Stacks and Queue

MCA291:Data Structure with Python

Dr Tumpa Banerjee, Assistant Professor, Department of MCA

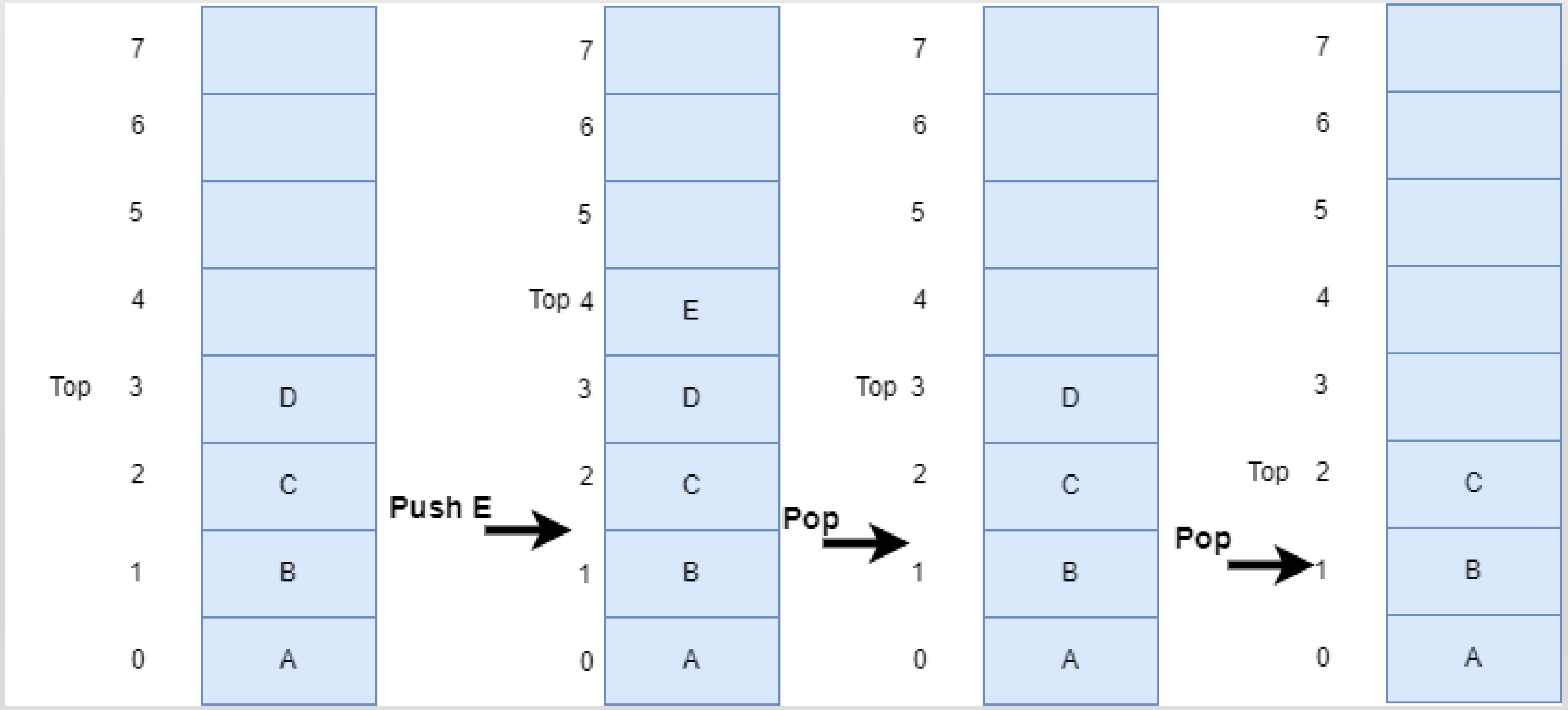
Table of Content

Abstract Data Type: Stack

- A stack is an ordered list in which insertion and deletion are done at one end, called top.
- The last element inserted is the first one to be deleted. Hence, it is called the Last in First out (LIFO) or First in Last out (FILO) list.
- When an element is inserted in a stack, the concept is called push, and when an element is removed from the stack, the concept is called pop.
- Trying to pop out an empty stack is called underflow and trying to push an element in a full stack is called overflow.

