

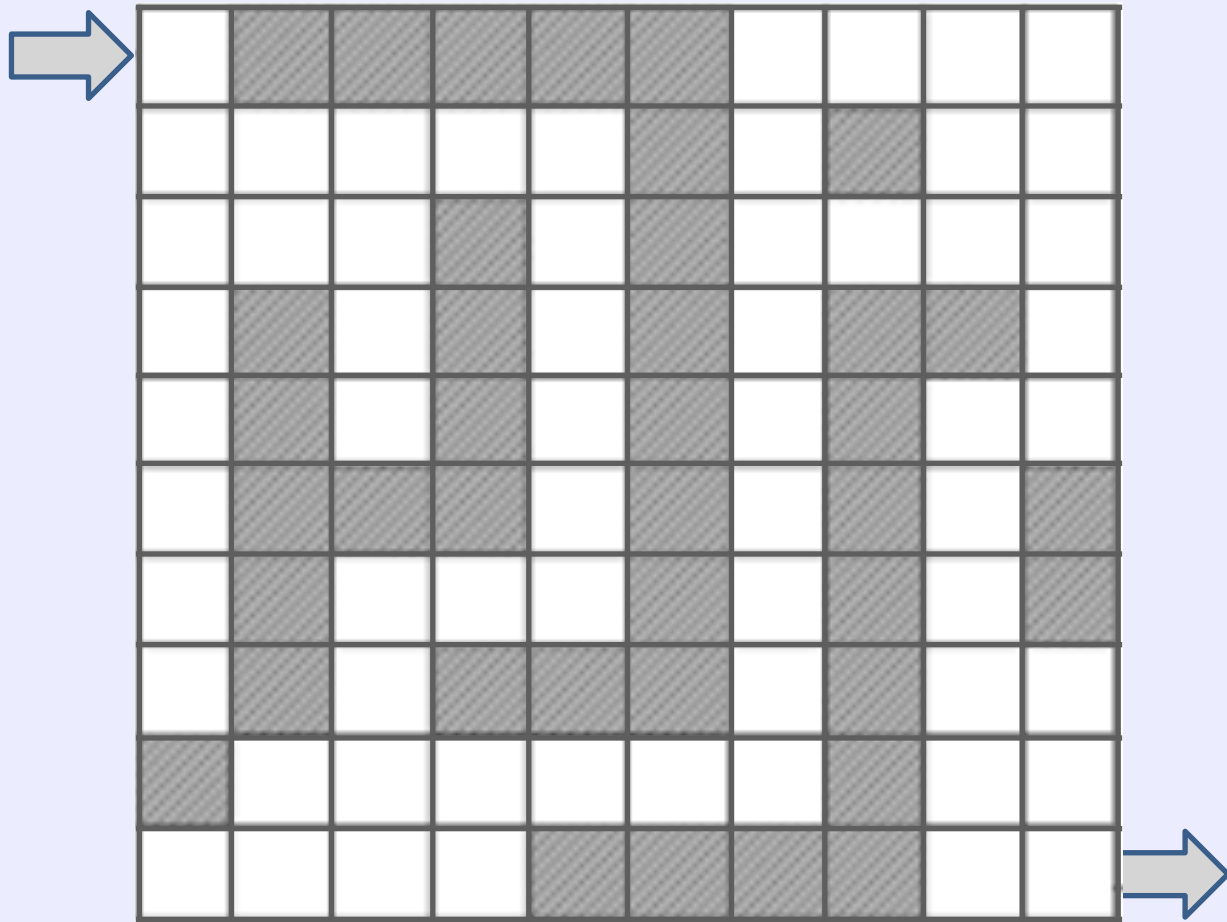
COL106: Data Structures and Algorithms

Lecture 4

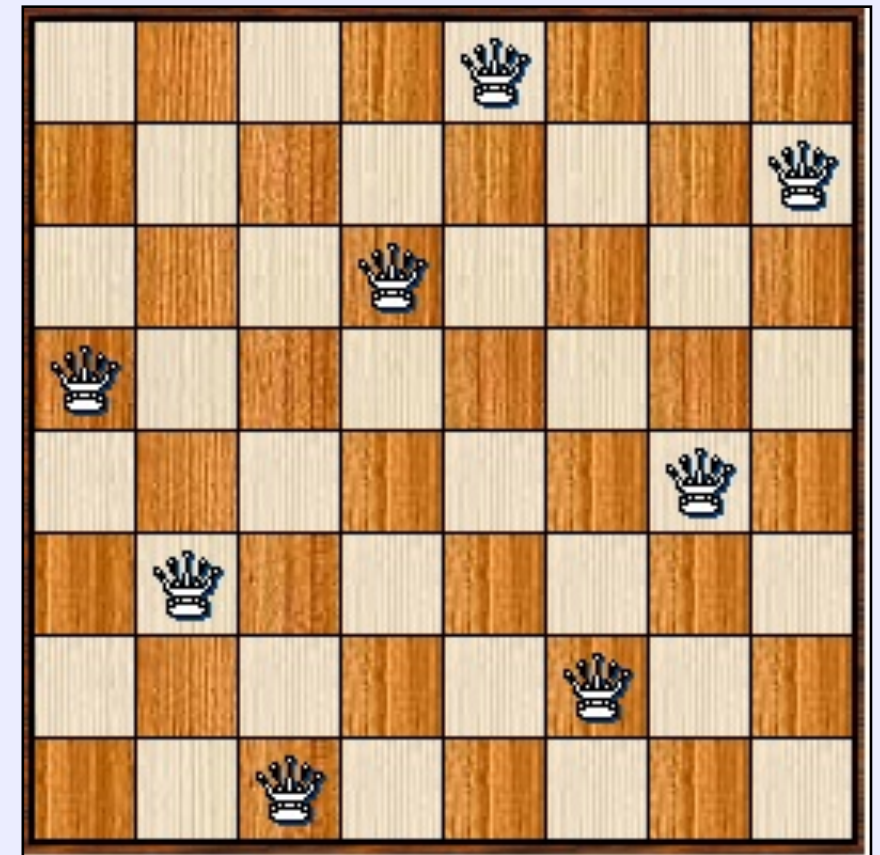
Keerti Choudhary
Department of Computer Science, IIT Delhi

Stack Continued..

A powerful data structure with applications in:



Maze solving



Solving 8x8 Queen Puzzle
(Place 8 queens so that no two
of them attach each other)

$$x = 3 + 6 * (4 - 8 * (9 + 7 * 6) + 2)$$

Expression evaluation

Operations on Stack

Query Operations

- **Top(S)**: returns the element at top of stack S.
Example: If $S = (a_1, a_2, \dots, a_n)$, then **Top(S)** return a_1 .
- **IsEmpty(S)**: determine if S is an empty stack.

Update Operations

- **Push(x, S)**: insert x at top of stack S.
Example: If $S = (a_1, \dots, a_n)$, then after **Push(x, S)** stack is $(x, a_1, \dots, a_n)$.
- **Pop(S)**: delete top element from stack S.
Example: If $S = (a_1, a_2, \dots, a_n)$, then after **Pop(S)** stack is $(a_2, \dots, a_n)$.

Expression Evaluation using Stack

Aim: To solve mathematical expression having following operators : + , - , * , / , ^

Example: $28 - 5 * 6 ^ 2 / 9 + 4$

Expression $(28 - (5 * (6 ^ 2)) / 9) + 4$ evaluates to 12.

Precedence Order:

1. ^ (power)
2. * and / (multiplication and division)
3. + and - (addition and subtraction)

Left associativity for + , - , * , / : $(25 - 12 - 4) = ((25 - 12) - 4) = 9$.

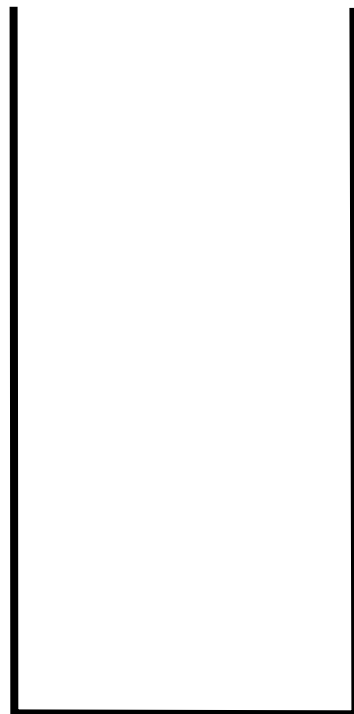
Right associativity for ^ : $(2 ^ 2 ^ 3)$ is $(2 ^ (2 ^ 3)) = 256$ and not 64.

Expression Evaluation using Stack

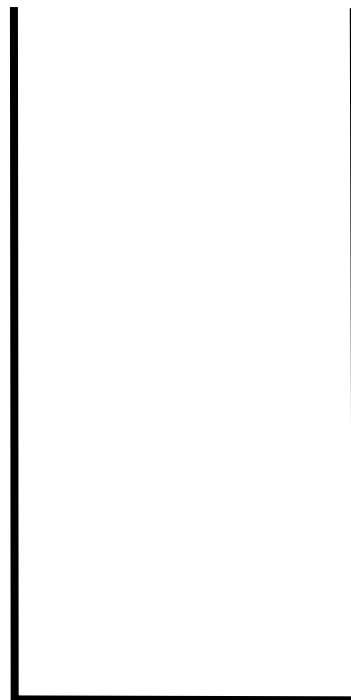
Example: $28 - 5 * 6^2 / 9 + 4$

Expression Evaluation using Stack

Example: $28 - 5 * 6 ^ 2 / 9 + 4$



Numbers

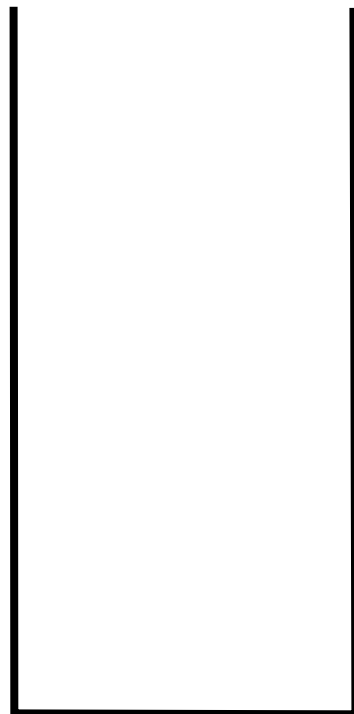


Operations

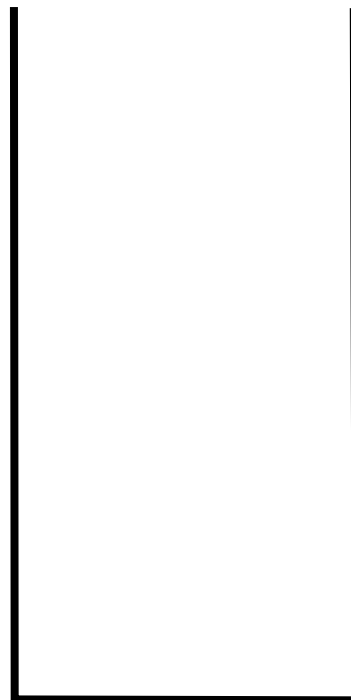
(in increasing order
of precedence)

Expression Evaluation using Stack

Example: 28 - 5 * 6 ^ 2 / 9 + 4



Numbers

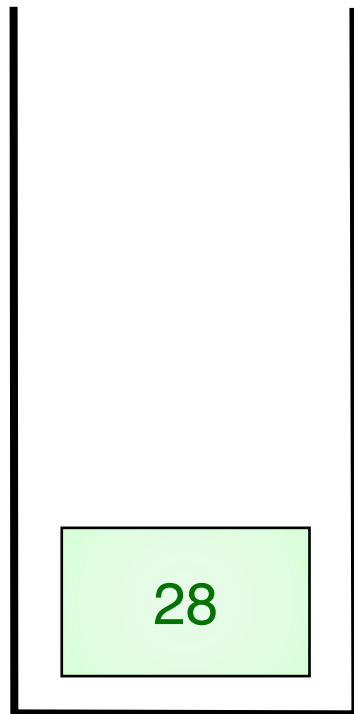


Operations

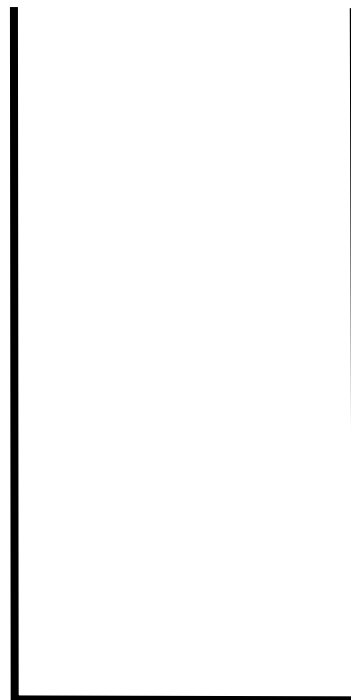
(in increasing order
of precedence)

Expression Evaluation using Stack

Example: 28 - 5 * 6 ^ 2 / 9 + 4



Numbers

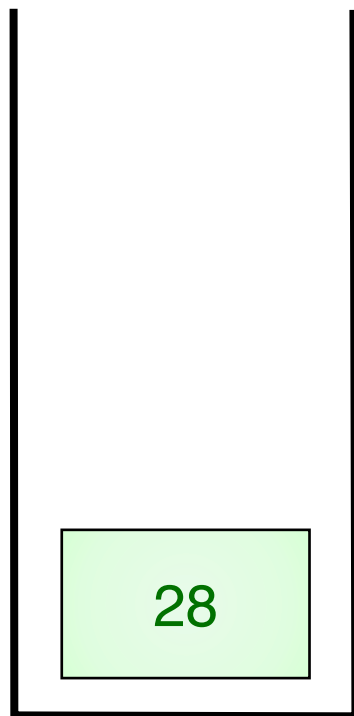


Operations

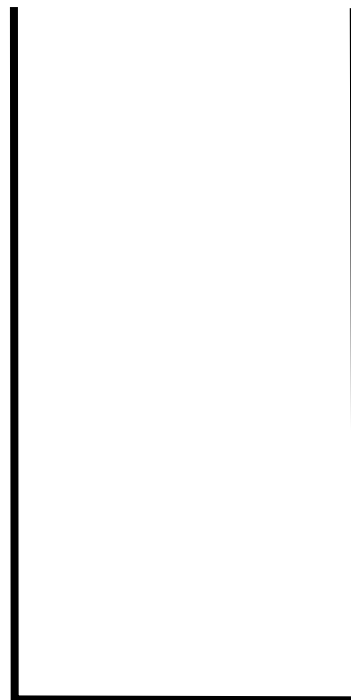
(in increasing order
of precedence)

Expression Evaluation using Stack

Example: 28 - 5 * 6 ^ 2 / 9 + 4



Numbers

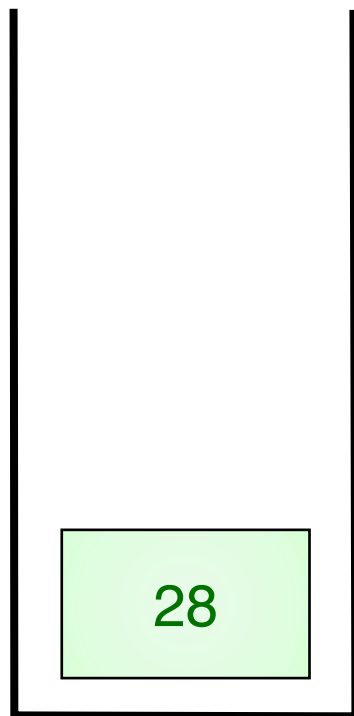


Operations

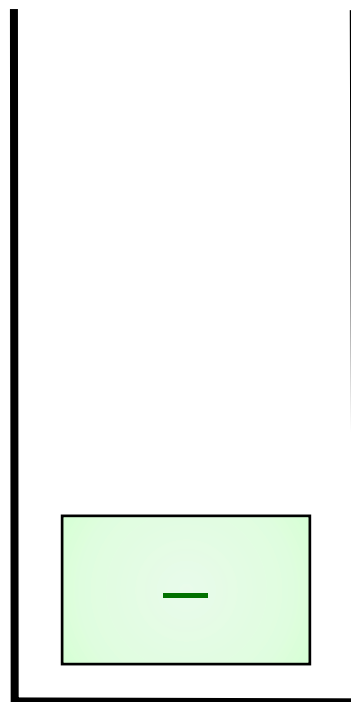
(in increasing order
of precedence)

Expression Evaluation using Stack

Example: 28 - 5 * 6 ^ 2 / 9 + 4



Numbers

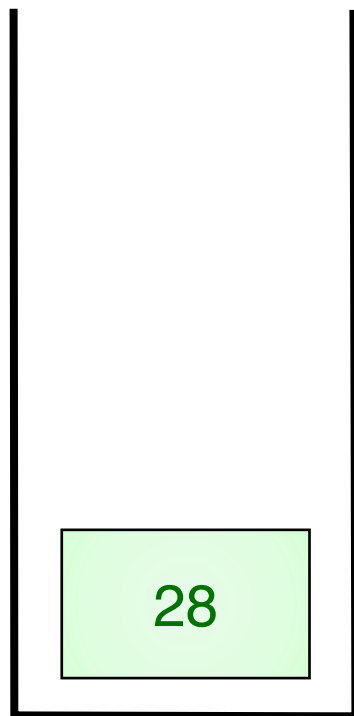


Operations

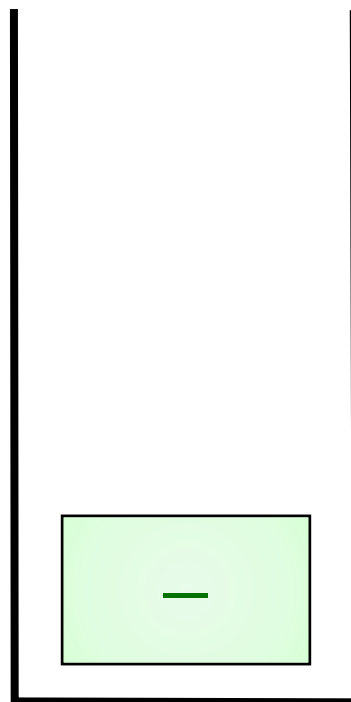
(in increasing order
of precedence)

Expression Evaluation using Stack

Example: $28 - 5 * 6 ^ 2 / 9 + 4$



Numbers

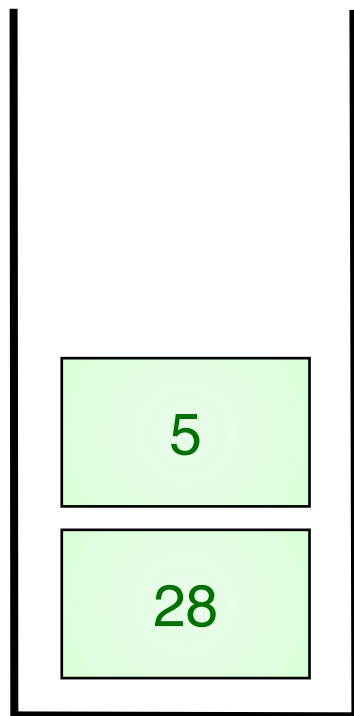


Operations

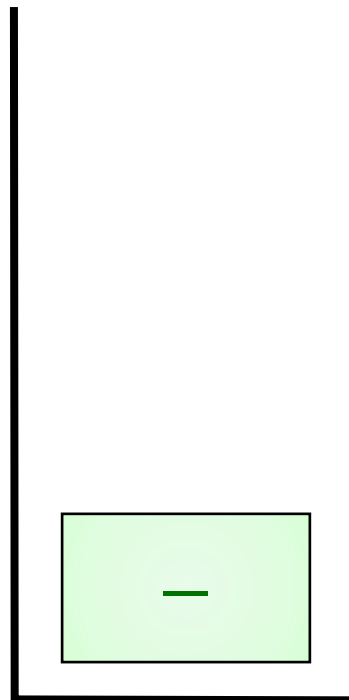
(in increasing order
of precedence)

Expression Evaluation using Stack

Example: $28 - 5 * 6 ^ 2 / 9 + 4$



Numbers

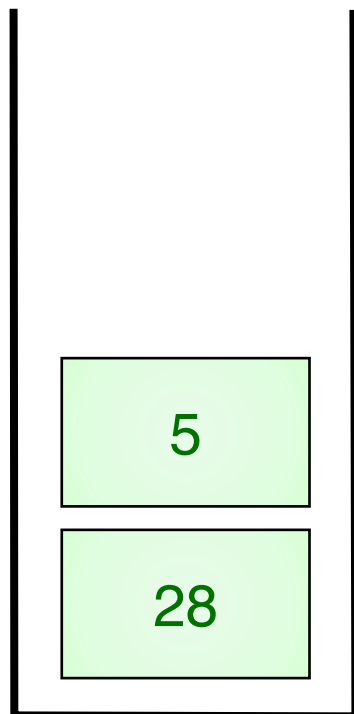


Operations

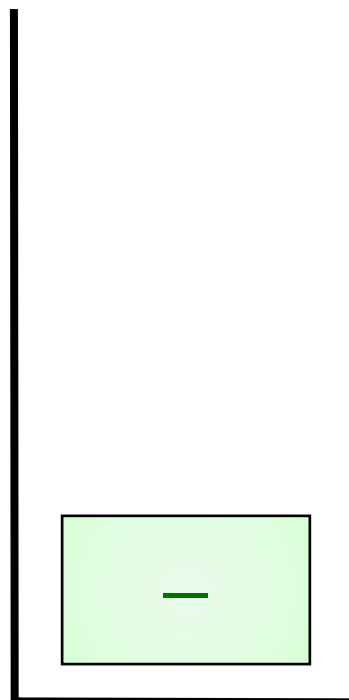
(in increasing order
of precedence)

Expression Evaluation using Stack

Example: $28 - 5 * 6 ^ 2 / 9 + 4$



Numbers

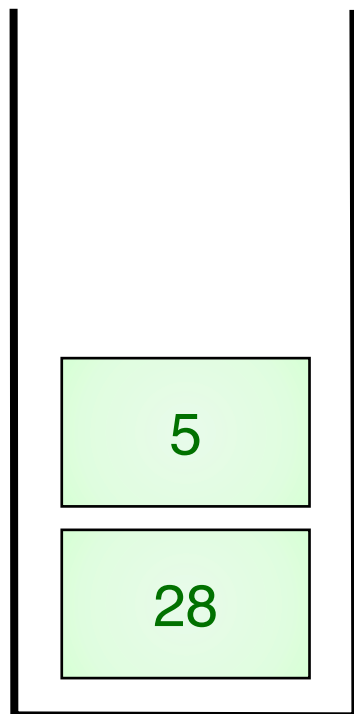


Operations

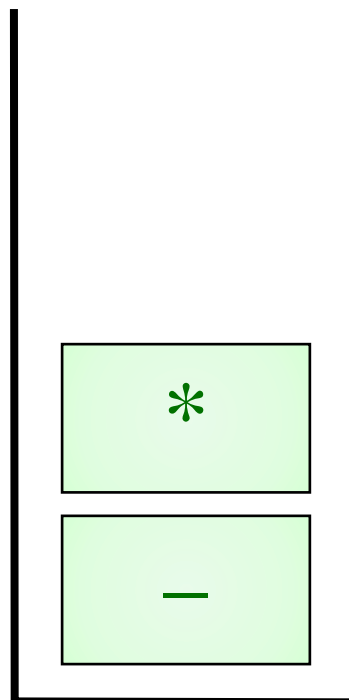
(in increasing order
of precedence)

