Data structures and Algorithms

By

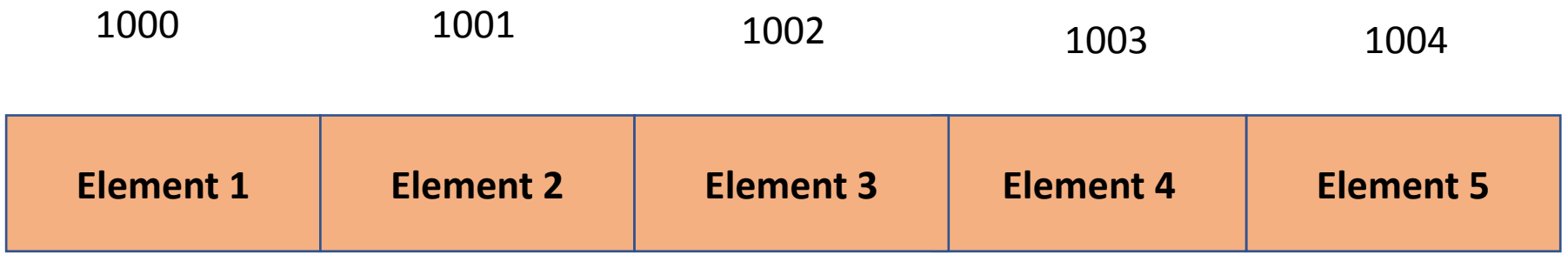
Engr Fatima Jaffar

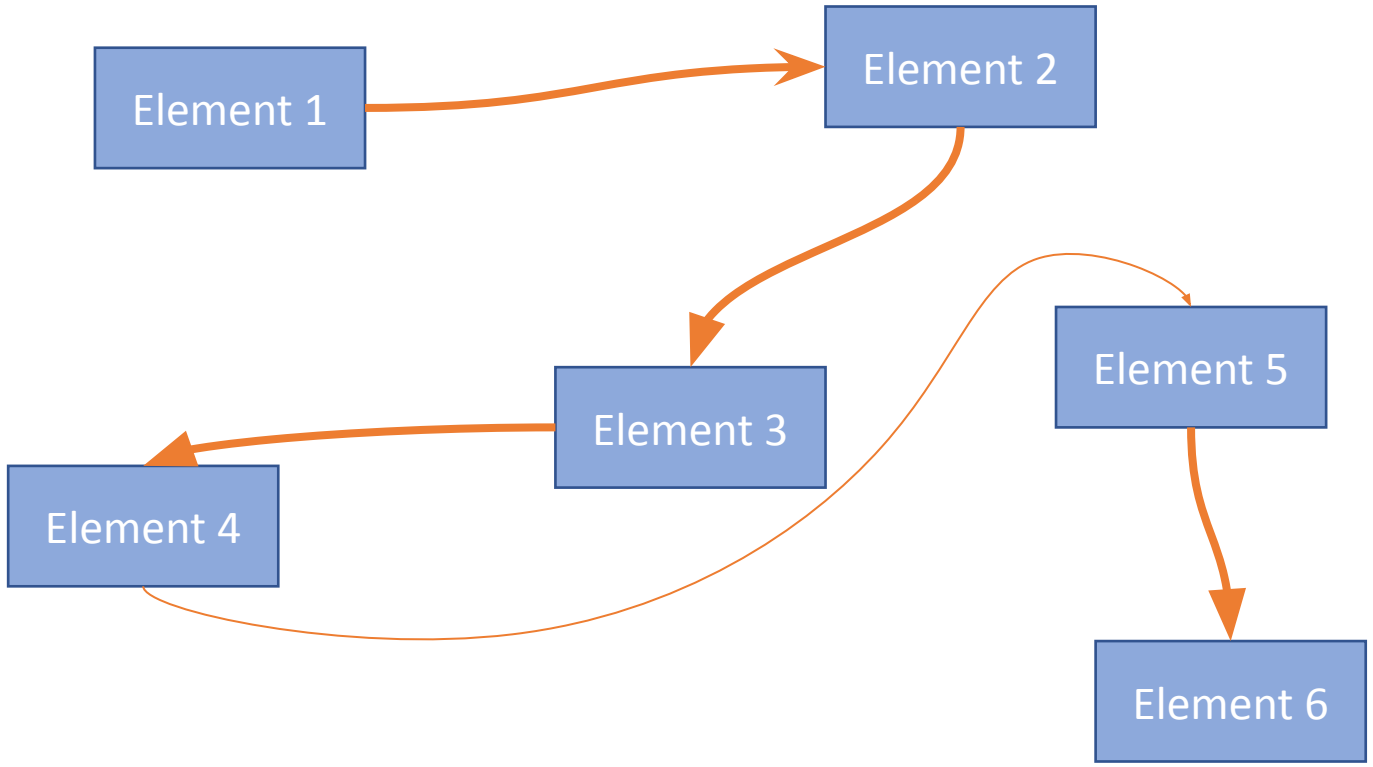
Limitations of Array

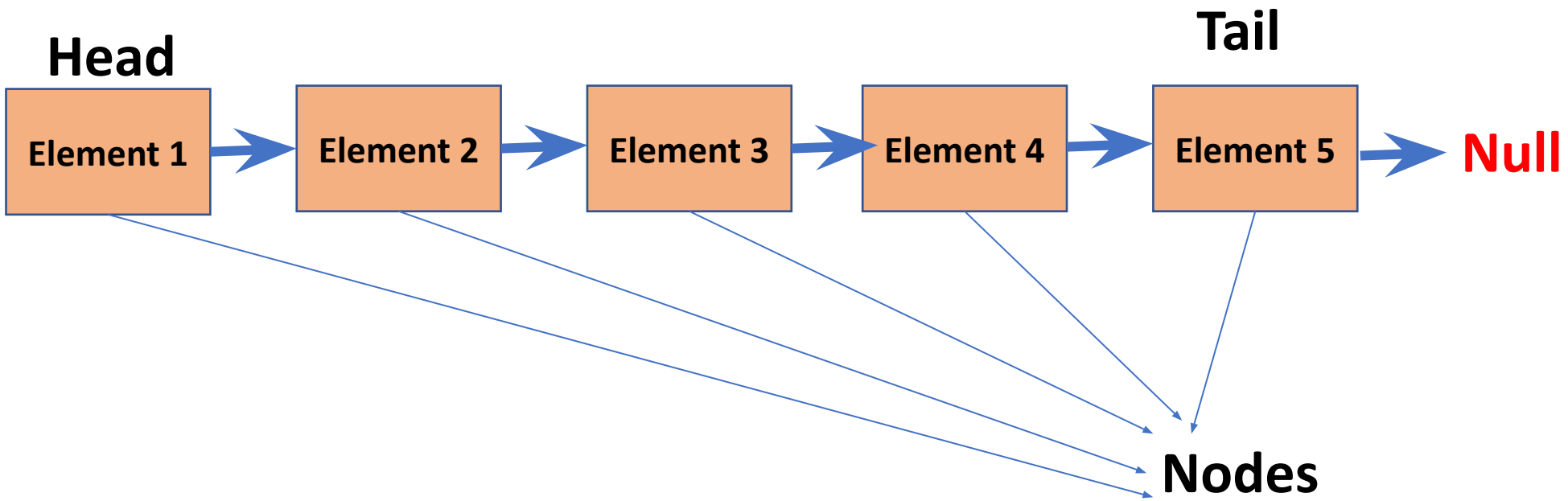
- Fixed size
- Continuous memory allocations

Linked List

- Not continuous memory allocations


1000
1001
1002
1003
1004
Element 1
Element 2
Element 3
Element 4
Element 5


Element 1
Element 2
Element 3
Element 4
Element 5
Element 6



Singly Linked List

Head and Tail

- First node of the list is head
- Last node of a linked list is tail

- We cannot access the elements through index
- Every node has just the information:

  - Next node
  - Its value

Types of Linked List

- There are three common types of Linked List.

1. Singly Linked List
2. Doubly Linked List
3. Circular Linked List

Singly Linked List

- It is the most common. Each node has data and a pointer to the next node.

Doubly linked list

- We add a pointer to the previous node in a doubly-linked list. Thus, we can go in either direction: forward or backward.

Circular Linked List

- A circular linked list is a variation of a linked list in which the last element is linked to the first element. This forms a circular loop.

