International Journal Of Engineering Research & Management Technology
ISSN: 2348-4039
September- 2014 Volume 1, Issue-5
Email: editor@ijermt.org www.ijermt.org

Dynamic Implementation Using Linked List

Karuna
Department of Information and Technology
Dronacharya College Of Engineering
Gurgaon, India

Garima Gupta
Department of Information amd Technology
Dronacharya College Of Engineering
Gurgaon, India

ABSTRACT –
Our research paper aims at linked list which is a data structure and it is the collection of nodes which together represent a sequence. Linked list are of many types. It can be singly linked list, doubly linked list, circular linked list. In this we will focus on how to traverse a linked list, insertion of a node at the beginning of the linked list, insertion of the node after a node, deletion of node from beginning and deletion of node from the end of the linked list and how two linked lists can be concatenated.

Keywords - Nodes; traverse; linked; concatenated; data structure.

1. INTRODUCTION

Data structure is an efficient way of organizing the data so that your program become efficient in terms of storage and execution. There are many types of data structures i.e., array, stack, queues, linked list, trees and graphs. We are here focusing on the linked list. Linked list is the collection of nodes and each node consist of the information part and the link part. The information part contains the data and the linked part contains the address of next node. The main benefit of linked list is that linked list elements can easily be inserted or removed without reallocation or reorganization of the entire structure because data items need not to be stored instantly in the memory. As insertion and removal of nodes can be done at any point in the list, we can do so with a constant number of operations if the link previous to the link being added or removed is maintained during list traversal. A linked list is a dynamic data structure. The number os nodes in a list are not fixed and can grow or shrink on demand. Any application which has to deal with an unknown number of objects will need to use a linked list. Linked list do not allow random access to the data by themselves or any form of indexing. Many basic operations such as obtaining the last node of the list, or finding a node that contains a given details, or locating the place where a new node should be inserted, may require scanning all of the elements of the list. Using linked list have some advantages as well as some of the disadvantages. Its advantages are that as linked lists are dynamic data structure, needed memory will be allocated when program is initiated and operations like insertion and deletion can easily be implemented using the linked list. Other data structures like stacks, queues can be easily executed using linked list. With advantages it has some disadvantages too. Like, linked list waste memory due to pointers, as they require extra storage space. Nodes in the linked list are not stored instantly. As linked list is a collection of nodes which together repesent a sequence. So, nodes in a linked list must be read in order from the beginning because linked list are inherently sequential access. In linked list, difficulties arise when it comes on reverse traversing. In reverse traversing, again memory is wasted in allocating space for a back pointer. As singly linked list is very difficult to traverse backward, whereas doubly linked list are quiet easier to read.
One way to visualize a linked list is as though it were a train. The programmer always stores the first node of the list in a pointer he won't lose access to. This would be engine of the train. The pointer itself is a connector between cars of the train. Every time the train adds a car, it uses the connectors to add a new car. This is like a programmer using malloc function to create a pointer to a new strut.

IJERMT

2. SINGLY LINKED LIST

Singly linked list is the most basic linked data structure. In this the elements can be placed anywhere in the heap memory unlike array which uses instant locations. Nodes in a linked list are linked together using a next field, which stores the address of the next node in the next field of the previous node i.e. each node of the list refers to its successor and the last node contains the NULL reference. It has a dynamic size, which can be determined only at run time. Structure of singly linked list:-

```
struct node
{
        int info;
        struct node *link;
};
struct node *first;
```

The first is a pointer which always points to very first node of the linked list.