Expression Evaluation using Stack

Example: $28 - 5 * 6 ^ 2 / 9 + 4$



Numbers

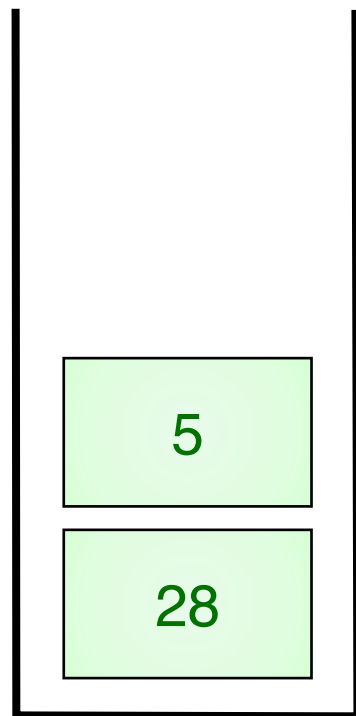


Operations

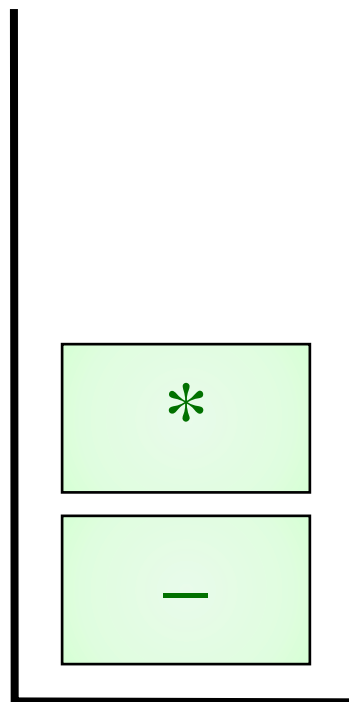
(in increasing order
of precedence)

Expression Evaluation using Stack

Example: $28 - 5 * 6 ^ 2 / 9 + 4$



Numbers



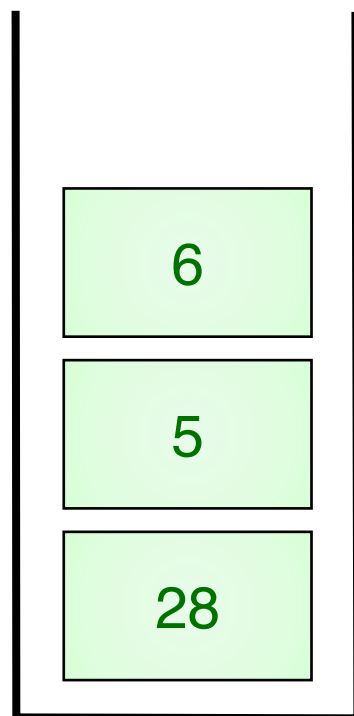
Operations

(in increasing order
of precedence)

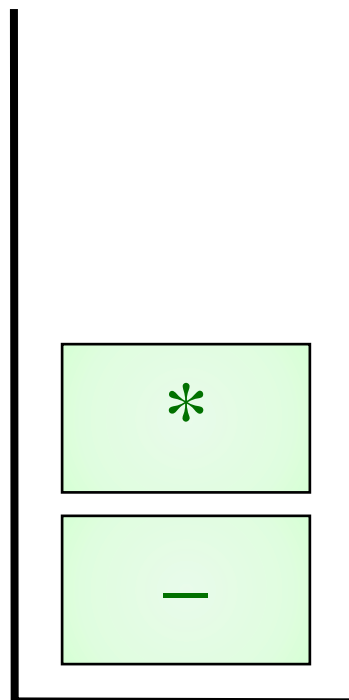
Expression Evaluation using Stack

Example: $28 - 5 * 6^2 / 9 + 4$

↑



Numbers

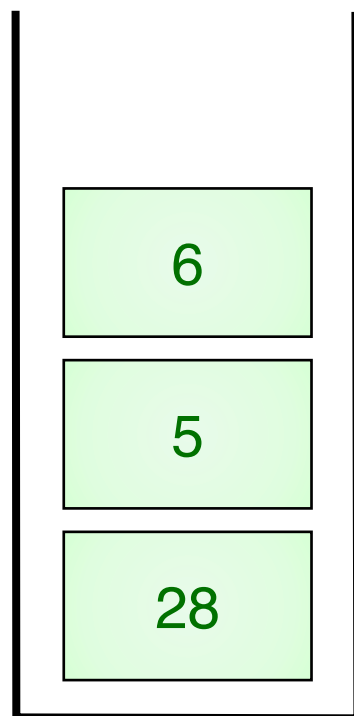


Operations

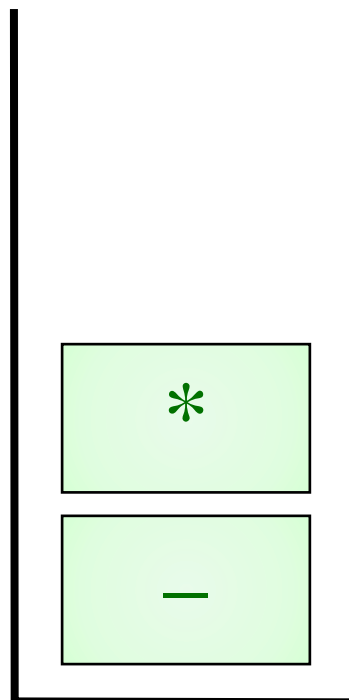
(in increasing order
of precedence)

Expression Evaluation using Stack

Example: $28 - 5 * 6 ^ 2 / 9 + 4$



Numbers



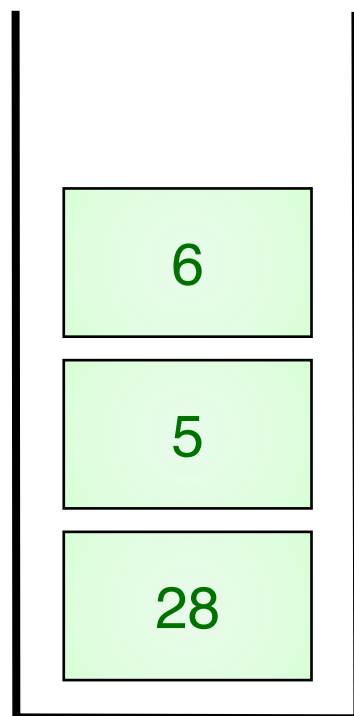
Operations

(in increasing order
of precedence)

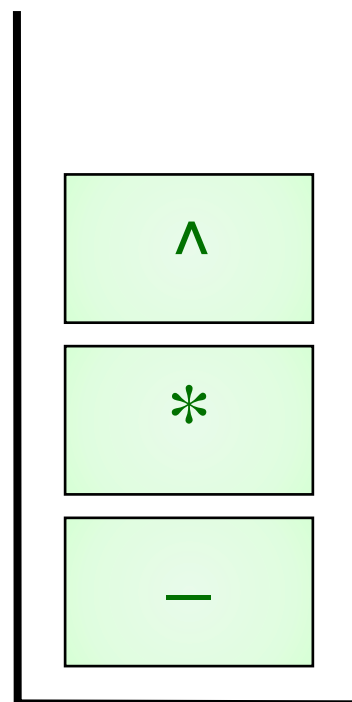
Expression Evaluation using Stack

Example: $28 - 5 * 6 ^ 2 / 9 + 4$

↑



Numbers



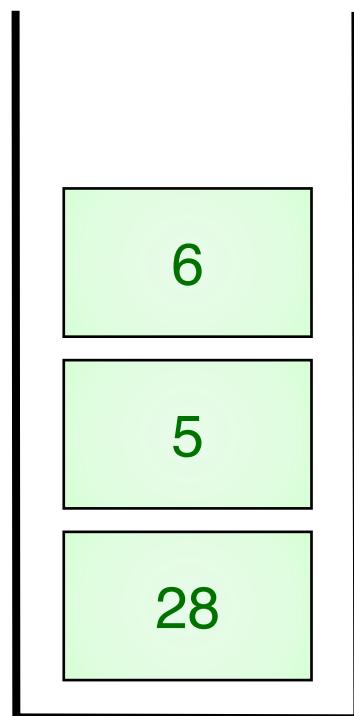
Operations

(in increasing order
of precedence)

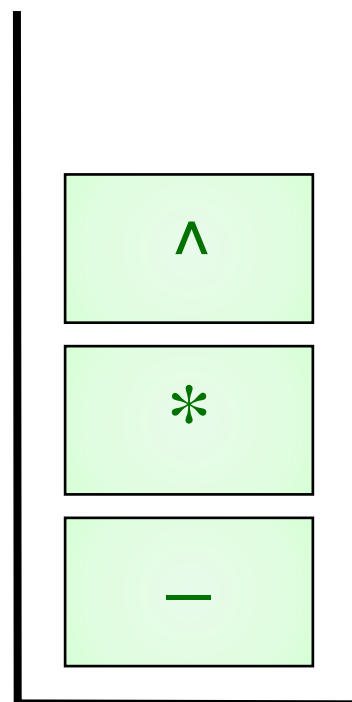
Expression Evaluation using Stack

Example: $28 - 5 * 6 ^ 2 / 9 + 4$

↑



Numbers

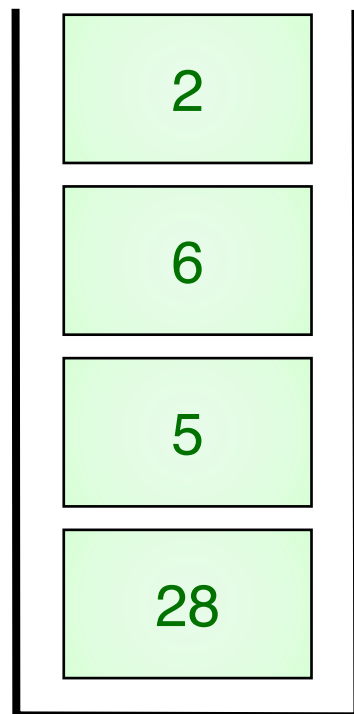


Operations

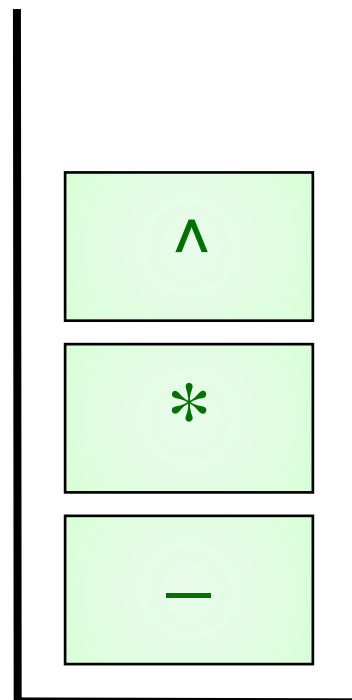
(in increasing order
of precedence)

Expression Evaluation using Stack

Example: $28 - 5 * 6 ^ 2 / 9 + 4$



Numbers

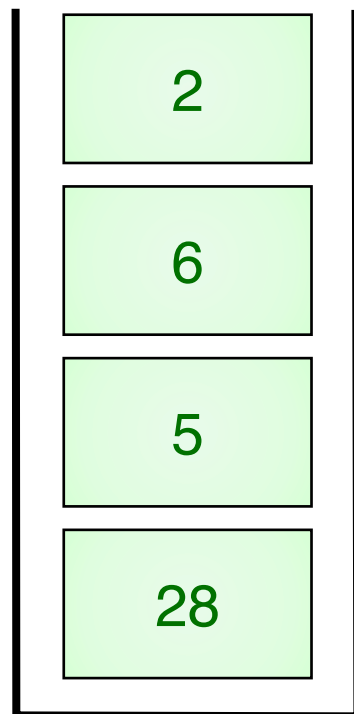


Operations

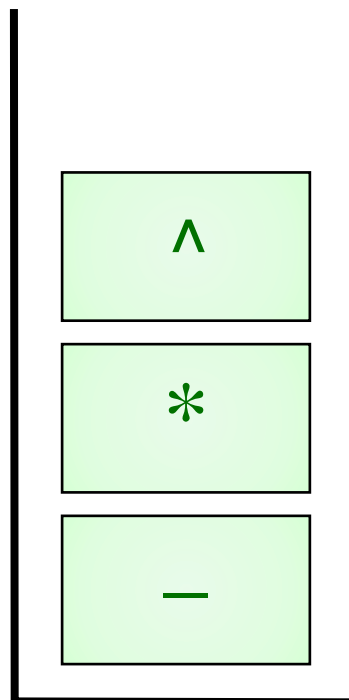
(in increasing order
of precedence)

Expression Evaluation using Stack

Example: $28 - 5 * 6 ^ 2 / 9 + 4$



Numbers

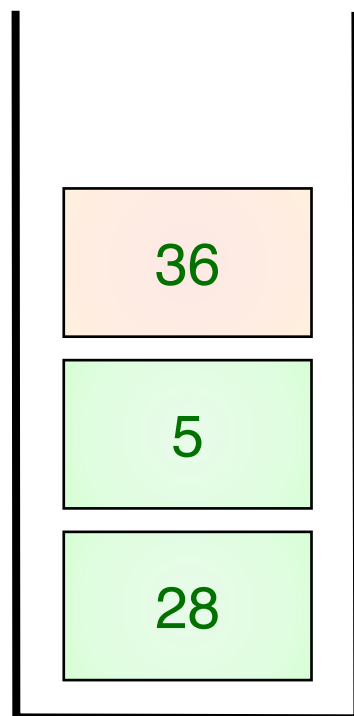


Operations

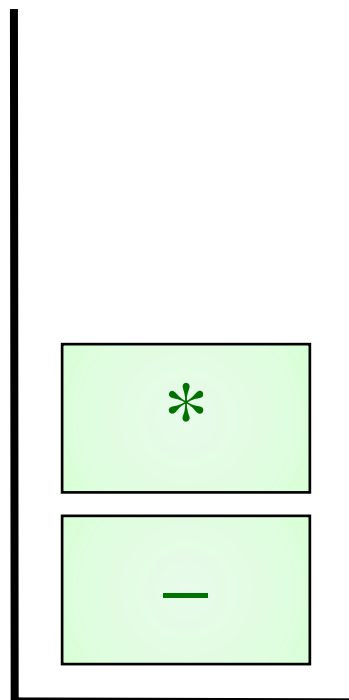
(in increasing order
of precedence)

Expression Evaluation using Stack

Example: $28 - 5 * 6 ^ 2 / 9 + 4$



Numbers

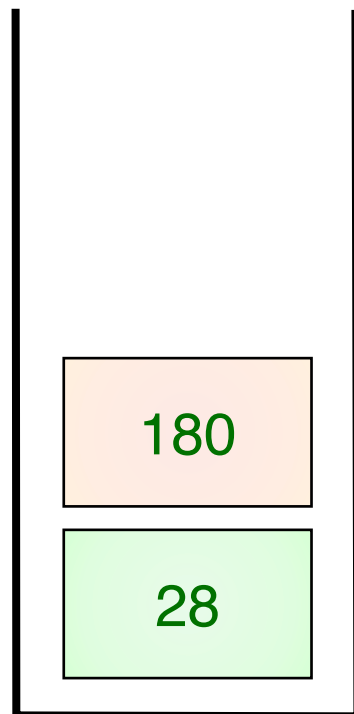


Operations

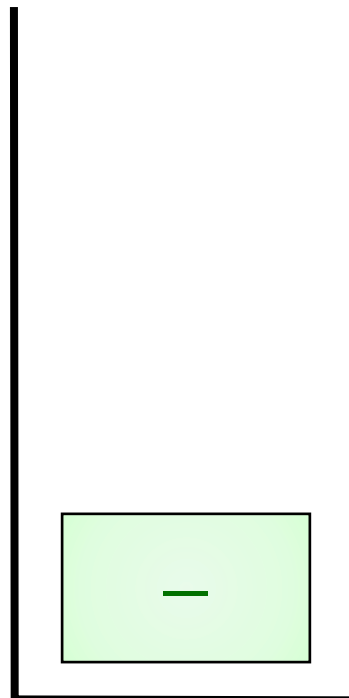
(in increasing order
of precedence)

Expression Evaluation using Stack

Example: $28 - 5 * 6 ^ 2 / 9 + 4$



Numbers

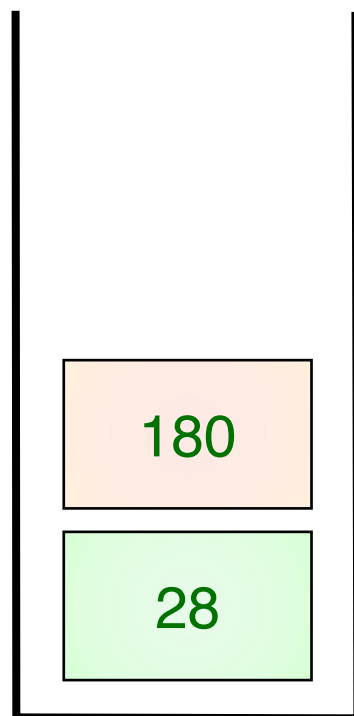


Operations

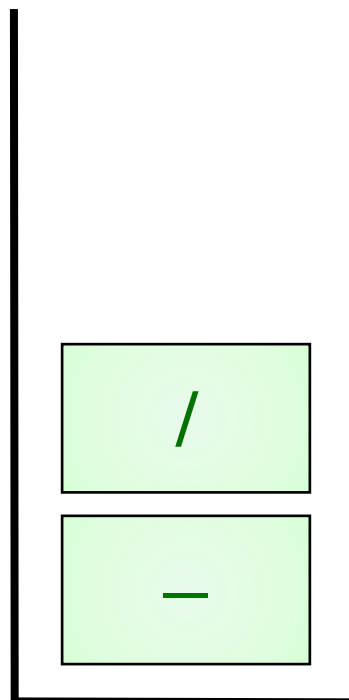
(in increasing order
of precedence)

Expression Evaluation using Stack

Example: $28 - 5 * 6 ^ 2 / 9 + 4$



Numbers

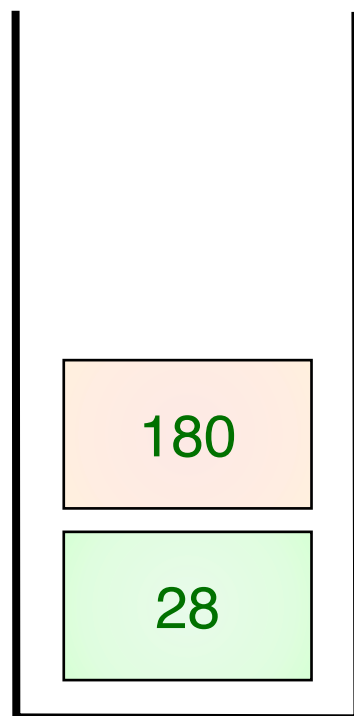


Operations

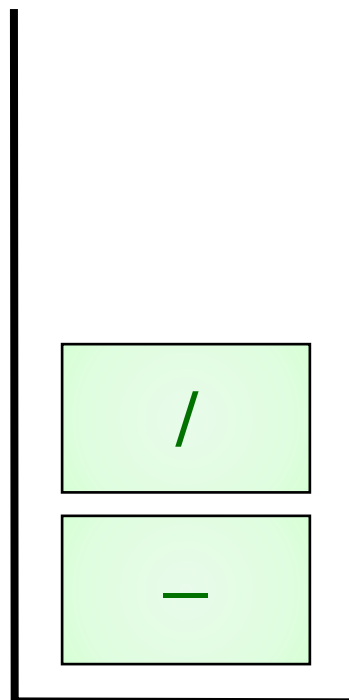
(in increasing order
of precedence)

Expression Evaluation using Stack

Example: $28 - 5 * 6 ^ 2 / 9 + 4$



Numbers



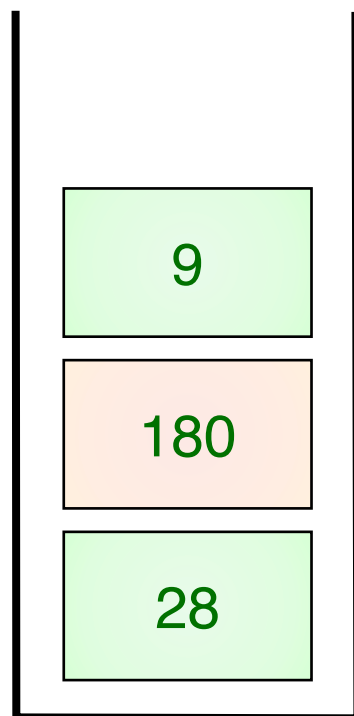
Operations

(in increasing order
of precedence)

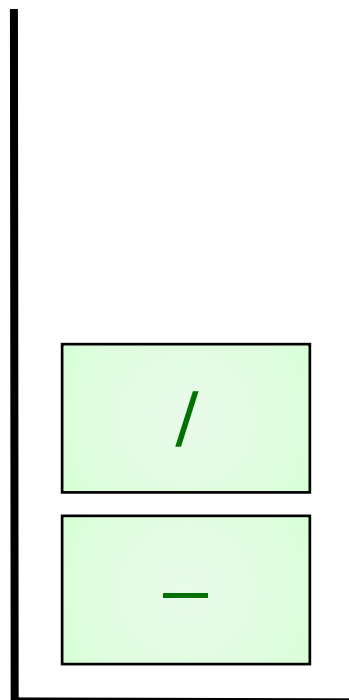
Expression Evaluation using Stack

Example: $28 - 5 * 6 ^ 2 / 9 + 4$

↑



Numbers

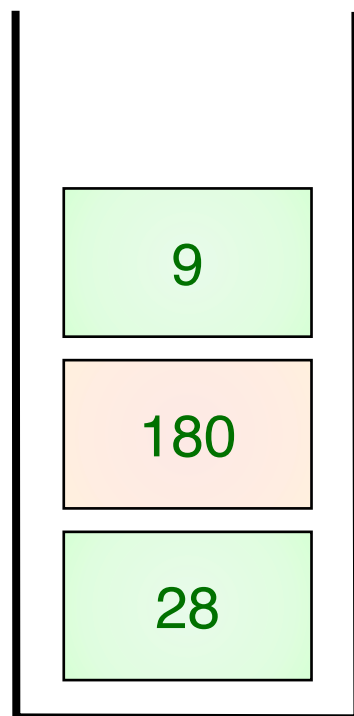


Operations

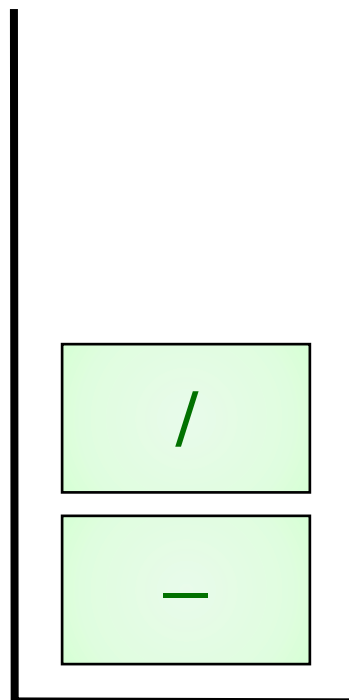
(in increasing order
of precedence)

