[redacted]

2D Grid

Adjacency, Directional Array and DFS/BFS

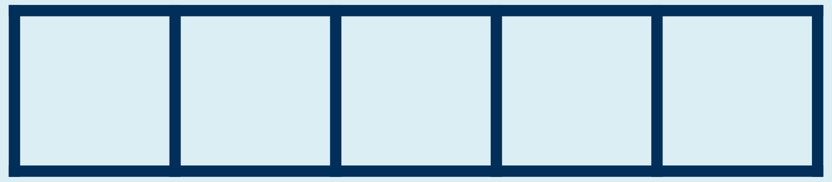
What is 2D grid?

2D grid is simply an array of array. Therefore, it is called 2-Dimensional array or 2D grid.

What is 2D grid?

2D grid is simply an array of array. Therefore, it is called 2-Dimensional array or 2D grid.

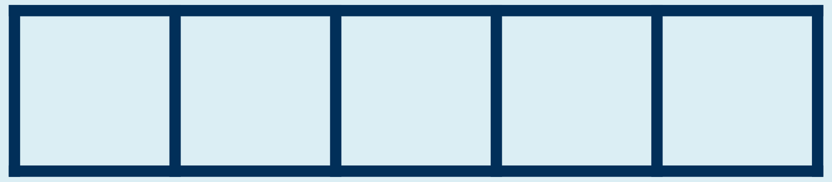
1-Dimensional Array



What is 2D grid?

2D grid is simply an array of array. Therefore, it is called 2-Dimensional array or 2D grid.

1-Dimensional Array

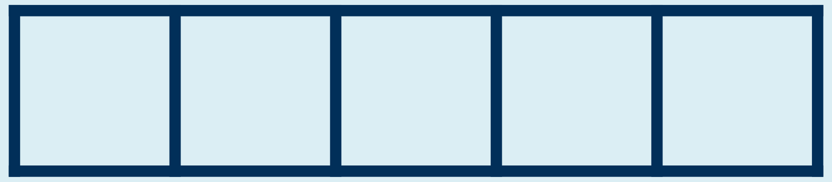


Declaration: `int A[5];`

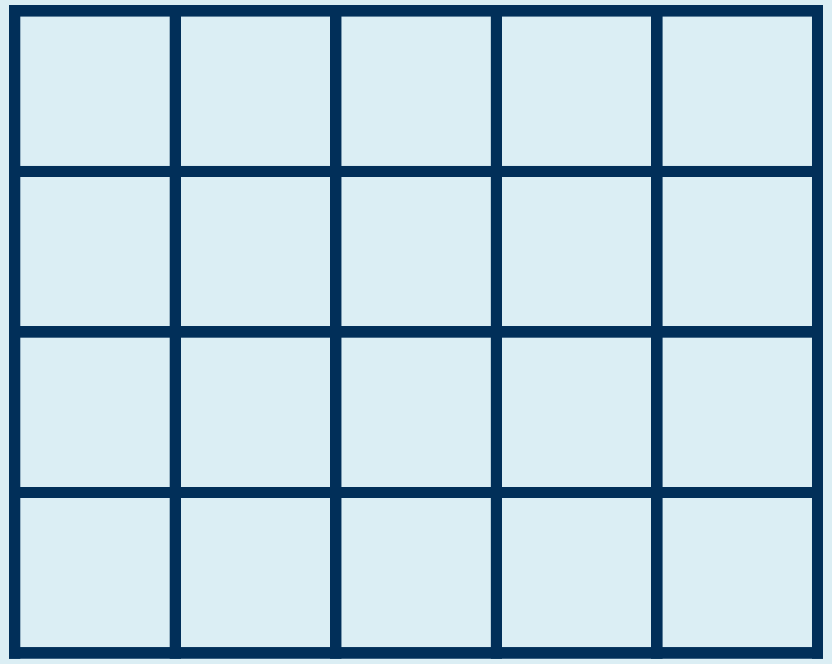
What is 2D grid?

2D grid is simply an array of array. Therefore, it is called 2-Dimensional array or 2D grid.

1-Dimensional Array



2-Dimensional Array

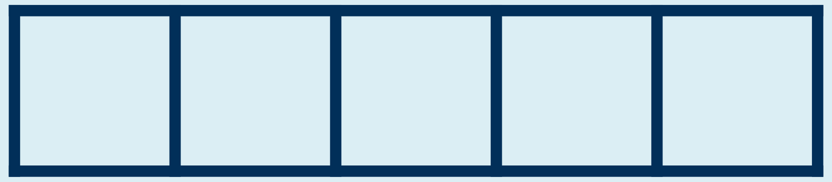


Declaration: `int A[5];`

What is 2D grid?

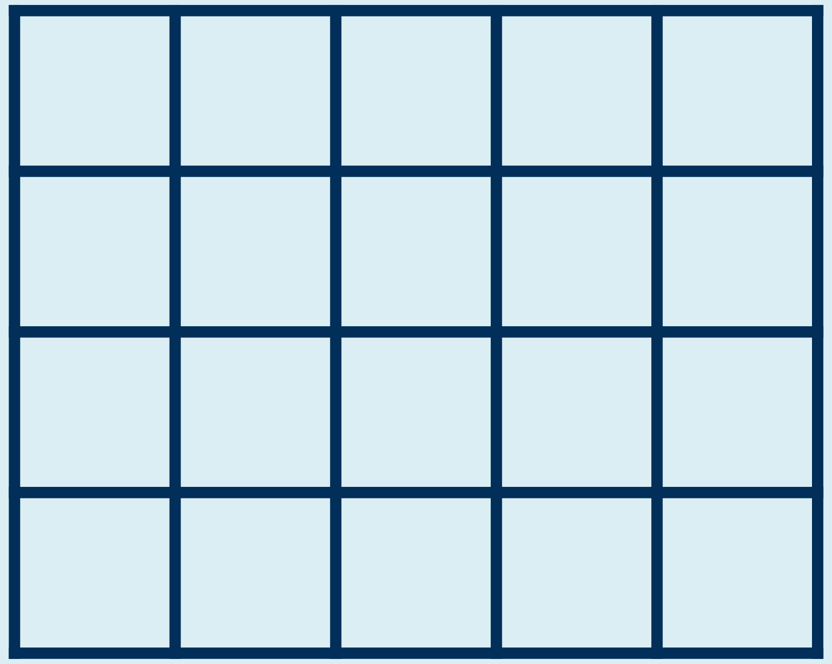
2D grid is simply an array of array. Therefore, it is called 2-Dimensional array or 2D grid.

1-Dimensional Array



Declaration: `int A[5];`

2-Dimensional Array

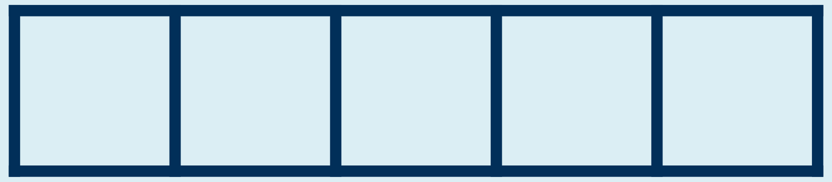


Declaration: `int A[4][5];`

What is 2D grid?

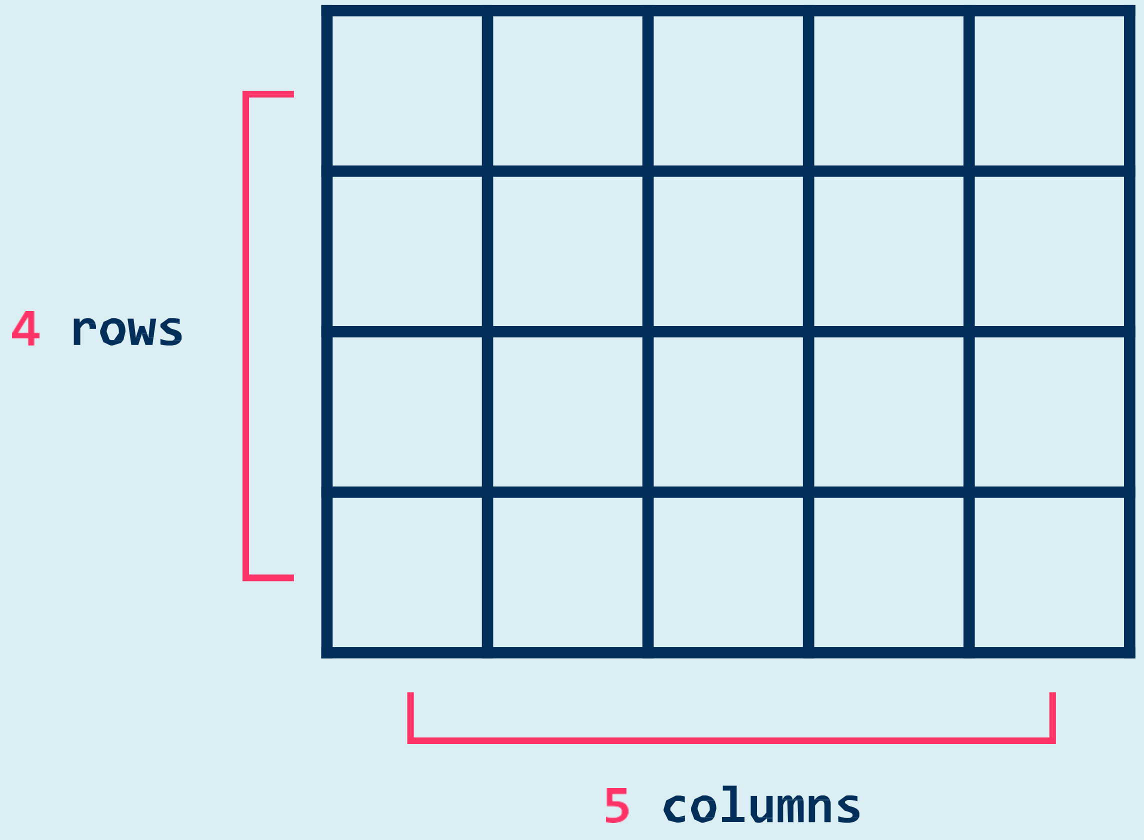
2D grid is simply an array of array. Therefore, it is called 2-Dimensional array or 2D grid.

1-Dimensional Array



Declaration: `int A[5];`

2-Dimensional Array



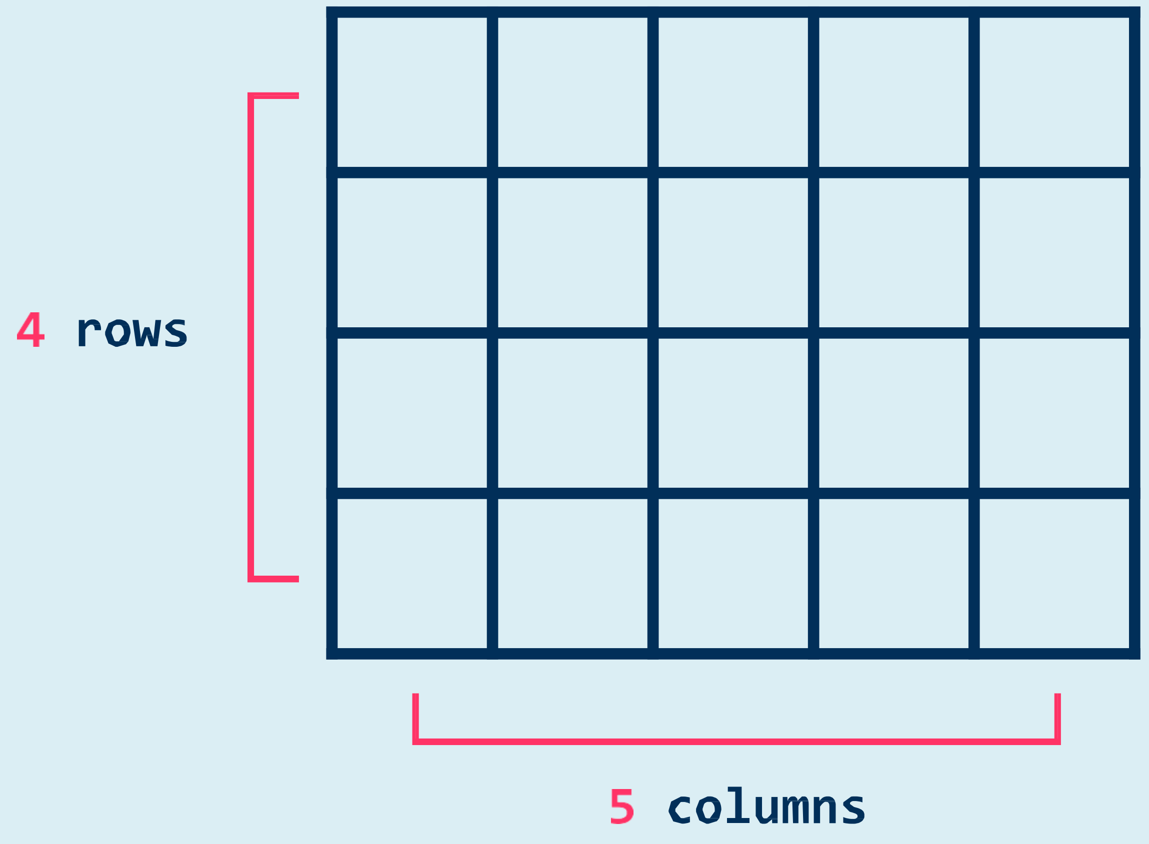
Declaration: `int A[4][5];`

What is 2D grid?