Singly Linked List

- **Insertion** – Adds a new element in the list.
- **Deletion** – Deletes an element from the list.
- **Display** – Displays the complete list.
- **Search** – Searches an element using the given key.
- **Delete** – Deletes an element using the given key.

Traversal

```
public void display(){
    temp=head;
    while(temp!=null)
    {
        System.out.print(temp.value+"--->");
        temp=temp.next;
    }
    System.out.println("END");
}
```

Insertion

- At the beginning
- At the end
- At an specific position

Insertion at the begining

```
public void insertfirst(int val){
    Node node=new Node(val);
                node.next=head;
                head=node;
 if(tail==null){
          tail=head;
          tail.next=null;
             }

    size++;
```

Insertion at the last

```
public void insertlast(int value){
    if(tail==null){
        insertfirst(value);
        size++;
    }
    Node n=new Node(value);
    tail.next=n;
    tail=n;
    tail.next=null;
    size++;
```

Insert value on a specific position

```
public void insert(int position, int value){
      if(position==0){
            insertfirst(value);
      }
      else if(position==size){
            insertlast(value);
      }
      else{
            temp=head;
            for(int i=1;i<position-1;i++){
                  temp=temp.next;
            }
            Node n=new Node(value,temp.next);
            temp.next=n;

      }
      size++;
```

Deletion Operations

1. Delete first node
2. Delete last node
3. Delete on a specific position

Delete first node

```
public void delete_first(){
        head=head.next;
        size--;
}
```

Delete last node

```
public void delete_last(){
        temp=head;
        for(int  i=1;i<size-2;i++){
              temp=temp.next;
        }
    Node   node=  temp.next;
    node.next=null;
    tail=node;
    size--;

}
```

Delete on the position

```
public void delete(int position){
      temp=head;
      for(int i=1;i<position-1;i++){
            temp=temp.next;
      }
     int value=temp.next.value;
     temp.next=temp.next.next;
     size--;
}
```

Update node value

```
public void update(int val,int position){
        temp=head;
        for(int i=1;i<position;i++){
        temp=temp.next;
}
        temp.value=val;
}
```

Search operation

```
public void search(int value){
        temp=head;
        while(temp!=null){
                if(temp.value==value){
                        System.out.println(value+" is Present");
                }
                temp=temp.next;

        }
}
```

Doubly Linked List Operations

Insertion

Deletion

Display

searching

Insertion--- insert on the first place

```
public void insert_first(int val){
      Node n=new Node(val);
      if(head==null&&tail==null){
         n.next=null;
         n.prev=null;
         head=n;
         tail=n;
}
      else{
            n.next=head;
            n.prev=null;
            head.prev=n;
            head=n;
      }
      size++;
}
```

Insertion--- insert on the last place

```
public void insert_last(int val){
      Node n=new Node(val);
      tail.next=n;
      n.prev=tail;
      tail=n;
      tail.next=null;
      size++;
}
```

Insertion--- insert on specific position

```
public void insert_pos(int val,int pos){
      temp =head;
      for(int i=1;i<pos-1;i++){
            temp=temp.next;

      }
      Node node=new Node(val);
      node.next=temp.next;
      node.prev=temp;
      temp.next.prev=node;
      temp.next=node;

}
```

Display

```
public void display_new(){
      temp=head;
      while(temp!=null){
            System.out.print(temp.value+"--->");

            last=temp;
            temp=temp.next;
      }
      System.out.println("Reverse");
      while(last!=null){
            System.out.print(last.value+"--->");
            last=last.prev;
      }

}
```

Doubly Linked List-Deletion Operation

```
public void delete_first(){
        head=head.next;
        head.prev=null;
}
```

Delete the last

```
public void delete_last(){

        last=tail.prev;
        last.next=null;
}
```

Delete on specific position

```
public void delete_position(int index){
        temp=head;
        for(int i=1;i<index-1;i++){
                temp=temp.next;
        }
        temp.next=temp.next.next;
        temp.next.prev=temp;
}
```

Searching-Search farward

```
public void search_farward(int val){
      temp=head;
      while(temp!=null){
            if(temp.value==val){
                  System.out.print("Value Present\n");

            }

            temp=temp.next;
      }
}
```