ZVRK

Observe that the zvrk of an integer X is 1 iff X is odd.

The zvrk of X is 2 iff X becomes odd when divided by 2.

In general, the zvrk of X is 2^n iff X is divisible by 2^{n-1}, and X divided by 2^{n-1} yields an odd number.

Since we can calculate the number of odd numbers in a given interval rather fast, we can keep a running total (the solution) and, in the i^{th} iteration, add the number of odd integers in the current interval multiplied by 2^{i-1}. We then divide the endpoints of the interval by 2, rounding the lower bound upwards and the upper bound downwards. In effect, we have discarded all odd numbers from the interval and divided all even numbers by 2.

The overall time complexity of the solution is $\Theta(\log_2 B)$. Note that there are also many other solutions with the same time complexity.

XL

In order to make things easier, we can first generate a master string of characters S containing all Roman numerals less than 900 (all numbers larger than 899 would require the digit M, which does not appear in this problem), complete with the dot characters separating the numerals. Our goal now is to start out at the beginning of S and go as far as possible, moving on the given board. The solution is the number of dot characters we've passed. This string contains 6300 characters in total.

This problem can be modeled as a state search where a state consists of two pieces of information: an index into S, the character which we stepped on last and the coordinates where we are positioned. A state is illegal if the two pieces are contradictory i.e. the square at the coordinates does not contain the required character. Each state is connected with at most four others, for which the index is larger by one and the coordinates are those of one of the four neighboring squares.

Observe that the graph we've constructed is a DAG (directed acyclic graph) and we're required to find the longest path from the starting state. We can find this path using either a breadth first search or dynamic programming. The BFS approach is more efficient since it will never consider unreachable states.

The overall time complexity of the solution is $O(length\_of\_S \cdot N^2)$.

DEVA

The first part of our solution consists of finding which pairs of cities are connected by paths at most L road segments long. Using a straightforward BFS approach would use more than $8 \cdot 4^{10}$ (millions of) questions.

We can, however, use a meet-in-the-middle approach to cut down on the number of questions. First we run a BFS starting in each of the cities to find all cities that are at most L/2 road segments away from the starting city. Now, if we can reach city C from city A using at most L/2 road segments, and we can reach city C from city B with the same upper bound, then we can surely reach city B from city A using at most L moves.

The number of questions required to do this is approximately the square root of the number of questions required for the first approach. In fact, we don't need more than $8 \cdot (1+5+5 \cdot 4+5 \cdot 42+5 \cdot 43) = 3408$ questions.

The second part of the solution requires us to find the Hamiltonian cycle in a graph consisting of at most 8 vertices, which we can do in reasonable time by exploring all 8! possibilities.

VID

We can solve the problem by calculating the following two quantities after adding each new point P:

· how many points point P is very visible with

· how many pairs of points have ceased being very visible after adding point P

The first number can be obtained by first translating all points so that point P is in the origin of our coordinate system and then find the number of pairs very visible with the origin in each of the four quadrants. For example, a point A from the first quadrant is very visible with the origin iff it is not blocked by the highest point in the first quadrant which is both lower than point A and very visible with the origin. This can be done efficiently if the points are kept sorted by their y-coordinate. We proceed in a similar fashion for the other three quadrants.

Note now that two points need to be in opposite quadrants in order to become blocked by point P. The second number is thus equal to the number of pairs of points in opposite quadrants which were very visible before adding point P. This can also be done efficiently by keeping the points in sorted order.

The time complexity of the solution is $O(N^2)$.