Discrete Mathematics
Program Assignment #02
2019.11.15

21800294 박수아 21700355 서인아
21300362 서상원 21500388 안제인
21300750 진영인

< 1~3. Fractal >

1. Introduction

In this study, we will show three fractal image using recursive function in JavaScript program. A fractal is a geometric object whose structure is identical to the structures of its components. Fractals exhibit similar patterns at increasingly small scales or decreasingly large scales called self-similarity, also know expanding symmetry or unfolding symmetry. Many structures in nature are formed as fractal. In order to represent as a computer image, the same structure is repeated by using Java script, size or depth are gradually reduced or increased. For check the fractal program, firstly we will look to see approach include solution design and Pseudo code or explanation of code. Second, we will show that our solution is good through the test. Finally, we will discussion about difficulty.

(Fractal 1)

1. Approach

1.1) Solution Design

Create a JavaScript program by receiving the user's input through text box with two parameters, depth and length. The length is set to 100 by default. If user inputs parameter and the draw button is pressed, the program is execution and result is seen in canvas. If the clear button is pressed, canvas is clear.

1.2) JavaScript Pseudo Code

```
function drawPlus(x,y,depth,length,dir)
        if depth is not 0, then
        newdir <- (dir+2) module 4

        for i = 0 to 4, i++
        if i is not newdir
        drawLine(x,y,x+(dx[i]*length),
y+(dy[i]*length))

        for i = 0 to 4, i++
        if I is not newdir
        drawPlus(x+(dx[i]*length),y+(dy[i]*le
ngth),    depth - 1, (1/2)*length, i)
```

The direction information is in the dx, dy array. It was set up to run for loop by dx= [1,0,-1,0], and dy=[0,1,0,-1].

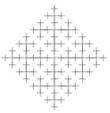
When the recursion, the self-directional information is memorized, and then (direction+2) % 4 is used to obtain the opposite direct, and draw only when it is not the opposite direction.

2. Result

A. Depth:1 Length:100 / Depth:5 Length:100

3. Discussion

A. Apply Recursive function to all lines



As like shown from the side image, at first, the fractal shape that had to be made by applying the recursive function to all lines did not come out. It was difficult to designate a line that did not apply the recursive function because we have to apply recursive function at four end points of the line made in the previous step. Because direction of four end points is each different. After continuing fix the code, we found that When the recursion, the self-directional information is memorized, and then (direction+2) % 4 is used to obtain the opposite direction and draw only when it is not the opposite direction.

(Fractal 2)

1. Approach

1.1) Solution Design

Create a JavaScript program by receiving the user's input through text box with one parameter, depth. If user inputs parameter and the draw button is pressed, the program is execution and result is seen in canvas. If the clear button is pressed, canvas is clear.

1.2) JavaScript Pseudo Code

```
angle = Math.PI/2.1;
function drawTree(x1,y1,x2,y2,angle,depth)
        if depth is 1, then
        context.moveTo(x1,y1);
        context.lineTo(x2,y2);
        if depth is more than 1, then
        xa <- x1+ 6/13*(x2-x1);
        ya <- y1+ 6/13*(y2-y1);
        xb <- x1+ 7/13*(x2-x1);
        yb <- y1+ 7/13*(y2-y1);
        xc <- xa + (xa - x1) * Math.cos(-angle) -
(ya - y1) * Math.sin(-angle);
        yc <- ya + (ya - y1) * Math.cos(-angle) +
(xa - x1) * Math.sin(-angle);
        drawTree(x1,y1, xa,ya, angle ,depth-1);
        drawTree(xa,ya, xc,yc, angle, depth-1);
        drawTree(xc,yc, xb,yb, angle, depth-1);
        drawTree(xb,yb, x2,y2, angle, depth-1);
```

2. Result

A. Depth:1 / Depth:5

3. Discussion

A. Function call condition

At first, we made the fractal image like you can see above. The reason for this creation, I called the drawLine function without any condition to the drawTree function, so the lines that should be removed when the fractal shape appeared remained intact. In other words, every time we call a drawTree function, It made a line connecting the coordinates corresponding to the parameter. To solve this problem, only if depth is one, call drawLine to draw a line, and if the depth is not 1, only a return function, drawTree, was called.

