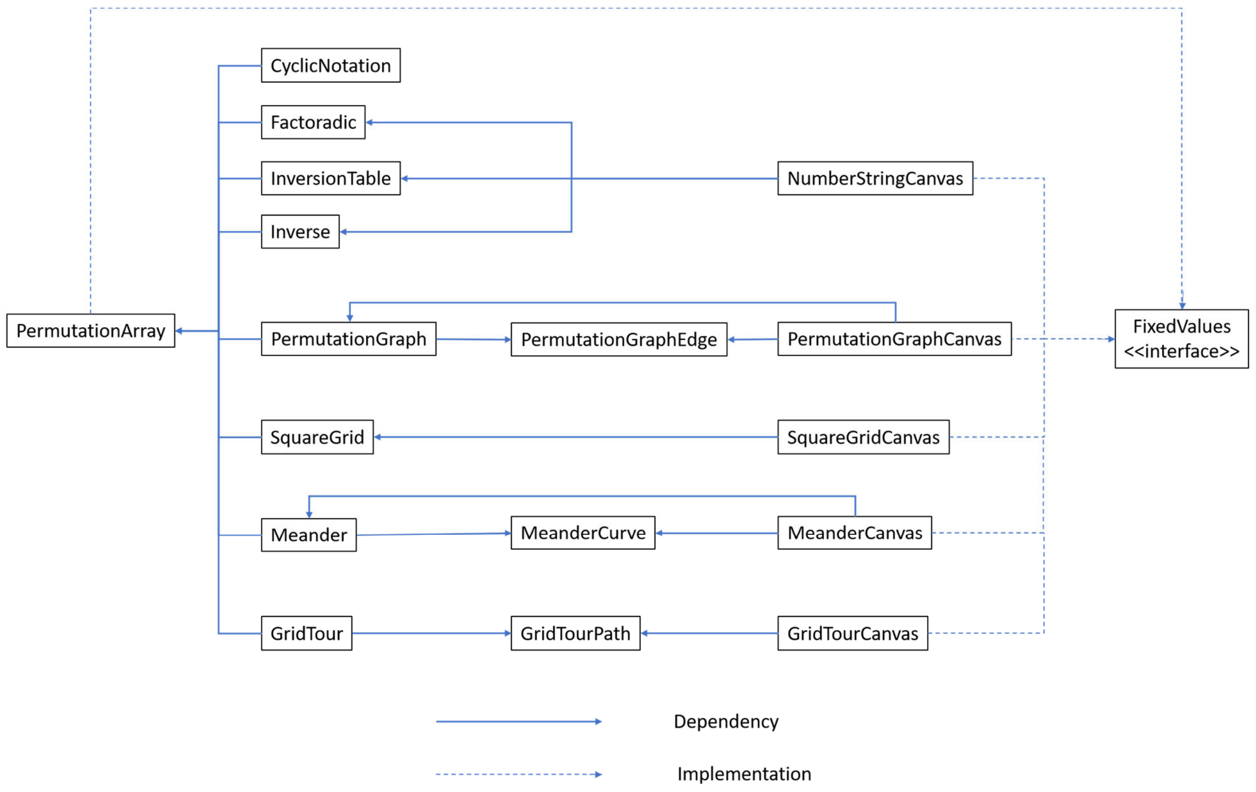
1. Framework



2. Permutation

The permutation is represented by an integer array (class PermutationArray). Each entry is a number in the permutation. Numbers are saved following their sequences in the permutation.

This class also has 2 important methods, checkInput() and reArrangement(). checkInput() checks if an user input is a real permutation, and if its size is more than 12. reArrangement() treats the permutation as a cyclic permutation, and re-writes it to make it starting from 1. For example, permutation (2 4 1 3) would be re-wrote to be (1 3 2 4). This method is useful when we find out the meander and grid tour. I will explain the usage in Section 8 and 9.

3. Cyclic notation

The class CyclicNotation has a helper class, sequence. A sequence object is an integer array. It represents a “cycle”. The cyclic notation is represented by an array of sequence objects.

