NCPC 2024

Presentation of solutions

October 5, 2024

Problems prepared by

- Hlíf Arnbjargardóttir (Atli's Mom)
- Arnar Bjarni Arnarson (Reykjavik University)
- Pål Grønås Drange (University of Bergen)
- Atli Fannar Franklín (University of Iceland)
- Nils Gustafsson (KTH Royal Institute of Technology)
- Björn Martinsson (KTH Royal Institute of Technology)
- Bergur Snorrason (University of Iceland)

C — Composed Rhythms

Problem

Given an integer $n \leq 10^6$, write it as a sum of twos and threes.

Solution

C — Composed Rhythms

Problem

Given an integer $n \leq 10^6$, write it as a sum of twos and threes.

Solution

1. Most greedy solutions succeed.

C — Composed Rhythms

Problem

Given an integer $n \leq 10^6$, write it as a sum of twos and threes.

Solution

1. Most greedy solutions succeed.
2. Subtract twos until sum is a multiple of threes, then print threes, or vice versa.

C — Composed Rhythms

Problem

Given an integer $n \leq 10^6$, write it as a sum of twos and threes.

Solution

1. Most greedy solutions succeed.
2. Subtract twos until sum is a multiple of threes, then print threes, or vice versa.
3. Solve $2x + 3y = n$ also works, but more involved since $x, y \geq 0$.

C — Composed Rhythms

Problem

Given an integer $n \leq 10^6$, write it as a sum of twos and threes.

Solution

1. Most greedy solutions succeed.
2. Subtract twos until sum is a multiple of threes, then print threes, or vice versa.
3. Solve $2x + 3y = n$ also works, but more involved since $x, y \geq 0$.

Statistics at 4-hour mark: 394 submissions, 209 accepted, first after 00:02

K — Knitting Pattern

Problem

Figure out how to centre a knitting pattern on a sweater.

Solutiom

K — Knitting Pattern

Problem

Figure out how to centre a knitting pattern on a sweater.

Solutiom

1. We place the pattern in the centre, reaching $p/2$ away from the middle in either direction. Then we have $(n-p)/2$ left on either side.

K — Knitting Pattern

Problem

Figure out how to centre a knitting pattern on a sweater.

Solutiom

1. We place the pattern in the centre, reaching $p/2$ away from the middle in either direction. Then we have $(n-p)/2$ left on either side.
2. We take this modulo p to get the leftover space on one side, so multiply by two to get our answer.

K — Knitting Pattern

Problem

Figure out how to centre a knitting pattern on a sweater.

Solutiom

1. We place the pattern in the centre, reaching $p/2$ away from the middle in either direction. Then we have $(n-p)/2$ left on either side.
2. We take this modulo p to get the leftover space on one side, so multiply by two to get our answer.
3. The only exception is if we fit one more pattern exactly in the middle on the back, so when the leftover space is exactly p or equivalently when p divides n. In this case the answer is zero instead.

K — Knitting Pattern

Problem

Figure out how to centre a knitting pattern on a sweater.

Solutiom

1. We place the pattern in the centre, reaching $p/2$ away from the middle in either direction. Then we have $(n-p)/2$ left on either side.
2. We take this modulo p to get the leftover space on one side, so multiply by two to get our answer.
3. The only exception is if we fit one more pattern exactly in the middle on the back, so when the leftover space is exactly p or equivalently when p divides n. In this case the answer is zero instead.

Statistics at 4-hour mark: 687 submissions, 152 accepted, first after 00:04

A — Avoiding the Abyss

Problem

Given a starting point, target point, and one point in a rectangular swimming pool, find a way to walk from the start to the target without intersecting the pool.

Solution

A — Avoiding the Abyss

Problem

Given a starting point, target point, and one point in a rectangular swimming pool, find a way to walk from the start to the target without intersecting the pool.

Solution

1. Observation 1
   You can't intersect the pool if you are outside of $-10^4 \leq x, y \leq 10^4$

A — Avoiding the Abyss

Problem

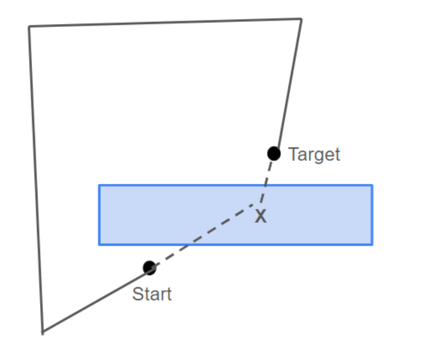
Given a starting point, target point, and one point in a rectangular swimming pool, find a way to walk from the start to the target without intersecting the pool.

