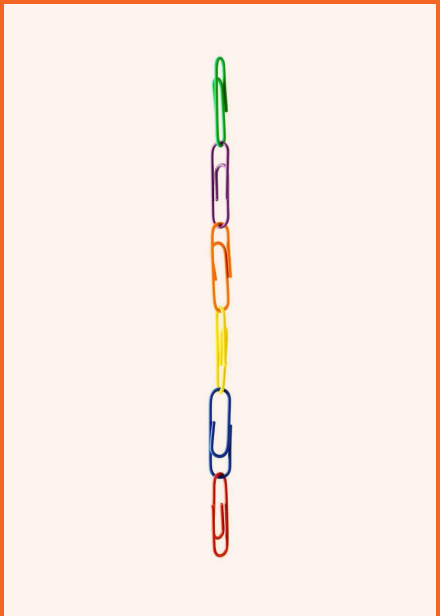
# Data Structures (I)




Stack
Queue
Linked List

# What are data structures?

A data structure is a particular way of organizing data so that they can be used efficiently

# What are data structures?

A data structure is a particular way of organizing data so that they can be used <u>efficiently</u>

unorganized

well organized

# Contents

Abstract Data Types - implemented by data structures

- Stack - LIFO
- Queue - FIFO

Data structures

- Array
- Linked List
  - Singly linked list
  - Doubly linked list
  - Circular linked list
  - XOR linked list

# Stack

A stack of books

# Stack

We can

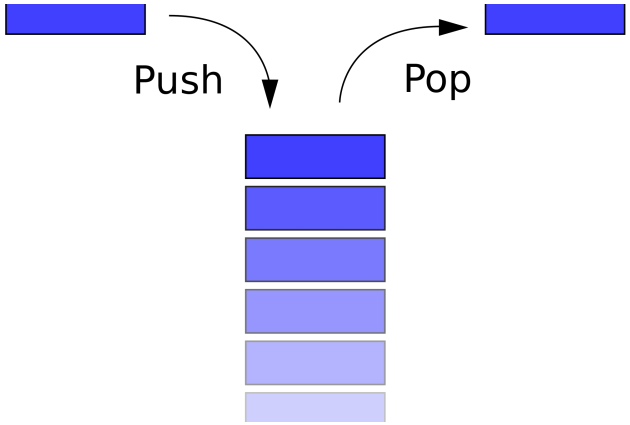
- put a book on top
- remove the top book

We cannot

- insert a book in the middle
- remove a book in the middle

A stack of books

# Stack




We can

- put an element on top
- remove the top element

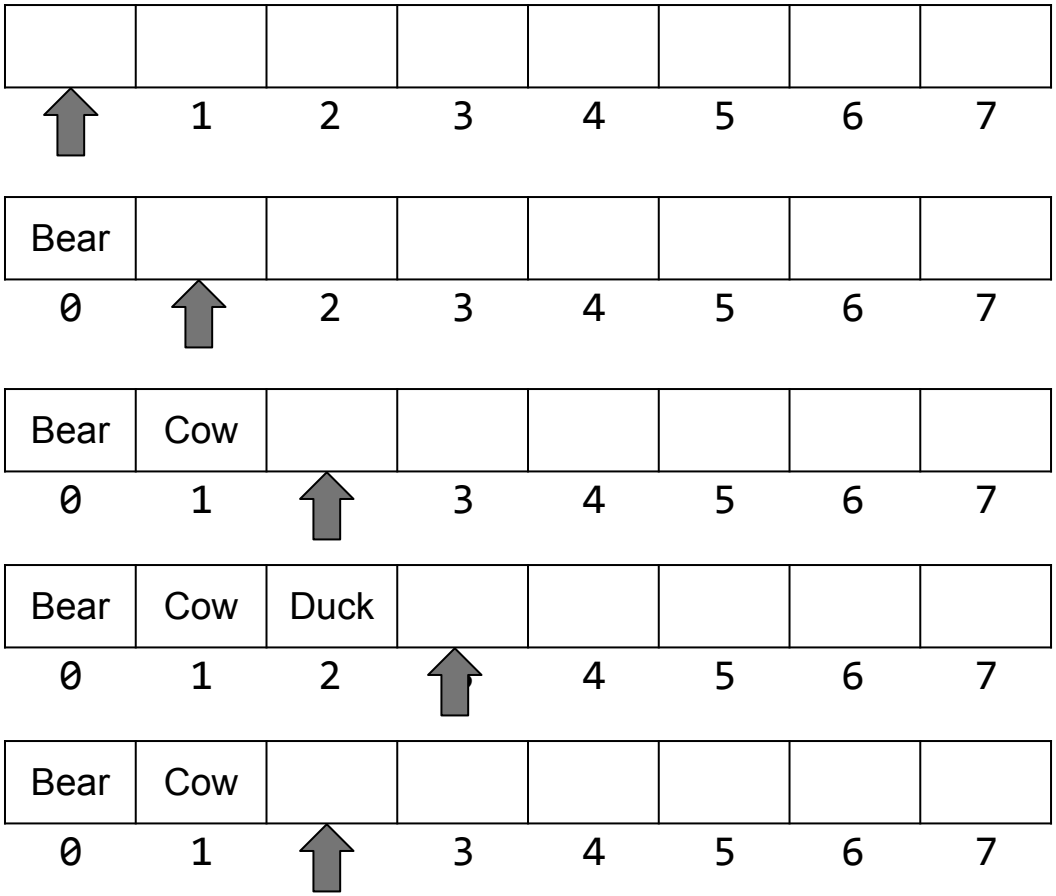
We cannot

- insert an element in the middle
- remove an element in the middle

Output order

- Last In First Out (LIFO)

# Implementation



# Why not use array?

- Array is too powerful, easier to make mistakes
- Stack is so simple, you can't make a mistake
- Stack is easier to understand than array

# Parentheses balance

Determine if a string of 1 <= N <= 10000 parentheses is balanced.

input

```
{()[]}
```

output

```
Yes
```

input

```
{(][)}
```

output

```
No
```

# Parentheses balance

Maintain a stack,

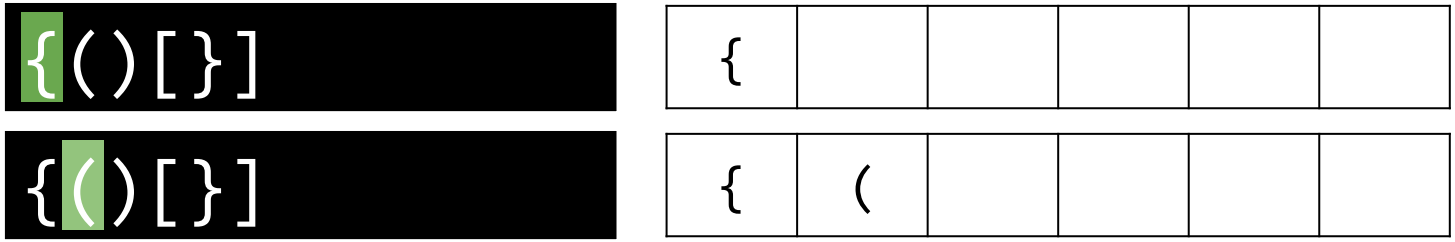
left parentheses → push stack
right parentheses → pop stack


