Date: / /
Page:

Day - 119

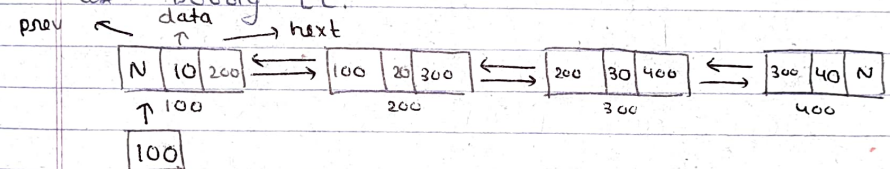
Linked List - 5

⇒ **Doubly Linked List:**

⇒ In singly Linked List, you can only go in forward direction.

⇒ But by using doubly LL, we can also go to the previous node.

⇒ If we want to move in both directions then we use Doubly LL.



⇒

```
class Node{
    public:
    int data;
    Node * next;
    Node * prev;

    Node( int value ){
        data = value;
        next = NULL;
        prev = NULL;
    }
};
```

Date: / /
Page:

* **Insertion:**

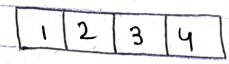
⇒ **At start:**

```
if (head == NULL) head = new Node(5); else{
Node *temp = new Node(5);
temp->next = head;
head->prev = temp;
head = temp;
}
```

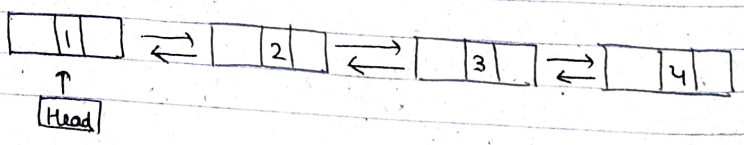
⇒ **At end:**

```
if (head){
Node *curr = head;
while( curr->next ){
        curr = curr->next;
}
Node *temp = new Node(5);
curr->next = temp;
temp->prev = curr;
}else{
    head = new Node(5);
}
```

⇒ Now, we given a array & we have to convert it into DLL.



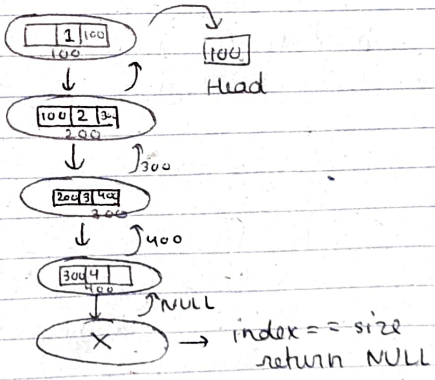
⇒



⇒
```
Node *head = NULL, *tail = NULL;
for(int i=0; i<5; i++){
    if(head == NULL){
        head = new Node(arr[i]);
        tail = head;
    }
    else{
        Node * temp = new Node(arr[i]);
        temp → prev = tail;
        tail → next = temp;
        tail = temp;
    }
}
```

⇒ Now, we have to create DLL by using recursion.

1	2	3	4




```
create DLL(int arr[], int index, int size, Node *back){
    if(index == size){
        return NULL;
    }
    Node *temp = new Node(arr[index]);
    temp → prev = back;
    temp → next = create DLL(arr, index+1,
                  size, temp);
}   return temp;
```

Date: / /
Page:

* Insert at a given position:

⇒
```
Node *curr = head;
while(--pos){
    curr = curr->next;
}
Node *temp = new Node(5);
temp->next = curr->next;
temp->prev = curr;
curr->next = temp;
temp->next->prev = temp;
```

⇒ There will two edge cases here when the value of pos == 0 & 1 or the value of pos is the last node.

* Deletion Operation:

⇒ First we will check if our LL is exist or not.

⇒ Delete at start:

```
if(head){
    Node *temp = head;
    head = head->next;
    delete temp;
  // head->prev = NULL;
  // Above line will give error, if there is only
  one node in our program.
    if(head)
        head->prev = NULL;
};
```

⇒ Delete at end:

```
if( head){
    Node * curr= head;
    while( curr→next){
        curr = curr→next;
    }
    curr→prev→next = NULL;
    delete curr;
}
if ( head→next == NULL){
    delete head;
    head = NULL;
}
```

⇒ Delete at any particular node:

int pos = 3

```
Node *curr= head;
while (--pos){
    curr = curr→next;
}
curr→prev→next = curr→next;
curr→next→prev = curr→prev;
delete curr;
```

⇒ There are many cases here.
⇒ when x=1 or when the value of x is the last node.