ICPC Thailand National Competition 2024

- Tutorial Slide -

8th September 2024

A- Auntie's Magical Cake

- Problem Author: Akarapon Watcharapalakorn
- Solved by 20 teams.
- First solved after 20 minutes.

A- Auntie's Magical Cake

- The best strategy is to eat the cakes in order, either from the leftmost to the rightmost or from the rightmost to the leftmost. This is because choosing the cake furthest from the middle will maximize the total deliciousness.

A- Auntie's Magical Cake

- The best strategy is to eat the cakes in order, either from the leftmost to the rightmost or from the rightmost to the leftmost. This is because choosing the cake furthest from the middle will maximize the total deliciousness.
- Consider eating the i-th cake, where $1 < i < N/2$, and let the increases in deliciousness from the left-side and right-side cakes be x and y, respectively.

A- Auntie's Magical Cake

- The best strategy is to eat the cakes in order, either from the leftmost to the rightmost or from the rightmost to the leftmost. This is because choosing the cake furthest from the middle will maximize the total deliciousness.
- Consider eating the i-th cake, where $1 < i < N/2$, and let the increases in deliciousness from the left-side and right-side cakes be x and y, respectively.
- If you change the position from i to $i+1$, the new increases in deliciousness will be $x+i$ and $y-(i+1)$. It's clear that the total in the first case, $x+y$, is always higher than in the second case, $x+i+y-(i+1) = x+y-1$.

A- Auntie's Magical Cake

- The best strategy is to eat the cakes in order, either from the leftmost to the rightmost or from the rightmost to the leftmost. This is because choosing the cake furthest from the middle will maximize the total deliciousness.
- Consider eating the i-th cake, where $1 < i < N/2$, and let the increases in deliciousness from the left-side and right-side cakes be x and y, respectively.
- If you change the position from i to $i + 1$, the new increases in deliciousness will be $x + i$ and $y - (i + 1)$. It's clear that the total in the first case, $x + y$, is always higher than in the second case, $x + i + y - (i + 1) = x + y - 1$.
- The same method can also be applied for cases where $i > N/2$.

A- Auntie's Magical Cake

- Time complexity is $O(N)$.

A- Auntie's Magical Cake

- Time complexity is $O(N)$.
- Can be optimized to $O(1)$ with math.

B- Back in the Day

- Problem Author: Mattanyu Tangngekkee
- Solved by 34 teams.
- First solved after 14 minutes.

B- Back in the Day

- Divide the number string into several parts, each contain only one number. For example, 2222888 is divided into 2222 and 888.

B- Back in the Day

- Divide the number string into several parts, each contain only one number. For example, 2222888 is divided into 2222 and 888.
- For each part, greedily choose the biggest character possible. Then put the smallest character remained in the front. For example 2222 becomes *ac* and 7777777777 becomes *qss*.

B- Back in the Day

- Alternatively, you can reverse the number string from last to first, then you can simply cut the string every time it reaches the highest character for each key number. Then reverse the answer before outputting.

B- Back in the Day

- Alternatively, you can reverse the number string from last to first, then you can simply cut the string every time it reaches the highest character for each key number. Then reverse the answer before outputting.
- **Be careful**, each number contains different amount of characters. '7' and '9' contains 4 characters each while the rest contains just 3.

B- Back in the Day

- Alternatively, you can reverse the number string from last to first, then you can simply cut the string every time it reaches the highest character for each key number. Then reverse the answer before outputting.
- **Be careful**, each number contains different amount of characters. '7' and '9' contains 4 characters each while the rest contains just 3.
- Time complexity is $O(|S|)$.

C- Cattering

- Problem Author: Natapong Sriwatanasakdi
- Solved by 1 teams.
- First solved after 44 minutes.

C- Cattering

- In this problem, we can use **binary search** to find the maximum possible value of the minimum happiness for all cats.

C- Cattering

- In this problem, we can use **binary search** to find the maximum possible value of the minimum happiness for all cats.
- We can determine whether all cats can have happiness value at least h by **maximum bipartite matching**.

C- Cattering

- In this problem, we can use **binary search** to find the maximum possible value of the minimum happiness for all cats.
- We can determine whether all cats can have happiness value at least h by **maximum bipartite matching**.
- We build a bipartite graph of cats and foods. There exist an edge between cat i and food j if and only if $A_{ij} \geq h$. All cats can have happiness value at least h if and only if the **maximum bipartite matching size** of this graph is N.

C- Cattering

- In this problem, we can use **binary search** to find the maximum possible value of the minimum happiness for all cats.
- We can determine whether all cats can have happiness value at least h by **maximum bipartite matching**.
- We build a bipartite graph of cats and foods. There exist an edge between cat i and food j if and only if $A_{ij} \geq h$. All cats can have happiness value at least h if and only if the **maximum bipartite matching size** of this graph is N.
- Therefore, we can binary search to find the maximum value of happiness that all cats can reach. The search is $O(\log M)$ times since there are NM possible values—values in matrix A.
- Time complexity is $O(Matching \cdot \log M)$.

C- Cattering

- The official solution used Hopcroft-Karp Algorithm for matching.
- Kuhn Algorithm with modifications also works. In each vertex, try finding an available vertex that it can reach first before DFS on matched vertices. The algorithm can be further improved by shuffling edges first.
- One of the problem testers also uses Dinic's Algorithm. While Dinic's Algorithm has the same time complexity as Hopcroft-Karp Algorithm in case of maximum bipartite matching, the overhead from creating network flow is larger, hence the actual execution time is significantly slower.

D- Disinfection Patch

- Problem Author: Attitarn Buathep
- Solved by 3 teams.
- First solved after 111 minutes.

D- Disinfection Patch

- Suppose that $min_x, max_x, min_y, max_y$ are the minimum and the maximum value of x and y coordinates for the drops of disinfection respectively, we can construct a rectangle that cover all of the drops which have (min_x, min_y) and (max_x, max_y) as the bottom-left and the top-right corner of the shape.

D- Disinfection Patch

- Suppose that $min_x, max_x, min_y, max_y$ are the minimum and the maximum value of x and y coordinates for the drops of disinfection respectively, we can construct a rectangle that cover all of the drops which have (min_x, min_y) and (max_x, max_y) as the bottom-left and the top-right corner of the shape.
- We can do the same for the bacteria, then compare the ratio between the length of each side of the two rectangles.

D- Disinfection Patch

- Suppose that $min_x, max_x, min_y, max_y$ are the minimum and the maximum value of x and y coordinates for the drops of disinfection respectively, we can construct a rectangle that cover all of the drops which have (min_x, min_y) and (max_x, max_y) as the bottom-left and the top-right corner of the shape.
- We can do the same for the bacteria, then compare the ratio between the length of each side of the two rectangles.
- If the ratio are not equal to each other, output -1. Otherwise, we must scale the disinfection rectangle with the ratio and shift the corners to the bacteria rectangle.

D- Disinfection Patch

- After that, we check each point by sorting both the scaled disinfection points and the bacteria points and comparing the points one-by-one in order.

D- Disinfection Patch

- After that, we check each point by sorting both the scaled disinfection points and the bacteria points and comparing the points one-by-one in order.
- If the points mismatch, output -1. Otherwise, output the scale ratio and the shift S, X, Y.