Expression Evaluation using Stack

Example: $28 - 5 * 6 ^ 2 / 9 + 4$



Numbers

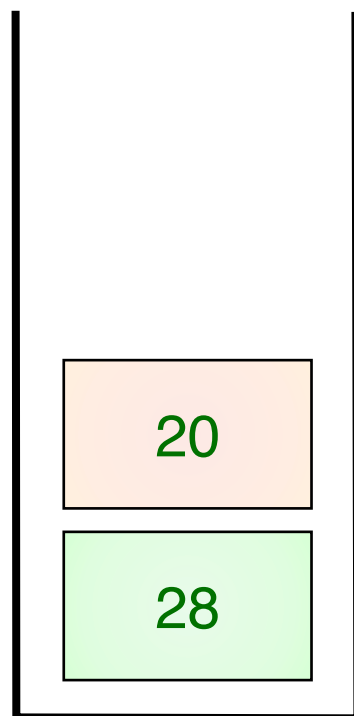


Operations

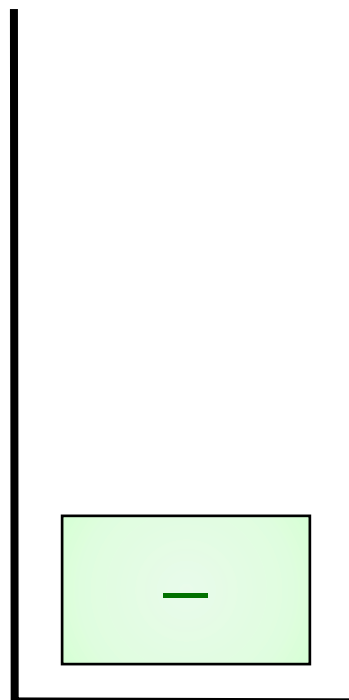
(in increasing order
of precedence)

Expression Evaluation using Stack

Example: $28 - 5 * 6 ^ 2 / 9 + 4$



Numbers

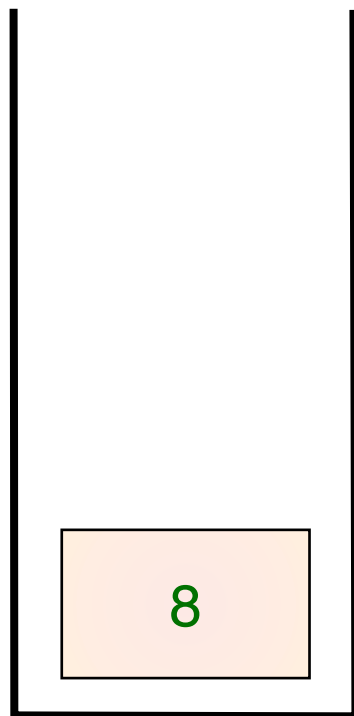


Operations

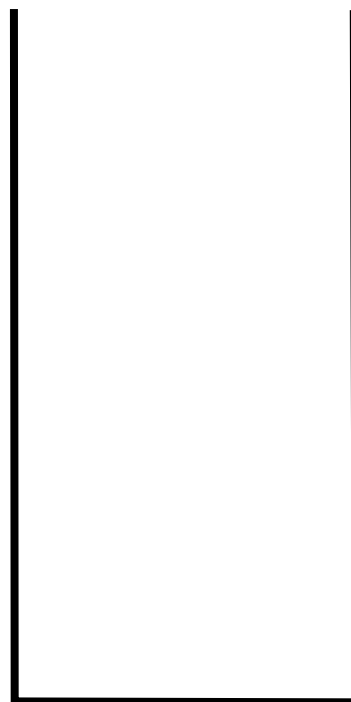
(in increasing order
of precedence)

Expression Evaluation using Stack

Example: $28 - 5 * 6 ^ 2 / 9 + 4$



Numbers

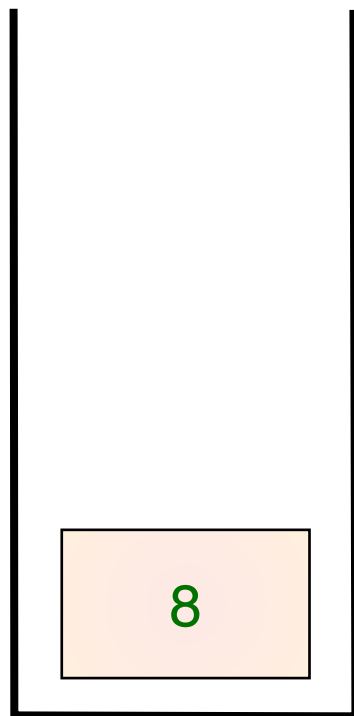


Operations

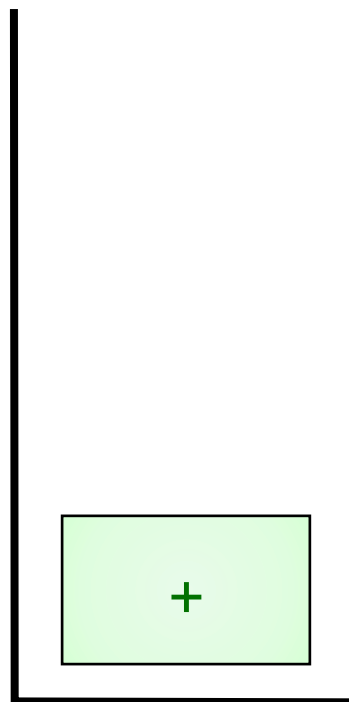
(in increasing order
of precedence)

Expression Evaluation using Stack

Example: $28 - 5 * 6 ^ 2 / 9 + 4$



Numbers

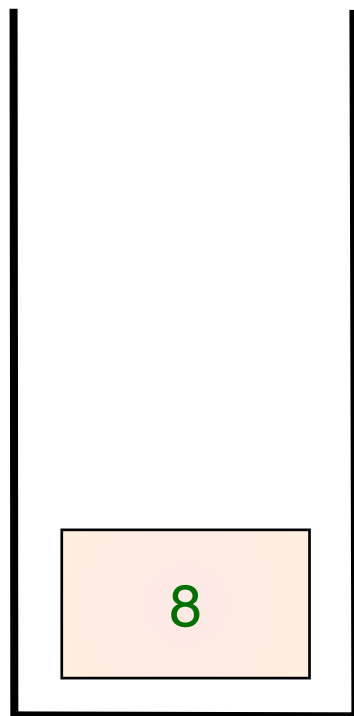


Operations

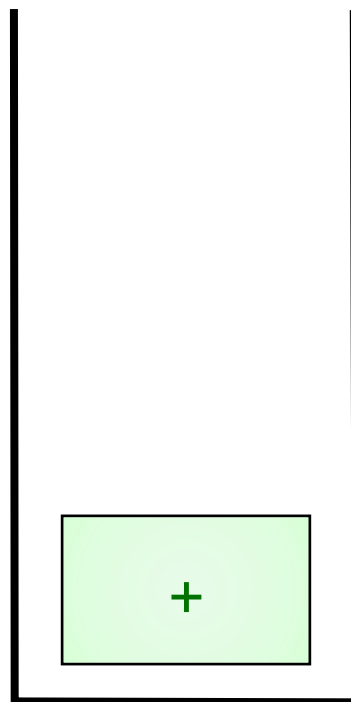
(in increasing order
of precedence)

Expression Evaluation using Stack

Example: $28 - 5 * 6 ^ 2 / 9 + 4$



Numbers

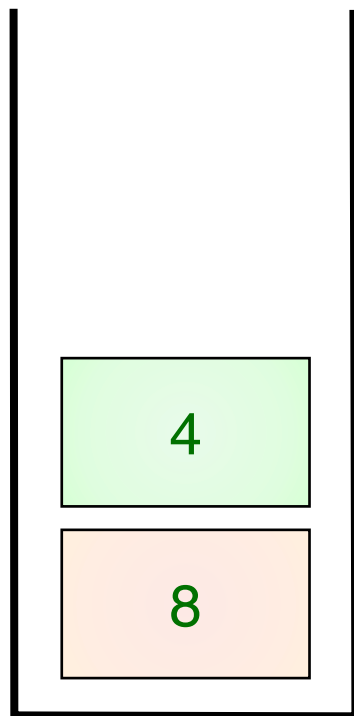


Operations

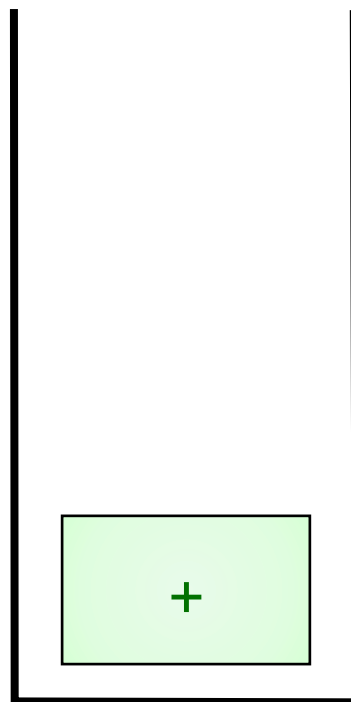
(in increasing order
of precedence)

Expression Evaluation using Stack

Example: $28 - 5 * 6 ^ 2 / 9 + 4$



Numbers

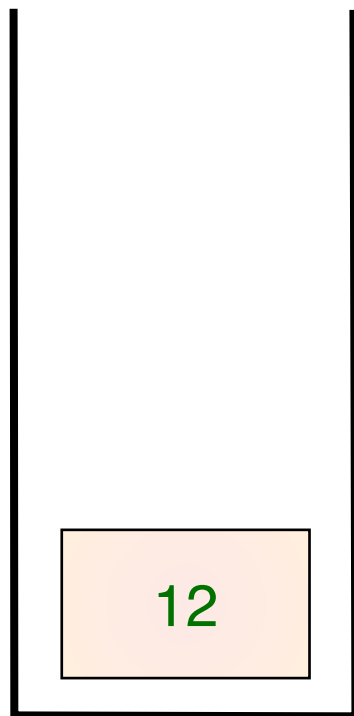


Operations

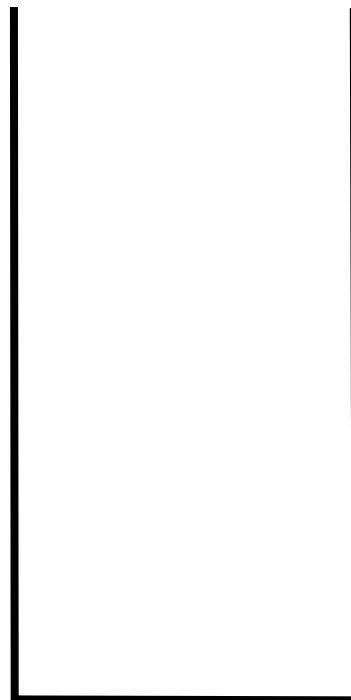
(in increasing order
of precedence)

Expression Evaluation using Stack

Example: $28 - 5 * 6 ^ 2 / 9 + 4$



Numbers

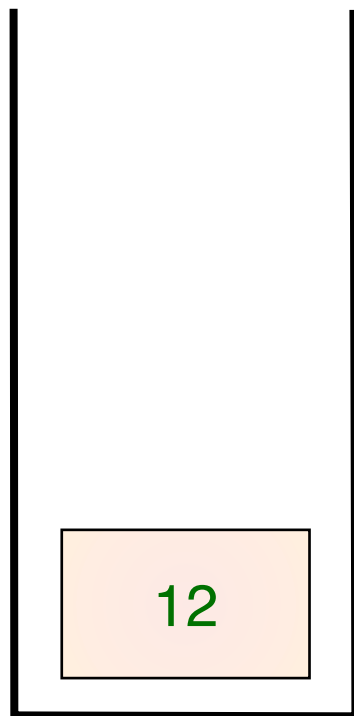


Operations

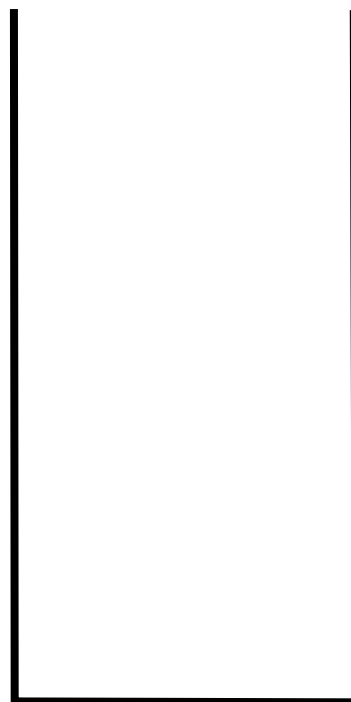
(in increasing order
of precedence)

Expression Evaluation using Stack

Example: $28 - 5 * 6 ^ 2 / 9 + 4$



Numbers



Operations

(in increasing order
of precedence)

The output is 12.

Expression Evaluation using Stack

Input: An array X of n tokens, each of which is either a number or a operation($+$, $-$, $*$, $/$, $^$)

```
Let Num, Op be two empty stacks.  
for  $i = 0$  to  $n - 1$  do  
    if  $X[i]$  is a number token then  
        Num.push( $X[i]$ )  
    else if  $X[i]$  is an operation token then  
        {Execute operations with priority larger than or same as  $X[i]$  }  
        while Priority( Op.top() )  $\geq$  Priority(  $X[i]$  ) then  
            Pop last two numbers, and last operation.  
            Execute the operation and Push result back in number stack.  
        Op.push( $X[i]$ )  
    while Op.(size)  $> 0$  then {emptying operation stack}  
        Pop last two numbers, and last operation.  
        Execute the operation and Push result back in number stack.  
return Num.top() {final output}
```

Expression Evaluation using Stack

Input: An array X of n tokens, each of which is either a number or a operation($+$, $-$, $*$, $/$, $^$)

How to handle
right associativity
of $^$

```
Let Num, Op be two empty stacks.  
for  $i = 0$  to  $n - 1$  do  
    if  $X[i]$  is a number token then  
        Num.push( $X[i]$ )  
    else if  $X[i]$  is an operation token then  
        {Execute operations with priority larger than or same as  $X[i]$  }  
        while Priority( Op.top() )  $\geq$  Priority(  $X[i]$  ) then ←  
            Pop last two numbers, and last operation.  
            Execute the operation and Push result back in number stack.  
        Op.push( $X[i]$ )  
    while Op.(size)  $> 0$  then {emptying operation stack}  
        Pop last two numbers, and last operation.  
        Execute the operation and Push result back in number stack.  
return Num.top() {final output}
```

Expression Evaluation using Stack

Input: An array X of n tokens, each of which is either a number or a operation($+$, $-$, $*$, $/$, $^$)

Let Num , Op be two empty stacks.

for $i = 0$ to $n - 1$ **do**

if $X[i]$ is a number token **then**

$Num.push(X[i])$

else if $X[i]$ is an operation token **then**

{Execute operations with priority larger than or same as $X[i]$ }

while $Priority(Op.top()) \geq Priority(X[i])$ and $X[i], Op.top() \neq ^$ **then**

Pop last two numbers, and last operation.

Execute the operation and Push result back in number stack.

$Op.push(X[i])$

while $Op.size() > 0$ **then** *{emptying operation stack}*

Pop last two numbers, and last operation.

Execute the operation and Push result back in number stack.

return $Num.top()$ *{final output}*

If both previous and current operations are $^$ then we simply push the operation in stack

Implementing Stack using Arrays

Array size = N

Stack size = 'size'

```
public class Stack{
```

```
    private int N=10;  
    private int[] X=new int[N];  
    private int size = 0;
```

```
    public void push(int data)  
    {  
        X[size]=data;  
        size=size+1;  
    }
```

```
    public int top()  
    {  
        return X[size-1];  
    }
```

```
    public int pop()  
    {  
        int value = X[size-1];  
        size = size-1;  
        return value;  
    }
```

```
}
```

Integer array **X** of size **N=10**



size=0

Implementing Stack using Arrays

Array size = N

Stack size = 'size'

```
public class Stack{
```

```
    private int N=10;  
    private int[] X=new int[N];  
    private int size = 0;
```

```
    public void push(int data)  
    {  
        X[size]=data;  
        size=size+1;  
    }
```

```
    public int top()  
    {  
        return X[size-1];  
    }
```

```
    public int pop()  
    {  
        int value = X[size-1];  
        size = size-1;  
        return value;  
    }
```

```
}
```

Integer array **X** of size **N=10**



↑
size=0

Push(10)

Implementing Stack using Arrays

Array size = N

Stack size = 'size'

```
public class Stack{
```

```
    private int N=10;  
    private int[] X=new int[N];  
    private int size = 0;
```

```
    public void push(int data)  
    {  
        X[size]=data;  
        size=size+1;  
    }
```

```
    public int top()  
    {  
        return X[size-1];  
    }
```

```
    public int pop()  
    {  
        int value = X[size-1];  
        size = size-1;  
        return value;  
    }
```

```
}
```

Integer array **X** of size **N=10**



↑
size=1

Push(10)

Implementing Stack using Arrays

Array size = N

Stack size = 'size'

```
public class Stack{
```

```
    private int N=10;  
    private int[] X=new int[N];  
    private int size = 0;
```

```
    public void push(int data)  
    {  
        X[size]=data;  
        size=size+1;  
    }
```

```
    public int top()  
    {  
        return X[size-1];  
    }
```

```
    public int pop()  
    {  
        int value = X[size-1];  
        size = size-1;  
        return value;  
    }
```

```
}
```

Integer array **X** of size **N=10**



↑
size=1

Push(11)

Implementing Stack using Arrays

Array size = N

Stack size = 'size'

```
public class Stack{
```

```
    private int N=10;  
    private int[] X=new int[N];  
    private int size = 0;
```

```
    public void push(int data)  
    {  
        X[size]=data;  
        size=size+1;  
    }
```

```
    public int top()  
    {  
        return X[size-1];  
    }
```

```
    public int pop()  
    {  
        int value = X[size-1];  
        size = size-1;  
        return value;  
    }
```

```
}
```

Integer array **X** of size **N=10**



↑
size=2

Push(11)

Implementing Stack using Arrays

Array size = N

Stack size = 'size'

```
public class Stack{
```

```
    private int N=10;  
    private int[] X=new int[N];  
    private int size = 0;
```

```
    public void push(int data)  
    {  
        X[size]=data;  
        size=size+1;  
    }
```

```
    public int top()  
    {  
        return X[size-1];  
    }
```

```
    public int pop()  
    {  
        int value = X[size-1];  
        size = size-1;  
        return value;  
    }
```

```
}
```

Integer array **X** of size **N=10**



↑
size=2

Push(12)

Implementing Stack using Arrays

Array size = N

Stack size = 'size'

```
public class Stack{
```

```
    private int N=10;  
    private int[] X=new int[N];  
    private int size = 0;
```

```
    public void push(int data)  
    {  
        X[size]=data;  
        size=size+1;  
    }
```

```
    public int top()  
    {  
        return X[size-1];  
    }
```

```
    public int pop()  
    {  
        int value = X[size-1];  
        size = size-1;  
        return value;  
    }
```

```
}
```

Integer array **X** of size **N=10**

0	1	2	3	4	5	6	7	8	9
10	11	12							



size=3

Push(12)

Implementing Stack using Arrays

Array size = N

Stack size = 'size'

```
public class Stack{
```

```
    private int N=10;  
    private int[] X=new int[N];  
    private int size = 0;
```

```
    public void push(int data)  
    {  
        X[size]=data;  
        size=size+1;  
    }
```

```
    public int top()  
    {  
        return X[size-1];  
    }
```

```
    public int pop()  
    {  
        int value = X[size-1];  
        size = size-1;  
        return value;  
    }
```

```
}
```

Integer array **X** of size **N=10**

0	1	2	3	4	5	6	7	8	9
10	11	12							



size=3

Pop()

Implementing Stack using Arrays

Array size = N

Stack size = 'size'

```
public class Stack{
```

```
    private int N=10;  
    private int[] X=new int[N];  
    private int size = 0;
```

```
    public void push(int data)  
    {  
        X[size]=data;  
        size=size+1;  
    }
```

```
    public int top()  
    {  
        return X[size-1];  
    }
```

```
    public int pop()  
    {  
        int value = X[size-1];  
        size = size-1;  
        return value;  
    }
```

```
}
```

Integer array **X** of size **N=10**



↑
size=2

Pop()

Implementing Stack using Arrays

Array size = N

Stack size = 'size'

```
public class Stack{
```

```
    private int N=10;  
    private int[] X=new int[N];  
    private int size = 0;
```

```
    public void push(int data)  
    {  
        X[size]=data;  
        size=size+1;  
    }
```

```
    public int top()  
    {  
        return X[size-1];  
    }
```

```
    public int pop()  
    {  
        int value = X[size-1];  
        size = size-1;  
        return value;  
    }
```

```
}
```

Integer array **X** of size **N=10**



↑
size=2

Push(13)

Implementing Stack using Arrays

Array size = N

Stack size = 'size'

```
public class Stack{
```

```
    private int N=10;  
    private int[] X=new int[N];  
    private int size = 0;
```

```
    public void push(int data)  
    {  
        X[size]=data;  
        size=size+1;  
    }
```

```
    public int top()  
    {  
        return X[size-1];  
    }
```

```
    public int pop()  
    {  
        int value = X[size-1];  
        size = size-1;  
        return value;  
    }
```

```
}
```

Integer array **X** of size **N=10**

0	1	2	3	4	5	6	7	8	9
10	11	13							



size=3

Push(13)

Implementing Stack using Arrays

Array size = N

Stack size = 'size'

```
public class Stack{
```

```
    private int N=10;  
    private int[] X=new int[N];  
    private int size = 0;
```

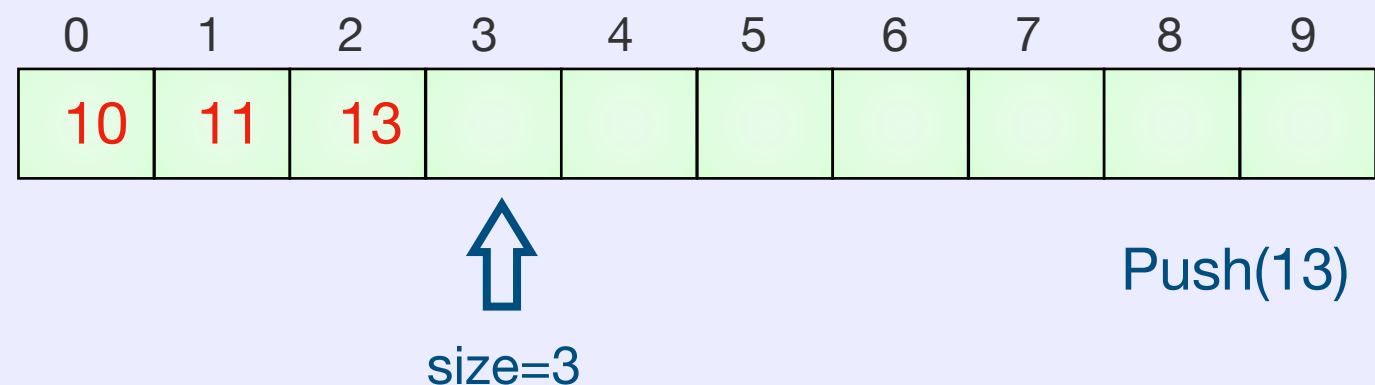
```
    public void push(int data)  
    {  
        X[size]=data;  
        size=size+1;  
    }
```

```
    public int top()  
    {  
        return X[size-1];  
    }
```

```
    public int pop()  
    {  
        int value = X[size-1];  
        size = size-1;  
        return value;  
    }
```

```
}
```

Integer array **X** of size $N=10$



Limitation: There is a bound (N) on the size of stack.