Solution

1. Observation 1
   You can't intersect the pool if you are outside of $-10^4 \leq x, y \leq 10^4$
2. Observation 2
   You can't intersect the pool while walking away from the pool point.

A — Avoiding the Abyss

Solution



Statistics at 4-hour mark: 412 submissions, 126 accepted, first after 00:16

I — Infinite Cash

Problem

Given the starting money, salary and salary frequency of a fiscally unsound man, find out how long he has before he goes broke (if ever).

Solution

I — Infinite Cash

Problem

Given the starting money, salary and salary frequency of a fiscally unsound man, find out how long he has before he goes broke (if ever).

Solution

1. There are two ways to go about this. The first is to notice that the number of days Svalur lasts can not be so high, and simply simulate $\approx 10^6$ days and return an answer, answering infinite if he has not gone broke yet.

I — Infinite Cash

Problem

Given the starting money, salary and salary frequency of a fiscally unsound man, find out how long he has before he goes broke (if ever).

Solution

1. There are two ways to go about this. The first is to notice that the number of days Svalur lasts can not be so high, and simply simulate $\approx 10^6$ days and return an answer, answering infinite if he has not gone broke yet.
2. The other is to do things more by hand. Let $b(x)$ be the number of binary digits in x. Then to check if he goes broke before his first payment, check whether $b(m) < d$. To check whether he never goes broke, check whether $d \leq b(s)$. Then if $b(d) > 12$ just print $b(m)$. Finally simulate the salary periods one by one and print when he runs out of money.

I — Infinite Cash

Problem

Given the starting money, salary and salary frequency of a fiscally unsound man, find out how long he has before he goes broke (if ever).

Solution

1. There are two ways to go about this. The first is to notice that the number of days Svalur lasts can not be so high, and simply simulate $\approx 10^6$ days and return an answer, answering infinite if he has not gone broke yet.
2. The other is to do things more by hand. Let $b(x)$ be the number of binary digits in x. Then to check if he goes broke before his first payment, check whether $b(m) < d$. To check whether he never goes broke, check whether $d \leq b(s)$. Then if $b(d) > 12$ just print $b(m)$. Finally simulate the salary periods one by one and print when he runs out of money.

Statistics at 4-hour mark: 575 submissions, 81 accepted, first after 00:12

D — Double Deck

Problem

Solve the longest common subsequence problem when the number of occurrences of any specific value is ≤ 15.

Solution

D — Double Deck

Problem

Solve the longest common subsequence problem when the number of occurrences of any specific value is ≤ 15.

Solution

1. We consider the classic dynamic programming solution and the space optimization of that solution.

D — Double Deck

Problem

Solve the longest common subsequence problem when the number of occurrences of any specific value is ≤ 15.

Solution

1. We consider the classic dynamic programming solution and the space optimization of that solution.
2. Next we place this single row of the dynamic programming memoization table into a data structure like a segment tree or fenwick tree that can query maximum values and update points.

D — Double Deck

Problem

Solve the longest common subsequence problem when the number of occurrences of any specific value is ≤ 15.

Solution

1. We consider the classic dynamic programming solution and the space optimization of that solution.
2. Next we place this single row of the dynamic programming memoization table into a data structure like a segment tree or fenwick tree that can query maximum values and update points.
3. Now the number of updates we have to make in the tree is bounded due to the few occurrences of a given value.

D — Double Deck

Problem

Solve the longest common subsequence problem when the number of occurrences of any specific value is ≤ 15.

Solution

1. We consider the classic dynamic programming solution and the space optimization of that solution.
2. Next we place this single row of the dynamic programming memoization table into a data structure like a segment tree or fenwick tree that can query maximum values and update points.
3. Now the number of updates we have to make in the tree is bounded due to the few occurrences of a given value.

Statistics at 4-hour mark: 214 submissions, 23 accepted, first after 00:10

F — Fence Fee

Problem

Compute the faces of the provided planar graph, and find the areas (squared) of all faces (except the outer face).