D- Disinfection Patch

- After that, we check each point by sorting both the scaled disinfection points and the bacteria points and comparing the points one-by-one in order.
- If the points mismatch, output -1. Otherwise, output the scale ratio and the shift S, X, Y.
- There will be some special cases such as when x or y are the same for all points, so the length of some side of the rectangle becomes 0.

D- Disinfection Patch

- After that, we check each point by sorting both the scaled disinfection points and the bacteria points and comparing the points one-by-one in order.
- If the points mismatch, output -1. Otherwise, output the scale ratio and the shift S, X, Y.
- There will be some special cases such as when x or y are the same for all points, so the length of some side of the rectangle becomes 0.
- Time complexity is $O(N \log N)$

E- Executive's Holidays

- Problem Author: Mattanyu Tangngekkee
- Solved by 0 teams.

E- Executive's Holidays

- Instead of assigning meeting for the executives, you can think about reversely assign them the break day.

E- Executive's Holidays

- Instead of assigning meeting for the executives, you can think about reversely assign them the break day.
- For the i-th day, if there requires a_i executives for the meeting, then it also means there can be at most $N - a_i$ executives unattended that day.

E- Executive's Holidays

- We can greedily choose, for each executive, the longest period possible they don't have to attend any meeting.

E- Executive's Holidays

- We can greedily choose, for each executive, the longest period possible they don't have to attend any meeting.
  - For example, let the quota for executives unattended each day are $[1, 2, 0, 2, 1, 3, 2]$.
  - The first executive can only take a break on the day that has the quota. Therefore he can at most take a break from day 4 to day 7. After that, the quota becomes $[1, 2, 0, 1, 0, 2, 1]$.
  - The next executive can take a break from day 1 to day 2, and the quota left will be $[0, 1, 0, 1, 0, 2, 1]$ and so on...

E- Executive's Holidays

- We can greedily choose, for each executive, the longest period possible they don't have to attend any meeting.
  - For example, let the quota for executives unattended each day are $[1, 2, 0, 2, 1, 3, 2]$.
  - The first executive can only take a break on the day that has the quota. Therefore he can at most take a break from day 4 to day 7. After that, the quota becomes $[1, 2, 0, 1, 0, 2, 1]$.
  - The next executive can take a break from day 1 to day 2, and the quota left will be $[0, 1, 0, 1, 0, 2, 1]$ and so on...
- It can be shown that repeating the above process until every executive's break is assigned is the optimal strategy, as the sum of the length of those breaks will always be maximized.

E- Executive's Holidays

- We can greedily choose, for each executive, the longest period possible they don't have to attend any meeting.
  - For example, let the quota for executives unattended each day are $[1, 2, 0, 2, 1, 3, 2]$.
  - The first executive can only take a break on the day that has the quota. Therefore he can at most take a break from day 4 to day 7. After that, the quota becomes $[1, 2, 0, 1, 0, 2, 1]$.
  - The next executive can take a break from day 1 to day 2, and the quota left will be $[0, 1, 0, 1, 0, 2, 1]$ and so on...
- It can be shown that repeating the above process until every executive's break is assigned is the optimal strategy, as the sum of the length of those breaks will always be maximized.
- The rest of the unattended quota can be distributed however since it would not increase the sum.

E- Executive's Holidays

- It is equivalent to finding the the longest non-zero subarray and remove one for every item on that subarray. Repeating the process for N times and the sum of the length of all subarray is the answer.

E- Executive's Holidays

- It is equivalent to finding the the longest non-zero subarray and remove one for every item on that subarray. Repeating the process for N times and the sum of the length of all subarray is the answer.
- Finding the longest non-zero subarray for an array takes $O(T)$. Removing one for every member in the subarray also takes $O(T)$.

E- Executive's Holidays

- It is equivalent to finding the the longest non-zero subarray and remove one for every item on that subarray. Repeating the process for N times and the sum of the length of all subarray is the answer.
- Finding the longest non-zero subarray for an array takes $O(T)$. Removing one for every member in the subarray also takes $O(T)$.
- Since you have to repeat the process N times, time complexity is $O(N \cdot T)$, which is not fast enough.

E- Executive's Holidays

- To improve the speed, you can first visualize the problem as a histogram and partitioned it horizontally.

E- Executive's Holidays

- To improve the speed, you can first visualize the problem as a histogram and partitioned it horizontally.

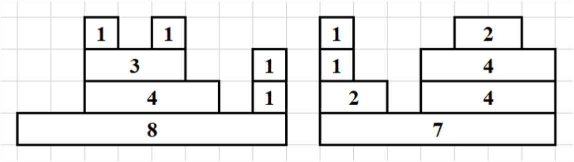


Figure: The histogram and its partition of the array $[1, 1, 4, 3, 4, 2, 1, 3, 0, 4, 2, 1, 3, 4, 4, 3]$

E- Executive's Holidays

- To improve the speed, you can first visualize the problem as a histogram and partitioned it horizontally.

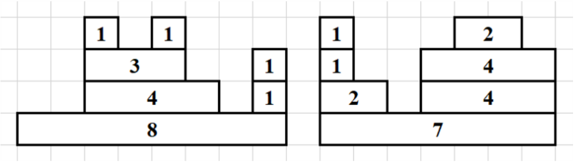


Figure: The histogram and its partition of the array $[1, 1, 4, 3, 4, 2, 1, 3, 0, 4, 2, 1, 3, 4, 4, 3]$

- When you sort the partition by its length from longest, it is the same range as the longest non-zero subarray previously, so the answer is the sum of the first N partition's length.

E- Executive's Holidays

- To improve the speed, you can first visualize the problem as a histogram and partitioned it horizontally.

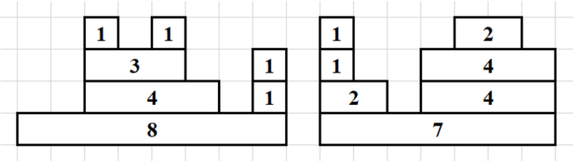


Figure: The histogram and its partition of the array $[1, 1, 4, 3, 4, 2, 1, 3, 0, 4, 2, 1, 3, 4, 4, 3]$

- When you sort the partition by its length from longest, it is the same range as the longest non-zero subarray previously, so the answer is the sum of the first N partition's length.
- You can obtain such partitions by iterating from left to right and keep the position and the height in a stack.

E- Executive's Holidays

- Since the partition count can potentially reach $O(N \cdot T)$, there is a need for a slight optimization.

E- Executive's Holidays

- Since the partition count can potentially reach $O(N \cdot T)$, there is a need for a slight optimization.
- It is unnecessary to slice the partition on every height. Instead, just partition the histogram into rectangles and storing its position, its length and its height.

E- Executive's Holidays

- Since the partition count can potentially reach $O(N \cdot T)$, there is a need for a slight optimization.
- It is unnecessary to slice the partition on every height. Instead, just partition the histogram into rectangles and storing its position, its length and its height.
- The height represents the amount of the actual slicing.

E- Executive's Holidays

- Since the partition count can potentially reach $O(N \cdot T)$, there is a need for a slight optimization.
- It is unnecessary to slice the partition on every height. Instead, just partition the histogram into rectangles and storing its position, its length and its height.
- The height represents the amount of the actual slicing.