Abstract Data Type: Stack

Top: is the position of the last inserted item



Abstract Data Type: Stack

The following operations make a stack an ADT. For simplicity, assume the data is an integer type.

Main stack operations

- Push (int data): Inserts data onto stack.
- int Pop(): Removes and returns the last inserted element from the stack

Auxiliary stack operations

- int Top(): Returns the last inserted element without removing it.
- int Size(): Returns the number of elements stored in the stack.
- int IsEmptyStack(): Indicates whether any elements are stored in the stack or not.

Application of Stack

- Balancing of symbols
- Infix-to-postfix conversion
- Evaluation of postfix expression
- Implementing function calls (including recursion)

Implementation of Stack

PUSH(data, top, item, size)

This algorithm will insert the $item$ on the top of the stack $data$

```
1. If $top = size - 1$ then
2. Print “$stack\ overflow$” and $return$
3. Set $data[top] := item$
4. Set $top = top + 1$
5. end
```

Implementation of Stack

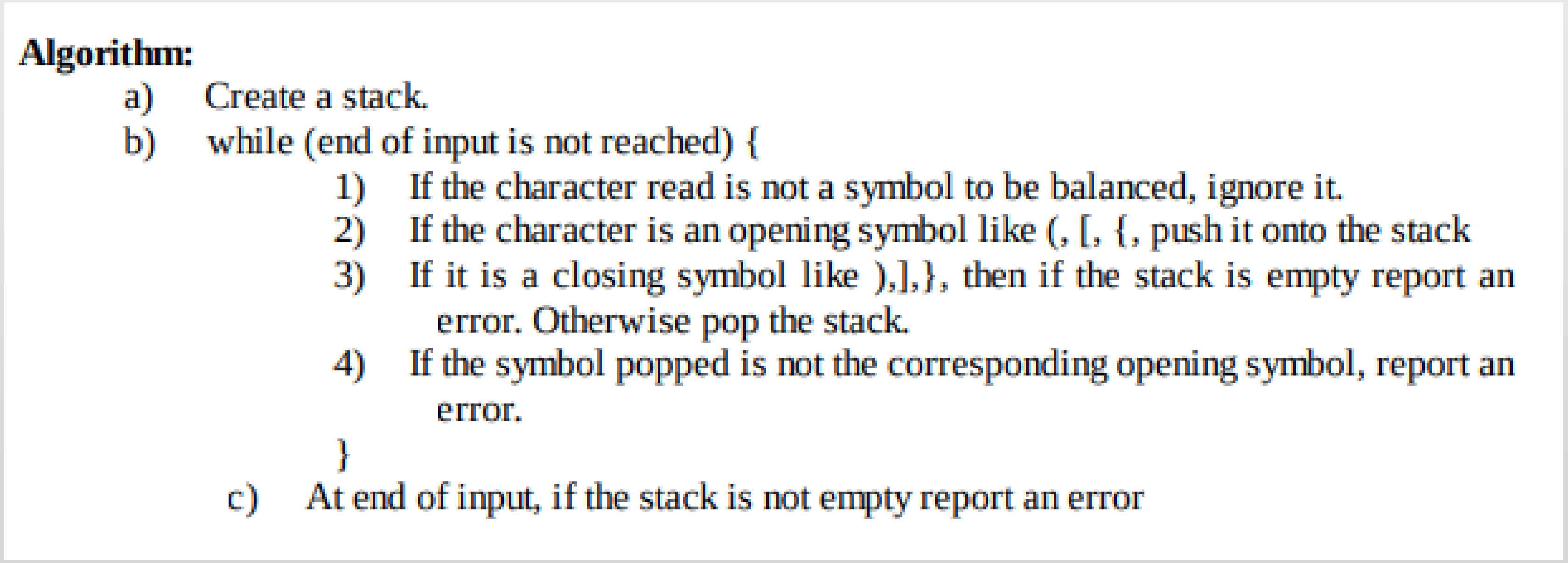
POP(data, top, item, size)

