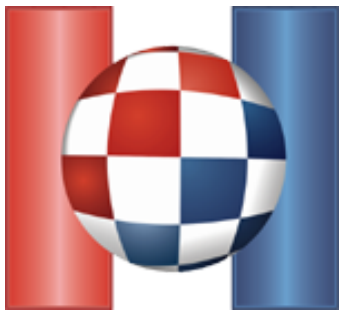
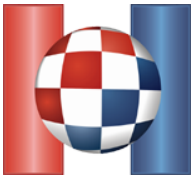


FINAL EXAM 2014

DAY ONE

Algorithms description



For all molecules that can be obtained from the molecule X, the following holds: from each of them, we can (using reverse mutations) obtain X again, which means that from each of these molecules we can get any other molecule. Now it is clear that the set of all molecules breaks down into mutually disjoint classes: from any molecule we can get all the molecules of the same class and no other molecules.

For each of N given molecules, we must determine which class it belongs to, after which we must check for every two molecules if they are in the same class. Furthermore, unfortunately, we can deduce hardly nothing until we come up with a solution which works for molecules small in length and see what the classes look like. Our motivation can be the following: if it turns out that, with a series of mutations, we can reduce any molecule of a certain length to a shorter molecule, then we have considerably simplified our task because then the longer molecules can be reduced to very short ones.

We will generate, for a chosen K, all molecules of the length less than or equal to K and detect their classes (flood-fill with the help of DFS or BFS algorithm). We will solve the second sample test only for $K = 8$, given the fact that `C` and `CGTAC` belong to the same class: therefore, the expansion should be limited to molecules which length is at least eight.

For $K = 8$ we already notice a key feature for solving the task: in each class we find at least one molecule shorter than 5 characters, which means that we can reduce every molecule consisting of 5 characters (using a series of mutations) to a molecule no longer than 4 characters, remaining in the same class. It also follows that molecules with more than 5 characters (such as given N molecules) can be reduced to at most 4 characters by repeating the following procedure: when a molecule is longer than 4 characters, replace its first 5 characters with a previously found appropriate shorter molecule. In the end, we compare the previously detected classes for N shortened molecules.

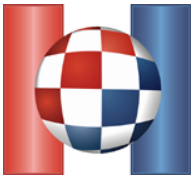
We must ask ourselves if this solution is certainly correct. Flood-fill tells us that molecules shorter than 5 characters generate 24 classes when we expand for $K = 8, 9, 10, 11, 12, \ldots$, which is a firm argument in favour of the claim that the number of these classes remains 24 when we expand for molecules of unlimited length. Intuitively, the following is clear: if between two molecules shorter than 5 characters we haven't found a procedure that can expand the molecule up to 12 characters, then it is almost certain that this procedure doesn't exist between the two molecules.

We provide exact proof to the interested reader. Let letters A, C, G i T represent the following permutations of 8 elements:

$$A = (1, 0, 3, 4, 5, 6, 7, 2),$$
$$C = (2, 5, 6, 0, 7, 3, 1, 4),$$
$$G = (7, 4, 5, 1, 6, 2, 0, 3),$$
$$T = (4, 7, 1, 6, 2, 0, 3, 5).$$

The permutations can be composed. For example, we observe the molecule `GTA` as a permutation we get by applying A, then T, then G to the initial permutation $I = (0, 1, ..., 7)$. In other words, $GTA = G(T(A(I)))$.

The above permutations A, C, G, T are chosen so that $A = TC$, $C = AG$, $G = CT$, $T = GA$. This means that the mutation doesn't change the permutation of the molecule! This means that molecules which generate different permutations can't belong to the same class. Now it is sufficient to find all permutations generated with molecules shorter than 5 characters and check that there are 24 different classes between them. We leave this for the reader to practise.



If we rotate the coordinate plane for 45 degrees (for example, by using substitutions $a = x + y$, $b = x - y$), the problem can be formulated in the following way: find the smallest number D such that all given points can be covered with K squares that have sides parallel to the coordinate axes, the length of the side $2 \cdot D$, and their centers located in some of the given points. In the rest of this description, when a square or rectangle is mentioned, it is assumed that their sides are parallel to the coordinate axes.

Let's first consider the case when $K = 1$, which means that we need to cover the points with exactly one square. The following holds: all points from the set S are covered with square A if and only if square A covers the topmost, lowermost, leftmost and rightmost point from the set S (these four points are called the edge points of set S). It is easy to see this claim is true: if a square covers two points X and Y then it must cover the whole rectangle which has the line segment $\overline{XY}$ as its diagonal. Therefore, if a square covers all edge points, then it covers the whole edge of the rectangle which has these points on its sides and then necessarily covers all other points from set S.

Now we can easily construct a fast $O(N)$ algorithm for the solution to this case:

- We find the topmost, lowermost, leftmost and rightmost point.
- For each point P we calculate the smallest required D if we assume the center of the square is exactly the point P. Since it is sufficient that P covers four edge points, this number is exactly equal to the greatest distance between P and some of the four points.

Before we consider the case when all points need to be covered with two squares, let's reformulate the previous claim about how simple it is to determine whether there is a square of given side length which covers all points from set S. For a given set of points S, let x_1, x_2, y_1, y_2 be x-coordinates of the leftmost and rightmost point and y-coordinates of the lowermost and topmost point, respectively. Therefore, the whole set S lies within the edge rectangle where its opposite angles are (x_1, y_1) and (x_2, y_2). Let D be a given integer. From the previous claim, it easily follows that the set S can be covered with a square, its side length being $2 \cdot D$, its center being a point from the set S, if and only if inside the rectangle whose upper right corner is $(x_1 + D, y_1 + D)$ and lower left corner $(x_2 - D, y_2 - D)$ there exists at least one point from the set S. For the given set S and integer D, that rectangle is considered the critical rectangle for set S and integer D. If the point $(x_1 + D, y_1 + D)$ is not beyond and to the right of the point $(x_2 - D, y_2 - D)$, then we consider that the critical rectangle is empty.

If $K = 2$, we solve the problem using binary search: it is obvious that if we have a procedure which, for a given D, determines whether the given set of points can be covered with two squares whose side is of the length $2 \cdot D$, then we can use binary search to find the smallest such D. If the mentioned procedure is of the complexity $O(f(N))$, then the total solution is of the complexity $O(f(N) \log M)$ where M is the greatest possible coordinate value. In the rest of this description, we only focus on this procedure.

The procedure can work in the following way: for a given set of points S and an integer D:

- For each point A from set S
  - Assume that one square will have its center in A
  - Calculate the edge rectangle for set SA that can be obtained so that all points are omitted from set S that are covered with the rectangle whose side is of the length $2 \cdot D$ and its center is in A

– Go through all points from set S and check whether there is a point located inside of the critical rectangle of SA.

The time complexity of the direct implementation of this procedure is $O(N^2)$ so we get a total solution of the complexity $O(N^2 \log M)$, which is not fast enough for all points.

A quick enough procedure works conceptually in the same way, but instead of calculating the edge rectangle point by point, it calculates all necessary edge rectangles in one step. Additionally, instead of checking whether a point exists inside of an edge rectangle, this procedure checks simultaneously whether a point exists inside any of the critical rectangles.

For a given set S and integer D:

1. For each point A from set S, calculate the edge rectangle of SA.
2. For each found edge rectangle, calculate the corresponding critical rectangle.
3. Check whether a point from S exists inside of any of the found critical rectangles.

Step 1 can be made in linear time using 4 sweeps.

Step 2 can be made in time complexity of $O(N \log N)$ by using sweep inside which we use the logarithmic structure to find the number of active rectangles which contain the given point.