(Fractal 3)

1. Approach

Create a fractal image that is interesting, creative unique fractal using JavaScript program.

1.1) Solution design

The third fractal, a free theme, represents tree fractal, which runs code that increases the depth every second to generate a tree fractal. Also, the angle of the fractal code with the original -90 angles is set to +90, and one more tree fractal code that grows upside down is executed to express it like a human lung. The tree fractal expressed like animation so shown growing tree. After showing the tree fractal as much as the desired depth, it expresses the flower with the fractal of pentagon shape.

The reason for choosing this theme starts with the question of whether there is something in the body that can be expressed as fractal. We found out that human lung stem cells were similar to the shape of trees. So, we thought it would be good idea to express that when tree grow, it is an important resource for the supply of oxygen, so they have a great influence on the lungs of people who need oxygen.

1.2) Explanation of code

A. drawTree function for tree fractal. Through this code, a tree of depth d (is mean depth parameter) has one stem and six branches. The length of the stem is d. The angles between the stem and the six branches are +150°,-150°,+174°,-186°,-125°,+128°. Each branch is in a form of a tree of depth d-1.

B. drawTree2 function is for express lung stem cell. While the same as the drawTree function, when calling drawTree2 function to express lung stem cells, the angle_d, the parameter of the function, is given at the original value+67, which returns to the original given angle_d when the depth is less than 3.

C. drawFlower function is for express flower using pentagon shape. When the depth is 0, pentagons are made using given x and y coordinates. When the depth is not 0, takes the length of the scaled side + length of the gap at the top of the highest pentagon to calculate the top vertices of the others. The top position pentagon form depth-1 pentagon, then move to the other side of pentagon by changing direction.

D. Using setInterval method to show the tree grows every 1 seconds. Also using setTimeout method to appear flower after trees growing.

2. Result

A. Canvas background is set up as image. The image background is a picture of the upper body of a person with a grassland picture and a lung shape.

B. Every 1 seconds, As the depth increases by one, fractal appears like a tree grows.

C. In the background of a human lung, the angle of the fractal tree is set to +90, so the tree grows upside down and expresses it like a lung stem cell. This also increases the depth every second.

D. When the tree fractals appeared as much as the given depth, the flower-shaped fractal is shown in the grass of the background image.

3. Discussion

A. Angle of tree branch.

5 branches look better in tree, but we need 6 branches for express lung stem cell. We wanted to find the idea satisfy both of these, then we overlapped the angle of the branch so two branches look like one branch when expressed the tree. But set angle that one is +174° and other is -186°, allowing it to split later when changing the angle.

< 4. Triangulation>

1. Introduction

Triangulation is a program that divides a simple polygon into several sub-triangles. Triangulated triangles' vertices are equal to polygon's vertices. (Triangulation of a simple polygon is a partitioning of its interior into triangles such that the vertices of triangles are also the vertices of the polygon.)

This program gets n numbers of (x,y) coordinates in order and form a polygon on 100 x 100 size canvas. Then, the polygon is divided into (n-2) triangles by drawing (n-3) diagonals.

If input coordinates are not in right order (neither counterclockwise nor clockwise), the program regards it as not a polygon and draws nothing. In geometry, a simple polygon is a polygon that does not intersect itself and has no holes.

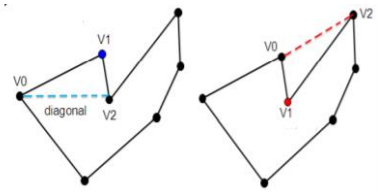
2. Approach

1.3) Solution Design

Firstly, the program checks if the input polygon is a simple polygon. A simple polygon is determined by checking whether the sum of the inner angles of polygon is equal to (n-2)*180. (We assumed that there could be a little error while measuring angles. Thus, we have set the range of the sum of the inner angles of polygon with ±0.5 degrees)

After determining that it is a simple polygon, the program tries to make inner sub-triangles by diagonalizing lines from a vertex to another. When dividing polygon, the diagonal must not be intersecting with any other lines of the polygon. Recursive function is used while finding appropriate diagonalized lines. To find a point where diagonal line is available, designate a point and make a triangle with the points aside to a designated point. If there does not exists any point inside a triangle, the corresponding triangle is valid. (It can be diagonalized)

However, if there exists any point(vertex) inside a triangle, the corresponding point cannot be set as diagonalized point.



