Rectilinear Polygon Operations for Physical Design

I. Introduction

Rectilinear polygon processing is a very common problem in Physical Design because there are many elements in the physical design flow that are represented by rectilinear polygons, such as macro blocks, cell pin shape, standard cell row regions, floorplan channels, ... and so on.

Therefore, in the physical design stage, it is often necessary to perform the merging, clipping, splitting, etc. processing on the rectilinear polygons, and obtain the results for further analysis or application. For example, "merge" operation is used to merge multiple rectilinear polygons connected to handle repeated descriptions of overlaps and reduce the number of rectilinear polygons to improve program execution efficiency. "Clip" operation is used to delete some special areas, and "split" operation is used to convert rectilinear polygons into rectangles to simplify the complexity of the problem.

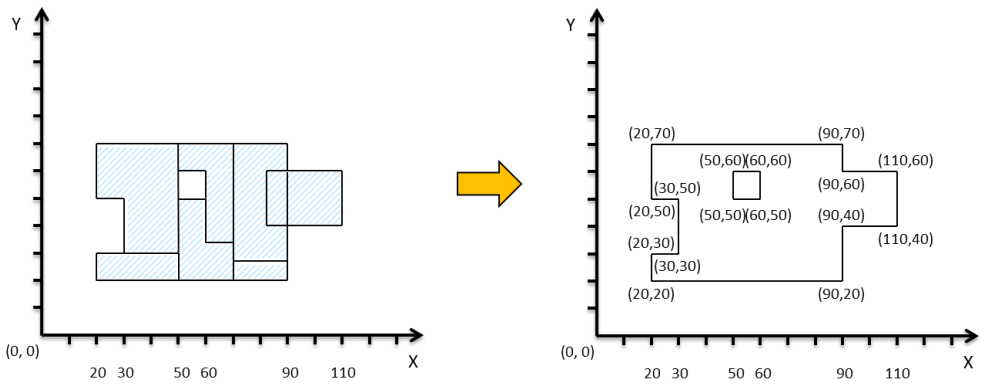
There are some references and methods for the basic processing of rectilinear polygons. But the key is that when applied to physical design flow, the number of polygons is often very large, which makes the program implementation more difficult.

II. Problem Description

The program must be able to perform the following rectilinear polygon operations, including support the rectilinear polygon with “hole”.

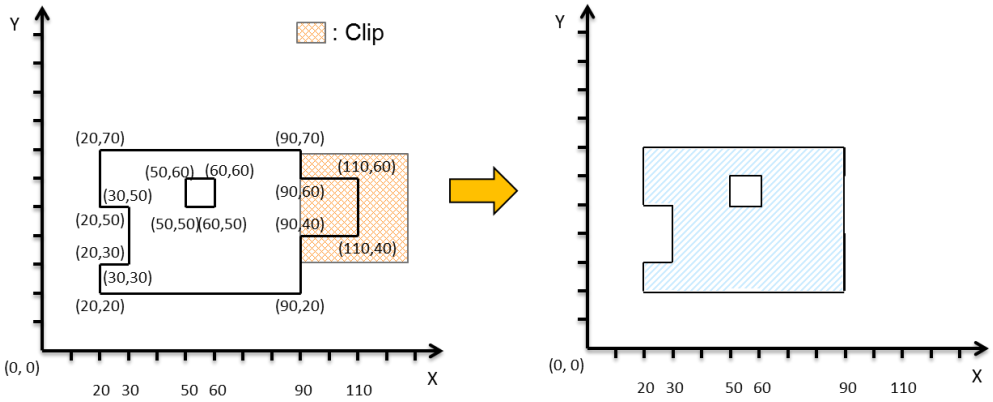
A. Merge

After the “Merge” operation, all connected rectilinear polygons (including edges) must be merged into a rectilinear polygon, as shown below:



B. Clip

The “clip” operation deletes the intersection between the original rectilinear polygons and the clip rectilinear polygons, and obtains one or more new rectilinear polygons, as shown below:

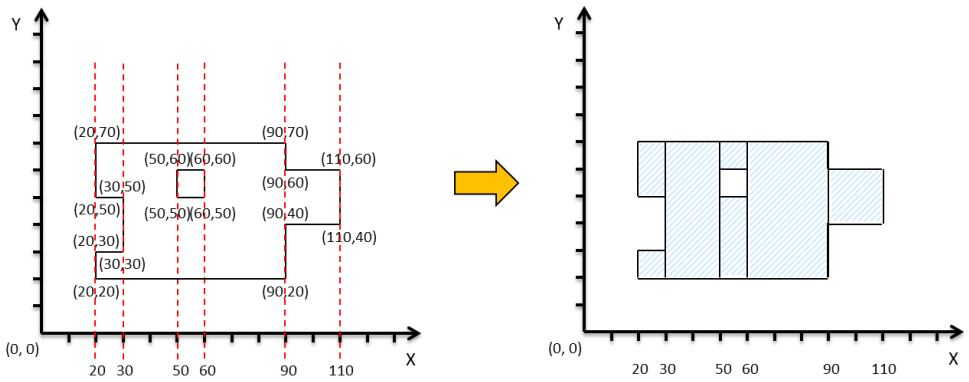


C. Split

Convert the rectilinear polygon into multiple rectangle representations, and there must be no overlap in the results except the rectangular edges. The function must contain these three types:

