Computer Architectures
Exam of 16.09.2024 – part II

Question 1

Write the `kruskal` subroutine in ARM assembly to generate a maze according to the Kruskal's algorithm. The maze will be stored in the following data structures:

- a matrix of bytes `maze`, where connected cells have the same value;
- an array of bytes `horizontal_walls`, which indicates the presence of a wall between each pair of consecutive cells in the same row;
- an array of bytes `vertical_walls`, which indicates the presence of a wall between each pair of consecutive cells in the same column.

Each data structure has NUM_ROW * NUM_COL elements.

The *k*-th element of `horizontal_walls` can have one of the following values:

- 0: the *k*-th cell (in row-major order) in the maze is connected with the cell at its right
- 1: there is a wall between the the *k*-th cell and the one at its right
- 2: the *k*-th cell is along the right border of the matrix (i.e., in the last column).

Similarly, the *k*-th element of `vertical_walls` can have one of the following values:

- 0: the *k*-th cell (in row-major order) in the maze is connected with the cell at its bottom
- 1: there is a wall between the the *k*-th cell and the one at its bottom
- 2: the *k*-th cell is along the bottom border of the matrix (i.e., in the last row).

Initially, there is a wall between every pair of cells, thus each cell has a different value because no one is connected. An example is given with NUM_ROW = 3 and NUM_COL = 4.

`maze`

0	1	2	3
4	5	6	7
8	9	10	11

`horizontal_walls` 1, 1, 1, 2, 1, 1, 1, 2, 1, 1, 1, 2

`vertical_walls` 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2

The `kruskal` subroutine receives the following parameters (in the order indicated):

1) address of the matrix `maze`
2) address of the array `horizontal_walls`
3) address of the array `vertical_walls`
4) NUM_ROW
5) NUM_COL
6) an increment `y`
7) an offset `x`

The `kruskal` subroutine implements the following algorithm:

```
do
    x = x + y
    if x < NUM_ROW * NUM_COL:
        if horizontal_walls[x] == 1:
            if maze[x] != maze[x + 1]:
                horizontal_walls[x] = 0          ; remove the wall
                m = min(maze[x], maze[x + 1])
                n = max(maze[x], maze[x + 1])
                change all elements of maze equal to n to m
        else if horizontal_walls[x] == 0:
            y = y + 1
    else:
        x = x - NUM_ROW * NUM_COL
```

Computer Architectures
Exam of 16.09.2024 – part II

```
if x < NUM_ROW * NUM_COL:        ; this part is very similar to the previous one
    if vertical_walls[x] == 1:
        if maze[x] != maze[x + NUM_COL]:
            vertical_walls[x] = 0          ; remove the wall
            m = min(maze[x], maze[x + NUM_COL])
            n = max(maze[x], maze[x + NUM_COL])
            change all elements of maze equal to n to m
    else if vertical_walls[x] == 0:
        y = y + 1
```

while there are elements in `maze` different from zero

Example of execution with a 4x3 matrix, `x` = 2 and `y` = 4.

