

COL106: Data Structures and Algorithms

Lecture 6

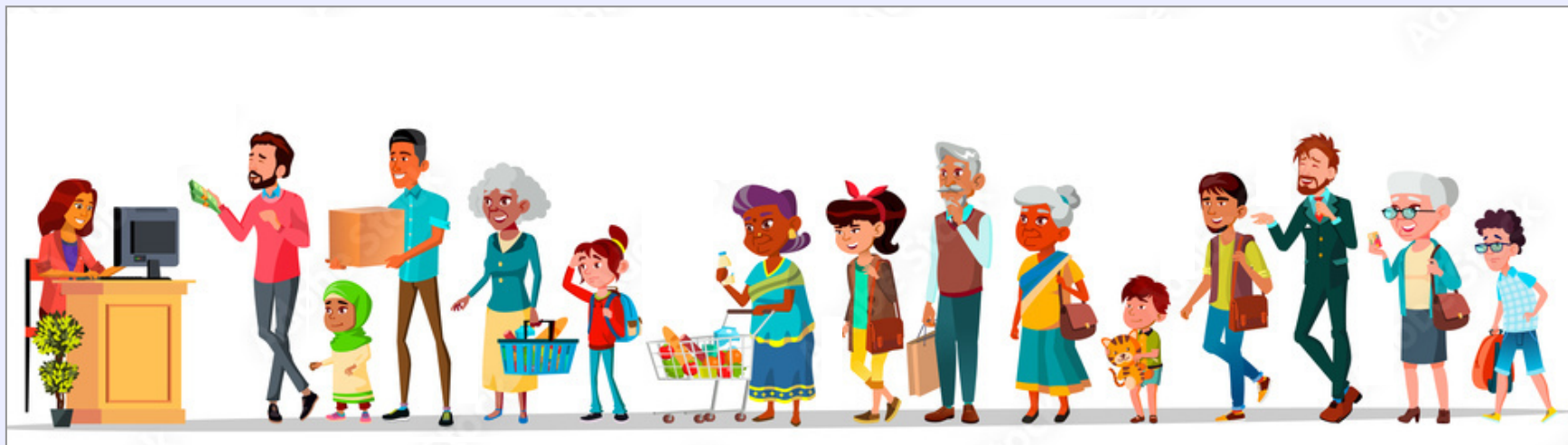
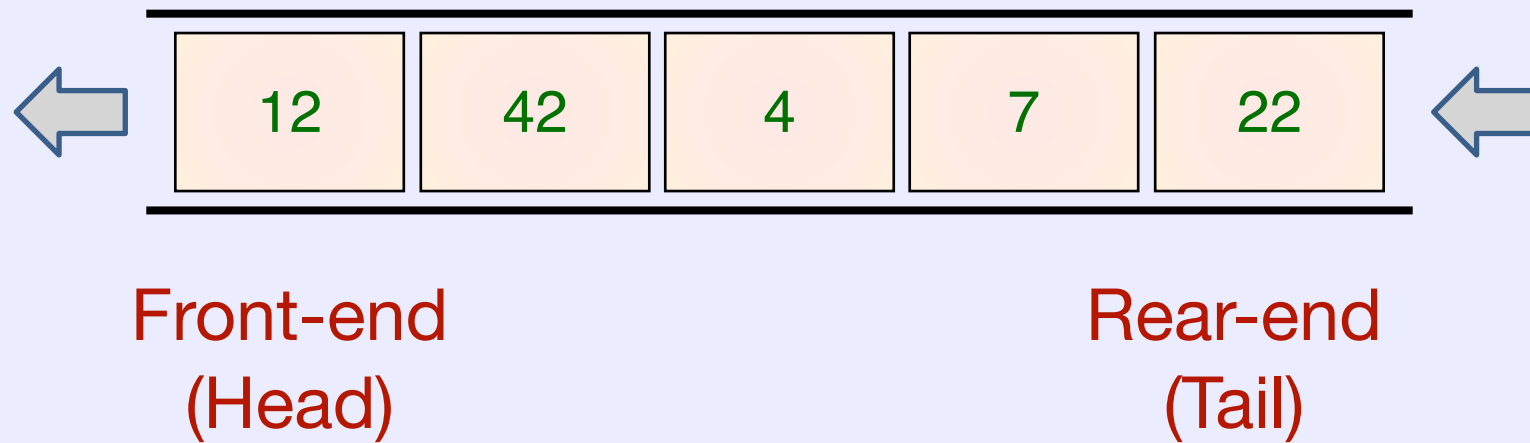
Keerti Choudhary
Department of Computer Science, IIT Delhi

Today's Lecture

1. Revisit Queues and shortest path problem in Maze
2. Deque data-structure
3. Range minima problem

Queue Data structure

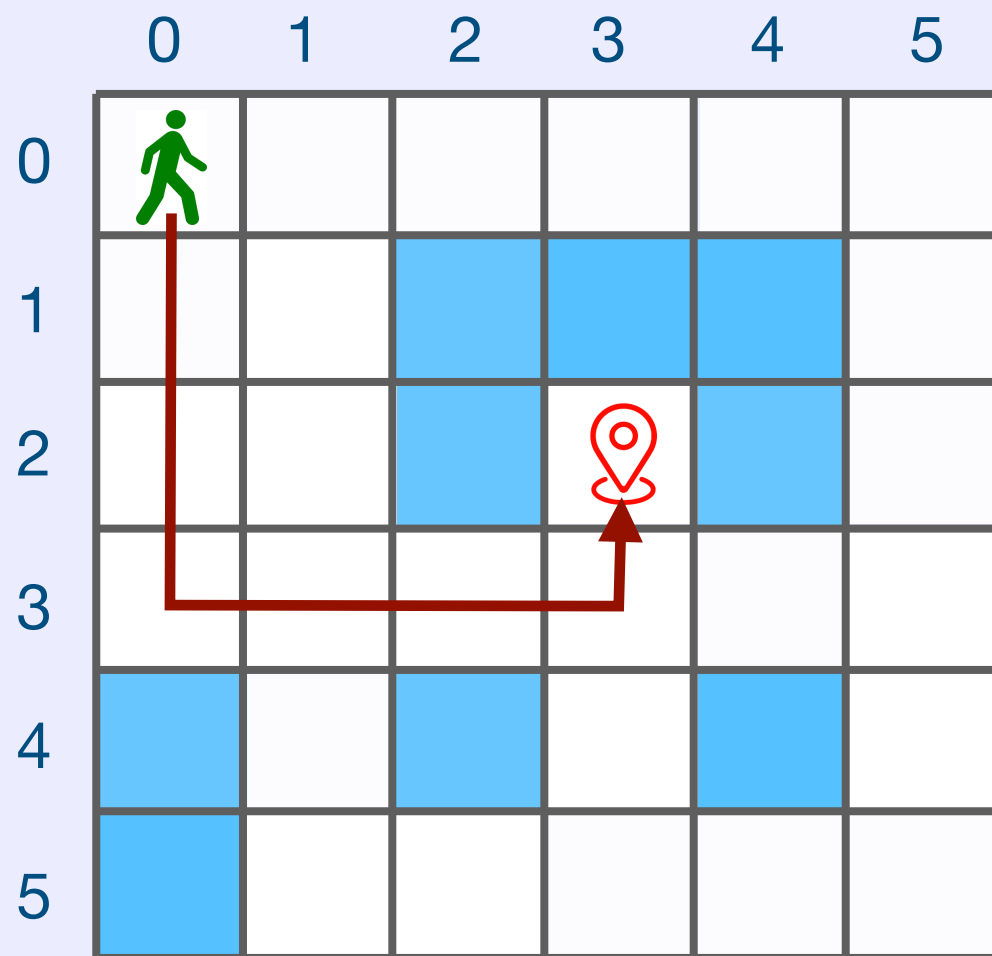
A data structure with FIFO (First In First Out):



Shortest Route in Maze

Given: A maze represented by a binary matrix M , starting cell, and destination cell.

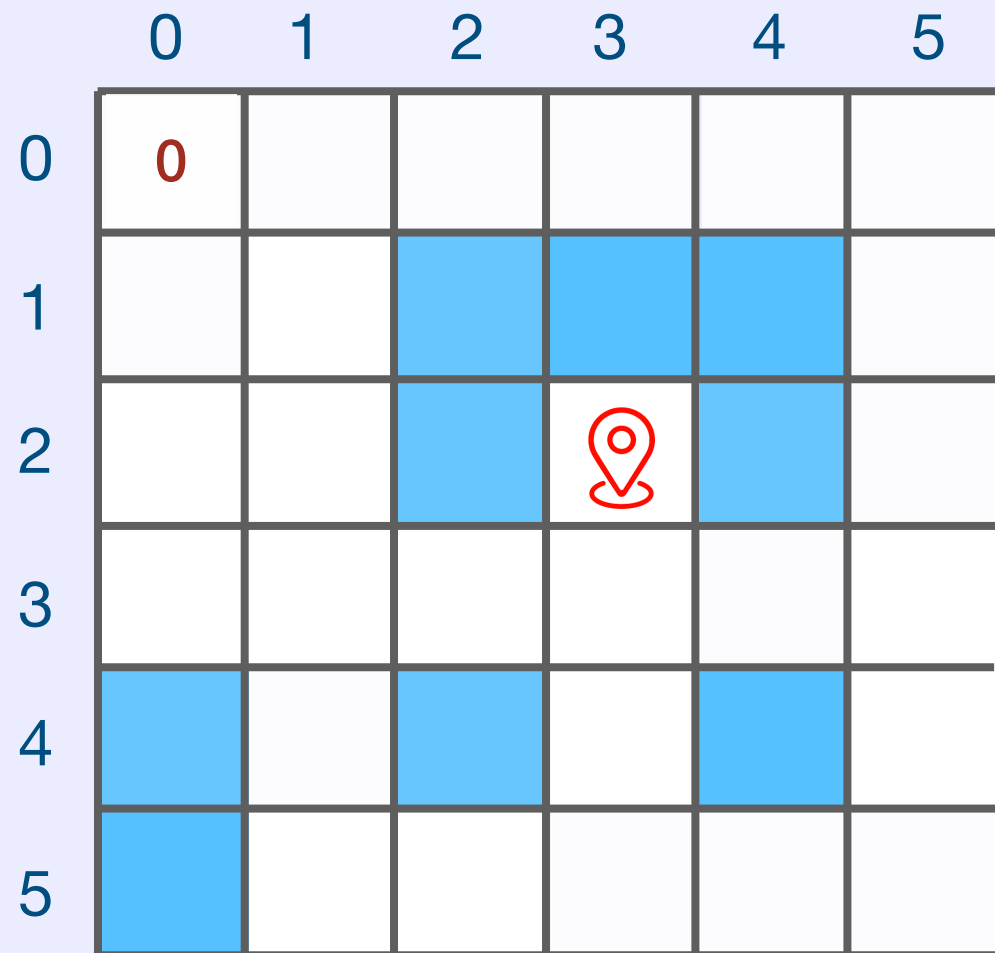
Goal: Compute shortest path to destination.



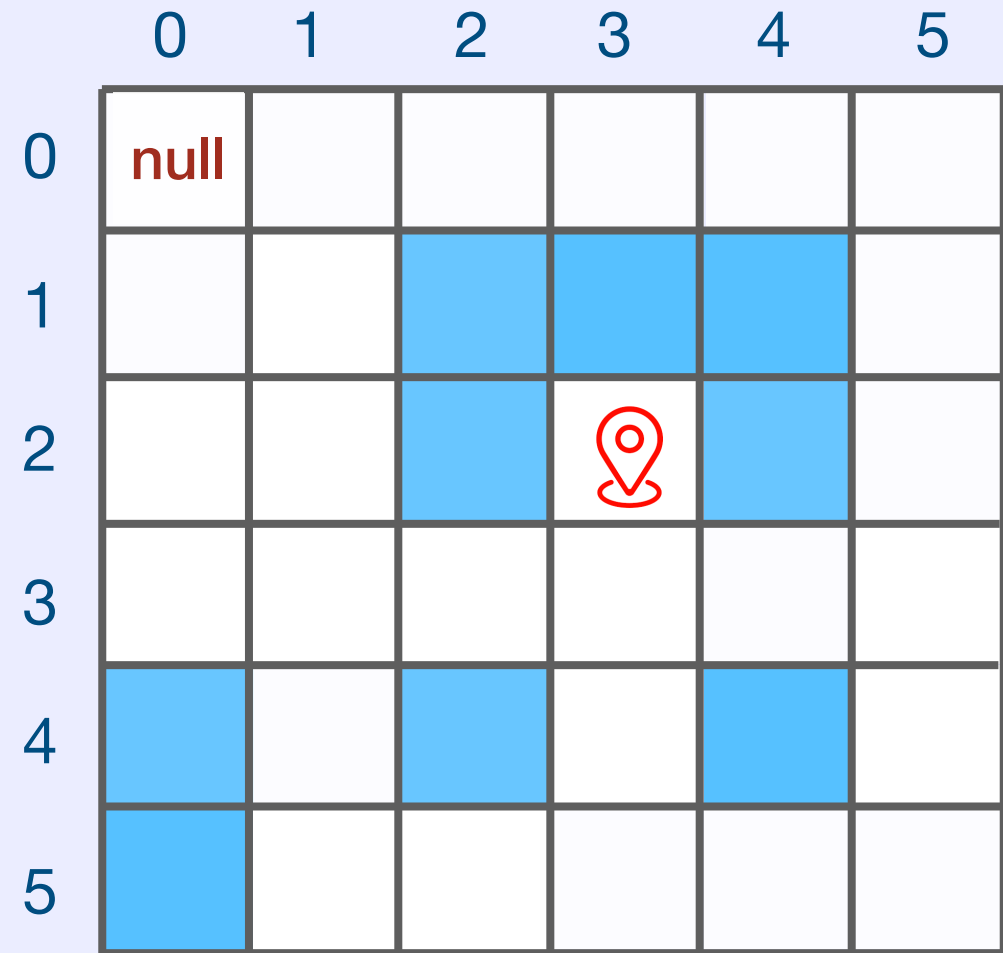
Algorithm sketch:

1. Create a queue and enqueue the cell (0,0); also set its distance as zero.
2. While queue is non-empty:
 - Dequeue first cell **c**
 - Enqueue(append) those neighbours **b** of **c** whose distance is not yet assigned.
 - Set $\text{Distance}(\mathbf{b}) = \text{Distance}(\mathbf{c}) + 1$.