Stack from Dynamic Arrays

Doubling approach

Make a new array of **double** size whenever the array is **full**.

```
public void push(int data)
{
    if(size == N)
    {
        int[ ] temp = X;
        N = 2*N;
        X = new int[N];
        for(int i=0; i< size; i++)
            X[i]=temp[i ];
    }

    X[size]=data;
    size=size+1;
}
```

```
public int pop()
{
```

```
    int value = X[size-1];
    size = size-1;
```

```
    if(size < N/4 && N>40)
    {
```

```
        int[ ] temp = X;
        N = N/2;
        X = new int[N];
        for(int i=0; i< size; i++)
            X[i]=temp[i ];
```

```
    }
```

```
    return value;
```

```
}
```

Halving approach

Make a new array of **half** size whenever the array is **3/4th empty**.

Stack from Dynamic Arrays

Doubling approach

Make a new array of **double** size whenever the array is **full**.

```
public void push(int data)
{
    if(size == N)
    {
        int[ ] temp = X;
        N = 2*N;
        X = new int[N];
        for(int i=0; i< size; i++)
            X[i]=temp[i];
    }

    X[size]=data;
    size=size+1;
}
```

```
public int pop()
{
    int value = X[size-1];
    size = size-1;

    if(size < N/4 && N>40)
    {
        int[ ] temp = X;
        N = N/2;
        X = new int[N];
        for(int i=0; i< size; i++)
            X[i]=temp[i];
    }

    return value;
}
```

Halving approach

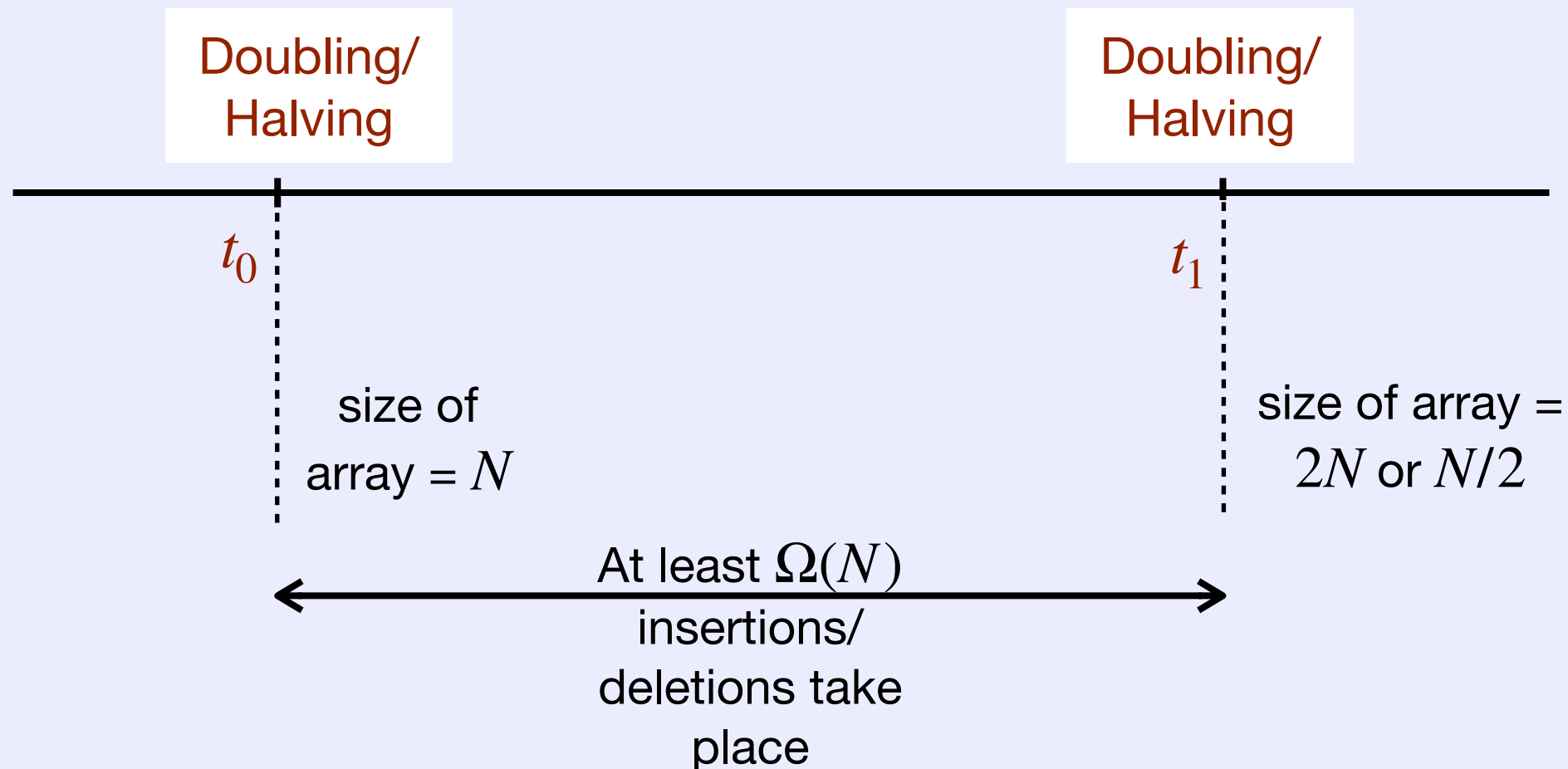
Make a new array of **half** size whenever the array is **3/4th empty**.

Claim: The amortized (average) time-complexity of “push” and “pop” operations is $O(1)$.

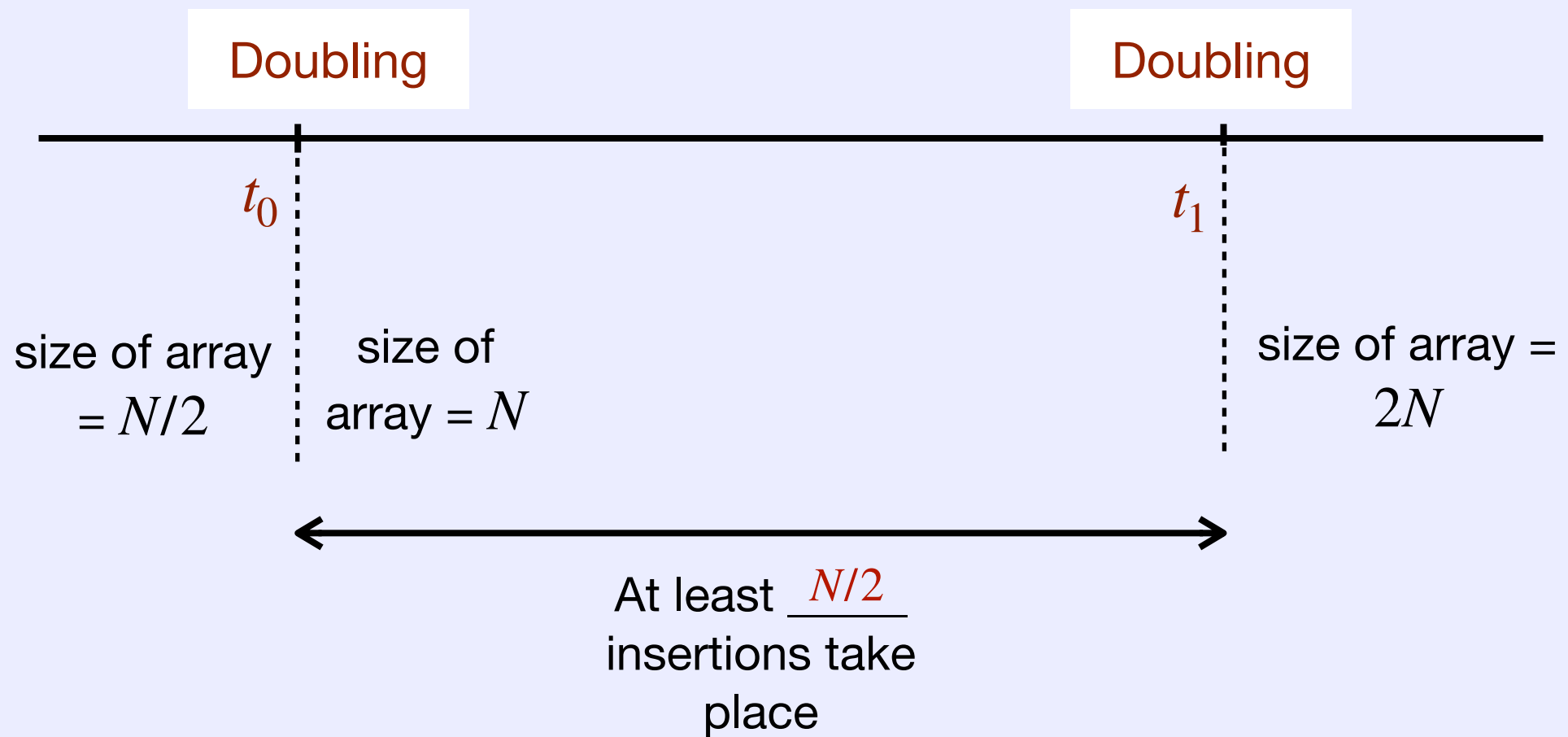
Analysis of Dynamic Arrays

Theorem: Let t_0, t_1 be time stamps of two consecutive doubling/halving operations, and let N = size of array immediately after t_0 . Then,

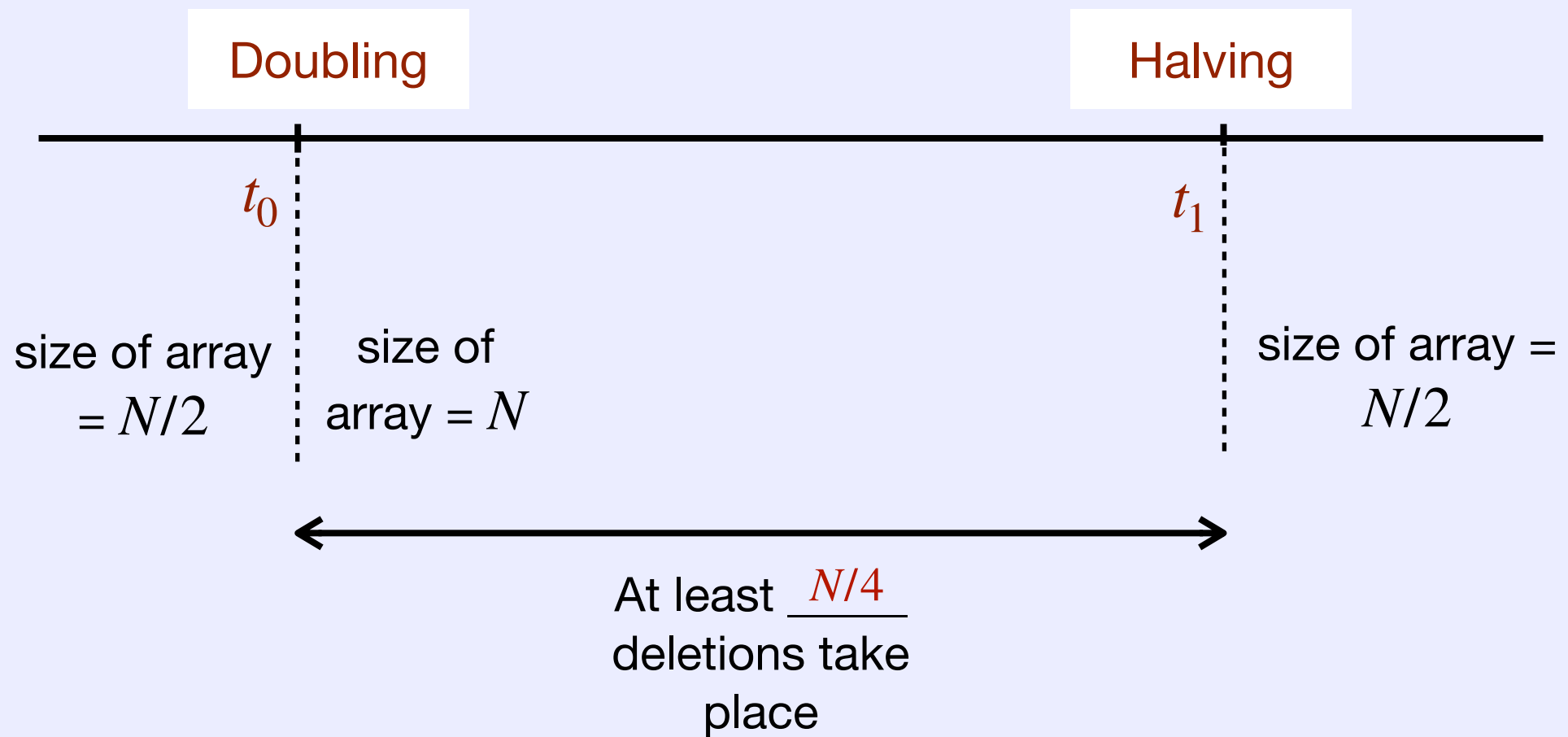
- The number of insertions/deletions in range $[t_0, t_1]$ is at least $\Omega(N)$
- The time to perform doubling/halving at time t_1 is $O(N)$.



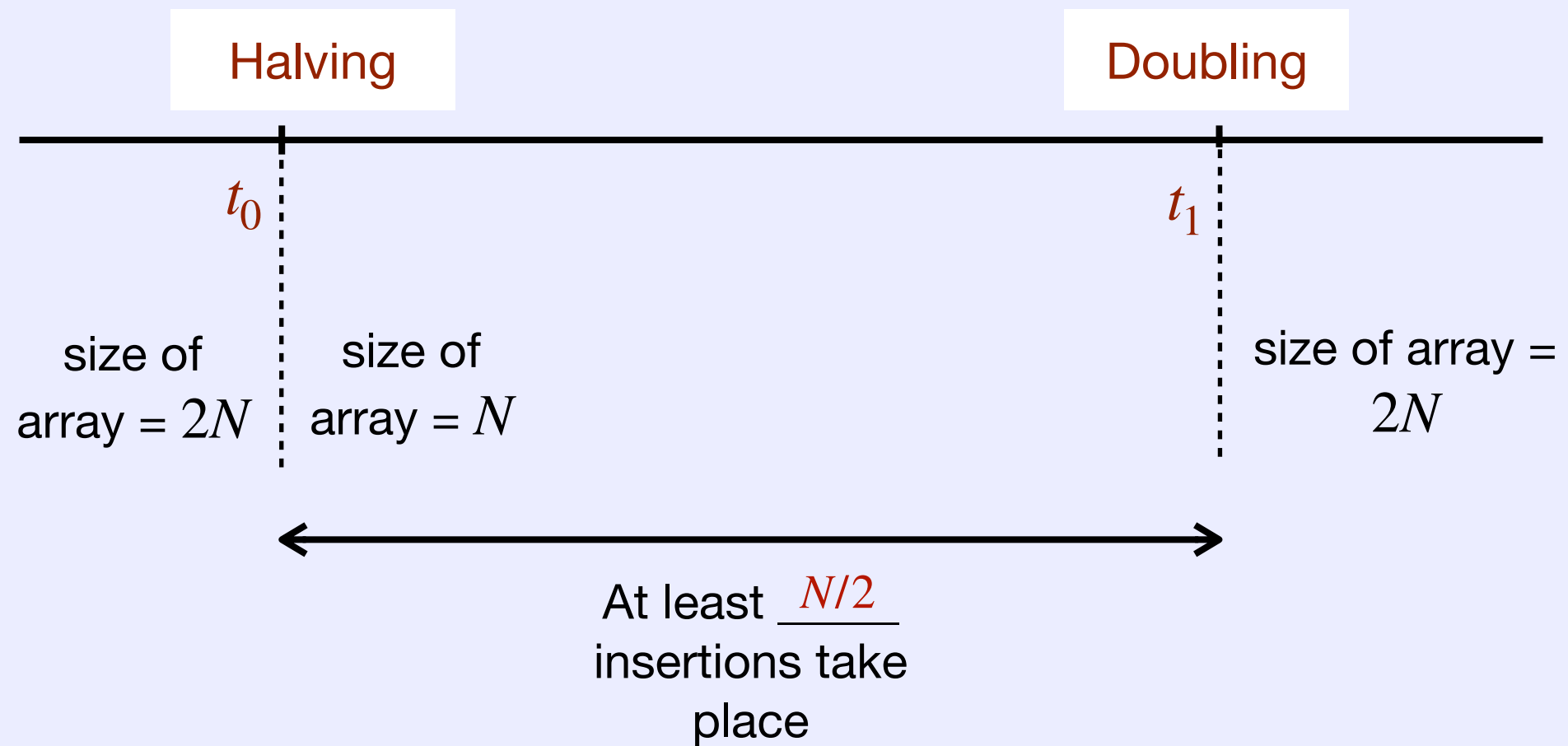
Case 1:



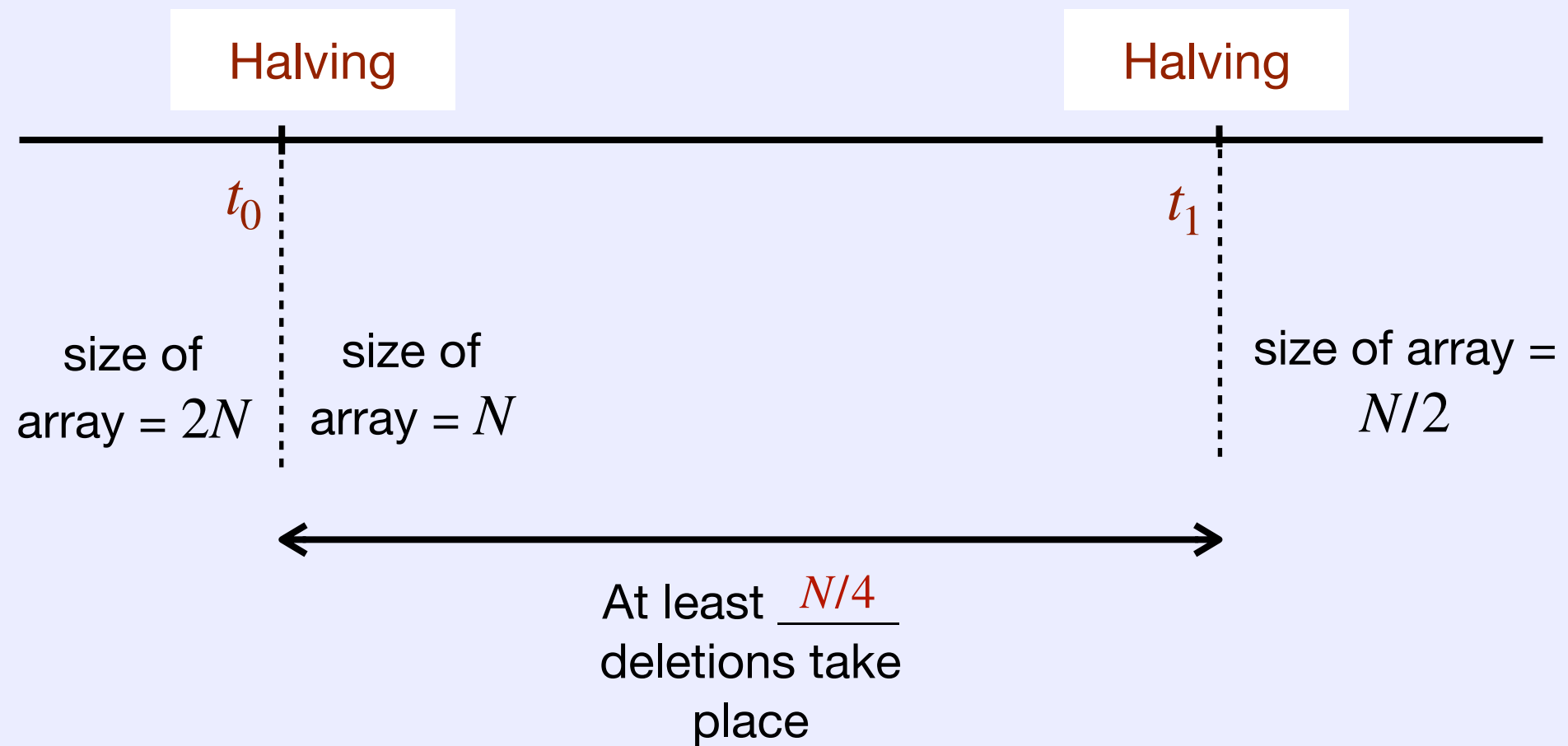
Case 2:



Case 3:



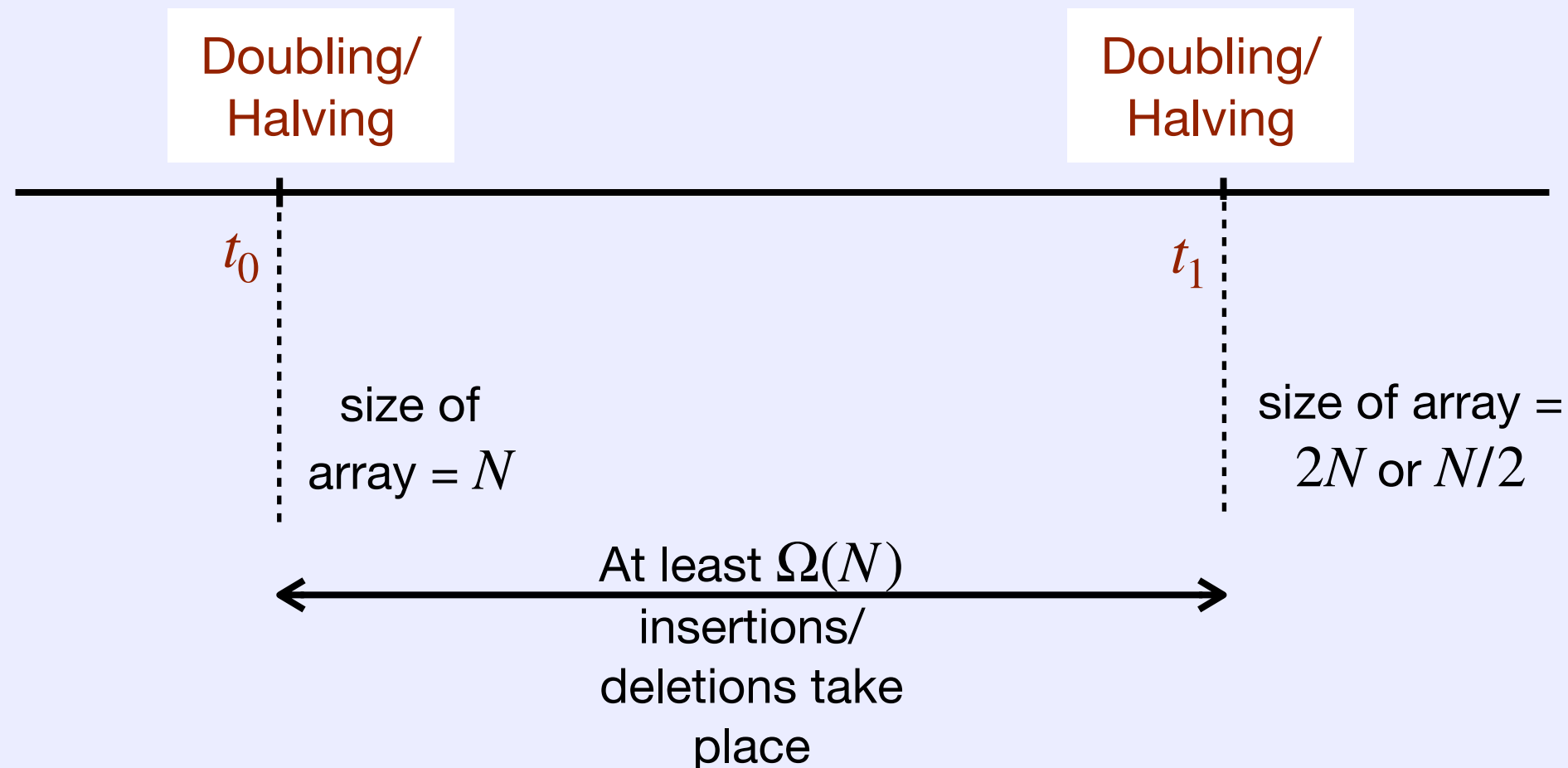
Case 4:



Analysis of Dynamic Arrays

Theorem: Let t_0, t_1 be time stamps of two consecutive doubling/halving operations, and let N = size of array immediately after t_0 . Then,

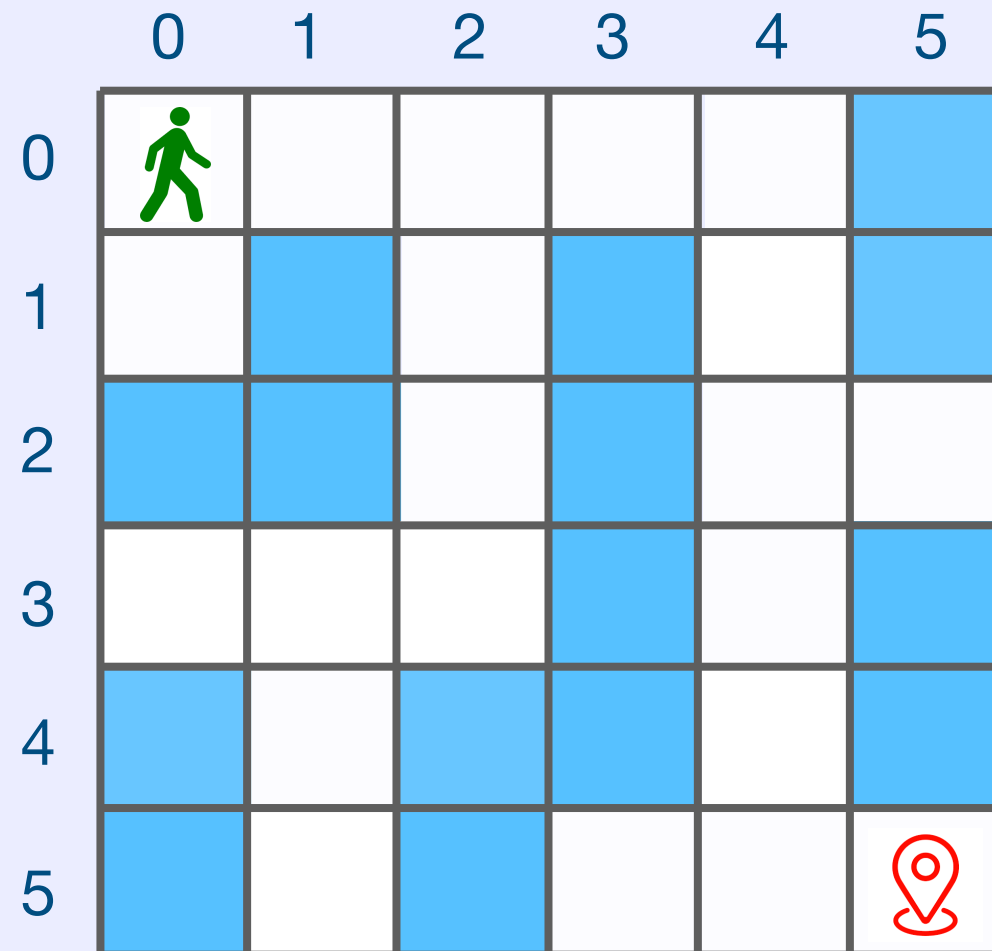
- The number of insertions/deletions in range $[t_0, t_1]$ is at least $\Omega(N)$
 - The time to perform doubling/halving at time t_1 is $O(N)$
- } So, amortized time complexity is $O(1)$

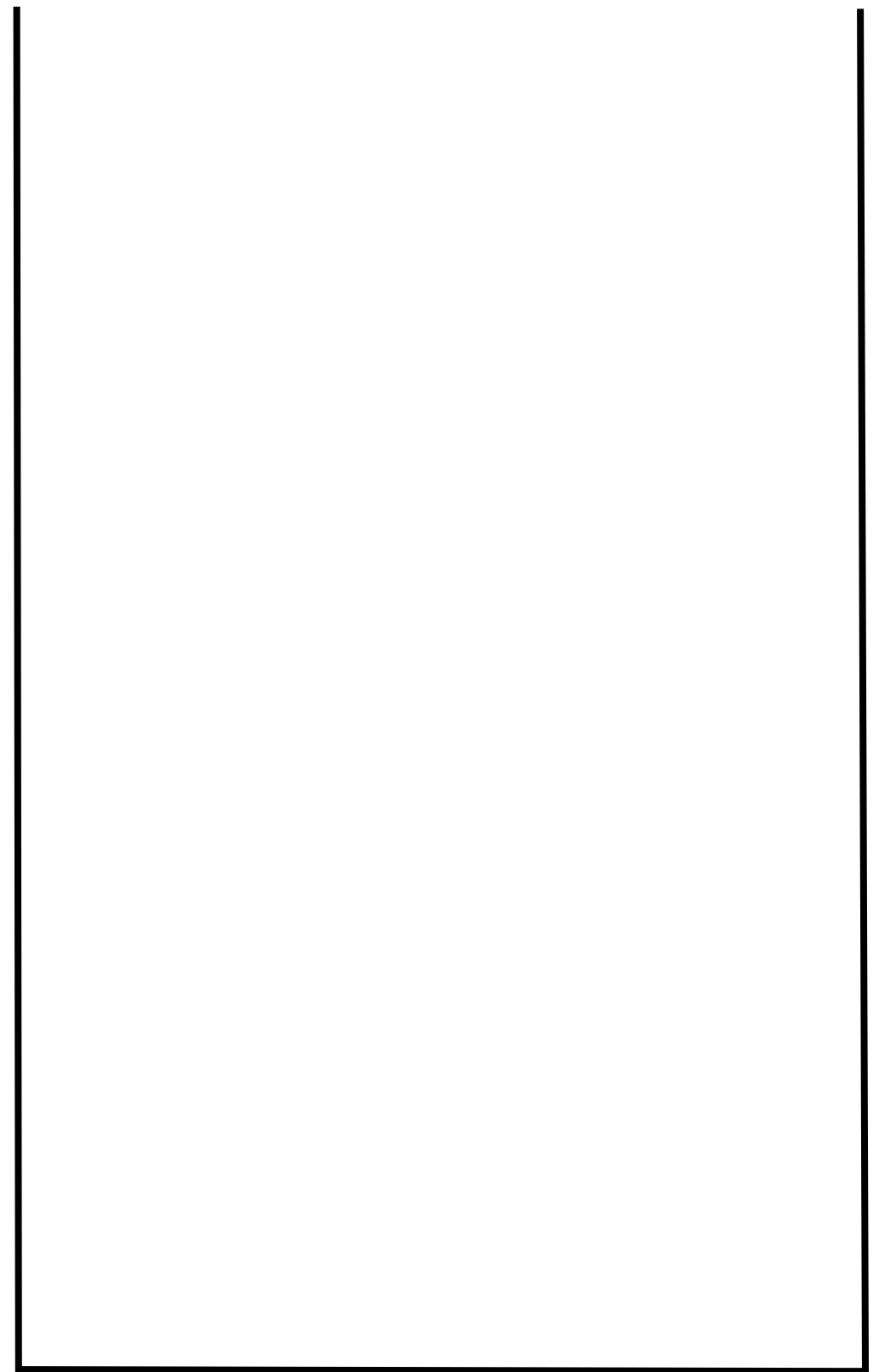
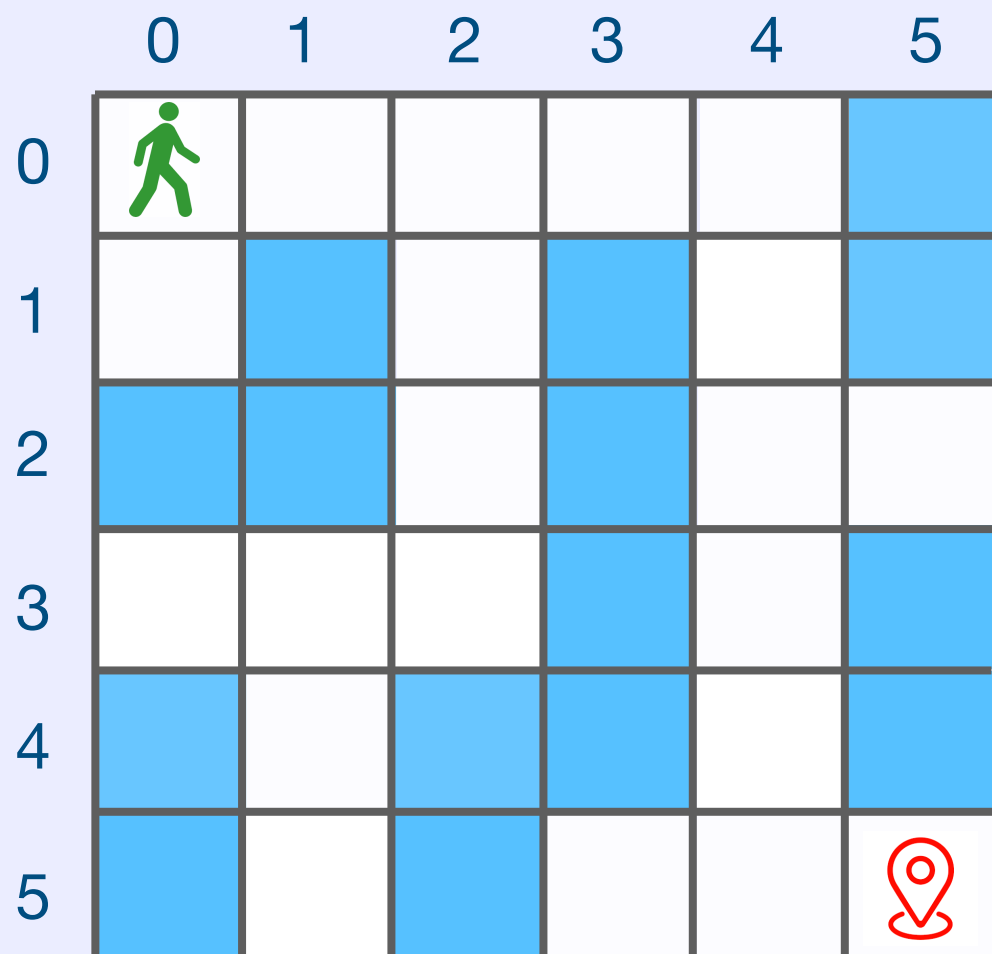


Maze Solving

Given: A maze is represented by a binary matrix M of size $n \times n$.

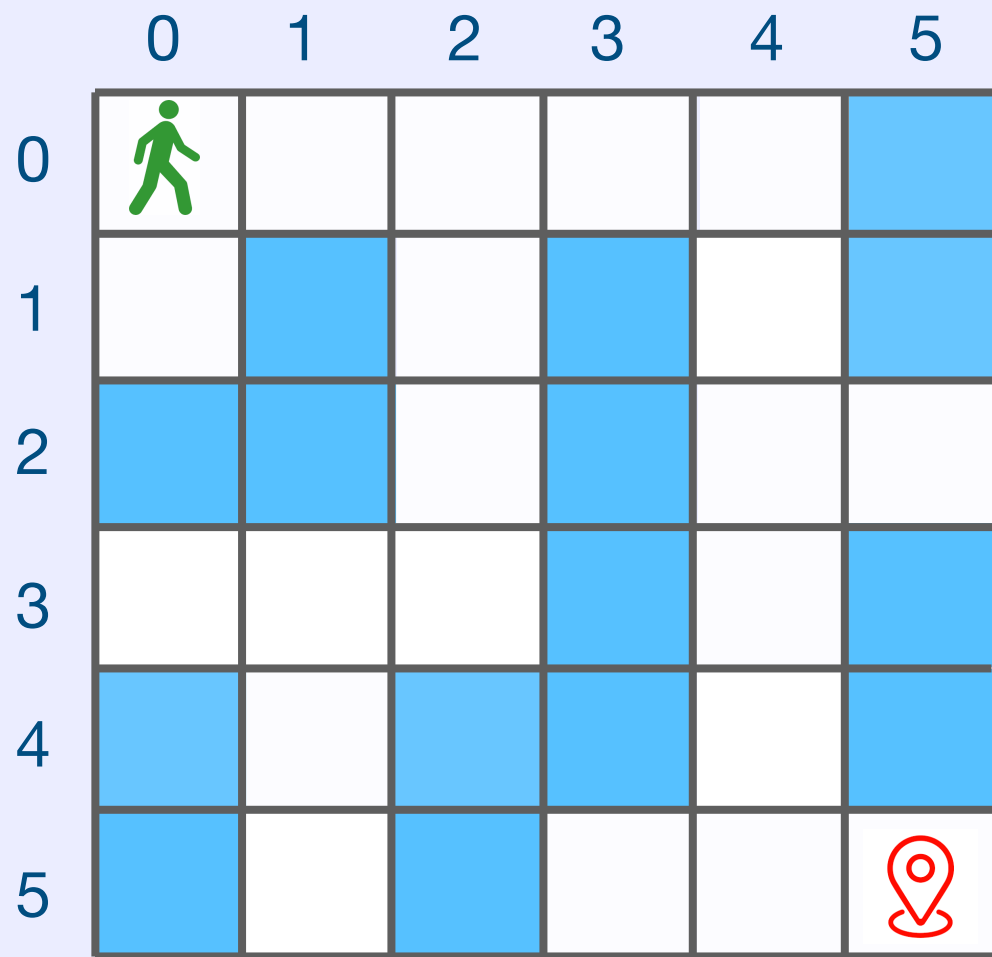
Goal: Use stack to obtain an algorithm for solving the given maze M .





Stack

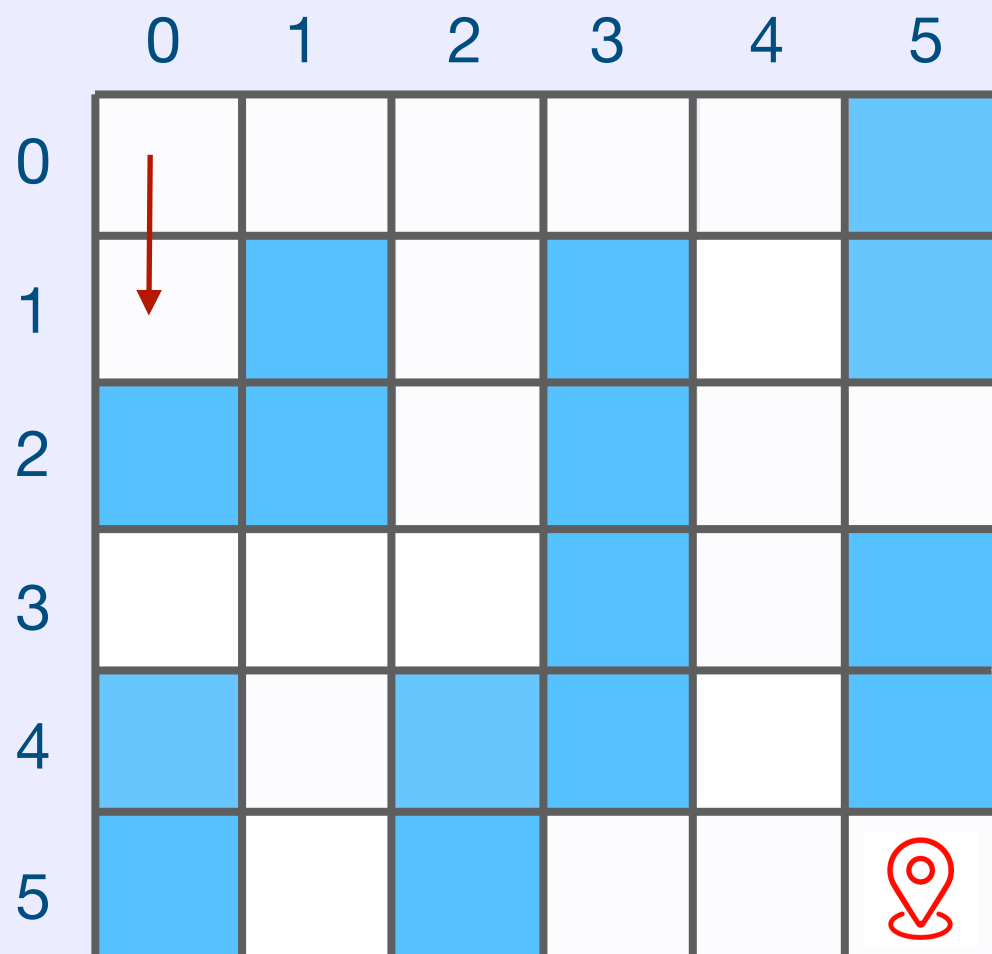
(Cells corresponding to the current path)



(0,0)	Explored = none	Par = -/-
-------	-----------------	-----------

Stack

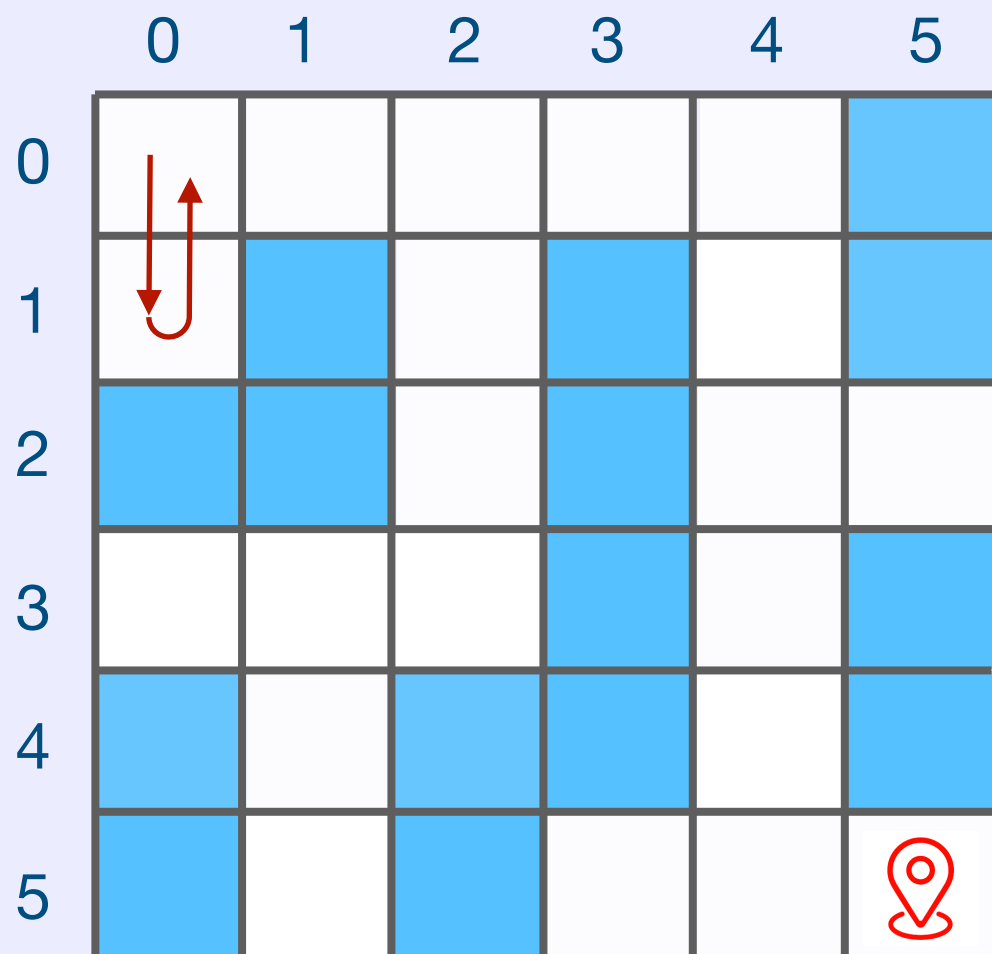
(Cells corresponding to the current path)



(1,0)	Explored = none	Par = T
(0,0)	Explored = none	Par = -/-

Stack

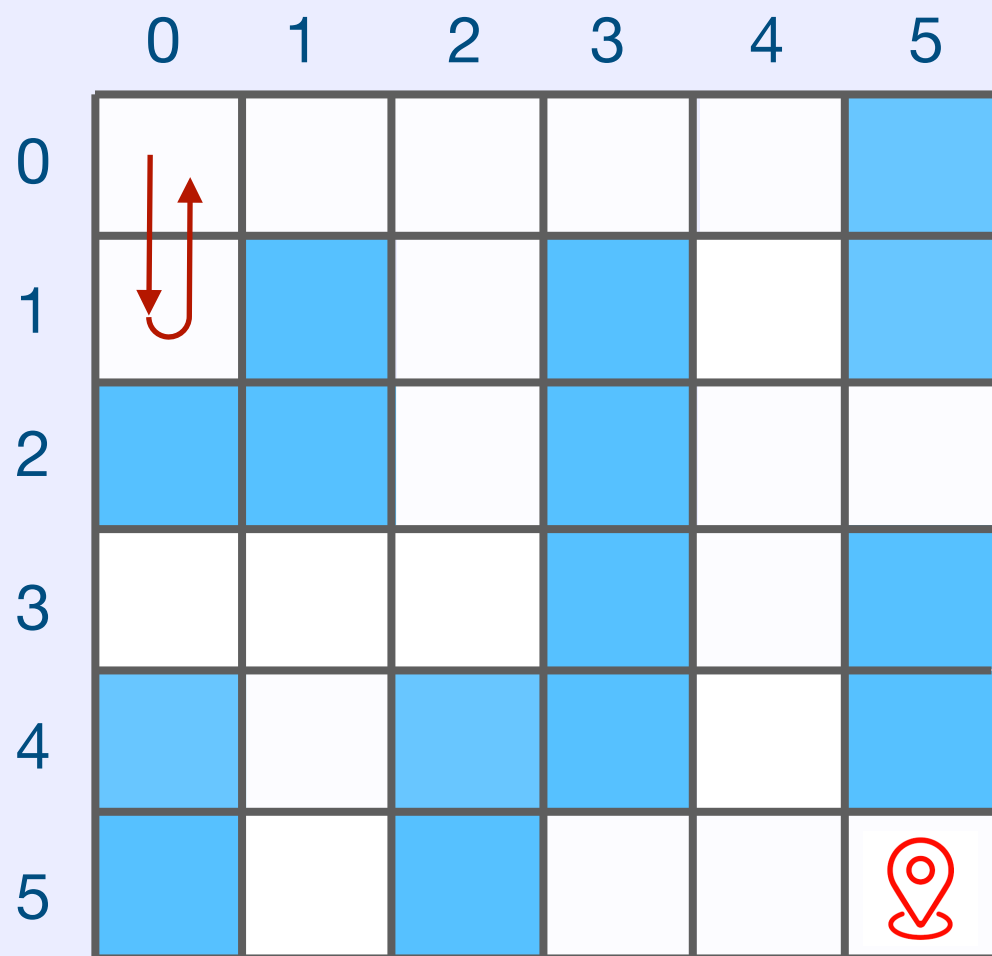
(Cells corresponding to the current path)



