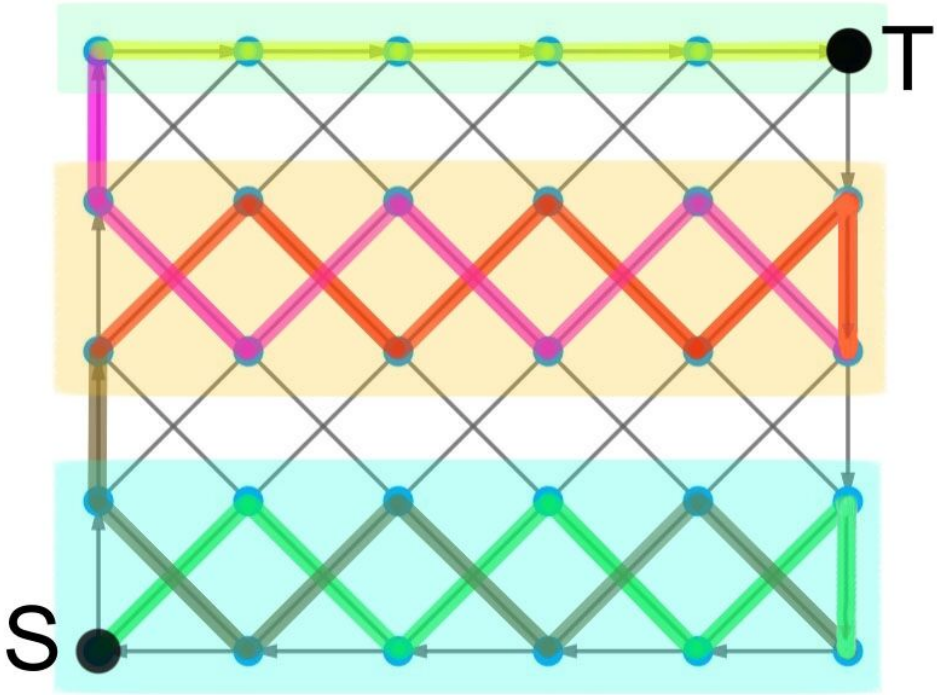
Solution

A. Nucleic Acid Testing

We will show that there exists a route visiting each isolation point exactly once. There's no faster route because you have to visit each isolation point at least once. And there are $n(n+1)$ isolation points, so it will take $n(n+1)-1$ units of time.

There are many ways to construct such a route. We'll introduce a simple one.



The picture above shows a route of $n=5$ (starting from isolation point S, going in the order of green, brown, red, purple, yellow, and finally reaching isolation point T).

If n is odd, then each row contains even isolation points, and there're odd rows. For two adjacent rows, we can start from the bottom-left corner of these two rows, visit each isolation point exactly once, and reach the top-left corner. Then we take the subway to the previous row. By repeating this, you will visit all isolation points except those in the first row and then reach the top-left corner of the whole matrix. At last, take the subway to visit the rest isolation points in the first row one by one.

If n is even, imagine that you rotated the matrix 90 degrees clockwise. The rotated matrix will also satisfy the conditions above (each row contains even isolation points, and there're odd rows), and will have the same structure of roads. So we can apply the method described in the previous paragraph.

Of course, many other methods can get accepted.

B. Fight Against the Epidemic

The description can be simply expressed as "There is a tree. Find out two nodes x, y. Denote $\mathrm{dis}(x, y)$ as the distance between x and y, and maximize $\max(a_x, a_y) \cdot \mathrm{dis}(x, y)$".

Because $\max(a_x, a_y) \cdot \mathrm{dis}(x, y) = \max(a_x \cdot \mathrm{dis}(x, y), a_y \cdot \mathrm{dis}(x, y))$, the problem is equivalent to "maximize $a_x \cdot \mathrm{dis}(x, y)$ by selecting x, y properly "

For each node, we just need to figure out the distance between the farthest node and itself.

Let the two ends of the diameter of the tree be u, v. Then u or v must be the farthest from x. So we just need to find out the diameter and calculate the distance between x and u and between x and v.

The complexity is $\mathcal{O}(n)$.

But In fact, because the n is not big, solution with complexity $\mathcal{O}(n \log n)$ is also allowed.

P.S. About the diameter of tree: The two ends of the diameter of a tree is a pair of points, which is the farthest in the tree. When calculating the diameter, we only need to find out the farthest node x from any node, and then find out the farthest node y from node x, then x and y are the two ends of the diameter.

C. Do Researches on Virus

It can be found that at any time, we can only determine that the activity of the virus is in a certain interval. This reminds us of a dynamic programming algorithm, and the state of this algorithm is the interval where the activity might be located.

Let $f(l,r)$ be the product of minimum expected cost multiplied by $r-l+1$, when the activity can be determined between l and r.

The transfer is as follows:

- If the numbers in the interval $[l,r]$ belong to different states, then $f(l,r)$ is equal to the sum of the dp values of the interval where each state in $[l,r]$ is located.
- If the numbers in the interval $[l,r]$ all belong to the state 1 , then $f(l,r)=0$.
- Otherwise, we can enumerate all m operations, then we will get $f(l,r)=\min\{f(l-w_i,r-w_i)+(r-l+1)\times v_i\}$.