Solution

F — Fence Fee

Problem

Compute the faces of the provided planar graph, and find the areas (squared) of all faces (except the outer face).

Solution

1. For each vertex, sort its neighborhood counter-clockwise (or cw).

F — Fence Fee

Problem

Compute the faces of the provided planar graph, and find the areas (squared) of all faces (except the outer face).

Solution

1. For each vertex, sort its neighborhood counter-clockwise (or cw).
2. For each edge uv, find the left-hand face of uv by going from u to v, and then go to the "next" neighbor (after u) of v, and repeat until you return to u.

F — Fence Fee

Problem

Compute the faces of the provided planar graph, and find the areas (squared) of all faces (except the outer face).

Solution

1. For each vertex, sort its neighborhood counter-clockwise (or cw).
2. For each edge uv, find the left-hand face of uv by going from u to v, and then go to the "next" neighbor (after u) of v, and repeat until you return to u.
3. Repeat the process for vu to find the right-hand face.

F — Fence Fee

Problem

Compute the faces of the provided planar graph, and find the areas (squared) of all faces (except the outer face).

Solution

1. For each vertex, sort its neighborhood counter-clockwise (or cw).
2. For each edge uv, find the left-hand face of uv by going from u to v, and then go to the "next" neighbor (after u) of v, and repeat until you return to u.
3. Repeat the process for vu to find the right-hand face.
4. Detect the outer face (the leftmost vertex (break even on y coordinate) and its neighbor with angle highest $\leq 90°$)

F — Fence Fee

Problem

Compute the faces of the provided planar graph, and find the areas (squared) of all faces (except the outer face).

Solution

1. For each vertex, sort its neighborhood counter-clockwise (or cw).
2. For each edge uv, find the left-hand face of uv by going from u to v, and then go to the "next" neighbor (after u) of v, and repeat until you return to u.
3. Repeat the process for vu to find the right-hand face.
4. Detect the outer face (the leftmost vertex (break even on y coordinate) and its neighbor with angle highest $\leq 90°$)
5. Use the shoelace algorithm for computing the area of each face/polygon.

Statistics at 4-hour mark: 61 submissions, 16 accepted, first after 00:50

J — Jungle Game

Problem

You are given N forbidden pairs (P_i, S_i) such that $1 \leq P_i, S_i \leq 2N$. Find N different pairs (p_i, s_i) satisfying $1 \leq p_i, s_i \leq N$, such that $(p_i + p_j, s_i + s_j)$ is never forbidden.

Solution

J — Jungle Game

Problem

You are given N forbidden pairs (P_i, S_i) such that $1 \leq P_i, S_i \leq 2N$. Find N different pairs (p_i, s_i) satisfying $1 \leq p_i, s_i \leq N$, such that $(p_i + p_j, s_i + s_j)$ is never forbidden.

Solution

1. Case 1.
   If there is an even number S such that (P, S) is never forbidden, then you can answer $(1, S/2), (2, S/2), \ldots, (N, S/2)$.

J — Jungle Game

Problem

You are given N forbidden pairs (P_i, S_i) such that $1 \leq P_i, S_i \leq 2N$. Find N different pairs (p_i, s_i) satisfying $1 \leq p_i, s_i \leq N$, such that $(p_i + p_j, s_i + s_j)$ is never forbidden.

Solution

1. Case 1.
   If there is an even number S such that (P, S) is never forbidden, then you can answer $(1, S/2), (2, S/2), \ldots, (N, S/2)$.
2. Case 2.
   If there is an even number P such that (P, S) is never forbidden, then you can answer $(P/2, 1), (P/2, 2), \ldots, (P/2, N)$.

J — Jungle Game

Problem

You are given N forbidden pairs (P_i, S_i) such that $1 \leq P_i, S_i \leq 2N$. Find N different pairs (p_i, s_i) satisfying $1 \leq p_i, s_i \leq N$, such that $(p_i + p_j, s_i + s_j)$ is never forbidden.

Solution

1. Case 1.
   If there is an even number S such that (P, S) is never forbidden, then you can answer $(1, S/2), (2, S/2), \ldots, (N, S/2)$.
2. Case 2.
   If there is an even number P such that (P, S) is never forbidden, then you can answer $(P/2, 1), (P/2, 2), \ldots, (P/2, N)$.
3. Case 3:
   Otherwise, both lists P_i and S_i are permutations of $2, 4, 6, \ldots, 2N$.