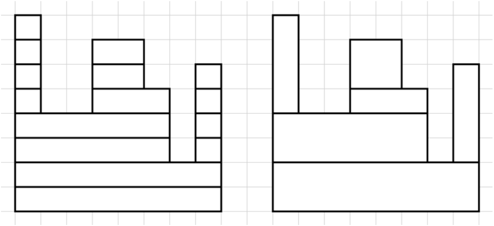


Figure: Example of an optimized partition.

E- Executive's Holidays

- The length of each partition cannot exceed T, so you can store the length in an array length $T+1$ to speed up the sorting and the counting.

E- Executive's Holidays

- The length of each partition cannot exceed T, so you can store the length in an array length $T + 1$ to speed up the sorting and the counting.
- Time complexity is $O(T)$

F- Fill T

- Problem Author: Mattanyu Tangngekkee
- Solved by 0 teams.

F- Fill T

- Let $R <= C$, otherwise you can rotate the grid by 90 degree.

F- Fill T

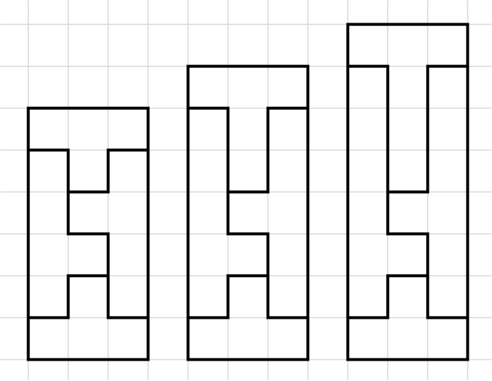
- Let $R <= C$, otherwise you can rotate the grid by 90 degree.
- If $R = 1$ or $R = 2$, it is obvious that the grid cannot be filled.

F- Fill T

- Let $R <= C$, otherwise you can rotate the grid by 90 degree.
- If $R = 1$ or $R = 2$, it is obvious that the grid cannot be filled.
- For $R = 3$ and $C \leq 5$, it also can be shown that the grid cannot be filled.
- That is all the grids that cannot be filled.

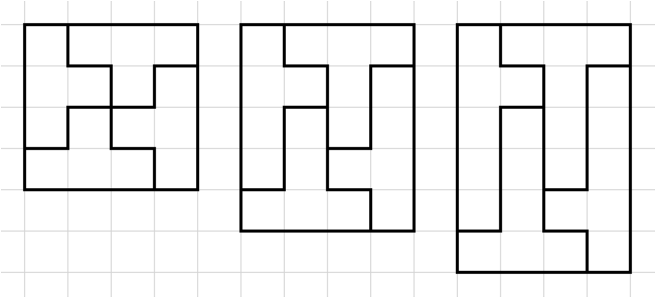
F- Fill T

- For $R = 3$ and $C \geq 6$, the grid can be filled with the pattern below.



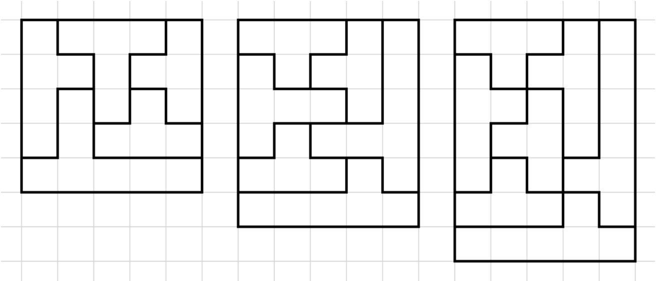
F- Fill T

- For $R = 4$, the grid can be filled with the pattern below.



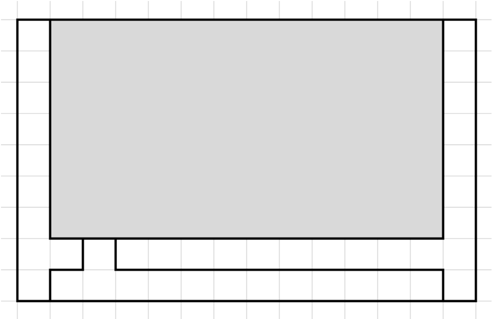
F- Fill T

- For $R = 5$ and $C = 5$, 6, or 7, the grid can be filled with the pattern below.



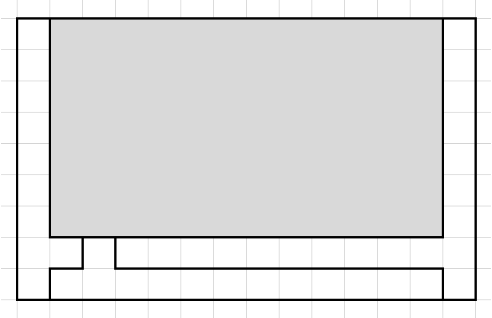
F- Fill T

- For $R = 5$ and $C \geq 8$ or $R \geq 6$, you can use 3 pieces of T-shape to form the border so that you can recursively fill out the $(R-2) \times (C-2)$ grid with the pattern below.



F- Fill T

- For $R = 5$ and $C \geq 8$ or $R \geq 6$, you can use 3 pieces of T-shape to form the border so that you can recursively fill out the $(R-2) \times (C-2)$ grid with the pattern below.



- Time complexity is $O(\max(R, C))$

G- Glory Road

- Problem Author: Mattanyu Tangngekkee
- Solved by 45 teams.
- First solved after 12 minutes.

G- Glory Road

- Let the vertices of the triangle be (A_x, A_y), (B_x, B_y) and (C_x, C_y) and the midpoint of each side be (P_x, P_y), (Q_x, Q_y) and (R_x, R_y).
- You can write the system equation as follows:

$$\begin{cases} \dfrac{A_x + B_x}{2} = P_x \\ \dfrac{B_x + C_x}{2} = Q_x \\ \dfrac{A_x + C_x}{2} = R_x \end{cases} \qquad \begin{cases} \dfrac{A_y + B_y}{2} = P_y \\ \dfrac{B_y + C_y}{2} = Q_y \\ \dfrac{A_y + C_y}{2} = R_y \end{cases}$$

G- Glory Road

- The system equation has a solution of:

$$\begin{cases} A_x = P_x - Q_x + R_x \\ B_x = P_x + Q_x - R_x \\ C_x = -P_x + Q_x + R_x \end{cases} \qquad \begin{cases} A_y = P_y - Q_y + R_y \\ B_y = P_y + Q_y - R_y \\ C_y = -P_y + Q_y + R_y \end{cases}$$

- Just be careful about the order of the input and the output.
- Time complexity is $O(1)$.

H- Heavenly Sequence

- Problem Author: Poonyapat Sriroth
- Solved by 0 teams.

H- Heavenly Sequence

- First, it is obvious that we will only choose the maximum X (by choosing $X = R$).

H- Heavenly Sequence

- First, it is obvious that we will only choose the maximum X (by choosing $X = R$).
- A sequence is considered **Heavenly** if and only if every permutation of every subsequence is **Good**.
- In other words, a sequence is **Heavenly** if, when we pick **any numbers in any order** from the sequence to create a new sequence, that new sequence is also **Good**.

H- Heavenly Sequence

- By using the rearrangement inequality (or through mathematical observation), we can simplify the definition of **Heavenly** to:

Lemma 1

A sequence $b[1], b[2], b[3], \ldots, b[n]$ is **Heavenly** if and only if, after sorting the sequence, for each $i \leq j$, the subsequence $b[i], b[i+1], b[i+2], \ldots, b[j]$ is **Good**.