| | | | | | |
|---|---|---|---|---|---|
| { | | | | | |

# Parentheses balance

Maintain a stack,

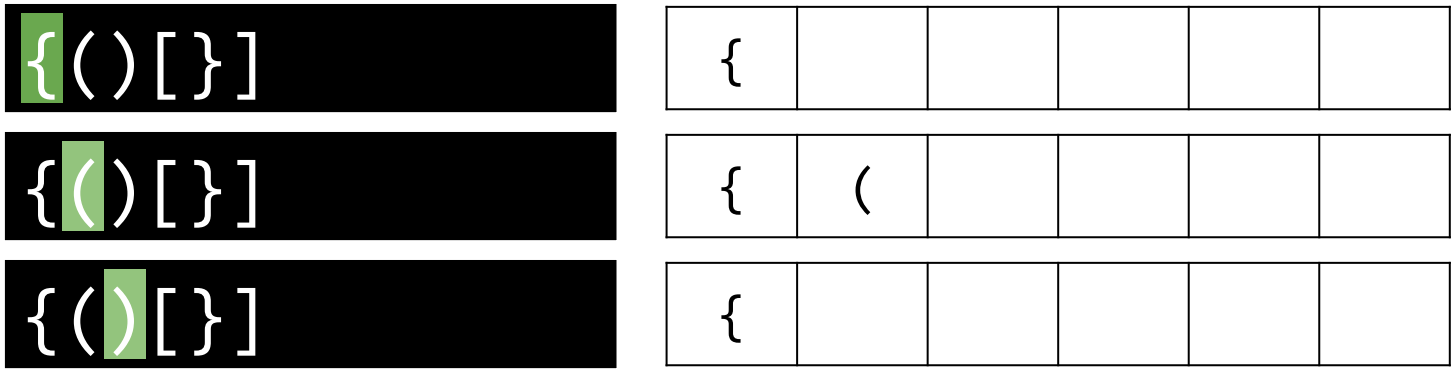
left parentheses → push stack
right parentheses → pop stack



# Parentheses balance

Maintain a stack,

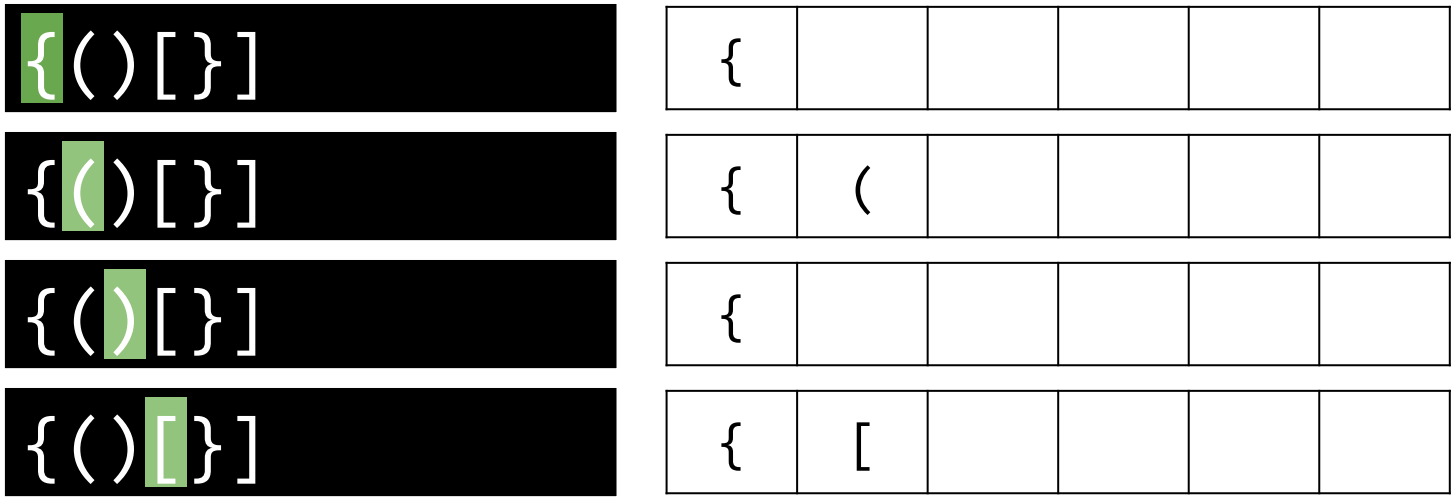
left parentheses → push stack
right parentheses → pop stack



# Parentheses balance

Maintain a stack,

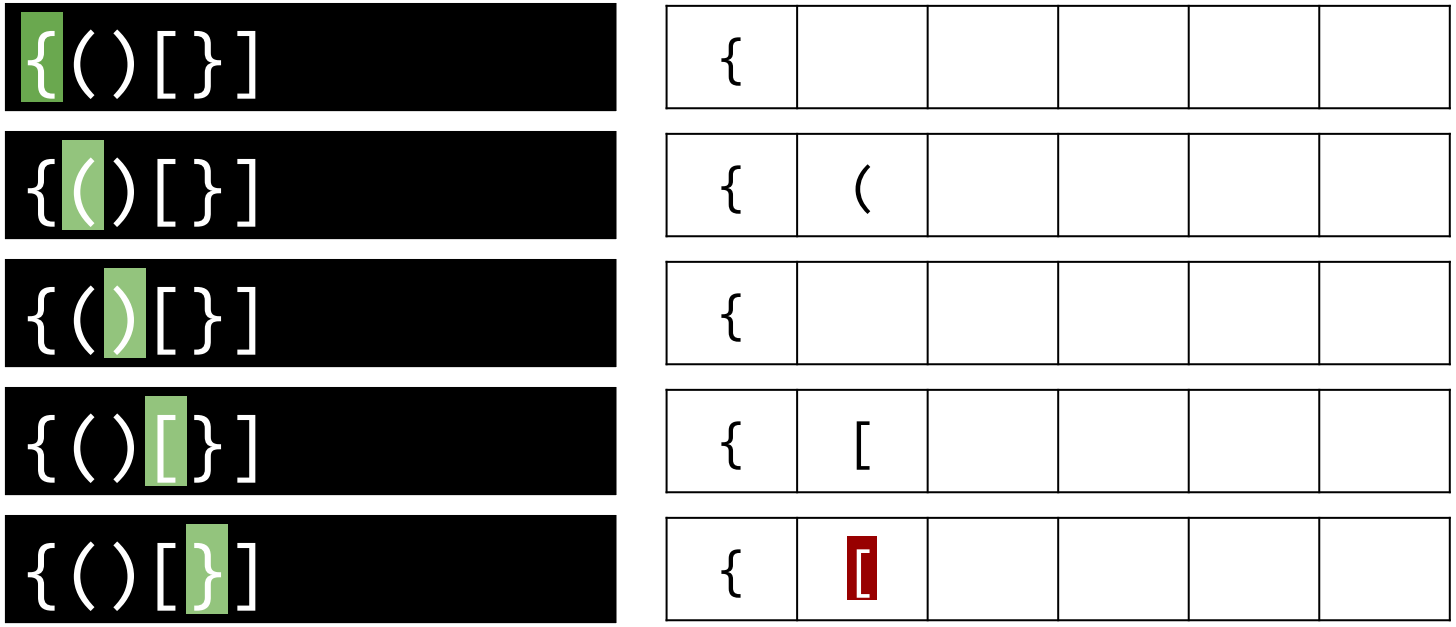
left parentheses → push stack
right parentheses → pop stack



# Parentheses balance

Maintain a stack,

left parentheses → push stack
right parentheses → pop stack



# Parentheses balance

How many 'push' were performed?

```
{()[}]
```

3

```
{()[]{}}
```

You tell me

```
()[()}{()]()
```

# Parentheses balance

How many 'push' were performed?

| | |
|---|---|
| `{()[}]` | 3 |
| `{()[]{}}` | 4 |
| `()[()}{()]()` | You tell me |

# Parentheses balance

How many 'push' were performed?

| | |
|---|---|
| `{()[}]` | 3 |
| `{()[]{}}` | 4 |
| `()[()}{()]()` | 3 |

# Parentheses balance

Time complexity: O(N)

Space complexity: O(N)

# Expression evaluation

input

```
1+2*3-4
```

output

```
3
```

# Expression

Infix notation
Operator **between** operands

```
1+2
```

# Expression

Note that we will only deal with single digits in the following slides, so "12" should be interpreted as "one two" rather than "twelve"

