Problem Tutorial: "Factory Balls"

We would like to apply a breadth-first search, but there are too many states to run a BFS on: each region has one of the K colors, and each piece of equipment is either equipped or unequipped, leading to $K^N 2^M$ states in total.

However, if the color of a certain region does not match the target color, then the exact color does not matter. All we need for each region is whether the color matches the target or not. Therefore each region has only two "colors." This reduces the number of states to 2^{N+M} and the time complexity of the BFS to $O(2^{N+M}(KN+M))$. To represent each state, we can use a pair of bitmasks (or a single bitmask if you combine them).

Using bitwise operators more cleverly, We can do even better by immersing the ball into a paint can in $O(1)$ time, leading to $O(2^{N+M}(K+M))$.

Problem Tutorial: "UCP-Clustering"

The RC of two clusters uniquely defines the set of next RC of two clusters, and such next RC can be computed in $O(N)$ time by computing the median in linear time. Therefore, the states and their transition takes the shape of a functional graph, where every node has an outdegree of one. Since the algorithm is guaranteed to terminate, every initial node will eventually reach a vertex that self-loops. The expected value of the iteration count can be considered as the average distance from all initial states that can reach the current loop. Since the graph takes the shape of a tree, this can be easily computed.

As a result, we can obtain a conceptually simple (though possibly tedious to implement) $O(MN)$ solution where M is the number of states. Naively, $M = O(N^8)$, since there are $O(N^2)$ possible values that each coordinate can take. However, we show how to obtain a much tighter upper bound.

Draw a line between the current RCs, and consider the perpendicular bisector. The bisector divides the plane into two halfplanes. Then, the points belonging in each halfplane are the clusters in the next iteration. As a result, for all states with in-degree greater than zero, there exists a dividing line between each cluster.

Suppose that two clusters have a dividing line. Then we can rotate the line until it hits the points from both clusters. The lines touch two points from the input and define a unique cluster. As a result, there are at most $O(N^2)$ states with in-degree greater than zero. States with in-degree zero are exactly the valid initial state, of which there are $N(N-1)/2$ possible candidates.

As a result, we obtain an $O(N^3)$ algorithm.

If you want a challenge, you can try to optimize the above algorithm to $O(N^2 \log N)$.

Problem Tutorial: "Triple Sword Strike"

Let's assume that we perform at least two sword strikes parallel to the x-axis. The other case can be handled by reflection along the line $x = y$.

For the case where three sword strikes are parallel to the x-axis, maintain the array $count[y]$ that denotes the sum of values for all monsters with y-coordinate y. You can see that the answer is the sum of the top three elements.

For the case where two sword strikes are parallel to x-axis, we will fix the x-coordinate of one sword strike which runs parallel to y-axis. Let this value p, and let S_p be the set of monsters with x-coordinate x.

If we decrease the $count[y]$ value for the monsters in S_x, we can simply pick the top two elements in the array. However, we need to recompute the top two elements every time, which results in $O(N^2)$ time complexity if done naively.

Construct a list that maintains all possible indices of y, sorted by decreasing order of $count[y]$. Let's say we decreased the value $count[y]$ for elements in S_x. You can observe that the top two elements are one of the top $|S_x| + 2$ elements in the list. The list contains at least 2 elements that have their $count[y]$ unchanged, so the following elements must have their value not larger than those two elements.

In conclusion, for each x coordinate, you can compute the top two in $O(|S_x|)$ time. This results in an $O(N)$ time algorithm except for the sorting of $count[y]$ value, which can be also done in $O(N)$ time with counting sort.

Problem Tutorial: "RPS Bubble Sort"

Suppose that there are exactly two distinct characters in the string. In this case, Yihwan's game actually tries to sort a string where the losing character comes before the winning character.

Here, it is helpful to analyze from the perspective of the losing character. In each pass of the game, the losing character will move left by one if there is a preceding winning character and not move otherwise. In conclusion, if the k-th losing character was in the position i, it will be at position $\max(i - T, k)$ after T iterations of the game. Using this observation, this special case can be solved in $O(N)$ time.

Now let's go back to the general case. Partition the string by repeatedly cutting the longest prefix of the string that has at most two distinct characters. Here are the key observations for this problem.

Observation 1. In the first pass of the game, no character is swapped across partitions.

Proof. Let A be a winning character, and B be a losing character in the first partition. Since the partition did not extend further, the first character of the second partition is C. After the first pass swapped all characters until the first partition, the last character of the first partition is A. A can beat B, but it loses to C. Thus, the swap does not happen. By induction, you can show that this holds for all subsequent partitions.

Observation 2. In any pass of the game, no character is swapped between different partitions.

Proof. The first pass leaves a losing pair at the border of different partitions, and this does not change after further passes.

These observations reduce the problem to the two-character case. The time complexity is $O(N)$.

Problem Tutorial: "Stones 1"

If a contiguous segment of stones contains only one color, you can gain points from at most one of the stones. Partition the array into maximal contiguous monochromatic segments, and only leave the stones with the largest weight for each segment. After this procedure, you can assume that adjacent stones have different colors. In other words, the color of the stones alternates.