This algorithm will return the $item$ from the stack $data$

```
1. If $top = -1$ then
2. Print "$stack\ underflow$" and $return$
3. Set item:= $data[top]$
4. Set $top := top - 1$
5. Return $item$
```

Stacks: Problems & Solutions

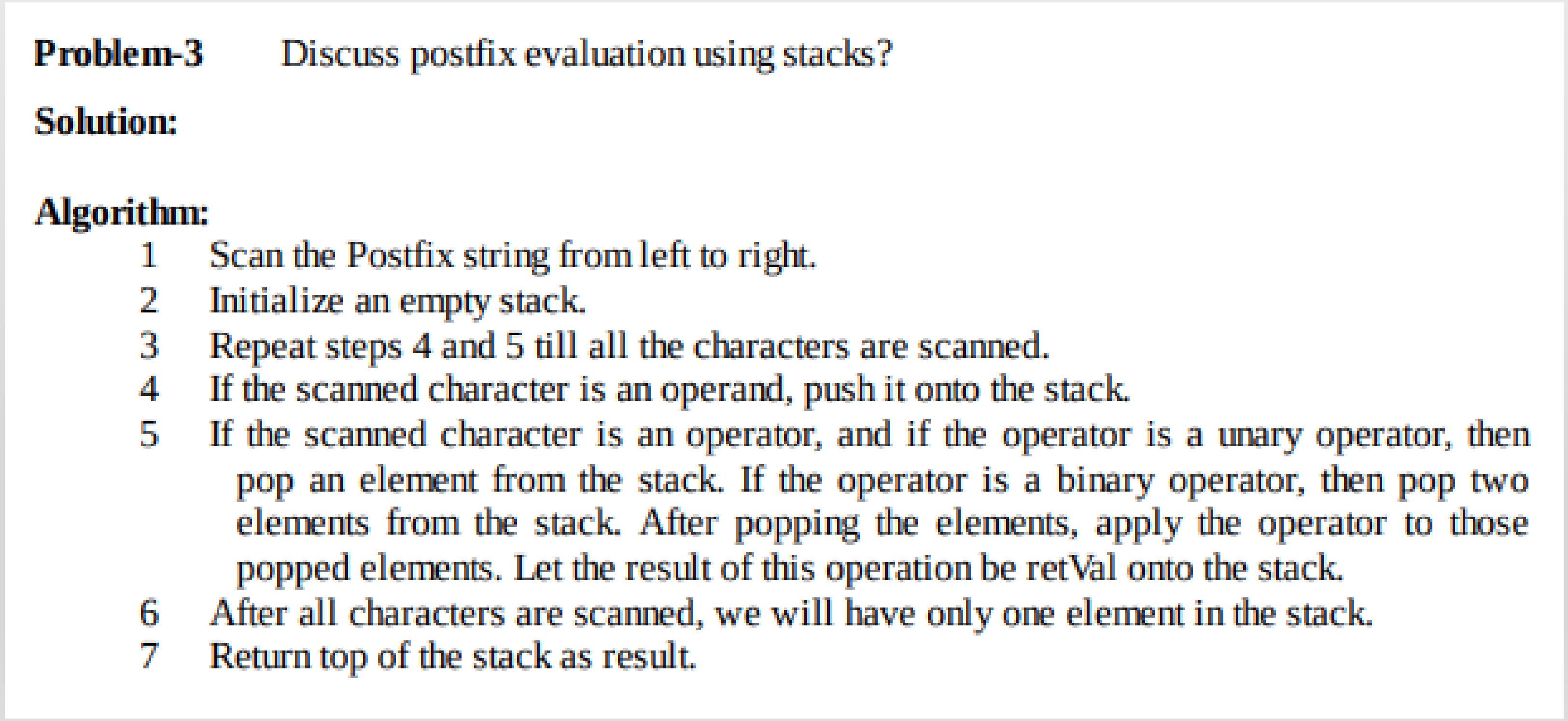
- Problem-1 How stacks can be used for checking balancing of symbols.