H- Heavenly Sequence

- By using the rearrangement inequality (or through mathematical observation), we can simplify the definition of **Heavenly** to:

Lemma 1

A sequence $b[1], b[2], b[3], \ldots, b[n]$ is **Heavenly** if and only if, after sorting the sequence, for each $i \leq j$, the subsequence $b[i], b[i+1], b[i+2], \ldots, b[j]$ is **Good**.

- This lemma can be proven by fixing the maximum and minimum of the new sequence and then maximizing the right-hand side of the inequality.
  - This is achieved by selecting every value between the maximum and minimum and sorting that sequence.
  - Considering all possible maximum and minimum values and combine everything will results in above observation.

H- Heavenly Sequence

- Suppose the sorted sequence is $b[1], b[2], b[3], \ldots, b[n]$. Given the previous observation, there are still many **Good** sequences to consider.

H- Heavenly Sequence

- Suppose the sorted sequence is $b[1], b[2], b[3], \ldots, b[n]$. Given the previous observation, there are still many **Good** sequences to consider.
- However, we can reduce this by noting that:

Lemma 2

If $b[1], b[2], b[3], \ldots, b[j]$ is **Good**, then for any i with $i \leq j$, the subsequence $b[i], b[i+1], b[i+2], \ldots, b[j]$ is also **Good**.

H- Heavenly Sequence

- Thus, we only need to consider K such that for every j (where $j \leq N$), $b[1], b[2], b[3], \ldots, b[j]$ remains **Good**.

H- Heavenly Sequence

- Thus, we only need to consider K such that for every j (where $j \leq N$), $b[1], b[2], b[3], \ldots, b[j]$ remains **Good**.
- To achieve this, we need to find K that satisfies the inequality (we know that the minimum is $b[1]$ and maximum is $b[j]$):

$$K \cdot b[1] \geq b[j]^2 + \sum_{i=1}^{j-1} b[i] \cdot b[i+1] - X \cdot b[j]$$

for $j = 2, 3, \ldots, N$.

H- Heavenly Sequence

- Consequently, we seek a data structure that can efficiently solve the right side of the inequality to find the minimum K:

$$K \cdot b[1] \geq \max \left(b[j]^2 + \sum_{i=1}^{j-1} b[i] \cdot b[i+1] - X \cdot b[j] \right)$$

for $j = 2, 3, \ldots, N$.

H- Heavenly Sequence

- Consequently, we seek a data structure that can efficiently solve the right side of the inequality to find the minimum K:

$$K \cdot b[1] \geq \max\left(b[j]^2 + \sum_{i=1}^{j-1} b[i] \cdot b[i+1] - X \cdot b[j]\right)$$

for $j = 2, 3, \ldots, N$.

- The right side of this inequality requires us to find the maximum value of linear equations at specific points. Since this operation must be performed online (both inserting lines and retrieving answers), one suitable data structure for this is the **lazy Li Chao tree**. You can learn more about it at `https://codeforces.com/blog/entry/86731`

H- Heavenly Sequence

- It's important to note that when we add new elements to the sequence, it may seem that we need to update almost every line. However, we can efficiently update only a few lines by using range addition.

H- Heavenly Sequence

- It's important to note that when we add new elements to the sequence, it may seem that we need to update almost every line. However, we can efficiently update only a few lines by using range addition.
- Time complexity is $O(N \log^2 N)$

I- Ideal Permutation Pairing

- Problem Author: Mattanyu Tangngekkee
- Solved by 4 teams.
- First solved after 81 minutes.

I- Ideal Permutation Pairing

- Let P is a permutation size N have a rank of p if it is the p-th smallest permutation of size N. For shorten, we can say that $rank(P) = p$.

I- Ideal Permutation Pairing

- Let P is a permutation size N have a rank of p if it is the p-th smallest permutation of size N. For shorten, we can say that $rank(P) = p$.

Lemma

Let $P = p_1 p_2 p_3 \dots p_N$ and $Q = q_1 q_2 q_3 \dots q_N$ be permutations of size N. If $rank(Q) = rank(P) + t \cdot (N-k)!$ for some k and t, then $p_{k+1} p_{k+2} \dots p_N$ and $q_{k+1} q_{k+2} \dots q_n$ have the same ordering.

I- Ideal Permutation Pairing

Lemma

Let $P = p_1 p_2 p_3 \ldots p_N$ and $Q = q_1 q_2 q_3 \ldots q_N$ be permutations of size N. If $rank(Q) = rank(P) + t \cdot (N - k)!$ for some k and t, then $p_{k+1} p_{k+2} \ldots p_N$ and $q_{k+1} q_{k+2} \ldots q_N$ have the same ordering.

I- Ideal Permutation Pairing

Lemma

Let $P = p_1 p_2 p_3 \dots p_N$ and $Q = q_1 q_2 q_3 \dots q_N$ be permutations of size N. If $rank(Q) = rank(P) + t \cdot (N-k)!$ for some k and t, then $p_{k+1} p_{k+2} \dots p_N$ and $q_{k+1} q_{k+2} \dots q_N$ have the same ordering.

Proof

We can prove by induction from $k = N \to 1$. The $k = N$ part is obvious. If $F(k)$ is true, considering the possible value of p_k if $p_1 p_2 \dots p_{k-1}$ is fixed, there are $N - k + 1$ possibles candidates, which also is the list $[p_k, p_{k+1}, \dots, p_n]$. That means if $rank(Q) = rank(P) + t \cdot (N-k)! = rank(P) + t \cdot (N-k-1) \cdot (N-k)!$, the rank of p_k among the list $[p_k, p_{k+1}, \dots, p_n]$ and the rank of q_k among the list $[q_k, q_{k+1}, \dots, q_N]$ is equal, making $p_k p_{k+1} \dots p_N$ and $q_k q_{k+1} \dots q_N$ have the same ordering, thus $F(k-1)$ is true.

I- Ideal Permutation Pairing

Lemma

Let $P = p_1p_2p_3 \ldots p_N$ and $Q = q_1q_2q_3 \ldots q_N$ be permutations of size N. If $rank(Q) = rank(P) + t \cdot (N - k)!$ for some k and t, then $p_{k+1}p_{k+2} \ldots p_N$ and $q_{k+1}q_{k+2} \ldots q_N$ have the same ordering.

- Since there are $N!$ permutations size N in total, if P and Q forms an ideal pair and P is smaller than Q, then $rank(Q) = rank(P) + \frac{N!}{2}$.

I- Ideal Permutation Pairing

Lemma

Let $P = p_1 p_2 p_3 \dots p_N$ and $Q = q_1 q_2 q_3 \dots q_N$ be permutations of size N. If $rank(Q) = rank(P) + t \cdot (N-k)!$ for some k and t, then $p_{k+1} p_{k+2} \dots p_N$ and $q_{k+1} q_{k+2} \dots q_N$ have the same ordering.

- Since there are $N!$ permutations size N in total, if P and Q forms an ideal pair and P is smaller than Q, then $rank(Q) = rank(P) + \frac{N!}{2}$.
- Because $\frac{N!}{2}$ can be written as $\frac{N \cdot (N-1)}{2} \cdot (N-2)!$. Thus, $p_3 p_4 \dots p_n$ and $q_3 q_4 \dots q_N$ have the same ordering.

