

Unit 2

Linked Lists

	Syllabus	Guidelines	Suggested number of lectures
1	Singly-linked, doubly-linked and circular lists, analysis of insert, delete and search operations in all the three types	Chapter 3, Section 3.2, 3.3, 3.4, Ref 2	
2	Implementing sparse matrices.	Chapter 7 (Section 7.4.upto pg. no. 253) Additional Resource 3	12
3	Introduction to Sequences.	Chapter 6, Section 6.3, Ref 2	

References

- 1. Ref 1:** . Drozdek, A., (2012), *Data Structures and algorithm in C++*. 3rd edition. Cengage Learning. Note: 4th edition is available. Ebook is 4th edition
- 2. Ref 2.:** Goodrich, M., Tamassia, R., & Mount, D., (2011). *Data Structures and Algorithms Analysis in C++*. 2nd edition. Wiley.
- 3. Additional Resource 3:** Sahni, S. (2011). Data Structures, Algorithms and applications in C++. 2nd Edition, Universities Press
- 4. Additional Resource 4:** Tenenbaum, A. M., Augenstein, M. J., & Langsam Y., (2009), *Data Structures Using C and C++*. 2nd edition. PHI.

Note: Ref1, Additional resource etc. as per the LOCF syllabus for the paper.

2 ch-3. Linked Lists (A. Drozdek).

Limitation of Arrays

- 1) Its size has to be known at compilation time.
- 2) The data in the array are separated in computer memory by the same distance.

These limitations can be overcome by using linked list structures.

Linked structure :- is a collection of nodes storing data & links to other nodes.

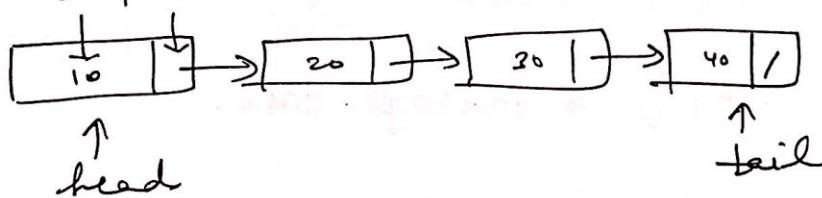
Types of Linked List

- 1) Singly linked list.
- 2) Doubly linked list.
- 3) circular singly l.list.
- 4) circular doubly l.list.

I) Singly linked list

Sequence of nodes, in which each node is holding some information & pointer to another node (successor) in the list, is called singly linked list.

Info next.



self-referential objects :- 'node' is defined in terms of itself because one data member 'next' is pointer to a node of the same type. Objects that include such datamembers are called self-referential objects.

```

class node
{
    public:
        int info;
        node* next;
    };
}

```

Analysis :-

```

Reverse()
{
    u = last;
    do
    {
        u = first;
        while (u->ptr != u)
            u = u->ptr;
        u->ptr = u;
        u = u->ptr;
    } while (u != first);
    first->ptr = NULL;
    temp = last;
    last = first;
    first = temp;
}

```

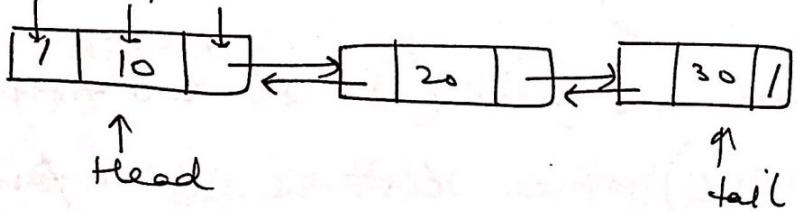
- 1) addtohead() :- $O(1)$ no. of operation performed by these
- 2) addtotail() :- $O(1)$: { two fⁿ does not exceed some constant no. 'c'.
- 3) Deletefromhead() :- $O(1)$ [Best case]
- 4) Deletefromtail () :- Since for loop performs ' $n-1$ ' iterations in a list of ' n ' nodes to reach second last node , it takes $O(n)$ time to delete last node. [worst case]
- 5) searchnode() :- $O(1)$ → Best case (first node)
 $O(n)$ → worst case (last node or no node)
 $O(n)$ → Average case.