If the maximum value of activity is L, then the answer is $f(1,L)$. The complexity is $\mathcal{O}(L^2m)$, which cannot pass this problem.

We first run a full backpack algorithm on all operations, then we can use one operation instead of multiple consecutive operations. After that, we've got new v_i and w_i.

After this, for the transfer from $f(l-w_i,r-w_i)$ to $f(l,r)$, we can force it to satisfy that the numbers in $[l-w_i,r-w_i]$ belong to several different states.

Then if we run the memorized search with the above restrictions, we can find that our legal state number is $\mathcal{O}(L)$. Because we will only use the following state: For all $1\le p\le L$, let i be the state when the activity is p, all $f(a_{i-1}+1,p)$ and $f(p,a_i)$.

If we use prefix sums to optimize when transferring, we can solve this problem with the complexity of $\mathcal{O}(Lm)$。

D. Fight Against Virus

Firstly there is a brute force algorithm. Let $A(i, j)$ record whether the player taking the first turn win in the cases where the remaining total action value is i and the action value in the previous round is j. We can use dynamic programming to calculate all h_m with the time complexity of $\mathcal{O}(n^2)$, which can pass **the first two subtasks**.

Then let us consider how to calculate h_m.

In the case where m is odd, the humans can take an action of value 1 to guarantee victory, just because only 1 can be chosen as the action value for both side in the following rounds.

In the case where m is even, the action value of humans' first round must be even too, otherwise m would be odd when the virus takes turn. Thus we can conclude that all the action values should is even. Just divide all the action values by 2 and we get $h_{2m} = 2h_m$.

In summary, we have

$$h_m = \begin{cases} 1, \text{ if } m = 2n + 1, \\ h_n, \text{ if } m = 2n. \end{cases}$$

It's obvious that $h_m = \max_{2^k | m} 2^k$, which is $h_m = \text{lowbit}(m)$.

The algorithm calculating all h_m with the time complexity of $\mathcal{O}(n)$ can pass **the third subtask**.

Later let us consider how to calculate the prefix sum of $f_i = \sum_{m|i} h_m$ as $S(n)$. We have

$$S(n) = \sum_{i=1}^{n} h_i \left\lfloor \frac{n}{i} \right\rfloor$$

Algorithms for number theory summation that do not reach the time complexity of $\mathcal{O}(\sqrt{n})$ can pass **the forth subtask**. For instance, one approach is to enumerate the value of $\left\lfloor \frac{n}{i} \right\rfloor$. But because there are so many algorithms, they will not be described in this problem solution.

We can enumerate the value of h_i and calculate all h_i with the same value simultaneously. For convenience, let $g(n) = \sum_i \left\lfloor \frac{n}{i} \right\rfloor$.

$$S(n) = g(n) + \sum_{k \geq 1} (2^k - 2^{k-1}) g\left(\frac{n}{2^k}\right)$$

We can calculate each $g(n)$ with time complexity of $\mathcal{O}(\sqrt{n})$. Thus we get the time complexity of this algorithm by the following equation.

$$\sum_{k \geq 0} \sqrt{\frac{n}{2^k}} = \sqrt{n} \sum_{k \geq 0} \left(\frac{1}{\sqrt{2}}\right)^k = (2 + \sqrt{2})\sqrt{n}$$

It's $\mathcal{O}(\sqrt{n})$.

Meanwhile, you can use the algorithm for SPOJ DIVCNT1 to calculate $g(n)$, which has a better time complexity of only $\mathcal{O}(n^{\frac{1}{3}} \log n)$. There's no subtask set for this algorithm because it's not meaningful.

E. The Hammer Went Down in History

The storage capacity is equal to `upper contour` minus `lower contour`. `lower contour` is easy to find.

`Upper contour` is regarded as a step in both directions. The steps on the left half are explained below.

In a section with a length of l. Define the `left ladder` of the section as the prefix $\max$. Over time, its `left ladder` will only change $\mathcal{O}(l)$ times, because the time when each number is greater than all the numbers on the left is an interval.

So we want to build a segment tree for a sequence. Each node of the segment tree is an interval. For each interval on the segment tree, find the change of its `left ladder`.

It is more troublesome to find the time segment when each number at each node of the segment tree is larger than all the numbers on the left. You can directly use dynamic half-plane intersection, or you can build a convex hull at each node of the segment tree. If you choose the latter, at this time, you need to pay attention to the nodes in the $\mathcal{O}(\log n)$ inquiries of each number. It seems that each point has $\mathcal{O}(\log n)$ single-point convex hull inquiries on the intervals. But in fact, these inquiries will only locate on a total of $\mathcal{O}(\log n)$ different nodes on the segment tree.

Then, for each node on the segment tree, build the inner segment tree to maintain the change of the `left ladder`.

When the interval is inquired, the nodes on the segment tree are accessed from left to right, and the inner segment tree on each node is used to find where the `left ladder` merged.

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

Maybe ... a little hard to write? So ... be patient.