LL

Wednesday, April 2, 2025 12:27 PM

LL

CSC 212: Data Structures and Abstractions

11: Linked Lists

Prof. Marco Alvarez

Department of Computer Science and Statistics
University of Rhode Island

Spring 2025

THINK BIG WE DO™

Practice

- Assuming that a character array starts at address `0x0100`
  - label the memory addresses of all elements

'h'	'e'	'l'	'l'	'o'	0				

- Assuming that an integer array starts at address `0x0100`
  - label the memory addresses of all elements

3	-12	2	4	10	20	22			

Practice

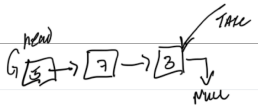
- Assume a dynamic array and efficient implementations
  - what is the cost of inserting 1 element at the end?
    O(1) - AMORTIZED
  - what is the cost of inserting 1 element at the front?
    O(N)
  - what is the cost of inserting 1 element at index `idx`?
    O(N)
  - what is the cost of performing deletions at those same locations?
    SAME THING

Linked lists

Linked list

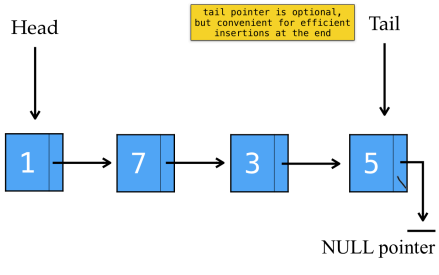
- Definition
  - a linked list is a **linear data structure** that consists of a sequence of elements stored at **non-contiguous** locations in memory
- Typical operations
  - **insert**: add a new node to the list (rear, front, at index, by value)
  - **delete**: remove a node from the list (rear, front, at index, by value)
  - **search**: find a node with a specific value
  - **get**: get a value at an specific index
  - **traverse**: "visit" each node in the list – PRINT etc.

Linked lists

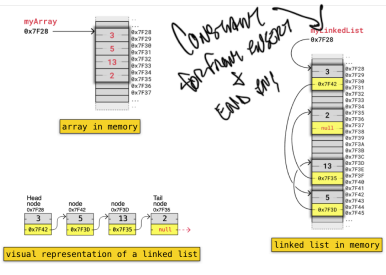


- Types of linked lists
  - **singly-linked list**: each node has a pointer to the next node
  - **doubly-linked list**: each node has a pointer to the next and previous nodes
  - **circular-linked list**: the last node has a reference to the first node
- Singly-linked list
  - each element is a **node** that contains a value and a pointer to a next node
  - the last node has a reference to **null**
  - the first node is called the **head**
  - the last node is called the **tail**
  - the length of the linked list is the number of nodes

Singly-linked list



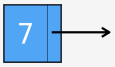
Singly-linked list and memory



https://www.w3schools.com/dsa/dsa_theory_linkedlists_memory.php

Implementing a linked list

Representing a node




```
struct Node {
    T data;
    Node *next;
    Node(const T& value) {
        data = value; next = nullptr;
    }
};
```

`struct` representing a node in a linked list using templates. It contains a value of type T, a pointer to the next node, and a constructor that initializes the value and sets the next pointer to `nullptr`

10

Representing a singly-linked list

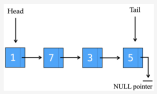
```
template <typename T>
class SLList {

    private:
        struct Node {
            T data;
            Node *next;
            Node(const T& value) { data = value; next = nullptr; }
        };

        Node *head;
        Node *tail;
        size_t size;

    public:
        SLList() { head = tail = nullptr; size = 0; }
        ~SLList() { clear(); }

        size_t get_size() { return size; }
        bool empty() { return size == 0; }
        void clear();
        T& front();
        T& back();
        void push_front(const T& value);
        void pop_front();
        void push_back(const T& value);
        void pop_back();
        void print();
};
```



11

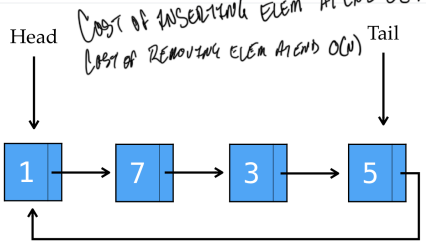
Methods

- Constructor
- Destructor
- clear

12

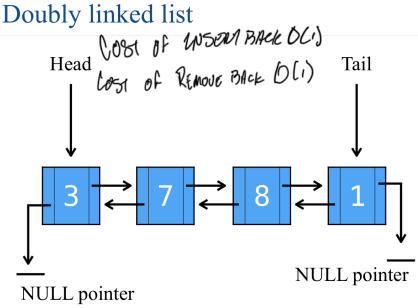
Other types of linked lists

Circular singly linked list



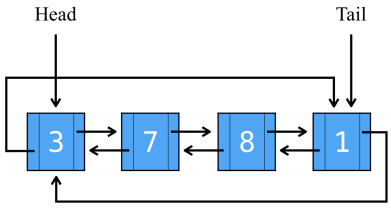
18

Doubly linked list



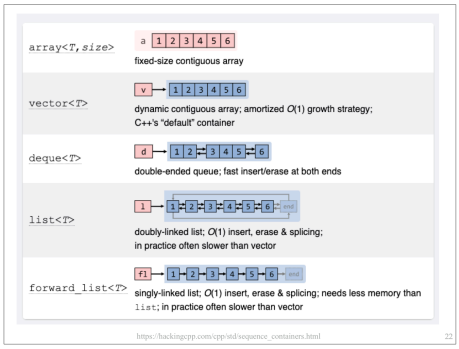
19

Circular doubly linked list



20

Linked lists in the STL



Others

Linked lists and other data structures

- Stacks
  - insert and remove from the same end
  - constant time complexity for both operations
- Queues
  - insert at one end and remove from the other end
  - constant time complexity for both operations
- Deques
  - insert and remove from both ends
  - constant time complexity for all insert/remove operations

24