Data Structures

Fall 2023

---

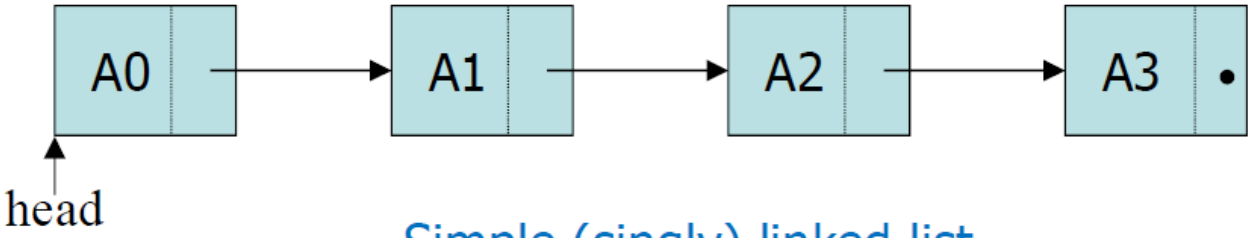
11. Circular Linked List

Introduction

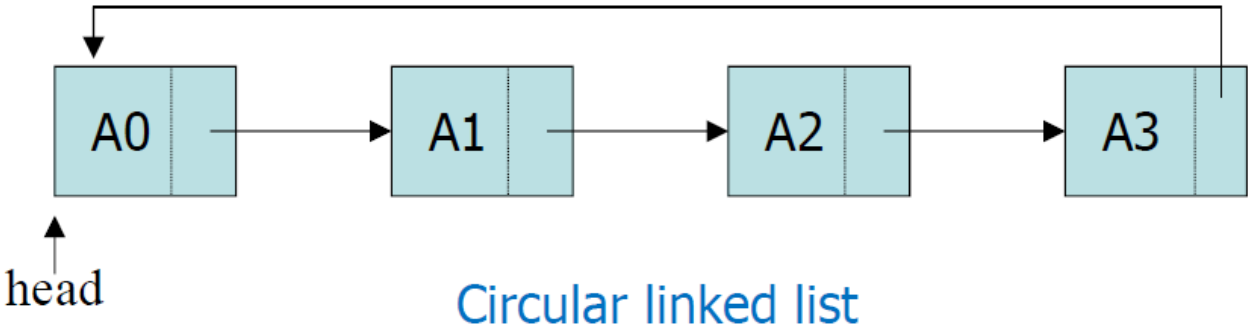
- Just like singly linked list contains only one pointer field i.e. every node holds an address of next node.
- The singly linked list is uni-directional i.e. we can only move from one node to its successor. Circular can be singly or doubly circular
- The last node is connected to first

Comparison of Linked List

- A linked list in which the last node points to the first node



Simple (singly) linked list

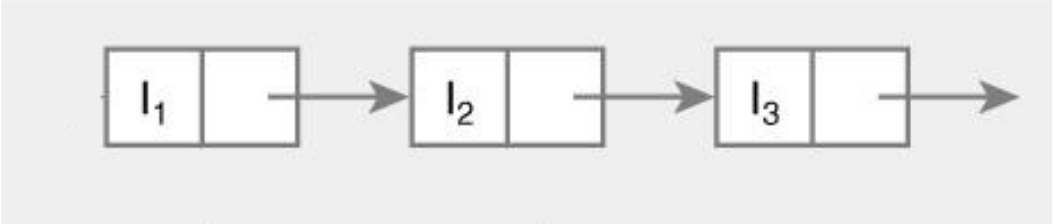


Circular linked list

Comparison of Linked List

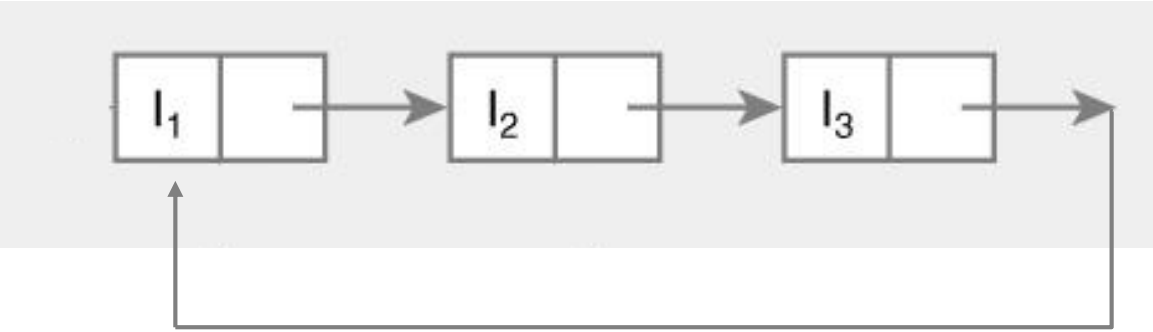
- Linked list

```
struct Node {
    int data;
    Node* next;
};
```