All points are saved in an array, so that program searches through the array for points that can draw diagonal and triangulate polygon.

If program finds a point that can draw a diagonal, it uses recursive function to draw a polygon with the rest of the points and search for points for triangulation. This process is repeated until number of remaining points equal to three.

3. Result

A. This program triangulates only when points are ordered in clockwise or counterclockwise. If input is not in one of two, the program does not triangulate.

If input points form a simple polygon, program shows triangulation result where polygon is divided into (n-2) triangles.

B. The test case went well when a convex polygon does not produce a diagonal intersection.

C. The test case for a simple polygon with a complex diagonal intersection went well.

e.g.

A.

Not a Simple Polygon!

B.

C.

4. Manual

3.1) Prerequisite

A. Input points must be in order of clockwise or counterclockwise.

B. If input points cannot form a simple polygon, program does not triangulate.

C. If input points form a simple polygon, program shows one of the possible results of triangulation.

3.2) Input format

A. X and Y coordinates must be entered in text boxes. All coordinates must be in range between 1 to 100.

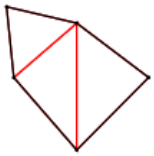
3.3) Instructions

Input: input x, y coordinates must be values between 1 to 100. Polygon is drawn along the order of inputs.

e.g. (1,1) (40,10) (80,40) (40,80) (5,40)

Output: Polygon is drawn in order of input, and if poly forms a simple polygon, triangulation result will be shown with divided sub-triangles.

e.g.



5. Discussion

A. Polygon must be determined as a simple polygon in order to do Triangulation. We found out several properties that must be satisfied in order to determine input polygon as a simple polygon.

The definition given above ensures the following properties:

- A polygon encloses a region (called its interior) which always has a measurable area.

- The line segments that make up a polygon (called sides or edges) meet only at their endpoints, called vertices (singular: vertex) or less formally "corners".

- Exactly two edges meet at each vertex.

- The number of edges always equals the number of vertices.

Two edges meeting at a corner are usually required to form an angle that is not straight (180°); otherwise, the collinear line segments will be considered parts of a single side.

Most of all, a simple polygon can be divided in to (n-2) triangles. Thus we learned that (n-2)*180 formula can be used to determine whether polygon is a simple polygon or not.

B. While calculating inner angles of polygon, we found that inputs are in reverse order when input are entered in clockwise. From this problem, we have found out that by reversing the index of points, angle calculation can be done correctly.

C. We had to draw a polygon in the order in which the user entered the points. The reason is that if you draw polygon regardless of the order, there can be several different kinds of polygons with the same dots. Therefore, we thought that it is necessary to specify the order of given points to determine whether it was a simple polygon or not.

D. Although there are various ways of triangulation, in order to use recursive function to do triangulation, we thought that we needed a method that can split triangles one by one. Thus, we set conditions for vertices first to look for divisible triangles and repeated the process to implement triangulation in recursive function.

<5. CNF Converter>

1. Introduction

CNF Converter is a program that transforms a propositional formula into a Conjunctive Normal Form (CNF). CNF is a two-level propositional formulas which has form of $(a_{11} \vee a_{12} \vee \cdots \vee a_{1n1}) \wedge (a_{21} \vee a_{22} \vee \cdots \vee a_{1n2}) \wedge \ldots (a_{m1} \vee a_{m2} \vee \ldots \vee a_{mnm})$ where a_{ij} is p or $\neg p$ for an atomic propositional variable p.

Every propositional formula has an equivalent CNF form. So our program will consume the propositional formula and produce the result through several procedures.

