Data Structures and Algorithms

Mohammad GANJTABESH

Department of Computer Science,
School of Mathematics, Statistics and Computer Science,
University of Tehran,
Tehran, Iran.

mgtabesh@ut.ac.ir

Dept. of Computer Science
University of Tehran

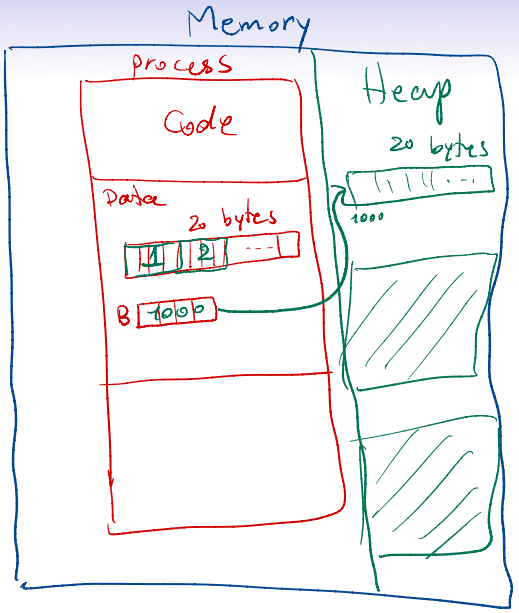
Linked List

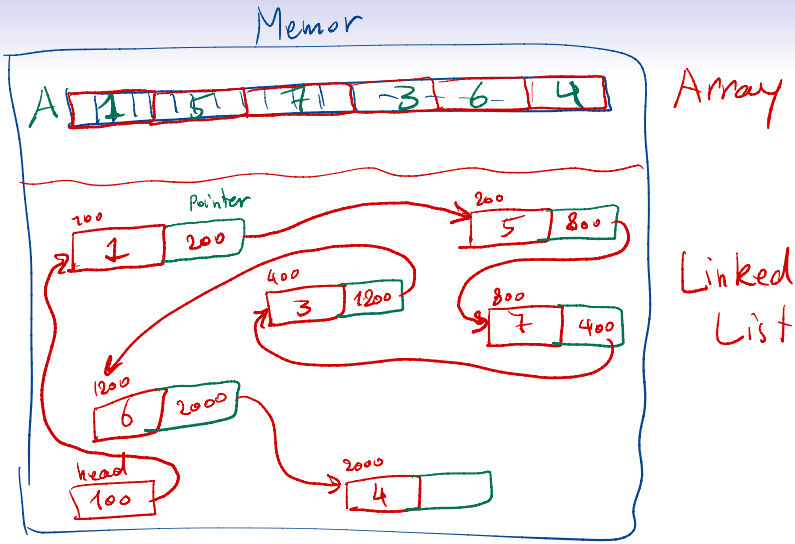
Data Structures and Algorithms
Undergraduate course

Mohammad GANJTABESH
mgtabesh@ut.ac.ir

Dept. of Computer Science
University of Tehran

```
int A[5];

A[0] = 1;
A[1] = 2;

int *B;
B = (int *) malloc (5 * sizeof(int));
B[0] = 1;  ≡  *B = 1;
B[1] = 2;  ≡  *(B+1) = 2;
```




Memor
A
1
5
7
-3
-6
4
Array
Pointer
100
1
200
200
5
800
400
3
1200
800
7
400
1200
6
2000
head
100
2000
4
Linked
List

Boundary Conditions

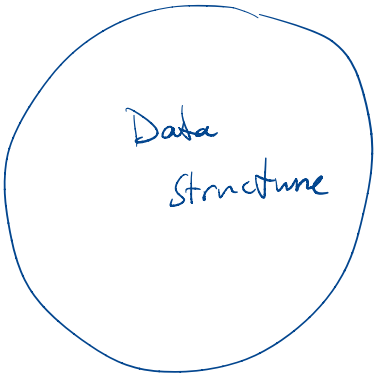
1- کار با داده ساختار خالی

2- کار با داده ساختار تک عنصری

3- اضافه کردن/حذف در ابتدای داده ساختار

4- اضافه کردن/حذف از انتهای داده ساختار

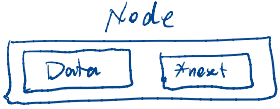
5- کار کردن در میانه داده‌ها.



