Problem Tutorial: "Airplane Cliques"

Let's call the given tree T, and the depth of vertex v in it is dep_v (let's root the tree for an arbitrary vertex).

Then, a graph where edge exists if and only if vertices are on a distance at most x is $G = T^x$.

After that, we can note one very useful fact.

- For the fixed vertex v, adjacent vertices u in G with $dep_u \le dep_v$ form a clique.

Let's sort all vertices in the increasing order of depth, like that you can get an order of vertices such that all vertices adjacent to the fixed vertex to the left in this order form a clique (it is a **PEO**).

Let's fix the rightmost vertex in a clique, let's find the number of adjacent vertices with it to the left of it (you can do it with your favorite tree data structure, centroid decomposition/heavy light/small to large).

You can find it in $\mathcal{O}(n \log^2 n)$ or in $\mathcal{O}(n \log n)$, it depends on your data structure skill (both are acceptable).

If this value is a_v, what is the number of cliques of size k with vertex v as the rightmost?

Of course, it is $\binom{a_v}{k-1}$.

To find the answer for each k, we need to find $s_k = \sum \binom{a_v}{k}$, for k in $0, 1, \ldots, n-1$.

Let's calculate t_x — the number of v such that $a_v = x$.

$s_k = \sum t_x \cdot \binom{x}{k}$.

$s_k = \frac{\sum \frac{t_x \cdot x!}{(x-k)!}}{k!}$.

Let's call $f_x = t_x \cdot x!$, and $g_k = s_k \cdot k!$ (it's trivial how to get f from t and s from g).

$g_k = \sum \frac{f_x}{(x-k)!}$.

So we can see that we can find g using polynomial multiplication of f and $\frac{1}{(n-i)!}$.

We can use FFT to do it in $\mathcal{O}(n \log n)$.

The total complexity is $\mathcal{O}(n \log^2 n)$ or $\mathcal{O}(n \log n)$.

Problem Tutorial: "Best Tree"

Let's assume that n is even (when it is odd, you can change $\frac{n}{2}$ in all following formulas to $\frac{n-1}{2}$).

At first, $n = 2$ is a tricky case (WA2), the answer for it is 1.

For $n > 2$, I claim that the answer is $\min(\frac{n}{2}, x)$ where x is the number of non-leaves (the number of vertices with $d_i > 1$).

Note that the answer can't be larger than $\frac{n}{2}$, and also can't be larger than the number of non-leaves (because you can't match two leaves if $n \neq 2$).

So we only need to prove that this value is achievable.

Let's use the following algorithm: $\min(\frac{n}{2}, x)$ times take a pair of vertices with the largest and with the smallest degree and merge them.

Note that the total degree of vertices in each of these pairs will be > 2 and also it will be $\le n - \min(\frac{n}{2}, x)$.

So we can merge all these pairs into one vertex, with the degrees $d_a + d_b - 1$, where $1 \le d_a + d_b - 1 \le (n - \min(\frac{n}{2}, x) - 1)$.

And now we can build the tree of these pairs with $n - \min(\frac{n}{2}, x)$ vertices!

Problem Tutorial: "Cells Blocking"

Let's find the *leftmost* and *rightmost* paths from $(1, 1)$ to (n, m) (if there are no such path, we can just print $\binom{empty}{2}$).

Let's calculate $p_{i,j}$ — whether it is possible to get from (i, j) to (n, m).

Leftmost path is the path that goes from (i, j) to $(i + 1, j)$ if it is possible (if $p_{i+1,j}$) and to $(i, j + 1)$ otherwise.

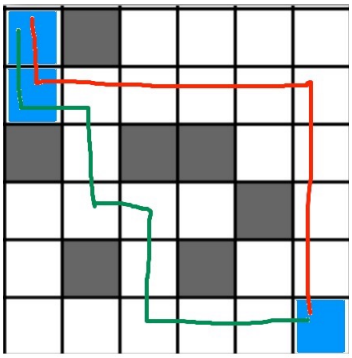
Rightmost path is the path that goes from (i, j) to $(i, j + 1)$ if it is possible (if $p_{i,j+1}$) and to $(i + 1, j)$ otherwise.