2. Approach

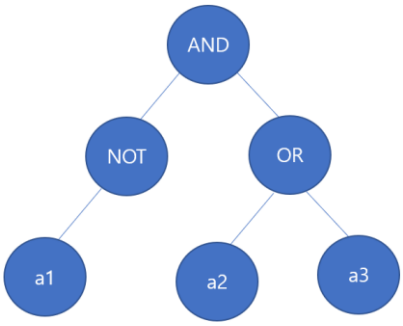
When given a propositional formula, program should change and print it in CNF format.
All input must be prefixed in the parentheses. At output expression, the numbers in the same row are all connected by disjunction, and each row is connected by conjunction. And the program must print out an error message if the given input does not follow that rule.

2.1) Solution Design

Our CNF Converter follow this procedure to solve the problem. Each procedure has detailed description to help reader understand.

[1] Parse the proposition formula

First, input will be split by space so that each expression goes into binary tree structure.
For example, if the propositional formula $(\text{and } (\text{not a1}) (or\ a2\ a3))$ converts into tree structure, the form or tree will be like this.



<Graph 1. Tree structure>

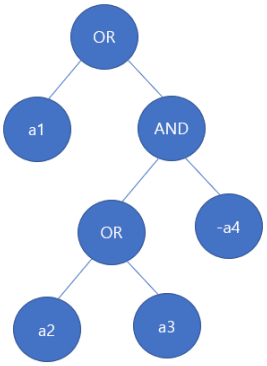
Each propositional variable will be a node and has it's own value. (AND : 1, OR : -1, NOT 0)
When each expression goes into binary tree, there are some rules. First, if the node data is zero, the subtree must be created on the left node. Second, if a node data has a value 1 or -1, it has to check if the left node or right node is null. If the left node has a null value, make a subtree on the left node, and if the left node is full, go to the right node and make a subtree.

[2] Negate all child nodes of 'NOT' nodes and delete it

After the all variable goes into tree, the NOT node must be eliminated. Before that, all child nodes of NOT node must be negated. AND becomes OR, OR becomes AND, and each numerical variable will be negative integer. After all nodes negated, NOT node will be deleted in tree.

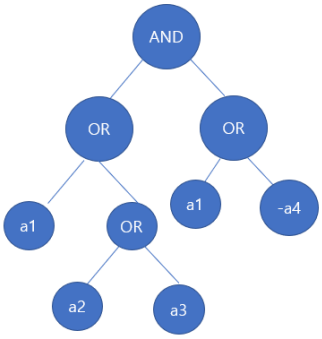
[3] Apply the distributive law

To apply distributive law, Let's see the structure of tree. If you see graph 2, you can see that the root node is OR node. To interpret the final expressions, we should make the form of disjunctive clauses.



<Graph 2. Before the distributive law>

So, to do that, we must place AND node to the root, and apply the distributive law. The left child of root node will be bound with child nodes of AND node.



<Graph 3. After the distributive law>

[4] Interpret the final expressions and print the result.

Now we can extract disjunctive clauses from tree. All numerical value node who has OR node for the parent will be the line of output result. With the graph above, the output will be a1, a2, a3 for one line and a1, -a4 for second line.

3. Results

For the right input,

```
(or a1 (not (or (not (or a2 a3)) a4)))
```

<Image 1. Right Input>

It will return the list of disjunctive clauses at the end of program.

```
1 2 3
1 -4
```

<Imgae 2. Right Result>

But, for the wrong input like incorrect parenthesis input,

```
(or a1 (not (or a2 a3))(
```

<Image3. Wrong Input>

It will return error message for the wrong input that isn't propositional formula.

```
Wrong input.. Exit program!
```

<Image4. Error message for the wrong input>

4. Discussions

The lesson we learned from this problem was it is really hard to make parser and interpreter of new expressions. The program we made is not perfect. Some input works perfectly, but some doesn't. We tried to make algorithm that converts certain expression to the value from several example expressions. But in that way, it couldn't cover all expressions. We needed some grammar that cover all kinds of expressions. From that point, it was like making new language interpreter. Propositional logic must be converted into binary tree and by its own grammar and finally it was interpreted. It was little bit different with coding that we did before. Because we had to handle new expressions, it was not easy to make perfect grammar that cover all expressions. It was really good chance to experience this kind of problem.