J — Jungle Game

Case 3

Both lists P_i and S_i are permutations of $2, 4, 6, \ldots, 2N$.

J — Jungle Game

Case 3

Both lists P_i and S_i are permutations of $2, 4, 6, \ldots, 2N$.

1. If $N = 1$, it is impossible.

J — Jungle Game

Case 3

Both lists P_i and S_i are permutations of $2, 4, 6, \ldots, 2N$.

1. If $N = 1$, it is impossible.
2. Otherwise, find the unique forbidden point $(2, S)$. Add the points $(2, S/2), (3, S/2), \ldots (N, S/2)$ to the solution. We now need to find just one more point.

J — Jungle Game

Case 3

Both lists P_i and S_i are permutations of $2, 4, 6, \ldots, 2N$.

1. If $N = 1$, it is impossible.
2. Otherwise, find the unique forbidden point $(2, S)$. Add the points $(2, S/2), (3, S/2), \ldots (N, S/2)$ to the solution. We now need to find just one more point.
3. If $S/2$ is even, then either $(1, 1)$ or $(2, 1)$ will work. Otherwise, either $(2, 1)$ or $(2, 2)$ will work. Try all four.
4. Running time: $O(N)$.

J — Jungle Game

Heuristic solutions

The intended solution is $O(N)$, but checking the answer takes $O(N^2)$. Unfortunately, this means that various $O(N^2)$ heuristics can pass as well.

1. Iterate through every pair (p, s) in a **random** order.
2. For each pair, check if it can be added to the answer.

Question: If the order is randomized, does the above solution work with high probability (or should we have made better test data)?

Statistics at 4-hour mark: 35 submissions, 8 accepted, first after 01:23

G — Guessing Passwords

Problem

Given a bipartite graph where all but the last node in the left half have the same degree, find an edge-colouring such that each colour touches exactly one node in the left half or only the last node in the left half.

Solution

G — Guessing Passwords

Problem

Given a bipartite graph where all but the last node in the left half have the same degree, find an edge-colouring such that each colour touches exactly one node in the left half or only the last node in the left half.

Solution

1. Ignore the last node for now. If such an edge colouring is possible, we can construct it by greedily making one colour at a time. The only constraint is that each colour must match every vertex on the right hand side of maximum degree.

G — Guessing Passwords

Problem

Given a bipartite graph where all but the last node in the left half have the same degree, find an edge-colouring such that each colour touches exactly one node in the left half or only the last node in the left half.

Solution

1. Ignore the last node for now. If such an edge colouring is possible, we can construct it by greedily making one colour at a time. The only constraint is that each colour must match every vertex on the right hand side of maximum degree.
2. This can be done with MCMF or by augmenting the standard bipartite match flow graph to use normal flow algorithms.

G — Guessing Passwords

Problem

Given a bipartite graph where all but the last node in the left half have the same degree, find an edge-colouring such that each colour touches exactly one node in the left half or only the last node in the left half.

Solution

1. Ignore the last node for now. If such an edge colouring is possible, we can construct it by greedily making one colour at a time. The only constraint is that each colour must match every vertex on the right hand side of maximum degree.
2. This can be done with MCMF or by augmenting the standard bipartite match flow graph to use normal flow algorithms.
3. The last row is dealt with by ignoring the Ys in the columns with the most Ys already, so that it has as many Ys as the other rows.

G — Guessing Passwords

Problem

Given a bipartite graph where all but the last node in the left half have the same degree, find an edge-colouring such that each colour touches exactly one node in the left half or only the last node in the left half.

Solution

1. Ignore the last node for now. If such an edge colouring is possible, we can construct it by greedily making one colour at a time. The only constraint is that each colour must match every vertex on the right hand side of maximum degree.
2. This can be done with MCMF or by augmenting the standard bipartite match flow graph to use normal flow algorithms.
3. The last row is dealt with by ignoring the Ys in the columns with the most Ys already, so that it has as many Ys as the other rows.

Statistics at 4-hour mark: 11 submissions, 1 accepted, first after 03:15

B — Baseball Court

Problem

Given bounds a, b how many partitions ρ of n satisfy $|\rho| \leq b$, $\max \rho \leq a$ and $\rho_i + i$ is constant across all indices i such that $\rho_i \neq \rho_{i+1}$.