2D grid is simply an array of array. Therefore, it is called 2-Dimensional array or 2D grid.

```
/*
 We created
 an array of size 4
 Where every index contains
 an array of size 5
*/
```

2-Dimensional Array



Declaration: `int A[4][5];`

Working with 2D Grid

We use a linear loop to get input to or print output of an 1D Array.

```
for (int i = 0; i < n; i++){
    // input/output of A[i]
}
```

Working with 2D Grid

We use a linear loop to get input to or print output of an 1D Array.

```
for (int i = 0; i < n; i++){
    // input/output of A[i]
}
```

We use a nested loop to get input to or print output of a 2D Array.

```
for (int i = 0; i < row; i++){
    for (int j = 0; j < col; j++){
        // input/output of A[i][j]
    }
}
```

How to traverse 2D grid using Graph Traversal Algorithms?

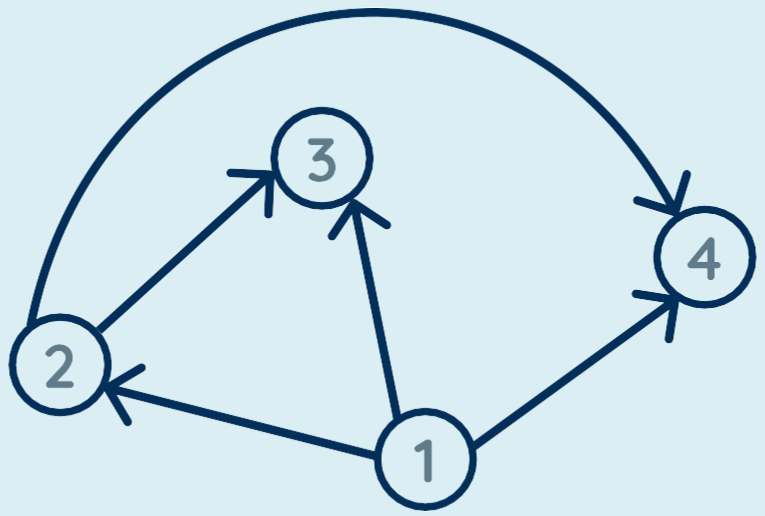
How to traverse 2D grid using Graph Traversal Algorithms?

> We need list of adjacent nodes/grid cells for every cell

How to traverse 2D grid using Graph Traversal Algorithms?

> We need list of adjacent nodes/grid cells for every cell

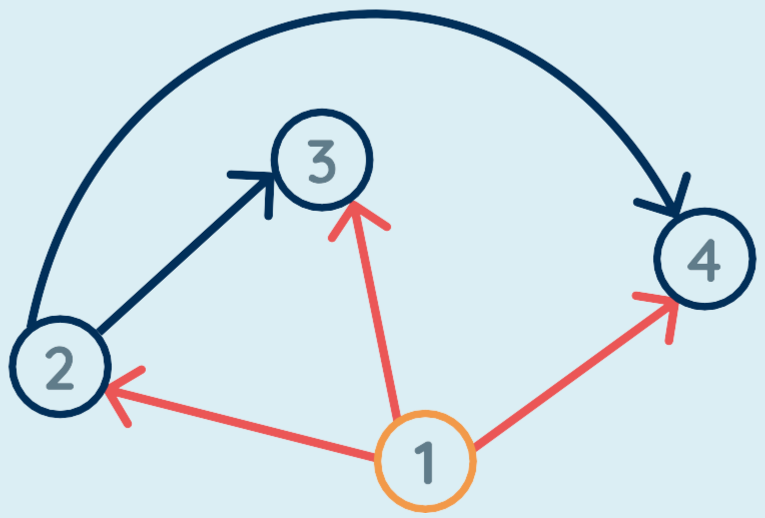
For explicit graphs, we are given edge information,
and we store the adjacent nodes in adjacency list



How to traverse 2D grid using Graph Traversal Algorithms?

> We need list of adjacent nodes/grid cells for every cell

For explicit graphs, we are given edge information,
and we store the adjacent nodes in adjacency list

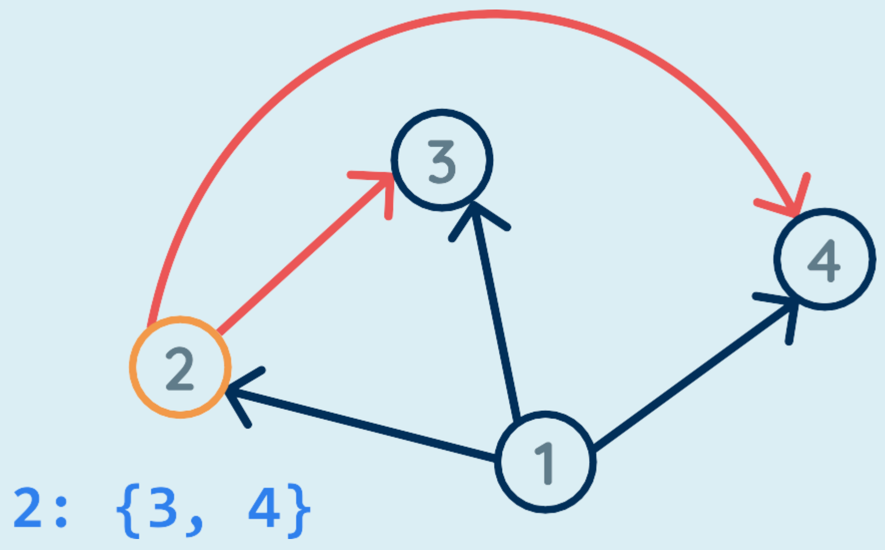


1: {2, 3, 4}

How to traverse 2D grid using Graph Traversal Algorithms?

> We need list of adjacent nodes/grid cells for every cell

For explicit graphs, we are given edge information,
and we store the adjacent nodes in adjacency list

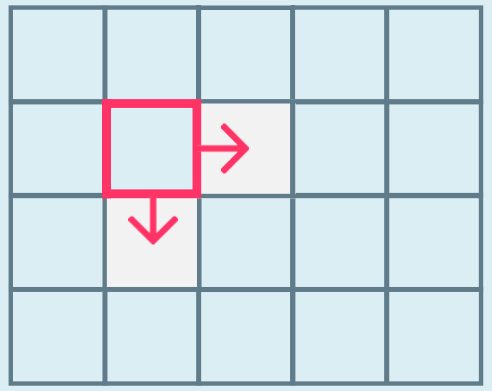


How to traverse 2D grid using Graph Traversal Algorithms?

> We need list of adjacent nodes/grid cells for every cell

In grid, edges are not explicitly defined. Instead, a formula is given on who are the adjacent cell(s) of the current cell.

1. From any cell (x, y), you can move to only right and bottom.



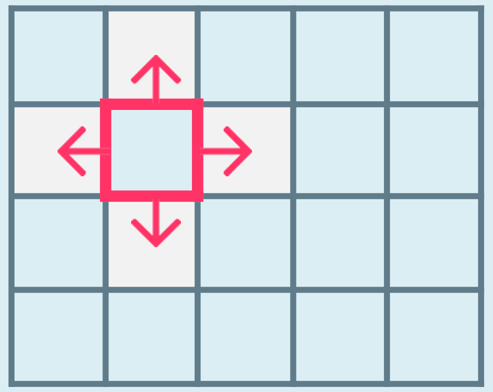
How to traverse 2D grid using Graph Traversal Algorithms?

> We need list of adjacent nodes/grid cells for every cell

In grid, edges are not explicitly defined. Instead, a formula is given on who are the adjacent cell(s) of the current cell.

1. From any cell (x, y), you can move to only right and bottom.

2. From any cell, you can move to any neighboring cell that shares a side.



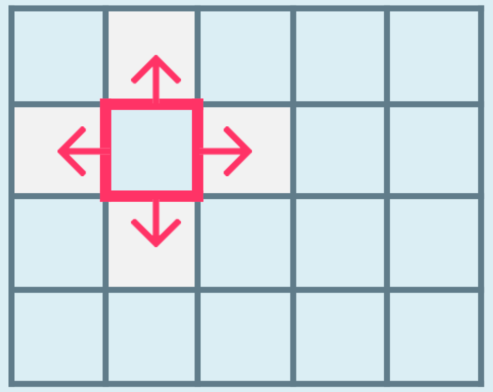
How to traverse 2D grid using Graph Traversal Algorithms?

> We need list of adjacent nodes/grid cells for every cell

In grid, edges are not explicitly defined. Instead, a formula is given on who are the adjacent cell(s) of the current cell.

1. From any cell (x, y), you can move to only right and bottom.

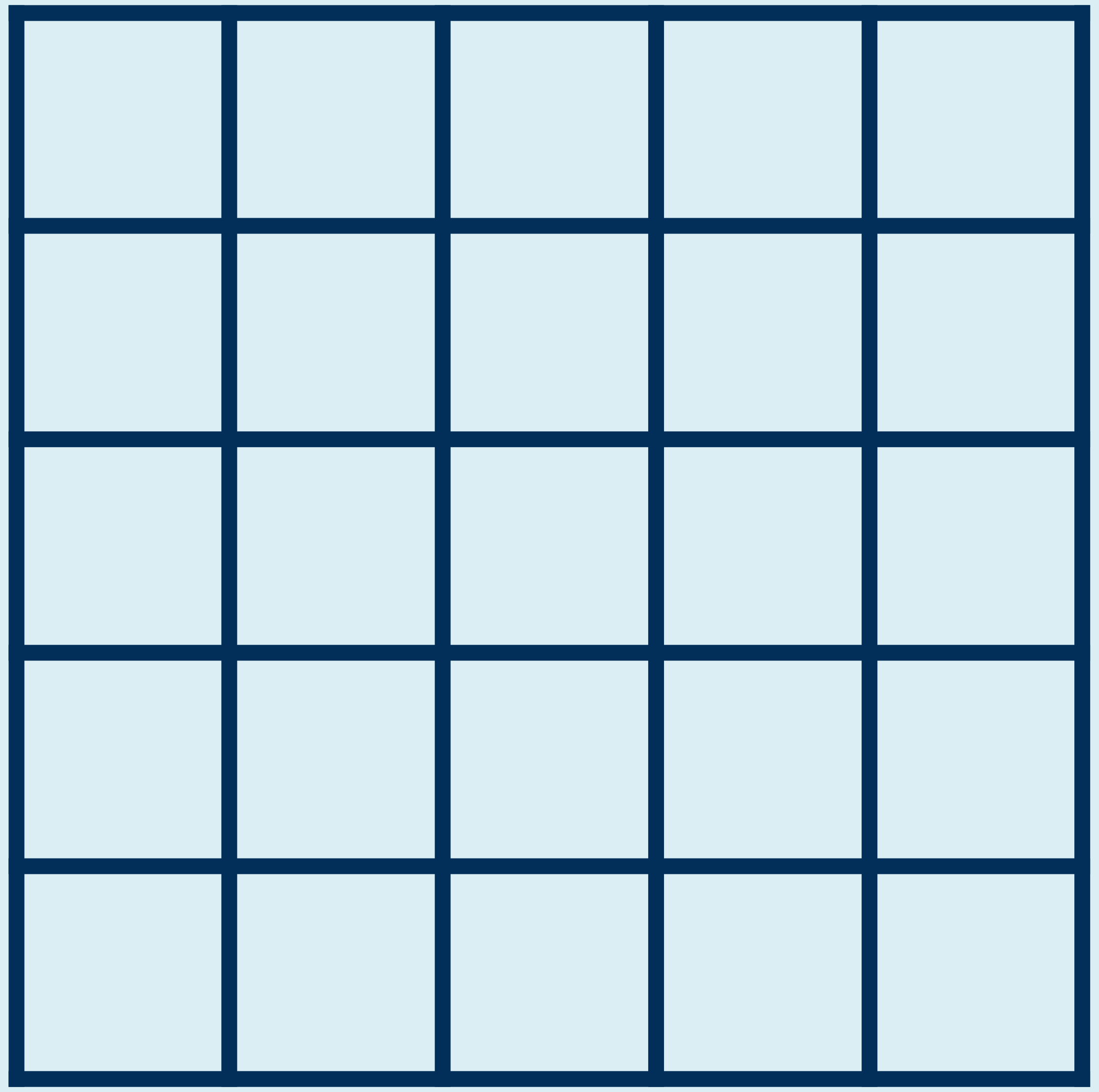
2. From any cell, you can move to any neighboring cell that shares a side.