Prefix notation
(Polish Notation)
Operator **before** operands

```
+12
```

Infix notation
Operator **between** operands

```
1+2
```

Postfix notation
(Reverse Polish Notation)
Operator **after** operands

```
12+
```

# Expression

Prefix notation
(Polish Notation)
Operator **before** operands

```
-+1*234
```

Infix notation
Operator **between** operands

```
1+2*3-4
```

Postfix notation
(Reverse Polish Notation)
Operator **after** operands

```
You tell me
```

# Expression

Prefix notation
(Polish Notation)
Operator **before** operands

```
-+1*234
```

Infix notation
Operator **between** operands

```
1+2*3-4
```

Postfix notation
(Reverse Polish Notation)
Operator **after** operands

```
123*+4-
```

# Expression

Prefix notation
(Polish Notation)
Operator **before** operands

```
You tell me
```

Infix notation
Operator **between** operands

```
(1+2)*3-4
```

Postfix notation
(Reverse Polish Notation)
Operator **after** operands

# Expression

Prefix notation
(Polish Notation)
Operator **before** operands

```
-*+1234
```

Infix notation
Operator **between** operands

```
(1+2)*3-4
```

Postfix notation
(Reverse Polish Notation)
Operator **after** operands

```
You tell me
```

# Expression

Prefix notation
(Polish Notation)
Operator **before** operands

```
-*+1234
```

Infix notation
Operator **between** operands

```
(1+2)*3-4
```

Postfix notation
(Reverse Polish Notation)
Operator **after** operands

```
12+3*4-
```

No parentheses are needed for both Polish notations

# Prefix notation evaluation

input

```
+*23-41
```

output

```
You tell me
```

# Prefix notation evaluation

input `+*23-41`

output `9`

# Prefix notation evaluation

Push to stack one by one, evaluate the top three elements whenever possible.

+*23-41

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| + | | | | | | | |

# Prefix notation evaluation

Push to stack one by one, evaluate the top three elements whenever possible.

`+*23-41`

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| + | | | | | | | |

`+*23-41`

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| + | * | | | | | | |

# Prefix notation evaluation

Push to stack one by one, evaluate the top three elements whenever possible.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| +*23-41 | + | | | | | | | |
| +*23-41 | + | * | | | | | | |
| +*23-41 | + | * | 2 | | | | | |

# Prefix notation evaluation

Push to stack one by one, evaluate the top three elements whenever possible.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| +*23-41 | + | | | | | | | |
| +*23-41 | + | * | | | | | | |
| +*23-41 | + | * | 2 | | | | | |
| +*23-41 | + | * | 2 | 3 | | | | |
| | + | 6 | | | | | | |

# Prefix notation evaluation

Push to stack one by one, evaluate the top three elements whenever possible.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| +*23-41 | + | | | | | | | |
| +*23-41 | + | * | | | | | | |
| +*23-41 | + | * | 2 | | | | | |
| +*23-41 | + | * | 2 | 3 | | | | |
| | + | 6 | | | | | | |
| +*23-41 | + | 6 | - | | | | | |

# Prefix notation evaluation

Push to stack one by one, evaluate the top three elements whenever possible.

+*23-41

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| + | 6 | - | 4 | | | | |

# Prefix notation evaluation

Push to stack one by one, evaluate the top three elements whenever possible.

```
+*23-41
```

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| + | 6 | - | 4 | | | | |

```
+*23-41
```

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| + | 6 | - | 4 | 1 | | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| + | 6 | 3 | | | | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 9 | | | | | | | |

# Postfix notation evaluation

input

```
23*41-+
```

We can use the same approach

# Infix notation evaluation

input

```
4+2*(3-1)
```

We have to handle

- Operator precedence `*` `/` `+` `-`
- Parentheses

Maintain 2 stacks, one for operators and one for operands

# Other applications

- Recursion
  - Exhaust permutations
  - Exhaust combinations
  - taught in **Recursion, Divide and Conquer**
- Depth-first search (DFS)
  - solve a maze
  - will be taught in **Graph (I)**

