    - And $v \mid a_j$
  - Keep track of interval tree or binary indexed tree, say t
  - Keep track of last number, that is divisible by each v
  - To increase R by one, iterate over all $v | a_R$
    - Make `t[last[v]] := max(t[last[v]], v)`
    - Update `last[v] := R`

J. GCD

Solution

- Let's answer all queries in order of increasing R
  - Consider gcd is equal to v
  - Consider rightmost two positions $i < j \leqslant R$:
    - so that $v \mid a_i$ and $v \mid a_j$
  - For every $L \leqslant i$ answer is at least v
  - For every $1 \leqslant i \leqslant R$ store the maximum possible divisor of a_i
    - Such v, so that there is $j > i$ and $j \leqslant R$
    - And $v \mid a_j$
  - Keep track of interval tree or binary indexed tree, say t
  - Keep track of last number, that is divisible by each v
  - To increase R by one, iterate over all $v|a_R$
    - Make `t[last[v]] := max(t[last[v]], v)`
    - Update `last[v] := R`
  - Answer for query (L, R) is maximum in `t[L..R]`

K. Points on a Plane

- You are given sequence of n points
- n is at most $5 \cdot 10^5$

K. Points on a Plane

- You are given sequence of n points
- n is at most $5 \cdot 10^5$
- Points are generated pseudorandomly
  - With coordinates up to n

K. Points on a Plane

- You are given sequence of n points
- n is at most $5 \cdot 10^5$
- Points are generated pseudorandomly
  - With coordinates up to n
- After each point answer, what is the distance between closest two points?

K. Points on a Plane

Solution

- Maintain sorted by x-coordinate array of all already added points

K. Points on a Plane

Solution

- Maintain sorted by x-coordinate array of all already added points
- When new (x_0, y_0) point added:
    - Let d be the current answer

K. Points on a Plane

Solution

- Maintain sorted by x-coordinate array of all already added points
- When new (x_0, y_0) point added:
  - Let d be the current answer
  - Check all points (x, y) such that $|x - x_0| < d$
    - Other points are not closer than d

K. Points on a Plane

Solution

- Maintain sorted by x-coordinate array of all already added points
- When new (x_0, y_0) point added:
  - Let d be the current answer
  - Check all points (x, y) such that $|x - x_0| < d$
    - Other points are not closer than d
  - Update answer by distance to these points

K. Points on a Plane

Solution

- Intuitively the runtime is explained like this:
  - Consider we added p points to our set

K. Points on a Plane

Solution

- Intuitively the runtime is explained like this:
  - Consider we added p points to our set
  - Let's make $r \times r$ grid of $[1 \ldots n] \times [1 \ldots n]$ square

K. Points on a Plane

Solution

- Intuitively the runtime is explained like this:
  - Consider we added p points to our set
  - Let's make $r \times r$ grid of $[1 \ldots n] \times [1 \ldots n]$ square
    - Choose r such that the probability of two points locating in the same cell is at least $\frac{1}{2}$

K. Points on a Plane

Solution

- Intuitively the runtime is explained like this:
  - Consider we added p points to our set
  - Let's make $r \times r$ grid of $[1 \ldots n] \times [1 \ldots n]$ square
    - Choose r such that the probability of two points locating in the same cell is at least $\frac{1}{2}$
    - Birthday paradox says that number of cells can be quadratic of p, so $r \approx p$

K. Points on a Plane

Solution

- Intuitively the runtime is explained like this:
  - Consider we added p points to our set
  - Let's make $r \times r$ grid of $[1 \ldots n] \times [1 \ldots n]$ square
    - Choose r such that the probability of two points locating in the same cell is at least $\frac{1}{2}$
    - Birthday paradox says that number of cells can be quadratic of p, so $r \approx p$
  - So expected number of points in $x_0 - d < x < x_0 + d$ is $\frac{2pd}{n}$

K. Points on a Plane

Solution

- Intuitively the runtime is explained like this:
  - Consider we added p points to our set
  - Let's make $r \times r$ grid of $[1 \dots n] \times [1 \dots n]$ square
    - Choose r such that the probability of two points locating in the same cell is at least $\frac{1}{2}$
    - Birthday paradox says that number of cells can be quadratic of p, so $r \approx p$
  - So expected number of points in $x_0 - d < x < x_0 + d$ is $\frac{2pd}{n}$
  - $d \approx \frac{n}{r} \approx \frac{n}{p}$
    - Expected number of points is $\frac{2pd}{n} \approx 2\frac{\frac{n}{p}p}{n} = 2$

K. Points on a Plane

Solution

- Intuitively the runtime is explained like this:
  - Consider we added p points to our set
  - Let's make $r \times r$ grid of $[1 \ldots n] \times [1 \ldots n]$ square
    - Choose r such that the probability of two points locating in the same cell is at least $\frac{1}{2}$
    - Birthday paradox says that number of cells can be quadratic of p, so $r \approx p$
  - So expected number of points in $x_0 - d < x < x_0 + d$ is $\frac{2pd}{n}$
  - $d \approx \frac{n}{r} \approx \frac{n}{p}$
    - Expected number of points is $\frac{2pd}{n} \approx 2\frac{\frac{n}{p}p}{n} = 2$
  - So summing up over all p, we get $O(n)$ runtime