Bundelkhand University

Jhansi

Data Structure Assignment -01

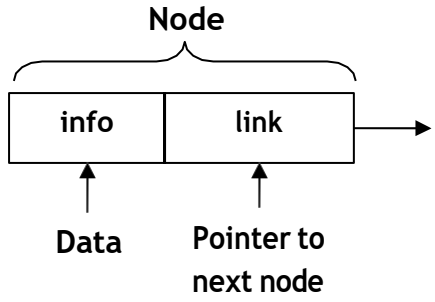
LINKED LIST

Submitted By:

- Sheetal Tyagi
- CSE (2nd year)
- 3rd semester
- Roll no.-201361031050

LINKED LIST

- A linked list is a collection of objects stored in a list form.
- A linked list is a sequence of items (objects) where every item is linked to the next.
- A linked list is a non primitive type of data structure in which each element is dynamically allocated and in which elements point to each other to define a linear relationship.
- Elements of linked list are called nodes where each node contains two things, data and pointer to next node.
- Linked list require more memory compared to array because along with value it stores pointer to next node.
- Linked lists are among the simplest and most common data structures. They can be used to implement other data structures like stacks, queues, and symbolic expressions, etc…




```
// C Structure to represent a node
struct node
{
      int info
      struct node *link
};
```

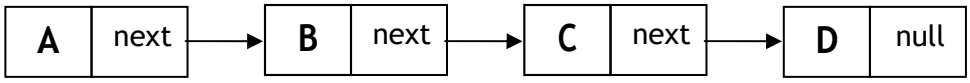
❖ Operations on linked list

- Insert
  - Insert at first position
  - Insert at last position
  - Insert into ordered list
- Delete
- Traverse list (Print list)
- Copy linked list

Types of linked list

➢ Singly Linked List

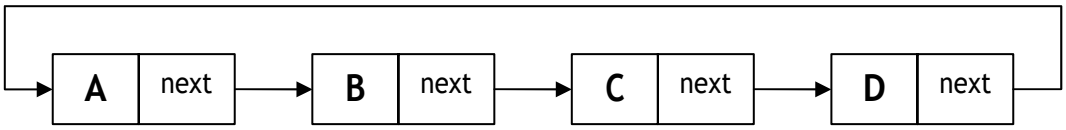
- It is basic type of linked list.
- Each node contains data and pointer to next node.
- Last node's pointer is null.
- Limitation of singly linked list is we can traverse only in one direction, forward direction.



Singly Linked List

➢ Circular Linked List

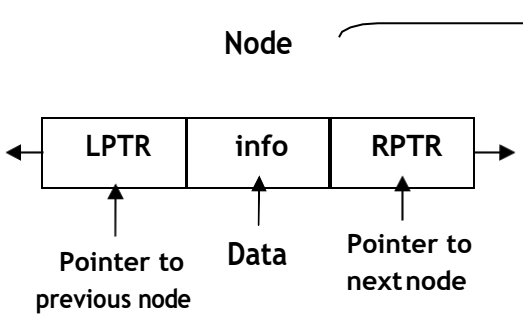
- Circular linked list is a singly linked list where last node points to first node in the list.
- It does not contain null pointers like singly linked list.
- We can traverse only in one direction that is forward direction.
- It has the biggest advantage of time saving when we want to go from last node to first node, it directly points to first node.
- A good example of an application where circular linked list should be used is a timesharing problem solved by the operating system.



Circular Linked List

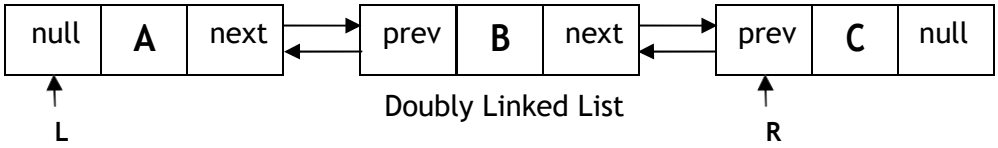
➢ Doubly Linked list

- Each node of doubly linked list contains data and two pointers to point previous (LPTR) and next (RPTR) node.




```
// C Structure to represent a node
struct node
{
      int info
      struct node *lptr;
      struct node *rptr;
};
```

- Main advantage of doubly linked list is we can traverse in any direction, forward or reverse.
- Other advantage of doubly linked list is we can delete a node with little trouble, since we have pointers to the previous and next nodes. A node on a singly linked list cannot be removed unless we have the pointer to its predecessor.
- Drawback of doubly linked list is it requires more memory compared to singly linked list because we need an extra pointer to point previous node.
- L and R in image denotes left most and right most nodes in the list.
- Left link of L node and right link of R node is NULL, indicating the end of list for each direction.