Basics

لیست پیوندی یک دنباله‌ای از داده‌ها است که هر داده به کمک اشاره‌گر به داده بعدی پیوند خورده است.


```
struct Node{
  int Data;
  Node *next;
}
```

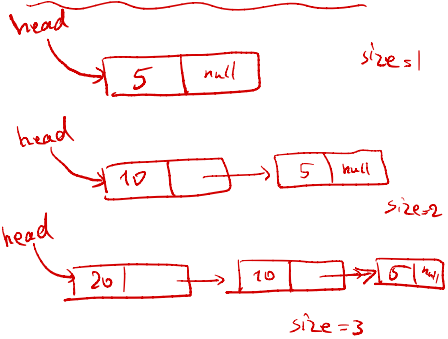


Building Block for Linked List.

Linked List:

```
class LinkedList {
    Node *head;
    int size;

    LinkedList(){
        head = null;
        size = 0
    }

    // other methods
}
```

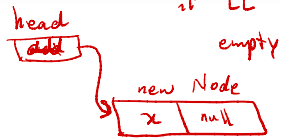
```
LinkedList L;
L.AddFirst(5);
L.AddFirst(10);
L.AddFirst(20);
```

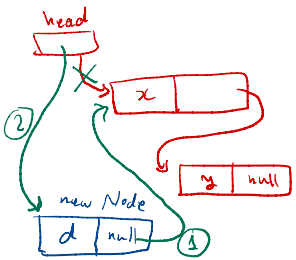


Linked List: Add an element at the beginning

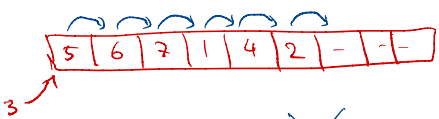
```
void AddFirst (int x) {
    Node node = new Node(x);
    node.next = head;
    head = &node;
    size ++;
}
```

O(1)





Array:

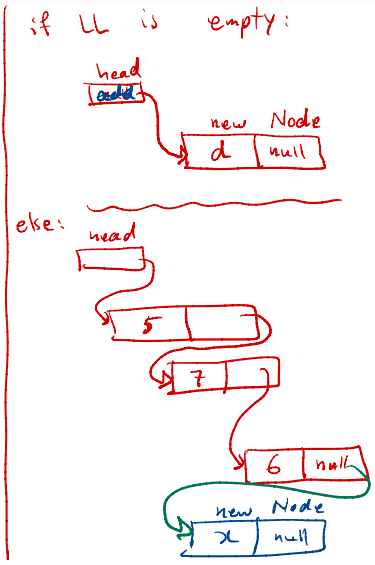


O(n) X

Linked List: Add an element at the end

```
void AddLast (x){
    Node node = new Node(x);
    if (head == null){ // LL is empty.
        head = &node;
        size ++;
        return;
    }
    Node temp = head;
    while (temp.next != null)
        temp = temp.next;
    temp.next = &node;
    size ++
}
```

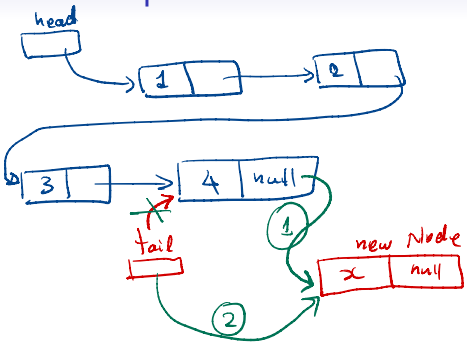
O(n)



Linked List: Efficient AddLast operation

```
class LinkedList {
    Node *head;
    Node *tail;
    int  size;
    ⋮
}

void AddLast(int x){
    Node node = new Node(x);
    if (head == null) {
        head = tail = &node;
        size ++;
        return;
    }
    tail.next = &node;
    tail = &node;
    size ++;
}
```

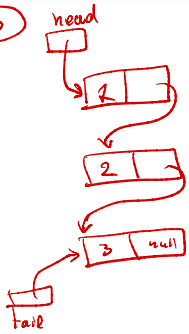
O(1)



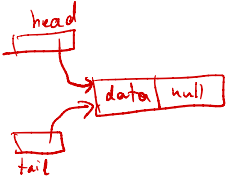
Linked List: Remove an element from the beginning

Boundary conditions:

1) Empty
2) single element
3) working at begin
4) " " end
5) " in the middle



if LL is empty




```
int RemoveFirst () {
    if (head == null) return (null);
    int data = head.Data;
    if (head == tail)
        head = tail = null;
    else
        head = head.next;

    size --;
    return (data);

}
```