> So we need to calculate neighbors cell coordinates based on current cell

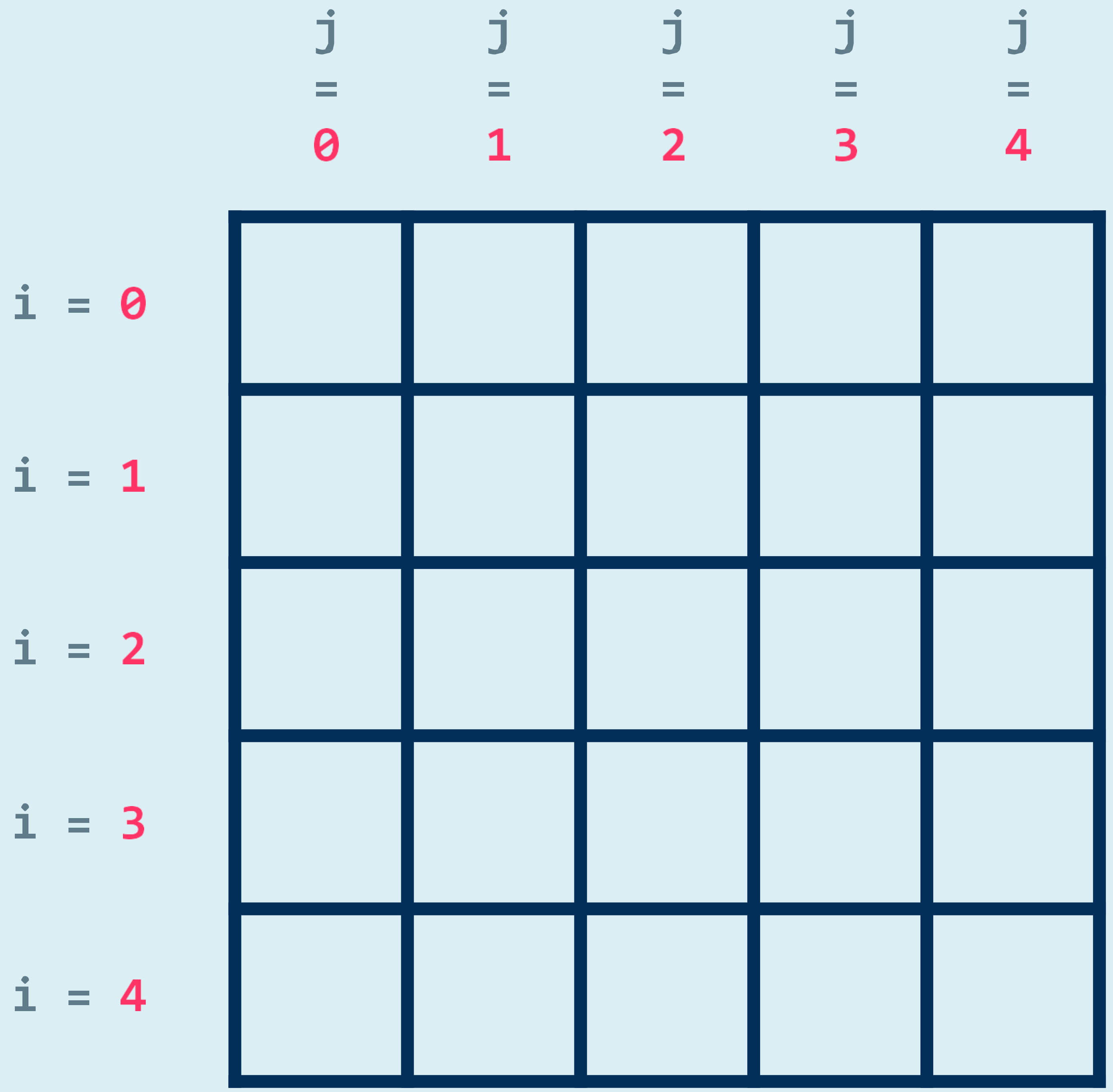
Calculating Neighbor Cells

Calculating Neighbor Cells



`A[5][5]`

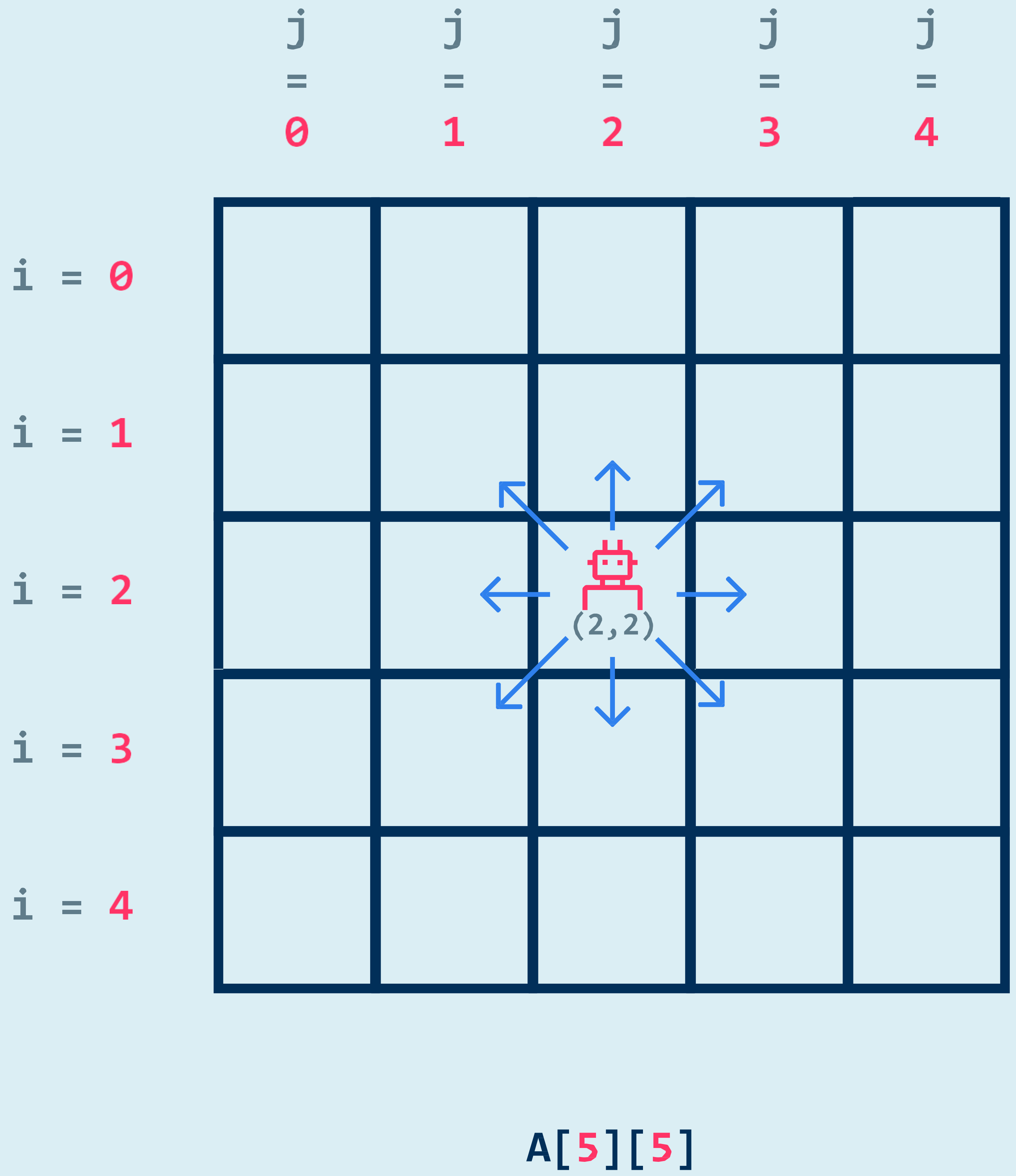
Calculating Neighbor Cells



A[5][5]

```
// identified rows as i
// and columns as j
```

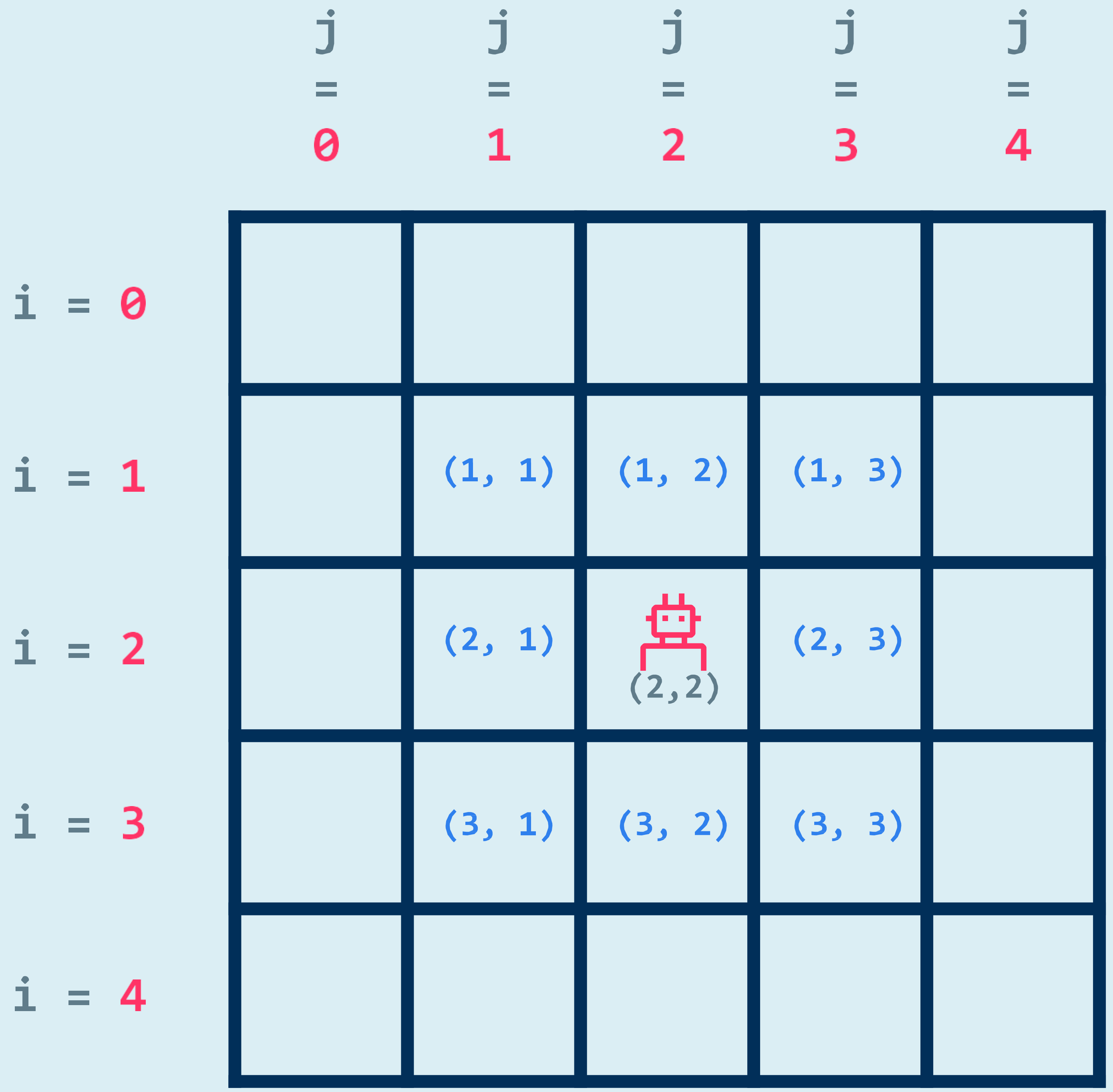
Calculating Neighbor Cells




```
// selected A[2][2] is
// current cell

// assume in the problem,
// it is given that you can
// move to any cell within
// the board that is one step
// away from current cell
// horizontally, vertically
// or diagonally
```