I- Ideal Permutation Pairing

- To find the value of q_1 and q_2, think of it as sorting pairs of u, v where $1 \leq u \neq v \leq N$.

I- Ideal Permutation Pairing

- To find the value of q_1 and q_2, think of it as sorting pairs of u, v where $1 \leq u \neq v \leq N$.
- If p_1, p_2 has the rank of x, then:
  - $x = (p_1 - 1) \cdot (N - 1) + p_2$ if $p_1 < p_2$.
  - $x = (p_1 - 1) \cdot (N - 1) + (p_2 - 1)1$ if $p_1 > p_2$.

I- Ideal Permutation Pairing

- To find the value of q_1 and q_2, think of it as sorting pairs of u, v where $1 \leq u \neq v \leq N$.
- If p_1, p_2 has the rank of x, then:
  - $x = (p_1 - 1) \cdot (N - 1) + p_2$ if $p_1 < p_2$.
  - $x = (p_1 - 1) \cdot (N - 1) + (p_2 - 1)1$ if $p_1 > p_2$.
- The rank of q_1, q_2 is $x' = x + \frac{N \cdot (N-1)}{2} \mod N \cdot (N - 1)$, which, conversely, can be used to find the value of q_1, q_2 by:
  - $q_1 = 1 + \lfloor \frac{x'-1}{N-1} \rfloor$
  - $q_2' = 1 + (x' - 1) \mod (N - 1)$
    - $q_2 = q_2'$ if $q_1 < q_2'$.
    - $q_2 = q_2' + 1$ if $q_1 \geq q_2'$.

I- Ideal Permutation Pairing

- To find the value of $q_3 q_4 \dots q_N$, first you find the rank of $[p_3, p_4, \dots p_N]$ for each member among the list.

I- Ideal Permutation Pairing

- To find the value of $q_3 q_4 \dots q_N$, first you find the rank of $[p_3, p_4, \dots p_N]$ for each member among the list.
- Then, you find the rank for each member of $\{1, 2, 3, \dots, N\} - \{q_1, q_2\}$.

I- Ideal Permutation Pairing

- To find the value of $q_3 q_4 \dots q_N$, first you find the rank of $[p_3, p_4, \dots p_N]$ for each member among the list.
- Then, you find the rank for each member of $\{1, 2, 3, \dots, N\} - \{q_1, q_2\}$.
- For each i, find q_i such that $rank(q_i) = rank(p_i)$, this can be done in $O(1)$ with counting sort.

I- Ideal Permutation Pairing

- To find the value of $q_3 q_4 \dots q_N$, first you find the rank of $[p_3, p_4, \dots p_N]$ for each member among the list.
- Then, you find the rank for each member of $\{1, 2, 3, \dots, N\} - \{q_1, q_2\}$.
- For each i, find q_i such that $rank(q_i) = rank(p_i)$, this can be done in $O(1)$ with counting sort.
- Time complexity is $O(N)$.

J- Jewel Collection

- Problem Author: Mattanyu Tangngekkee
- Solved by 0 teams.

J- Jewel Collection

- **This is not a maximum weight edge cover problem.**
- **This is not a FLOW problem either.**
- Still, it is a graph problem.

J- Jewel Collection

- Since each jewel can only feature up to **two** colors. You can model a graph for this problem as follows:

J- Jewel Collection

- Since each jewel can only feature up to **two** colors. You can model a graph for this problem as follows:
  - Each color is a node.

J- Jewel Collection

- Since each jewel can only feature up to **two** colors. You can model a graph for this problem as follows:
  - Each color is a node.
  - Each jewel is an edge connecting two colors that it features. If the jewel only contains one color, then it is a self-loop on that color. The price is the weight of the edge.

J- Jewel Collection

- Since each jewel can only feature up to **two** colors. You can model a graph for this problem as follows:
  - Each color is a node.
  - Each jewel is an edge connecting two colors that it features. If the jewel only contains one color, then it is a self-loop on that color. The price is the weight of the edge.

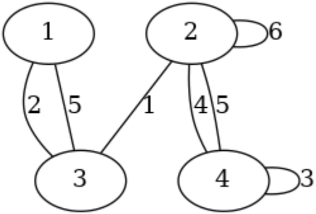


Figure: Graph of the first example.

J- Jewel Collection

- So what exactly do we want ?

J- Jewel Collection

- So what exactly do we want ?
- A valid collection will always form a maximal pseudoforest, a subgraph that spans every node and each component contains exactly one cycle.

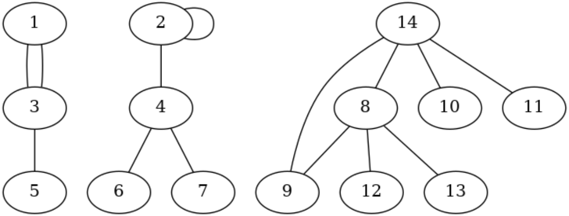


Figure: Example of a maximal pseudoforest.

J- Jewel Collection

- So what exactly do we want ?
- A valid collection will always form a maximal pseudoforest, a subgraph that spans every node and each component contains exactly one cycle.

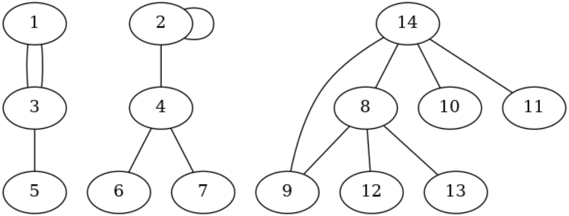


Figure: Example of a maximal pseudoforest.

- Therefore, you need to find a maximal pseudoforest with the maximum sum.

J- Jewel Collection

- It can be done similarly to Kruskal's algorithm for MST.

J- Jewel Collection

- It can be done similarly to Kruskal's algorithm for MST.
- But instead of each set contains a tree, allow them to contain at most one cycle.

J- Jewel Collection

- It can be done similarly to Kruskal's algorithm for MST.
- But instead of each set contains a tree, allow them to contain at most one cycle.
- It is impossible to make a collection if one of the connected component of the graph that is a tree.
- To print the matching, just do a simple DFS until you find a cycle.

J- Jewel Collection

- It can be done similarly to Kruskal's algorithm for MST.
- But instead of each set contains a tree, allow them to contain at most one cycle.
- It is impossible to make a collection if one of the connected component of the graph that is a tree.
- To print the matching, just do a simple DFS until you find a cycle.
- Time complexity is $O(N \log N)$

K- Kid Rally

- Problem Author: Attitarn Buathep
- Solved by 2 teams.
- First solved after 186 minutes.

K- Kid Rally

- Let us assume that Alice be the one starting at $(1, 1)$ and Bob starting at $(1, M)$.

K- Kid Rally

- Let us assume that Alice be the one starting at $(1, 1)$ and Bob starting at $(1, M)$.
- If Alice is at (a, b), it can be shown that if she decide to move forward optimally, her new X coordinate will increase by exactly 1. Same goes for Bob.

K- Kid Rally

- Let us assume that Alice be the one starting at $(1, 1)$ and Bob starting at $(1, M)$.
- If Alice is at (a, b), it can be shown that if she decide to move forward optimally, her new X coordinate will increase by exactly 1. Same goes for Bob.
- Generally speaking, until one of them stops moving, Alice's and Bob's move will increase one X coordinate at a time.