(1,0)	Explored = none	Par = T
(0,0)	Explored = none	Par = -/-

Stack

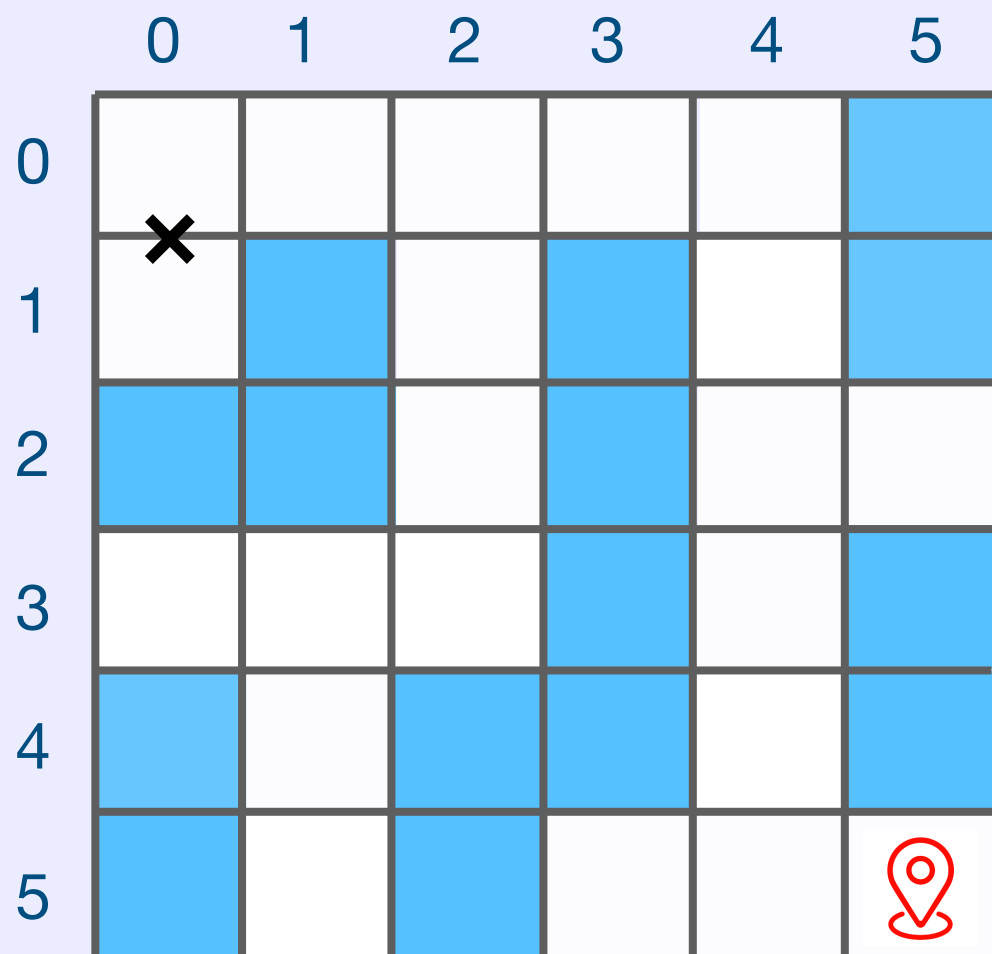
(Cells corresponding to the current path)



(0,0)	Explored = B	Par = -/-
-------	--------------	-----------

Stack

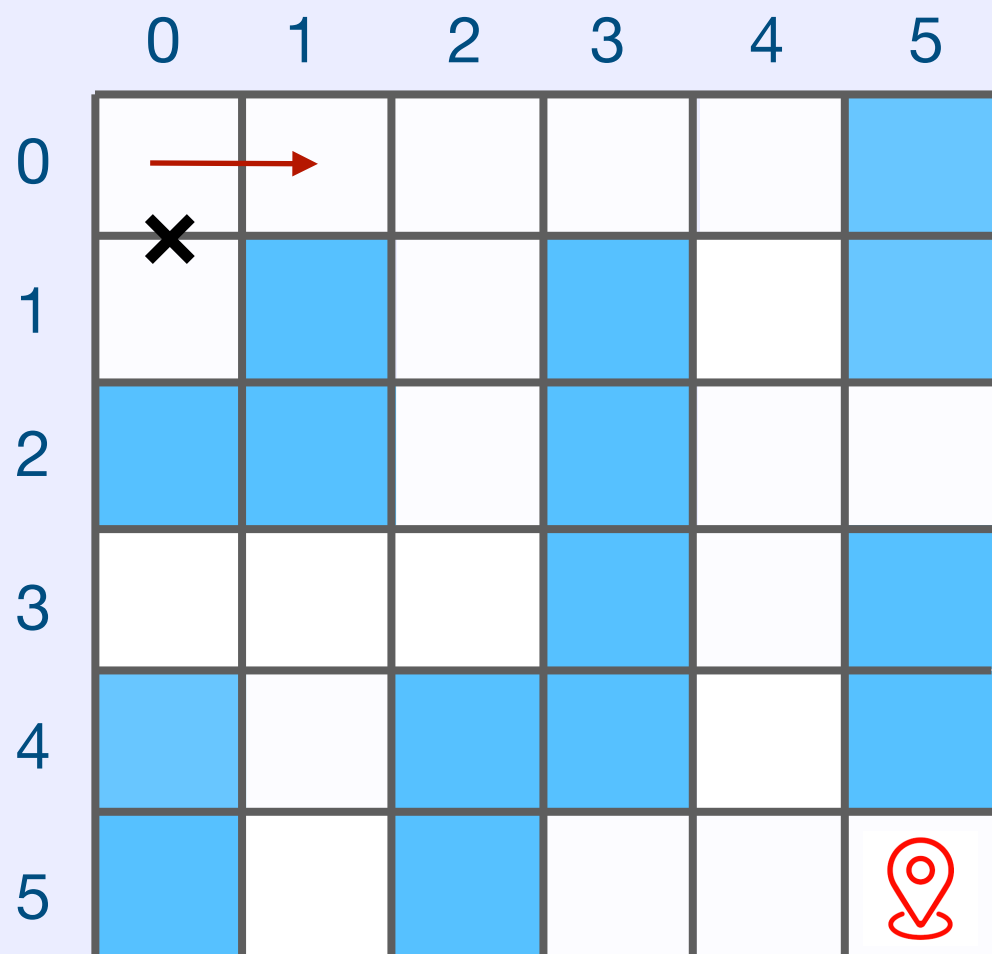
(Cells corresponding to the current path)



(0,0)	Explored = B	Par = -/-
-------	--------------	-----------

Stack

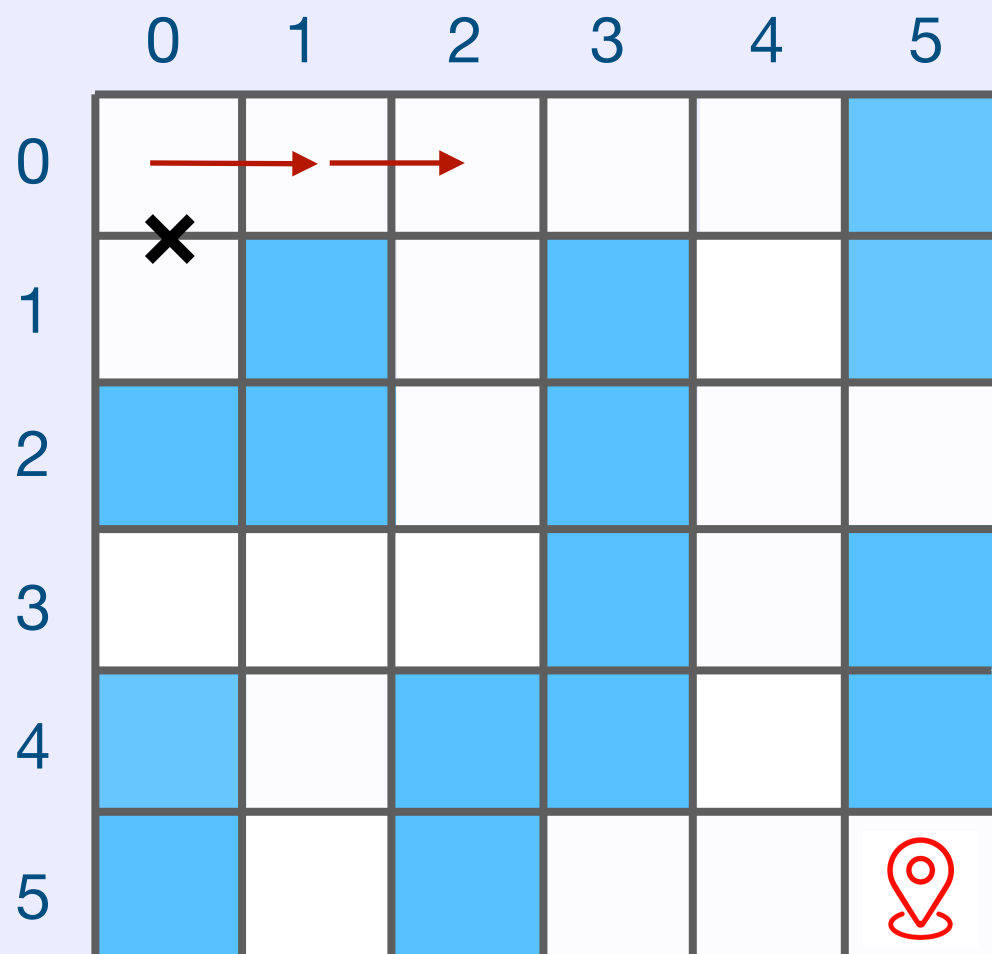
(Cells corresponding to the current path)



(0,1)	Explored = None	Par = L
(0,0)	Explored = B	Par = -/-

Stack

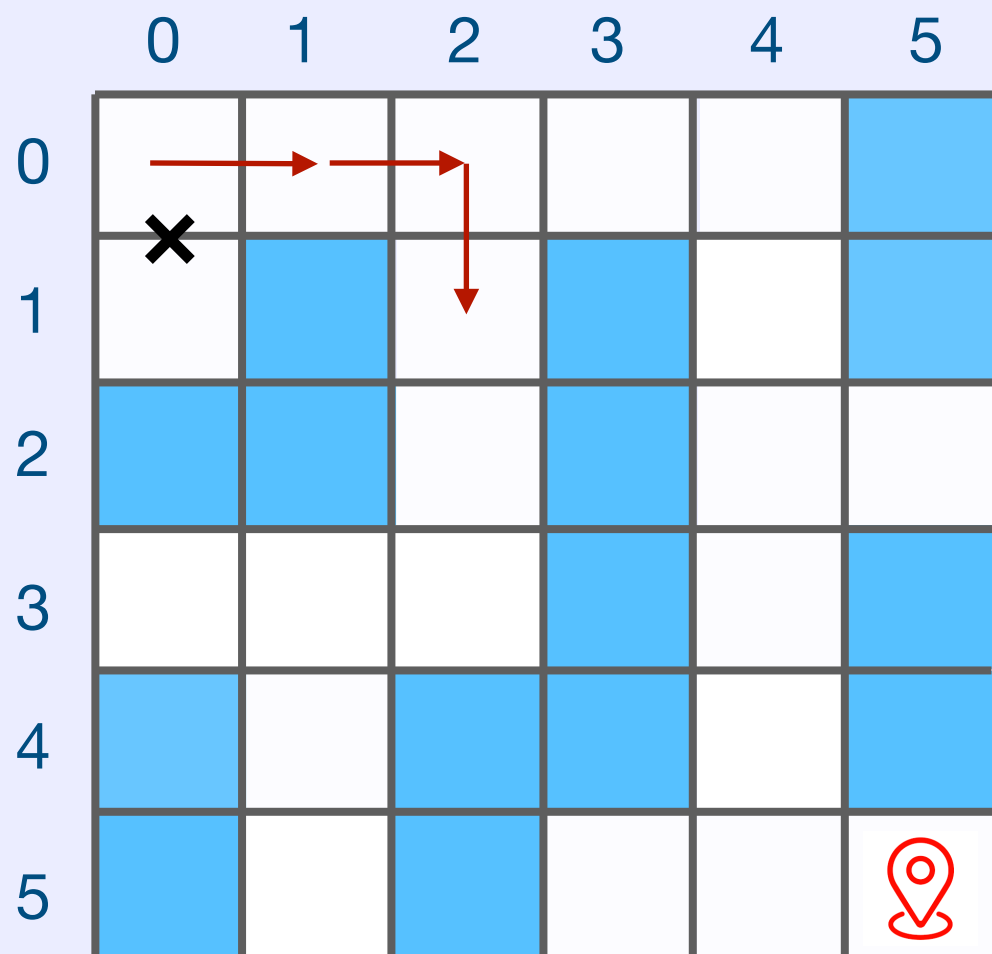
(Cells corresponding to the current path)



(0,2)	Explored = None	Par = L
(0,1)	Explored = None	Par = L
(0,0)	Explored = B	Par = -/-

Stack

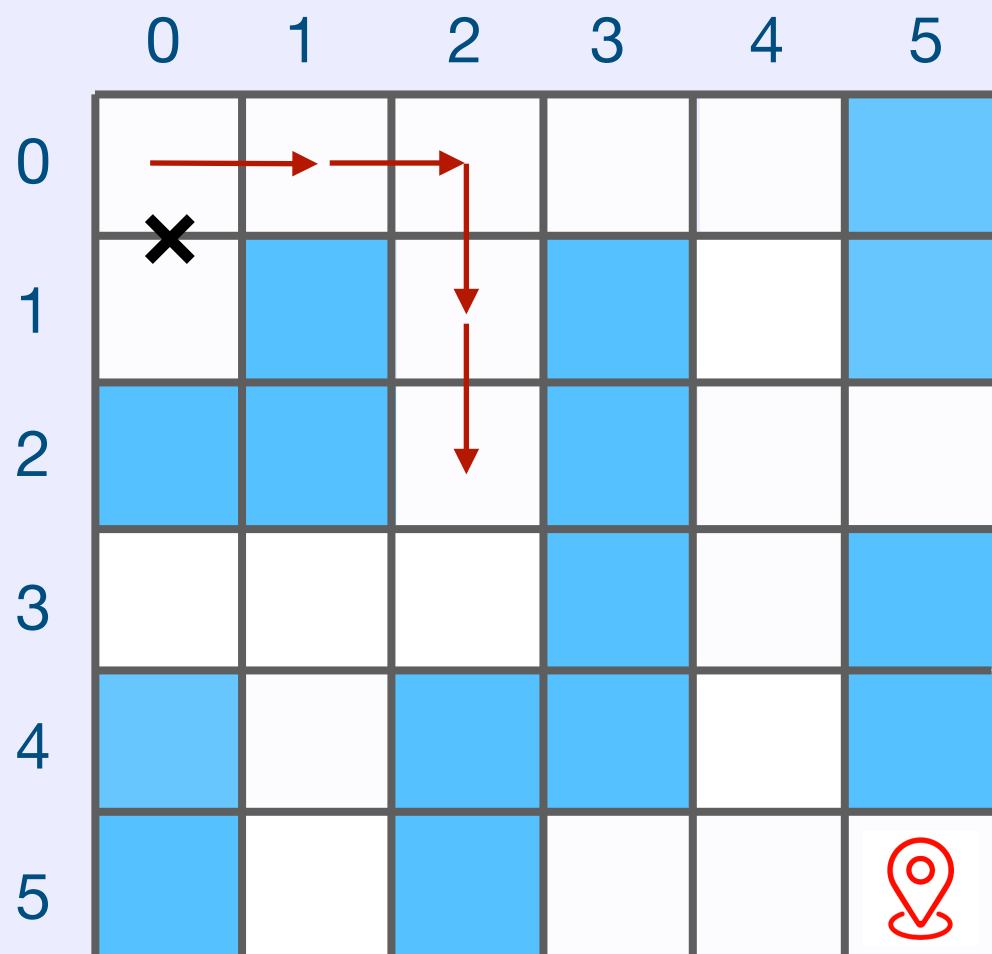
(Cells corresponding to the current path)



(1,2)	Explored = None	Par = T
(0,2)	Explored = None	Par = L
(0,1)	Explored = None	Par = L
(0,0)	Explored = B	Par = -/-

Stack

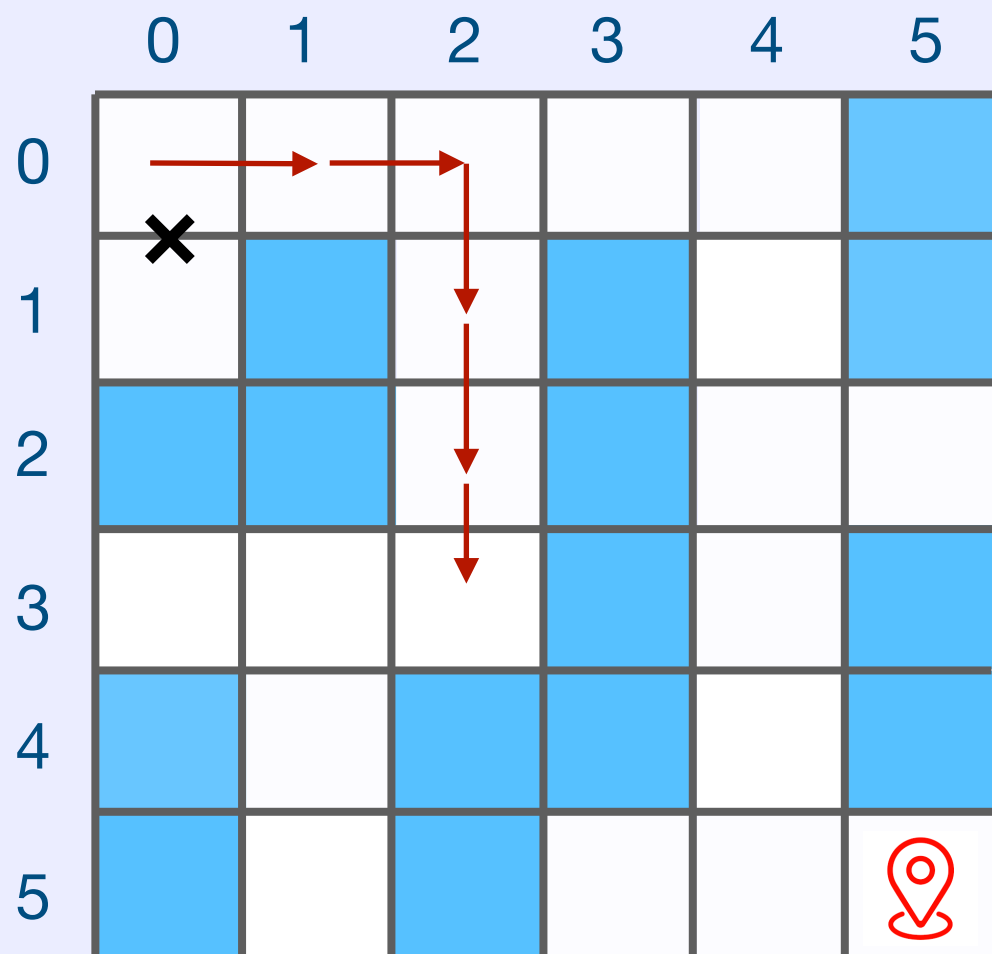
(Cells corresponding to the current path)



(2,2)	Explored = None	Par = T
(1,2)	Explored = None	Par = T
(0,2)	Explored = None	Par = L
(0,1)	Explored = None	Par = L
(0,0)	Explored = B	Par = -/-

Stack

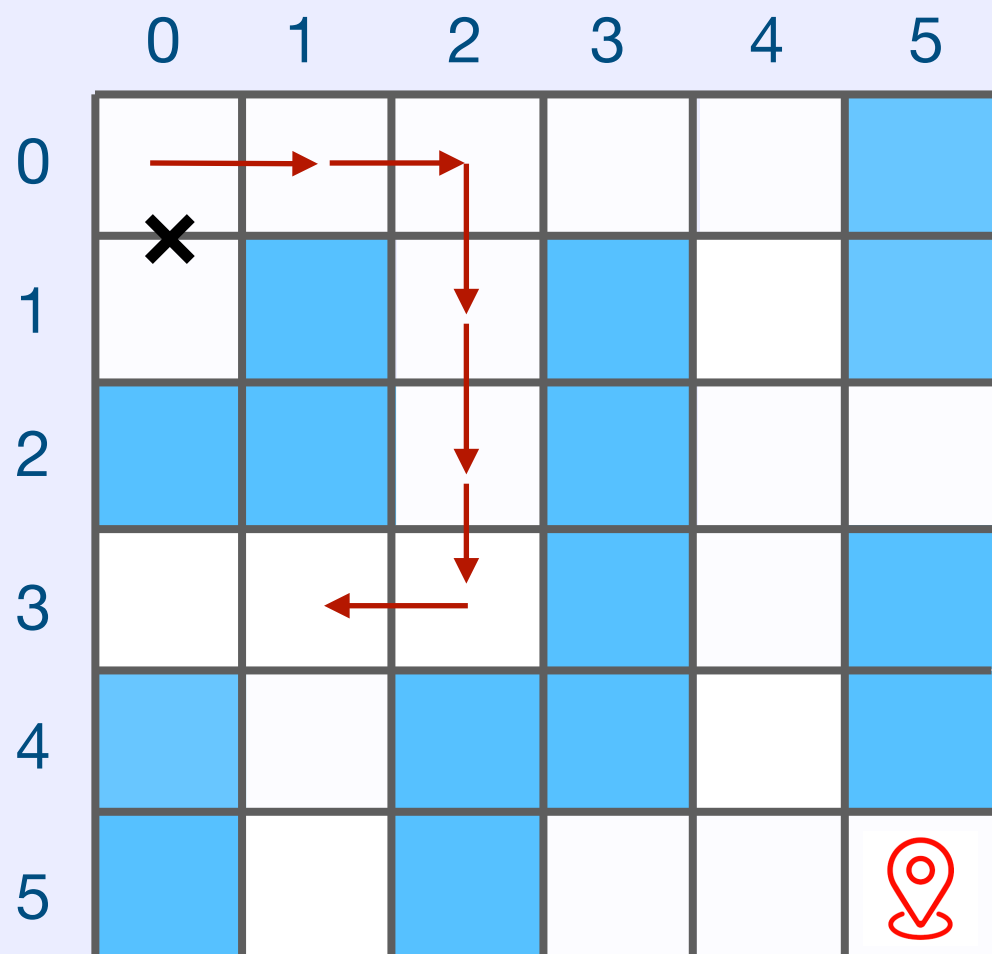
(Cells corresponding to the current path)



(3,2)	Explored = None	Par = T
(2,2)	Explored = None	Par = T
(1,2)	Explored = None	Par = T
(0,2)	Explored = None	Par = L
(0,1)	Explored = None	Par = L
(0,0)	Explored = B	Par = -/-