Calculating Neighbor Cells

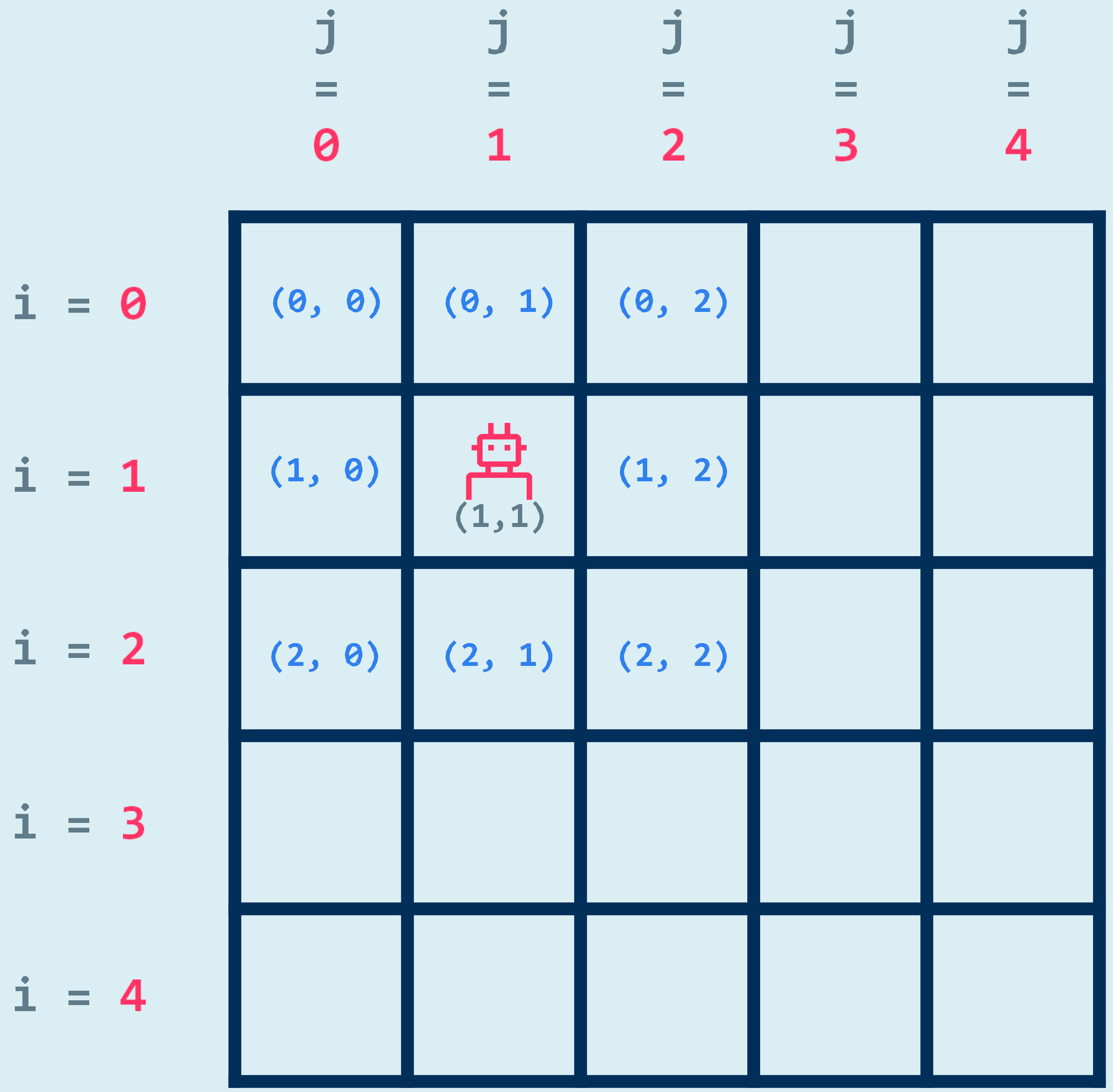


A[5][5]

neighbors of (2, 2)

(2, 2) -> { (1, 1), (1, 2),
(1, 3), (2, 1),
(2, 3), (3, 1),
(3, 2), (3, 3) }

Calculating Neighbor Cells



A[5][5]

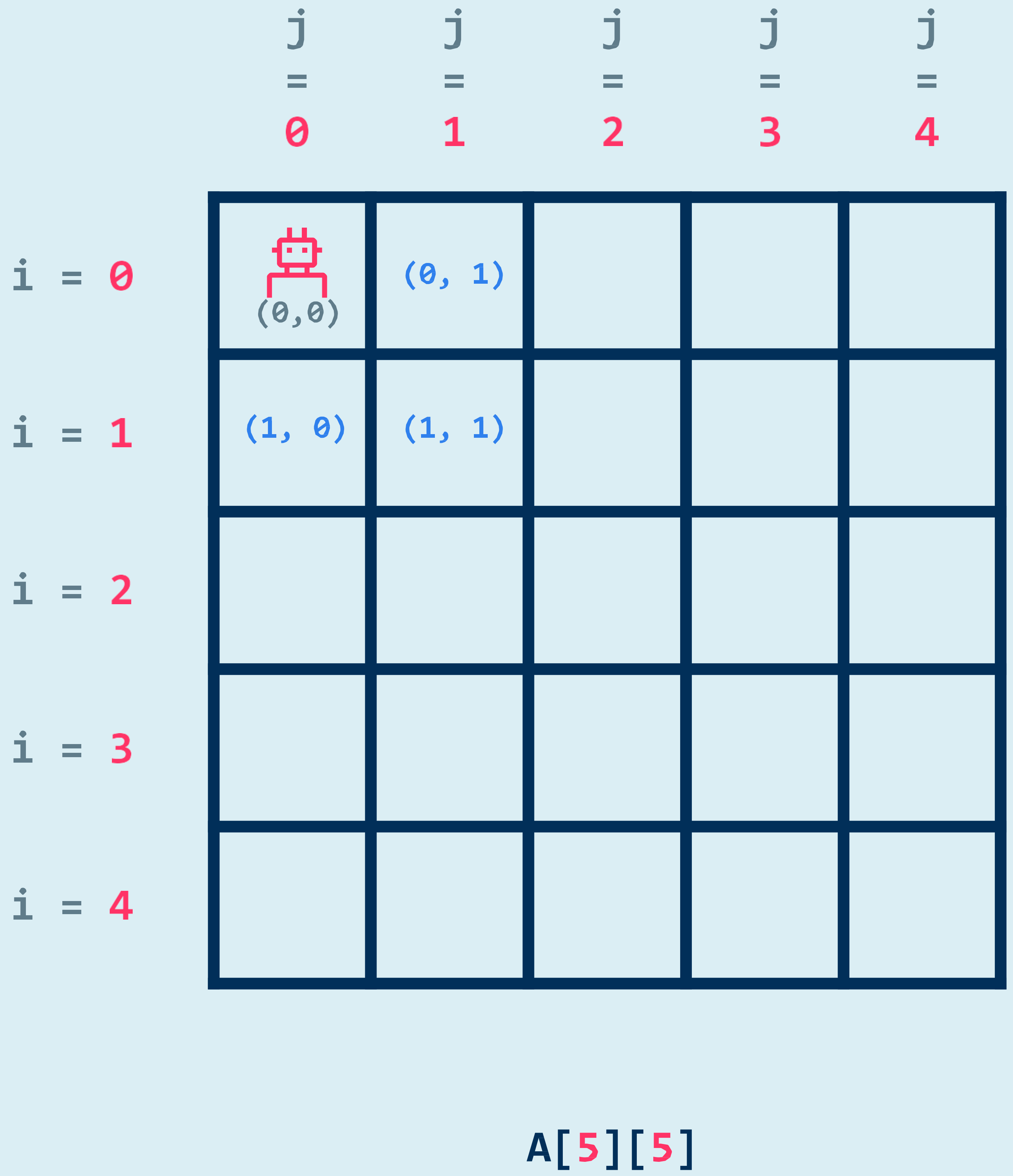
neighbors of (2, 2)

```
(2, 2) -> { (1, 1), (1, 2),
            (1, 3), (2, 1),
            (2, 3), (3, 1),
            (3, 2), (3, 3) }
```

neighbors of (1, 1)

```
(1, 1) -> { (0, 0), (0, 1),
            (0, 2), (1, 0),
            (1, 2), (2, 0),
            (2, 1), (2, 2) }
```

Calculating Neighbor Cells



neighbors of (2, 2)

```
(2, 2) -> { (1, 1), (1, 2),
            (1, 3), (2, 1),
            (2, 3), (3, 1),
            (3, 2), (3, 3) }
```

neighbors of (1, 1)

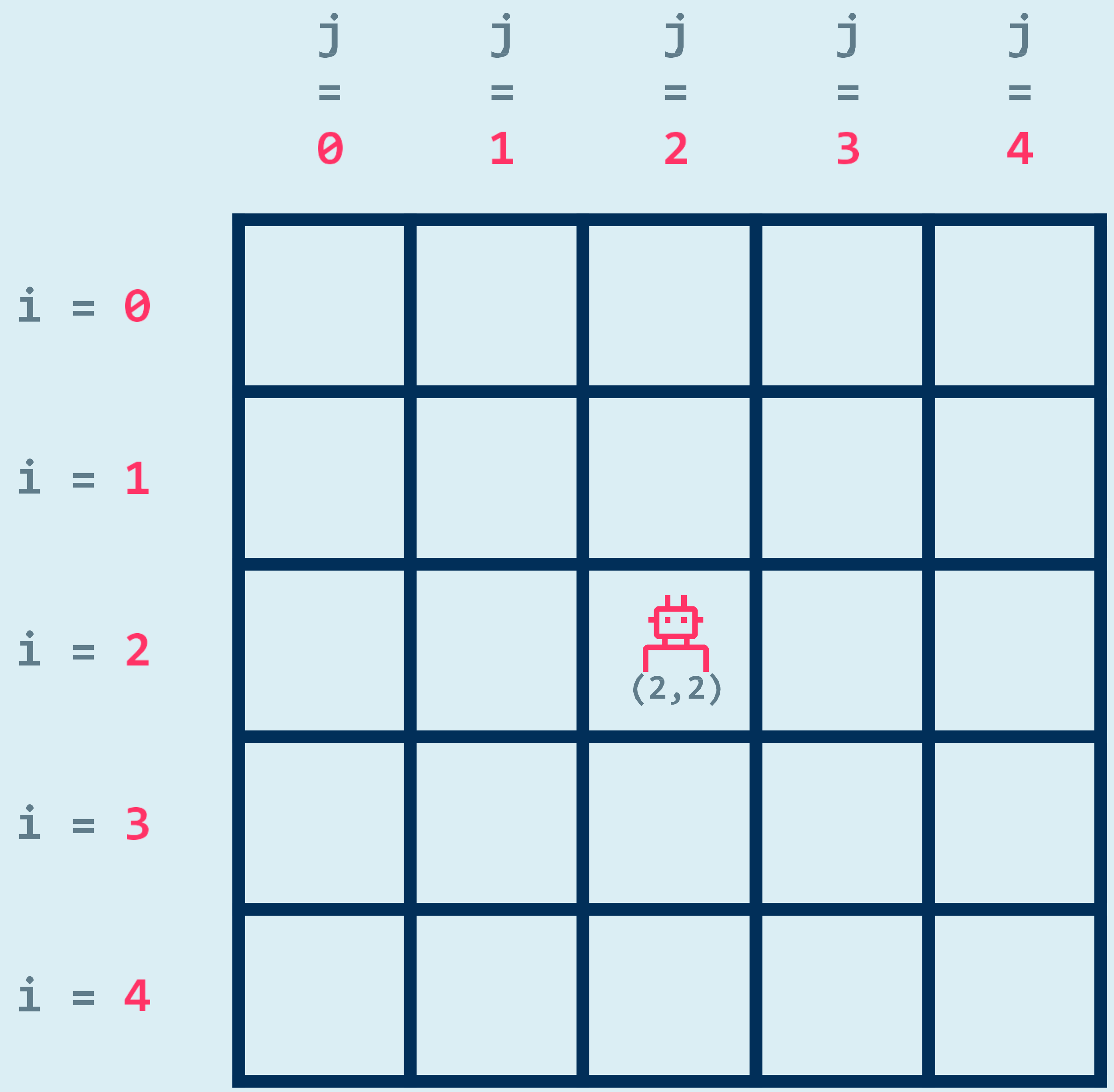
```
(1, 1) -> { (0, 0), (0, 1),
            (0, 2), (1, 0),
            (1, 2), (2, 0),
            (2, 1), (2, 2) }
```