Solution

Problem

Given bounds a, b how many partitions ρ of n satisfy $|\rho| \leq b$, $\max \rho \leq a$ and $\rho_i + i$ is constant across all indices i such that $\rho_i \neq \rho_{i+1}$.

Solution

1. First consider the case without the bounds a, b.

B — Baseball Court

Problem

Given bounds a, b how many partitions ρ of n satisfy $|\rho| \leq b$, $\max \rho \leq a$ and $\rho_i + i$ is constant across all indices i such that $\rho_i \neq \rho_{i+1}$.

Solution

1. First consider the case without the bounds a, b.
2. We define $f(n, d)$ as the number of partitions of n with $\rho_i + i = d + 1$ for the aforementioned indices.

B — Baseball Court

Problem

Given bounds a, b how many partitions ρ of n satisfy $|\rho| \leq b$, $\max \rho \leq a$ and $\rho_i + i$ is constant across all indices i such that $\rho_i \neq \rho_{i+1}$.

Solution

1. First consider the case without the bounds a, b.
2. We define $f(n, d)$ as the number of partitions of n with $\rho_i + i = d + 1$ for the aforementioned indices.
3. The problem can now be solved using dynamic programming.

B — Baseball Court

Solution

1. Consider the maximum index k such that $\rho_1 = \rho_k$ and $\rho_k \neq \rho_{k+1}$. Then $\rho_k + k = d + 1$. Then by cutting off these front values we remove k values and a sum of $k(d+1-k)$. This gives us the recurrence

$$f(n,d) = \sum_{k=1}^{d} f(n - k(d+1-k), d-k)$$

Solution

1. Consider the maximum index k such that $\rho_1 = \rho_k$ and $\rho_k \neq \rho_{k+1}$. Then $\rho_k + k = d + 1$. Then by cutting off these front values we remove k values and a sum of $k(d+1-k)$. This gives us the recurrence

$$f(n, d) = \sum_{k=1}^{d} f(n - k(d+1-k), d - k)$$

2. This is $\mathcal{O}(n^3)$ which is not quite good enough. But $f(n - k(d+1-k), d-k)$ has the first argument < 0 for all but the first and last $\sqrt{n}$ terms. So we can skip those and get a time complexity of $\mathcal{O}(n^{2.5})$.

Solution

1. Consider the maximum index k such that $\rho_1 = \rho_k$ and $\rho_k \neq \rho_{k+1}$. Then $\rho_k + k = d + 1$. Then by cutting off these front values we remove k values and a sum of $k(d+1-k)$. This gives us the recurrence

$$f(n,d) = \sum_{k=1}^{d} f(n - k(d+1-k), d-k)$$

2. This is $\mathcal{O}(n^3)$ which is not quite good enough. But $f(n - k(d+1-k), d-k)$ has the first argument < 0 for all but the first and last $\sqrt{n}$ terms. So we can skip those and get a time complexity of $\mathcal{O}(n^{2.5})$.
3. Then we simply subtract all partitions that have $|\rho| > a$ or $\max \rho > b$, then add back all the ones that violate both to get the right answer by inclusion-exclusion.

B — Baseball Court

Solution

1. Consider the maximum index k such that $\rho_1 = \rho_k$ and $\rho_k \neq \rho_{k+1}$. Then $\rho_k + k = d + 1$. Then by cutting off these front values we remove k values and a sum of $k(d+1-k)$. This gives us the recurrence

$$f(n,d) = \sum_{k=1}^{d} f(n - k(d+1-k), d-k)$$

2. This is $\mathcal{O}(n^3)$ which is not quite good enough. But $f(n-k(d+1-k), d-k)$ has the first argument < 0 for all but the first and last $\sqrt{n}$ terms. So we can skip those and get a time complexity of $\mathcal{O}(n^{2.5})$.
3. Then we simply subtract all partitions that have $|\rho| > a$ or $\max \rho > b$, then add back all the ones that violate both to get the right answer by inclusion-exclusion.

Statistics at 4-hour mark: 14 submissions, 0 accepted, first after ??:??

E — Elapid Errands

Problem

Given $N = 20$ random points in a grid, visit all of them in order while never intersecting yourself.