Example



2d-array Distance

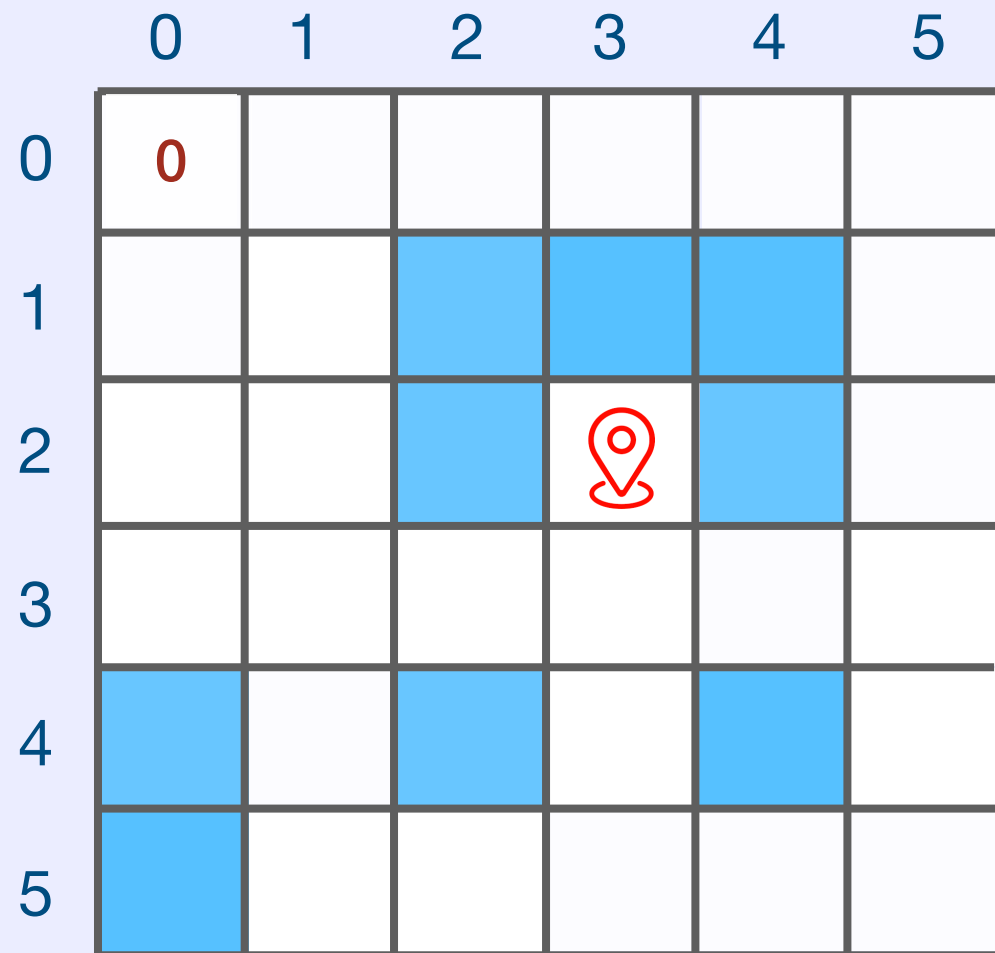


2d-array Parent

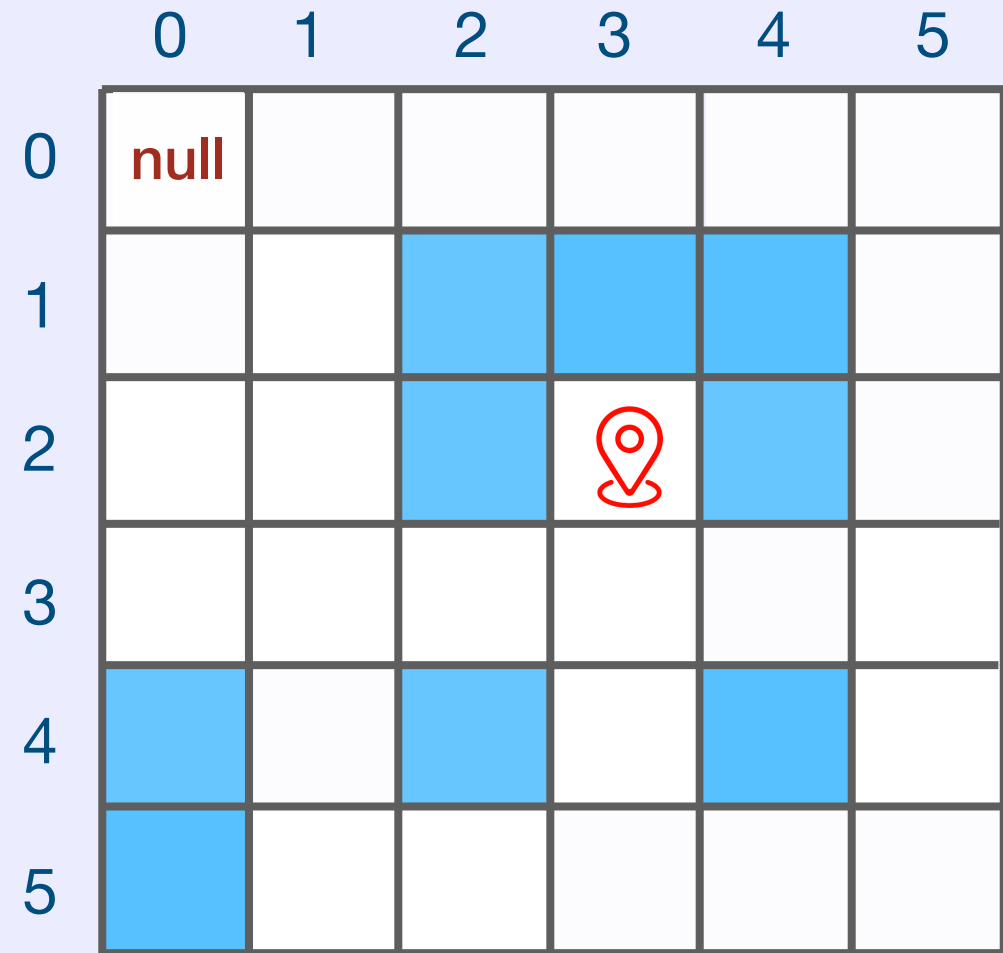
Queue:

(0,0)

Example



2d-array Distance



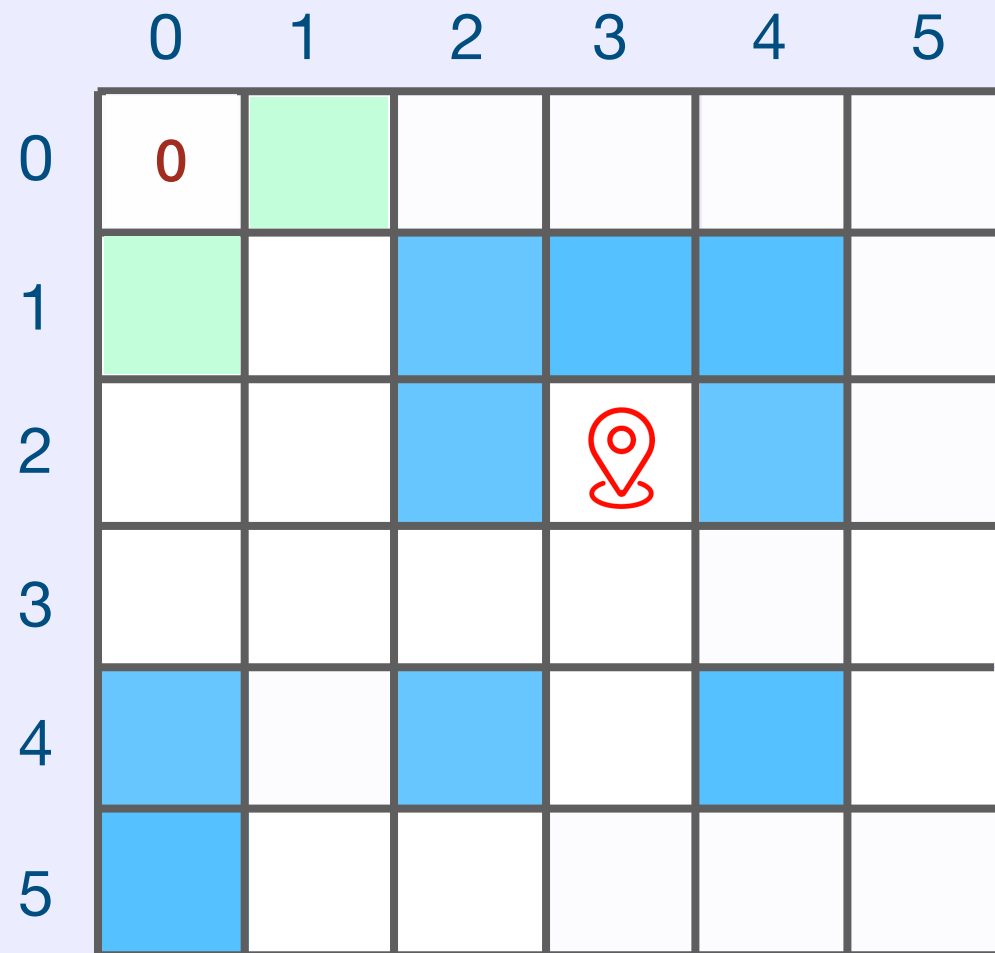
2d-array Parent



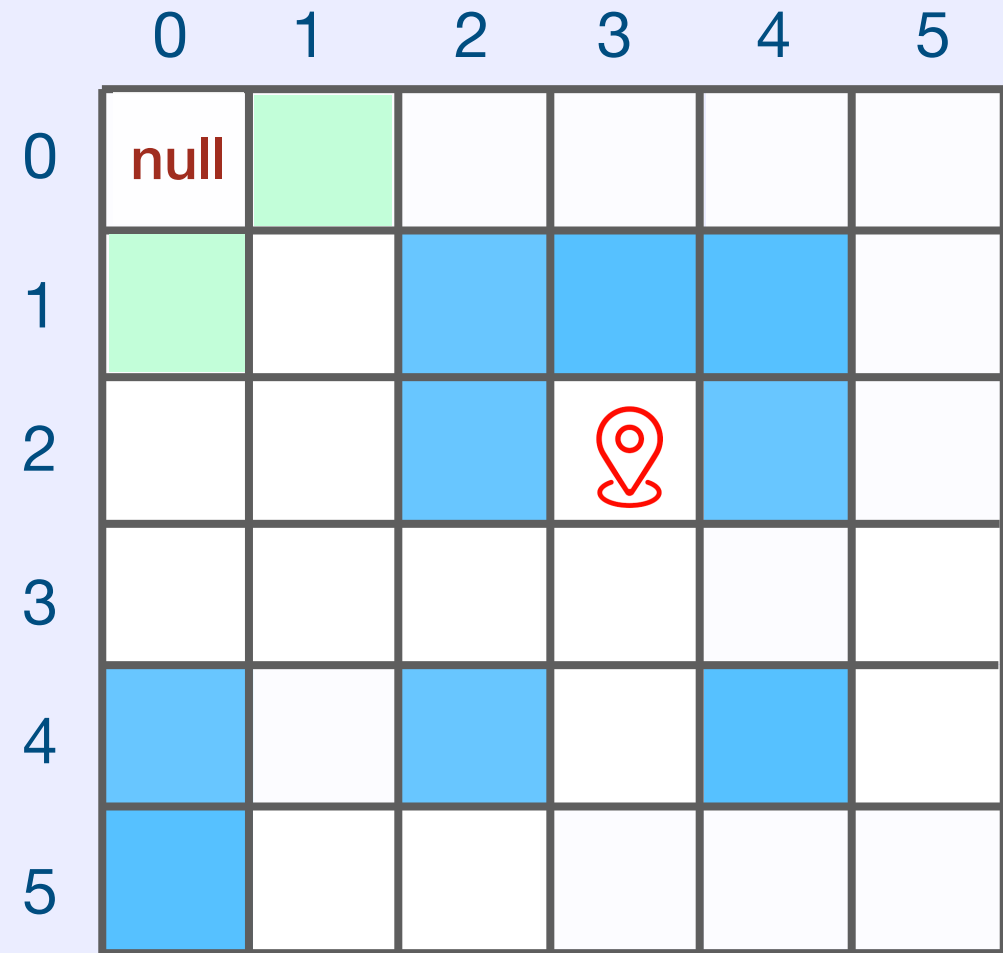
Queue:

(0,0)

Example



2d-array Distance



2d-array Parent



Queue:

(0,0)	(1,0)	(0,1)
-------	-------	-------

Example

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

2d-array Distance

	0	1	2	3	4	5
0	null	(0,0)				
1	(0,0)					
2						
3						
4						
5						

2d-array Parent



Queue:

(0,0)	(1,0)	(0,1)
-------	-------	-------

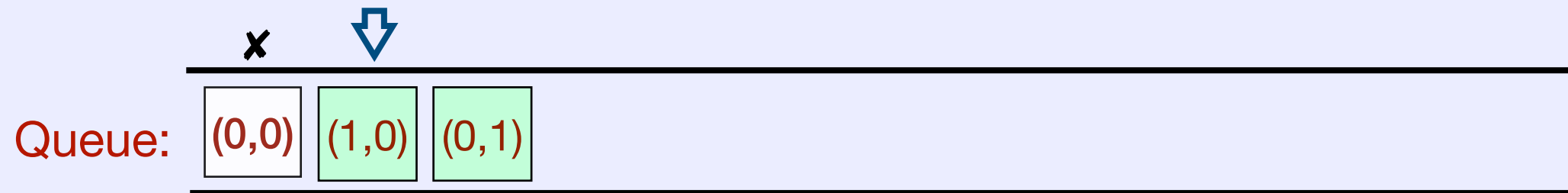
Example

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

2d-array Distance

	0	1	2	3	4	5
0	null	(0,0)				
1	(0,0)					
2						
3						
4						
5						

2d-array Parent



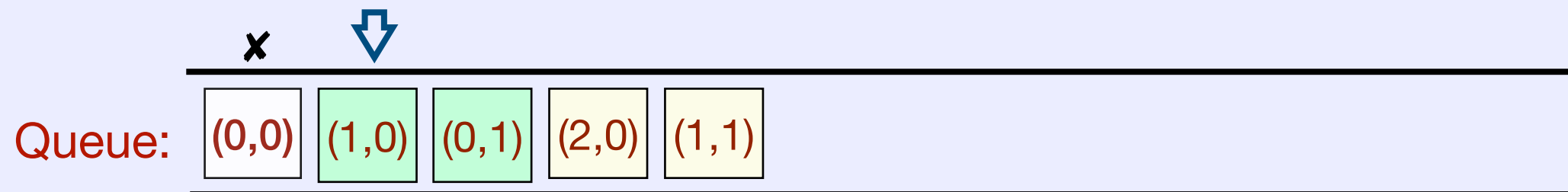
Example

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

2d-array Distance

	0	1	2	3	4	5
0	null	(0,0)				
1	(0,0)					
2						
3						
4						
5						

2d-array Parent



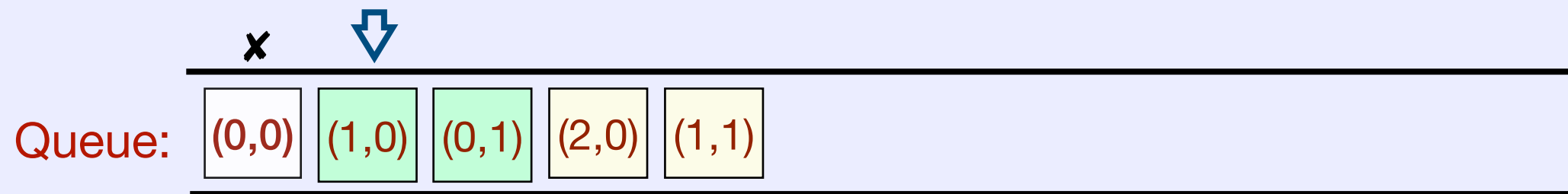
Example

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

2d-array Distance

	0	1	2	3	4	5
0	null	(0,0)				
1	(0,0)	(1,0)				
2	(1,0)					
3						
4						
5						

2d-array Parent



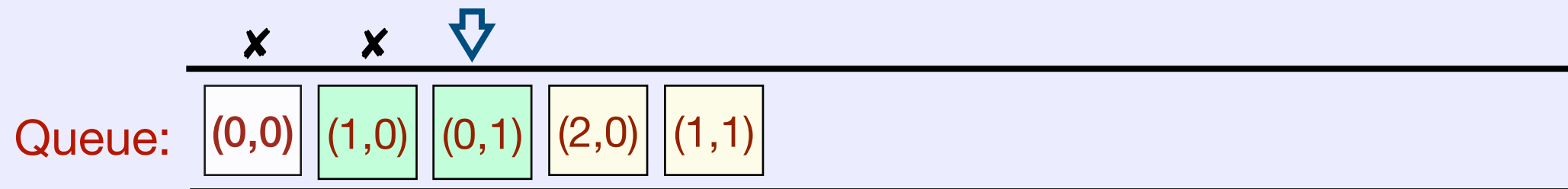
Example

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

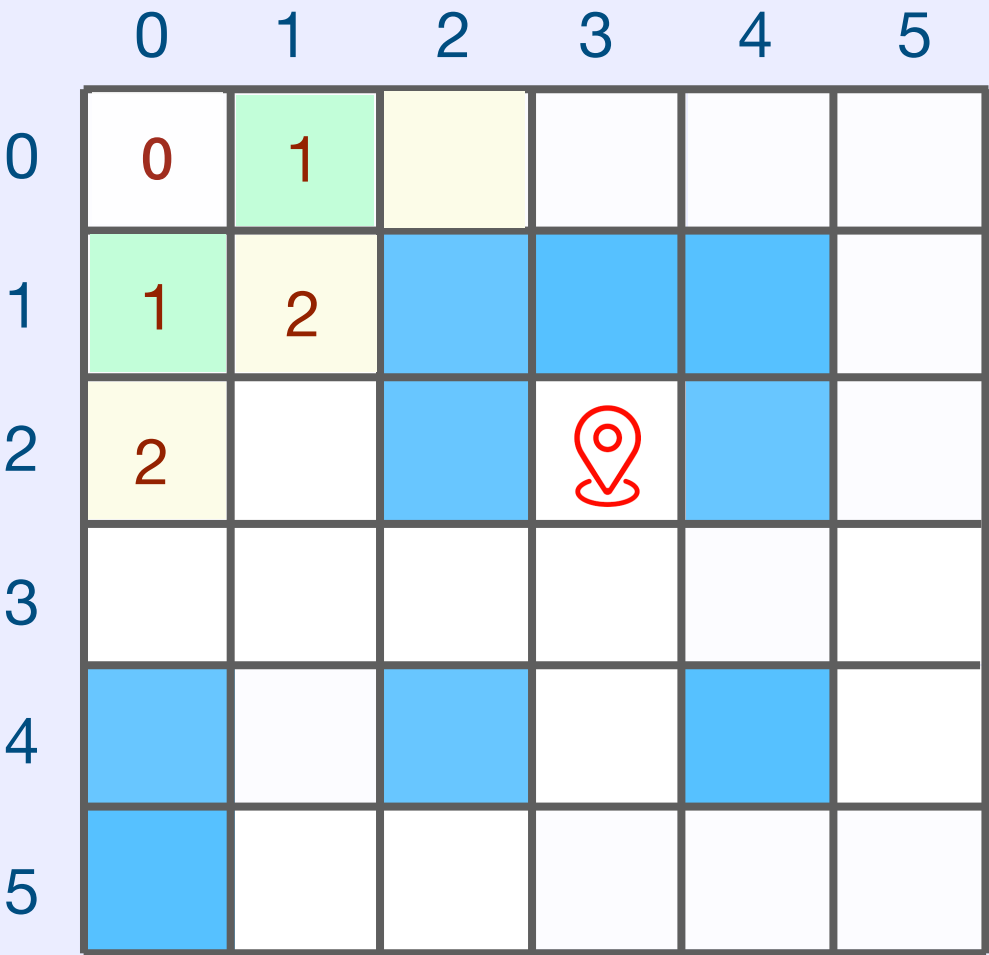
2d-array Distance

	0	1	2	3	4	5
0	null	(0,0)				
1	(0,0)	(1,0)				
2	(1,0)					
3						
4						
5						

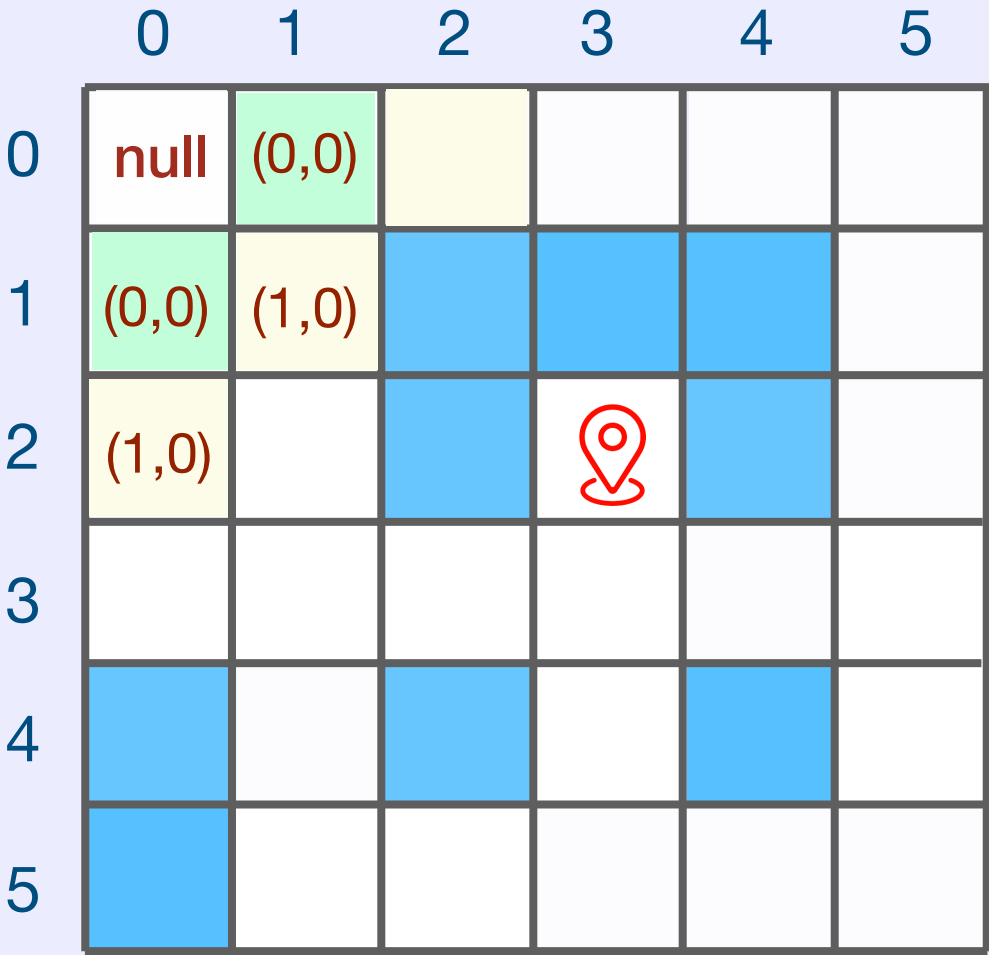
2d-array Parent



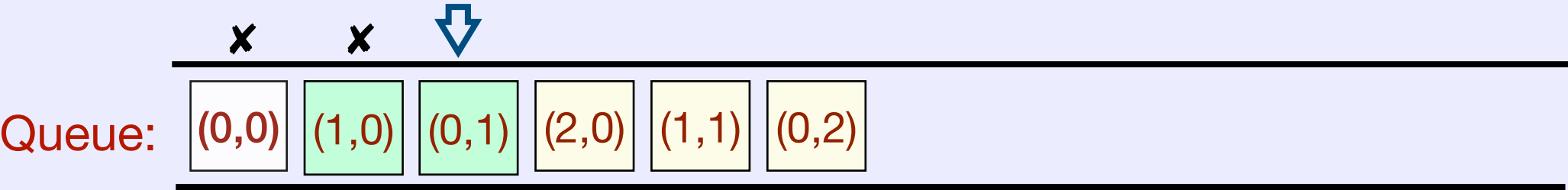
Example



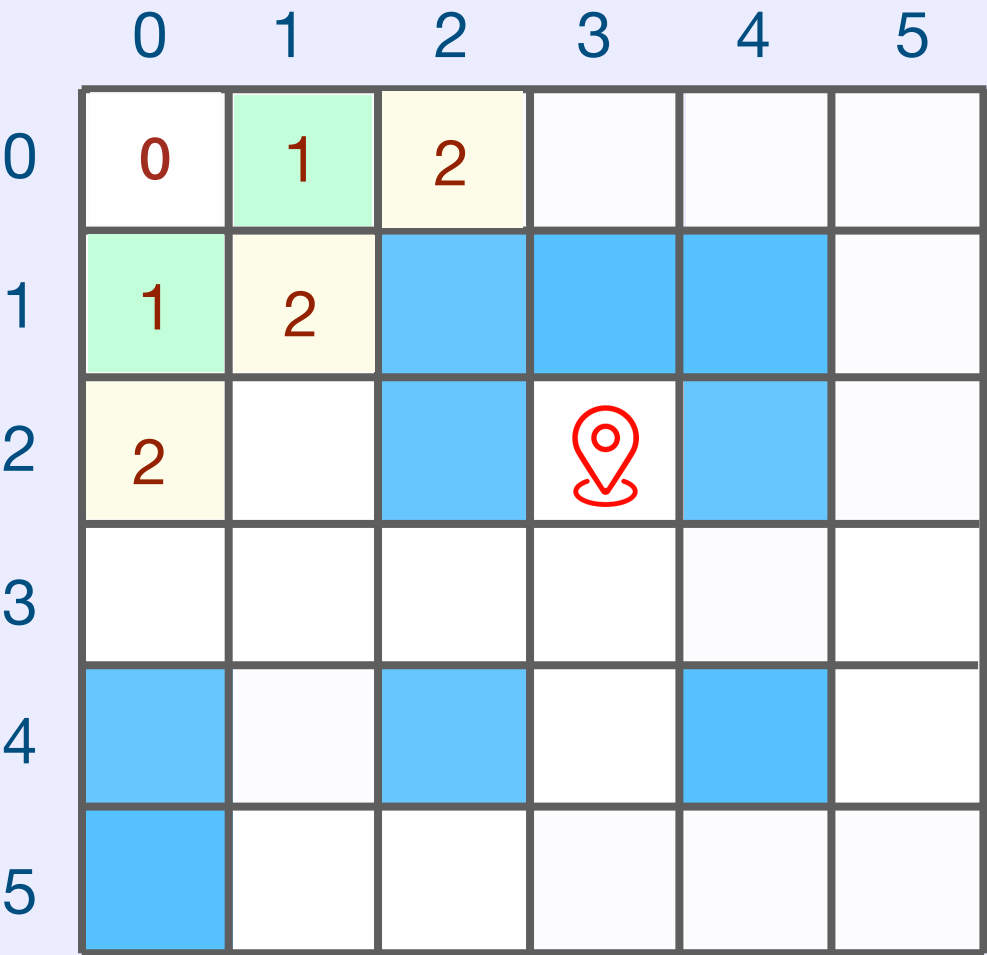
2d-array Distance



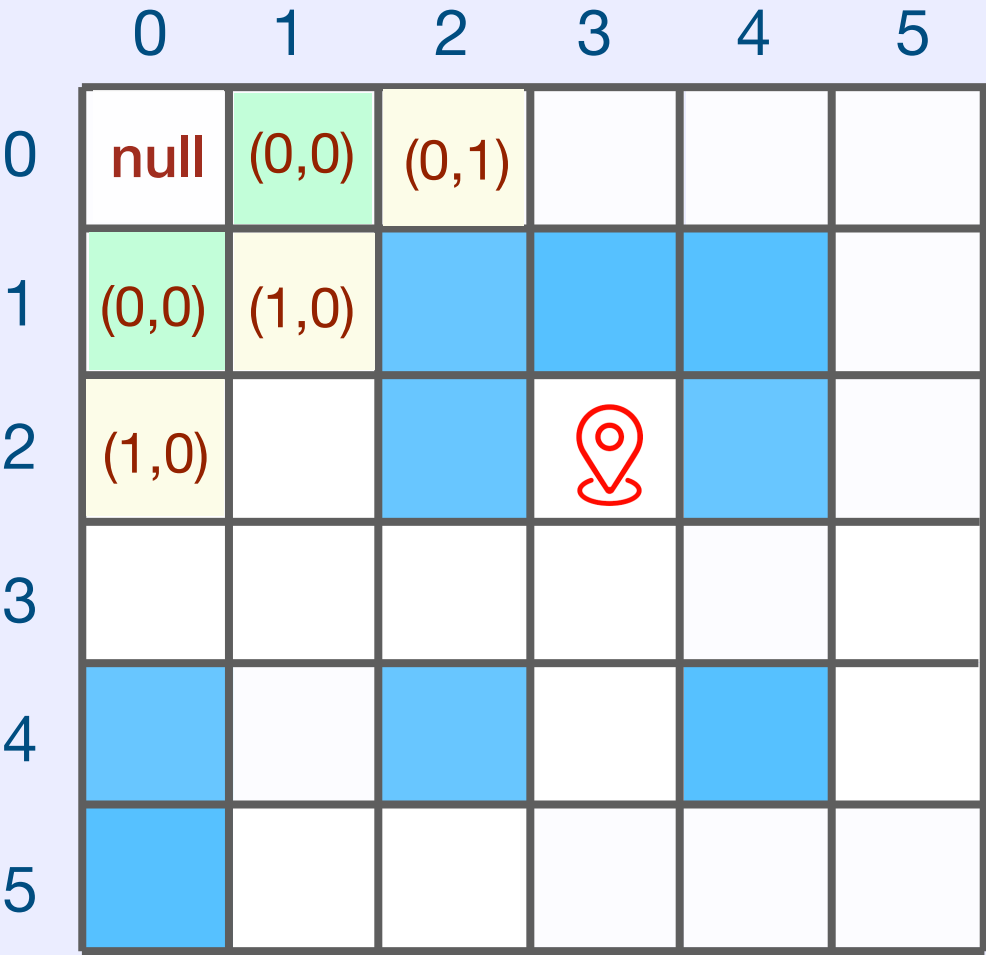
2d-array Parent



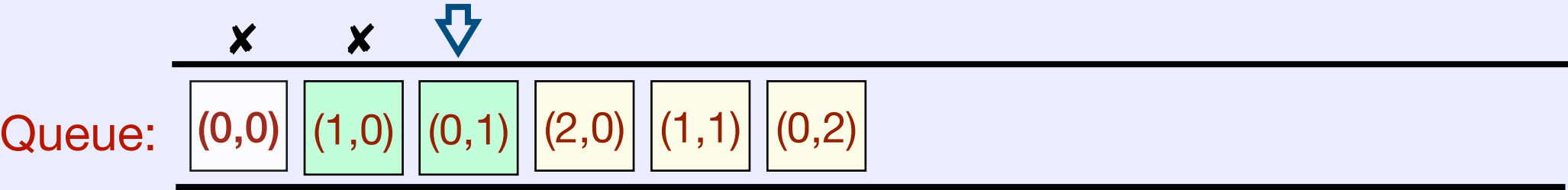
Example



2d-array Distance



2d-array Parent



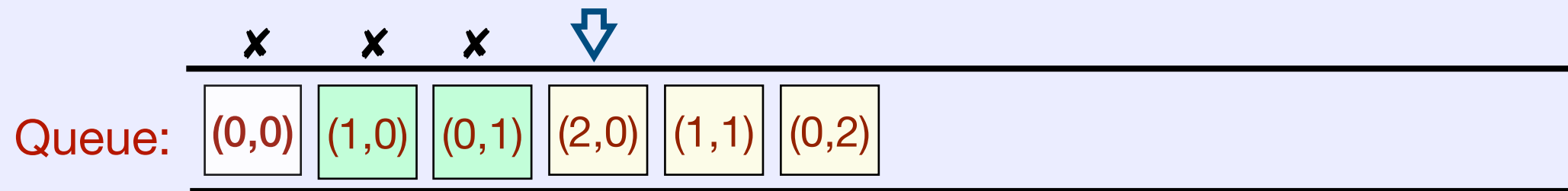
Example

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

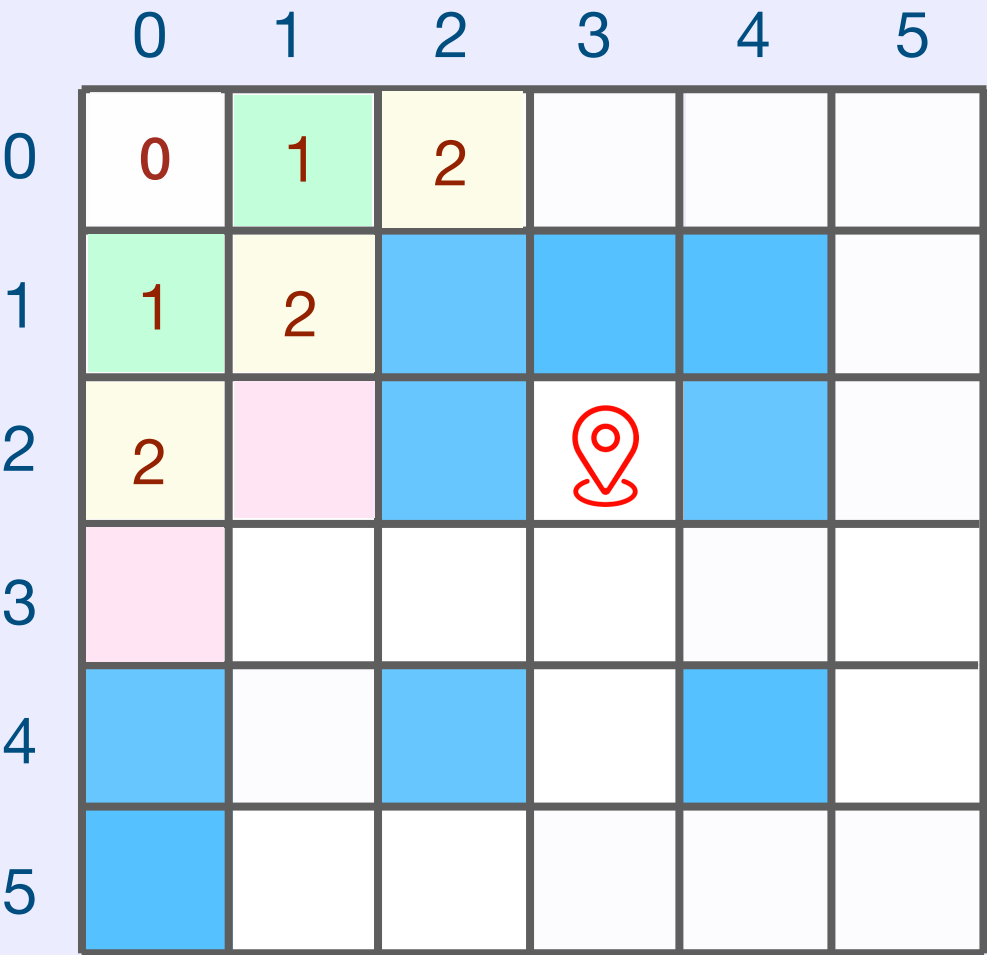
2d-array Distance

	0	1	2	3	4	5
0	null	(0,0)	(0,1)			
1	(0,0)	(1,0)				
2	(1,0)					
3						
4						
5						