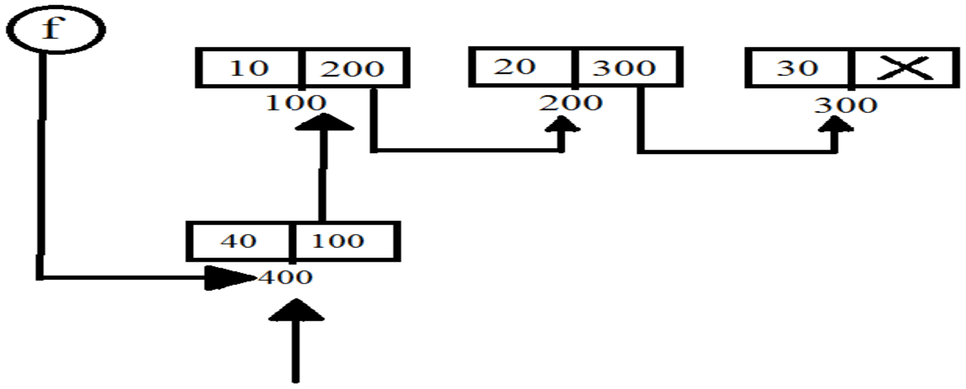
2.1 Traversing Of A Singly Linked List

Traversal of a singly link list is simple, beginning at the first node and following each next link until we come to the end.

```
void traverse()
{
        stuct node *ptr;
        ptr = first;
        while (ptr! = NULL)
{
        printf (“%d”, ptr---> info)
        ptr = ptr--->link;
}
```

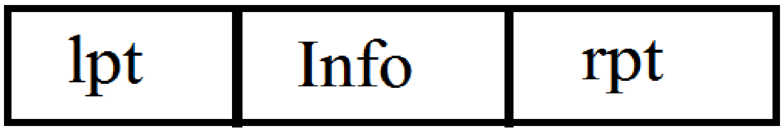
2.2 Insertion Of Node At The Beginning Of Singly Linked List

```
void insert()
{
        struct node *ptr;
        ptr=(struct node*) malloc (sizeof(struct node));
        scanf(“%d”, &ptr ---> info);
        ptr ---> link=first;
        first=ptr;
}
```



3. DOUBLY LINKED LIST

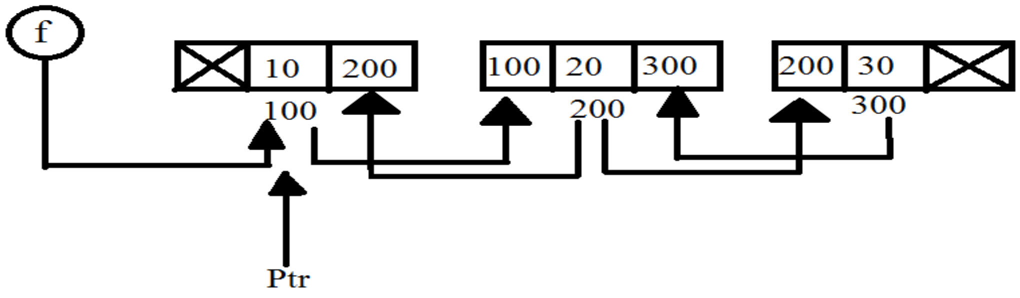
Doubly link list contains three parts, first is left pointer which contains the address of left node and next is second part is info part which contains the data and third part contains the address of next node.



```
struct node
{
        struct node *lpt;
        int info;
        srtuct node *rpt;
};
        struct node *first;
```

3.1 Backward Traversing Of Doubly Linked List

```
void traverse()
{
        struct node *ptr;
        ptr=first;
        while(ptr--->rpt!= NULL)
        ptr = ptr---> rpt
        while( ptr!= NLL)
{
        printf (“%d”, ptr---> info);
        ptr = ptr ---> lpt;
}
```



3.2 Deletion from the Beginning of Doubly Linked List

```
void delete()
{
        struct node *ptr;
        ptr=first;
        first = ptr ---> rpt
        first---> lpt = NULL
        free (ptr);
}
```

4. CIRCULAR LINKED LIST

In a circular link list, the link part of last node contains the address of first node.
In a circular linked list, all nodes are linked in continuous circle, without using NULL. Circularly linked list can be either singly or doubly linked.