Stack

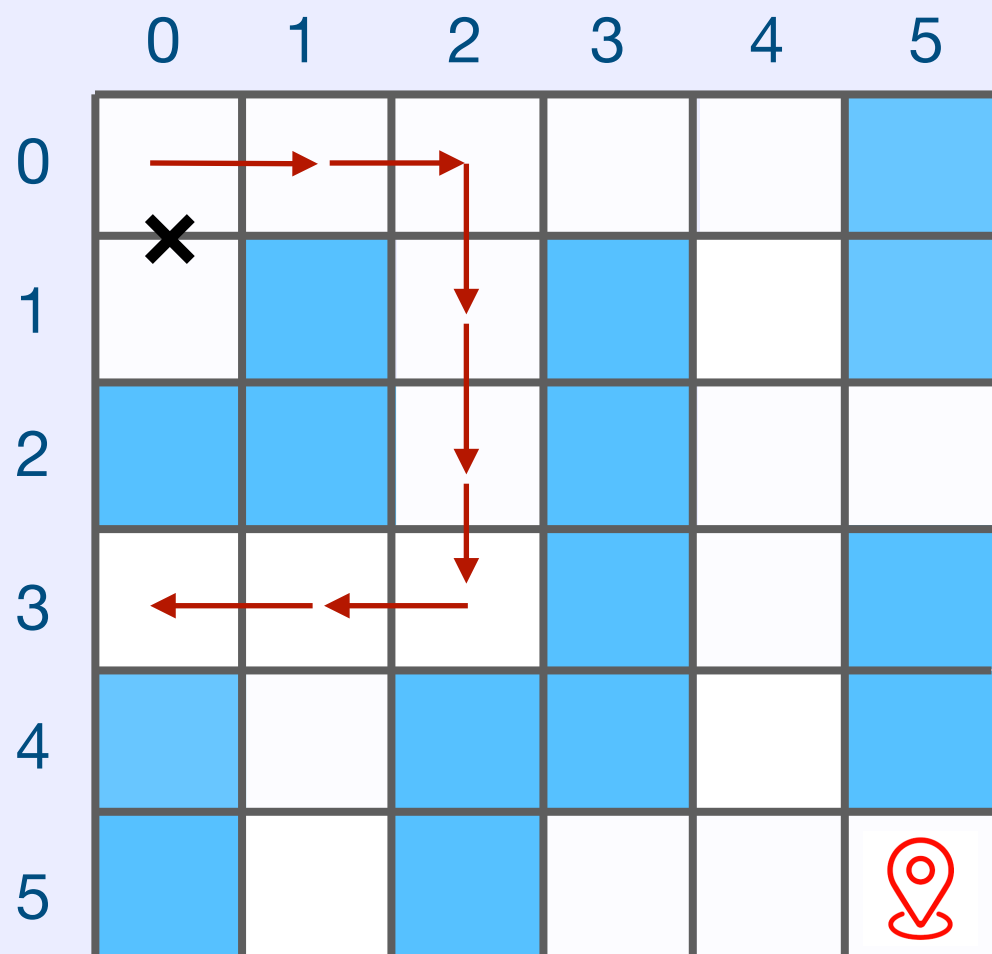
(Cells corresponding to the current path)



(3,1)	Explored = None	Par = R
(3,2)	Explored = None	Par = T
(2,2)	Explored = None	Par = T
(1,2)	Explored = None	Par = T
(0,2)	Explored = None	Par = L
(0,1)	Explored = None	Par = L
(0,0)	Explored = B	Par = -/-

Stack

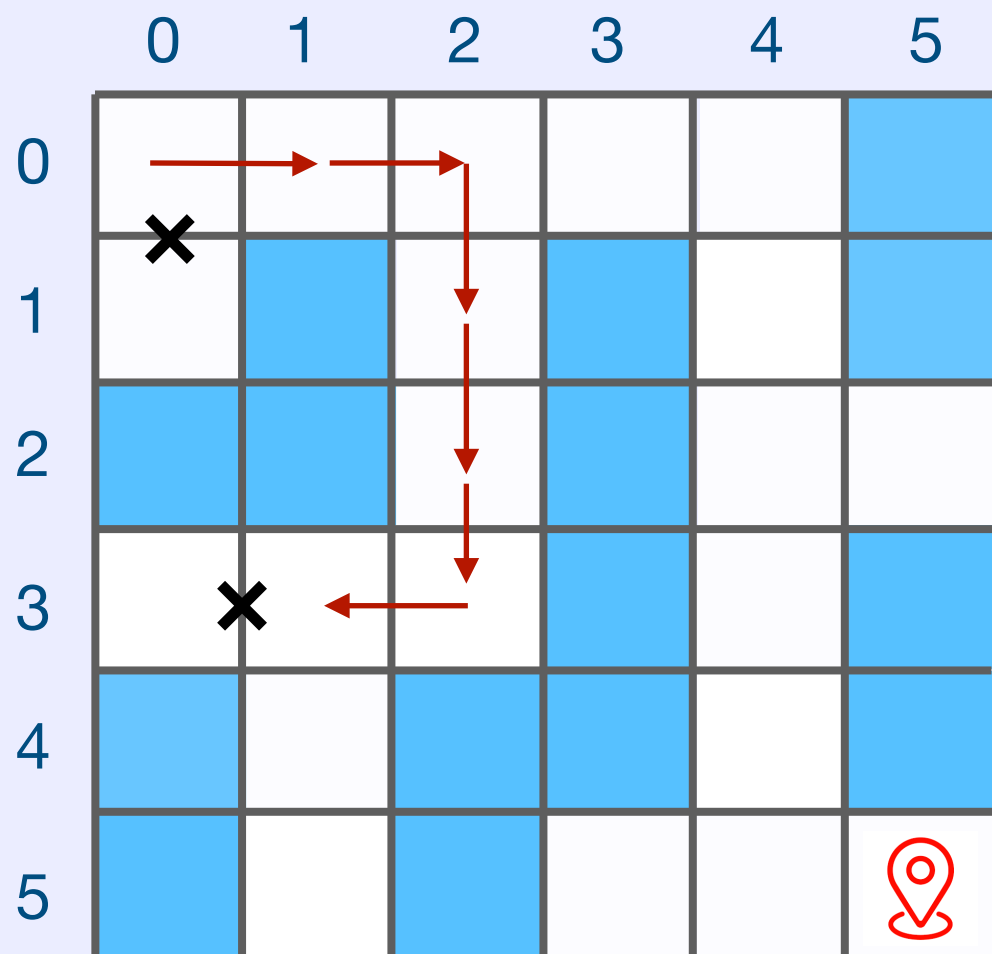
(Cells corresponding to the current path)



(3,0)	Explored = None	Par = R
(3,1)	Explored = None	Par = R
(3,2)	Explored = None	Par = T
(2,2)	Explored = None	Par = T
(1,2)	Explored = None	Par = T
(0,2)	Explored = None	Par = L
(0,1)	Explored = None	Par = L
(0,0)	Explored = B	Par = -/-

Stack

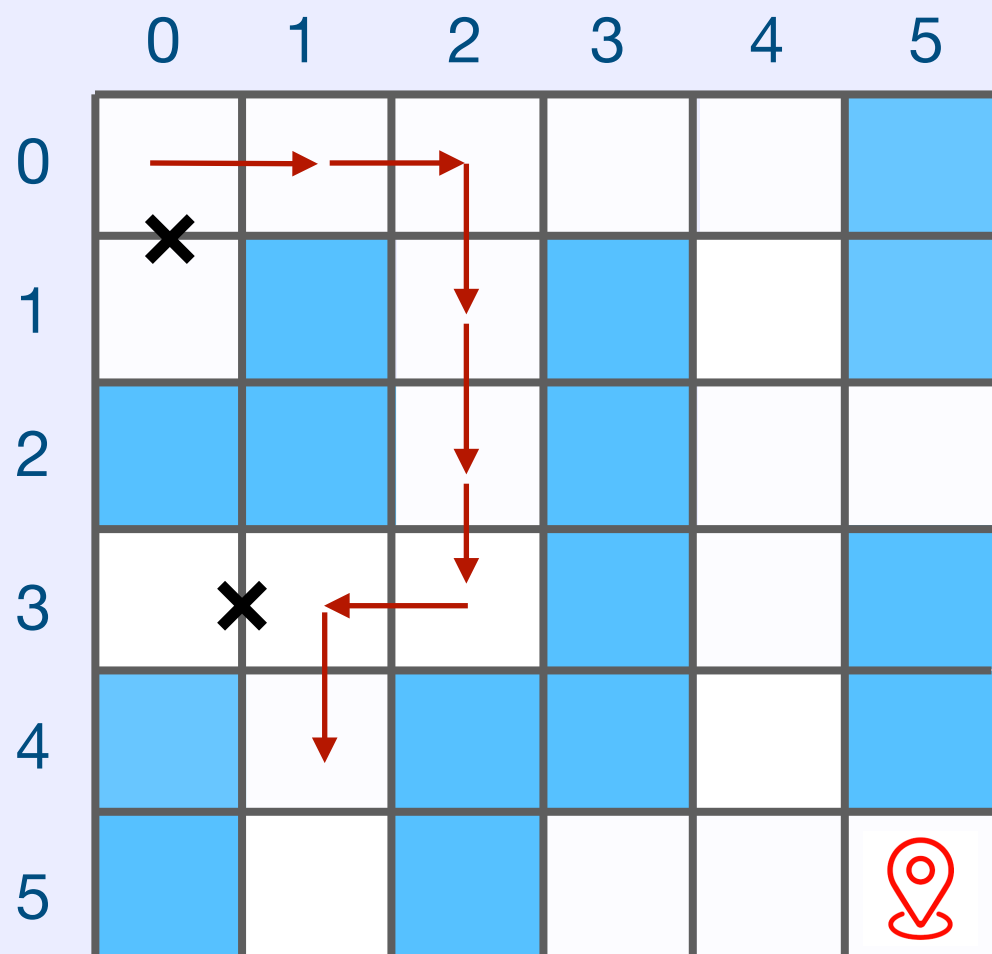
(Cells corresponding to the current path)



(3,1)	Explored = L	Par = R
(3,2)	Explored = None	Par = T
(2,2)	Explored = None	Par = T
(1,2)	Explored = None	Par = T
(0,2)	Explored = None	Par = L
(0,1)	Explored = None	Par = L
(0,0)	Explored = B	Par = -/-

Stack

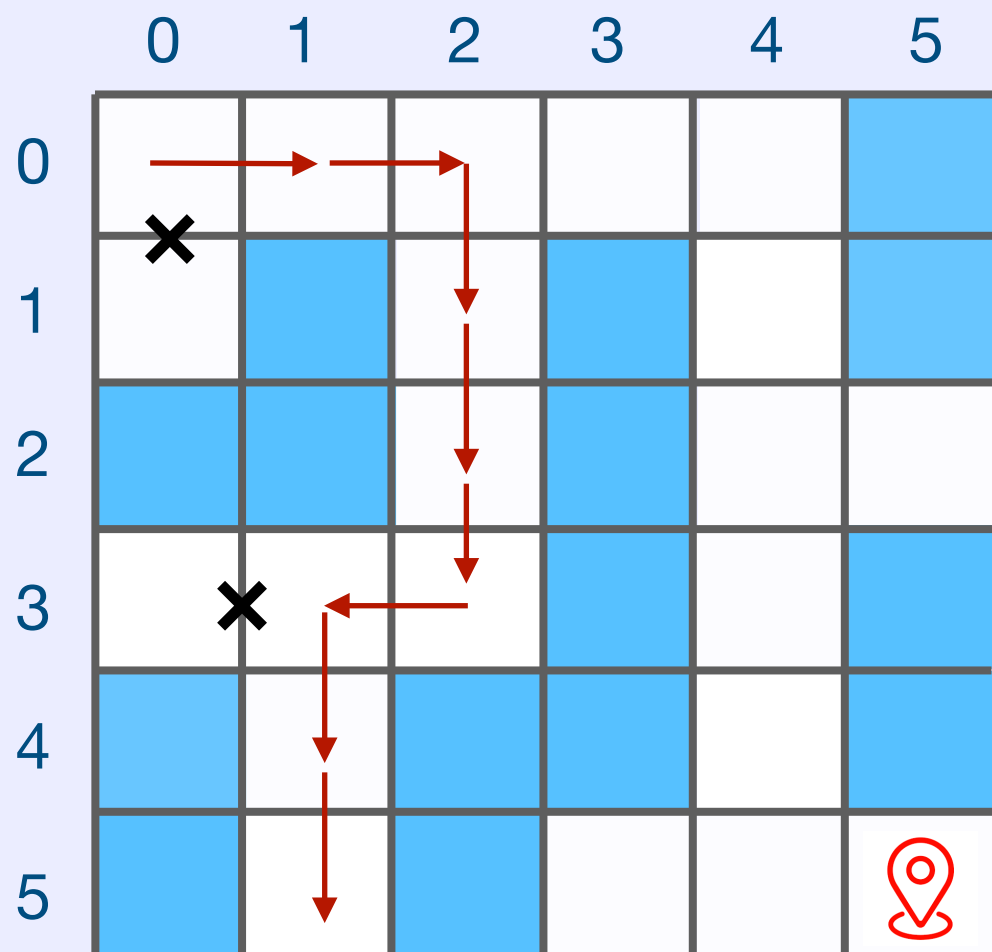
(Cells corresponding to the current path)



(4,1)	Explored = None	Par = T
(3,1)	Explored = L	Par = R
(3,2)	Explored = None	Par = T
(2,2)	Explored = None	Par = T
(1,2)	Explored = None	Par = T
(0,2)	Explored = None	Par = L
(0,1)	Explored = None	Par = L
(0,0)	Explored = B	Par = -/-

Stack

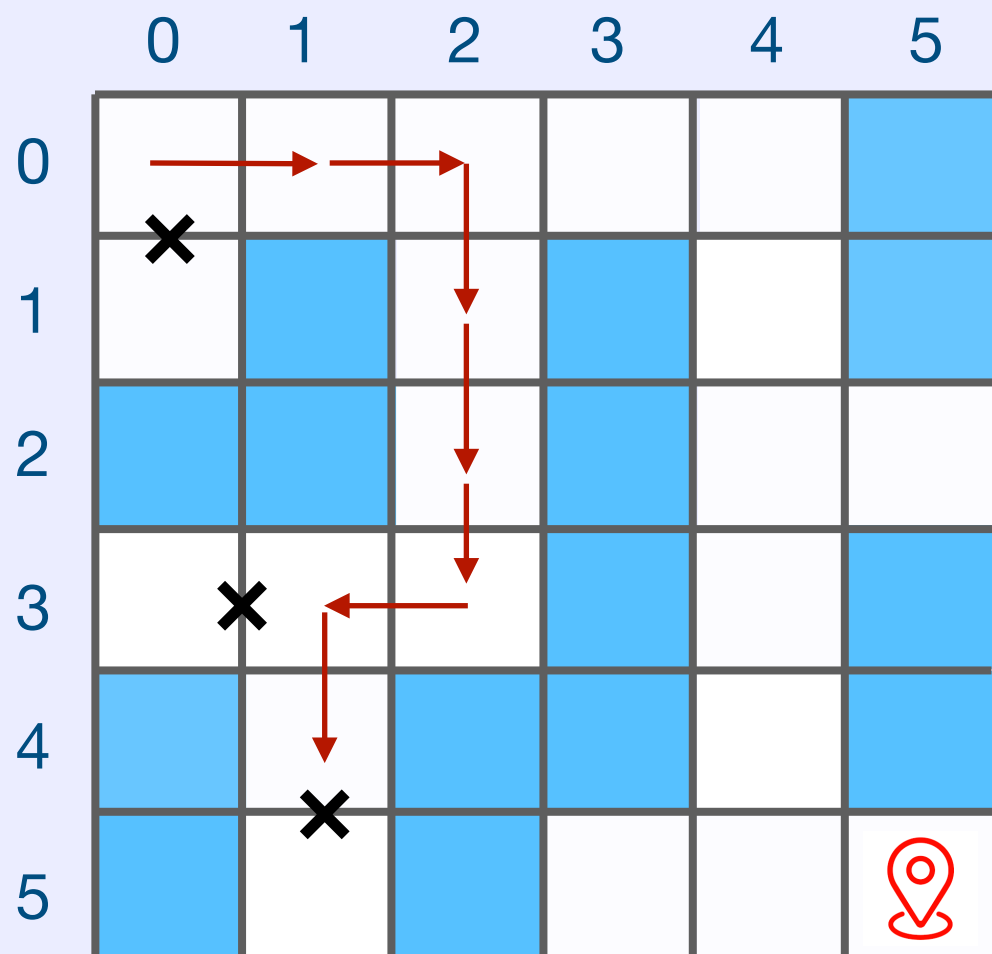
(Cells corresponding to the current path)



(4,2)	Explored = None	Par = T
(4,1)	Explored = None	Par = T
(3,1)	Explored = L	Par = R
(3,2)	Explored = None	Par = T
(2,2)	Explored = None	Par = T
(1,2)	Explored = None	Par = T
(0,2)	Explored = None	Par = L
(0,1)	Explored = None	Par = L
(0,0)	Explored = B	Par = -/-

Stack

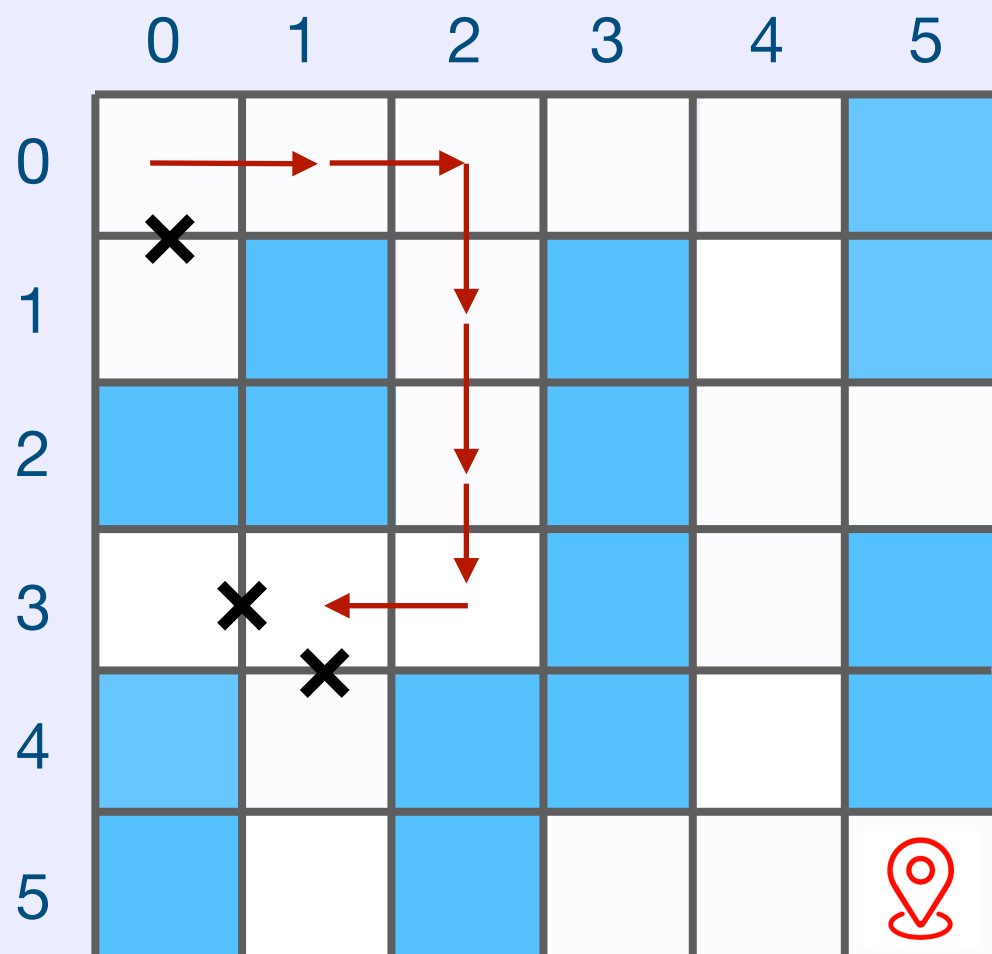
(Cells corresponding to the current path)



(4,1)	Explored = B	Par = T
(3,1)	Explored = L	Par = R
(3,2)	Explored = None	Par = T
(2,2)	Explored = None	Par = T
(1,2)	Explored = None	Par = T
(0,2)	Explored = None	Par = L
(0,1)	Explored = None	Par = L
(0,0)	Explored = B	Par = -/-

Stack

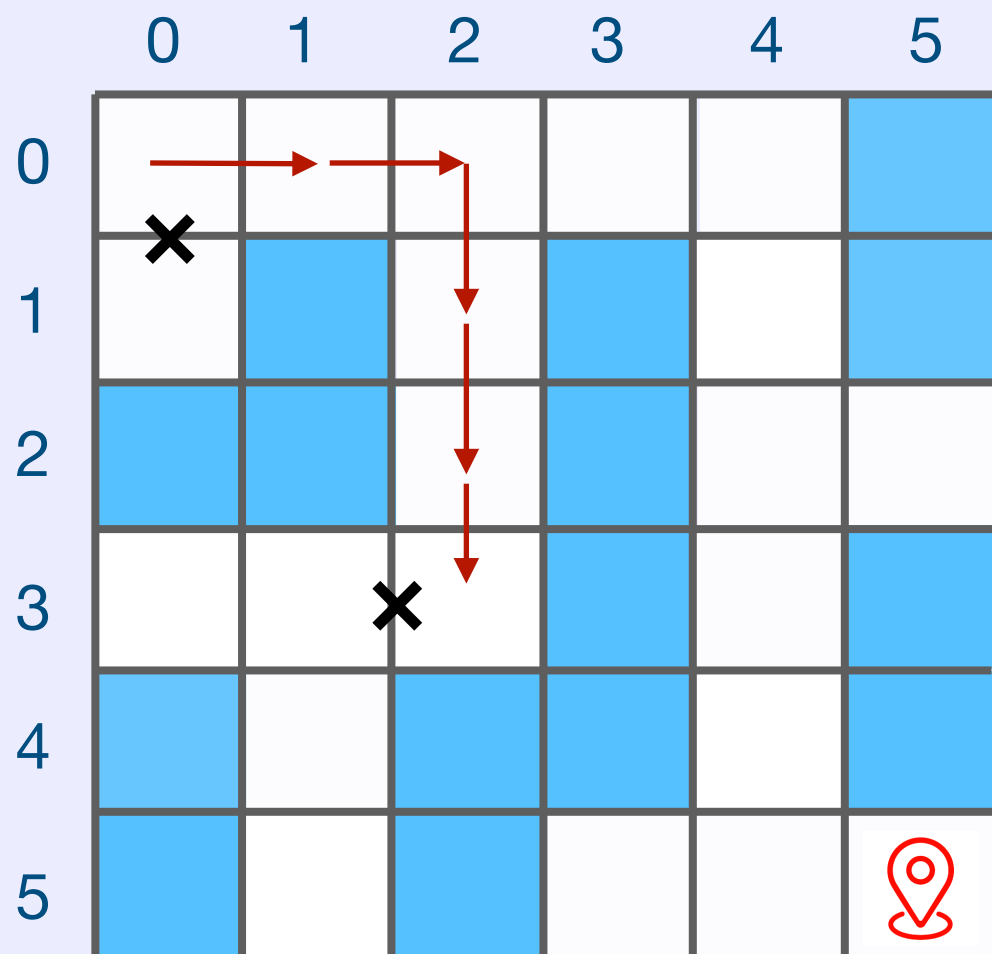
(Cells corresponding to the current path)



(3,1)	Explored = L, B	Par = R
(3,2)	Explored = None	Par = T
(2,2)	Explored = None	Par = T
(1,2)	Explored = None	Par = T
(0,2)	Explored = None	Par = L
(0,1)	Explored = None	Par = L
(0,0)	Explored = B	Par = -/-