3.2. Doubly linked list

Each node in the list has two pointers :-

one to the successor & one to the predecessor.

prev. info next



close node

{ public:

```
int info;  
node* prev;  
node* next;  
};
```

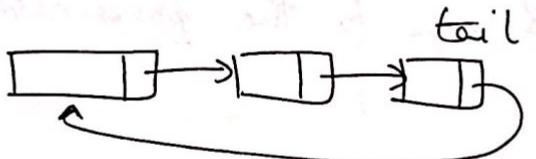
Analyze :-

- 1) Addto head() = O(1)
- 2) Addto tail = O(1)
- 3) Delete from head() = O(1)
- 4) delete from tail() = O(1)
- 5) search node(-) = O(n).

3.3 circular lists

@ C. singly.

All the nodes form a ring structure.

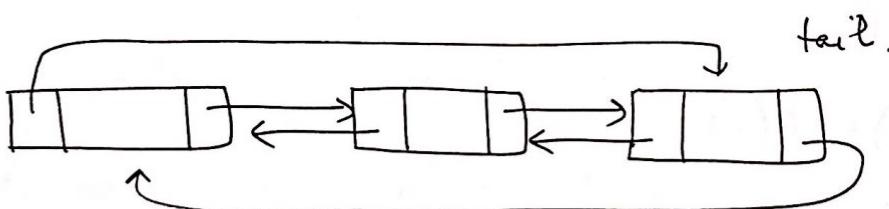


single pointer 'tail' is there (& not two pointers - head / tail as in SLL) since list is in the form of ring.

Analysis:-

- 1) Insert at begin() = $O(1)$
- 2) Insert at end() = $O(1)$
- 3) Delete from tail() = $O(n)$
- 4) Delete from head() = $O(1)$

⑥ circular Doubly linked list



The list forms two rings:-

one going forward through 'next' members and one going backward through 'prev' members.

Analysis

- 1) Insert at end() = $O(1)$
- 2) Insert at begin() = $O(1)$
- 3) Delete from tail() = $O(1)$
- 4) Delete from head() = $O(1)$.

Destructors

① for singly link list:-

$\sim \text{SLL}();$ // declaration.

$\text{SLL}\langle t \rangle :: \sim \text{SLL}()$

{

for ($\text{node}\langle t \rangle * \text{temp}; !\text{isempty}();$)

{

$\text{temp} = \text{first} \rightarrow \text{next};$

$\text{delete first};$

$\text{first} = \text{temp};$

}

}

class SLL

{

≡

$t \text{ isempty}()$

{

$\text{return } (\text{first} == \text{NULL});$

}

}

changes in Linked List when Templates are used

① template < class X >
 class node
 {
 }

template < class X >
 class list
 {

→ node < X > * head, * tail;

}

② List < X > * operator + (List < X > &);

③ Def of f's.

template < class X >
 void list < X > :: reverse()
 {

}

④ template < class X >

list < X > * list < X > :: operator + (list < X > & p)
 {

}

⑤ Destructor

~SLL () // using for

{ for (node* p; head != 0;)

{ p = head -> next;

delete head;

head = p;

void main()

{ cout << "welcome to integer format";

SLL < int > ob, ob1;

;

;

cout << "character format,"

SLL < char > ob3, ob1;

;

;

}

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

// using while

~SLL ()

{ node* p;

while (head != 0)

{ p = head -> next;

delete head;

head = p;