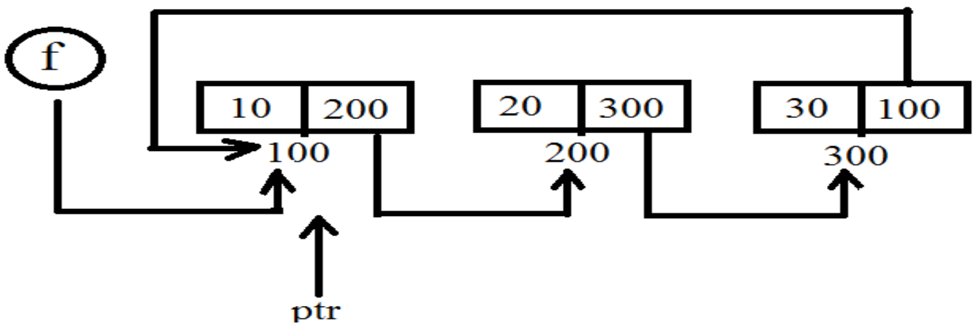
4.1 Insertion of Node at the End of the Circular Linked List

```
        Void insert()
        {
        struct node *ptr, *cpt;
        ptr= (stuct node*) malloc (sizeof(struct node));
        scanf(“%d”, &ptr ---> info);
        cpt=first;
        while(cpt---> link! = first)
        {
                cpt= cpt---> link;
        }
                cpt---> link = ptr;
                ptr---> link= first;
        }
```

4.2 Traversing Of Circular Link List

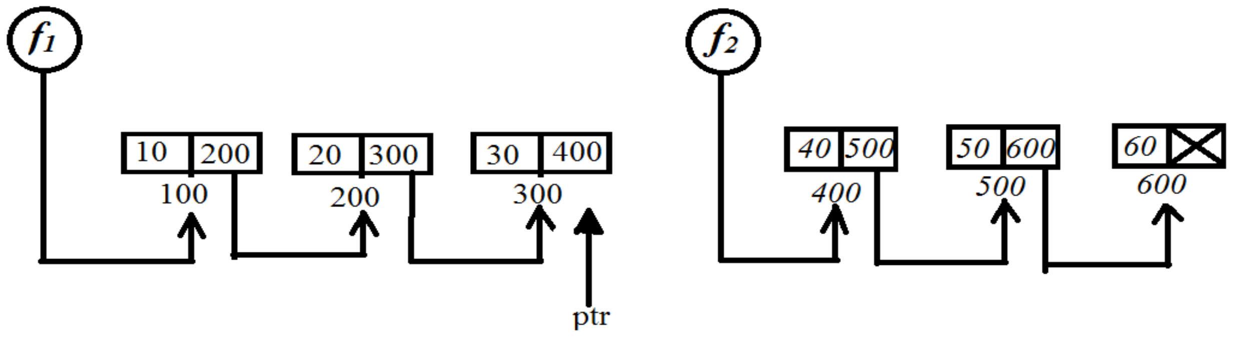
```
Void traverse ()
{
        stuct node *ptr;
        ptr = first;
        while(ptr ---> link! = first)
{
        printf (“%d”, ptr---> info);
        ptr= ptr---> link;
}
        printf(“%d”, ptr---> info);
}
```

IJERMT



5. CONCATENATION OF TWO LINKED LISTS

```
void con()
{
        struct node *ptr;
        ptr= f1 ;
        while( ptr! = NULL)
        ptr= ptr---> link;
        ptr---> link = f2;
        }
```



6. CONCLUSION

Singly linked lists are useful data structures, especially if you need to automatically allocate and de-allocate space in a list. The code and complexity of these algorithms is bigger, but the tradeoff is ease of use. As far as complexity is concerned, a linked list should never exceed O (n^2). If you are very lucky, linear time can be achieved (only under several conditions, such as there being only 1 item in the list, or the current element pointed at is the head or the tail). Sorting linked lists can be a chore, but with careful selection of sorting algorithms, nearly constant time can be achieved. Counting Sort works the best for integers, and Quick sort works great for non-integer items (such as floats).

7. REFERENCES

1. **Allen Newell, Cliff Shaw and Herbert A. Simon "Linked List".**
2. **Search engines like google and yahoo.**
3. T. Lev-Ami and S. Sagiv. TVLA: A system for implementing static analyses. InSAS 00: Static Analysis Symposium, volume 1824 ofLNCS, pages 280–301.Springer-Verlag, 2000.
4. A. Møller and M. I. Schwartzbach. The pointer assertion logic engine.In PLDI 01: Programming Language Design and Implementation,pages 221–231, 2001.
5. G. Nelson. Verifying reachability invariants of linked structures.InPOPL 83: Principles of Programming Languages, pages 38–47.ACM Press, 1983

IJERMT