Algorithm:

```
a) Create a stack.
b) while (end of input is not reached) {
        1) If the character read is not a symbol to be balanced, ignore it.
        2) If the character is an opening symbol like (, [, {, push it onto the stack
        3) If it is a closing symbol like ),],}, then if the stack is empty report an
           error. Otherwise pop the stack.
        4) If the symbol popped is not the corresponding opening symbol, report an
           error.
        }
   c) At end of input, if the stack is not empty report an error
```

- Problem-2 Discuss postfix evaluation using stacks?



Problem-3 Discuss postfix evaluation using stacks?

Solution:

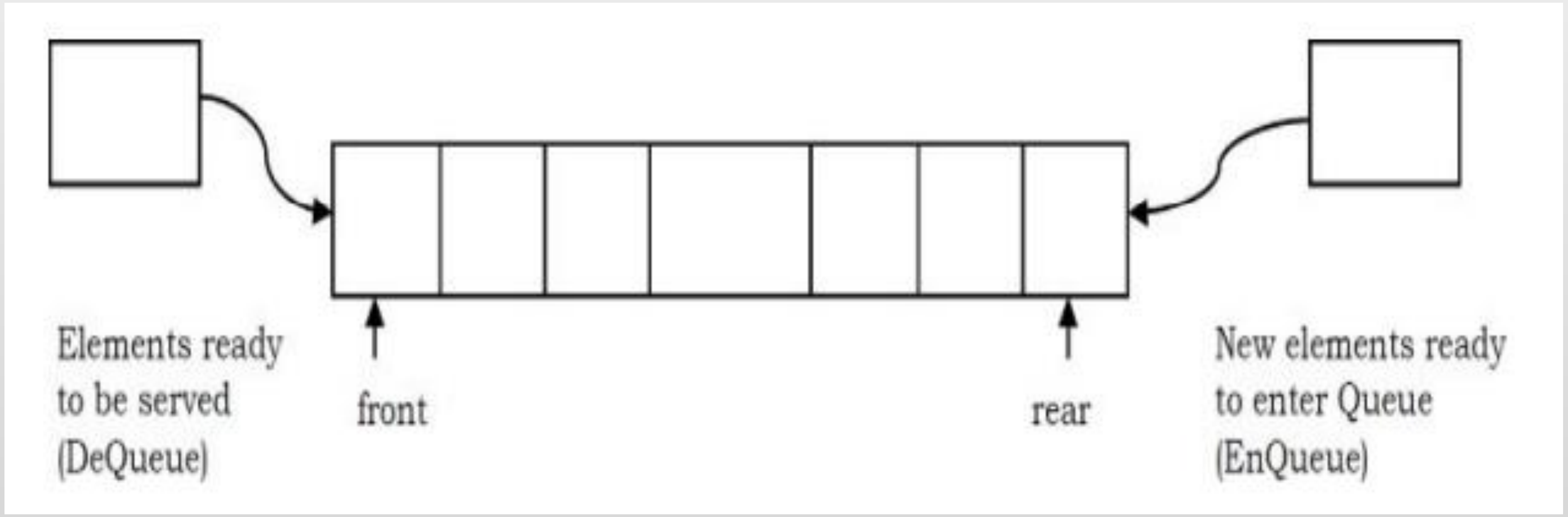
Algorithm:

1. Scan the Postfix string from left to right.
2. Initialize an empty stack.
3. Repeat steps 4 and 5 till all the characters are scanned.
4. If the scanned character is an operand, push it onto the stack.
5. If the scanned character is an operator, and if the operator is a unary operator, then pop an element from the stack. If the operator is a binary operator, then pop two elements from the stack. After popping the elements, apply the operator to those popped elements. Let the result of this operation be retVal onto the stack.
6. After all characters are scanned, we will have only one element in the stack.
7. Return top of the stack as result.

ADT: Queue

- A queue is an ordered list in which insertions are done at one end (rear) and deletions are done at other end (front).
- The first element to be inserted is the first one to be deleted. Hence, it is called First in First out (FIFO) or Last in Last out (LILO) list.
- In queue, the order in which data arrives is important.
- When an element is inserted in a queue, the concept is called EnQueue, and when an element is removed from the queue, the concept is called DeQueue.