Iteration 1

0	1	2	3
4	5	6	6
8	9	10	11

`x` = 2 + 4 = 6

`horizontal_walls` 1, 1, 1, 2, 1, 1, 0, 2, 1, 1, 1, 2

`vertical_walls` 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2

Iteration 2

0	1	2	3
4	5	6	6
8	9	10	10

`x` = 6 + 4 = 10

`horizontal_walls` 1, 1, 1, 2, 1, 1, 0, 2, 1, 1, 0, 2

`vertical_walls` 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2

Iteration 3

0	1	2	3
4	5	2	2
8	9	10	10

`x` = 10 + 4 = 14 14 >= 12 → x = 14 – 12 = 2

`horizontal_walls` 1, 1, 1, 2, 1, 1, 0, 2, 1, 1, 0, 2

`vertical_walls` 1, 1, 0, 1, 1, 1, 1, 1, 2, 2, 2, 2

Iteration 4

0	1	2	3
4	5	2	2
8	9	10	10

`x` = 2 + 4 = 6 → `y` = 4 + 1 = 5

`horizontal_walls` 1, 1, 1, 2, 1, 1, 0, 2, 1, 1, 0, 2

`vertical_walls` 1, 1, 0, 1, 1, 1, 1, 1, 2, 2, 2, 2

Iteration 5

0	1	2	3
4	5	2	2
8	9	10	10

`x` = 6 + 5 = 11

`horizontal_walls` 1, 1, 1, 2, 1, 1, 0, 2, 1, 1, 0, 2

`vertical_walls` 1, 1, 0, 1, 1, 1, 1, 1, 2, 2, 2, 2

Iteration 6

0	1	2	3
4	5	2	2
4	9	10	10

`x` = 11 + 5 = 16 16 >= 12 → x = 16 – 12 = 4

`horizontal_walls` 1, 1, 1, 2, 1, 1, 0, 2, 1, 1, 0, 2

`vertical_walls` 1, 1, 0, 1, 0, 1, 1, 1, 2, 2, 2, 2

Iteration 7

0	1	2	3
4	5	2	2
4	9	9	9

`x` = 4 + 5 = 9

`horizontal_walls` 1, 1, 1, 2, 1, 1, 0, 2, 1, 0, 0, 2

`vertical_walls` 1, 1, 0, 1, 0, 1, 1, 1, 2, 2, 2, 2

Iteration 8

0	1	2	3
4	5	2	2
4	9	9	9

`x` = 9 + 5 = 14 → x = 14 – 12 = 2 → `y` = 5 + 1 = 6

`horizontal_walls` 1, 1, 1, 2, 1, 1, 0, 2, 1, 0, 0, 2

`vertical_walls` 1, 1, 0, 1, 0, 1, 1, 1, 2, 2, 2, 2

Iteration 9

0	1	2	3
4	5	2	2
4	4	4	4

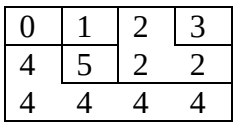
`x` = 2 + 6 = 8

`horizontal_walls` 1, 1, 1, 2, 1, 1, 0, 2, 0, 0, 0, 2

`vertical_walls` 1, 1, 0, 1, 0, 1, 1, 1, 2, 2, 2, 2

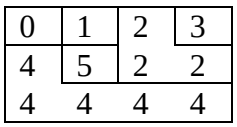
Computer Architectures
Exam of 16.09.2024 – part II

Iteration 10



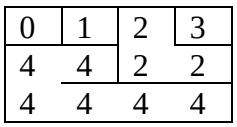
x = 8 + 6 = 14 → x = 14 – 12 = 2 → y = 6 + 1 = 7
horizontal_walls 1, 1, 1, 2, 1, 1, 0, 2, 0, 0, 0, 2
vertical_walls 1, 1, 0, 1, 0, 1, 1, 1, 2, 2, 2, 2

Iteration 11



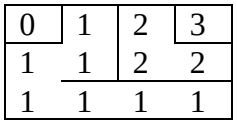
x = 2 + 7 = 9 → y = 7 + 1 = 8
horizontal_walls 1, 1, 1, 2, 1, 1, 0, 2, 0, 0, 0, 2
vertical_walls 1, 1, 0, 1, 0, 1, 1, 1, 2, 2, 2, 2

Iteration 12



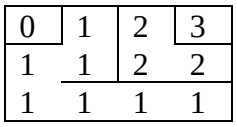
x = 9 + 8 = 17 17 >= 12 → x = 17 – 12 = 5
horizontal_walls 1, 1, 1, 2, 1, 1, 0, 2, 0, 0, 0, 2
vertical_walls 1, 1, 0, 1, 0, 0, 1, 1, 2, 2, 2, 2

Iteration 13



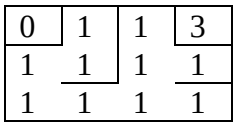
x = 5 + 8 = 13 13 >= 12 → x = 13 – 12 = 1
horizontal_walls 1, 1, 1, 2, 1, 1, 0, 2, 0, 0, 0, 2
vertical_walls 1, 0, 0, 1, 0, 0, 1, 1, 2, 2, 2, 2

Iteration 14



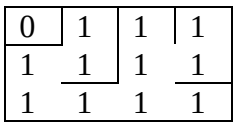
x = 1 + 8 = 9 → y = 8 + 1 = 9
horizontal_walls 1, 1, 1, 2, 1, 1, 0, 2, 0, 0, 0, 2
vertical_walls 1, 0, 0, 1, 0, 0, 1, 1, 2, 2, 2, 2

Iteration 15



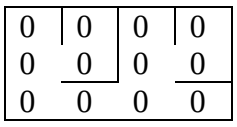
x = 9 + 9 = 18 18 >= 12 → x = 18 – 12 = 6
horizontal_walls 1, 1, 1, 2, 1, 1, 0, 2, 0, 0, 0, 2
vertical_walls 1, 0, 0, 1, 0, 0, 0, 1, 2, 2, 2, 2

Iteration 16



x = 6 + 9 = 15 15 >= 12 → x = 15 – 12 = 3
horizontal_walls 1, 1, 1, 2, 1, 1, 0, 2, 0, 0, 0, 2
vertical_walls 1, 0, 0, 0, 0, 0, 0, 1, 2, 2, 2, 2

Iteration 17



x = 3 + 9 = 12 12 >= 12 → x = 12 – 12 = 0
horizontal_walls 1, 1, 1, 2, 1, 1, 0, 2, 0, 0, 0, 2
vertical_walls 0, 0, 0, 0, 0, 0, 0, 1, 2, 2, 2, 2

Computer Architectures
Exam of 16.09.2024 – part II

Question 2

Add C code that after the pressure of two buttons calls the `kruskal` subroutine written in the previous exercise. In details, you have to define the matrix `maze` and the arrays in C. Then, after the user presses the first button, the program initializes the variable `increment` as follows:

- If the user pressed button INT0, then `increment` = 2
- If the user pressed button Key1, then `increment` = 3
- If the user pressed button Key2, then `increment` = 4

Then, after the user presses the second button, the program initializes the variable `offset` as follows:

- If the user pressed button INT0, then `offset` = 2
- If the user pressed button Key1, then `offset` = 3
- If the user pressed button Key2, then `offset` = 4

After the pressure of the second button, the program calls the `kruskal` subroutine passing the parameters in the right order.