neighbors of (0, 0)

```
(0, 0) -> { (0, 1), (1, 0),
            (1, 1) }
```

Calculating Neighbor Cells

Finding Co-ordinate Pattern

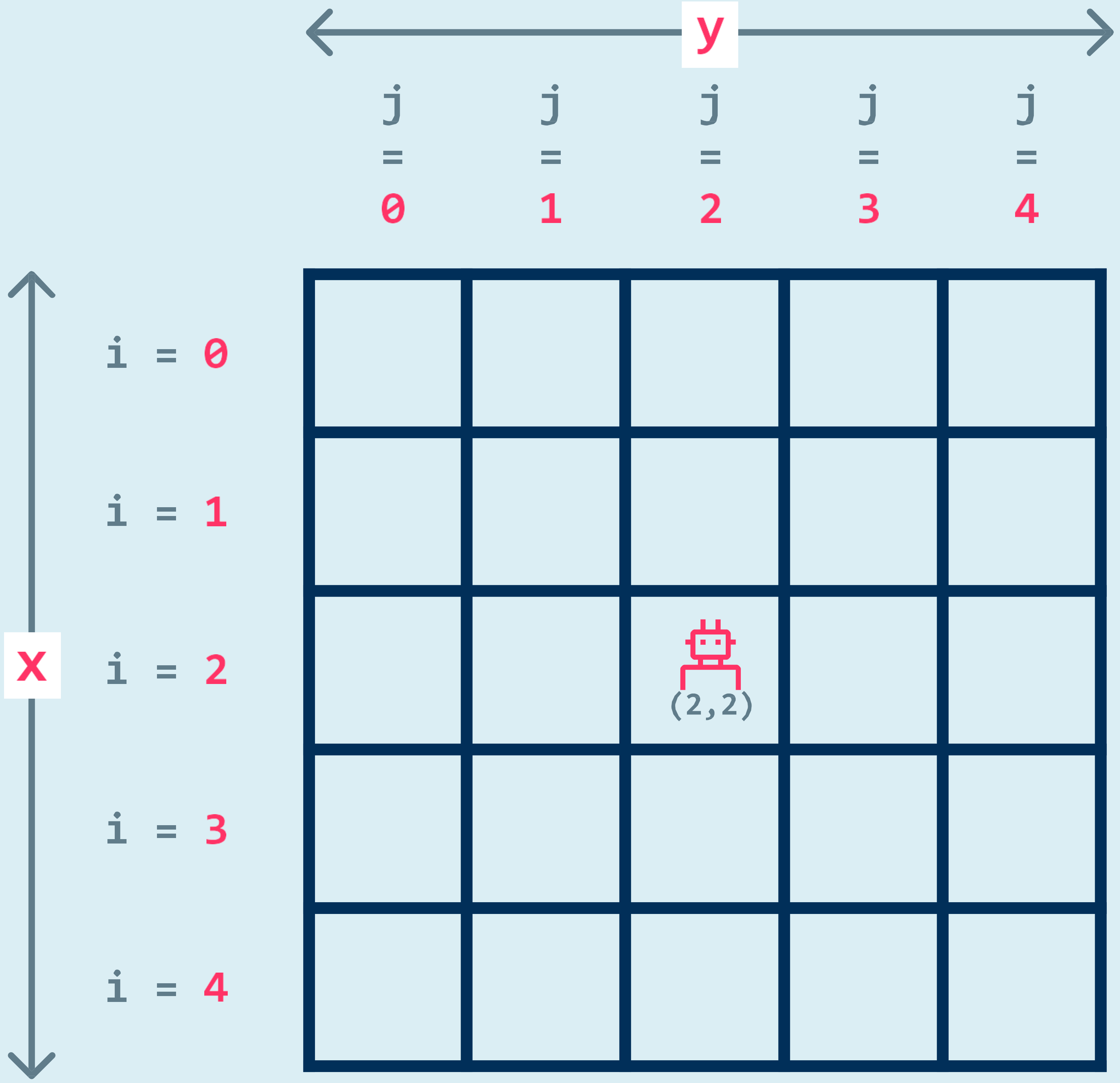


A[5][5]

Calculating Neighbor Cells

Finding Co-ordinate Pattern

- x = row-axis
- y = col-axis



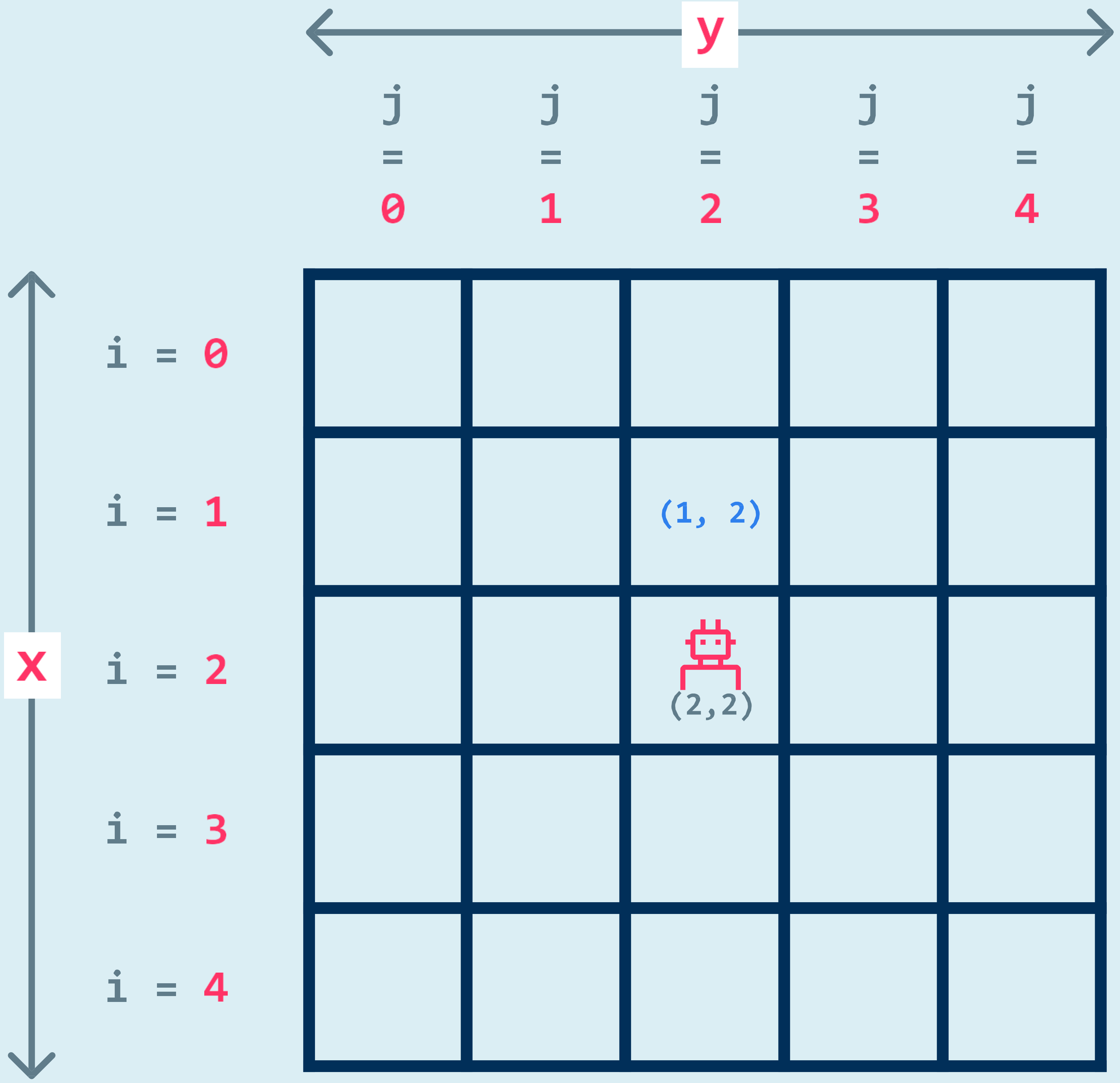
A[5][5]

adjacent cell co-ordinate	change in x (dx)	change in y (dy)

Calculating Neighbor Cells

Finding Co-ordinate Pattern

- x = row-axis
- y = col-axis



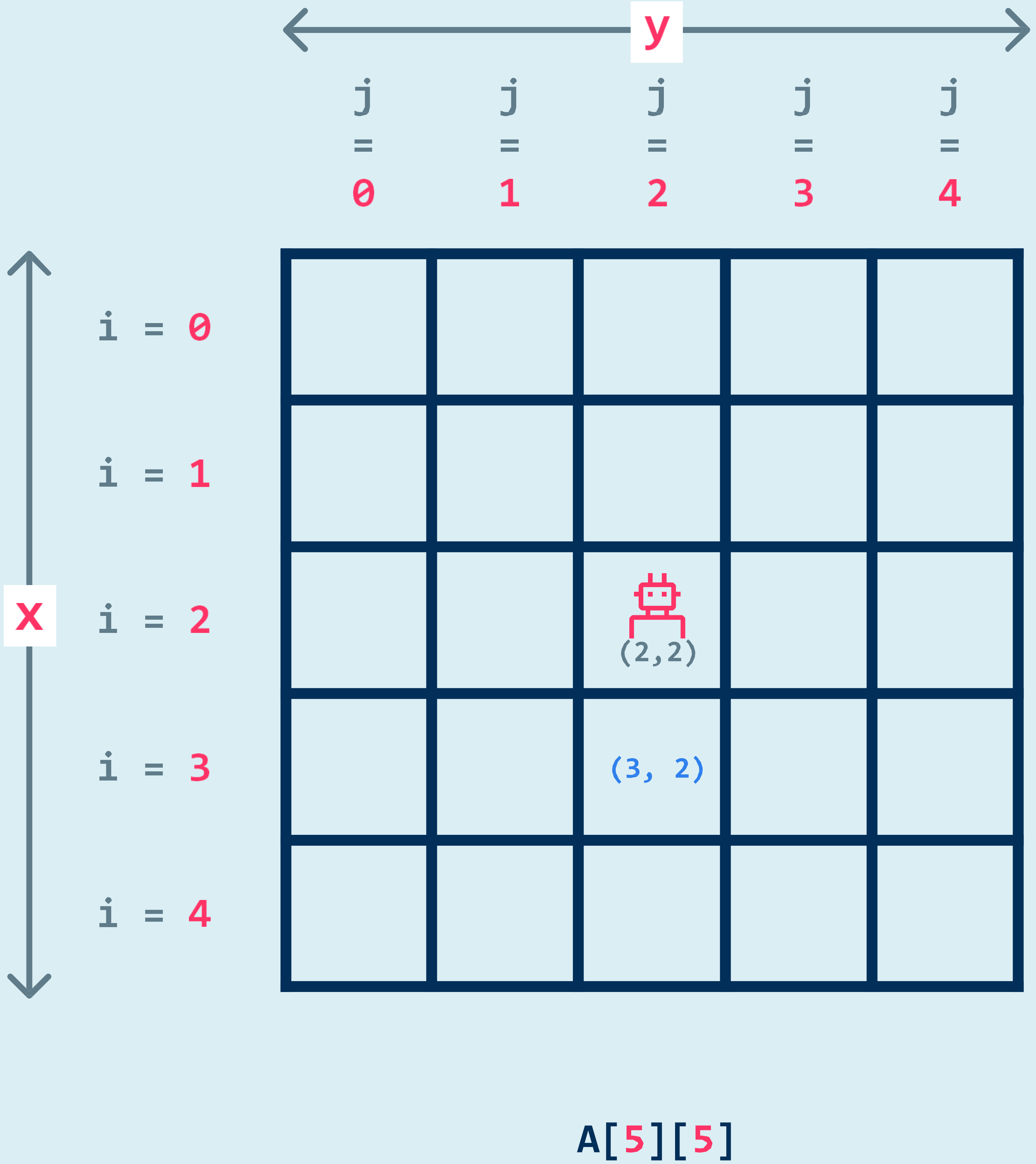
A[5][5]