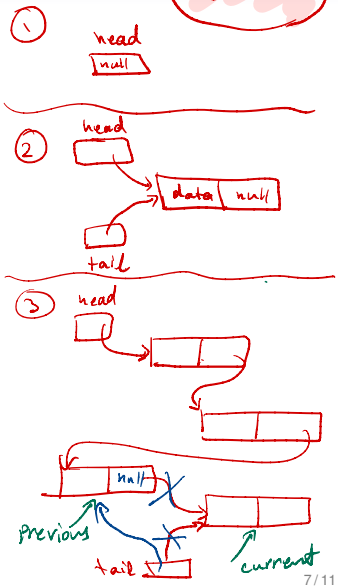
Linked List: Remove an element from the end

Boundary conditions.

```
int RemoveLast () {
    if (head == null)  return (null);
    int data = tail.Data;
    if ( head == tail ){
        head = tail = null ; size-- ;
        return (data);
    }
    else {
        Node *current = head;
        Node *previous = null ;
        while( current != tail){
            previous = current;
            current = current.next;
        }
        previous.next = null
        tail = previous ;
        size--;   return (data);
    }
}
```

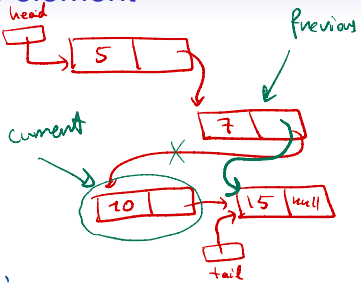
O(1)

O(n)



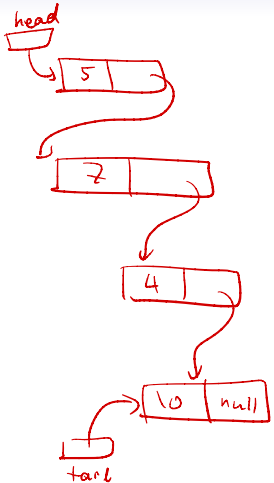
Linked List: Remove a specific element

```
int Remove (int x) {
    Node *current = head;
    Node *previous = null;
    while (current != null) {
        if (current.Data == x) {
            if (current == head)
                return (RemoveFirst());
            if (current == tail)
                return (RemoveLast());

            size--;
            previous.next = current.next;
            return (current.Data);
        }
        previous = current;
        current = current.next;
    }
    return (null);
}
```



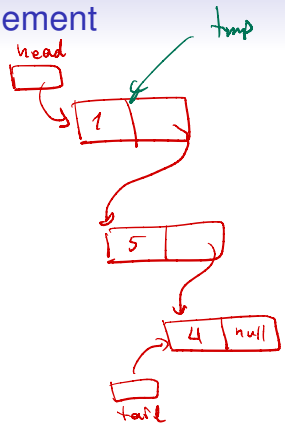
Linked List: Access to the first/last element

```
int PeekFirst(){
    if(head == null) return(null);
    return(head.Data);
}

int PeekLast(){
    if(tail == null) return(null);
    return(tail.Data);
}
```



Linked List: Find a specific element

```
bool Contains(int x){
    Node *tmp = head;
    while (tmp != null){
        if(tmp.Data == x)
            return (true);
        tmp = tmp.next;
    }
    return (false);
}
```



Linked List: Other operations

isEmpty ()

AddBefore (int x, int key)

AddAfter (int x, int key)

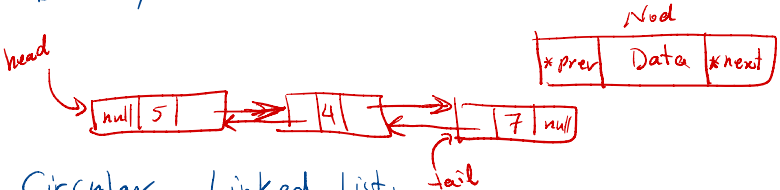
Reverse ()

LinkedList Range (int min, int max)

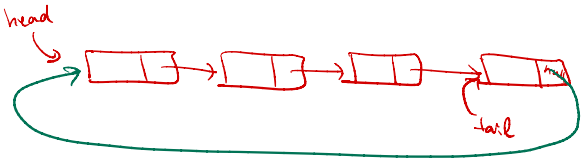
Type of LinkedList :

1) Single Linked List. ⟶ discussed.

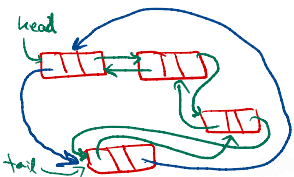
2) Doubly Linked List



3) Circular Linked List;



4) Circular Doubly Linked List




Linked List
vs.
Array
Random Access
Cache Friendly
Sequential Access
Ease of use
Fit the required size
memory usage
Dynamicity