# Practice time

- HKOJ 01015 Parentheses Balance
- HKOJ 01033 Simple Arithmetic

# Queue


A queue of human

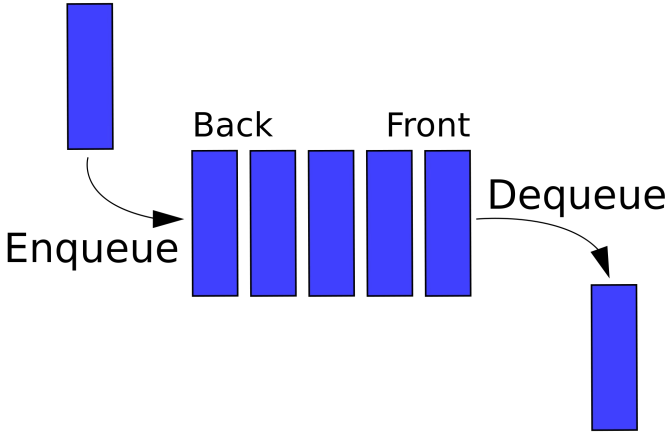
We can

- put a new comer at the back
- serve the customer in the front

We cannot

- insert a new comer in the middle
- serve a customer in the middle

# Queue




We can

- put an element at the back
- remove the front element

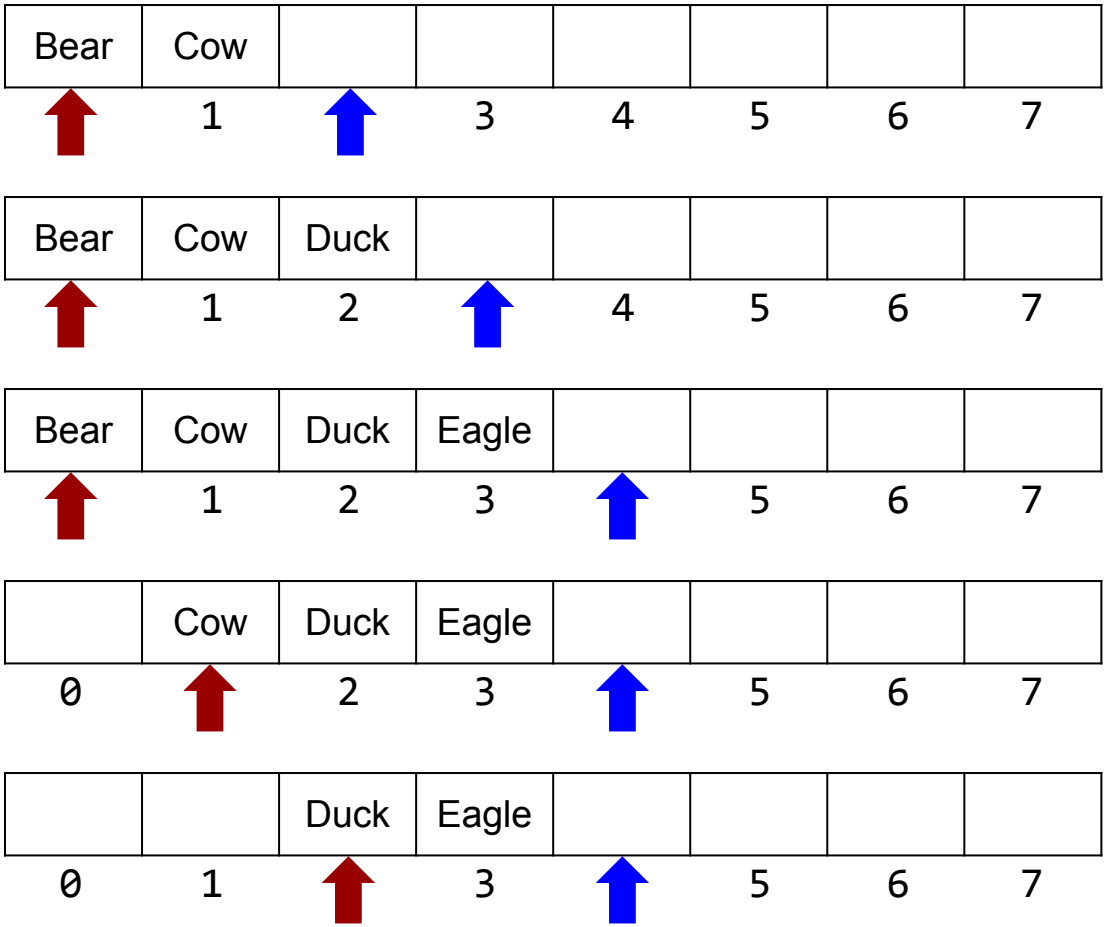
We cannot

- insert an element in the middle
- remove an element in the middle

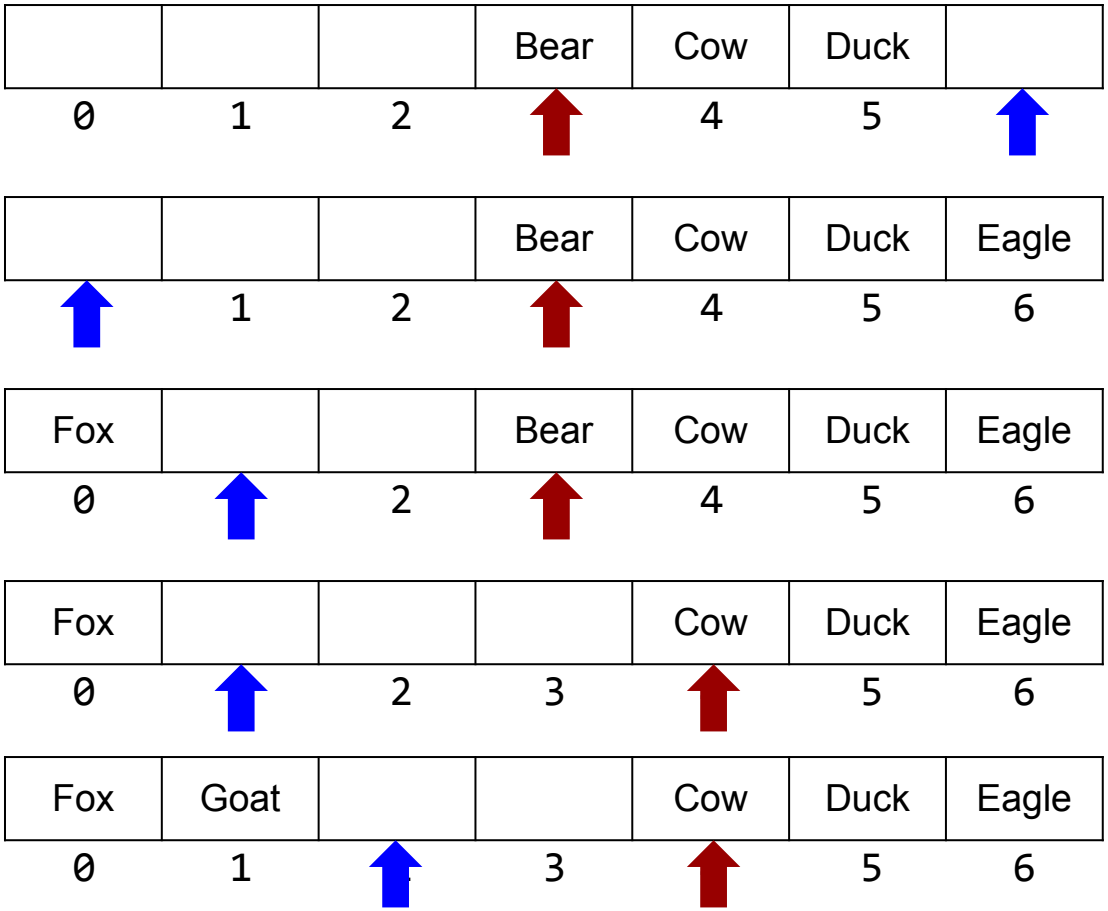
Output order

- First In First Out (FIFO)

# Implemention



# Implemention - circular buffer



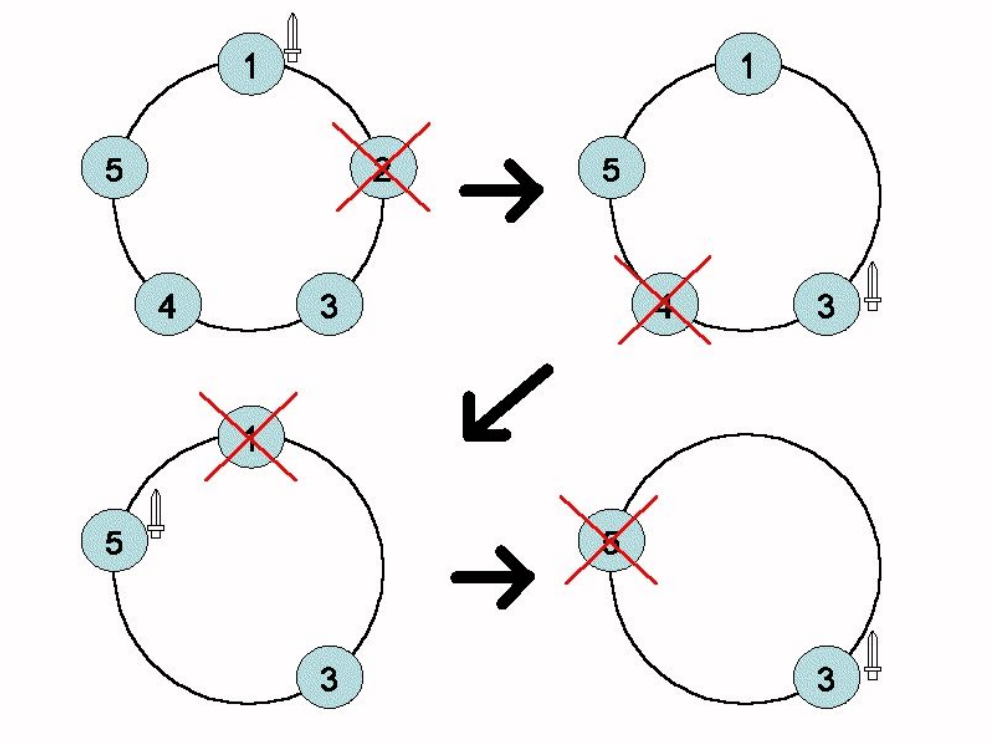
# Why not array?