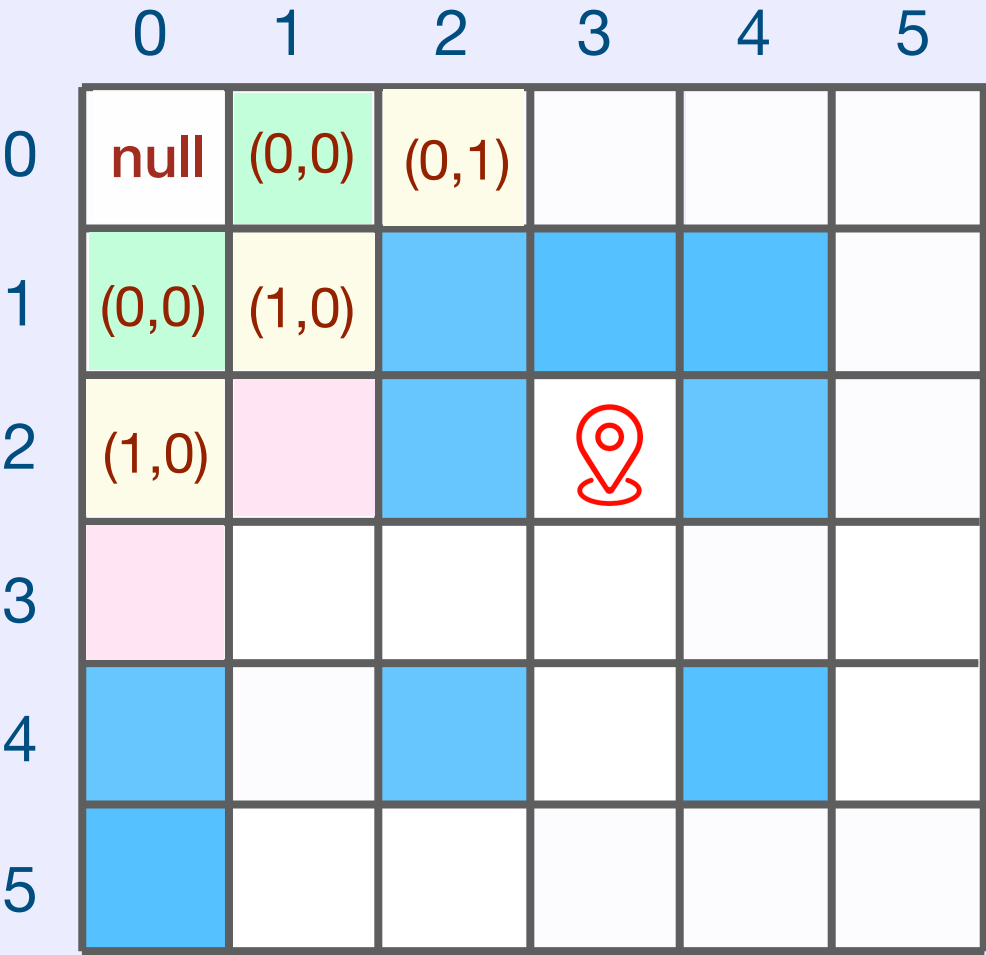
2d-array Parent



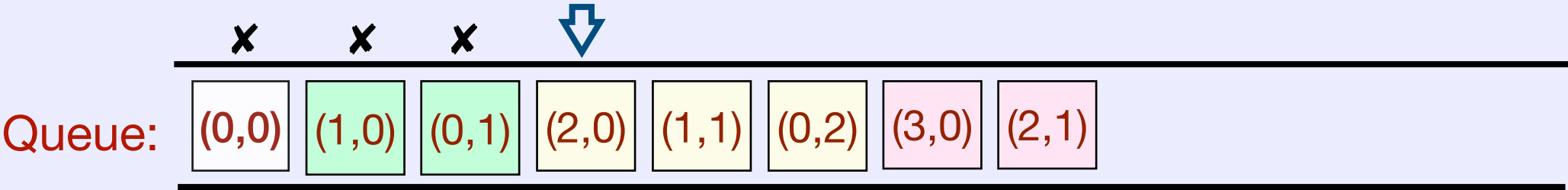
Example



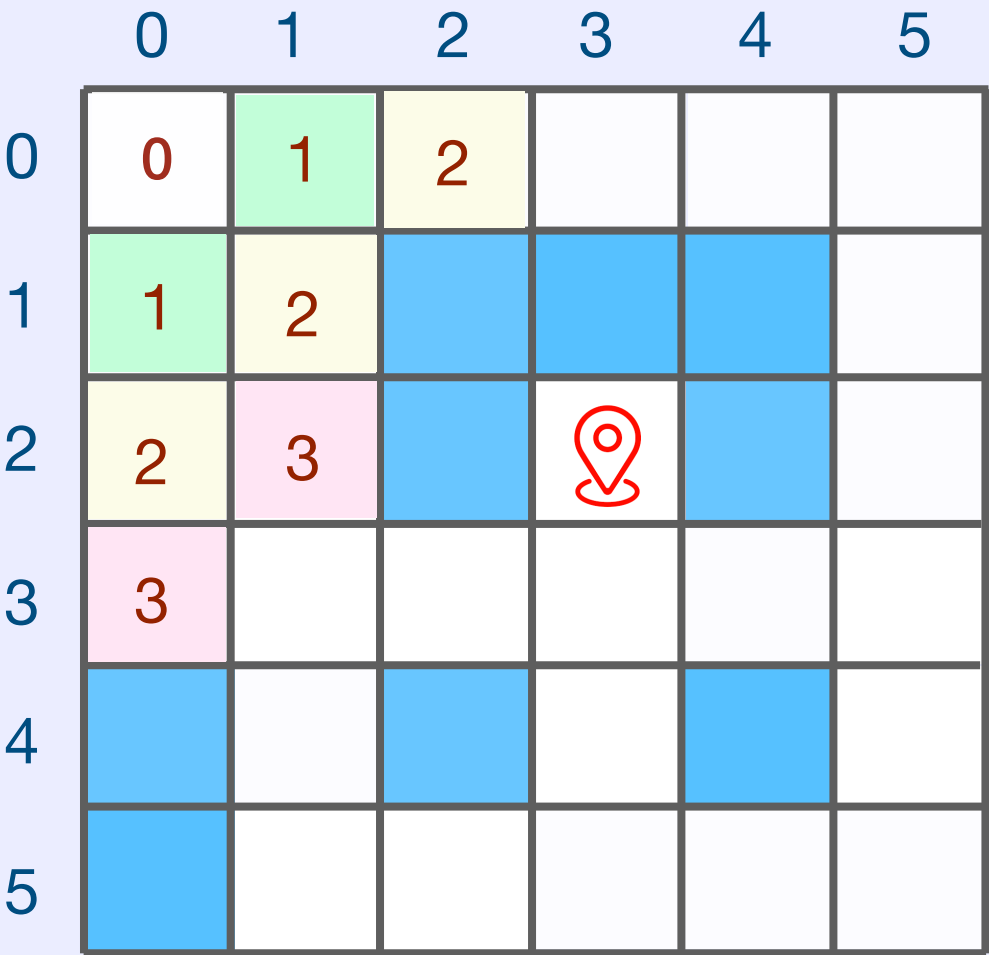
2d-array Distance



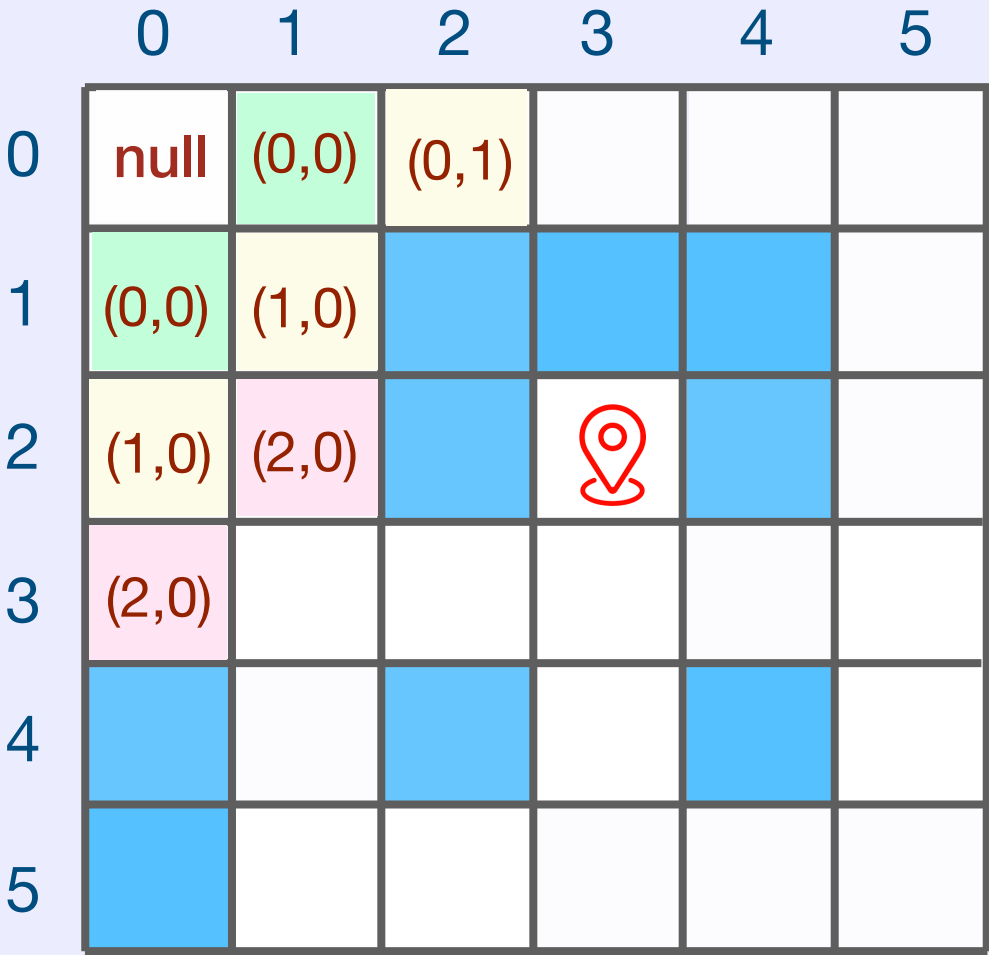
2d-array Parent



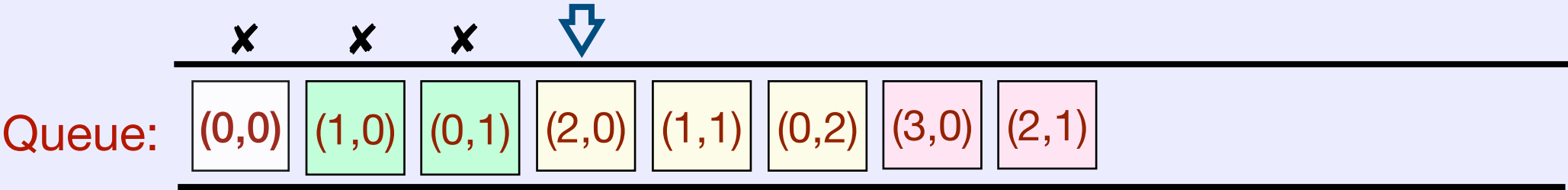
Example



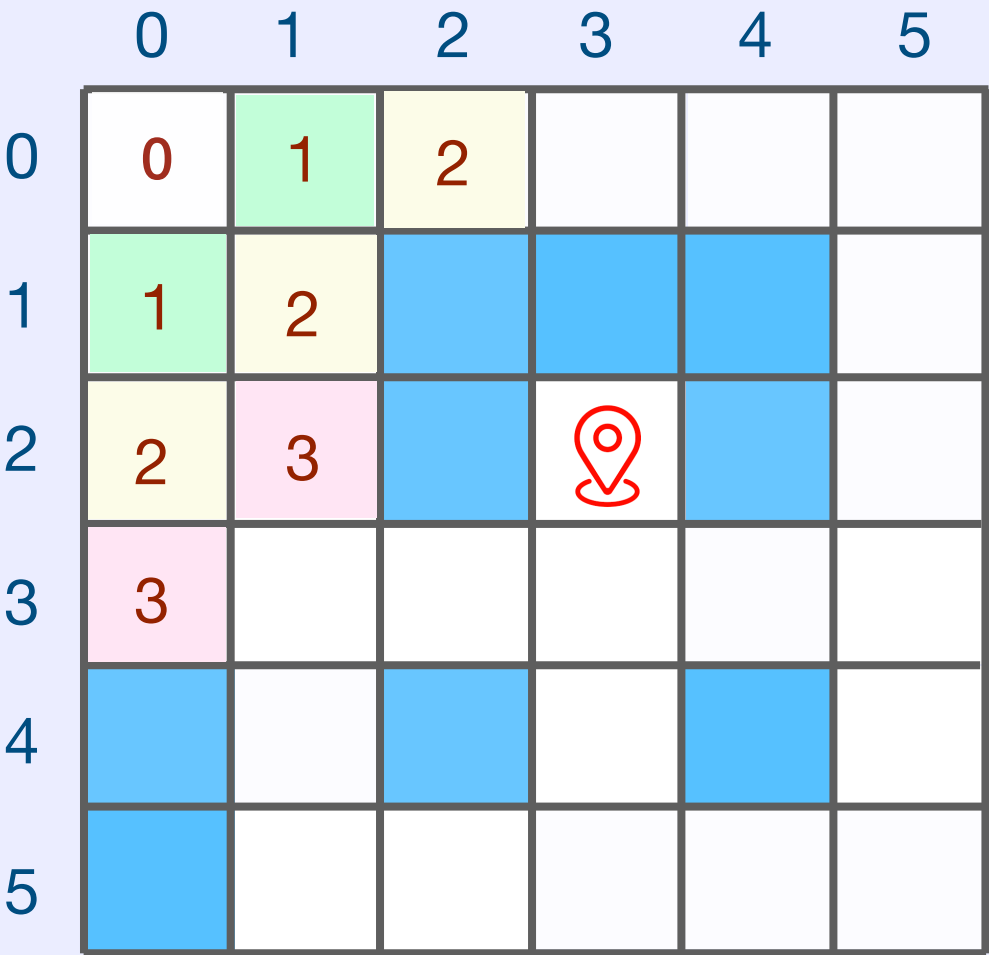
2d-array Distance



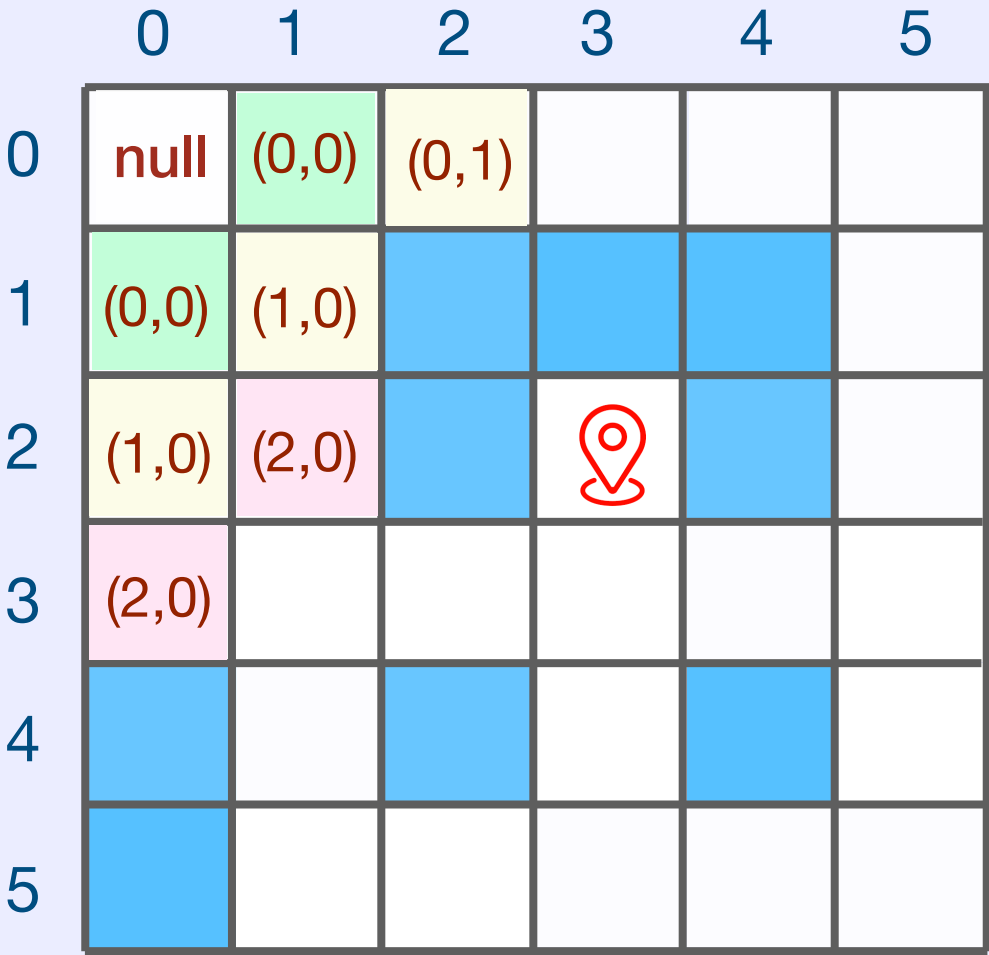
2d-array Parent



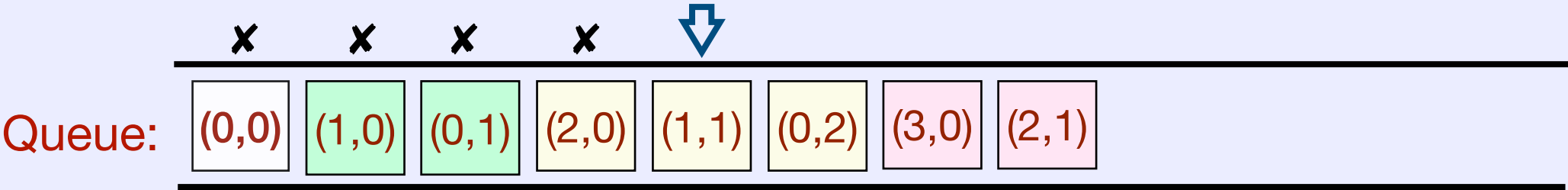
Example



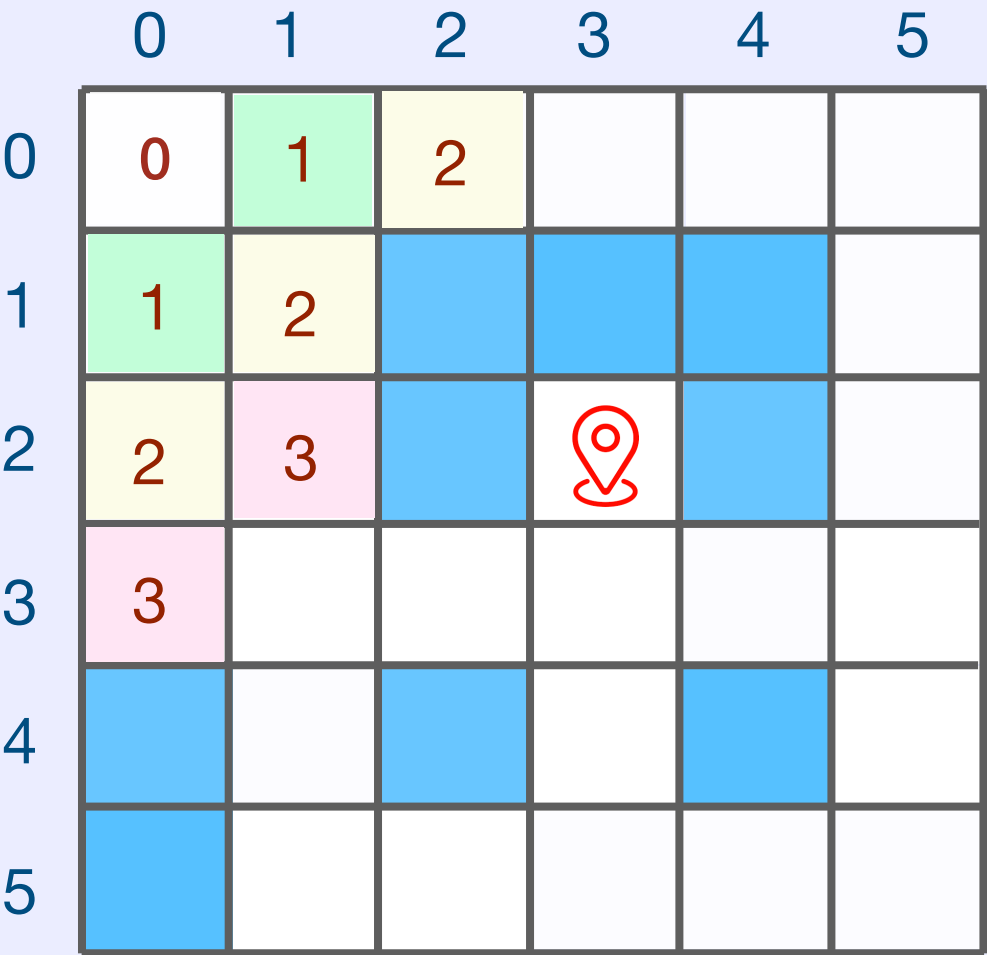
2d-array Distance



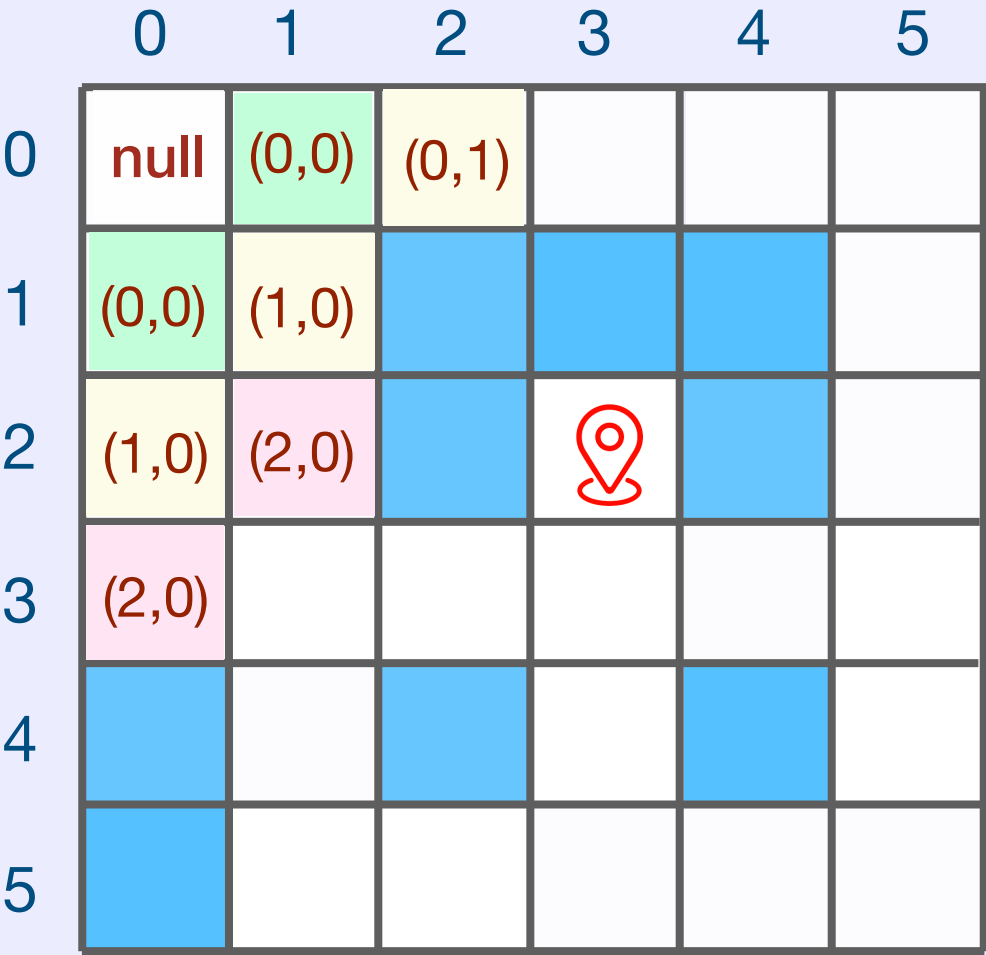
2d-array Parent



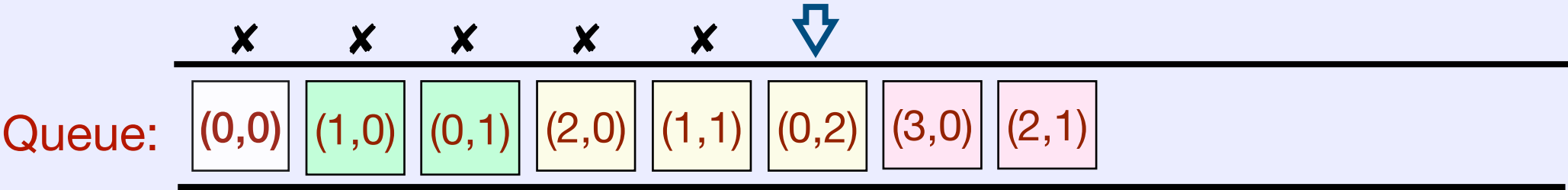