For example, the cyclic notation of (1 6 2 4 3 5) is (1)(2 6 5 3)(4). The number 1 and 4 stay in their “correct” places, so 1 is stored in the object sequence which is stored in notation[0], and 4 is stored in the object sequence which is stored in notation[3]. The “cycle” (2 6 5 3) has its smallest number 2, so those 4 numbers are stored in the object sequence which is stored in notation[1], one by one.

4. Inverse, inversion table and factoradic number.

The inverse, inversion table and factoradic number of a permutation all are represented by an integer array.

The inverse just inverses the permutation, via the method explained in our lecture.

The inversion table, for each number in the permutation, counts the number of numbers which are before this number and also bigger than this number.

The factoradic number can be easily obtained from the inversion table, just remember to add the "!" at the end. I define a method factorial() in this class which returns the factorial of an integer. This method is used to compute the order of the permutation on that numbers. For example, (4 3 1 2) is the 17th permutation on 4 numbers.

5. Visualization of inverse, inversion table and fatoradic number.

Because all of them are represented by an array, so I use the same canvas, NumberStringCanvas, to visualize them. This class defines titles for each object, location of title and location of numbers.

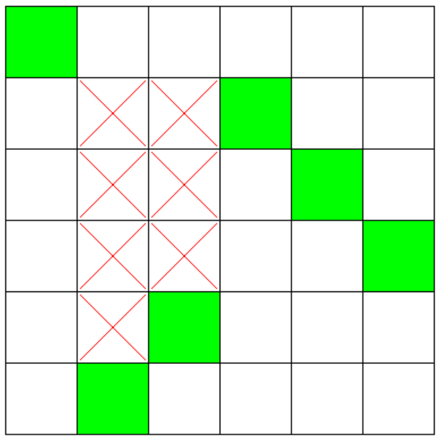
6. Square gird

Square grid is represented by a 2-D array. Each entry represents a small square in the grid. If this small square needs to be colored, it should equal to 1. If it needs to have a cross, it equals to -1. Otherwise, it equals to 0 which indicates it is an empty square. We first color all squares which need to be colored. Then empty all squares which are just below those colored squares. Then we check each row, empty any square which is behind the colored square in this row.

The canvas SquareGridCanvas is used to visualize the square grid. It is simple, just defining the location of the grid, the size of each small square and the title.

Following is an example of a square grid.

Square Grid of this permutation:



7. Permutation graph

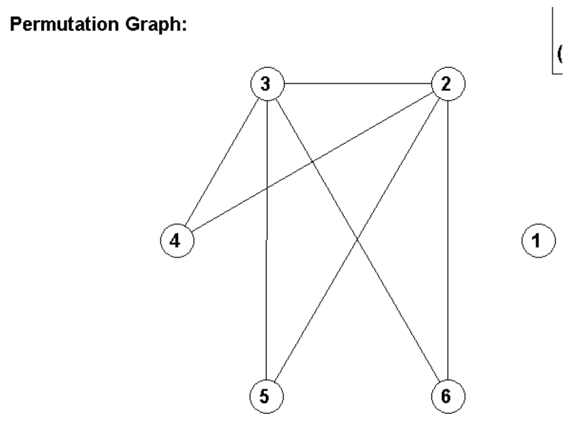
A helper class, PermutationEdge, is used. A permutationEdge objec holds the 2 numbers that this edge connects.

Permutation graph is represented by an array of edges. For each number in the permutation, it has a connection with the numbers which are ahead of it and also bigger than it. Check each number and add edges to the array. For example, for the permutation (3 4 2 1), there are 5 edges totally: (1 3), (1 4), (1 2), (2 3) and (2 4).

The class PermutationGraphCanvas visualizes the graph. It has an inner helper class, Circle, which represents a number on the canvas. A circle object defines the size of the circle and its location.

The PermutationGraphCanvas class is a little more complex than the canvases we talk before. Besides title and the location of title, it also defines the locations of all circles (numbers). Those circles are evenly placed on a big circle whose center is the center of that part.

Following is an example of permutation graph.



8. Meander

I write the number from 1 to the largest then add "path" to form the meander. The path which is from an odd number to an even number is drawn on the top side, and all paths which are from even number to odd number are on the other side.