Again,

- Queue is much simpler to understand
- Queue avoid mistakes
- Problems that can be solved by queue is definitely easier than problems that can only be solved by array

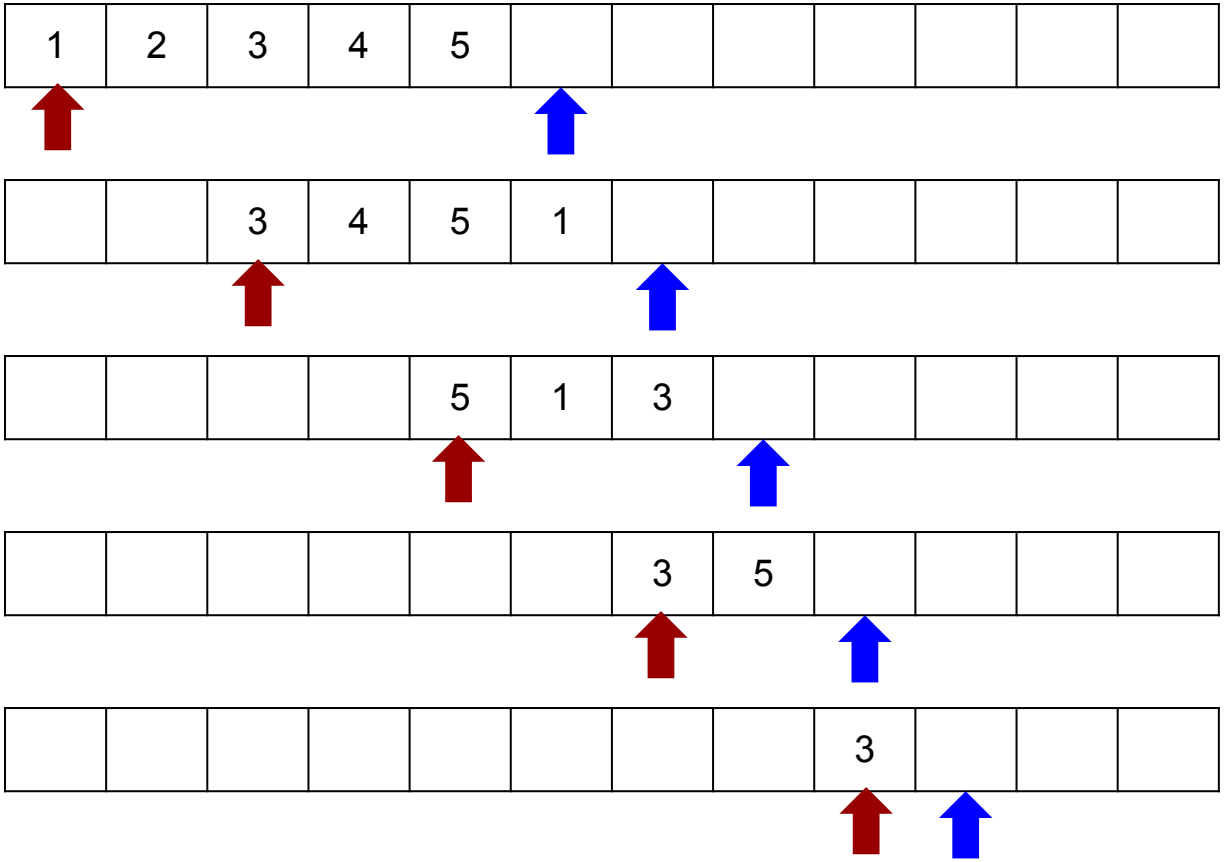
# The Josephus Problem

Given 1<=N<=1000 people arranged in a circle and 1 sword. The one holding a sword will kill the one on his left and pass the sword to the left. Who will survive?



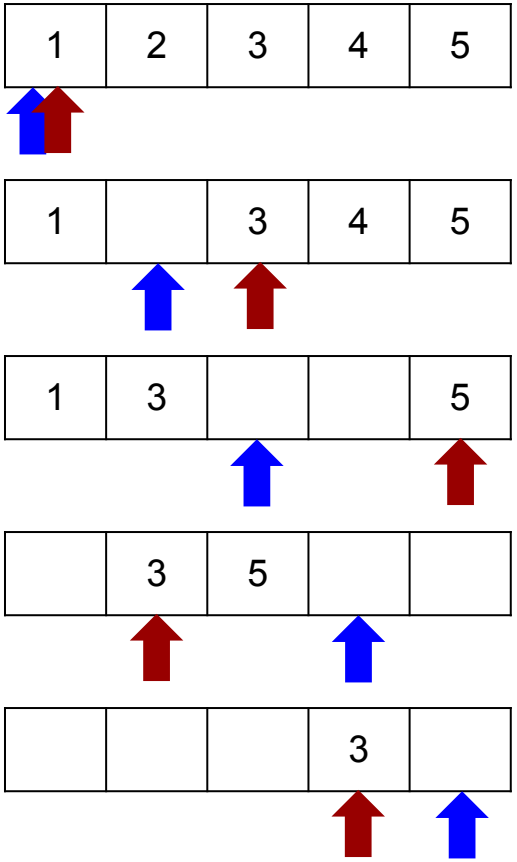
# The Josephus Problem

Enqueue 1 to N, repeat {Enqueue(Dequeue()); Dequeue();}



# The Josephus Problem

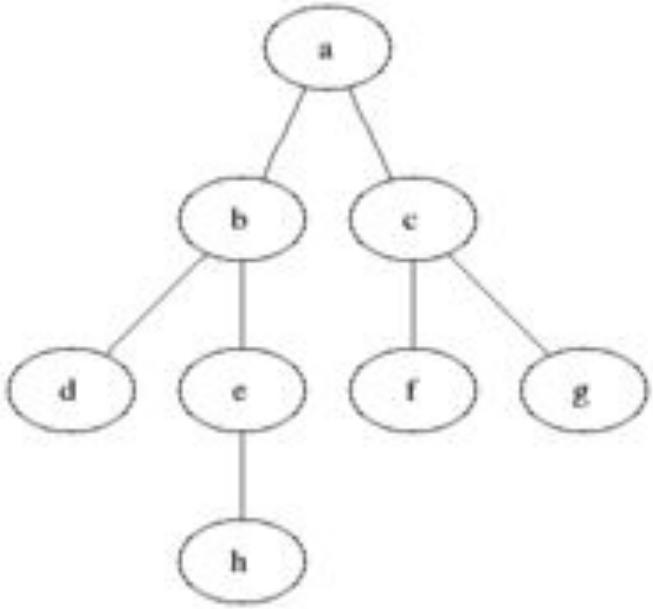
Using circular queue



# Breath-first search (BFS)

Grey nodes are in queue

Will be taught in Graph (I)



