Demarcation spoiler

Spoiler by Karolis Kusas

First of all, let's introduce some operations which are used in each subtask.

1. *Rotate* - the figure is rotated 90 degrees around the axes origin. This is done by swapping x and y coordinates and negating the y coordinate of each point.
2. *Move* - the figure is moved in the way that the leftmost (and lowermost if there are many) point is moved to the axes origin.
3. *Reflect* - the figure is reflected by negating the y coordinate of each point.
4. *Compare* - checking whether two figures are congruent. This is done by *moving* the first figure and *rotating*, *moving* the second figure (four times) and each time checking if both sets of points are equal. If no, then it is tried to *reflect* the second figure and run the same operations once again. If this is unsuccessful, then the figures are not congruent.

Additionally, it is needed to implement a function which cuts the figure along the given segment. The result of this function - two new figures.

The main idea - when considering two figures, they can be congruent only if they are of equal perimeters or areas. There are at most one horizontal and one vertical segment cut that gives such two figures. It can not cross any other edges of the polygon. Otherwise, there may be many segments.

Moreover, we can solve the problem considering only the vertical cuts. We can rotate the polygon and use the same solution - the horizontal cuts will be considered too.

Partial solutions

Subtask 1 (12 points). $4 \leq N \leq 100000$. Any horizontal or vertical line that divides the polygon divides it into exactly two parts.

We can see that it is possible to apply the binary search to find the x coordinate of the cut. If the left perimeter is bigger that the right, we consider the left part, if the left perimeter is smaller, we consider the right part. If they are equal, we get the end points of the cut. Then we cut the figure and *compare* the parts.
The complexity of this algorithm is $O(n \cdot log(x_{max} - x_{min}))$.

Subtask 2 (15 points). $4 \leq N \leq 200$.

Let's loop over all pairs of horizontal edges. In each loop, calculate the sum of lengths of edges between these edges. Get the x coordinate of a cut which divides the polygon into two parts. Check whether this segment intersects other edges. If no, then make a cut and *compare* the parts.
The complexity of this algorithm is $O(n^3)$.

Subtask 3 (23 points). $4 \leq N \leq 2000$.

We can easily improve the algorithm which was used in the previous case by precomputing the prefix sums of lengths of edges.
Other more sophisticated algorithm uses *two pointers* paradigm.
Firstly, we set both pointers (E and F) to point to any horizontal edge. We keep variables A and B (distances between the edges pointed by E and F from the both ends). While $edgeLength(E) + edgeLength(F) \geq A - B$ we move pointer F to the next horizontal edge, otherwise we move pointer F to the previous edge and E to the next edge. Before moving pointers, we try to find the x coordinate of a cut whose ends are on these edges and which divides the polygon into two figures of equal perimeters. Then we check whether this segment intersect other edges. If no, then make a cut and *compare* the parts.
The complexity of both algorithms is $O(n^2)$.

Subtask 4 (50 points). $4 \leq N \leq 100000$.

There are two main ways to implement an effective algorithm (based on equal perimeters or equal areas). We will describe the first one.
The idea is to use *line sweep* paradigm.
Make events for all edges which include following information: the x coordinate (the end of the horizontal edge), the y coordinate of the edge, the index of the edge, two status variables which show whether this is the left or the right end and the orientation of the edge.
Also let's keep a set with the y coordinates and some additional information for all edges intersected by the cut with the given x coordinate.
When processing each event we find the influenced edge in the set and the uppermost lower edge as well as the lowermost upper edge. Then we calculate the x coordinates of possible segment cuts. If we find any, we insert it to the queue of events. When our x sweep line hits that x coordinate, we check whether the segment ends (two y coordinates) are still neighbours (one immediately after another) in the set. If yes, we can perform a standard check with this segment cut. Furthermore, we are guaranted that it won't intersect any other edges.
The complexity of this algorithm is $O(n \cdot logn)$.