Rapid (`rapid`)

Author: Alexandru Gheorghies

Developer: Alexandru Gheorghies

Solution

The main idea of the solution is to calculate the positions of the two largest elements for every prefix of the permutation.

Trivially, the positions of the two largest elements from the prefix $[p_1, p_2]$ are 1 and 2.

Now, let's consider 3 indices i, j and k, where i and j are the positions of the two largest elements from the prefix $[p_1, \ldots, p_{k-1}]$.

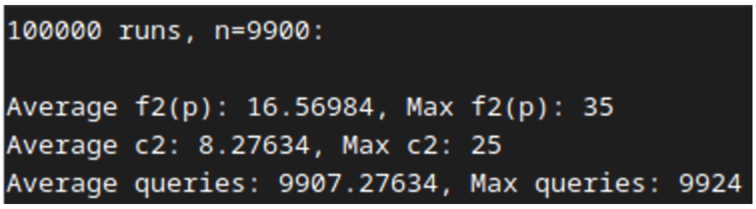
Suppose that $\min(p_i, p_j) = x$ and $\min(p_i, p_k) = y$. There are three cases:

- If $x < y$, then $p_j < p_i$ and $p_j < p_k$. As such, the positions of the two largest elements from the prefix $[p_1, \ldots, p_k]$ will be i and k.
- If $x = y$, then $p_i < p_j$ and $p_i < p_k$. As such, the positions of the two largest elements from the prefix $[p_1, \ldots, p_k]$ will be j and k. Since the value of $min(p_j, p_k)$ is not yet known, an extra query is needed to determine its value.
- If $x > y$, then $p_k < p_i$ and $p_k < p_j$. As such, the positions of the two largest elements from the prefix $[p_1, \ldots, p_k]$ remain i and j.

The total number of queries is $n - 1 + c_2$, where c_2 is the number of times the second case occurs.

Note that c_2 is at most equal to $f_2(p)$, where $f_2(p)$ represents the number of times the value of the second largest element changes throughout all of the prefixes of the permutation p.

If p is the identity permutation, then $c_2 = f_2(p) = n - 2$, which is the worst possible case for this solution. However, if the permutation p is randomly generated, then $f_2(p)$ (and by extension c_2) will be very small:



```
100000 runs, n=9900:

Average f2(p): 16.56984, Max f2(p): 35
Average c2: 8.27634, Max c2: 25
Average queries: 9907.27634, Max queries: 9924
```

Therefore, if the order in which the permutation is traversed in is randomized, then the probability that the total number of queries used by this solution exceeds $n + 100$ is astronomically small.

Time complexity per test case: $O(N)$

Craiova (craiova)

Author: Alexandru Gheorghies

Developer: Alexandru Gheorghies

Solution

We can manually compute the answer for $n \leq 5$. From now on, we will assume $n \geq 6$.

The first observation is that if we want a certain group to win, we will try to minimize the number of people remaining to the left and the right of that group, as they will have to brawl with our group regardless.

The left side can be treated in a straight-forward manner, because we can always reach a state where only 0 or 1 person remains, depending on the parity of the sum of numbers on that prefix.

Computing the minimum number of people left on the right side is not that trivial, but we can observe that we don't really need to minimize that number, but rather make it at most $x-2$, if we want position x to win. (we will treat positions 1 and 2 separately).

We are left with the following problem: Given a list of consecutive numbers that start at a number a, determine if we can reach a state where only a number less than or equal to $a-3$ remains (considering $a = x+1$). It is obvious that if the length of the list is even, we can always have 0 or 1 remaining, so we are only interested in odd lengths.

Proposition 1: It is possible to reach a number less than or equal to $a-3$ if and only if the length of the list is even, or if it is greater than or equal to 5.

Proof: It is possible to reach $a-4$ starting from the list $[a,\ a+1,\ a+2,\ a+3,\ a+4]$:

$$[a, a+1, a+2, a+3, a+4] \to [a, a+1, 1, a+4] \to [a, a, a+4] \to [a, 4] \to [a-4]$$

For odd length lists, adding an even amount of numbers to the right can only decrease our answer or make it equal to 1. If the length is 1 it's obvious we can't make it any smaller, and for a length of 3 the best we can achieve is $a-1$. Thus, we have proven the proposition for any $a \geq 4$ and since we said that we will deal with $x = 1$ and $x = 2$ separately, our proof is complete.