# Practice time

- HKOJ P005 Rails
- HKOJ 01017 Car Sorter
- HKOJ 01030 The Josephus Problem

# Linked list

a chain of paper clips

We can

- insert a paper clip anywhere
- remove a paper clip anywhere

# Linked list

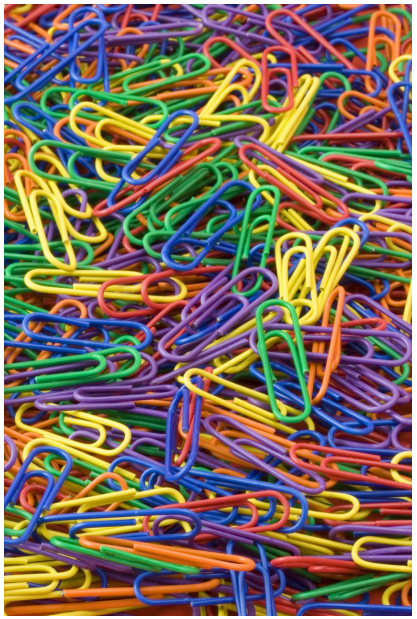
a chain of paper clips

We can

- insert a paper clip anywhere
- remove a paper clip anywhere

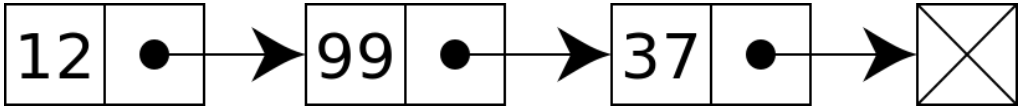
We cannot

- locate the $n^{th}$ paper clip quickly

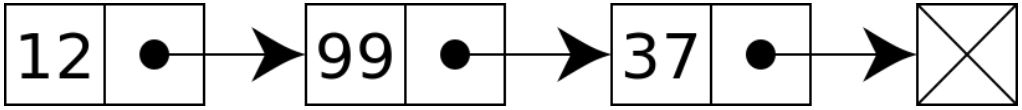


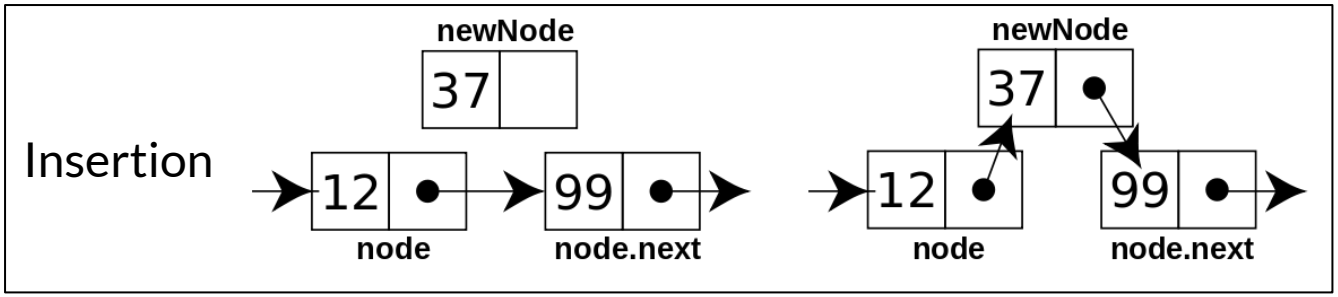
a long chain of paper clips

# Linked list

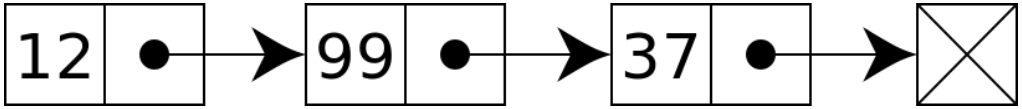


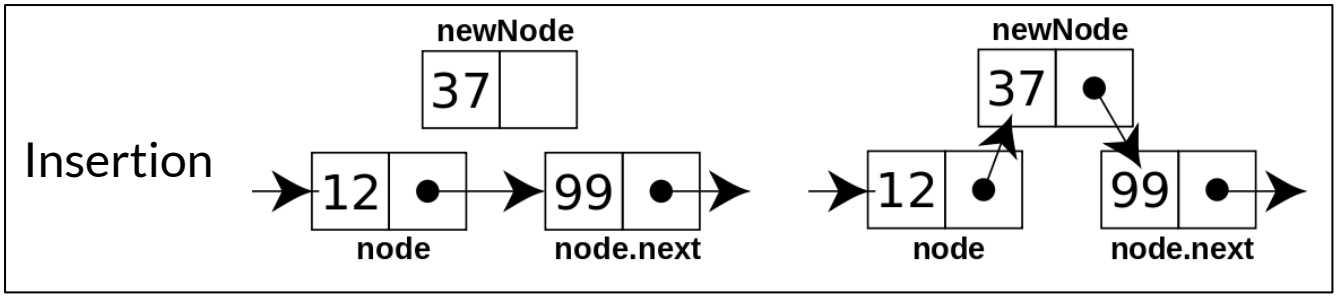
# Linked list

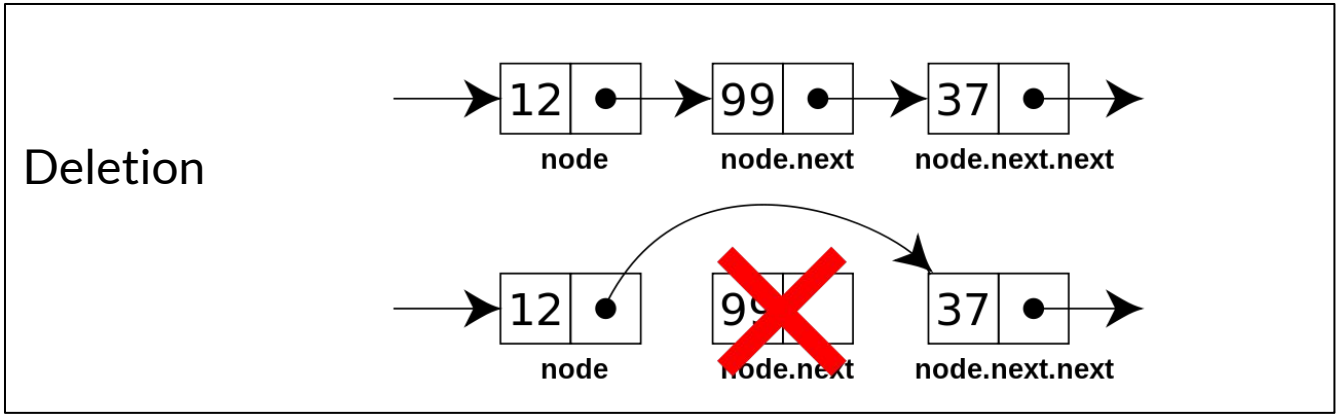

12
99
37


Insertion
newNode
37
12
99
node
node.next
newNode
37
12
99
node
node.next

# Linked list


12
99
37


Insertion
newNode
37
12
99
node
node.next
newNode
37
12
99
node
node.next

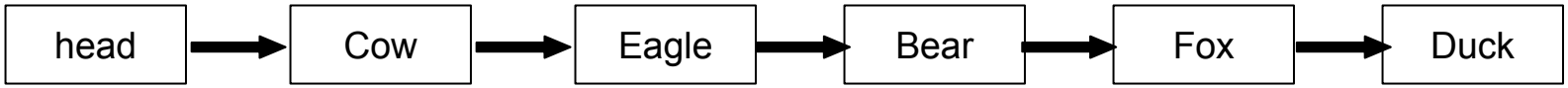

Deletion
12
99
37
node
node.next
node.next.next
12
99
37
node
node.next
node.next.next

# Implementation

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **value** | | Bear | Cow | Duck | Eagle | Fox | | |
| **next** | 2 | 5 | 4 | -1 | 1 | 3 | | |
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

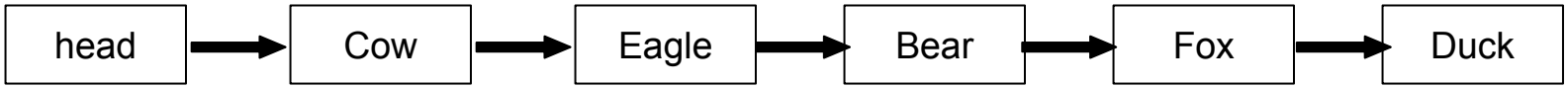
# Implementation

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **value** | | Bear | Cow | Duck | Eagle | Fox | | |
| **next** | 2 | 5 | 4 | -1 | 1 | 3 | | |
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

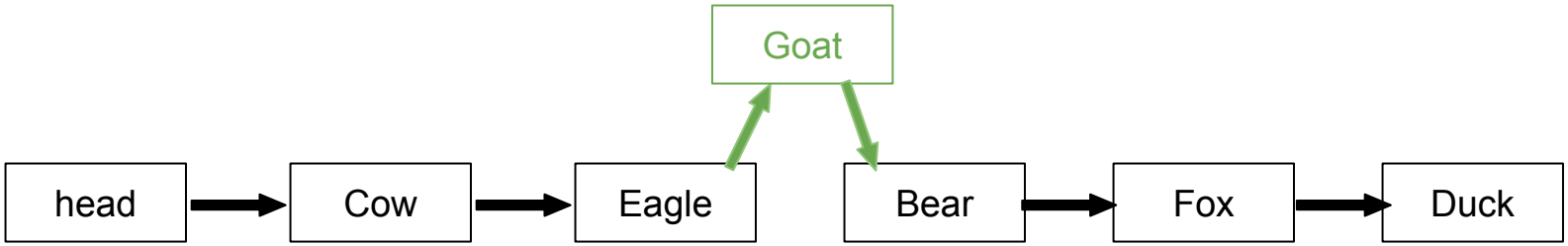


# Implementation

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **value** | | Bear | Cow | Duck | Eagle | Fox | | |
| **next** | 2 | 5 | 4 | -1 | 1 | 3 | | |
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |




| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **value** | | Bear | Cow | Duck | Eagle | Fox | Goat | |
| **next** | 2 | 5 | 4 | -1 | 6 | 3 | 1 | |
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