Solution

E — Elapid Errands

Problem

Given $N = 20$ random points in a grid, visit all of them in order while never intersecting yourself.

Solution

1. Greedily walk to each point? You will probably intersect yourself.
2. Also avoid visiting the same point twice? The plane will probably get divided into disjoint regions where future points can't be visited.

E — Elapid Errands

Problem

Given $N = 20$ random points in a grid, visit all of them in order while never intersecting yourself.

Solution

1. Greedily walk to each point? You will probably intersect yourself.
2. Also avoid visiting the same point twice? The plane will probably get divided into disjoint regions where future points can't be visited.
3. Main idea: Try to make a snake region that doesn't contain any holes. This way, points will not become unreachable.

E — Elapid Errands

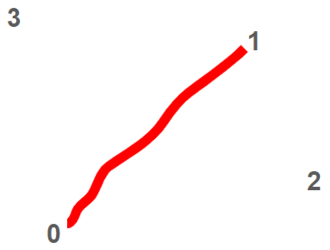
Solution

For each point P you want to visit:

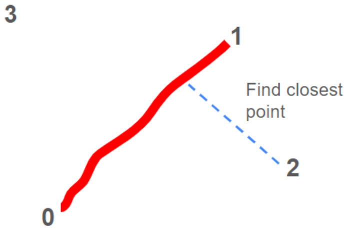
1. Find a point Q on the snake boundary that is as close as possible to P.
2. Walk along the boundary of the snake to Q.
3. Greedily walk from Q to P. You will not intersect yourself since Q was as close as possible to P.


3
1
2
0

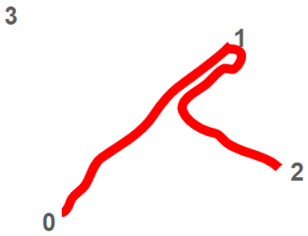
E — Elapid Errands



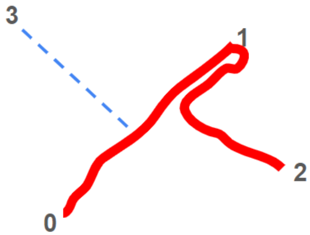
E — Elapid Errands



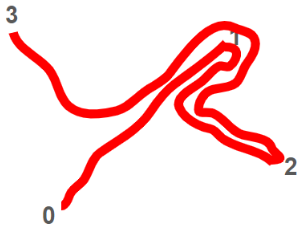
E — Elapid Errands



E — Elapid Errands



E — Elapid Errands



E — Elapid Errands

Solution

For each point P you want to visit:

1. Find a point Q on the snake boundary that is as close as possible to P.
2. Walk along the boundary of the snake to Q.
3. Greedily walk from Q to P. You will not intersect yourself since Q was as close as possible to P.

Issues

1. In the third step, it is possible that you accidentally pass too close to a future point. Put a "danger zone" around every future point so that you don't step on them too early.
2. The snake region will probably create 1x1 holes, some care is needed to make sure you don't walk into a dead end.

Statistics at 4-hour mark: 25 submissions, 0 accepted, first after ??:??

H — Hotfix

Problem

Given a program that prints every unique substring of a string along with its number of occurrences, find out how many copies of each non-whitespace character that program would print for a given string.

Solution

H — Hotfix

Problem

Given a program that prints every unique substring of a string along with its number of occurrences, find out how many copies of each non-whitespace character that program would print for a given string.

Solution

1. To solve this we need suffix arrays and segment trees. We start by constructing the suffix array and its longest common prefix array.

H — Hotfix

Problem

Given a program that prints every unique substring of a string along with its number of occurrences, find out how many copies of each non-whitespace character that program would print for a given string.

Solution

1. To solve this we need suffix arrays and segment trees. We start by constructing the suffix array and its longest common prefix array.
2. We now solve the characters and numbers separately.

H — Hotfix

Problem

Given a program that prints every unique substring of a string along with its number of occurrences, find out how many copies of each non-whitespace character that program would print for a given string.

Solution

1. To solve this we need suffix arrays and segment trees. We start by constructing the suffix array and its longest common prefix array.
2. We now solve the characters and numbers separately.
3. For the letters we use a lazy propagation segment tree that allows for a range update where the first element is incremented by b, the next by $a + b$, the third by $2a + b$ and so on.