K- Kid Rally

- Let us assume that Alice be the one starting at $(1, 1)$ and Bob starting at $(1, M)$.
- If Alice is at (a, b), it can be shown that if she decide to move forward optimally, her new X coordinate will increase by exactly 1. Same goes for Bob.
- Generally speaking, until one of them stops moving, Alice's and Bob's move will increase one X coordinate at a time.
- Moreover, at the same X coordinate, Alice will have to be on the left side of Bob.

K- Kid Rally

- Let us define $DP[x]$ as the maximum points Bob can get when Alice has x points.

K- Kid Rally

- Let us define $DP[x]$ as the maximum points Bob can get when Alice has x points.
- For each coordinate X valued u, we update $DP[v + s[u][i]]$ with the value of $DP[v] + \max(s[u][i+1], s[u][i+2], ..., s[u][M])$ if it is higher than its current value.

K- Kid Rally

- Let us define $DP[x]$ as the maximum points Bob can get when Alice has x points.
- For each coordinate X valued u, we update $DP[v + s[u][i]]$ with the value of $DP[v] + \max(s[u][i+1], s[u][i+2], ..., s[u][M])$ if it is higher than its current value.
- Right now, time complexity should be $O(MaxValue \cdot N \cdot M)$, but we can reduce to $O(MaxValue \cdot N)$ by observing that there are at most only 10 values per coordinate X that we must consider.

K- Kid Rally

- There are some special cases. For example, considering this map grid size 2×2:

$$\begin{matrix} 9 & 1 \\ 9 & 0 \end{matrix}$$

K- Kid Rally

- There are some special cases. For example, considering this map grid size 2×2:

$$\begin{matrix} 9 & 1 \\ 9 & 0 \end{matrix}$$

- In this particular grid map, the optimal way is for Alice to stay at $(1, 1)$,earning a score of 9 while Bob moves from $(1, 2)$ to $(2, 1)$, earning a score of 10. Hence, the final score becomes 90.

L- Lulu and Friends

- Problem Author: Supakorn Kijwattanachai
- Solved by 23 teams.
- First solved after 6 minutes.

L- Lulu and Friends

- With very low constraints on string, almost any brute forces should work.

L- Lulu and Friends

- With very low constraints on string, almost any brute forces should work.
- For string matching, we can do two pointers technique.

L- Lulu and Friends

- With very low constraints on string, almost any brute forces should work.
- For string matching, we can do two pointers technique.
- We will maintain a pointer at string T and we will move the pointer until the character match with the current character in S, then we move the pointer on S and so on.

L- Lulu and Friends

- With very low constraints on string, almost any brute forces should work.
- For string matching, we can do two pointers technique.
- We will maintain a pointer at string T and we will move the pointer until the character match with the current character in S, then we move the pointer on S and so on.
- The minimum deletion is $|S| - (last - first)$ where *last* is the final position of pointer in the string T and *first* is the first position of pointer.

L- Lulu and Friends

- With very low constraints on string, almost any brute forces should work.
- For string matching, we can do two pointers technique.
- We will maintain a pointer at string T and we will move the pointer until the character match with the current character in S, then we move the pointer on S and so on.
- The minimum deletion is $|S| - (last - first)$ where *last* is the final position of pointer in the string T and *first* is the first position of pointer.
- This can be done in $O(|S| + |T|)$ and do it from every *first* matched position will result in $O(|T||S + T|)$ per query.

L- Lulu and Friends

- With very low constraints on string, almost any brute forces should work.
- For string matching, we can do two pointers technique.
- We will maintain a pointer at string T and we will move the pointer until the character match with the current character in S, then we move the pointer on S and so on.
- The minimum deletion is $|S| - (last - first)$ where *last* is the final position of pointer in the string T and *first* is the first position of pointer.
- This can be done in $O(|S| + |T|)$ and do it from every *first* matched position will result in $O(|T||S + T|)$ per query.
- Or you can generate all $2^{|T|}$ possible strings from deletion and memorize the answer.

M- Marriage Proposals

- Problem Author: Natapong Sriwatanasakdi
- Solved by 0 teams.

M- Marriage Proposals

- In this problem, the town can be visualized as a tree that have villages as vertices. Each bidirectional road between two villages is an edge.
- Each edge should contain the detail about the shop on the road — it can be described by the pearl type and the purchase limit.

M- Marriage Proposals

- For naive solution, We store the price of each pearl type in an array. For every type 2 event, we simply update the corresponding price in the array, which can be done in constant time.

M- Marriage Proposals

- For naive solution, We store the price of each pearl type in an array. For every type 2 event, we simply update the corresponding price in the array, which can be done in constant time.
- For each *type* 1 event, we can perform a breadth-first search (BFS) or depth-first search (DFS) from vertex x to find a path to vertex y.

M- Marriage Proposals

- For naive solution, We store the price of each pearl type in an array. For every type 2 event, we simply update the corresponding price in the array, which can be done in constant time.
- For each *type* 1 event, we can perform a breadth-first search (BFS) or depth-first search (DFS) from vertex x to find a path to vertex y.
- Let P be the set of edges along this path. Let e_i be edge i.
- For each pearl type j that appears on the path, total expense is calculated as

$$\sum_{e_i \in P, a_i = j} b_i \cdot c_j$$

M- Marriage Proposals

- Finding the expense for each type and determining the maximum expense among all types takes $O(N)$ time, where N is the number of vertices.

M- Marriage Proposals

- Finding the expense for each type and determining the maximum expense among all types takes $O(N)$ time, where N is the number of vertices.
- Therefore, the overall time complexity for all queries is $O(NQ)$

M- Marriage Proposals

- Finding the expense for each type and determining the maximum expense among all types takes $O(N)$ time, where N is the number of vertices.
- Therefore, the overall time complexity for all queries is $O(NQ)$
- We can optimize this approach by rooting the tree and letting each vertex store the path to its parent.

M- Marriage Proposals

- Finding the expense for each type and determining the maximum expense among all types takes $O(N)$ time, where N is the number of vertices.
- Therefore, the overall time complexity for all queries is $O(NQ)$
- We can optimize this approach by rooting the tree and letting each vertex store the path to its parent.
- To find a path from x to y, we can first determine the lowest common ancestor (LCA) of x and y, then traverse from x to the LCA and from y to the LCA.

M- Marriage Proposals

- Finding the expense for each type and determining the maximum expense among all types takes $O(N)$ time, where N is the number of vertices.
- Therefore, the overall time complexity for all queries is $O(NQ)$
- We can optimize this approach by rooting the tree and letting each vertex store the path to its parent.
- To find a path from x to y, we can first determine the lowest common ancestor (LCA) of x and y, then traverse from x to the LCA and from y to the LCA.
- While this improves the performance, the overall time complexity remains $O(NQ)$.

M- Marriage Proposals

- If the tree is a simple line, the problem can be efficiently solved using Mo's Algorithm with Updates. You can learn more about it at `https://codeforces.com/blog/entry/72690`.

M- Marriage Proposals

- If the tree is a simple line, the problem can be efficiently solved using Mo's Algorithm with Updates. You can learn more about it at `https://codeforces.com/blog/entry/72690`.
- In this scenario, we track the number of pearls available , the unit price, and the total price for each pearl type.