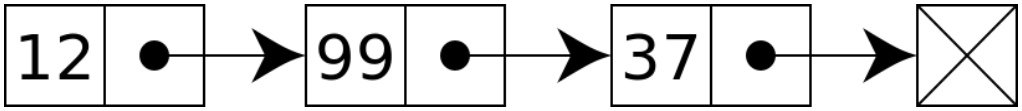


# Linked list vs Array

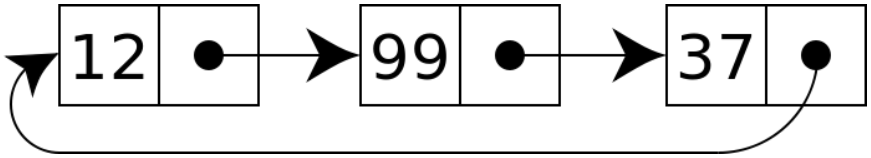
| Time Complexities | Linked list | Array |
|---|---|---|
| Insert at the front | O(1) | O(N) |
| Insert in the middle | O(1) | O(N) |
| Insert at the back | O(1) | O(1) |
| Find an element | O(N) | O(N) |
| Find the $k^{th}$ element | O(k) | O(1) |

# Linked lists

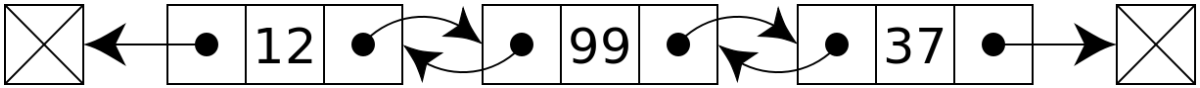
Singly linked list



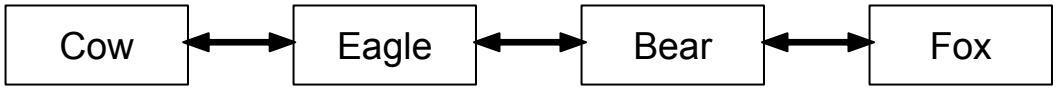

Circularly linked list




Doubly linked list




XOR linked list



# XOR linked list

| | Singly linked list | Doubly linked list | XOR linked list |
|---|---|---|---|
| Space complexity | O(2N) = O(N) | O(3N) = O(N) | O(2N) = O(N) |
| Backward traversal | No | Yes | Yes |

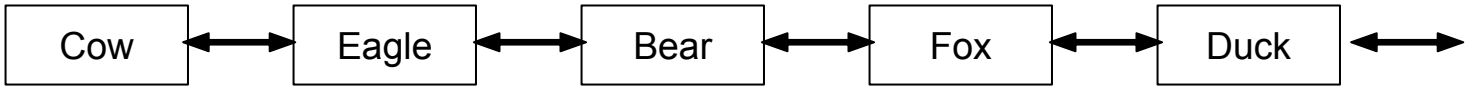
Not necessary in HKOI, I won't teach this today.

Read the slides below by yourself.

# XOR linked list

Recall:
'⊕' stands for XOR
$0 \oplus 0 = 0$
$0 \oplus 1 = 1$
$1 \oplus 0 = 1$
$1 \oplus 1 = 0$
$A \oplus B = B \oplus A$
$\mathbf{A \oplus (B \oplus A) = B}$

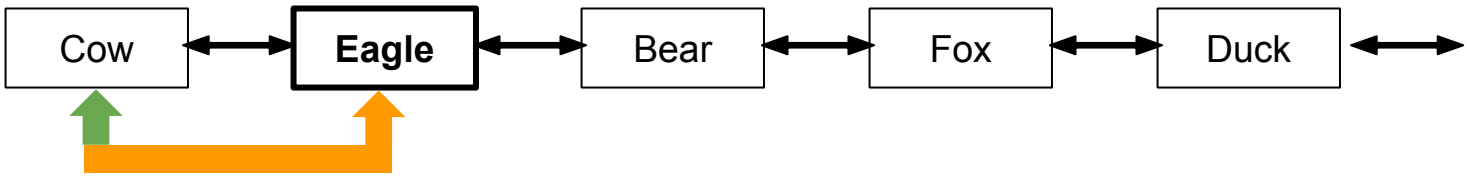
| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **value** | Bear | Cow | Duck | Eagle | Fox | | | |
| **next⊕prev** | 4 ⊕ 3 | 3 ⊕ 2 | 1 ⊕ 4 | 0 ⊕ 1 | 2 ⊕ 0 | | | |
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |




| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **value** | Bear | Cow | Duck | Eagle | Fox | | | |
| **next** | 4 | 3 | 1 | 0 | 2 | | | |
| **prev** | 3 | 2 | 4 | 1 | 0 | | | |
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

# XOR linked list - traverse forward

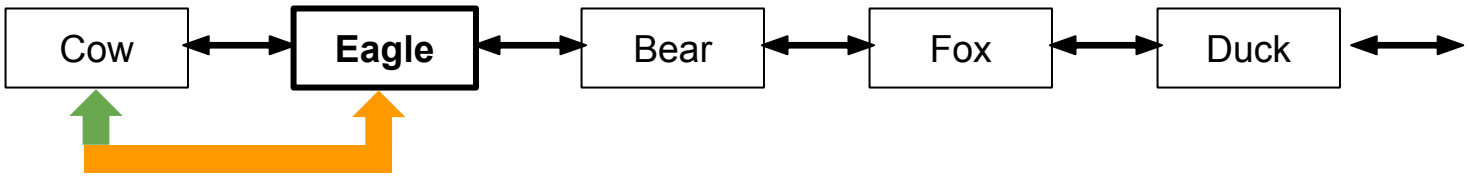
| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **value** | Bear | Cow | Duck | **Eagle** | Fox | | | |
| **next⊕prev** | 4 ⊕ 3 | 3 ⊕ 2 | 1 ⊕ 4 | **0 ⊕ 1** | 2 ⊕ 0 | | | |
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |




1 ⊕ (0 ⊕ 1) = 0

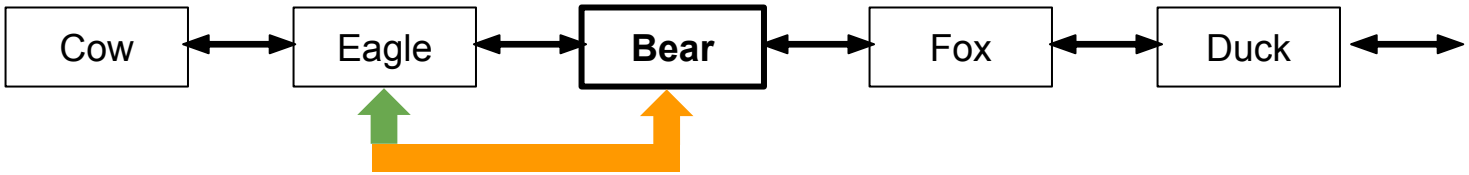
# XOR linked list - traverse forward

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **value** | Bear | Cow | Duck | **Eagle** | Fox | | | |
| **next⊕prev** | 4 ⊕ 3 | 3 ⊕ 2 | 1 ⊕ 4 | **0 ⊕ 1** | 2 ⊕ 0 | | | |
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |



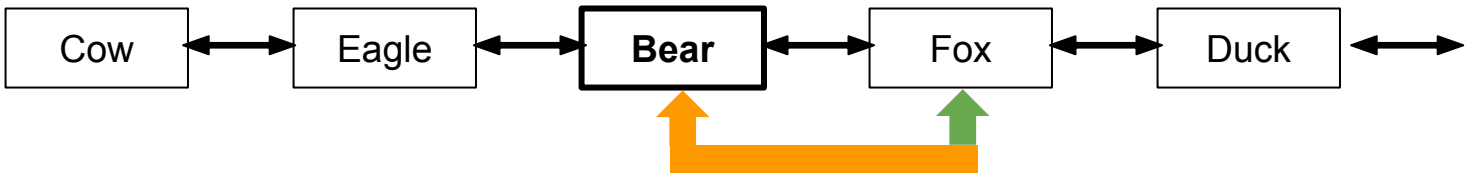

1 ⊕ (0 ⊕ 1) = 0

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **value** | Bear | Cow | Duck | Eagle | Fox | | | |
| **next⊕prev** | **4 ⊕ 3** | 3 ⊕ 2 | 1 ⊕ 4 | 0 ⊕ 1 | 2 ⊕ 0 | | | |
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |



# XOR linked list - traverse backward

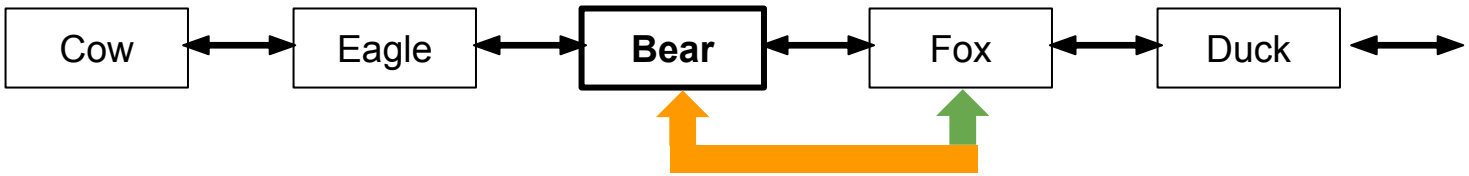
| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| value | **Bear** | Cow | Duck | Eagle | Fox | | | |
| next⊕prev | **4 ⊕ 3** | 3 ⊕ 2 | 1 ⊕ 4 | 0 ⊕ 1 | 2 ⊕ 0 | | | |
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |




4 ⊕ (4 ⊕ 3) = 3

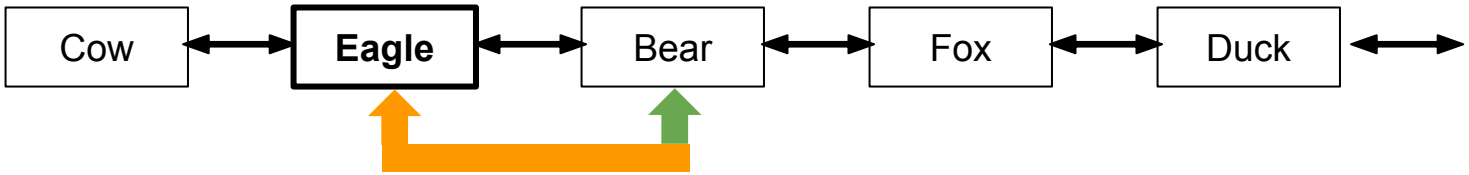
# XOR linked list - traverse backward

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| value | **Bear** | Cow | Duck | Eagle | Fox | | | |
| next⊕prev | **4 ⊕ 3** | 3 ⊕ 2 | 1 ⊕ 4 | 0 ⊕ 1 | 2 ⊕ 0 | | | |
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |



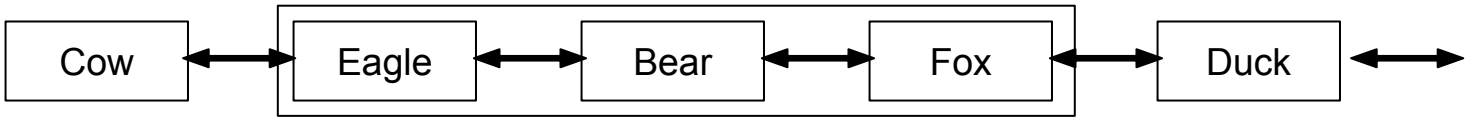

4 ⊕ (4 ⊕ 3) = 3

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| value | Bear | Cow | Duck | **Eagle** | Fox | | | |
| next⊕prev | 4 ⊕ 3 | 3 ⊕ 2 | 1 ⊕ 4 | **0 ⊕ 1** | 2 ⊕ 0 | | | |
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |



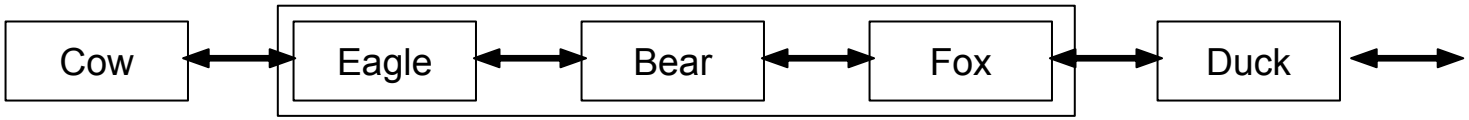
# XOR linked list - rotate segment

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **value** | Bear | Cow | Duck | Eagle | Fox | | | |
| **next⊕prev** | 4 ⊕ 3 | 3 ⊕ 2 | 1 ⊕ 4 | 0 ⊕ 1 | 2 ⊕ 0 | | | |
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |



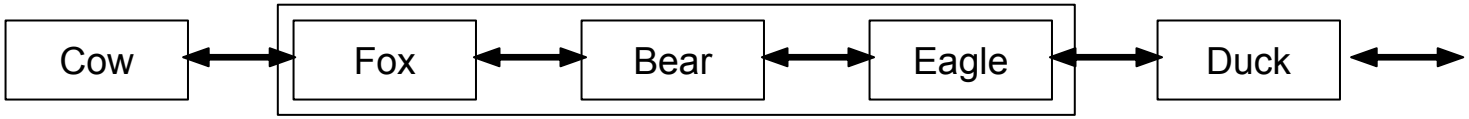
# XOR linked list - rotate segment

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **value** | Bear | Cow | Duck | Eagle | Fox | | | |
| **next⊕prev** | 4 ⊕ 3 | 3 ⊕ 2 | 1 ⊕ 4 | 0 ⊕ 1 | 2 ⊕ 0 | | | |
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |




| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **value** | Bear | Cow | Duck | Eagle | Fox | | | |
| **next⊕prev** | 3 ⊕ 4<br>=4⊕3 | 3 ⊕ 2 | 1 ⊕ 4 | **2 ⊕ 0** | **0 ⊕ 1** | | | |
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

Only the two end points were changed!



# XOR linked list

| | Singly linked list | Doubly linked list | XOR linked list |
|---|---|---|---|
| Space complexity | O(2N) = O(N) | O(3N) = O(N) | O(2N) = O(N) |
| Backward traversal | No | Yes | Yes |
| Rotate segment | O(N) | O(N) | O(1) |

| Time complexities | Unrolled linked list | Splay tree | XOR linked list |
|---|---|---|---|
| Locate segment | O(N) | O(log(N)) | O(N) |
| Rotate segment | O(sqrt(N)) | O(log(N)) | O(1) |
| Diffculty | medium-hard | hard | easy |

# Summary

Abstract Data Types - implemented by data structures

- Stack - LIFO
- Queue - FIFO

Data structures

- Array
- Linked List
  - Singly linked list
  - Doubly linked list
  - Circular linked list
  - XOR linked list

# More practices

- HKOJ P004 Broken Keyboard
- HKOJ P006 Largest Rectangle in Histogram
- HKOJ M1803 I love you I love you
- HKOJ M1313 Bookstack