A permutation which could generate a meander must have even number of numbers, which can be easily checked by checking the size of the array. The permutation also needs to alternate between even numbers and odd numbers. So, I re-write the cyclic permutation to make it starting from number 1, as I mentioned in Section 2. After re-writing, all even numbers have odd indices (our array starts from 0), and all odd numbers have even indices.

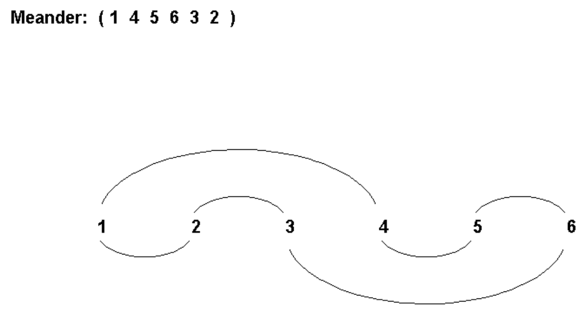
If a permutation passes these 2 checking, we are going to check if there is a crossing. A helper class, MeanderCurve, is used. A MeanderCurve represents a segment of the meander, from the "source" number to its "destination" number. For example, the meander generated by the permutation (1 2 3 4) has 4 MeanderCurve objects, 1-->2, 2-->3, 3-->4, and 4-->1.

We add curves one by one. When we add one curve, we will check if this curve has a crossing with any existing curves. If so, this permutation cannot be used to generate a meander.

A meander is represented by an array of MeanderCurve.

MeanderCanvas is used to visualize a meander. It defines the locations for title, numbers and curves. Each curve is an arc, which needs several parameters, such as start coordinates, start angle and angle, and the width and height. The start coordinates, width and height are decided in class MeanderCanvas, the start angle and angle are defined in class MeanderCurve, because the arc which is from an odd number to an even number is drawn on the top side, and all arcs which are from even number to odd number are on the other side.

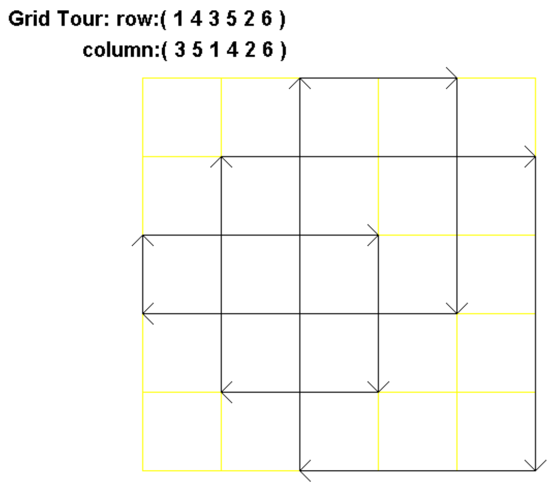
Following is an example of a meander:



9. Grid Tour

Only the permutation which has even number of numbers could generate a grid tour. Besides this requirement, we need guarantee that all turns are right-turn.

To check this, I firstly re-write the permutation to make it starts from number 1. For any grid tour, if we start from the row 1, we must go a row whose number is bigger than 1, then go back to a row whose number is smaller than the second row. Please see the following example.