ADT: Queue


Elements ready to be served (DeQueue)
front
rear
New elements ready to enter Queue (EnQueue)

ADT: Queue

- Main Queue Operations
  - ➢ EnQueue(int data): Inserts an element at the end of the queue
  - ➢ int DeQueue(): Removes and returns the element at the front of the queue
- Auxiliary Queue Operations
  - ➢ int Front(): Returns the element at the front without removing it
  - ➢ int QueueSize(): Returns the number of elements stored in the queue
  - ➢ int IsEmptyQueueQ: Indicates whether no elements are stored in the queue or not

Applications

- Operating systems schedule jobs (with equal priority) in the order of arrival (e.g., a print queue).
- Simulation of real-world queues such as lines at a ticket counter or any other firstcome first-served scenario requires a queue.
- Multiprogramming.
- Asynchronous data transfer (file IO, pipes, sockets).
- Waiting times of customers at call center.
- Determining number of cashiers to have at a supermarket.

Implementation

$insertion(queue, front, rear, item, size)$

This algorithm will insert $item$ to the next $rear$ position of the $queue$

```
1.  if $rear = size - 1$
2.  then $print$ "queue overflow" and $return$
3.  set $rear = rear + 1$
4.  set $queue[rear] = item$
5.  if $fron = -1$
6.  then set $front = 0$
7.  End
```

$Deletion(queue, front, rear, size)$

This algorithm will delete an item from front and return it

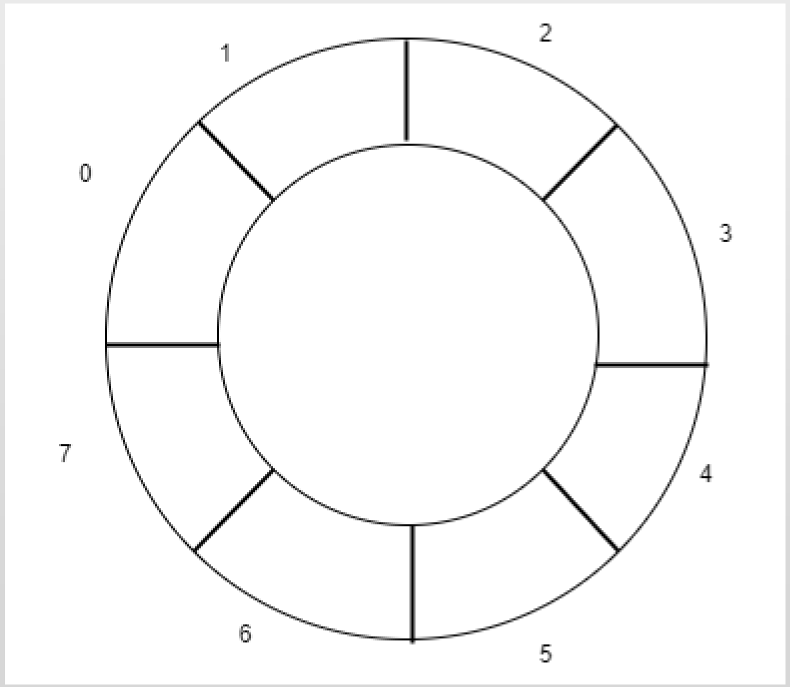
```
1. if $front = rear = -1$
2. then print "queue underflow" and $return$
3. set $item = queue[front]$
4. if $front = size - 1$
5. then set $front = rear = -1$
6. else set $front = front + 1$
7. return $item$
```

Circular Queue

- The array storing the queue elements may become full.
- An EnQueue operation will then throw a full queue exception.
- Similarly, if we try deleting an element from an empty queue it will throw empty queue exception.
- the initial slots of the array are getting wasted.
- simple array implementation for queue is not efficient. Circular queue is proposed to solve this problem

Circular Queue

- The structure of the Circular Queue is with the end connected back to the start.
- The circular queue connects the last element of the queue to the first element of the queue.
- The memory used is **Efficient** and utilizes all available space



Circular Queue Implementation

CQ$insertion(queue, front, rear, item, size)$

This algorithm will insert $item$ to the next $rear$ position of the $queue$

```
1.  if $front = (rear + 1)\%size$
2.  then $print$ "queue overflow" and $return$
3.  set $rear = (rear + 1)\%size$
4.  set $queue[rear] = item$
5.  if $fron = -1$
6.  then set $front = 0$
7.  End
```

