Azugand City (azugand)

Author: Andrei Ivan

Developer: Péter Varga

Solution

Consider the graph consisting of the original N vertices and $\log(\max V_i)$ auxiliary vertices. The auxiliary vertices correspond to bit positions in the vertex values. We number these vertices starting from $N+1$.

Now we add edges to the graph as follows. For each vertex i ($1 \le i \le N$) and bit position j, if the bit at position j in V_i is set (i.e., equals to 1), then we add two directed edges between vertex i and vertex $U = N + 1 + j$:

- an edge from i to U with cost 0; and
- an edge from U to i with cost 1.

Next, starting from each auxiliary vertex U, we precompute the minimum distance to each node i using 0-1 BFS. Denote these distances by $D[U][i]$.

For each query (X, Y), we know that the minimum path must include at least one auxiliary vertex. So it is enough to consider the values $D[U][X] + D[U][Y] - 1$ for each auxiliary vertex U, and return the minimum as the answer. Total time complexity is $O((N + Q)\log(\max V_i))$.

Library of Binaria (`binaria`)

Author: Gabriella Blénessy

Developer: Péter Csorba

Solution

On each shelf, we can store a number of books that is one less than a power of two. This is because:

$$\sum_{i=0}^{n} 2^i = 1 + 2 + 4 + \ldots + 2^n = 2^{n+1} - 1.$$

For topic i we need to find a smallest *power of two minus one* which is at least T_i. Formally we need to find $n(\geq 1)$ such that:

$$2^{n-1} - 1 < T_i \leq 2^n - 1.$$

In this case wee need to buy $2^n - 1 - T_i$ extra books for topic i.

Since $T_i \leq 10^{13} < 2^{44}$: we can efficiently determine n by checking each power of two up to this limit. Even calculating powers with `(long) pow(2, j)` is feasible here.

Note: Precomputing the powers of two and using binary search to find n could speed up the process, but this isn't necessary to solve the problem in this case.

Binary Grid (binarygrid)

Author: Alexandru Gheorghies
Developer: Bernard Ibrahimcha

Solution

Let $f(b_0, b_1, \ldots, b_{k-1})$ be the longest contiguous subsequence of equal elements in an array b.
Additionally, let's define:

- The *value* of the i-th line as: $f(a_{i,0}, a_{i,1}, \ldots, a_{i,m-1})$;
- The *value* of the j-th column as: $f(a_{0,j}, a_{1,j}, \ldots, a_{n-1,j})$;

A binary matrix is good if and only if the values of its lines and its columns are all strictly less than 3.
Two important observations are that:

1. Toggling a line in the matrix does not change the value of any line.
2. Similarly, toggling a column in the matrix does not change the value of any column.

Therefore, the problem can be split into two independent subproblems:

1. Fixing every line simultaneously by toggling a subset of the columns;
2. Fixing every column simultaneously by toggling a subset of the lines;

The solution for the first subproblem is as described below:
If $n \le 2$, then the answer is 0.
Otherwise, the answer can be computed using dynamic programming:

- $dp[j][flag_1][flag_2]$ — the smallest number of operations needed to fix every line, if:
- Only the first j columns are considered;
- $flag_1$ represents whether the $j-1$-th column was toggled or not.
- $flag_2$ represents whether the j-th column was toggled or not.

Initially:

- $dp[1][flag_1][flag_2] = flag_1 + flag_2$, for all $0 \le flag_1, flag_2 \le 1$.
- $dp[j][flag_1][flag_2] = +\infty$, for all $j \ne 1$

For $j > 1$, the transitions are as follows:

$$dp[j][flag_1][flag_2] = flag_2 + \min(dp[j-1][0][flag_1], dp[j-1][1][flag_1])$$

However, if two consecutive states are incompatible (i.e. they create three consecutive equal elements), then $dp[j-1][0/1][flag_1]$ will be considered $+\infty$ instead of its normal value.

The final answer for this subproblem is equal to $ans_1 = \min\{dp[m-1][flag_1][flag_2]\}$.

The answer for the second subproblem ans_2 is computed similarly.

The final answer is equal to $ans_1 + ans_2$. If this final answer is equal to $+\infty$, then it's impossible to obtain a "good" matrix.

Time complexity: $O(n \cdot m)$

Memory complexity: $O(n \cdot m)$

Blitz Division (`divisor`)

Author: Stefan Dascalescu

Developer: Stefan Dascalescu

Solution

In order to solve this problem, we need to find a divisor d such that we can make both A and B multiples of d with the given K increments. In other words, this means that we want $A + B + K$ to be multiple of that said d.

A slow solution would be to check all possible ways to distribute the K increments between A and B, and check for each way what would the GCD of the operation be. However, this would be too slow and it would not work within the time limits.

As $A + B + K$ must be multiple of the answer, we can start with the divisors of $A + B + K$ and try each of them, because we know that the answer must be one of these divisors. For a given divisor d, all we have to do is to find out how much we need to increment A and B to make both of them divisible by d. If we know $r_A = A \bmod d$ and $r_B = B \bmod d$, we know that the number of operations required is

$$(d - r_A) \bmod d + (d - r_B) \bmod d,$$

unless one or both of the integers is a multiple of d: in that case the number in one or both of the brackets is zero.

In order to check the divisors, we can apply the standard trial division algorithm, which allows us to find all divisors of a given number x in $O(\sqrt{x})$. Take note that because $A + B + K$ can go up to $3 \cdot 10^9$, you might have to deal with overflow related issues, and you need to use the appropriate data types for this to not cause a problem.

Road Expansion Plan (expansionplan)

Author: Valerio Stancanelli

Developer: Tommaso Dossi

Solution

This problem can be rephrased in terms of graphs. For a graph G, let c be the number of connected components it has, its score is 2^c. The problem is asking to compute the sum of the scores of all graphs with N nodes whose set of edges contains the given one.

2^c is equal to the number of subsets of connected components, which is equal to the number of subsets of nodes $X \subset [0, N)$ such that there are no edges between X and $[0, N) \setminus X$.

We can now use double counting. The answer is equal to the sum, over the subsets $X \subset [0, N)$ of the number of graphs in which there are no edges between X and its complementary set or, equivalently, X is the union of some connected components. Let's call such subsets of nodes *good*. Let C_X be this quantity. $C_X = 2^{\frac{N(N-1)}{2} - M - |X|(N-|X|)}$.

We can observe that C_X only depends on the siz e of X. The problem is hence reduced to counting the *good* subsets of $[0, N)$ of each size. As said before the *good* subsets are only those who are union of connected components in the given graph. It is possible to answer this question in $\mathcal{O}(N^2)$ using knapsack. For a full solution it is possible to optimize bounded knapsack to $\mathcal{O}(N log^2 N)$ using FFT.

Dangerous Parkour (parkour)

Author: Davide Bartoli

Developer: Davide Bartoli

Solution

This problem has a beautiful solution, which is described below.

Problem Setting (`problemsetting`)

Author: Alexandru Gheorghieș

Developer: Alexandru Gheorghieș

Solution

First assume that x is given. We want to answer a simpler question:

Can we organize x rounds?

Luckily greedy algorithm works here: let's use all the 1-difficulty problems as rating 1 problems, and if more are needed, we can supplement with the 1.5-difficulty problems. Then, the remaining 1.5-difficulty problems can be counted as 2-difficulty problems. If more 2-difficulty problems are needed, we can supplement with the 2.5-difficulty problems. Then, the remaining 2.5-difficulty problems can be counted as 3-difficulty problems, and so on. If we do not run out of problems at any step: we can organize x rounds, otherwise we can not.
This can be done in $O(N)$ time.

After that we can use binary search, since if we can organize x rounds then we can organize any fewer rounds as well. Clearly we can organize 0 rounds, and we can not organize $\max(a_i) + \max(b_i) + 1$ rounds. This binary search part could be implemented like this:

```
lo = 0
hi = 2*10**9+1
while hi - lo > 1:
    mid = (hi + lo) // 2
    if we can organize mid rounds:
        lo = mid
    else:
        hi = mid
```

After this variable `lo` contains the maximum number of rounds that can be held using the available problems.
The time complexity is $O(N \log(2 \cdot 10^9 + 1))$.

Morse (`walrus`)

Author: Alexandru Gheorghieș

Developer: Alexandru Gheorghieș

Solution

If multiple sleeping walruses (`.`) are lined up together, waking up one will trigger a chain reaction that wakes all the others in the same group. This process stops when it reaches an awake walrus (`-`) or the end of the line.

To wake up all the walruses efficiently:

1. **Count the Groups**: Identify how many separate groups of sleeping walruses (`.`) there are. You'll need to wake up one walrus in each group.
2. **Choose the Best Walrus to Wake**: Within each group, wake the walrus closest to the middle. If the group has an even number of walruses, wake any one of the two middle ones. This ensures the awakening spreads most efficiently.
3. **Prioritize Larger Groups**: Start waking up groups in descending order of their size. Start the largest groups first. This greedy strategy minimizes the total time required, as you can only wake one group per second.

Word by Word (`wordle2`)

Author: Alexandru Gheorghies

Developer: Alexandru Gheorghies

Solution

The first step in finding the hidden word is to determine the letters of the word. Since we have 26 letters, we split the alphabet into blocks of 5 characters. We can find whether the hidden word contains the letters `a`, `b`, `c`, `d` or `e` by querying the word `abcde`. Then, we query for `f`, `g`, `h`, `i`, and `j` and so on. Notice that five queries are enough here as we can skip querying for `z`, since it is the only letter in the last block.

The next step is to determine the exact position of each letter. Lets say the letters are `a`, `b`, `c`, `d`, and `e`. We can query `aaaaa`, `bbbbb`, `ccccc` and `ddddd` to learn the exact positions of these four letters. The last unclaimed position must belong to letter `e`, allowing us to determine the hidden word in at most $5+4=9$ queries. Note that the same technique works if the word contains less than five different letters.