Solution

1. The segment tree will keep track of the number of occurrences of each character in the input string as we iterate over the suffix array. Let S be the i-th element of the suffix array and L be the $(i-1)$-st element of the longest common prefix array.

Solution

1. The segment tree will keep track of the number of occurrences of each character in the input string as we iterate over the suffix array. Let S be the i-th element of the suffix array and L be the $(i-1)$-st element of the longest common prefix array.
2. Then $n - S - L$ new prefixes start at position S. The first L possible substrings starting at S are duplicates. So we increment the segment tree by $n - S - L$ at positions S through $S + L - 1$. Then we increment the segment tree by $n - S - L$, $n - S - L - 1$, $n - S - L - 2$, and so on at positions $S + L$ through $n - 1$. At the end of this all we can read off the number of occurrences of each character from the segment tree.

Solution

1. Finally we consider the digits. For this we use a lazy propagation segment tree that allows incrementing values on a range and then a collect operation that both counts the digits in the values on a range before then zeroing them out.

Solution

1. Finally we consider the digits. For this we use a lazy propagation segment tree that allows incrementing values on a range and then a collect operation that both counts the digits in the values on a range before then zeroing them out.
2. The collect operation might be slow worst case, but is amortized fast. We let the values of the segment tree count the number of occurrences of the substrings starting at our current position in the string as we iterate through it.

Solution

1. Finally we consider the digits. For this we use a lazy propagation segment tree that allows incrementing values on a range and then a collect operation that both counts the digits in the values on a range before then zeroing them out.
2. The collect operation might be slow worst case, but is amortized fast. We let the values of the segment tree count the number of occurrences of the substrings starting at our current position in the string as we iterate through it.
3. As we move from position $i-1$ to i, let S and L be as before.

Solution

1. Finally we consider the digits. For this we use a lazy propagation segment tree that allows incrementing values on a range and then a collect operation that both counts the digits in the values on a range before then zeroing them out.
2. The collect operation might be slow worst case, but is amortized fast. We let the values of the segment tree count the number of occurrences of the substrings starting at our current position in the string as we iterate through it.
3. As we move from position $i - 1$ to i, let S and L be as before.
4. Everything but the first L substrings are no longer valid, so we count those and delete them using the collect operation on indices L to $n - 1$.

Solution

1. Finally we consider the digits. For this we use a lazy propagation segment tree that allows incrementing values on a range and then a collect operation that both counts the digits in the values on a range before then zeroing them out.
2. The collect operation might be slow worst case, but is amortized fast. We let the values of the segment tree count the number of occurrences of the substrings starting at our current position in the string as we iterate through it.
3. As we move from position $i-1$ to i, let S and L be as before.
4. Everything but the first L substrings are no longer valid, so we count those and delete them using the collect operation on indices L to $n-1$.
5. Next we add the substrings found at our current position, incrementing the values at indices 0 through $n-S-1$.

Solution

1. Finally we consider the digits. For this we use a lazy propagation segment tree that allows incrementing values on a range and then a collect operation that both counts the digits in the values on a range before then zeroing them out.
2. The collect operation might be slow worst case, but is amortized fast. We let the values of the segment tree count the number of occurrences of the substrings starting at our current position in the string as we iterate through it.
3. As we move from position $i - 1$ to i, let S and L be as before.
4. Everything but the first L substrings are no longer valid, so we count those and delete them using the collect operation on indices L to $n - 1$.
5. Next we add the substrings found at our current position, incrementing the values at indices 0 through $n - S - 1$.
6. This way we collect all digits in the output.

Solution

1. Finally we consider the digits. For this we use a lazy propagation segment tree that allows incrementing values on a range and then a collect operation that both counts the digits in the values on a range before then zeroing them out.
2. The collect operation might be slow worst case, but is amortized fast. We let the values of the segment tree count the number of occurrences of the substrings starting at our current position in the string as we iterate through it.
3. As we move from position $i-1$ to i, let S and L be as before.
4. Everything but the first L substrings are no longer valid, so we count those and delete them using the collect operation on indices L to $n-1$.
5. Next we add the substrings found at our current position, incrementing the values at indices 0 through $n-S-1$.
6. This way we collect all digits in the output.

Statistics at 4-hour mark: 42 submissions, 0 accepted, first after ??:??

Results!