- Circular linked list

```
struct Node {
    int data;
    Node *next;
};
```



Linked List Class

Singly Circular

```
struct node{
    int data;
    node *next;
};

class clist{
    private:
        node *tail;

    public:
        clist()
        {
            tail=NULL;
        }
```

Doubly Circular

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

class dclist{
    private:
        node *tail;

    public:
        dclist()
        {
            tail=NULL;
        }
```

Insertion

- In insertion process, element can be inserted in three different places
  - At the beginning of the list
  - At the end of the list
  - At the specified position.

Insertion at head

Singly Circular

```
void add_to_head(int v)
{
    node *temp=new node;
    temp->data=v;
    temp->next=NULL;

    if(tail==NULL)
    {
        tail=temp;
        tail->next=temp;
    }
    else
    {
        temp->next=tail->next;
        tail->next=temp;
    }
}
```

Doubly Circular

```
void add_to_head(int v)
{
    node *temp=new node;
    temp->data=v;
    temp->prev=NULL;
    temp->next=NULL;

    if (tail==NULL) {
        tail=temp;
        tail->prev=temp;
        tail->next=temp;
    }
    else {
        temp->next=tail->next;
        temp->prev=tail;
        tail->next->prev=temp;
        tail->next=temp;
    }
}
```

Insertion at tail

Singly Circular

```
void add_to_tail(int v)
{
    node *temp=new node;
    temp->data=v;
    temp->next=NULL;

    if(tail==NULL)
    {
        tail=temp;
        tail->next=temp;
    }
    else
    {
        temp->next=tail->next;
        tail->next=temp;
        tail=temp;
    }
}
```

Doubly Circular

```
void add_to_tail(int v)
{
    node *temp=new node;
    temp->data=v;
    temp->prev=NULL;
    temp->next=NULL;

    if (tail==NULL) {
        tail=temp;
        tail->prev=temp;
        tail->next=temp;
    }
    else {
        temp->next=tail->next;
        temp->prev=tail;
        tail->next->prev=temp;
        tail->next=temp;
        tail=temp;
    }
}
```

Insertion at location

Singly Circular

```
void add_to_loc(int l,int v)
    {
        node *temp=new node;
    temp->data=v;
    temp->next=NULL;

    if(tail==NULL)
    {
        tail=temp;
        tail->next=temp;
    }
    else
    {
    node *pre=tail,*curr=tail;
    for(int i=1;i<=l;i++)
        {
            pre=curr;
            curr=curr->next;
        }
        pre->next=temp;
        temp->next=curr;
    }
```

Doubly Circular

```
void add_to_loc(int l, int v)
{
    node *temp=new node;
    temp->data=v;
    if (tail==NULL) {
        tail=temp;
        tail->prev=temp;
        tail->next=temp;
    }
    else {
        node *pre=tail,*curr=tail;
        for(int i=1;i<=l;i++) {
            prev=curr;
            curr=curr->next;
        }
        prev->next=temp;
        temp->next=curr;
        curr->prev=temp;
        temp->prev=prev;
    }
}
```

Deletion

- In deletion process, element can be deleted from three different places
  - From the beginning of the list
  - From the end of the list
  - From the specified position in the list.

- When the node is deleted, the memory allocated to that node is released and the previous and next nodes of that node are linked

Deletion at head and end

Singly Circular

```
void delete_at_start()
{
    node *q=tail->next;
    tail->next=q->next;
    delete q;
    q=NULL;
}

void delete_at_end()
{
    node *q=tail;
    node *temp=tail;
    while(temp->next!=tail)
    {
        temp=temp->next;
    }
    tail=temp;
    tail->next=q->next;
}
```

Doubly Circular

```
void delete_at_start()
{
    node *q=tail->next;
    tail->next=q->next;
    q->next->prev=q->prev;
    delete q;
}

void delete_at_end()
{
    tail=tail->next;
    node *q=tail->next;
    tail->next=q->next;
    q->next->prev=q->prev;
    delete q;
}
```

Display Linked List

Singly Circular

```
void display()
{
    node *p=tail->next;
do
{
        cout<<p->data<<"\t";
        p=p->next;
        }while(p!=tail->next);
}
```

Doubly Circular

```
void display()
{
    node *p=tail->next;
    do
    {
        cout<<p->data<<“\t”;
        p=p->next;
    } while(p!=tail->next);
}

void display_reverse()
{
    node *p=tail;
    do
    {
        cout<<p->data<<“\t”;
        p=p->prev;
    } while(p!=tail);
}
```

Advantages

- Whole list can be traversed by starting from any point
  - Any node can be starting point
  - What is the stopping condition?
- Fewer special cases to consider during implementation
  - All nodes have a node before and after it
- Used in the implementation of other data structures
  - Circular linked lists are used to create circular queues
  - Circular doubly linked lists are used for implementing Fibonacci heaps

Disadvantages

- Finding end of list and loop control is harder
  - No NULL to mark beginning and end

Any Question So Far?