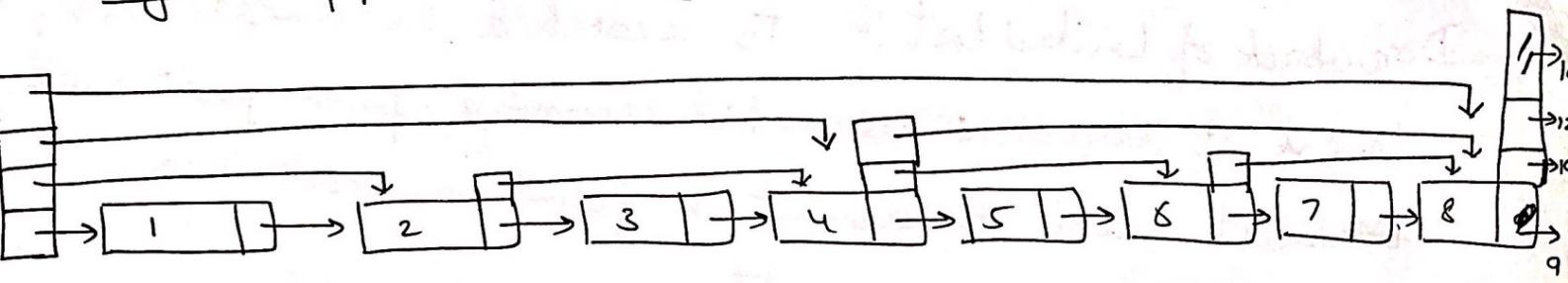
3.4 skip lists

Drawback of Linked List :- To search a particular element, it required sequential scanning from first node till last until the element is found or list has terminated giving no result.

Solution :-

- 1) ordering elements (ascending / desc) on the list can speed up searching , but sequential search is still required.
- 2) There can be a list that allow for skipping certain nodes to avoid sequential processing .
- 3) A skip list is one of the solⁿ of ordered linked list that makes such a non-sequential search possible.
- # In a skip list of ' n ' nodes, for each ' k ' & ' i ', such that $1 \leq k \leq \lfloor \log_2 n \rfloor$ and $1 \leq i \leq \lfloor n / 2^{k-1} \rfloor - 1$
- # The node in position $2^{k-1}(i)$ $\xrightarrow{\text{Points to}}$ the node in pos. $2^{(k-1)}(i+1)$

Eg. Suppose $n=8$ (8 nodes in list)



Now,

$$(\alpha^{k-1})(i) \xrightarrow{\text{pts. to}} (\alpha^{k-1})(i+1)$$

Also, $1 \leq k \leq \lfloor \log_2 n \rfloor$ ie $1 \leq k \leq \lfloor \log_2 8 \rfloor$ or
 $1 \leq k \leq 3$.

and.

$$1 \leq i \leq \lfloor n / \alpha^{k-1} \rfloor - 1 \text{ ie if } k=1, \text{ then}$$

$$1 \leq i \leq \lfloor 8 / 1 \rfloor - 1 \text{ ie } 1 \leq i \leq 7.$$

$$K=1, i = 1 \dots 7.$$

$$K=1, i = 1, \quad \alpha^{1-1}(1) \xrightarrow{\text{pts. to}} \alpha^{1-1}(2) \text{ ie } 1 \rightarrow 2$$

$$i = 2, \quad 2 \longrightarrow 3$$

$$i = 3, \quad 3 \longrightarrow 4$$

:

$$i = 7, \quad 7 \longrightarrow 8.$$

$$K=2, i \text{ is } 1 \leq i \leq \lfloor 8 / \alpha^1 \rfloor - 1 \text{ ie } 1 \leq i \leq 3.$$

$$K=2, i = 1, \quad \alpha^{2-1}(1) \xrightarrow{\text{pts. to}} \alpha^{2-1}(1+1) \text{ ie } 2 \xrightarrow{\text{pts. to}} 4$$

$$i = 2, \quad 4 \longrightarrow 6$$

$$i = 3, \quad 6 \longrightarrow 8.$$

$$K=3, i \text{ is } 1 \leq i \leq \lfloor 8 / \alpha^2 \rfloor - 1 \text{ ie } 1 \leq i \leq 1$$

$$K=3, i = 1, \quad \alpha^{3-1}(1) \xrightarrow{\text{pts. to}} \alpha^{3-1}(2) \text{ ie } 4 \longrightarrow 8$$

Half of the nodes have just one pointer (4) ie nodes 1, 3, 5, 7.
 one-fourth of the nodes have two pointers (2) ie nodes 2, 6.
 one-eighth of the nodes have three pointers (1) ie node 4.