Circular Queue Implementation

$CQDeletion(queue, front, rear, size)$

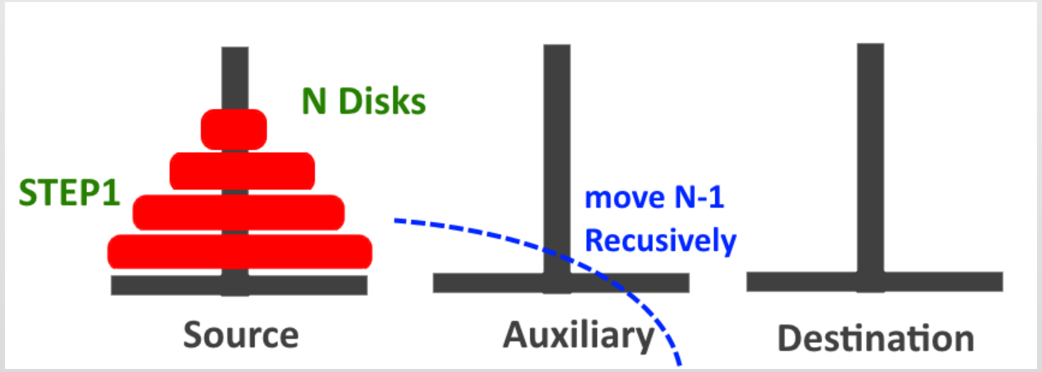
This algorithm will delete an $item$ from $front$ of the queue and return it

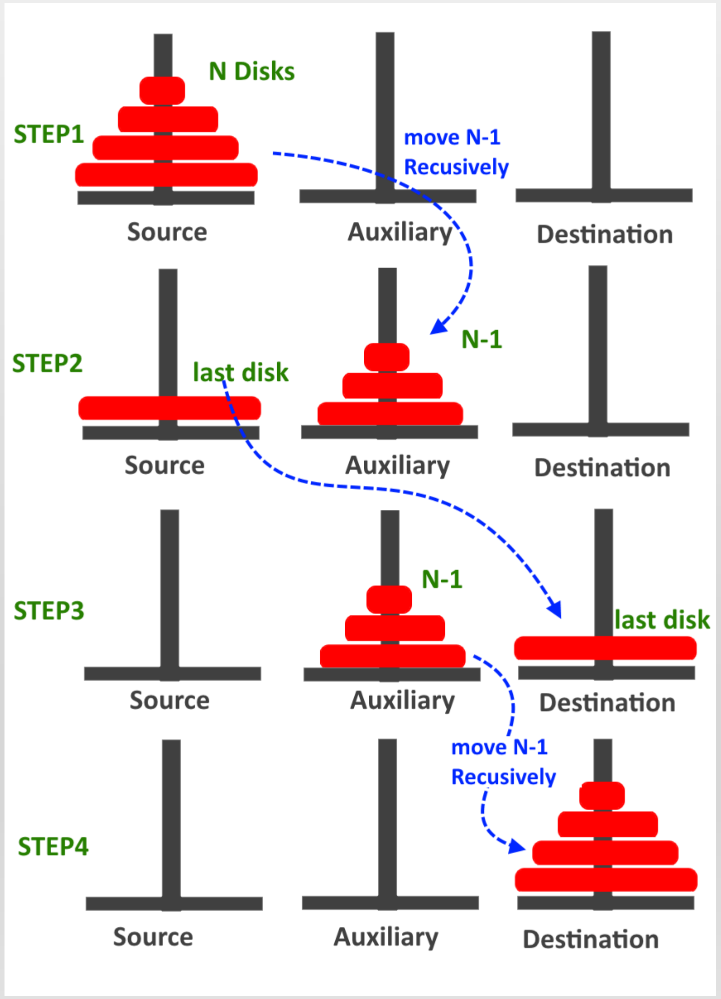
```
1. if $front = rear = -1$
2. then print "queue underflow" and $return$
3. set $item = queue[front]$
4. if $front =$ rear
5. then set $front = rear = -1$
6. else set $front = (front + 1)\%size$
7. return $item$
```

Tower of Hanoi Problem

- Tower of Hanoi is a mathematical puzzle where we have three rods (B,A,E) and N disks. Initially, all the disks are stacked in decreasing value of diameter i.e., the smallest disk is placed on the top and they are on rod B. The objective of the puzzle is to move the entire stack to another rod (here considered E), obeying the following simple rules:
- Only one disk can be moved at a time.
- Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.
- No disk may be placed on top of a smaller disk.