Let's say that s is $(1, 1)$ and t is (n, m), and the vertex is reachable from t if there is the path from (i, j) to t which goes only to the right and to the down.

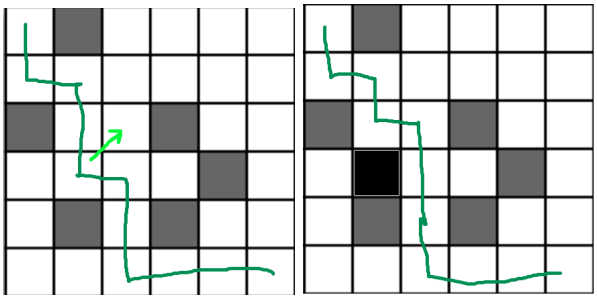
One simple fact about these paths is that on each diagonal $i + j = x$ leftmost path is always behind the rightmost (and also all vertices on this diagonal that are below leftmost or rightmost path (not on the segment between them) are unreachable either from s or from t).

Using this fact, we can note:

- The number of ways to block **one** cell to ban all the paths is the number of vertices in the intersection of these two paths.



To find the number of ways to block two cells, let's fix one of these cells (the one that is on the leftmost path, because at least one of them should be on it) after that let's find the new leftmost path (after blocking this cell) to do this, we can find next cell which is reachable from s and t on the same diagonal as the blocked cell. This vertex should lie on the new leftmost path, and we can restore the whole path, going from it back to get the prefix of the path, (we should precalculate for each vertex also whether it is possible to get from $(1, 1)$ to (i, j)), and using the same way as we described before to get the suffix of the path. After that, we can find the intersection of this path and the initial rightmost path and add this value to the answer.



That will work in $\mathcal{O}((n + m)^2)$, because the length of the path is $n + m - 1$.

Note that also it is possible to solve the problem in $\mathcal{O}(nm)$ because you don't need to restore the whole path naively, you can precalculate with DP for each vertex the intersection of the backward and forward path with the initial rightmost path.

Problem Tutorial: "Disjoint LIS"

- Understanding everything

  How to check that the permutation is good?

  There is a well-known algorithm for the largest sum of k increasing subsequences, and for $k = 2$ it can find the largest sum of two disjoint increasing subsequences, and we can check that this value is equal to $2\times$ LIS.

  So, we just need to find the number of permutations, such that after you run this process on them, two first rows will have equal size.

- Counting

  Let's fix the *shape* λ of the young diagram, there are $P(n)$ of them.

  Let's calculate the number of permutations with such shape.

  I claim that the answer is $f(\lambda)^2$, where $f(\lambda)$ is equal to the number of ways to put values inside of the diagram, such that all rows and column will be increasing.

  Why? During the algorithm for finding the best k disjoint increasing subsequences, let's maintain the other young tableaux with the same shape as the main, and on the i-th iteration let's write i to the new cell into this tableaux, so they still have the same shape.

1	2	7	1	3	4
3	8	10	2	5	6

1	2	6	1	3	4
3	7	10	2	5	6
8			7		

  Note that now by permutation you can uniquely get the two young tableaux, and also by the two young tableaux you can uniquely restore the permutation (by induction it is easy to show that we can always undo the last move), so it is a bijection.

  How to calculate $f(\lambda)$? The hook-length formula will help you!

  So in $O(n)$ for each of the $P(n)$ partitions you can find the answer.

  Total complexity is $O(n \cdot P(n))$.

Problem Tutorial: "Easy Win"

Let's note that the grundy number of the pile with n stones if in one move you can take at most x from it is $n \bmod (x+1)$.

So for each x, we need to find $(a_1 \bmod (x+1)) \oplus (a_2 \bmod (x+1)) \oplus \ldots \oplus (a_n \bmod (x+1))$.

Let's fix $y = (x+1)$.

We need to find $(a_1 \bmod y) \oplus (a_2 \bmod y) \oplus \ldots \oplus (a_n \bmod y)$.

Let c_x be the number of i, such that $a_i = x$.

Let's fix a $k \le \frac{n}{y}$, and let's deal with integers in interval $[ky, (k+1)y)$.

if $ky \le a_i < (k+1)y$, then $a_i \bmod y$ is equal to $a_i - ky$.

For fixed bit j we need to find the sum of c_t among $ky \le t < (k+1)y$, such that $(t - ky)$ has bit j.

Let's precalculate $f_{i,j}$ — sum of c_x among $i \le x$, such that $(x - i)$ has bit j.

$f_{i,j} = f_{i+2^{j+1},j}+$ sum on the segment $i + 2^j \dots i + 2^{j+1} - 1$.

Then to calculate on the segment we need to take $f_{ky,j}$ and then subtract $f_{t,j}$ where t is the smallest integer $\ge (k+1)y$ with the same remainder as ky modulo 2^{j+1} and also subtract some sum on the border segment.

Summing this up by all segments, we can find out the parity for each bit, so we can check whether the xor is zero or not.

Total complexity is $O(n \log^2 n)$.

Problem Tutorial: "Farm of Monsters"

At first, let's decrease all h_i by 1, and now let's say that monster will die if $h_i < 0$, not ≤ 0 as was initially.

Then, for the monster let's find the smallest r_i, such that $(h_i - r_i \cdot a) \mod b < a$.

Note: it is equal to $\lfloor \frac{(h_i \bmod b)}{a} \rfloor$.

If you decide to kill some monster yourself, then it does make sense only to kick him $r_i + 1$ times.

Because you can't kick him a smaller number of times, and instead of kicking him the larger number of times you can just skip moves (or kick another monster instead).

Now, if you decided not to kill the monster then your opponent will make h_i kicks and you will make zero. And if you decided to kill the monster, then you will make $r_i + 1$ kicks and your opponent $\lfloor \frac{(h_i - r_i \cdot a)}{b} \rfloor$.

Imagine that you've already chosen which monster you decided to kill. What is the criterium that you can kill all of them?

Let's say that x_i is the number "`saved moves`", the difference between the number of your opponent kicks on i-th monster and yours (maybe negative).

Let's assume that also $x_0 = 1$ because you are moving first so you have one saved move.

I claim that you can kill chosen monsters if $x_0 + x_1 + x_2 + \ldots + x_i$ is non-negative for each i.

To prove it let's note that if at some moment this value is negative then you definitely can't kill all the monsters on this prefix. Otherwise, you always can wait for the moment when you should start kicking the monster and kick it.

Okay, then how to find the largest number of monsters?

If you decided to kick the monster, then $x_i = \lfloor \frac{(h_i - r_i \cdot a)}{b} \rfloor - (r_i + 1)$, otherwise $x_i = h_i$.

And you need to choose one of these for each i to make the sum on prefix at least zero for each i.

Let's assume that you've decided to kill none monsters, then $x_i = h_i$.

Then, if you will decide to kill i-th monster, prefix sums on the suffix will change on $y_i = \lfloor \frac{(h_i - r_i \cdot a)}{b} \rfloor - (r_i + 1) - h_i$.

And now the problem is to choose the largest number of y_i, so for each i the prefix sum of chosen y_i will be at most $h_1 + h_2 + \ldots + h_i$.

To solve this problem, let's maintain the chosen y_i in heap, and while the sum is too large delete the largest from it.

Note: also this problem is solvable with greedy with segment tree, or with the DP which maintains the set of sorted slopes (tho this solution is equivalent to that greedy with heap).

Problem Tutorial: "Giant Penguin"

We will build a centroid decomposition, which will help us to answer the queries.

Let's take any spanning tree and the centroid in it. Then, all subtrees of the spanning tree are of size $\leq \frac{n}{2}$, but there may be some edges between the different subtrees.

But note that the number of edges between the different subtrees is at most k, so if you will choose centroid and arbitrary end of each edge, and delete all these $\leq k+1$ vertices from the graph, there will be no edges from the different subtrees of the current centroid.

