APP1: Be Amazed
Report

Members:

- **DO Son Tung**
- **SOSNIN Andrey**
- **MAJDEDDINE ALHAFEZ**
- **Eman AL-SHAOUR**
- **Antoine COLOMBIER**

Maze Generation

Problem:

- Input: 2 dimensions p,q as height and width.
- Output: A tree which represents different steps of subdivision. Leaves correspond to elementary 1x1 cells of the maze. Two nodes are siblings iff each of them has a cell (an entry or an exit point, depending on the point of view) from which we can move (in the maze) to the other one in one move. Each node contains a reference to its parent and offspring, if any. Each non-leaf node contains the position of the door which connects its offspring (see below). Each leaf contains its coordinate. There is no imposed order on the nodes at this stage.

Solution:

```
subdivide(node,p,q){            #p,q are the dimension of the node
        while (p!=1 and q!=1)
                if(p<q)
                        Generate a wall between 1 and q-1:
                                Create 2 child nodes with sizes p x cut and p x (q - cut)
                        On this wall generate a door between 0 and p-1:
                                Get the door coordinates {a, b} (procedure described further)
                                and add them to node
                        Apply subdivide on 2 children.
                else
                        Do the same as if statement but reverse p and q
}
```

We number individual cells on the grid top to bottom, left to right from 0 to p*q - 1 where p and q are vertical and horizontal dimensions, e.g. for n = p*q = 16:

```
_____________   ____________   ____________   ____________
|            |  |     |      | |     |      | |  |  |      |
| 0  4   8  12 | | 0  4 | 8  12 | | 0  4 | 8  12 |  | 0 |4 | 8  12|
|            |  |     |      | |     |      | |  |  |   |  |
```

```
| 1  5   9  13 |   | 1  5 | 9  13 |   | 1  5 | 9  13 |    | 1  5  |  9  |13|
|              |   |      |         |   |__   |___     |    |__    |___|    |
| 2  6  10  14 |   | 2  6 |10  14 |   | 2  6 |10  14 |    | 2 | 6 |10 |14 |
|              |   |      |         |   |      |         |    |   |   |    |    |
| 3  7  11  15 |   | 3  7  11  15 |   | 3  7  11  15  |    | 3  7  11  15  |
|_________ _|    |___________|   |___________|    |____________|
      A                 B                 C                 D
```

Every time a split is made we save the position of the gateway represented by a pair of cells adjacent to it (e.g. {7,11}, {5,6}, {13,14}...).

One can obtain cartesian coordinates of a cell by performing a euclidean division by p. It is also easy to express the condition that exit E should be on the outer wall: E \\ p == 0 || E \\ p == q-1 || E%p == 0 || E%p == p-1. We can also note that the split is horizontal only if the gateway has the form {g, g+1} (if the height of the grid is at least 2, the assumption that we will take from now on).

Solving the maze

Input:
The coordinate of the exit cell
The coordinate of the starting cell

General outline(precise procedures will be detailed later):

Descend from root, ordering necessary branches of the tree as you go. The process is recursive: we start by considering the starting and the exit cells; then, at each meaningful subdivide we order the sons and we apply anew the method on the left, then on the right son, substituting one of the points of interest (i.e. start or exit) by the position of the gate of that subdivide. (A subdivide is ‘meaningful’ when the points of interest are in different sons)

```
find_path(node, start, exit)
    if node is leaf:
        add node to the path
        if node == exit:
            return
    if start == exit:                     # non-essential check
        add start to the path
        return
    else:
```

```
        if start and exit are both in one of the sons of node:
                find_path(node, start, exit)     # the other son can be ignored
        else:
                order sons of node so that exit is in the rightmost one
                let {a,b} be the gateway of node such that b is in the right son of node
                find_path(left_son, start, a)
                find_path(right_son, b, exit)
```

determining position of a cell wrt to a gateway:

```
    let {a,b} be the gateway, such that b > a;
    if b == a + 1:    # horizontal gate
        if cell % p <= a % p:                # p is vertical dimension
            cell is on the side of x
        else:
            cell is on the side of y
    else:    # vertical gate
        if cell <= a:
            cell is on the side of x
        else:
            cell is on the side of y
```

(the comparisons work only because cell is always in one of the rooms separated by {a,b})

Notes (not to include in the report):

we consider 1x1 rather than 1xn elementary cell

we do not consider that nodes are ordered wrt to the exit cell after generation because I don not see how to do it with random exit without additional operations;

Numbering from 0 to n-1 rather than using (x,y) coordinates barely serves any purpose, but it’s significantly easier to draw.

Complexity Analysis

Generation:

The generation algorithm is a “divider and conquer” method. The algorithm method is called twice; one for every branches of the node:

Best case: The tree is balanced; every sub-areas located on same parent node own the same dimensions.
$T(n) = 2T(\frac{n}{2}) + C$, therefore $T(n) = O(n)$ by Master theorem
Worst case: Random subdivision leads to the tree being unbalanced; one of the sub-areas will have a smallest area, while the other will have the maximum, and this on every levels. However, the asymptotic complexity remains the same because the tree with n leaves has at most 2n leaves.

Note: as well as during the solving part of the problem (as we will see), this algorithm creates a stack of recursive calls equal (up to an additive constant) to the height of the tree.

Path finding:

In the worst case, we have to go through every cell (for example: if the maze have the spiral shape), so the total solution time must be at least n. On the other hand, the proposed solution pass through each node at most once. So we take at most 2n nodes therefore the complexity is $O(n)$