ETH Zurich
Institut für Theoretische Informatik
Prof. Dr. Angelika Steger
Prof. Dr. Emo Welzl
Prof. Dr. Peter Widmayer

Algorithms Lab

Exercise – *Antenna*

After the invention of radio, Theirland wants to demonstrate its technological superiority and builds a first radio transmitter. The transmitter must cover the whole population. It is characterized by a location and a transmission radius (within which a reception of the signal is guaranteed). Not surprisingly, transmitters with a higher radius require more advanced technology and more time to build and—last but not least—they cost much more. Thus, the government decided to find a location where the transmission radius is as small as possible, but every single citizen can receive the signal at home. This is not an easy goal to achieve, though. . .

Input The input contains several test cases. Each of them begins with a line containing one integer n ($1 \leq n \leq 200'000$), denoting the number of citizens. The next n lines contain coordinates x_i y_i of homes of citizens (x_i, y_i integral with $|x_i|, |y_i| < 2^{48}$). All numbers on a single line are separated by a single space. The input is terminated by a single line containing `0` (i.e., an empty testcase).

Output For each input, write on a single line the smallest integral transmission radius needed to cover all citizens.

Sample Input

```
2
1 7
31 -6
5
0 0
1 0
2 0
3 0
4 0
0
```

Sample Output

```
17
2
```

S 1: Using the library

Time: $O(n)$, Space: $O(1)$

There is nothing clever on my part going on here. The minimum bounding circle is computed using CGAL. A little ugly conversion code brings the result back to a `long`.

```
typedef CGAL::Exact_predicates_exact_constructions_kernel_with_sqrt K;
typedef CGAL::Min_circle_2_traits_2<K> Traits;
typedef CGAL::Min_circle_2<Traits> Min_circle;
typedef K::Point_2 P;

long ceil_to_long(const K::FT& x) {
  long a = std::floor(CGAL::to_double(x));
  while (a < x) a += 1;
  while (a - 1 >= x) a -= 1;
  return a;
}

long solve(std::vector<P> homes) {
  Min_circle mc2(homes.begin(), homes.end(), true); // Randomise input.
  CGAL::set_pretty_mode(std::cout);
  K::FT radius = sqrt(mc2.circle().squared_radius());
  return ceil_to_long(radius);
}
```