The no. of pointers indicates the level of each node.

ie node 1 has level 1

node 2 has level 2 & so on.

maximum levels :- $\lfloor \log_2 n \rfloor + 1$ $\lfloor \log_2 8 = 3 \rfloor$ i.e. $3+1=4$

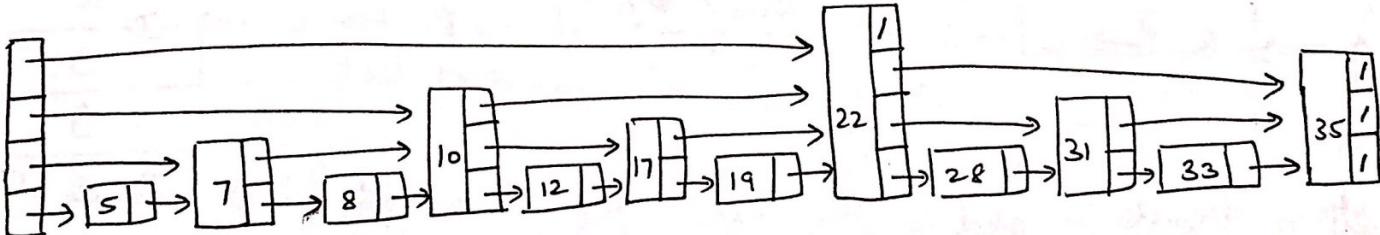
last node has 4 levels.
 (8)

~~★ $\lfloor \log_2 n \rfloor + 1$~~

Disadvantages of skip list

Design of skip list can lead to very inefficient insertion & deletion procedures.

e.g. $n = 12$



3.5 self-organizing lists

we can improve the efficiency of search by dynamically organizing the list in a certain manner. There are many different ways to organize the lists :-

- 1) move-to front method :- After the desired element is located, put it at the beginning of the list.

$$A \rightarrow B \rightarrow C \rightarrow D \xrightarrow{\text{Access } D} D \rightarrow A \rightarrow B \rightarrow C.$$

- 2) Transpose method :- After the desired element is located, swap it with its predecessor, unless it is at the head of the list.

eg 1. $A \rightarrow B \rightarrow C \rightarrow D \xrightarrow{\text{Access } D} A \rightarrow B \rightarrow D \rightarrow C$

eg 2. $A \rightarrow B \rightarrow C \rightarrow D \xrightarrow{\text{Access } A} A \rightarrow B \rightarrow C \rightarrow D.$
(A is at the head of list)

- 3) count method :- order the list by the no. of times elements are being accessed

$$\boxed{A \atop 3} \rightarrow \boxed{B \atop 1} \rightarrow \boxed{C \atop 1} \rightarrow \boxed{D \atop 1} \xrightarrow{\text{Access } D} \boxed{A \atop 3} \rightarrow \boxed{D \atop 2} \rightarrow \boxed{B \atop 1} \rightarrow \boxed{C \atop 1}$$

- 4) ordering method :- order the list using certain criteria natural for the information under scrutiny

$$A \rightarrow B \rightarrow D \xrightarrow{\text{Access } C} A \rightarrow B \rightarrow C \rightarrow D$$

(5)

In first three methods, new inf" is stored in a node added to the end of list.

for eg:-

$A \rightarrow B \rightarrow D$. Access C
move-to-front $A \rightarrow B \rightarrow D \rightarrow C$

Access C
transpose $A \rightarrow B \rightarrow D \rightarrow C$

Access C
count $A \rightarrow B \rightarrow D \rightarrow C$

But in fourth method, new inf" is stored in a node inserted somewhere in the list to maintain the order of the list.

