2003 IC/CAD Contest

Problem 6: Area Computation of Hierarchical Layouts

Source: Synopsys Taiwan Limited

March 19, 2003

1. Introduction.

Physical layouts are composed of many layers of geometric data. Each layer, by nature, includes a large amount of polygons. They are usually represented as hierarchical structures of cells to reduce complexities. Given a hierarchy of a layer, the objective of the problem is to compute the total area of all polygons in the flatten sense. For example, consider Fig. 1. The top cell, Cell A, contains one 30 by 30 polygon at (10, 20) and two reference cells of Cell B at (20,10) and (60,10) respectively. Cell B contains only a 30 by 10 polygon at (0,10). If we flatten all of polygons in Cell B into Cell A, the flattened geometry is shown in Fig. 2. The objective is to find the total area of polygons in Fig.2 and the answer is 1300.

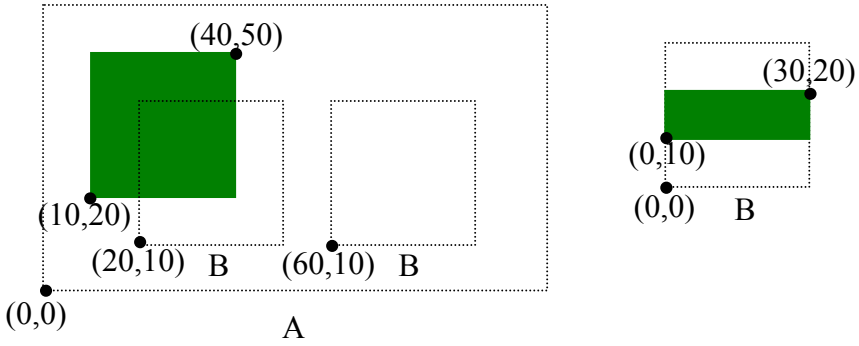


Fig. 1

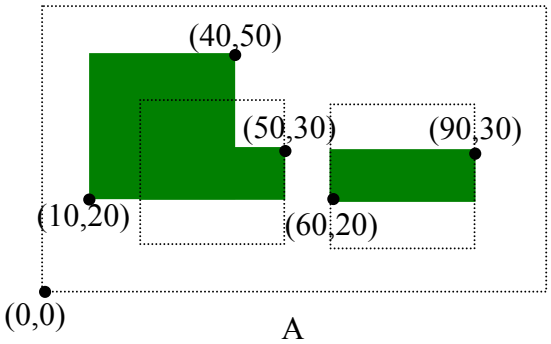


Fig. 2

2. Input/Output Format.

The following is the Bachus Naur Form (BNF) of our input format:

```
<Input> ::= 'BGNLIB' <Cell>* 'ENDLIB'
<Cell> ::= 'BGNCEL' <Name> <Element>* 'ENDCEL'
<Element> ::= <Polygon> | <Reference>
<Polygon> ::= 'BGNPOLY' <XY>* 'ENDPOLY'
<Reference> ::= 'BGNREF' <Name> <XY> <Reflection> <Rotation>
'ENDREF'
<Reflection> ::= '0' | '1'
<Rotation> ::= '0' | '1' | '2' | '3'
<XY> ::= <X-coord> <Y-coord>
<Name> ::= <String>
<X-coord> ::= <Number>
<Y-coord> ::= <Number>
```

Each token is separated by white space, tab or new line characters. <String> could be composed by any other printable ASCII characters including digits and non-alphabet characters. <String> should not conflict with any keywords used in this format. <Number> could be any 32-bit signed integer. A cell reference could be rotated about its original point (0,0) or reflected about its X-axis. The values 0, 1, 2 and 3 of <Rotation> mean to rotate the corresponding cell about the original point (0,0) counterclockwise by 0, 90, 180 and 270 degrees respectively. If <Reflection> is 1, the cell is reflected about the X-axis BEFORE rotation. Otherwise, no reflection is necessary. Please notice that, for a cell reference B in the definition of cell A, cell B will be put into a containing cell A by the following steps: First, do the necessary reflection and rotation for cell B according to the values of <Reflection> and <Rotation>. Then, put the result into A by matching the original point (0,0) of the result of B with the point <XY> of A. For a <Polygon>, the <XY> point list could start at any point by either clockwise or counterclockwise order. Furthermore, the edges formed by the point list should have no self-intersection. Notice that it is ok if there is no obvious (unique) top cell in a file.