adjacent cell co-ordinate	change in x (dx)	change in y (dy)
Top (1, 2)	-1	0

Calculating Neighbor Cells

Finding Co-ordinate Pattern

- x = row-axis
- y = col-axis

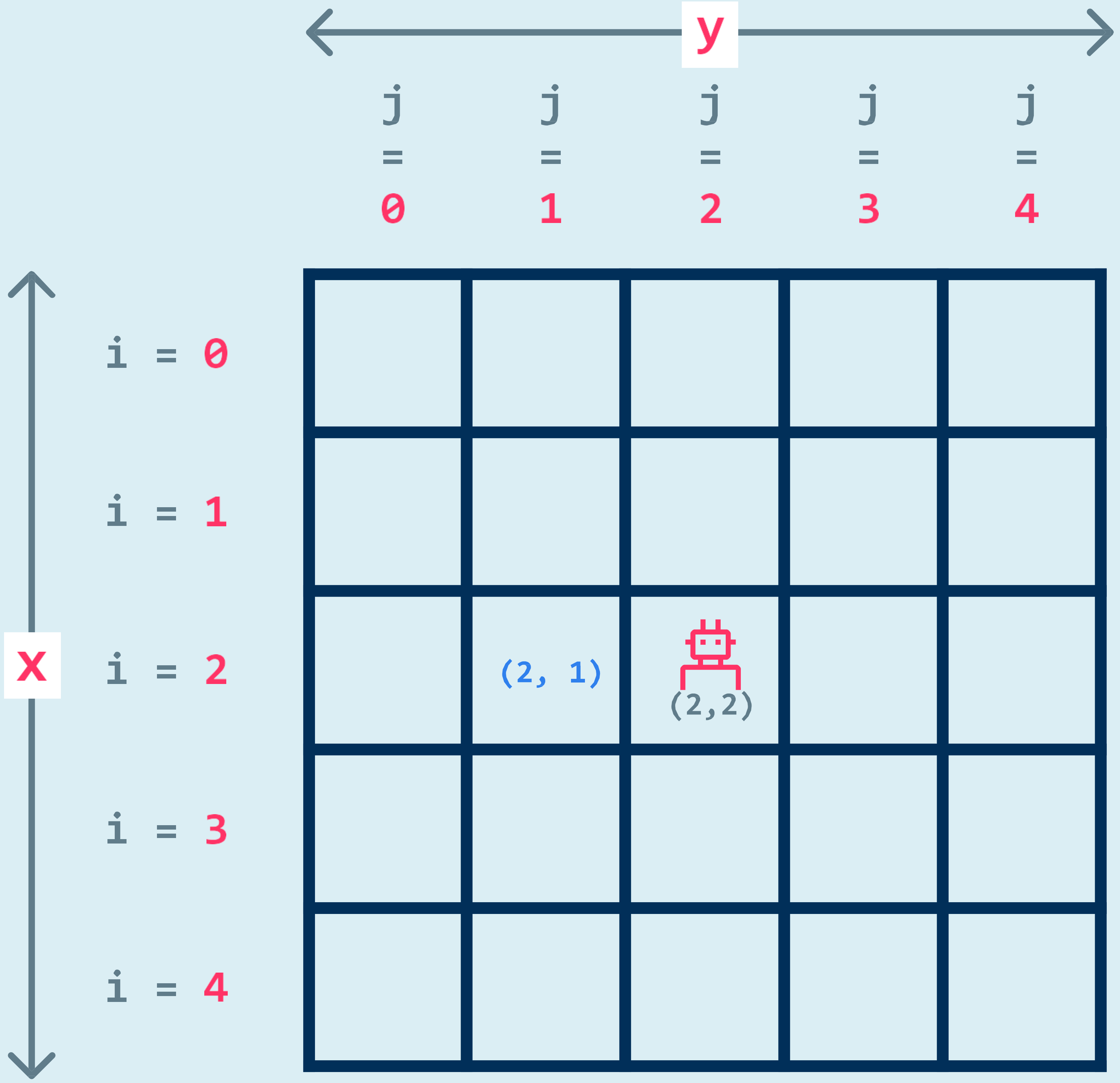


adjacent cell co-ordinate	change in x (dx)	change in y (dy)
Top (1, 2)	-1	0
Bottom (3, 2)	+1	0

Calculating Neighbor Cells

Finding Co-ordinate Pattern

- x = row-axis
- y = col-axis



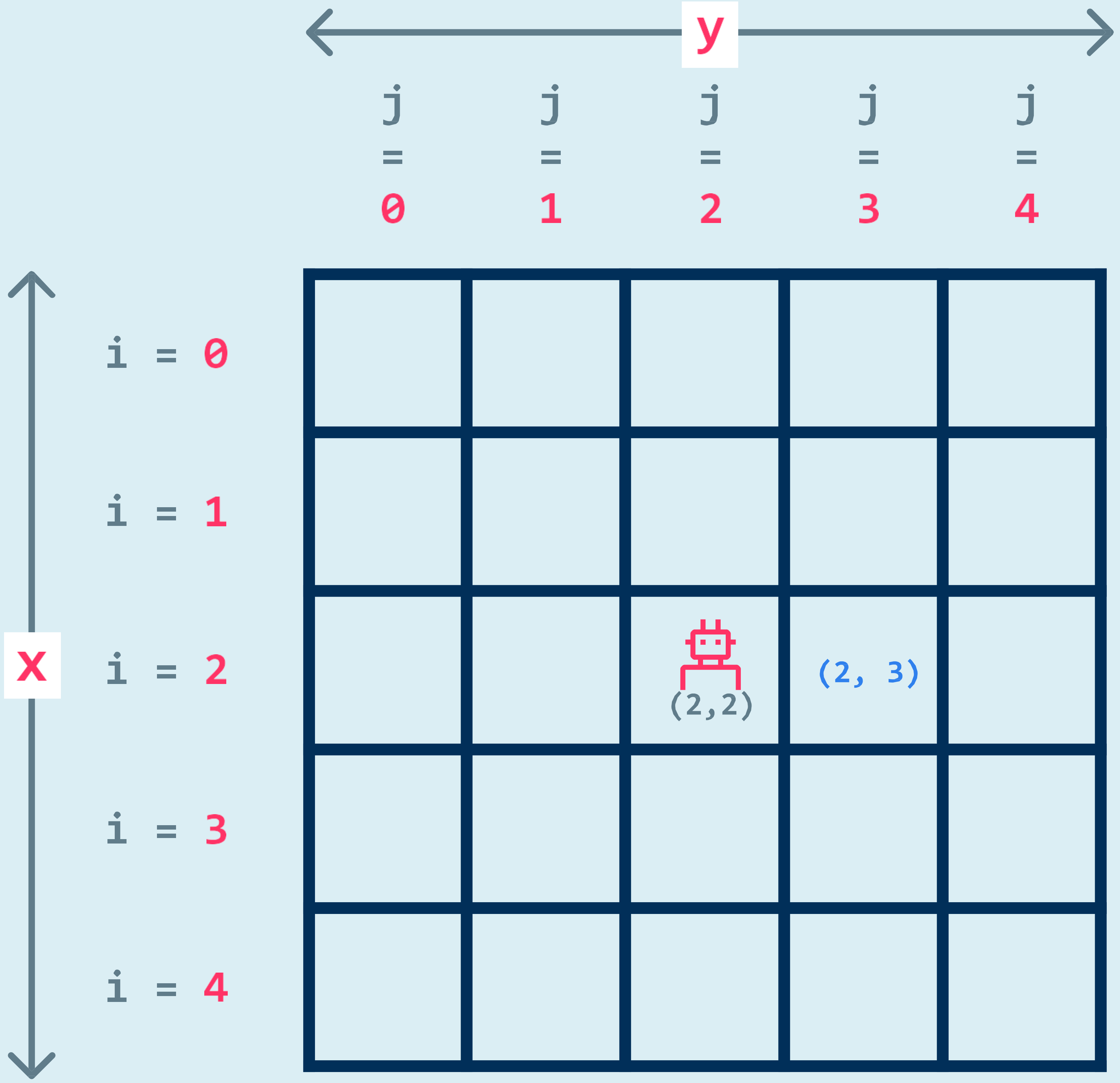
A[5][5]

adjacent cell co-ordinate	change in x (dx)	change in y (dy)
Top (1, 2)	-1	0
Bottom (3, 2)	+1	0
Left (2, 1)	0	-1

Calculating Neighbor Cells

Finding Co-ordinate Pattern

- x = row-axis
- y = col-axis



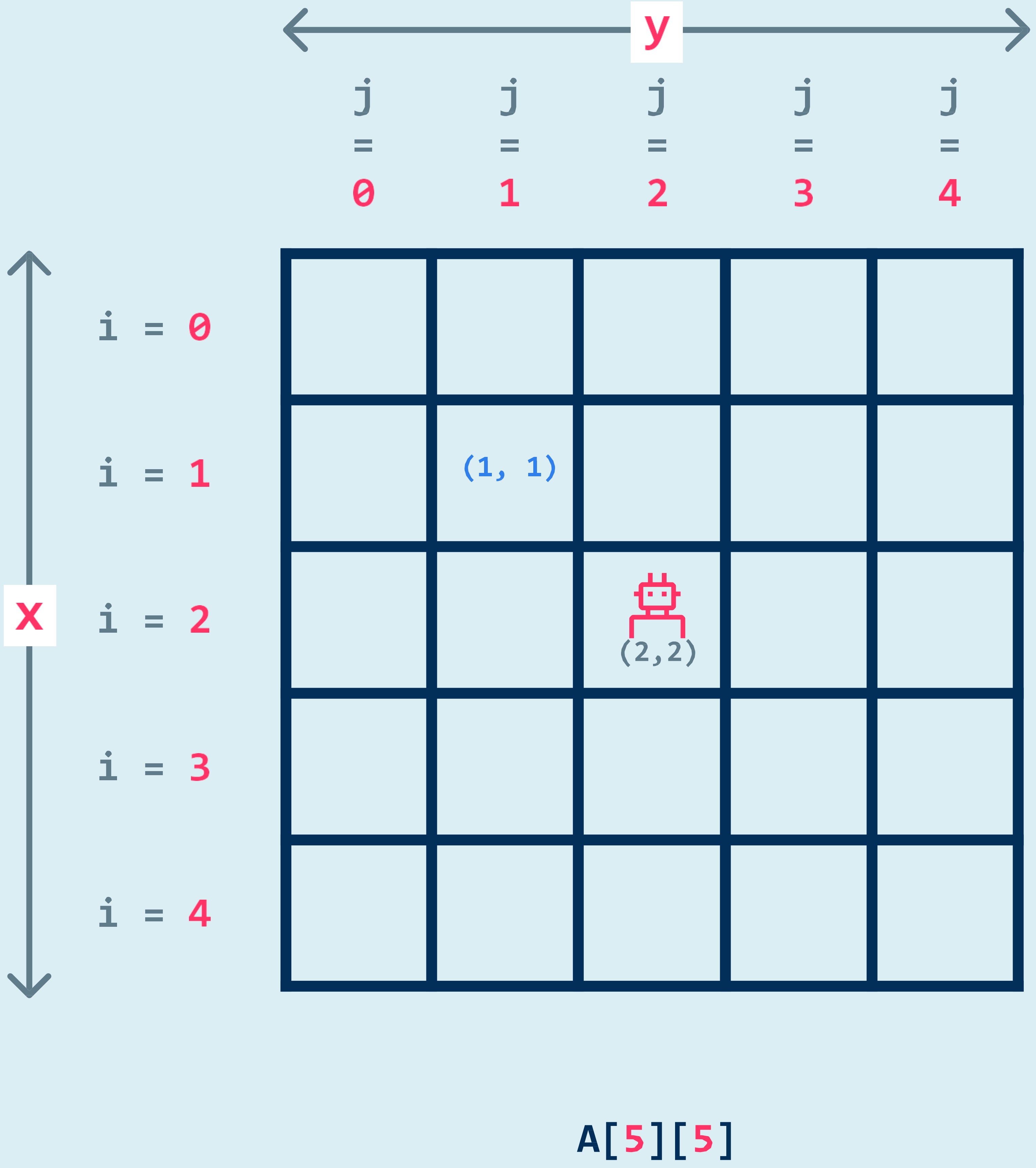
A[5][5]