Let's see how many times we can gain points from removing the stones. Assume $N \geq 3$ since otherwise, you can gain no points.

You can not gain any points from the leftmost and rightmost stones. If you remove the stones in the middle, you can only possibly gain points from one of the adjacent stones. Repeatedly applying this argument, observe that you gain the points from at most $\lceil (N-2)/2 \rceil$ stones.

Mark any $\lceil (N-2)/2 \rceil$ stones of your choice that are not the leftmost or rightmost stones. There is a strategy that enables you to get points that are at least the sum of points of all marked stones.

We will show this by induction. If $N \leq 3$, the claim is trivial. Otherwise, there exists a pair of adjacent stones such that one of them is marked, and the other is unmarked. If you take the marked stone, then the adjacent stones are merged into a single stone with a point maximum from both, and one of the adjacent stones is not marked. If the other one was marked, mark the new stone. Otherwise, do not mark the new stone.

By inductive argument, there is a strategy that enables you to get points that are at least the sum of points of all marked stones. If the previously marked stone had a larger point than the other, this exactly gives what we want, otherwise, this gives slightly more than what we want. ■

In conclusion, there is a strategy that enables you to gain points that are at least the sum of the top $\lceil (N-2)/2 \rceil$ stones. Obviously, you can't get more points than that. The maximum possible point can be calculated by sorting all points. The time complexity is $O(N \log N)$.

Problem Tutorial: "Stones 2"

Each action of obtaining points from a stone can be described as a triplet of integers $1 \le i < j < k \le N$, where you gain A_j points by removing a stone j which is adjacent to the stone i in the left, and stone k in the right. The color of the stone (S_i, S_j, S_k) should be either `('W', 'B', 'W')` or `('B', 'W', 'B')`. We will only consider the former case as the other can be solved symmetrically.

To obtain a situation where two stones i, k are adjacent to j when j is being removed, all elements $\{i+1, \ldots, j-1, j+1, \ldots, k-1\}$ should be removed before the j's removal, and i, k should be removed afterwards. It can be shown, that the number of permutation which satisfies these conditions is:

$$2\frac{(k-i-2)!}{(k-i+1)!}n!$$

Let this quantity be $f(k-i)$, and let W_i be an indicator which is 1 if $S_i =$ `'W'` and 0 otherwise, We can obtain an $O(n^3)$ algorithm which precomputes all values $f(k-i)$ and computes the following value.

$$\sum_{1 \le i < j < k \le N} W_i(1-W_j)W_k \times A_j \times f(k-i)$$

Take a prefix sum on $A_j(1-W_j)$, and denote this value Sum_i. We can rewrite the above formula as

$$\sum_{1 \le i < k \le N} W_i W_k \times (Sum_k - Sum_i) \times f(k-i)$$

which is

$$\sum W_i W_k \times Sum_k \times f(k-i) - \sum W_i W_k \times Sum_i \times f(k-i)$$

which can be computed with two convolutions. By using FFT, you can obtain an $O(N \log N)$ algorithm.

Problem Tutorial: "Beacon Towers"

Let the village i be *important* if $h_j < h_i$ for all $1 \le j < i$. You can make the following two observations.

Observation 1. If a segment contains no important villages, then the division does not satisfy the rules.

Proof. For a beacon installed in such segment, there exists a preceding beacon that is higher.

Observation 2. If all segments contain at least one important village, then the division satisfies the rules.

Proof. Every segment will install a beacon on one of the important villages. The subset of important villages has increasing heights.

Let the indices of important villages be $i_1 < i_2 < \ldots < i_k$. Such sequence can be computed in $O(N)$ time.

Consider the problem now as putting the barriers between two adjacent villages $i, i+1$ if we want to put them in different segments. By the above observation, we can install at most one barrier between two important villages. The answer is therefore $\prod_{j=1}^{k-1}(i_{j+1} - i_j + 1)$.

Problem Tutorial: "Exam"

Apply meet-in-the-middle along the minor diagonal of the grid ($i + j = N + 1$). There are 2^{N-1} ways to reach some cell in the diagonal from $(1, 1)$, and 2^{N-1} ways to reach (N, N) from some cell in the diagonal. Enumerate all such paths and store the following two values:

- The maximum subarray sum of the elements of the path (x_i).
- The maximum non-empty suffix or prefix sum of the elements of the path (y_i). For a path from $(1, 1)$, we are interested in the suffix, and for a path to (N, N), we are interested in the prefix.

Let U_x be the set of paths from $(1,1)$ to $(x, N+1-x)$ and D_x be the set of paths from $(x, N+1-x)$ to (N,N). For each cell $(x, N+1-x)$, we want to count the number of pairs $i \in U_x, j \in D_x$ such that $\max(x_i + x_j, y_i, y_j) = K$. This is equal to the number of pairs where $\max(x_i + x_j, y_i, y_j) \le K$ minus the number of pairs where $\max(x_i + x_j, y_i, y_j) \le K - 1$.