Example



2d-array Distance



2d-array Parent



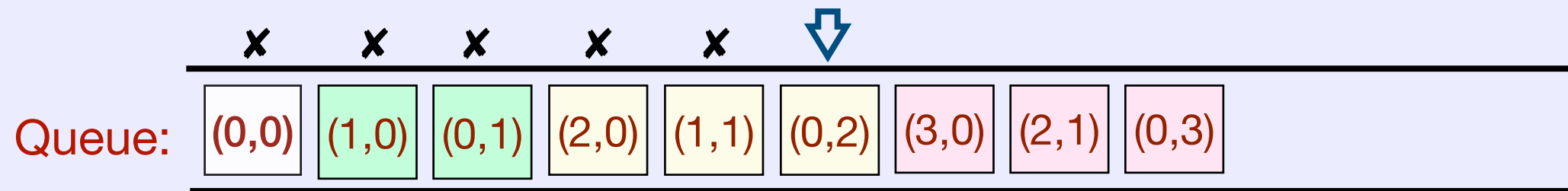
Example

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

2d-array Distance

	0	1	2	3	4	5
0	null	(0,0)	(0,1)			
1	(0,0)	(1,0)				
2	(1,0)	(2,0)				
3	(2,0)					
4						
5						

2d-array Parent



Example

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

2d-array Distance

	0	1	2	3	4	5
0	null	(0,0)	(0,1)	(0,2)		
1	(0,0)	(1,0)				
2	(1,0)	(2,0)				
3	(2,0)					
4						
5						

2d-array Parent

×

×

×

×

×



Queue:

(0,0)

(1,0)

(0,1)

(2,0)

(1,1)

(0,2)

(3,0)

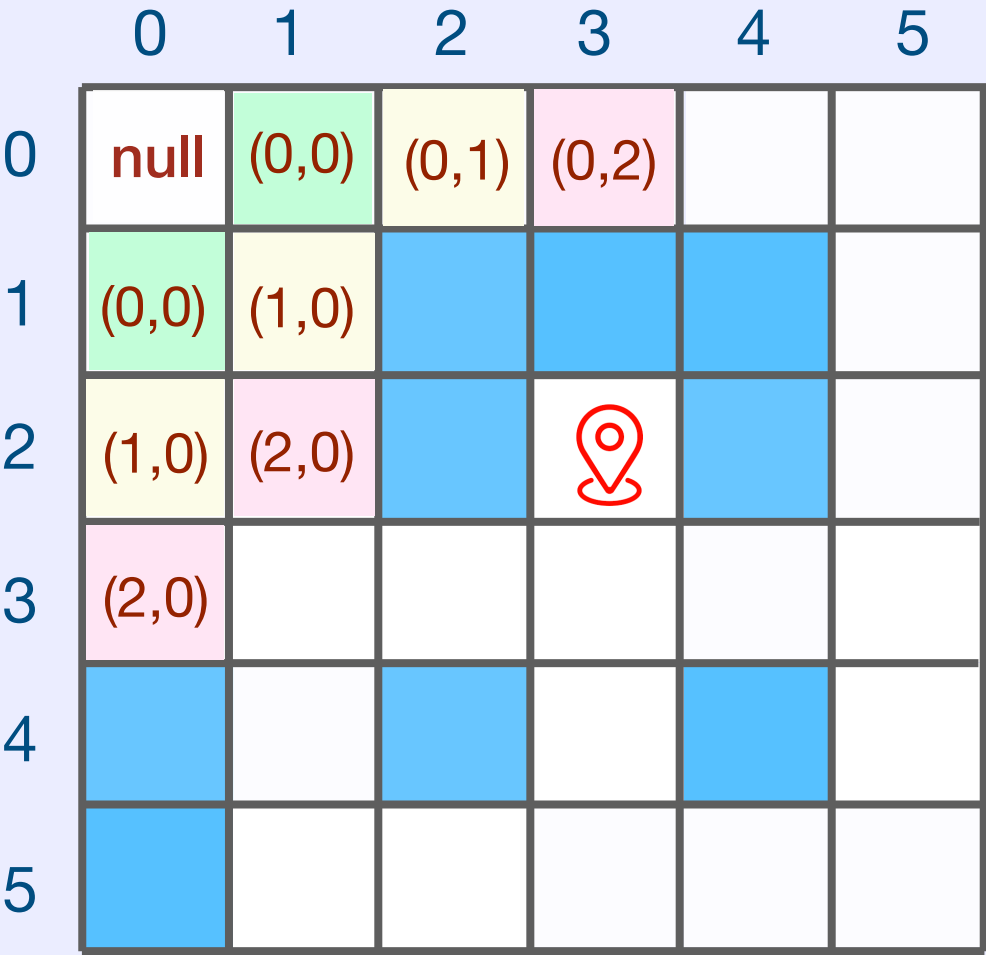
(2,1)

(0,3)