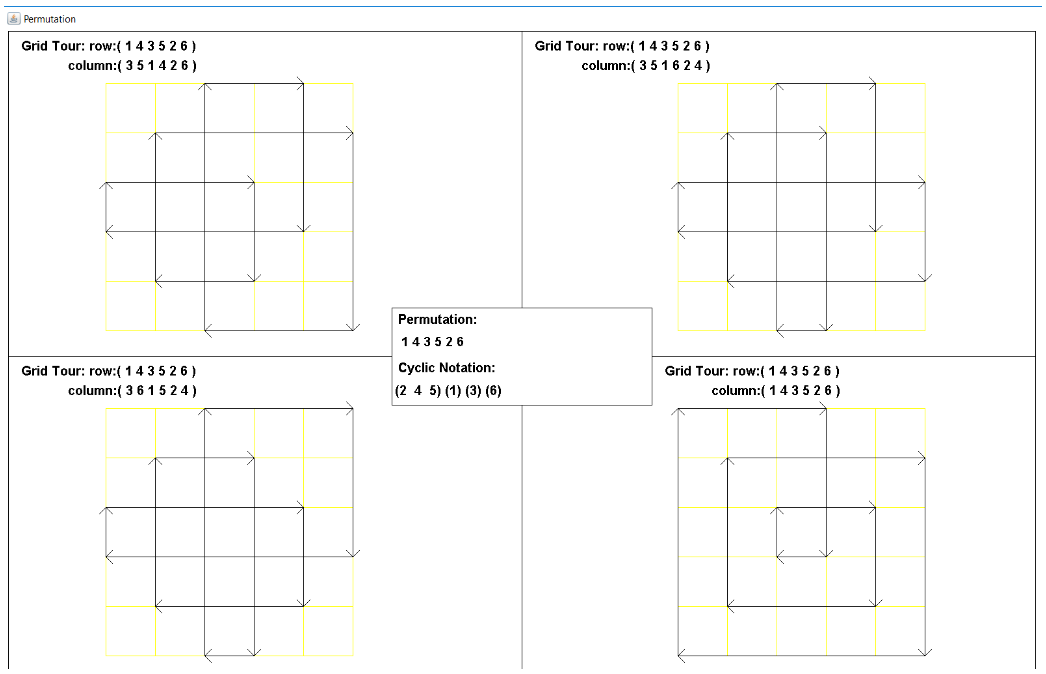
So, we keep going from a smaller row to a bigger row, then a smaller row, then a bigger row, on and on, until we are done. This rule also works for columns, if we set the smaller column of row 1 as the starting column. For example, in the grid tour shown above, the starting point is the point at row 1, column 3.

Because we re-write the permutation, now it starts from number 1. Thus, all numbers which have even indices should be smaller than the next number (array index starts from 0), and the numbers that have odd indices are greater than the next number. And also, the last number must be greater than the first one.

If a permutation meets this rule, it can generate a grid tour. Now we need to find a corresponding column permutation for it. I choose to randomly generate a permutation of same size, and check if it satisfies that rule. If so, this pair can be used to generate a grid tour.

Since the permutation we used to check and generate column is the permutation which has been re-wrote, to make the column permutation matches with the original permutation, we need to write the generated column permutation back to pair the original row permutation.

A row permutation could have several or a lot of matching column permutation. See the example below. For the row permutation (1 4 3 5 2 6), each time the program generates a different column permutation, and all of the pairs define a grid tour successfully.



Grid Tour: row:(1 4 3 5 2 6)
column:(3 5 1 4 2 6)

Grid Tour: row:(1 4 3 5 2 6)
column:(3 5 1 6 2 4)

Permutation:
1 4 3 5 2 6
Cyclic Notation:
(2 4 5) (1) (3) (6)

Grid Tour: row:(1 4 3 5 2 6)
column:(3 6 1 5 2 4)

Grid Tour: row:(1 4 3 5 2 6)
column:(1 4 3 5 2 6)

Each segment of the path (the line and the arrow) on the tour path is represented by an object of GridTourPath. It defines the starting point and end point. For example, the grid tour, which is represented by the row (1 4 3 5 2 6) column (1 4 3 5 2 6), has 12 paths. From point (1 1) to (1 4), then from (1 4) to (4 4), the from (4 4) to (4 3), … . Those paths are kept in a GridTourPath array. The grid tour is represented by this path array.

GridTourCanvas class defines the locations of title and grid, and the small square size. The GridTourPath object has some attributes to help locating itself.

10.Panel class

This class defines all panels, buttons, listeners and methods that we use to communicate with user, draw required graphs, etc. The user can input a permutation or randomly generate one. The user can choose any type to shown in any part. If the action is not permitted, a warning message is given.

11.Independent Statement

This project is done independently. I just learn from our lectures and JAVA tutorial. I have talked with Mr. Ningxuan Wen and Miss. Siva Likitha Valluru about the project requirements and progress. Thanks to Dr. Hedetniemi. She gives me some help via our emails.