M- Marriage Proposals

- If the tree is a simple line, the problem can be efficiently solved using Mo's Algorithm with Updates. You can learn more about it at `https://codeforces.com/blog/entry/72690`.
- In this scenario, we track the number of pearls available , the unit price, and the total price for each pearl type.
- We also maintain the total expenses for all pearl types using a multiset, allowing us to easily update and query the maximum expense.

M- Marriage Proposals

- If the tree is a simple line, the problem can be efficiently solved using Mo's Algorithm with Updates. You can learn more about it at `https://codeforces.com/blog/entry/72690`.
- In this scenario, we track the number of pearls available , the unit price, and the total price for each pearl type.
- We also maintain the total expenses for all pearl types using a multiset, allowing us to easily update and query the maximum expense.
- This solution achieves a time complexity of $O(QN^{\frac{2}{3}}\log N)$.

M- Marriage Proposals

- For general cases, we can use the **Euler Tour Technique** (ETT) to flatten the tree.

M- Marriage Proposals

- For general cases, we can use the **Euler Tour Technique** (ETT) to flatten the tree.
- We perform DFS on the tree starting from any root vertex. Before starting, we initialize three variables: *turn*, *tout*, and *tour*.
  - *turn* : keeps track of the current DFS step.
  - *tout*[ ] : is an array that records the DFS exit time for each vertex.
  - *tour*[ ] : is a list that tracks the edges traversed during the DFS.

M- Marriage Proposals

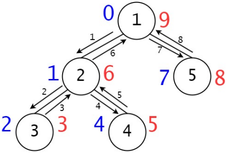
- Initially, *turn* is set to 0. Every time we enter or exit a vertex, we increment *turn*.

M- Marriage Proposals

- Initially, *turn* is set to 0. Every time we enter or exit a vertex, we increment *turn*.
- When moving from a parent to a child vertex, we append the corresponding edge to *tour*.

M- Marriage Proposals

- Initially, *turn* is set to 0. Every time we enter or exit a vertex, we increment *turn*.
- When moving from a parent to a child vertex, we append the corresponding edge to *tour*.
- Similarly, when returning from a child to its parent, we append the edge again and update *tout*[*child*] to the current value of *turn*.



DFS Order: 1→2→3→2→4→2→1→5→1

edges

1	2	3	4
(1,2)	(2,3)	(2,4)	(1,5)

tout

1	2	3	4	5
9	6	3	5	8

tour

1	2	3	4	5	6	7	8
1	2	2	3	3	1	4	4

Figure: DFS traversal of the example tree. Blue numbers represent the entry turn. Red numbers represent the exit turn.

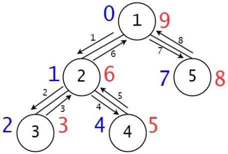
M- Marriage Proposals

- Consider the scenario where we want to travel from vertex x to vertex y such that $tout[x] < tout[y]$.

M- Marriage Proposals

- Consider the scenario where we want to travel from vertex x to vertex y such that $tout[x] < tout[y]$.
- Let $l = tout[x]$ and $r = tout[y]$. An edge e is part of the path **if and only if** it appears exactly once in the subarray $tour[l \dots (r-1)]$.

M- Marriage Proposals



DFS Order: 1→2→3→2→4→2→1→5→1

edges

1	2	3	4
(1,2)	(2,3)	(2,4)	(1,5)

tout

1	2	3	4	5
9	6	3	5	8

tour

1	2	3	4	5	6	7	8
1	2	2	3	3	1	4	4

- For example, when traveling from vertex 3 to vertex 5, we have $l = 3$ and $r = 8$.
- The path from vertex 3 to vertex 5 is $3 \to 2 \to 1 \to 5$, corresponding to the edges $2 \to 1 \to 4$.
- The subarray $tour[3 \ldots 7]$ is $[2, 3, 3, 1, 4]$.
- The edges that appear exactly once in this subarray are edges 2, 1, and 4, which match the set of edges in the actual path from vertex 3 to vertex 5.

M- Marriage Proposals

- Using this observation, we can now apply Mo's Algorithm with Updates.

M- Marriage Proposals

- Using this observation, we can now apply Mo's Algorithm with Updates.
- For each type 1 event, we query the edges between x and y by examining those that appear exactly once in $tour[l \ldots (r-1)]$, where $l = \min(tout[x], tout[y])$ and $r = \max(tout[x], tout[y])$.

M- Marriage Proposals

- Using this observation, we can now apply Mo's Algorithm with Updates.
- For each type 1 event, we query the edges between x and y by examining those that appear exactly once in $tour[l \ldots (r-1)]$, where $l = \min(tout[x], tout[y])$ and $r = \max(tout[x], tout[y])$.
- Extending this from the line graph case, we track the appearance count of each edge in the subarray to update pearl quantities and total expenses correctly.

M- Marriage Proposals

- Using this observation, we can now apply Mo's Algorithm with Updates.
- For each type 1 event, we query the edges between x and y by examining those that appear exactly once in $tour[l \ldots (r-1)]$, where $l = \min(tout[x], tout[y])$ and $r = \max(tout[x], tout[y])$.
- Extending this from the line graph case, we track the appearance count of each edge in the subarray to update pearl quantities and total expenses correctly.
- Time complexity is $O(QN^{\frac{2}{3}} \log N)$.

M- Marriage Proposals

- The $O(\log N)$ factor from using a Balanced Binary Search Tree (BBST) can slow down the solution, especially given the overhead of maintaining the BBST structure.

M- Marriage Proposals

- The $O(\log N)$ factor from using a Balanced Binary Search Tree (BBST) can slow down the solution, especially given the overhead of maintaining the BBST structure.
- To overcome this, we can replace the BBST with a data structure that supports updates in $O(1)$ and queries the maximum value in $O(\sqrt{\text{MaxValue}})$, where MaxValue is the maximum possible value in the set.

M- Marriage Proposals

- The $O(\log N)$ factor from using a Balanced Binary Search Tree (BBST) can slow down the solution, especially given the overhead of maintaining the BBST structure.
- To overcome this, we can replace the BBST with a data structure that supports updates in $O(1)$ and queries the maximum value in $O(\sqrt{\text{MaxValue}})$, where MaxValue is the maximum possible value in the set.
- In this problem, the total expense cannot exceed 10^7, since the unit prices are capped at 1,000 dollar and the purchase limits for each type of pearl do not exceed 10,000.

M- Marriage Proposals

- The new data structure uses two arrays:
  - *count*[ ] : Tracks the frequency of each expense value.
  - *blockCount*[ ] : Tracks the number of elements in blocks of size $\lceil\sqrt{\text{MaxValue}}\rceil$, where block b covers a range of values from $b\lceil\sqrt{\text{MaxValue}}\rceil$ to $(b+1)\lceil\sqrt{\text{MaxValue}}\rceil - 1$.

M- Marriage Proposals

- The new data structure uses two arrays:
  - *count*[ ] : Tracks the frequency of each expense value.
  - *blockCount*[ ] : Tracks the number of elements in blocks of size $\lceil\sqrt{\text{MaxValue}}\rceil$, where block b covers a range of values from $b\lceil\sqrt{\text{MaxValue}}\rceil$ to $(b+1)\lceil\sqrt{\text{MaxValue}}\rceil - 1$.
- When querying the maximum value, we first find the highest non-zero block in *blockCount*.

