Discussion 1: Basic Structures, Algorithm Analysis

CS101 Fall 2024

CS101 Course Team

Oct 2024

Contact Us

- 周守琛
  qq: 1354038619
  email: zhoushch@shanghaitech.edu.cn
- 卢鳌
  email: luao@shanghaitech.edu.cn

Array

- Representation of polynomial coefficients:
- Increase array capacity:

	Copies per Insertion	Unused Memory
Increase by 1	$n-1$	0
Increase by m	n/m	$m-1$
Increase by a factor of 2	1	n
Increase by a factor of $r>1$	$1/(r-1)$	$(r-1)n$

Linked List

- Given value v and header h, how to find a specific node with value v?
- only by traversing the linked list!
- Time complexity $O(n)$

	Front/1st node	kth node	Back/nth node
Find	$O(1)$	$O(n)$	$O(n)$
Insert After	$O(1)$	$O(1)$	$O(1)$
Replace	$O(1)$	$O(1)$	$O(1)$
Next	$O(1)$	$O(1)$	n/a
Previous	n/a	$O(n)$	$O(n)$

- How to implement a linked list? Array, Stack, Queue
- Doubly Linked List: $O(1)$ previous, $O(n)$ extra space.

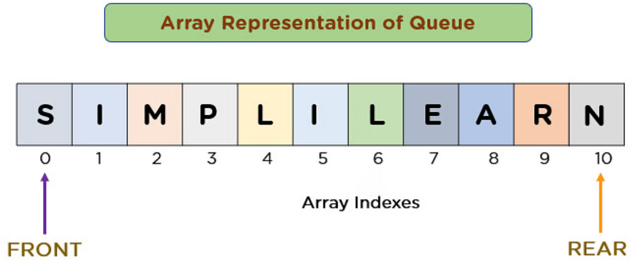
Stack

- LIFO(Last In First Out) Data Structure.
- Basic Operation:
  - **Push**: Adds an element to the top of the stack.
  - **Pop**: Removes the top element from the stack.
  - **Peek**: Returns the top element without removing it.
  - **IsEmpty**: Checks if the stack is empty.
  - **IsFull**: Checks if the stack is full (in case of fixed-size arrays).
- ALL implemented in $O(1)$
- use linked list to implement a stack.
- Check if the given push and pop sequence of the stack is valid or not:

$$1\ 3\ 4\ 7\ 5\ 2\ 6$$

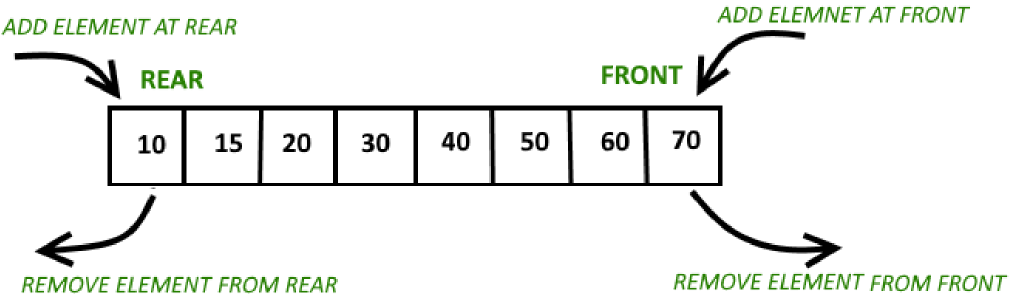
Queue

- FIFO(First In First Out) Data Structure.
- How to use an array to represent a queue?



- use linked list to implement a queue.
- Circular Array: Index using a modulo operation.

Deque


ADD ELEMENT AT REAR
ADD ELEMNET AT FRONT
REAR
FRONT
10
15
20
30
40
50
60
70
REMOVE ELEMENT FROM REAR
REMOVE ELEMENT FROM FRONT

Time Complexity

Here is the definition of Landau Symbols without using the limit:

$$f(n) = \Theta(g(n)) : \exists c_1, c_2 \in \mathbb{R}^+, \exists n_0, \forall n > n_0, 0 \leq c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n).$$
$$f(n) = O(g(n)) : \exists c \in \mathbb{R}^+, \exists n_0, \forall n > n_0, 0 \leq f(n) \leq c \cdot g(n).$$
$$f(n) = \Omega(g(n)) : \exists c \in \mathbb{R}^+, \exists n_0, \forall n > n_0, 0 \leq g(n) \leq c \cdot f(n).$$
$$f(n) = o(g(n)) : \forall c \in \mathbb{R}^+, \exists n_0, \forall n > n_0, 0 \leq f(n) < c \cdot g(n).$$
$$f(n) = \omega(g(n)) : \forall c \in \mathbb{R}^+, \exists n_0, \forall n > n_0, 0 \leq g(n) < c \cdot f(n).$$

Precise Form: $\Theta(), o(), \omega()$

Order:

$$1 < \log n < n < n \log n < n^2 < n^2 \log n < n^3 < 2^n < 3^n < n! < n^n$$

Worst-, Best-, Average-Case

- **Worst Case Analysis**: Upper bound of running time.
  e.g. The worst-case time complexity of the linear search would be $O(n)$.
- **Best Case Analysis**: Lower bound of running time.
  Very rarely used! for linear search, the lower bound should be $\Omega(1)$
- **Average Case Analysis**: take all possible inputs and calculate the computing time for all of the inputs.
  Must know (or predict) the distribution of cases! For linear search, all cases are uniformly distributed.