1. split_H: completed by **horizontal** cutting
2. split_V: completed by **vertical** cutting
3. split_O: Horizontal and vertical cuts can be mixed to cut out the **minimum rectangles** for the optimize target

The following is an example of "Split_V":



In testcases, a combination of various processes will be described. For example, “M C SH” means to perform Merge → Clip → Split_H for the input rectilinear polygon.

III. Example of Input/Output Files

- Input File:

OPERATION M1 C1 M2 SV ;

DATA MERGE M1 ;
POLYGON 0 0 100 0 100 100 0 100 0 0 ;
POLYGON 100 0 200 0 200 100 100 100 100 0 ;
END DATA

DATA CLIPPER C1 ;
POLYGON 50 50 150 50 150 150 50 150 50 50 ;
END DATA

DATA MERGE M2 ;
POLYGON 0 100 200 100 200 200 0 200 0 100 ;
END DATA

- Output File:

RECT 0 0 50 200 ;
RECT 50 0 150 50 ;
RECT 50 100 150 200 ;
RECT 150 0 200 200 ;

- Description:

The first line of the input file describes the operation requirements, and please follow the order of the operations. The operation definition begins with the keyword "OPERATION" and each operation is separated by a space. In the example, "M1" represents a “merge” operation, and the polygons that need to be processed are described in the "Data M1" section below. “C1” represents a “clip” operation, and the polygons data are described in the “Data C1” section. “M2” continues to merge the polygons described by “DATA M2” base on the operation “M1 C1” result.

The "split" operation does not require the data section, but there are three keywords that represent different ways of cutting: "SV" means completion by vertical cutting, "SH" means completion by horizontal cutting, and "SO" means completion by optimized cutting. In the example, the last operation is to split the previous result by vertical cutting. The important rule

is that each merge operation must contain the previous results and the final operation will always be "split".

The data section of each merge or clip operation begins with the keyword "DATA" and continues with an operation. All of operation shapes are defined as rectilinear polygon points, such as X0 Y0 X1 Y1 and so on, and end with “END DATA”. The description of polygon may be clockwise or counterclockwise, and the last set of coordinates may not be the same as the first set of coordinates. Hole is not described in the test file, so there is no format for the description of the hole. But multiple polygon descriptions may cause holes, please consider when developing the program.

Since the last operation is always "split", the output file should only contain rectangles. Each rectangle should start with the keyword "RECT", and the points are the lower left X, the lower left Y, the upper right X, and the upper right Y sequentially.

IV. Language

Please implement your program in C or C++. The binary file should be called as “myPolygon”. Please follow the following usage format:

./myPolygon input_file output_file

V. Platform

OS: Linux
Compiler: gcc/g++

VI. Testcases

5 open testcases can be downloaded
3 non-disclosure testcases

VII. Evaluation

The score is divided into two stages. When the first stage score is the same, the second stage score is performed. The two-stage score is as follows:

Stage1:

The score is 25 for the open testcases and 75 for the non-disclosure testcases. If the program encounters compilation errors, crash (core dump),

runs on the test platform provided by the conference for more than 8 hours, the result incorrect, the score for this testcase will be 0. Additionally, if the last operation is SO, the number of result rectangles are must less than or equal to min(SV, SH), or the result is considered incorrect. Here min(SV,SH) represent an max base for normalizing and it will be calculated by scoring checker at first. When the first stage total score is the same, the second stage will be scored.

Stage2:

The second stage score includes the CPU time and the number of rectangles of the final result, with weights of 70% and 30%, respectively.

Suppose the testcase has m teams completed correctly, T_i means the i^{th} team time performance**,** and the score will be St_i. N_i means the i^{th} team number of result rectangles and the score will be Sn_i. Additionally, if the last operation is not SO, the CPU time performance will dominate all of score for that testcase.

Normalize functions

$$St_i = 1 - \frac{T_i - \min(T)}{\max(T) - \min(T)}$$

Where $T = (T_1, \dots, T_m)$ and St_i is now i^{th} team normalized CPU time score

$$Sn_i = 1 - \frac{N_i - \min(N)}{\min(SV, SH) - min(N)}$$

Where $N = (N_1, \dots, N_m)$ and Sn_i is now i^{th} team normalized number of result rectangle score, and Sn_i is calculated by 0 if Sn_i is less than 0.

Total score function

$$S_i = \sum_{C=1}^{5} \left(St_i^C * 70\%\right) + (Sn_i^C * 30\%)$$

Where C means case and $C = (C_1, \dots, C_5)$; $S = (S_1, \dots, Sm)$ and S_i is now i^{th} team total score