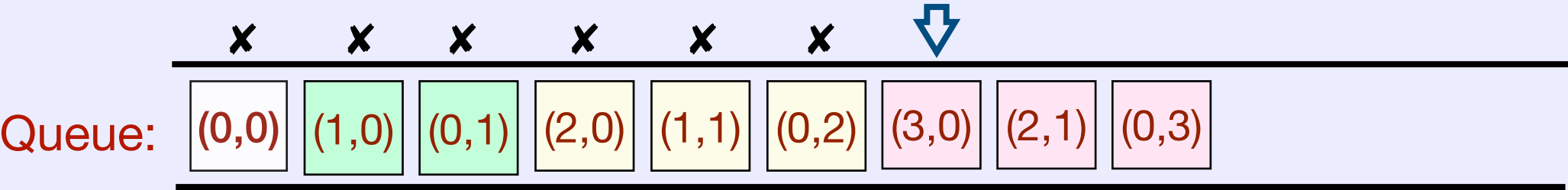
Example



2d-array Distance



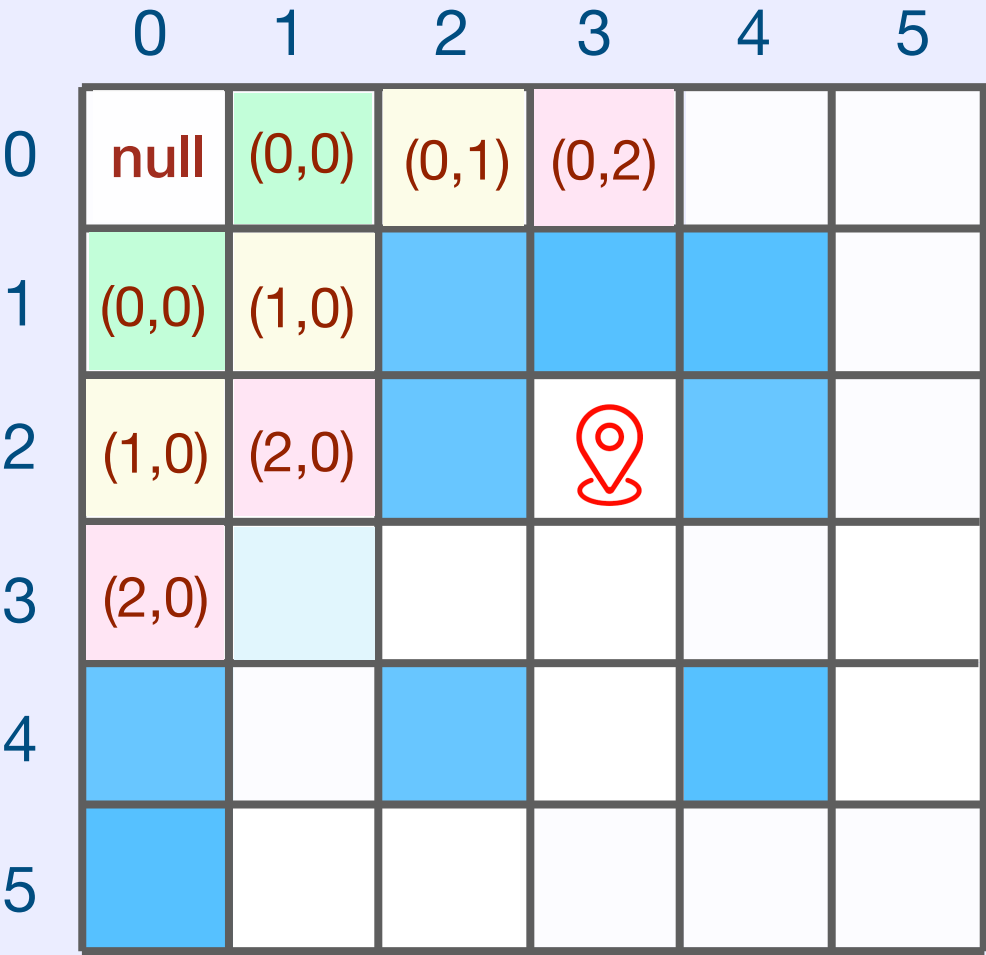
2d-array Parent



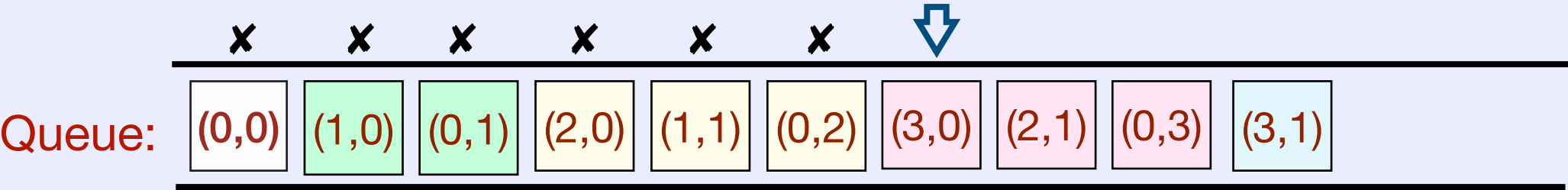
Example



2d-array Distance



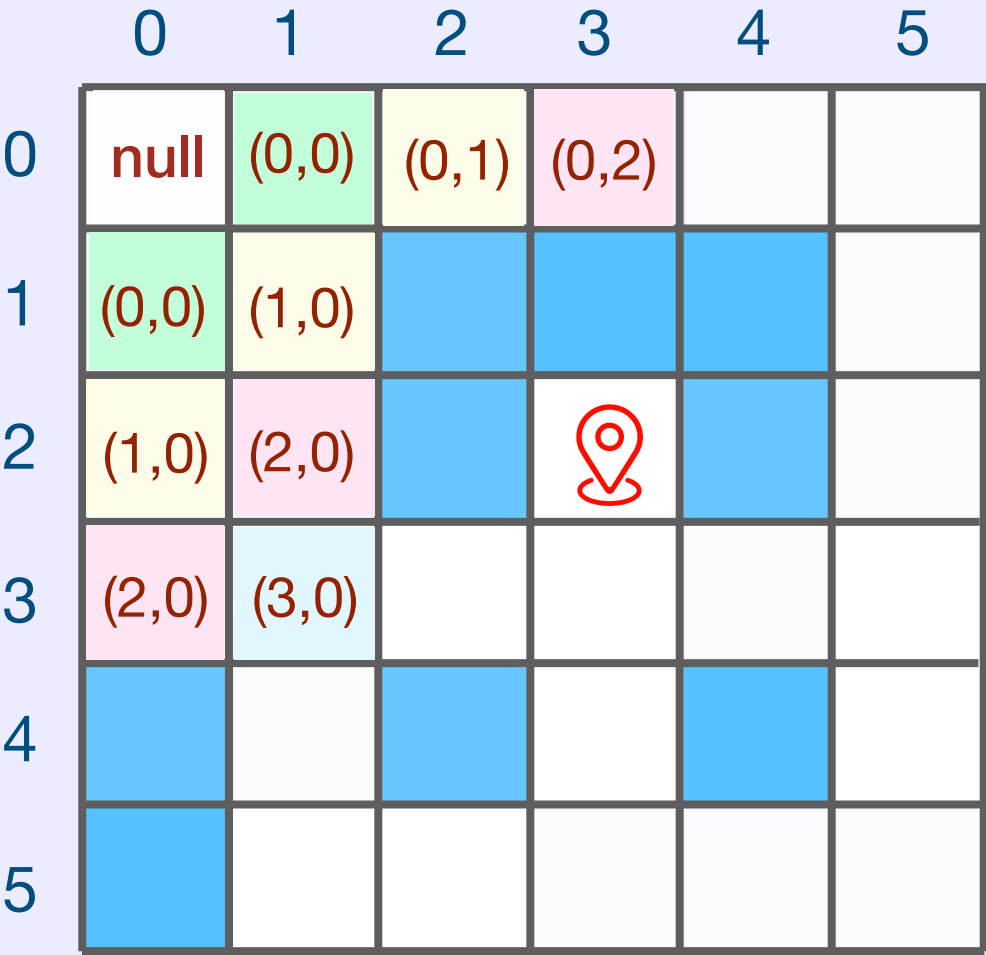
2d-array Parent



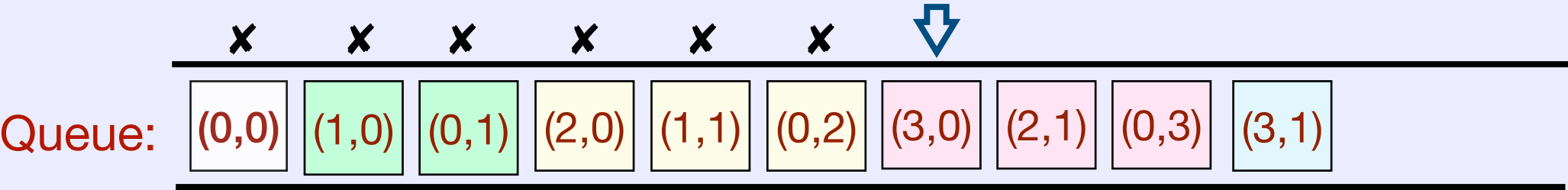
Example



2d-array Distance



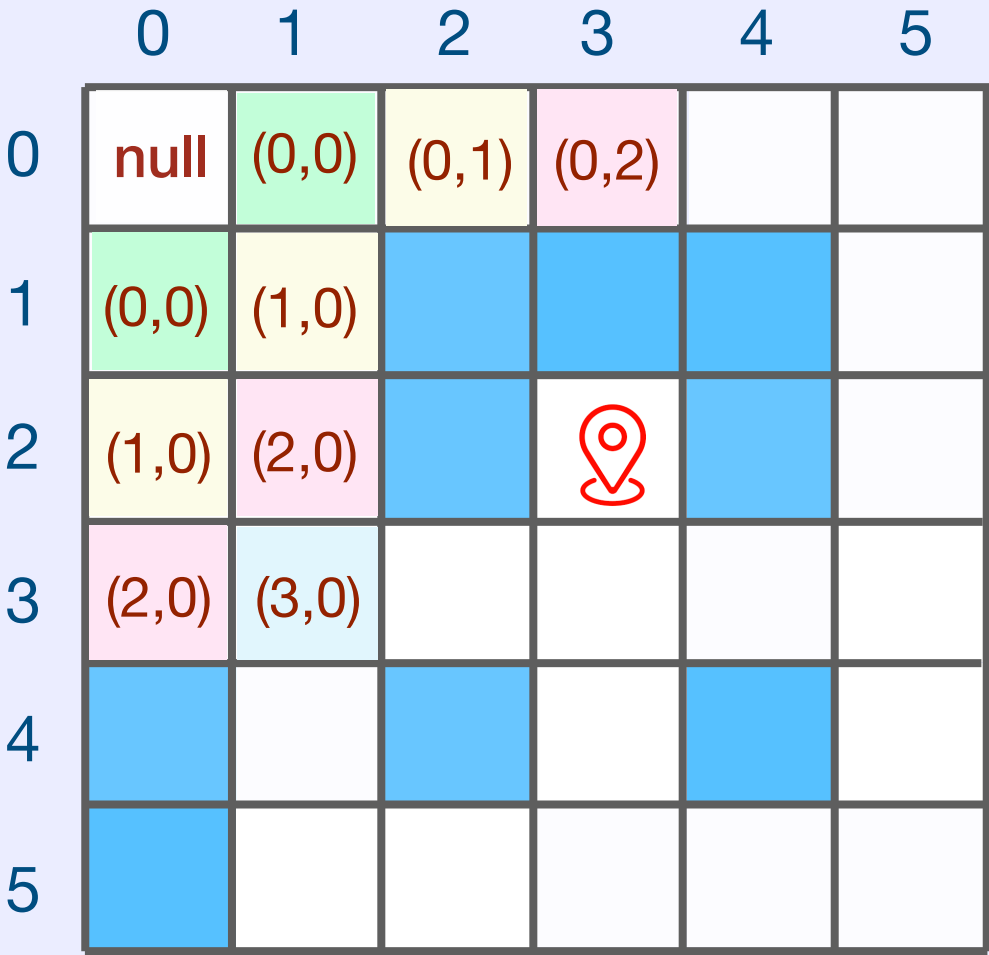
2d-array Parent



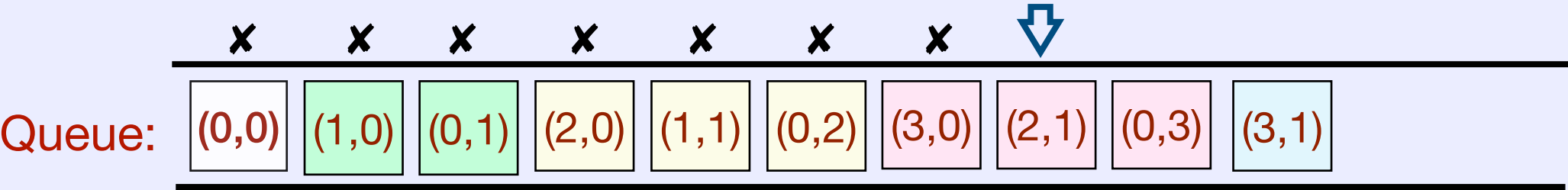
Example



2d-array Distance



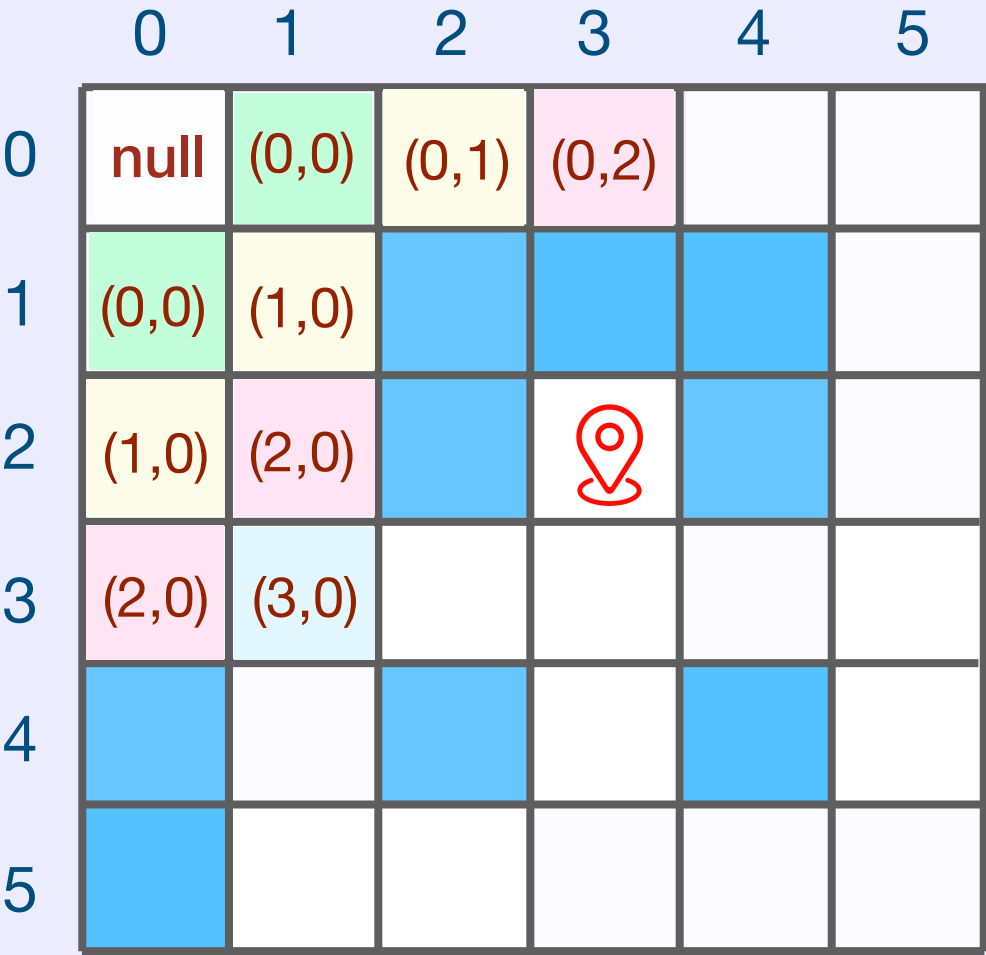
2d-array Parent



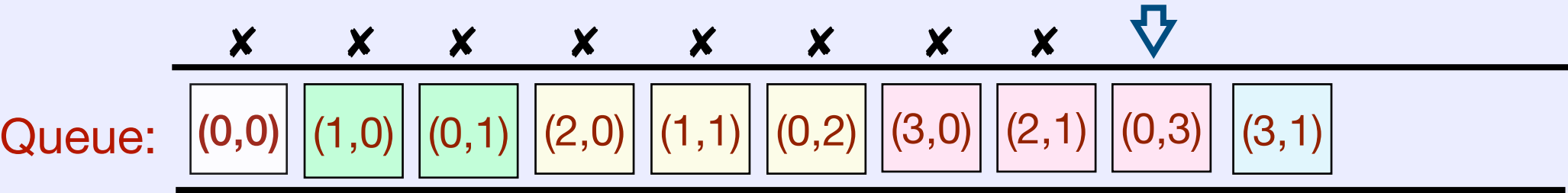
Example



2d-array Distance



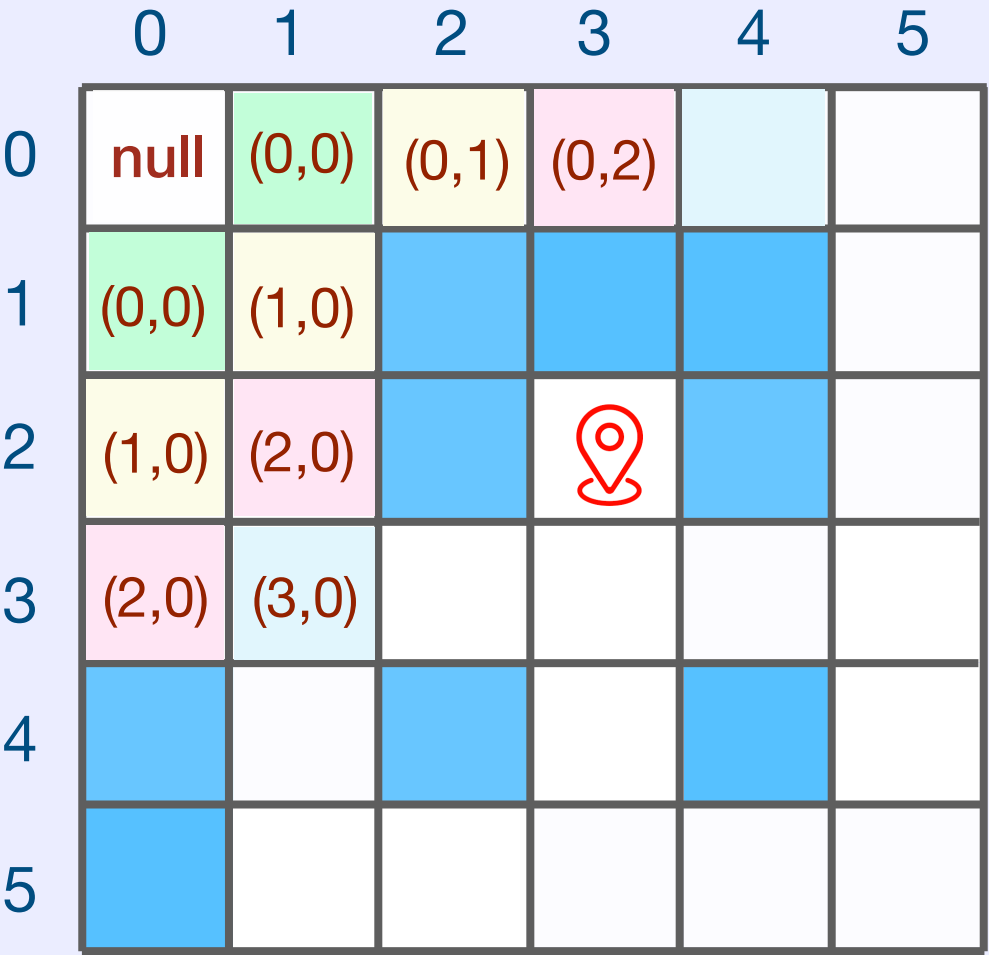
2d-array Parent



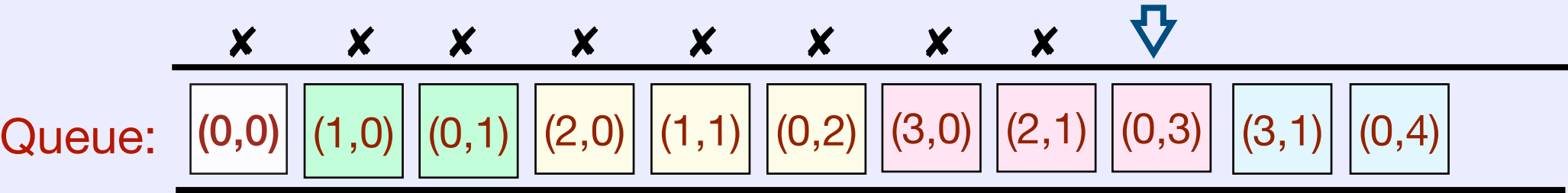
Example



2d-array Distance