The above statement tells us that any x except 1, 2, $n-3$ and $n-1$ is always a winning position that needs to be counted towards our answer. That is because we can always reach a state of the form y, x, z with $y \leq 1$ and $z \leq x-2$. For $n-3$ and $n-1$ we cannot make the number to their right sufficiently small, so they will never be counted towards our answer.

Proposition 2: If $n \equiv 1 \ (mod\ 4)$ or $n \equiv 2 \ (mod\ 4)$ positions 1 and 2 are winning, otherwise they are not.

Proof: If $n \equiv 3 \ (mod\ 4)$ or $n \equiv 0 \ (mod\ 4)$, the sum of numbers to the right of 1 and 2 is odd, so the minimum value achievable on the right side is 1, thus making it impossible for both 1 and 2 to win. Let's analyze the other 2 cases for position 1:

- If $n \equiv 1 \ (mod\ 4)$ we are left with a list of length divisible by 4 to the right of 1, which can always be reduced to 0.

- If $n \equiv 2 \ (mod\ 4)$ we are left with the number 2, 4 consecutive numbers, and a list of length divisible by 4. All the numbers to the right of the number 2 will be paired to form ones, the first 2 ones will be subtracted from the number 2 and the other ones will cancel eachother, leaving the number 0.

The cases for position 2 will be treated as follows:

- If $n \equiv 2 \ (mod\ 4)$ we are left with a list of length divisible by 4 to the right of 2, which can always be reduced to 0.
- If $n \equiv 1 \ (mod\ 4)$, since we assumed that $n \geq 6$, we now have $n \geq 9$. We will group all the numbers to the right of 3 into pairs that make the number 1, the first three pairs will be subtracted from 3 and the other pairs (there will be an even number of these) will cancel eachother, thus leaving the number 0 overall.

We have now proven everything, so the answer for any $n \geq 6$ is $n - 2$ if $n \equiv 1 \ (mod\ 4)$ or $n \equiv 2 \ (mod\ 4)$ and $n - 4$ otherwise.

Time complexity per testcase: $O(1)$
Memory complexity: $O(1)$

Bucuresti, Steaua Bucuresti (Steaua)

Author: Alin-Gabriel Raileanu

Developer: Alin-Gabriel Raileanu

Solution

We consider the scores obtained by Steaua in the last N years as an array A with N elements, indexed from 1 to N.

Regardless of the chosen approach, we first need to construct two auxiliary arrays, st and dr, with the following properties:

- $dr[i]$ $(1 \leq i \leq N)$: the smallest index j $(j > i)$ such that $A[j] > A[i]$ (or $N+1$ if no such index exists);
- $st[i]$ $(1 \leq i \leq N)$: the largest index j $(j < i)$ such that $A[j] > A[i]$ (or 0 if no such index exists).

This part of the solution can be computed in $O(N)$ using a classical stack-based approach.

Now, for each index i $(1 \leq i \leq N)$, we need to determine the minimum index j $(j \geq i)$ such that the subarray (i, j) in A has exactly K_1 changes of maximum from left to right.

For this part, we can use two different approaches:

- $O(N \cdot \log(K_1))$ solution: We construct a two-dimensional array bl, defined as follows:
  - $bl[i][pw]$: the minimum index such that the subarray $(i, bl[i][pw])$ has $2^{pw} + 1$ changes of maximum from left to right;
  - Initialization: $bl[i][0] = dr[i]$;
  - Recurrence relation: $bl[i][pw] = bl[bl[i][pw-1]][pw-1]$ $(1 \leq pw \leq \log(K_1))$.

  Both the time and space complexity for constructing this table are $O(N \cdot \log(K_1))$.

  To obtain the desired results, we can apply a **binary lifting** technique.

  For each i, we iterate in decreasing order over the powers of 2 in the table bl, updating the index at each step, until the sum of these powers equals K_1.

  Since this process can be implemented in $O(\log(K_1))$ and is applied for each index from 1 to N, the final complexity is $O(N \cdot \log(K_1))$.

- $O(N)$ solution: We construct a tree with $N+1$ nodes, numbered from 1 to $N+1$, with the root at node $N+1$, as follows:
  - The parent of each node i $(1 \leq i \leq N)$ is node $dr[i]$.

  Now, we observe that the desired result for each index i $(1 \leq i \leq N)$ is equivalent to finding the K_1-th ancestor of node i in this tree.

  This process can be solved in $O(N)$ for all nodes by maintaining the path from the root to the current node in a stack, during a DFS traversal.

Similarly, we need to find the maximum index k such that the subarray (k, i) has K_2 changes of maximum from right to left. In this case, one of the above methods can be applied to the reversed vector A.

The purpose of these results is to compute the arrays kst and kdr, with the following properties:

- $kst[i]$ $(1 \le i \le N)$: the smallest index j $(1 \le j \le i)$ such that $\max_{k=j}^{i} A[k] = A[i]$, and the subarray (j, i) has K_1 changes of maximum from left to right. ($N + 1$ if no such index exists);
- $kdr[i]$ $(1 \le i \le N)$: the largest index j $(i \le j \le N)$ such that $\max_{k=i}^{j} A[k] = A[i]$, and the subarray (i, j) has K_2 changes of maximum from right to left. (0 if no such index exists).

For the subtask where all numbers are distinct, this step can be directly computed using the method explained above.

However, if the numbers are not distinct, the approach does not always lead to a correct solution.

In this case, we need to propagate the values of kst and kdr as follows:

- For each i $(1 \le i \le N)$, we propagate the result from $kst[i]$ to all positions j $(i \le j < dr[i])$ where $A[i] = A[j]$;
- For each i $(1 \le i \le N)$, we propagate the result from $kdr[i]$ to all positions j $(st[i] < j \le i)$ where $A[i] = A[j]$.

The final answer to the problem is:

$$\max_{i=1}^{N} (kdr[i] - kst[i] + 1).$$

Time Complexity: $O(N)$ or $O(N \cdot \log(N))$, depending on the implementation.

Space Complexity: $O(N)$ or $O(N \cdot \log(N))$, depending on the implementation.

Dinamo (dinamo)

Author: Alexandru Gheorghies

Developer: Alexandru Gheorghies

Solution

This problem can be solved using dynamic programming.

Let $dp[m][mask]$ be the number of ways to tile the first m columns, such that the bitmask of the tiles from the m-th column which can be extended to the next column (that is, they are either horizontal or single tiles) is equal to $mask$.

Let $f(n)$ be the number of ways to tile a $n \times 1$ surface by using only vertical tiles. $f(n)$ is computed in the following way:

$$f(n) = \begin{cases} 1, & \text{if } n = 0 \\ 0, & \text{if } n = 1 \\ \sum_{i=2}^{n-2} f(i), & \text{if } n \geq 2 \end{cases}$$

Additionally, let $cnt(mask)$ be the number of ways to tile the uncontinuable cells (i.e. the 0-bits) from a column whose bitmask is equal to $mask$, by using vertical tiles.

If the lengths of the contiguous subsequences of 0-bits from $mask$ are equal to $l_1, l_2, \ldots, l_k$, then:

$$cnt(mask) = \prod_{i=1}^{k} f(l_i)$$

Now, it is possible to write the recurrence relation of the main dp:

$$dp[m][mask] = \sum_{mask2=0}^{2^n-1} dp[m-1][mask2] \cdot cnt(mask2) \cdot 2^{\text{popcount}(mask \& mask2)}$$

where & denotes the bitwise AND operation.

The naive computation of the above recurrence relation yields an $O(4^n \cdot m)$ solution, which can be optimized to $O(8^n \cdot \log(m))$ by using matrix exponentiation.

However, for most bitmasks, $cnt(mask) = 0$ (i.e. $mask$ contains a 0-bit which is not adjacent to other 0-bits). As such, for these bitmasks, $dp[m][mask] = 0$ for all values of m. Therefore, these masks can be safely ignored from all computations.

For $n = 10$, there are exactly 200 bitmasks for which $cnt(mask) \neq 0$, meaning that the time complexity for this problem becomes $O(200^3 \cdot \log(m))$

CFR Cluj (`cfr`)

Author: Alexandru Gheorghies

Developer: Alexandru Gheorghies, Gabriel Turbinca

Solution

The main observation required to solve this problem is that the vertical line and the horizontal line from the *plus* with the highest number of intersections can be calculated independently from each other.