ie

$A \rightarrow B \rightarrow D$ Access C
ordering $A \rightarrow B \rightarrow C \rightarrow D$

eg. element searched for		move to front	transpose	count	ordering
A	A	A	A	A	A
C	AC	AC	AC	AC	AC
B	ACB	ACB	ACB	ACB	ABC
C	ACB	CAB	CAB	CAB	ABC
D	ACBD	CABD	CABD	CABD	ABCD
A	ACBD	ACBD	ACBD	ACBD	ABCD
D	ACBD	DACB	ACDB	ACDB	ABCD
A	ACBD	ADCB	ACDB	ACDB	ABCD
C	ACBD	CADB	CADB	CADB	ABCD
A	ACBD	ACDB	ACDB	ACDB	ABCD
C	ACBD	CADB	CADB	(ACDB)★	ABCD
C	ACBD	CAJB	CADB	CAJB	ABCD
E	ACBDE	CADBE	CADBE	CAJBE	ABCDE
E	ACBDE	E CADB	CADEB	CAJD B	ABCDE

special matrices

A square matrix has same no. of rows & columns.

Special form of square matrices are:-

1) Diagonal :- M is diagonal iff $m(i,j) = 0$ for $i \neq j$.

$$i \begin{bmatrix} & j \\ 2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 6 \end{bmatrix}$$

2) Tridiagonal :- M is tridiagonal iff $m(i,j) = 0$ for $|i-j| \geq 1$.

$$i \begin{bmatrix} & j \\ 2 & 1 & 0 & 0 \\ 3 & 1 & 3 & 0 \\ 0 & 5 & 2 & 7 \\ 0 & 0 & 9 & 0 \end{bmatrix}$$

3) Lower Triangular :- M is lower-Triangular iff $m(i,j) = 0$ for $i < j$.

$$\begin{bmatrix} & & & \\ 2 & 0 & 0 & 0 \\ 5 & 1 & 0 & 0 \\ 0 & 3 & 1 & 0 \\ 4 & 2 & 7 & 0 \end{bmatrix}$$

4) Upper Triangular :- M is upper-Triangular iff $m(i,j) = 0$ for $i > j$.

$$\begin{bmatrix} & & & \\ 2 & 1 & 3 & 0 \\ 0 & 1 & 3 & 8 \\ 0 & 0 & 1 & 6 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

5) Symmetric :- Matrix M is symmetric iff ~~$M_{i,j} \neq M_{j,i}$~~
 $M(i,j) = M(j,i)$ for all $i & j$.

$$\begin{bmatrix} 2 & 4 & 6 & 0 \\ 4 & 1 & 9 & 5 \\ 6 & 9 & 4 & 7 \\ 0 & 5 & 7 & 0 \end{bmatrix}$$

Sparse Matrices

- # An $m \times n$ matrix is said to be sparse if "many" of its elements are zero.
- # A matrix that is not sparse is dense.
- # eg:- Diagonal & Tridiagonal matrices are sparse.
each has $O(n)$ non-zero terms & $O(n^2)$ zero terms.
- # Triangular matrix (lower & upper) has -
 - at least $\frac{n(n-1)}{2}$ zero terms &
 - at most $\frac{n(n+1)}{2}$ non-zero terms.
- # The no. of non-zero terms in SPARSE matrices need to be less than $n^2/3$ & in some cases $n^2/5$.
In this context, triangular matrices are dense.

e.g:- 4×8 sparse matrix its representation

$$\begin{bmatrix} 0 & 0 & 0 & 2 & 0 & 0 & 10 \\ 0 & 6 & 0 & 0 & 7 & 0 & 0 \\ 0 & 0 & 0 & 9 & 0 & 8 & 0 \\ 0 & 4 & 5 & 0 & 0 & 0 & 0 \end{bmatrix}$$

a[]	0	1	2	3	4	5	6	7	8
row	1	1	2	2	2	3	3	4	4
col	4	7	2	5	8	4	6	2	3
value	2	1	6	7	3	9	8	4	5

linked Representation

first