Tower of Hanoi Problem





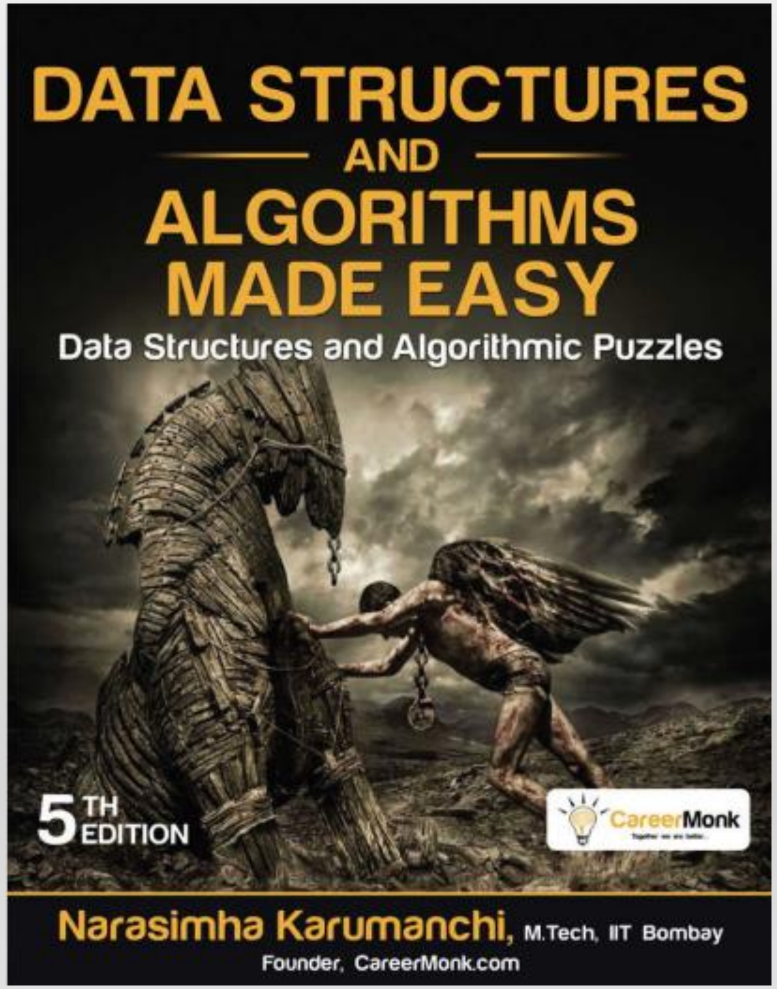
TOH Algorithm

```
TOH(N,SOURCE, AUXILIARY, DESTINATION)
1. If $N == 1$
2. then $Move\ source -> destination$
3. else
4.     $TOH(N-1, SOURCE, DESTINATION, AUXILIARY)$
5.     $Move\ source -> destination$
6.     $TOH(N-1, AUXILIARY, SOURCE, DESTINATION)$
7. End
```

Priority Queue

- A **priority queue** is a type of queue that arranges elements based on their priority values.
- Elements with higher priority values are typically retrieved or removed before elements with lower priority values.
- Each element has a priority value associated with it.
- When we add an item, it is inserted in a position based on its priority value.
- Binary heap is used to implement priority queue.

References


DATA STRUCTURES AND ALGORITHMS MADE EASY

Data Structures and Algorithmic Puzzles

5TH EDITION

CareerMonk

Narasimha Karumanchi, M.Tech, IIT Bombay

Founder, CareerMonk.com