Doubly Linked List

Advantages and Disadvantages of linked list over array

- **Advantages of an array**

1. We can access any element of an array directly means random access is easy
2. It can be used to create other useful data structures (queues, stacks)
3. It is light on memory usage compared to other structures

- **Disadvantages of an array**

1. Its size is fixed
2. It cannot be dynamically resized in most languages
3. It is hard to add/remove elements
4. Size of all elements must be same.
5. Rigid structure (Rigid = Inflexible or not changeable)

- **Advantages of Linked List**

1. Dynamic size
2. It is easy to add/remove/change elements
3. Elements of linked list are flexible, it can be primary data type or user defined data types

- **Disadvantages of Linked List**

1. Random access is not allowed. We have to access elements sequentially starting from the first node. So we cannot do binary search with linked lists.
2. It cannot be easily sorted
3. We must traverse 1/2 the list on average to access any element
4. More complex to create than an array
5. Extra memory space for a pointer is required with each element of the list.

✓ Insertion & Deletion Operation

- Insertion and deletion operations are known as push and pop operation in stack and as insert and delete operation in queue.
- In the case of an array, if we have n-elements list and it is required to insert a new element between the first and second element then n-1 elements of the list must be moved so as to make room for the new element.
- In case of linked-list, this can be accomplished by only interchanging pointers.
- Thus, insertion and deletions are more efficient when performed in linked list then array.

Searching a node

- If a particular node in a linked list is required, it is necessary to follow links from the first node onwards until the desired node is found.
- Where as in the case of an array, directly we can access any node

Join & Split

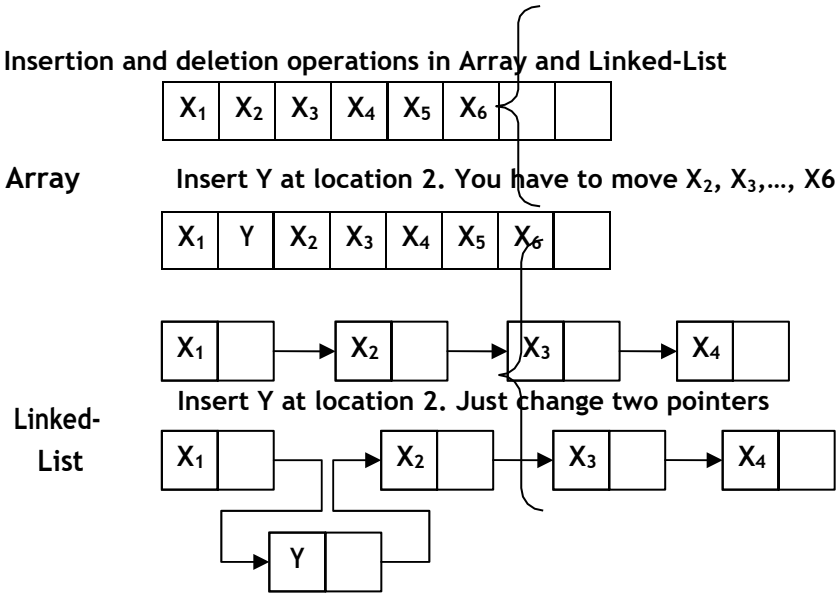
- We can join two linked list by assigning pointer of second linked list in the last node of first linked list.
- Just assign null address in the node from where we want to split one linked list in two parts.
- Joining and splitting of two arrays is much more difficult compared to linked list.

Memory

- The pointers in linked list consume additional memory compared to an array

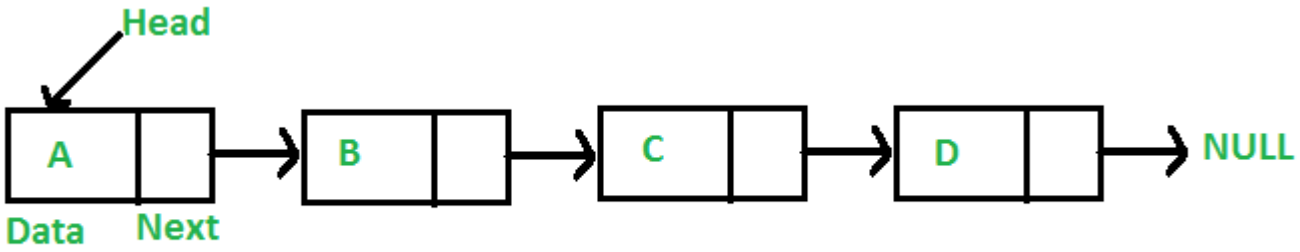
Size

- Array is fixed sized so number of elements will be limited in stack and queue.
- Size of linked list is dynamic and can be changed easily so it is flexible in number of elements



Applications of linked list data structure

A linked list is a linear data structure, in which the elements are not stored at contiguous memory locations. The elements in a linked list are linked using pointers as shown in the below image:



Applications of linked list –

1. Implementation of stacks and queues
2. Implementation of graphs : Adjacency list representation of graphs is most popular which is uses linked list to store adjacent vertices.
3. Dynamic memory allocation : We use linked list of free blocks.
4. Maintaining directory of names
5. Performing arithmetic operations on long integers
6. Manipulation of polynomials by storing constants in the node of linked list
7. representing sparse matrices