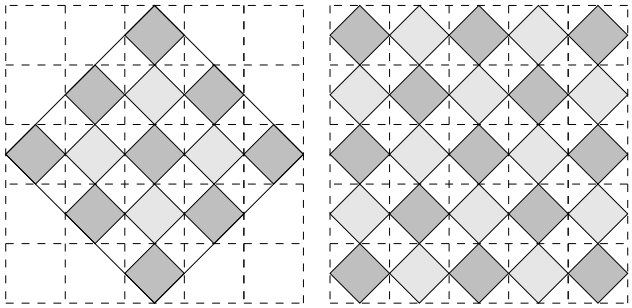
Short Task Spoilers

Dominance

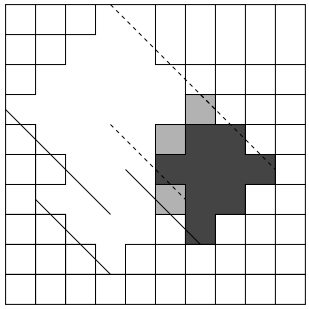
This kind of task is a typical scanline problem. Nontheless, considering the shape of the squares that can be attacked by one inhabited square, a scanline in x or y direction is not suitable.

By using a diagonal scanline, it is possible to take advantage of the square shape of the attack range (left picture). Then it becomes axes-parallel.

Using a diagonal scanline is equivalent to doing a simple transformation on the coordinates: $(x, y) \mapsto (x + y, x - y)$, for example. That transformation results in a lattice with every second square left out (right picture). Note that the lattice itself can be divided in even and odd squares (either $x + y$ and $x - y$ are both odd or both even, depicted in different gray shades).



While sweeping over the board, the diagonal scanline keeps track of all attack ranges it intersects. Then the lower left and the upper right side of the attack ranges mark the entries (complete lines) and exits (dashed lines) for the scanline:



The important observation is, that you can quickly calculate the number of dominated squares between two consecutive entry or exit points. In order to do so, you calculate for one step the number of even and odd squares dominated by white and black bugs, and multiply that numbers by the distance you move your scanline before the next event.

The total runtime is then governed by the time for sorting the entry and exit points, and the access time of the data structure used. For $N \leq 3\,000$ it is not necessary to use sophisticated data structures. Those with linear access time suffice, resulting in a total runtime of $O(N^2)$.

Information

Edmonds proved in 1973 that the maximum number of edge-disjoint spanning arborescences rooted at a fixed vertex r equals the minimum cardinality of any r-cut. In the task we are only looking for 2.

One method is to build the first arborescence T_1 starting at r using Prim's method, adding an edge $e = (x, y)$ only, if there exists an r-y-path edge-disjoint from T_1 and e, then build T_2 from the remaining edges; this results in $\mathcal{O}(VE)$.

Another method is to first choose any spanning arborescence T_1, then start building T_2 starting from r using the remaining edges as long as possible. If (i) there is no remaining edge, (ii) T_2 does not yet cover V and (iii) there are two edge-disjoint spanning arboescences in the graph, the T_2-cut now consists of at least two edges in T_1. We now have to "steal" one edge $e = (x, y)$ from T_1 to use it for T_2. We therefore try to fix T_1 with a r-y-path that does not use T_2 or e. If we search for this path backwards, we get $\mathcal{O}(V^2)$.

Knights

First consider the same game with only one knight on the board. Then there's winning and losing positions (e.g. $R_0 = \{(1,1),(1,2),(2,1),(2,2)\}$ are obviously losing positions - all positions that can reach any of these four are winning positions and so on). In the case of $K = 1$ one could use DP to solve the problem. There's actually a very simple pattern of winning and losing positions (0=losing, 1=winning):

```
..........
1111111111.
1111111111.
0011001100.
0011001100.
1111111111.
1111111111.
0011001100.
0011001100.
```

So the case $K = 1$ can be solved in $O(1)$.

For the case $K > 1$ the strategy for both players is obvious now: The current player wants to move all knights on losing positions to the rectangle R_0 as quickly as possible and delay the movement of the knights on the winning positions as much as possible.