adjacent cell co-ordinate	change in x (dx)	change in y (dy)
Top (1, 2)	-1	0
Bottom (3, 2)	+1	0
Left (2, 1)	0	-1
Right (2, 3)	0	+1

Calculating Neighbor Cells

Finding Co-ordinate Pattern

- x = row-axis
- y = col-axis

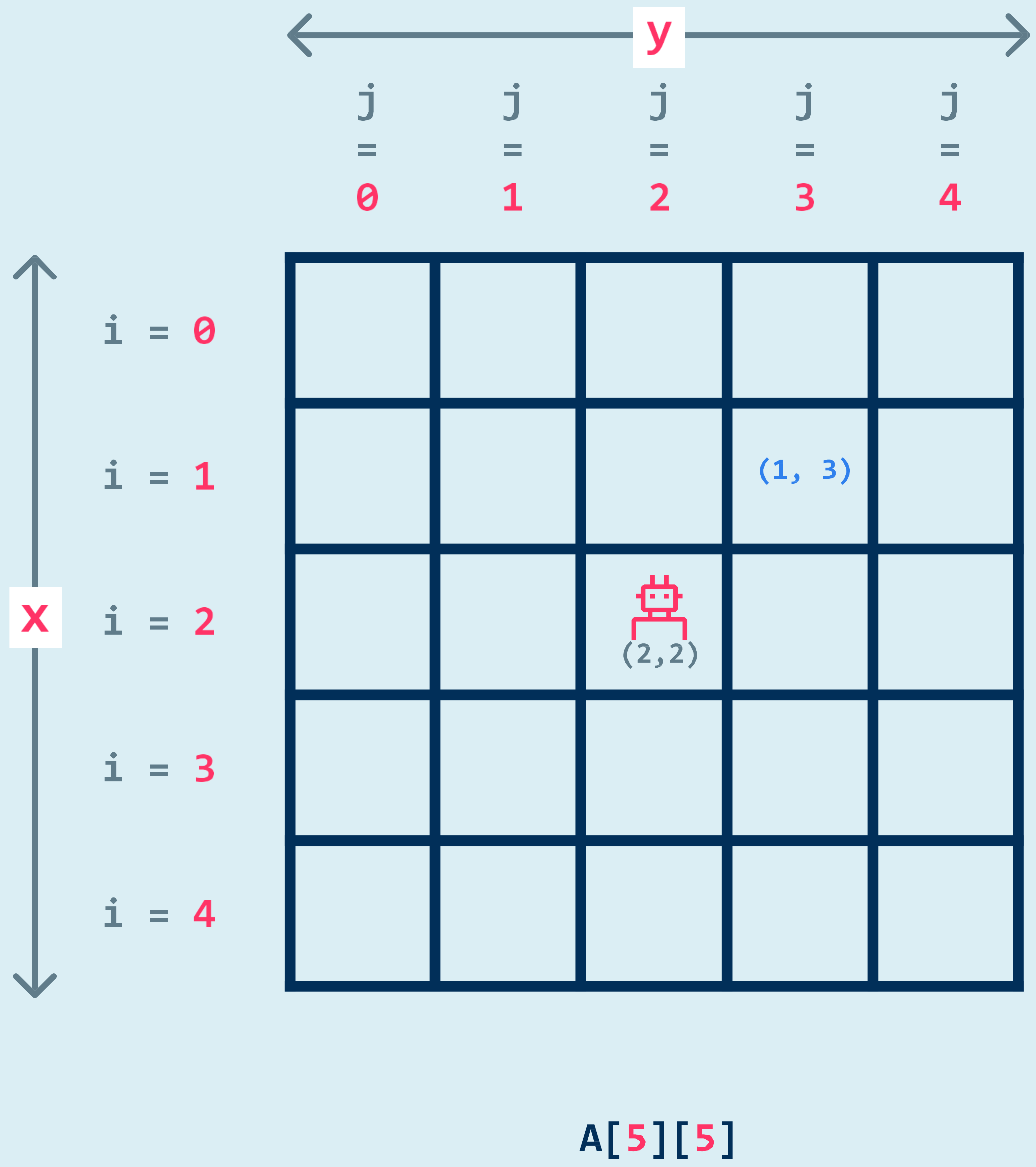


adjacent cell co-ordinate	change in x (dx)	change in y (dy)
Top (1, 2)	-1	0
Bottom (3, 2)	+1	0
Left (2, 1)	0	-1
Right (2, 3)	0	+1
Top-Left (1, 1)	-1	-1

Calculating Neighbor Cells

Finding Co-ordinate Pattern

- x = row-axis
- y = col-axis

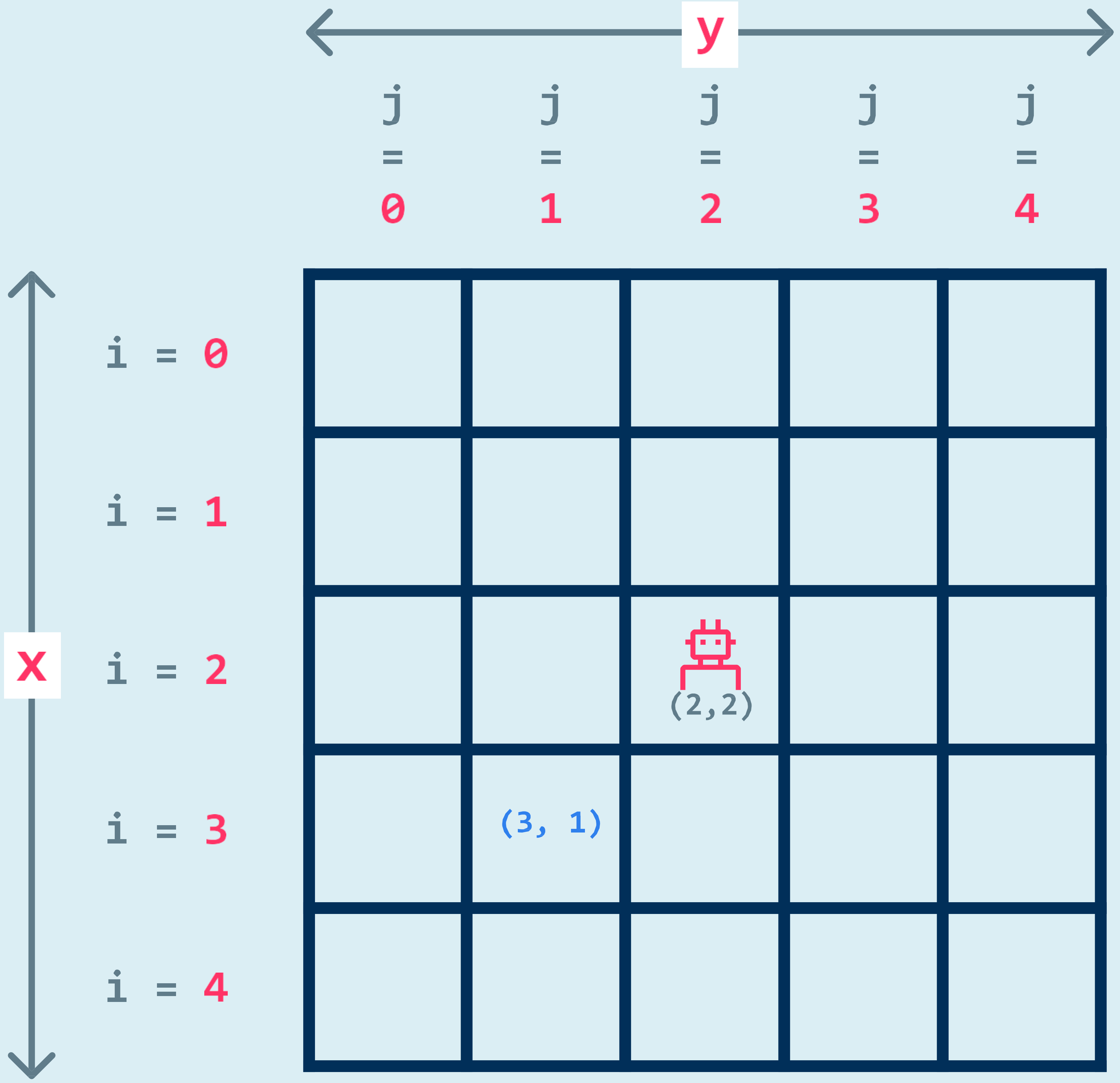


adjacent cell co-ordinate	change in x (dx)	change in y (dy)
Top (1, 2)	-1	0
Bottom (3, 2)	+1	0
Left (2, 1)	0	-1
Right (2, 3)	0	+1
Top-Left (1, 1)	-1	-1
Top-Right (1, 3)	-1	+1

