Stacks and Queues

Unit 3

CC4 Data Structures and Algorithms

Lovey Jenn A. Reformado

Table of Contents

Introduction to Stacks and Queues

Importance of Stacks and Queues | Applying Stacks and Queues | Stacks v. Queues

Importance of Stacks and Queues

- Both stacks and queues are an example of data structures
- Most common way to arrange data in different algorithms of the same data type
    - Expressions (infix, prefix, postfix)
    - Binary search trees
    - AVL trees
    - Graphs
    - Searching algorithms
    - Sorting algorithms

Applying Stacks and Queues

- Usually implemented in a one-dimensional array
  - Can use other variations: list, linked list, etc.
  - Possible to use for any programming language
- Can also be used in multidimensional arrays
  - Not recommended
  - Increases the time complexity and the space required

Stacks and Queues

Stack

- Last-in first-out policy
- First element is at the bottom
- Last element is at the topmost part
- Examples: pancakes, Pringles can, stack of books, etc.

Queue

- First-in first-out policy
- First element is at the beginning / front
- Last element is at the end of the queue
- Examples: lines at the supermarket or jeepney

Stacks

Introduction to Stacks | Inserting Elements in Stacks | Deleting Elements in Stacks | Example

Stacks

- Container based on the last-in-first-out (LIFO) policy
  - New data is inserted at the last index (push)
  - Data to be deleted starts off with the last index (pop)
- Uses only one (1) pointer
  - Starts off at array[-1]: empty
- Maximum number of elements is the limit of the array
- Practical examples:
  - Pringles can
  - Pancake stack
  - Stack of clothing

Inserting Elements in Stacks (Push)

- Create a one-dimensional array
  - Pointer is at -1
- Place value to be inserted in another variable
- Locate the pointer
- Iterate the value of the pointer
- Place value to the array where is the index is pointer (array[pointer])

Deleting Elements in Stacks (Pop)

- Locate the pointer
- Assign the value of the pointer to the index of the array
- Remove the value at array[pointer]
- Decrement the value of the pointer by 1

Stacks – Example

0	1	2	3	4	5	6	7

Values: 8, -1, 5, 7, 2, 6, 3, 4, 9, 10

Pointer:

Values – push:

Values – pop:

Queues

Introduction to Queues | Inserting Elements in Queues | Deleting Elements in Queues | Queues as a Circular Structure | Examples

Queues

- Container based on the first-in-first-out (FIFO) policy
  - New data is inserted at the last index
  - Data to be deleted starts off with the first index
- Uses two (2) pointers
  - One pointer is for the first element in the array
  - Another pointer is for the space after the last element in the array
- Maximum number of elements is array.length-1
  - Queue is considered empty if both pointers are on the same position
  - Arrays can be considered circular
- Practical examples:
  - People queuing on a line

Inserting Elements in Queues (Enqueue)

- Create a one-dimensional array
- Place the value to be inserted into a variable
- Locate the value of the second pointer (position of the index number after the last element that was placed)
- Place value to the array where is the index is the second pointer (array[pointer_two])
- Iterate the second pointer

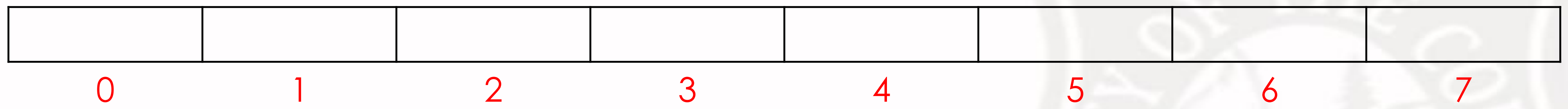
Deleting Elements in Queues (Dequeue)

- Locate where the first pointer is (pointer that shows the earliest element inserted)
- Remove the value at array[pointer_one]
- Iterate first pointer

Queues as a Circular Structure

- In theory, the elements of the queue does not change positions
  - Index numbers don't change if the first element is deleted
- Queue is a circular tape structure
  - Elements may be placed at earlier indices provided that they are empty, and the last index of the array was occupied
  - Done to maximize the array
  - Pointers would go back to [0] after array.length-1

Queues – Example



Values: 8, -1, 5, 7, 2, 6, 3, 4, 9, 10

Pointer1:

Pointer2:

Values – insert:

Values – delete: