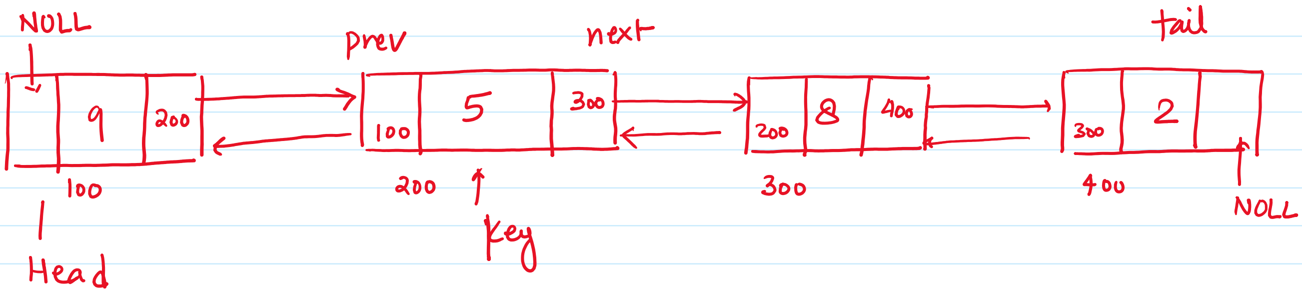
5) Lecture 7 Linked List

11 August 2024 11:34

Doubly linked list

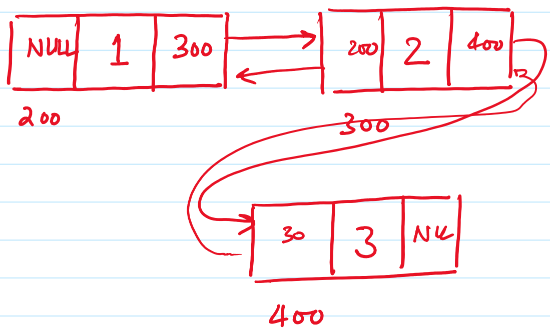



```
struct node
 {
    int key;
    struct node * prev;
    struct node * next;
 }
```

one ← 200
two ← 300
three ← 400
head ← 200

/* Initialize nodes */

struct node *head;
struct node *one = NULL;
struct node *two = NULL;
struct node *three = NULL;



/* Allocate memory */

one = malloc(sizeof(struct node));
two = malloc(sizeof(struct node));
three = malloc(sizeof(struct node));

/* Assign data values */
one->data = 1;
two->data = 2;
three->data = 3;

/* Connect nodes */
one->next = two;
one->prev = NULL;

```
two->next = three;
two->prev = one;

three->next = NULL;
three->prev = two;

/* Save address of first node in head */
head = one;
```

Searching in linked list : Checking if item K is present in list?

List_search (head, K)

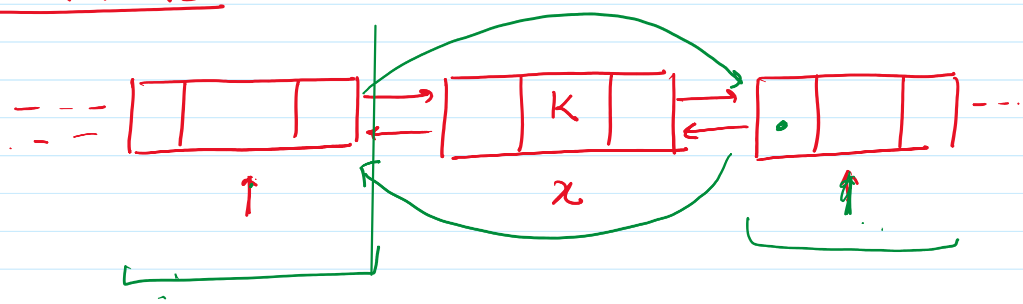
x = head

while (x ≠ NULL & x → key ≠ K)

[x = x → next

return x

Deleting an element from linked list



Delete(head, K)

{

x = list_search(head, K) .

/* update previous pointer

if (x → prev ≠ NULL)

{ x → prev → next = x → next

}

else

{ head = x → next

}

/* update next pointer

Time complexity

Search Θ(n)

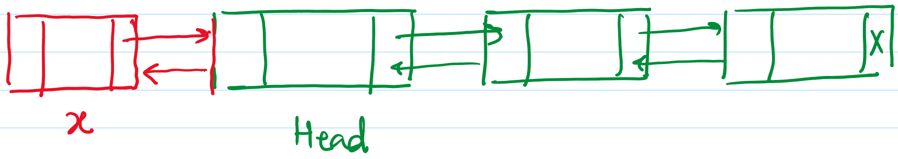
deletion Θ(1)

Θ(n)

if ($x \rightarrow next \neq NULL$)
{ $x \rightarrow next \rightarrow prev = x \rightarrow prev$
}

else
{ $x \rightarrow prev \rightarrow next = NULL$
}

Inserting on top of linked list



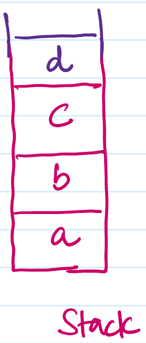
List-Insert-begining(head, x)
{
$x \rightarrow next = head$

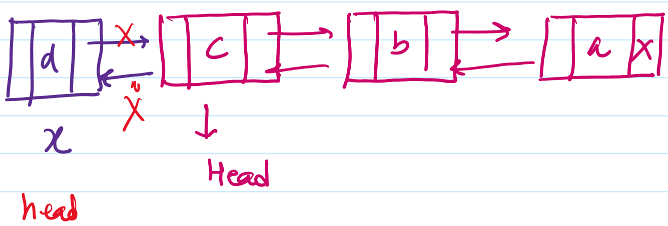
$head \rightarrow prev = x$

$head \leftarrow x$
}

x : add. of the node that need to be inserted

Implementing stack using linked list





Push(head, x)
{
List-Insert-begining(head, x)

return head

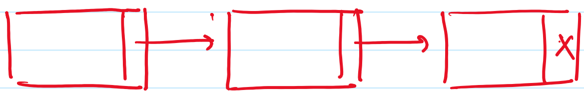
x : ptr to node that contain key as d
prev & next ptr are NULL

```
List-Insert-begining( head, x )
  return head
)

Pop ( head)
{  List- delete- begin (head)
   {
      x = head
      key = x→key
       x→ next→ prev = NULL
         x→ next = NULL
      head ← head → next
         delete (x)
   }
      return    key.
}
```

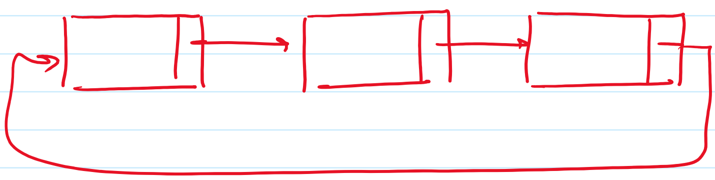
Singlely Linked

Head



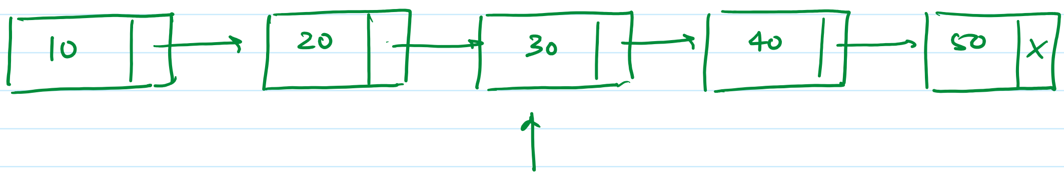
Circular Linked list

Head



Finding middle element of Linked list

Head

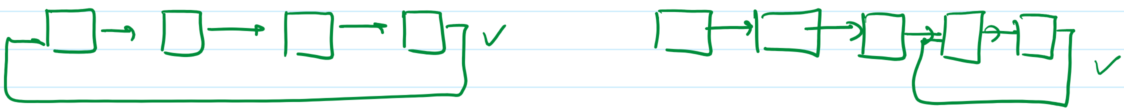


middle node

Use two pointer - initialize both ptr to head

- increment first ptr one step each time
- increment second ptr two step each time
- when 2^{nd} ptr reach to last node, return first ptr.

Checking if Linked list consist of a loop



Ptr to first node is given

Use two ptr : initialize both ptr to head

- increment first ptr one step each time
- " Second ptr two step " "
- if (first ptr == Second ptr)
  {
  return · 'list consist of loop'
  }
- if (Second ptr == NULL)
  { retur "list doesn't contain loop
  }

Complexity of Linked list operation

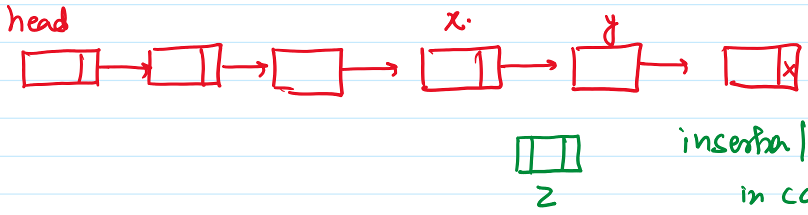
list consist of n nodes, Head ptr

① Search (Key) - checking if certain key is present in linked list - $\Theta(n)$

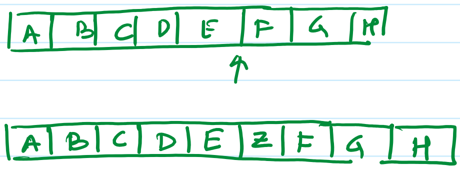
② Insertion / delete - $\Theta(n)$ - (if location of adjacent nodes are not give)
due to search

Comparision between Linked list & Array

(i) Ptr to x & y adjacent nodes are given



insertion / deletion can be done in const time $\Theta(1)$



Insert Z after E — $\Theta(n)$
Delete D — due to shift

(ii) Reading / Updation at specific index in array can be done in const time

Linked list requires $\Theta(n)$ time