Let's count the number of pairs with $\max(x_i + x_j, y_i, y_j) \le K$. If you ignore all paths with $y > K$, the problem is reduced to the counting the number of pairs with $x_i + x_j \le K$. This can be solved with sorting and two pointers. The total time complexity is $O(2^N \times N)$.

Problem Tutorial: "Short Question"

As a prerequisite, let's compute the value $\sum_{i=1}^{N}\sum_{j=1}^{N}|p_i - p_j|$. By sorting a sequence p, you can get rid of the absolute value and simply compute $2\sum_{i=1}^{N}\sum_{j=1}^{i-1}(p_i - p_j)$. Each i is added for $i-1$ times and subtracted for $(N-i)$ times, so the answer is $2\sum_{i=1}^{N}(2i - N - 1)p_i$. Let's denote this value as a function of sequence p as $value(p)$.

Note that $min(a,b) = a + b - max(a,b)$, and $max(|p_i - p_j|, |q_i - q_j|)$ can be interpreted as a Chebyshev distance between two point (p_i, q_i) and (p_j, q_j). It is known that the Chebyshev distance is equivalent to the Manhattan distance if the point set are rotated by 45 degrees. In other words, $max(|p_i - p_j|, |q_i - q_j|) = \frac{1}{2}(|p_i + q_i - p_j - q_j| + |p_i - q_i - p_j + q_j|)$. If we create a auxiliary sequence $a_i = p_i + q_i, b_i = p_i - q_i$, this value simply becomes $\frac{1}{2}(value(a) + value(b))$.

In conclusion, the answer is $value(p) + value(q) - \frac{1}{2}(value(a) + value(b))$. The time complexity of this solution is $O(N \log N)$.

Problem Tutorial: "Four Cards"

We can just go through all $4! = 24$ permutations and check directly if the condition is held. With at most 5000 test cases the time limit is big enough to allow that solution to go.

Problem Tutorial: "Decimal Expression"

Let's prove the fact that you shall do not use any signs to achieve the maximal result.

First, consider that if we will replace all '-' with '+' then the value will not decrease. So we can consider only additions and multiplications.

1. For the one-digit integer the fact is trivial (we just do not have the place for the signs).
2. Assume we have the fact proven for all integers less than k-digit. Consider k-digit integer.

   Let the maximal solution have the sign somewhere. If it is addition, then we have two blocks V_l of length $l < k$ and V_r of length $r < k$ digits each $(l + r = k)$ and the value is $V_l + V_r$, if it is multiplication, then we have $V_l \cdot V_r$. By our assumption, V_l and V_r maximize when no operation signs used, so we can replace V_r and V_l with the blocks B_l and B_r without the operation signs, and the sum or product does not decrease.

   But if we will use no sign between B_l and B_r, we will have the result $B_l \cdot 10^r + B_r$. $B_r < 10^r$, so $B_l \cdot B_r < B_l \cdot 10^r \le B_l \cdot 10^r + B_r$, and $B_l + B_r \le B_l \cdot 10^r + B_r$, so without any sign of operation used the answer for length k is maximal.

So all we need is to read the given string and print it as it is.

Problem Tutorial: "Two Princes"

Removal of the edge connecting the vertex i and its parent vertex splits the tree into 2 parts, one of those parts is a subtree with the root in the vertex i, where all vertices of degree 1 (except for the root) still are the sea ports.

So we can run DFS from the root, with the following modification: when we enter the vertex u, then $ports[u] = 1$, when we return from some vertex v, then $ports[u] = ports[v] + ports[v]$. When we leave u, we decrease $ports[u]$ by 1, if it is greater than 1.

The array $ports[i]$ contains number of the sea ports in the subtree with the root i. So we can iterate through all i vertices and find the maximal value of $min(ports[i], ports[1] - ports[i])$, that will be the answer.

Problem Tutorial: "Rock, Paper, Scissors"

Consider three integers: $F(S)$ — number of Scissors in the sequence, $F(R)$ — number of Rocks, $F(P)$ — number of Papers. Let's sort them in non-decreasing order and renumerate as $F1$, $F2$, $F3$.

If $F1 < F2$ and $F1 < F3$, then answer is 0: we are using the move that is beaten by the letter, corresponding to $F1$. Then we have $F1$ losses, and $F3$ (or $F2$) wins. Both $F3$ and $F2$ are greater than $F1$, so Alice wins.

If $F1 < F3$ and $F2 < F3$, then answer is 0: we are using the move that beats the letter, corresponding to $F3$. Then we have $F3$ wins, and $F1$ (or $F2$) loses. Both $F1$ and $F2$ are less than $F3$, so Alice wins.

And the only remaining case if that $F1 = F2 = F3$. Then spamming any move gives a tie, so we need at least one change: start the game with same character as Bob and after the first turn we have the same problem but where the equality is not held (and we can win it without change). So the answer is 1.