Let's call these vertices *interesting* for the fixed centroid. From each of them, let's run BFS and precalculate the distances to all vertices from the subtree.

On all remaining connected components run this algorithm recursively.

Like that, you can get a tree of centroid decomposition for this graph.

To answer the query, for each interesting vertex let's maintain the closed marked vertex from the subtree.

Note that you only need to update $\mathcal{O}(\log n \cdot k)$ these values then the vertex is marked because you only need to update the values of $\leq k+1$ interesting vertices of $\mathcal{O}(\log n)$ parents in the centroid decomposition tree.

To find the closest marked vertex, let's check all parent centroids in the centroid decomposition in the tree, and for each interesting vertex (same $\mathcal{O}(\log n \cdot k)$ vertices as in the previous query), update the answer to the query with the distance to it + smallest distance to the marked vertex for this interesting vertex.

Note that all distances that you maintain are distances on the current subtree, without going beyond its vertices, but all this enough, because if the shortest path does not lie strictly inside the current subtree, then it should pass through one of the interesting vertices of the parents in centroid decomposition, so you will process these cases too.

Problem Tutorial: "Horrible Cycles"

Let's build this bipartite graph using two operations:

- Add an isolated vertex to the right part
- Add a vertex to the left part and connect it with all vertices of the right part

After that, you will have a sequence of operations, let's code it as a binary string, where '`0`' will denote the first operation and '`1`' the second.

Note that all vertices which were added by the second operation, are connected with all previous vertices which were added by the first operation.

Let's go in the order of these operations and maintain some information about the cycle that we are forming.

When the new operation is of the first type, you can add this isolated vertex as some part of the cycle (which in the future should be connected with the other parts).

Otherwise, if it is of the second type, you can take some two parts of the cycle, and merge them into one using this new vertex, or if there is only one part, you can take it and using the new vertex "`glue`" one part into the cycle and add it to answer.

Note that all parts of the cycle are of the form '`01010101010`', and what you need to know about them is only their quantity.

So you can maintain $dp_{i,j}$ — the prefix of operations and the number of *chains*, as we described before.

Total complexity is $\mathcal{O}(n^2)$.

Problem Tutorial: "Ignore Submasks"

a_i is not a sub mask of x if it has some bit, which x does not have.

For each bit let's calculate the leftmost position of the number with this bit.

Then $f(x)$ is equal to the minimum such value among all bits which x does not have.

Note that you are only interested in the sum of f, so the answer is equal to the sum of minimums of values among all subsequences (because each subsequence will correspond to the x which contains none of the bits from the subsequence).

To calculate this value, we can sort all values for the bits, fix the leftmost minimum in it, and add $value \cdot 2^{k-i}$ to the answer.

Problem Tutorial: "Just Counting"

From basic linear algebra, we know that the answer is $5^{m-basis}$.

To calculate the basis, at first, let's study when it is possible to write integers to get zero-sum.

Let's assume that the graph is connected.

- When it is a tree, it is not possible because edge with the largest depth should have non-zero integer on it, so one of its ends will have non-zero sum
- When it is an even cycle, it is possible, we can just write an alternating sequence of 1 and -1.
- If $m > n$, it is possible because basis can't be larger than n (also it is possible to show because if you have two odd cycles, if they are intersecting, then they produce an even cycle, and otherwise we can write an alternating sequence of 1 and -1 on them (with two equal values near the connection of this cycle to path to the other cycle), and on the path between them write alternating sequence of 2 and -2.
- If graph is a tree + one odd cycle, then it is not possible, because all leaves should have zero-sum, so weights of edges to them should be zero, in the end, you will have an odd cycle, and it's not possible because if first value is x, then values should alternate, so we should have $x = -x$, which is impossible for $x \neq 0$.

So, if the graph is bipartite we can only get a forest as a basis ($n - 1$ edges). Otherwise, we can only get a forest + one edge (n edges).

Summing this up by all connected components, we will get the total base so we can calculate the answer to the problem