2d-array Parent



Example

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

2d-array Distance

	0	1	2	3	4	5
0	null	(0,0)	(0,1)	(0,2)	(0,3)	
1	(0,0)	(1,0)				
2	(1,0)	(2,0)				
3	(2,0)	(3,0)				
4						
5						

2d-array Parent

Queue:

x

x

x

x

x

x

x

x



(0,0)

(1,0)

(0,1)

(2,0)

(1,1)

(0,2)

(3,0)

(2,1)

(0,3)

(3,1)

(0,4)

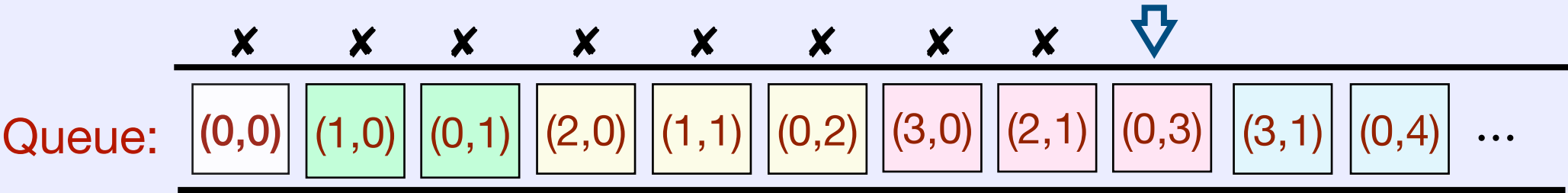
Example

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

2d-array Distance

	0	1	2	3	4	5
0	null	(0,0)	(0,1)	(0,2)	(0,3)	
1	(0,0)	(1,0)				
2	(1,0)	(2,0)				
3	(2,0)	(3,0)				
4						
5						

2d-array Parent



Correctness

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

Define S_i : Cells at distance exactly i from $(0,0)$.