So for each losing position p we determine the minimum number of moves $m(p)$ that the knight takes to get to R_0 assuming the other player plays optimally. Equally we compute for each winning position p the maxmimum number of moves $m(p)$ that the knight takes to get to R_0 assuming the other player plays optimally. Apparently $m(R_0) = \{0\}$. Recursively we can now define

$$m(p) = \min_{p' \text{ reachable from } p} \{m(p') + 1\}$$

for each losing position p, and

$$m(p) = \min_{p' \text{ reachable from } p, p' \text{ losing position}} \{m(p') + 1\}$$

for each winning position.

Once we know m for every knight respectively we are in a winning position if we have a knight in a winning position p such that for all knights in losing positions p' the inequality $m(p) \geq m(p')$ holds.

Computing the values of m can be done with a dynamic programming approach. Again a quite simple pattern shows which yields an $O(1)$ formula for m.

Fence

One main observation is that we only need one polygon to get the optimal answer. The cost of erecting 3 fences (and thus making a triangle) to enclose a single tree is still less than losing that tree. So we simply include the 3 fence posts in the polygon.

This algorithm is divided into two main parts: convex hull and finding the shortest cycle.

First, we do a convex hull of the holes, and then we check for each tree whether it is within the convex hull. Of course, if there are no trees in that convex hull, we can immediately return the result as $111M$.

Then we collect the trees contained within the convex hull. Here we build a directed graph by having the holes as the vertices, and the pairs of holes as the edges. We only include edge v_1, v_2 iff all trees contained in the convex hull are on the left side of that edge. Using this graph, we try to find a cycle that uses the least number of edges. This can be done using Floyd-Warshall all shortest paths algorithm. We can use this cycle to pick the fences Farmer Fred uses.

The time needed for the convex hull is $O(N \log N)$ by using Graham scan, and checking whether the trees are inside the convex hull can be done in $O(NM)$. The graph reconstruction needs $O(N^2M)$ steps, while finding the cycle needs $O(N^3)$ steps. Therefore, this algorithm runs in $O(N^3 + N^2M)$.

Order

Maximize the flow in the following network: Vertices are a source, a drain and one node per order and one per machine. Edges connect (i) the source with each order with a capacity equal to the income value of the respective order, (ii) the orders with the machines with a capacity equal to the rent of the respective machine and (iii) the machines with the drain with a capacity equal to the purchase price of the respective machine.

If an edge from the source to an order has no remaining capacity, the order is rejected.

If an edge from an order to a machine has no remaining capacity, the machine is rented (the rent is substracted from the purchase price).

If an edge from a machine to the drain has no remaining capacity, the machine is purchased.

Snake

The main idea to solve this problem is to utilise a combination of ternary search and binary search to simultaneously find the left and the right bounds (meaning head and tail) of the snake. Assume that we have an interval that covers x units, and we want to locate the head or tail in that interval. We need to take into account that the snake may move forward, which results in an interval shown in Figure 1.

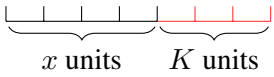


Figure 1: Interval of x units, appended with K units

By asking at the middle of the interval (rounded up or down depending on whether we search for the head or the tail), we can guarantee the resulting interval has the size $\leq \lceil \frac{x+K}{2} \rceil$.

What we need to do now is to determine how many times we need to iterate the binary search until we are certain that we have found a sufficiently good estimate. This happens when $x \leq K + 1$. If we combine both the left bound and right bound intervals, we have the total size to be at most $2K + 1$. We simply pick the median of all possible snake lengths left, and we arrive at an estimate which differs at most K units from the real length of the snake.

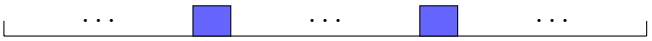


Figure 2: Ternary search

When we use the $\lceil \frac{x+K}{2} \rceil$ formula, we find that for initial range size $x = 12122$ and all K values up to ten, we still need 14 pairs of questions to ensure the we reach a good estimate. To go around this limit, we first do ternary search as illustrated in Figure 2. By doing a ternary search for the first iteration, we ensure that regardless the snake's reply the size of the resulting left and right intervals to be at most 4041. With $x = 4041$, we only need extra 12 pairs of questions to reach the good estimate. So we conclude that the worst case number of calls is at most 13.