M- Marriage Proposals

- The new data structure uses two arrays:
  - $count[\,]$: Tracks the frequency of each expense value.
  - $blockCount[\,]$: Tracks the number of elements in blocks of size $\lceil\sqrt{\text{MaxValue}}\rceil$, where block b covers a range of values from $b\lceil\sqrt{\text{MaxValue}}\rceil$ to $(b+1)\lceil\sqrt{\text{MaxValue}}\rceil - 1$.
- When querying the maximum value, we first find the highest non-zero block in *blockCount*.
- After identifying the block, we search through *count* for the exact maximum value.

M- Marriage Proposals

- The new data structure uses two arrays:
  - *count*[ ] : Tracks the frequency of each expense value.
  - *blockCount*[ ] : Tracks the number of elements in blocks of size $\lceil\sqrt{\text{MaxValue}}\rceil$, where block b covers a range of values from $b\lceil\sqrt{\text{MaxValue}}\rceil$ to $(b+1)\lceil\sqrt{\text{MaxValue}}\rceil - 1$.
- When querying the maximum value, we first find the highest non-zero block in *blockCount*.
- After identifying the block, we search through *count* for the exact maximum value.
- Both the number of blocks and the elements within each block are bounded by $O(\sqrt{\text{MaxValue}})$.

M- Marriage Proposals

- The new data structure uses two arrays:
    - $count[\ ]$: Tracks the frequency of each expense value.
    - $blockCount[\ ]$: Tracks the number of elements in blocks of size $\lceil\sqrt{\text{MaxValue}}\rceil$, where block b covers a range of values from $b\lceil\sqrt{\text{MaxValue}}\rceil$ to $(b+1)\lceil\sqrt{\text{MaxValue}}\rceil - 1$.
- When querying the maximum value, we first find the highest non-zero block in $blockCount$.
- After identifying the block, we search through $count$ for the exact maximum value.
- Both the number of blocks and the elements within each block are bounded by $O(\sqrt{\text{MaxValue}})$.
- This allows our algorithm to run in $O(QN^{\frac{2}{3}} + Q\sqrt{\text{MaxValue}})$.

N- [N]ew YoRHa Security

- Problem Author: Nitit Jongsawatsataporn
- Solved by 0 teams.

N- [N]ew YoRHa Security

Lemma 1

For any prime number p if $\gcd(y, p-1) = 1$, then $f(a) = a^y \pmod{p}$ maps $\{1, 2, \ldots, p-1\}$ to $\{1, 2, \ldots, p-1\}$.

N- [N]ew YoRHa Security

Lemma 1

For any prime number p if $\gcd(y, p-1) = 1$, then $f(a) = a^y \pmod{p}$ maps $\{1, 2, \ldots, p-1\}$ to $\{1, 2, \ldots, p-1\}$.

Proof

Suppose not for contradiction. Then, there are $c, d \in \{1, 2, \ldots, p-1\}$ such that $c^y \equiv d^y \pmod{p}$. Let z be a natural number that $yz \equiv 1 \pmod{p}$ (exists by Bézout's identity). Raising both sides by z and using Fermat's little theorem yields

$$c \equiv (c^y)^z \equiv (d^y)^z \equiv d \pmod{p}$$

Thus, a contradiction.

N- [N]ew YoRHa Security

Lemma 2

If x, y are two natural number such that $\gcd(x, p-1) = \gcd(y, p-1)$, then $f(a) = a^x$ and $g(a) = a^y$ has the same image.

N- [N]ew YoRHa Security

Lemma 2

If x, y are two natural number such that $\gcd(x, p-1) = \gcd(y, p-1)$, then $f(a) = a^x$ and $g(a) = a^y$ has the same image.

Proof

We can write $a^x = (a^{\frac{x}{\gcd(x,p-1)}})^{\gcd(x,p-1)}$. The inner part preserves the whole domain, so the image is only affected by $gcd(x, p-1)$.

N- [N]ew YoRHa Security

Lemma 2

If x, y are two natural number such that $\gcd(x, p-1) = \gcd(y, p-1)$, then $f(a) = a^x$ and $g(a) = a^y$ has the same image.

Proof

We can write $a^x = (a^{\frac{x}{\gcd(x,p-1)}})^{\gcd(x,p-1)}$. The inner part preserves the whole domain, so the image is only affected by $gcd(x, p-1)$.

- We can solve the problem from these two lemmas by iterating the divisor of $p-1$ and computing the image of a^d. Then we can multiply their contribution and get the desired result.
- Time complexity $O(pd(p-1)\log p)$

N- [N]ew YoRHa Security

- For another solution, Since $(\mathbb{Z}_p^\times, \times) \cong (\mathbb{Z}_{p-1}, +)$, the problem reduces to:

Given a natural number n, compute the number of pairs (a, b) such that $0 \leq a, b \leq n-1$ and $ab \equiv k \pmod{p}$.

N- [N]ew YoRHa Security

- For another solution, Since $(\mathbb{Z}_p^\times, \times) \cong (\mathbb{Z}_{p-1}, +)$, the problem reduces to:

Given a natural number n, compute the number of pairs (a, b) such that $0 \leq a, b \leq n-1$ and $ab \equiv k \pmod{p}$.

- We can solve this easily by dividing the number between $0 \to n-1$ into groups based on their greatest common divisor with $p-1$.

N- [N]ew YoRHa Security

- For another solution, Since $(\mathbb{Z}_p^\times, \times) \cong (\mathbb{Z}_{p-1}, +)$, the problem reduces to:

Given a natural number n, compute the number of pairs (a, b) such that $0 \leq a, b \leq n-1$ and $ab \equiv k \pmod{p}$.

- We can solve this easily by dividing the number between $0 \to n-1$ into groups based on their greatest common divisor with $p-1$.
- Elements from the same group will have the same answer, and we can compute from their class instead.

N- [N]ew YoRHa Security

- To convert back, we need a map between $(\mathbb{Z}_{p-1}, +)$ to $(\mathbb{Z}_p^\times, \times)$.

N- [N]ew YoRHa Security

- To convert back, we need a map between $(\mathbb{Z}_{p-1}, +)$ to $(\mathbb{Z}_p^\times, \times)$.
- Since both are a cyclic group, we can convert elements if we can find one generator (primitive root, in number theory language).

N- [N]ew YoRHa Security

- To convert back, we need a map between $(\mathbb{Z}_{p-1}, +)$ to $(\mathbb{Z}_p^\times, \times)$.
- Since both are a cyclic group, we can convert elements if we can find one generator (primitive root, in number theory language).
- If p is the generator, then we can map $(\mathbb{Z}_{p-1}, +)$ to $(\mathbb{Z}_p^\times, \times)$ by $a \mapsto p^a$.

N- [N]ew YoRHa Security

- To convert back, we need a map between $(\mathbb{Z}_{p-1}, +)$ to $(\mathbb{Z}_p^\times, \times)$.
- Since both are a cyclic group, we can convert elements if we can find one generator (primitive root, in number theory language).
- If p is the generator, then we can map $(\mathbb{Z}_{p-1}, +)$ to $(\mathbb{Z}_p^\times, \times)$ by $a \mapsto p^a$.
- To find primitive root, you can simply check each element whether it generates the group or not.
- Time complexity is $O(p \log p + d(p-1)^2) = O(p \log p)$.