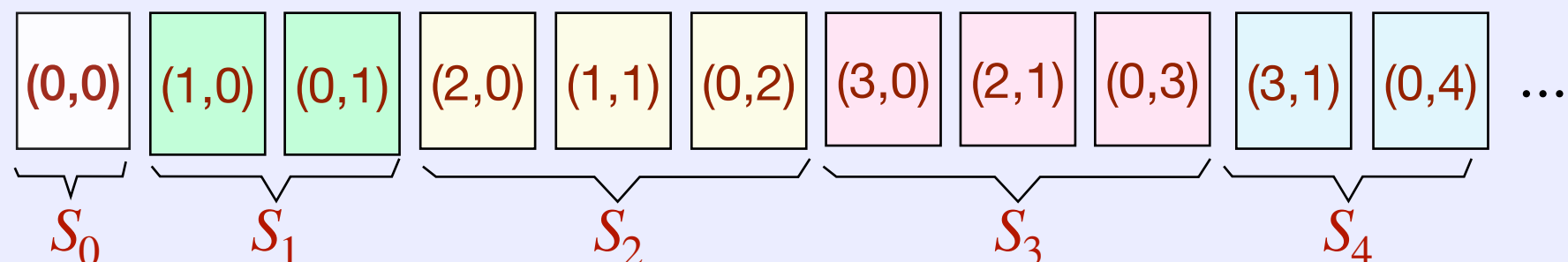
Hypothesis(i):

- (a) All elements of S_i are enqueued immediately after elements of S_{i-1} .
- (b) Distance of all cells in S_i is correctly computed.

Base case: Hypothesis(1) holds true.

Theorem: $\text{Hyp}(1) , \dots , \text{Hyp}(i) \Rightarrow \text{Hyp}(i + 1)$

Distances are assigned in order:



Implementation details

```
class Node
{
    int x;
    int y;
    Node next;
    public Node(int x, int y)
    {
        this.x = x;
        this.y = y;
        this.next = null;
    }
}
```

```
public class Queue{

    public Node head = null;
    public Node tail = null;

    public void enqueue(int x, int y)
    { }

    public Node dequeue()
    { }

    public Node front()
    { }
```

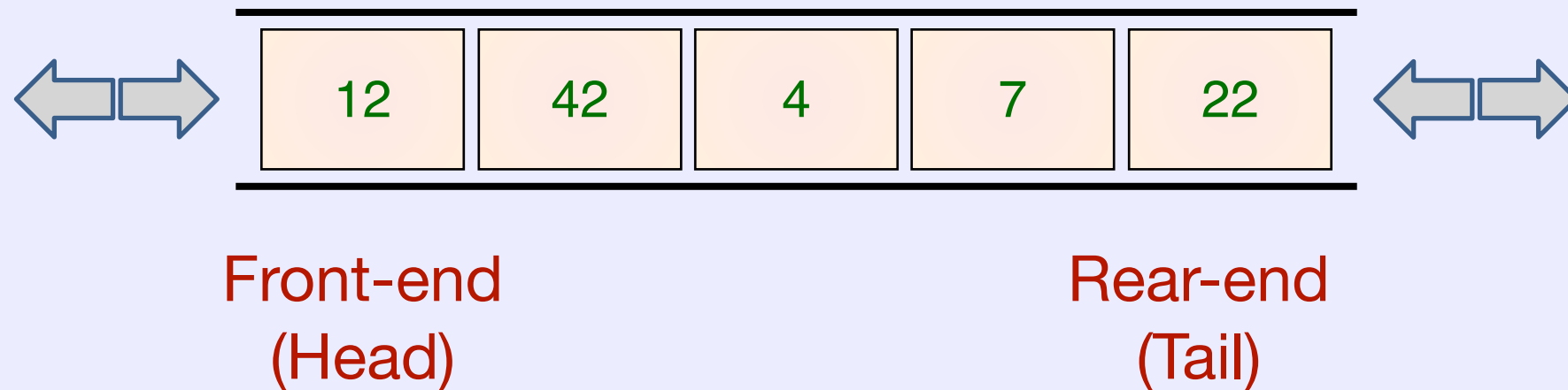
```
/* 2d-array to store distances */
int[ ][ ] distance = new int[n][n];
```

```
/* 2d-array to store parent */
Node[ ][ ] parent = new Node[n][n];
```

```
/* Queue */
Queue Q = new Queue();
```

Deque Data structure (Double Ended Queue)

Insertion and Deletion can occur at both ends.



Operations on Deque

Query Operations

- **Front(Q)**: returns the front-most element of Q.
Eg: If $Q = (a_1, a_2, \dots, a_n)$, then **Front(Q)** return a_1 .
- **End(Q)**: returns the last element of Q.
Eg: If $Q = (a_1, a_2, \dots, a_n)$, then **End(Q)** return a_n .
- **IsEmpty(Q)**: determines if Q is empty.

Update Operations

- **InsertFront(x, Q)**: Inserts x at the beginning of Q.
- **InsertEnd(x, Q)**: Inserts x at the end of Q.
- **DeleteFront(Q)**: deletes the first element of Q.
- **DeleteEnd(Q)**: deletes the last element of Q.

Homework: Obtain an implementation of Deque that performs all operations in $O(1)$ time.

Range-Minima

Design a data-structure with following functionalities:

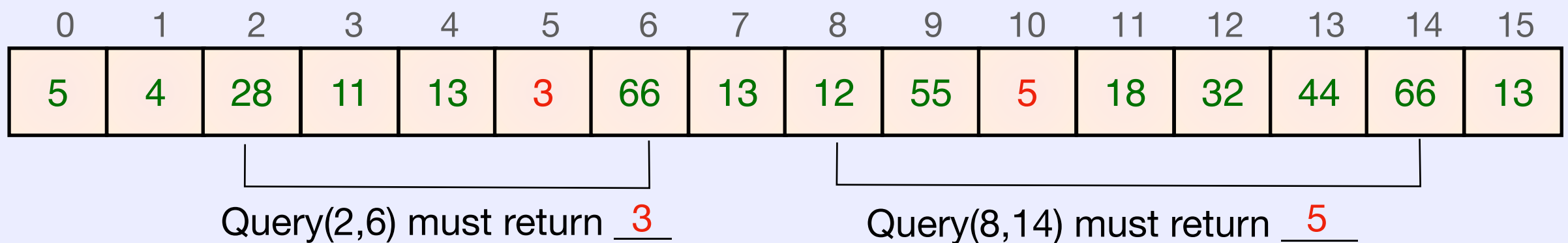
- **Initialisation**($a_0, a_1, \dots, a_{n-1}$):

Processes and stores the elements ($a_0, a_1, \dots, a_{n-1}$)

- **Query**(i, j):

Returns the minimum element from the set $\{a_i, \dots, a_j\}$ if $i \leq j$

Returns null if $i \not\leq j$



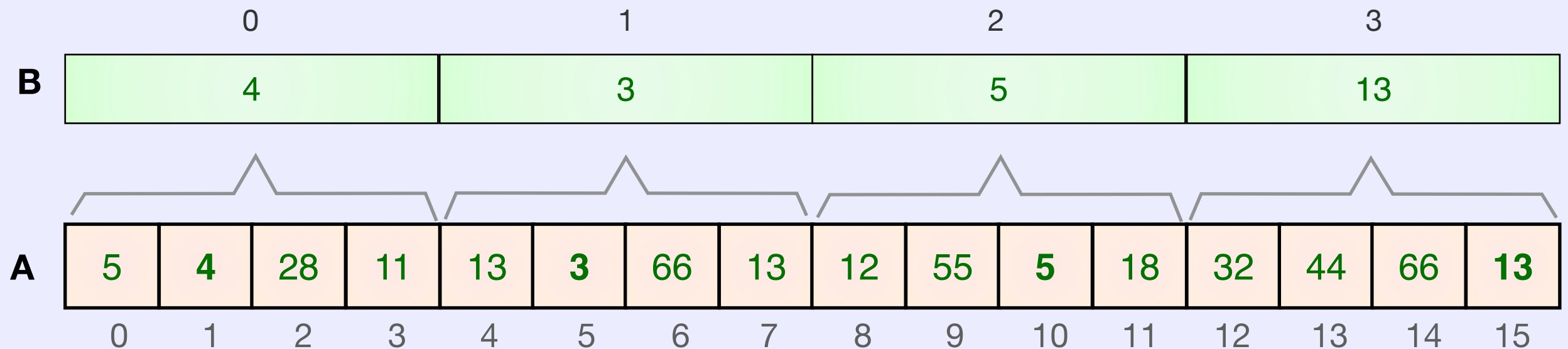
Question: Can you answer queries in $O(\sqrt{n})$ time using $O(n)$ space?

(You are allowed to spend $O(n^c)$ time in the initialisation phase, for some $c \geq 0$).

Range-Minima: $O(n)$ space and $O(n^{1/2})$ query time

Store the following extra information during initialisation phase by creating a new array **B**:

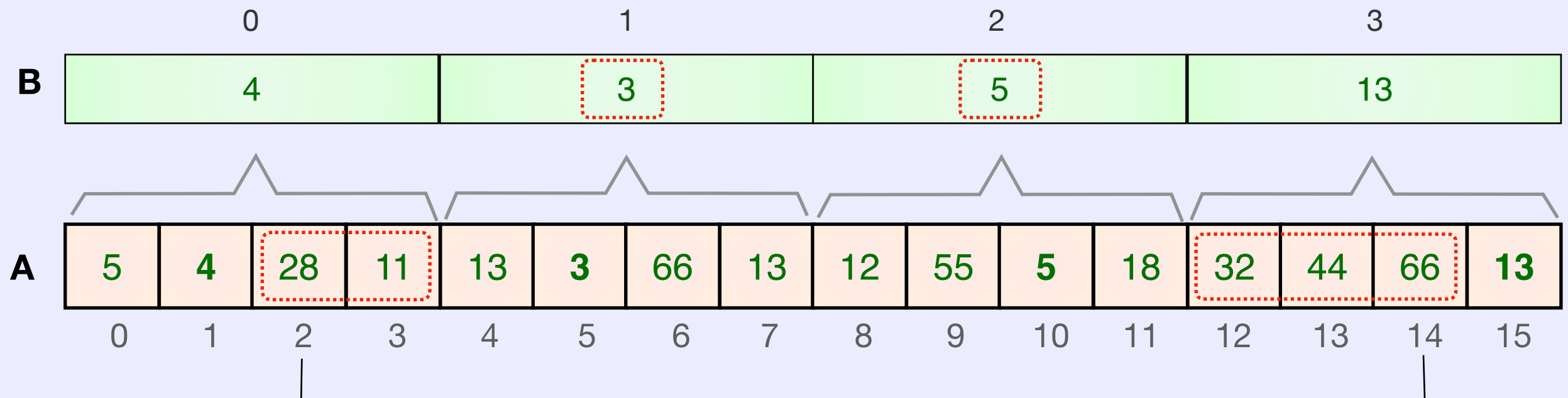
B[*i*]: Stores the minimum element from the set $\{a_{i\sqrt{n}}, \dots, a_{(i+1)\sqrt{n}-1}\}$.



Range-Minima: $O(n)$ space and $O(n^{1/2})$ query time

Store the following extra information during initialisation phase by creating a new array **B**:

B[*i*]: Stores the minimum element from the set $\{a_{i\sqrt{n}}, \dots, a_{(i+1)\sqrt{n}-1}\}$.

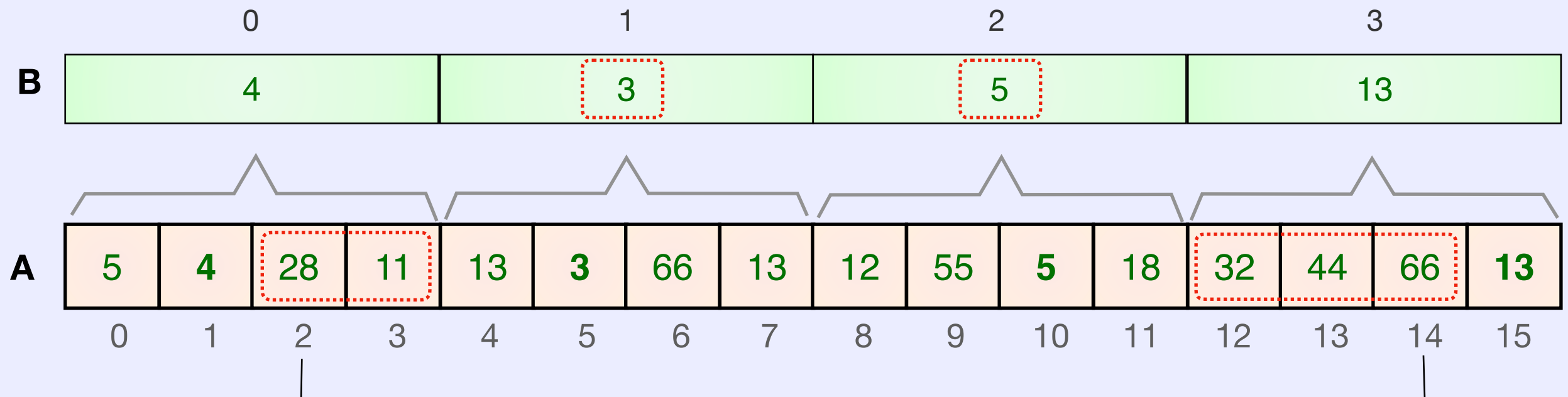


To answer Query(2,14) we need to output $\min\{28, 11, 3, 5, 32, 44, 66\}$

Range-Minima: $O(n)$ space and $O(n^{1/2})$ query time

Store the following extra information during initialisation phase by creating a new array **B**:

B[i]: Stores the minimum element from the set $\{a_{i\sqrt{n}}, \dots, a_{(i+1)\sqrt{n}-1}\}$.



To answer Query(2,14) we need to output $\min\{28, 11, 3, 5, 32, 44, 66\}$

Homework: Can you answer queries in $O(\log n)$ time using $O(n)$ space?

(You are allowed to spend $O(n^c)$ time in the initialisation phase, for some $c \geq 0$).