Stack

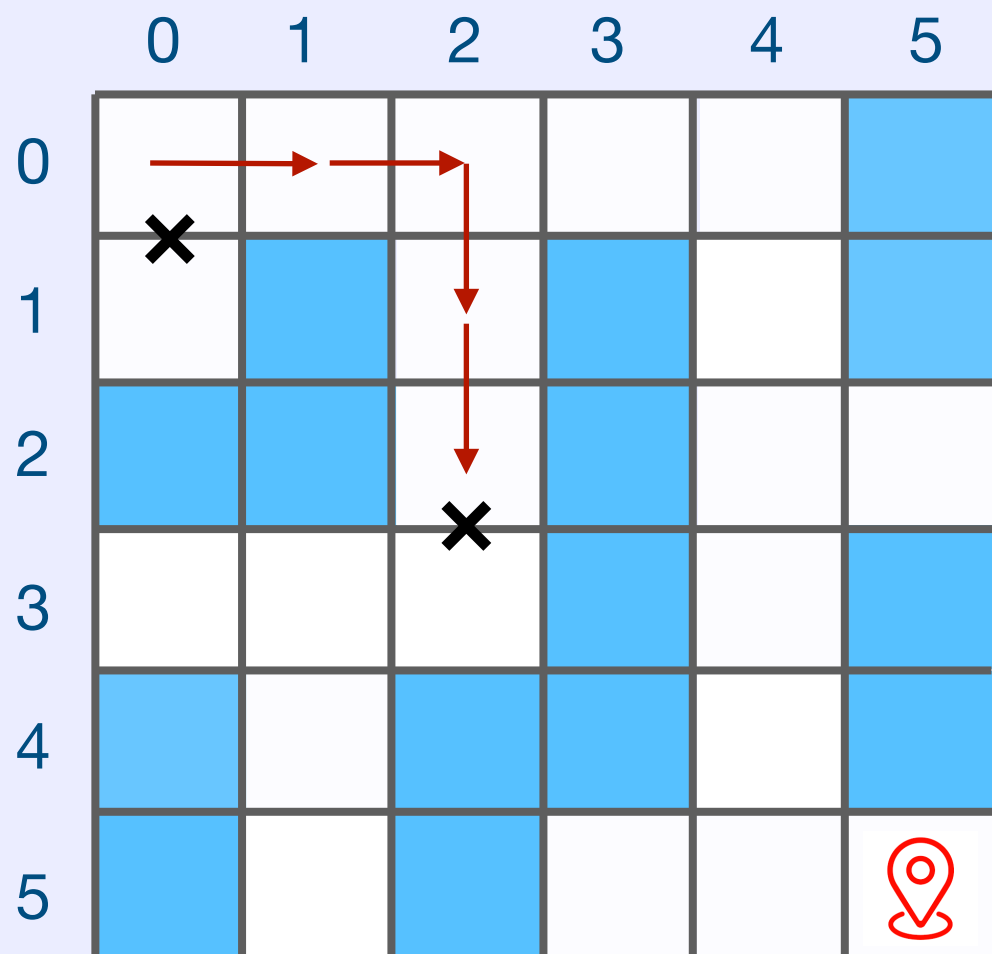
(Cells corresponding to the current path)



(3,2)	Explored = L	Par = T
(2,2)	Explored = None	Par = T
(1,2)	Explored = None	Par = T
(0,2)	Explored = None	Par = L
(0,1)	Explored = None	Par = L
(0,0)	Explored = B	Par = -/-

Stack

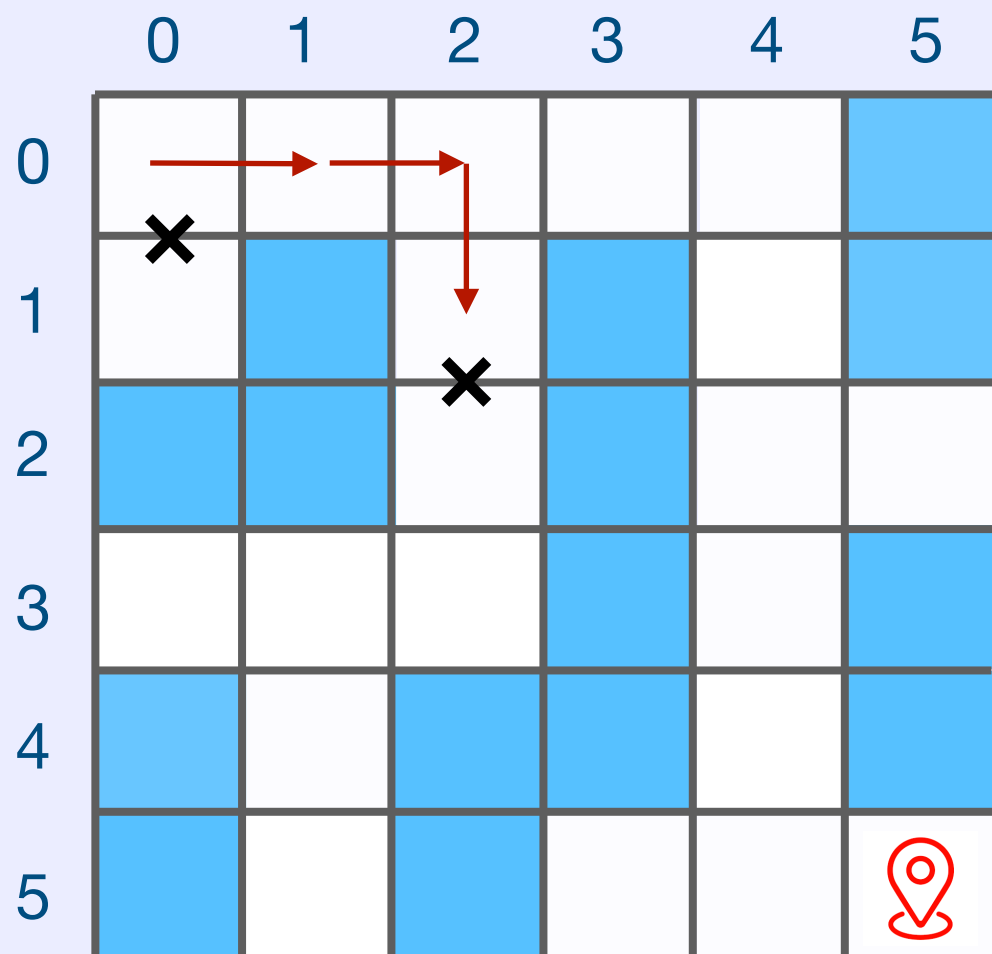
(Cells corresponding to the current path)



(2,2)	Explored = B	Par = T
(1,2)	Explored = None	Par = T
(0,2)	Explored = None	Par = L
(0,1)	Explored = None	Par = L
(0,0)	Explored = B	Par = -/-

Stack

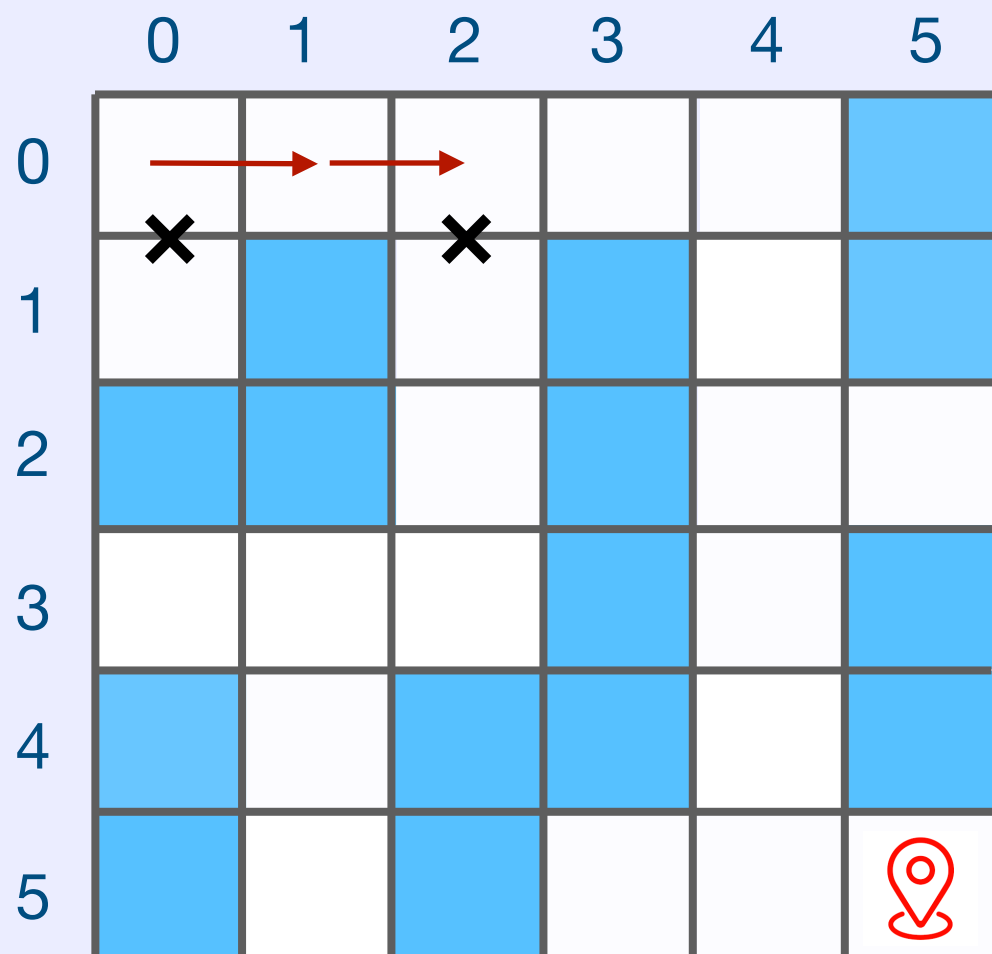
(Cells corresponding to the current path)



(1,2)	Explored = B	Par = T
(0,2)	Explored = None	Par = L
(0,1)	Explored = None	Par = L
(0,0)	Explored = B	Par = -/-

Stack

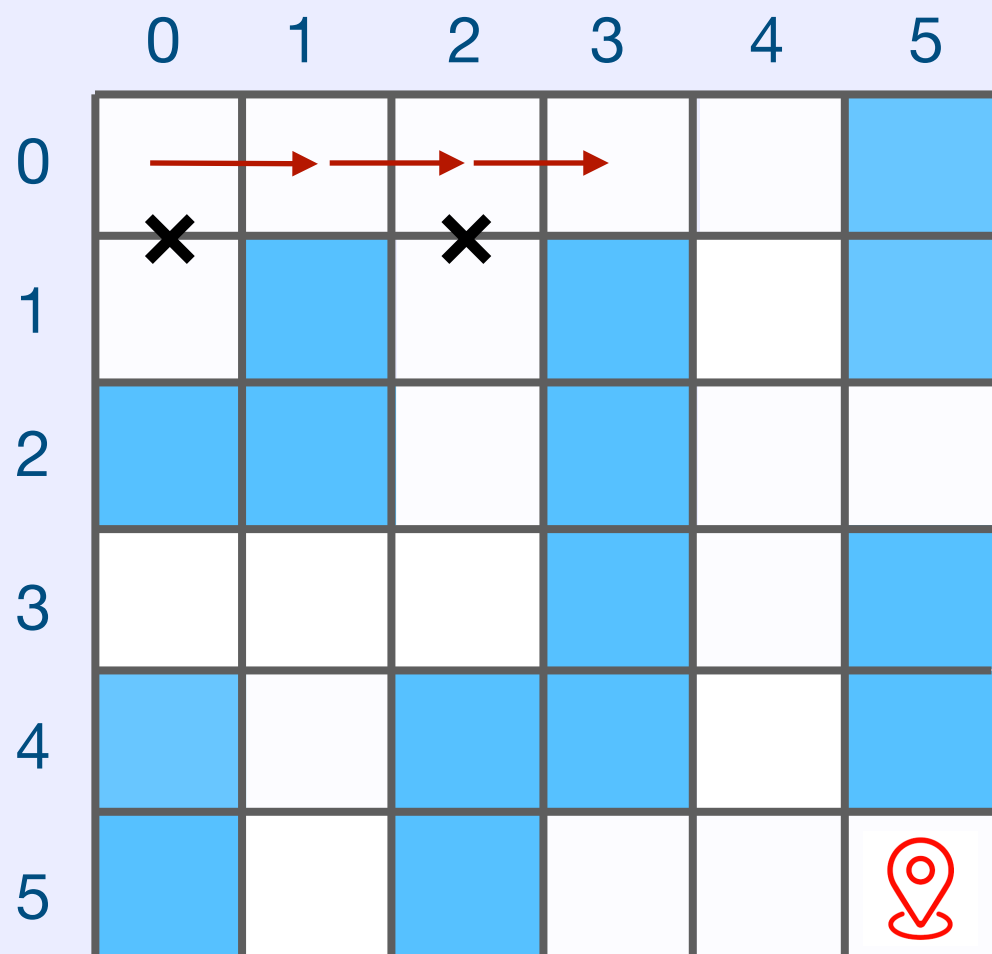
(Cells corresponding to the current path)



(0,2)	Explored = B	Par = L
(0,1)	Explored = None	Par = L
(0,0)	Explored = B	Par = -/-

Stack

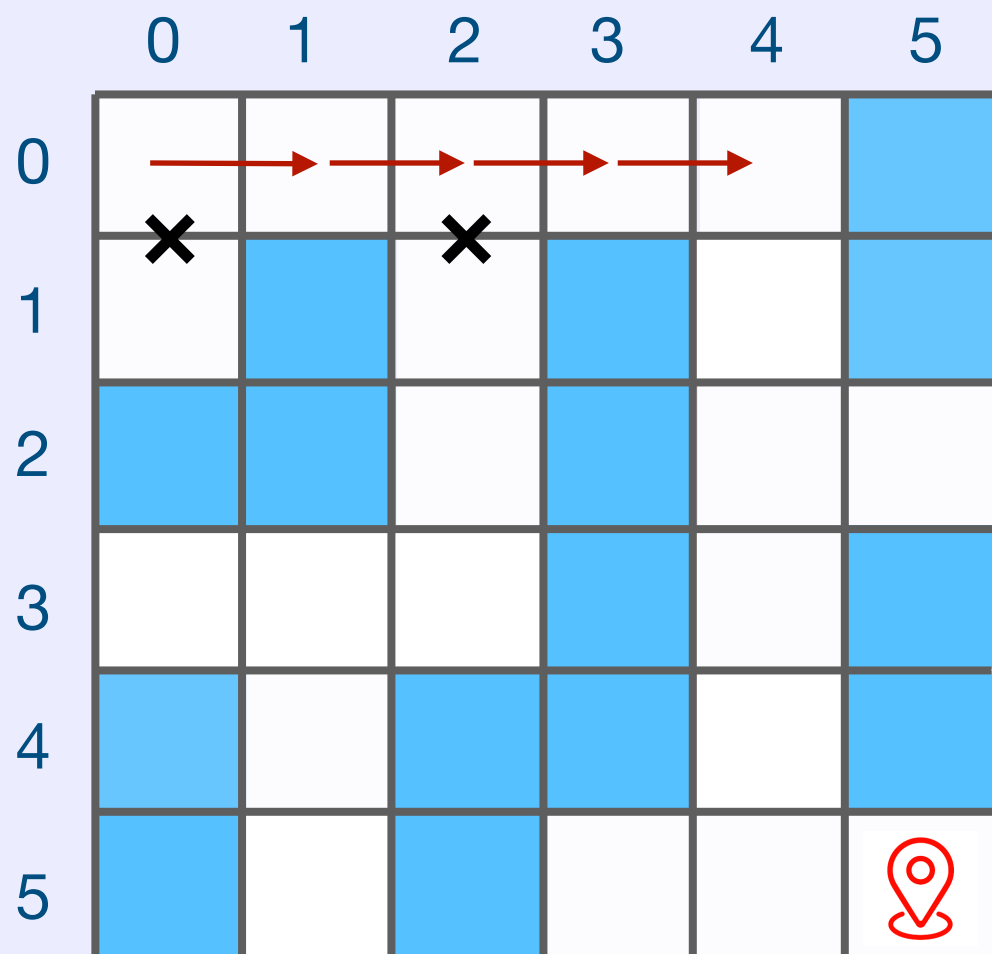
(Cells corresponding to the current path)



(0,3)	Explored = None	Par = L
(0,2)	Explored = B	Par = L
(0,1)	Explored = None	Par = L
(0,0)	Explored = B	Par = -/-

Stack

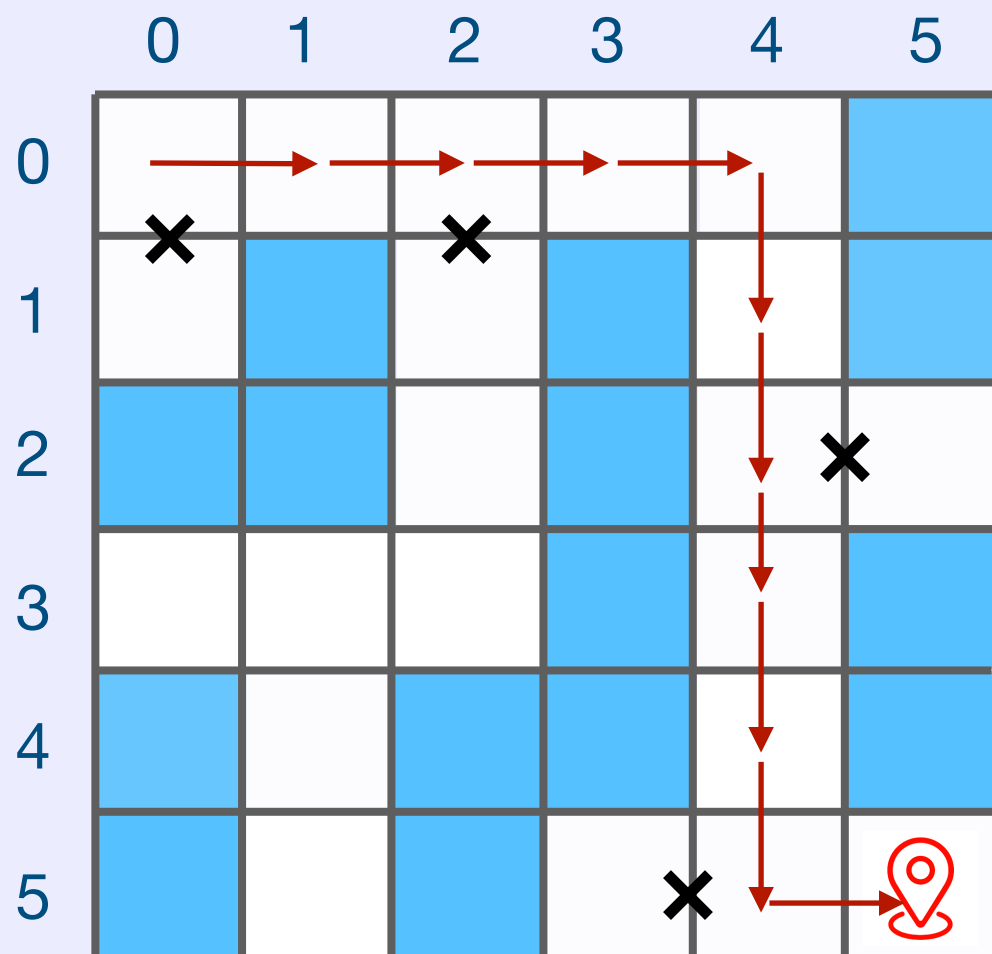
(Cells corresponding to the current path)



(0,4)	Explored = None	Par = L
(0,3)	Explored = None	Par = L
(0,2)	Explored = B	Par = L
(0,1)	Explored = None	Par = L
(0,0)	Explored = B	Par = -/-

Stack

(Cells corresponding to the current path)



(5,5)	Explored = None	Par = L
-------	-----------------	---------

⋮

(0,4)	Explored = None	Par = L
-------	-----------------	---------

(0,3)	Explored = None	Par = L
-------	-----------------	---------

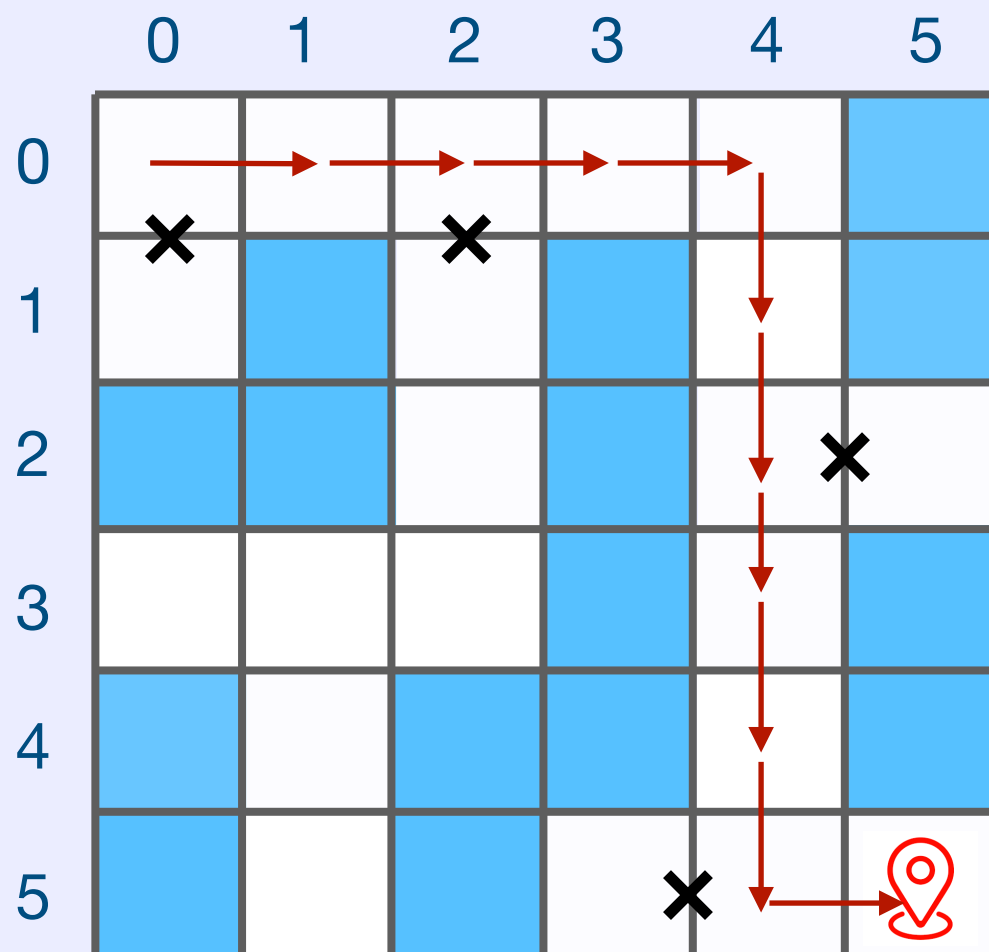
(0,2)	Explored = B	Par = L
-------	--------------	---------

(0,1)	Explored = None	Par = L
-------	-----------------	---------

(0,0)	Explored = B	Par = -/-
-------	--------------	-----------

Stack

(Cells corresponding to the current path)



Remark: The algorithm may enter into a cyclic loop, so you will need to maintain an array of visited cells to ensure that no cell is visited twice.

(5,5)	Explored = None	Par = L
⋮		
(0,4)	Explored = None	Par = L
(0,3)	Explored = None	Par = L
(0,2)	Explored = B	Par = L
(0,1)	Explored = None	Par = L
(0,0)	Explored = B	Par = -/-

Stack

(Cells corresponding to the current path)

Homework

Write pseudocode to solve maze using stacks.