As such, the answer to a query `3` x_P y_P can be computed as follows:

- The optimal position for the vertical line is either $x = x_P - \epsilon$ or $x = x_P + \epsilon$, where ϵ is an arbitrarily small positive real number. The first of these lines intersects with all segments (PP') where $x_{P'} < x_P$, while the second line intersects with all segments (PP') where $x_{P'} > x_P$.
- Similarly, the optimal position for the horizontal line is either $y = y_P - \epsilon$ or $y = y_P + \epsilon$, where ϵ is an arbitrarily small positive real number. The first of these lines intersects with all segments (PP') where $y_{P'} < y_P$, while the second line intersects with all segments (PP') where $y_{P'} > y_P$.

This means that the problem statement can be rewritten as follows:

Find a data structures that supports the following operations:

1. Insert a point (x, y).
2. Erase a point (x, y).
3. Given a point (x_P, y_P), compute the following values:
   - s — the number of points in the data structure.
   - a — the number of points P' in the data structure for which $x_{P'} < x_P$.
   - b — the number of points P' in the data structure for which $y_{P'} < y_P$.

   and print $\max(a, s - a - 1) + \max(b, s - b - 1)$.

A data structure that supports these operations consists of two ordered_sets, one for the x coordinates, and another for the y coordinates.

Note that the ordered_sets can be replaced by other data structures, including bitwise tries, treaps, and even segment trees or fenwick trees after mapping the coordinates to the range $[1, Q]$, while preserving their relative order.

Total time complexity: $O(Q \cdot \log(Q))$

Halftime Changes (halftime)

Author: Alin-Gabriel Raileanu

Developer: Alin-Gabriel Raileanu

Solution

There are multiple ways one can modify the array such that the number of nodes for which the condition holds is big enough, here we will show one of them:

We assign numbers in the following way:

- $A_1 = 2^{\lfloor \log(n) \rfloor + 1} - 1$
- $A_i = 2^{height_i - 1}$, if $\text{height}_i \leq \lfloor \log(n) \rfloor + 1$ where $height_i$ is equal to the distance from node i to the root, and $i \neq 1$

Because each level of the tree contains at most 100 nodes, we will end up doing at most $k = 100(\lfloor \log(n) \rfloor + 1) + 1$ operations, which is good enough to fit our constraints.

First we will prove that, by doing these operations, we will obtain enough nodes which satisfy the condition shown in the statement. If we analyze the nodes i with $\text{height}_i > \lfloor \log(n) \rfloor + 1$ we can observe that the path from node i to the root contains all 2^j with $0 \leq j \leq \lfloor \log(n) \rfloor$ and the number $2^{\lfloor \log(n) \rfloor + 1} - 1$, which is at least equal to n. Because of that, we can obtain any number $x \leq n \leq 2^{\lfloor \log(n) \rfloor + 1} - 1$, as we select the root as the number which we subtract from and the nodes which contain the bits that do not appear in x. As $B_i \leq n$, we can obtain any such B_i, so, as a result, the condition holds. But, these nodes i are exactly those for which we have not modified their A_i values earlier, and as the number of necessary nodes is equal to $n - k$, where k is the maximum amount of nodes which we can modify, we will have enough nodes for which the condition holds, as we will select all such nodes.

U Cluj (`ucluj`)

Author: Bogdan-Ioan Popa

Developer: Bogdan-Ioan Popa

Solution

Let $F(N, K)$ be the number of ways to write N as a sum of K powers of 2.

We can see that the problem asks us to compute $F(\frac{N+K}{2}, K)$ for every $1 \leq K \leq P$, where $(N + K) \mod 2 = 0$. If $(N + K) \mod 2 = 1$, then the answer is 0. The recurrence relation is $F(N, K) = F(N - 1, K - 1) + F(\frac{N}{2}, K)$ if N is even, or $F(N, K) = F(N - 1, K - 1)$ if N is odd.

The number of distinct states the recurrence visits is $O(P^2 logN)$.

In order to not compute the same state twice, we need to use some memoization technique. Impelementing hash tables by hand should give you 100 points.