Consider the example in Fig.1. The input file could be as follows:

```
BGNLIB
BGNCELL A
```

```
BGNREF B 20 10 0 0 ENDREF
BGNPOLY 40 20 40 50 10 50 10 20 ENDPOLY
BGNREF B 60 10 0 0 ENDREF
ENDCELL
BGNCELL B
BGNPOLY 0 10 0 20 30 20 30 10 ENDPOLY
ENDCELL
ENDLIB
```

The output format is as follows:

Runtime = xxx.xx seconds
Memory = xxx.xx MB
Area = xxxx.x

Please report the run-time and memory consumption for your program. Please report the "Area" as a floating-point number with at least 5-digit precision (more is ok). A possible answer for the example in Fig.1 is as follows:

Runtime = 1.5 seconds
Memory = 5.25 MB
Area = 1300.0

III. Language and Platform

1. Language: C or C++
2. Compiler: GNU gcc or g++ with optimization flag -O
3. Platform: Sun Solaris and Linux

IV. Performance Evaluation.

The performance evaluation will base upon correctness, speed and memory consumption. Robustness is also taken into consideration. Your program should have the ability to detect the syntax errors of the input file and the self-intersection of edges in a <Polygon>. In order to simplify the problem, all the edges of polygons in the testcases are either vertical or horizontal. It is a plus if your program or

algorithm can handle cases with non-vertical and non-horizontal edges. Please specially demonstrate this capability in your report.

Please make sure that your source code could be compiled successfully with optimization flag –O by gcc or g++ under Solaris AND Linux platforms. Your program should accept two arguments in the command line. The first one is the input file name and the second one is the name of the cell that you want to compute the area. Suppose that the file name of the compiled executable is "a.out" and the input file name of our previous example is "case1.dat". Please make sure your program could be launched by the following command in UNIX:

```
% a.out case1.dat A
```

The total area should be equal to 1300.0 in this case. If you invoke your program as

```
% a.out case1.dat B
```

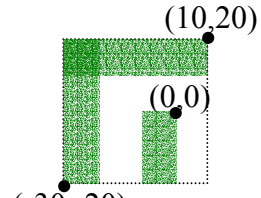
Then, only the total area of cell B is computed and it is equal to 300.0 in this case.

Due to ease of evaluation, please do not add unnecessary message in the output.

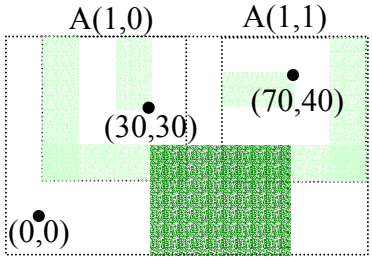
V. Further Examples

A more complicated example is shown below. We use A(p, q) is to denote a cell reference of cell A with <Reflection> = p and <Rotation> = q. The total area of cell TOP is 3800.0.

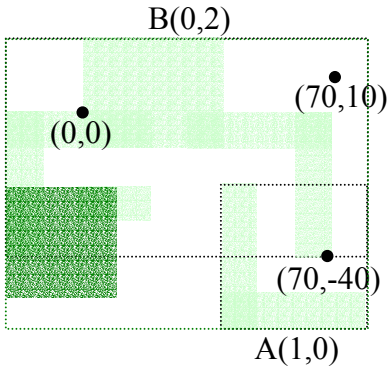
```
BGNLIB

BGNCELL A
BGNPOLY
10 20 10 10 -20 10
-20 -20 -30 -20 -30 20
ENDPOLY
BGNPOLY
-10 -20 0 -20 0 0 -10 0
ENDPOLY
ENDCELL

BGNCELL B
BGNREF A 30 30 1 0 ENDREF
BGNREF A 70 40 1 1 ENDREF
BGNPOLY
30 20 30 -10 70 -10 70 20
ENDPOLY
ENDCELL

BGNCELL TOP
BGNREF A 70 -40 1 0 ENDREF
BGNREF B 70 10 0 2 ENDREF
BGNPOLY
-20 -20 10 -20 10 -50 -20 -50
ENDPOLY
ENDCELL

ENDLIB
```



cell A



cell B



cell TOP