Calculating Neighbor Cells

Finding Co-ordinate Pattern

- x = row-axis
- y = col-axis

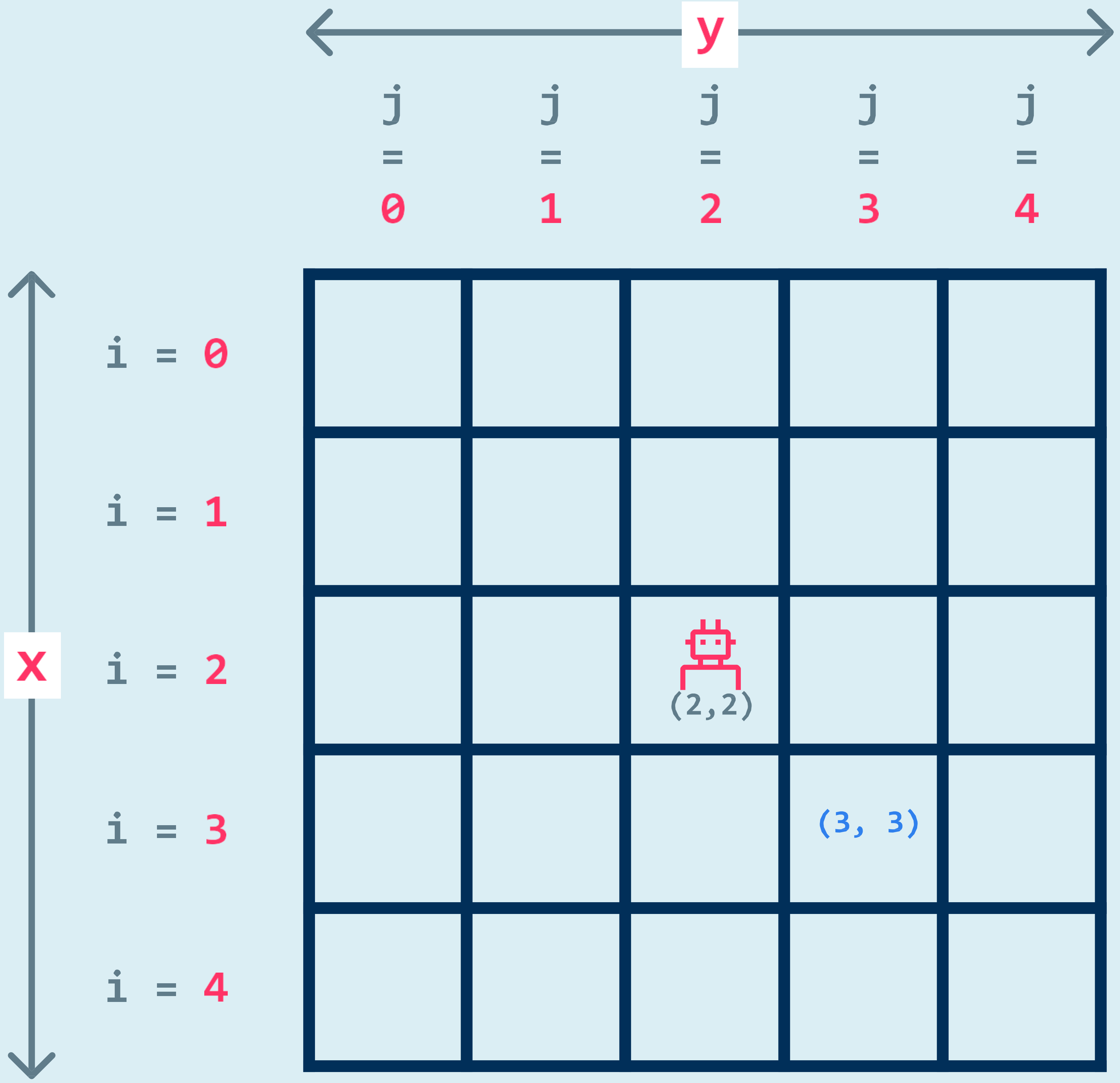


A[5][5]

adjacent cell co-ordinate	change in x (dx)	change in y (dy)
Top (1, 2)	-1	0
Bottom (3, 2)	+1	0
Left (2, 1)	0	-1
Right (2, 3)	0	+1
Top-Left (1, 1)	-1	-1
Top-Right (1, 3)	-1	+1
Bottom-Left (3, 1)	+1	-1

Calculating Neighbor Cells

- x = row-axis
- y = col-axis



A[5][5]

adjacent cell co-ordinate	change in x (dx)	change in y (dy)
Top (1, 2)	-1	0
Bottom (3, 2)	+1	0
Left (2, 1)	0	-1
Right (2, 3)	0	+1
Top-Left (1, 1)	-1	-1
Top-Right (1, 3)	-1	+1
Bottom-Left (3, 1)	+1	-1
Bottom-Right (3, 3)	+1	+1

So, the neighbor calculations would be..

- x = row-axis
- y = col-axis

```
void dfs(int x, int y){
    // make sure the cell falls within the grid

    // visit the cell

    dfs(x-1, y);
    dfs(x+1, y);
    dfs(x, y-1);
    dfs(x, y+1);
    dfs(x-1, y-1);
    dfs(x-1, y+1);
    dfs(x+1, y-1);
    dfs(x+1, y+1);
}
```

How can we simplify this?

Directional Array

Array to hold dx and dy so we can add the corresponding values to current cell value and get the neighbors.

- x = row-axis
- y = col-axis

Directional Array

Array to hold dx and dy so we can add the corresponding values to current cell value and get the neighbors.

- x = row-axis
- y = col-axis

adjacent cell co-ordinate	change in x (dx)	change in y (dy)
Top (1, 2)	-1	0
Bottom (3, 2)	+1	0
Left (2, 1)	0	-1
Right (2, 3)	0	+1
Top-Left (1, 1)	-1	-1
Top-Right (1, 3)	-1	+1
Bottom-Left (3, 1)	+1	-1
Bottom-Right (3, 3)	+1	+1

Directional Array

Array to hold dx and dy so we can add the corresponding values to current cell value and get the neighbors.

- x = row-axis
- y = col-axis

adjacent cell co-ordinate	change in x (dx)	change in y (dy)
Top (1, 2)	-1	0
Bottom (3, 2)	+1	0
Left (2, 1)	0	-1
Right (2, 3)	0	+1
Top-Left (1, 1)	-1	-1
Top-Right (1, 3)	-1	+1
Bottom-Left (3, 1)	+1	-1
Bottom-Right (3, 3)	+1	+1

```
dx[] = {}

dy[] = {}
```

- we take two arrays, where ith index of dx array will correspond to change in x and ith index of dy will correspond to change in y to get ith neighbor

Directional Array

Array to hold dx and dy so we can add the corresponding values to current cell value and get the neighbors.

- x = row-axis
- y = col-axis

adjacent cell co-ordinate	change in x (dx)	change in y (dy)
Top (1, 2)	-1	0
Bottom (3, 2)	+1	0
Left (2, 1)	0	-1
Right (2, 3)	0	+1
Top-Left (1, 1)	-1	-1
Top-Right (1, 3)	-1	+1
Bottom-Left (3, 1)	+1	-1
Bottom-Right (3, 3)	+1	+1

```
dx[] = {-1}

dy[] = {0}
```

- we take two arrays, where ith index of dx array will correspond to change in x and ith index of dy will correspond to change in y to get ith neighbor

Directional Array

Array to hold dx and dy so we can add the corresponding values to current cell value and get the neighbors.

- x = row-axis
- y = col-axis

adjacent cell co-ordinate	change in x (dx)	change in y (dy)
Top (1, 2)	-1	0
Bottom (3, 2)	+1	0
Left (2, 1)	0	-1
Right (2, 3)	0	+1
Top-Left (1, 1)	-1	-1
Top-Right (1, 3)	-1	+1
Bottom-Left (3, 1)	+1	-1
Bottom-Right (3, 3)	+1	+1

```
dx[] = {-1, 1}

dy[] = {0,  0}
```

- we take two arrays, where ith index of dx array will correspond to change in x and ith index of dy will correspond to change in y to get ith neighbor

Directional Array

Array to hold dx and dy so we can add the corresponding values to current cell value and get the neighbors.

- x = row-axis
- y = col-axis

adjacent cell co-ordinate	change in x (dx)	change in y (dy)
Top (1, 2)	-1	0
Bottom (3, 2)	+1	0
Left (2, 1)	0	-1
Right (2, 3)	0	+1
Top-Left (1, 1)	-1	-1
Top-Right (1, 3)	-1	+1
Bottom-Left (3, 1)	+1	-1
Bottom-Right (3, 3)	+1	+1

```
dx[] = {-1, 1,  0}

dy[] = {0,  0, -1}
```

- we take two arrays, where ith index of dx array will correspond to change in x and ith index of dy will correspond to change in y to get ith neighbor

Directional Array

Array to hold dx and dy so we can add the corresponding values to current cell value and get the neighbors.

- x = row-axis
- y = col-axis

adjacent cell co-ordinate	change in x (dx)	change in y (dy)
Top (1, 2)	-1	0
Bottom (3, 2)	+1	0
Left (2, 1)	0	-1
Right (2, 3)	0	+1
Top-Left (1, 1)	-1	-1
Top-Right (1, 3)	-1	+1
Bottom-Left (3, 1)	+1	-1
Bottom-Right (3, 3)	+1	+1

```
dx[] = {-1, 1,  0, 0}

dy[] = {0,  0, -1, 1}
```

- we take two arrays, where ith index of dx array will correspond to change in x and ith index of dy will correspond to change in y to get ith neighbor

Directional Array

Array to hold dx and dy so we can add the corresponding values to current cell value and get the neighbors.

- x = row-axis
- y = col-axis

adjacent cell co-ordinate	change in x (dx)	change in y (dy)
Top (1, 2)	-1	0
Bottom (3, 2)	+1	0
Left (2, 1)	0	-1
Right (2, 3)	0	+1
Top-Left (1, 1)	-1	-1
Top-Right (1, 3)	-1	+1
Bottom-Left (3, 1)	+1	-1
Bottom-Right (3, 3)	+1	+1

```
dx[] = {-1, 1,  0, 0, -1}

dy[] = {0,  0, -1, 1, -1}
```

- we take two arrays, where ith index of dx array will correspond to change in x and ith index of dy will correspond to change in y to get ith neighbor

Directional Array

Array to hold dx and dy so we can add the corresponding values to current cell value and get the neighbors.

- x = row-axis
- y = col-axis

adjacent cell co-ordinate	change in x (dx)	change in y (dy)
Top (1, 2)	-1	0
Bottom (3, 2)	+1	0
Left (2, 1)	0	-1
Right (2, 3)	0	+1
Top-Left (1, 1)	-1	-1
Top-Right (1, 3)	-1	+1
Bottom-Left (3, 1)	+1	-1
Bottom-Right (3, 3)	+1	+1

```
dx[] = {-1, 1,  0, 0, -1, -1}

dy[] = {0,  0, -1, 1, -1,  1}
```

- we take two arrays, where ith index of dx array will correspond to change in x and ith index of dy will correspond to change in y to get ith neighbor

Directional Array

Array to hold dx and dy so we can add the corresponding values to current cell value and get the neighbors.

- x = row-axis
- y = col-axis

adjacent cell co-ordinate	change in x (dx)	change in y (dy)
Top (1, 2)	-1	0
Bottom (3, 2)	+1	0
Left (2, 1)	0	-1
Right (2, 3)	0	+1
Top-Left (1, 1)	-1	-1
Top-Right (1, 3)	-1	+1
Bottom-Left (3, 1)	+1	-1
Bottom-Right (3, 3)	+1	+1

```
dx[] = {-1, 1,  0, 0, -1, -1,  1}
dy[] = {0,  0, -1, 1, -1,  1, -1}
```

- we take two arrays, where ith index of dx array will correspond to change in x and ith index of dy will correspond to change in y to get ith neighbor

Directional Array

Array to hold dx and dy so we can add the corresponding values to current cell value and get the neighbors.

- x = row-axis
- y = col-axis

adjacent cell co-ordinate	change in x (dx)	change in y (dy)
Top (1, 2)	-1	0
Bottom (3, 2)	+1	0
Left (2, 1)	0	-1
Right (2, 3)	0	+1
Top-Left (1, 1)	-1	-1
Top-Right (1, 3)	-1	+1
Bottom-Left (3, 1)	+1	-1
Bottom-Right (3, 3)	+1	+1

```
dx[] = {-1, 1,  0, 0, -1, -1,  1, 1}

dy[] = {0,  0, -1, 1, -1,  1, -1, 1}
```

- we take two arrays, where ith index of dx array will correspond to change in x and ith index of dy will correspond to change in y to get ith neighbor

Directional Array

- x = row-axis
- y = col-axis

```
dx[] = {-1, 1,  0, 0, -1, -1,  1, 1}
dy[] = {0,  0, -1, 1, -1,  1, -1, 1}


dfs(x, y){

   // visit x,y

   // calculate neighbors
   for (i = 0; i < 8; i++){
        next_x = x + dx[i];
        next_y = y + dy[i];

        dfs(next_x, next_y);
   }
}
```

Directional Array

- x = row-axis
- y = col-axis

```
dx[] = {-1, 1,  0, 0, -1, -1,  1, 1}
dy[] = {0,  0, -1, 1, -1,  1, -1, 1}
```

side moves | diagonal moves

- in some problems, only 4 side moves are allowed, so if we keep them in the first four indices, we can use the same directional array and iterate over first 4 indices instead of all 8

Directional Array for Knight Moves

- x = row-axis
- y = col-axis

```
kx[] = {1, 1, 2, 2, -1, -1, -2, -2}
ky[] = {2, -2, 1, -1, 2, -2, 1, -1}
```

think how it is constructed!