Problem A. Count Special

Subtask 2

N is small enough that we can check every number x from 1 to $N-1$, incrementing a counter whenever x is special. We then output the final value of the counter.

Subtask 3

N is huge now, so the loop in Subtask 2 will time out.

Instead, we notice that there are $\lfloor (N-1)/A \rfloor$ numbers below N that are divisible by A, and $\lfloor (N-1)/B \rfloor$ numbers below N that are divisible by B. We can add these together to find how many numbers are divisible by A or B.

However, now we've counted numbers divisible by both A and B twice. So we need to subtract the number of numbers divisible by both A and B. For a number to be divisible by both A and B, it must be divisible by $\operatorname{lcm}(A, B)$. Thus there are $\lfloor (N-1)/\operatorname{lcm}(A, B) \rfloor$ such numbers.

We can loop from 1 to $\max(A, B)$ to find $\gcd(A, B)$. Then $\operatorname{lcm}(A, B) = A/\gcd(A, B) \cdot B$.

Finally, the answer is $\lfloor (N-1)/A \rfloor + \lfloor (N-1)/B \rfloor - \lfloor (N-1)/\operatorname{lcm}(A, B) \rfloor$.

Problem B. All about that Base

We simply extend long addition from base 10 to base b.

It's useful to define two helper functions, one which maps a base b digit to an integer, and a second which maps an integer (less than b) to a base b digit.

We set a carry value to 0. Now for every digit x_i and y_i of x and y, starting from the least significant digit, we can:

- convert x_i and y_i to integers
- add them together with the carry value
- if their sum is greater than b, then subtract b and set the carry value to 1
- convert the resulting integer to a base b digit

Finally, you print out the resulting digits.

If x and y have differing lengths, it's convenient to pad the start of the shorter one with 0's.

Don't forget the final carry, if there is one.

Problem C. Grid Walk

Subtask 2

If any cell is blocked, there is no way to reach the end. Otherwise, you just walk right. So output 0 if the input contains a `#`, and 1 otherwise.

Subtask 3

Two paths can now only differ by when you choose to perform a `D` step. You have to perform this step if you encounter a `#` in the first row. You also cannot reach the end after this `D` step there is another `#`. So just look for where the first `#` occurs in the first row, and where the last `#` occurs in the second row, and output their difference (or 0 if this is not positive).

Subtask 4

Perform a DFS or BFS of all possible paths.

Subtask 5

This is now just all possible permutations of $N-1$ `D`'s and $M-1$ `R`'s.

So you can calculate $\frac{(N+M-2)!}{(N-1)!(M-1)!}$. To deal with the modulo restriction, you should divide the individual factors in the denominator with factors in the numerator. Once only factors are left in the numerator, you can begin multiplying them and only keeping the remainder mod 10^8.

Subtask 6

Let $P(i,j)$ be the number of ways of reaching the cell (i,j) (row i, column j). If cell (i,j) is blocked, clearly $P(i,j)=0$. Note $P(1,1)=1$, as that's where you start (alternatively, you can see $P(1,2)=P(2,1)=1$, provided they are not blocked).

Now suppose (i,j) is not blocked. Any path reaching (i,j) must pass through either $(i-1,j)$ or $(i,j-1)$. So $P(i,j)=P(i-1,j)+P(i,j-1)$. We can then loop through cells from the top left to the bottom right in $O(NM)$ time to determine $P(N,M)$.

Problem D. Tree Slugs

Subtask 2

Note a slug walking from a to b or b to a doesn't make a difference. So if we have a and b with $a>b$, we can swap them so that $a<b$.

We can just as well store the path as an array A. For each a, increment $A[a]$, and for each b, decrement $A[b]$. Now walk from the left of the array to the right, keeping track of the sum. After each step, if the sum is positive, it means a slug would traverse the current edge, making it lichenfree, hence increment a counter. Then output the value of the counter.

Subtask 3

There is just one slug. So we can find the shortest path with a BFS. The length of this path is the answer.

Subtask 4

Root the tree at the shared vertex, running a BFS and recording vertices visited in order in a list. Now go through the nodes again in reverse order. Each time you encounter a leaf that is nota favourite vertex of any slug, delete it from the tree. Then if there are V vertices remaining, there must be $V-1$ lichen-free edges.

Subtask 5

Root the tree at some vertex, and set it's height to 0. Set the height of each vertex to one less than the height of it's parent.

Note the path of each slug from a to b will consist of walking up the tree to some vertex, then walking down to b. Now, this vertex they walk up to is the lca (see lowest common ancestor algorithm) of a and b. You can calculate these lca's in $O(n \log n)$. In each vertex v, store, into a variable $X[v]$ the minimum of all the heights of the computed lca's corresponding to that vertex. In vertices with no corresponding lca, you can store negative infinity.

Now process the vertices from the bottom of the tree upwards. The edge to the parent of each vertex v will be traversed by a slug if and only if there is a vertex u in the subtree of v (including v) such that $X[u]$ is greater than the height of v. To simplify, whenever we process a vertex v with parent p, we update $X[p]$ to the maximum between $X[p]$ and $X[v]$. In this way, we can count the lichen-free edges in $O(n)$ time, for a total time complexity of $O(n \log n)$.

Glossary

lcm Lowest Common Multiple

gcd Greatest Common Divisor
`https://www.geeksforgeeks.org/c-program-find-gcd-hcf-two-numbers/?ref=lbp`

DFS Depth-first search
`https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/`

BFS Breadth-first search
`https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/`

LCA Lowest Common Ancestor
`https://cp-algorithms.com/graph/lca.html`

Links

Addition base b
`https://www.geeksforgeeks.org/program-to-add-two-integers-of-given-base/`

Traversing a matrix
`https://www.geeksforgeeks.org/count-the-number-of-ways-to-traverse-a-matrix/`