Unit-1

Advanced Linked List

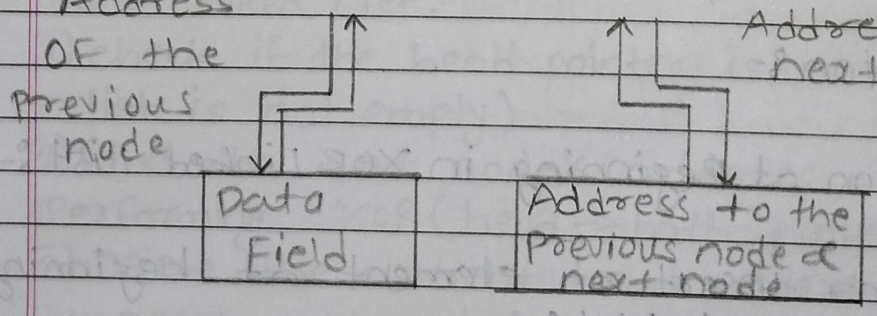
Page No.
Date

Que- Explain Memory Efficient Doubly Linked List or XOR Linked List with operations, Advantage Disadvantage and complexity., Application.

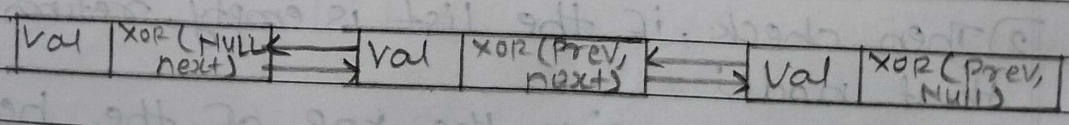
⇒

Memory Efficient Doubly Linked List or XOR Linked List :-

- A memory-efficient doubly linked list also known as an XOR linked list, store only one address per node instead of two.
- In a doubly linked list, each node contain two pointers - one for the Previous node and one for the next node.
- In an XOR linked list, instead of storing two pointers, each node stores the XOR of the addresses of the Previous and next nodes.
- Keep track of the address of the Previous ~~node~~ node which can be computed using the XOR of the current node's next pointer.

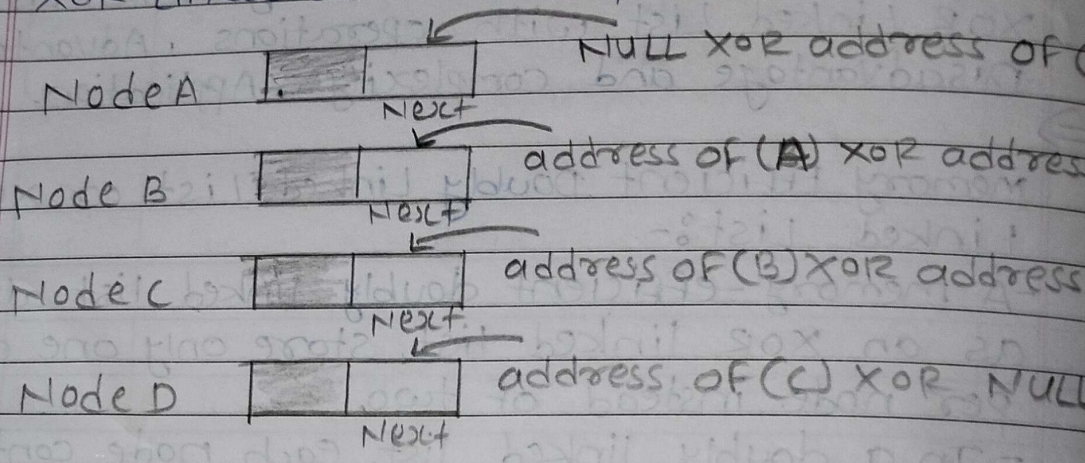


XOR Linked List :-



Page No.
Date

XOR Linked List node Representation:-



- Uses Bitwise XOR (^) operation to save memory.
- Each node consists of two main components
  - Data (Payload of that node)
  - XOR (memory address of the previous and next nodes)

- operations of XOR Linked List

1) Insertion:-

a) Insertion at Beginning in XOR Linked List:-

steps for insert an element at beginning in XOR Linked List:-

1) Create a new node, initialize the data and address to the (NULL ^ address of head) that node
2) Then check, if the list is empty, return with
3) otherwise, assign the XOR of the head node to the

XOR (new_node address, XOR (head -> both, nullptr))

b] Insertion at end in XOR Linked List :-

steps for insert an element at end in XOR Linked List :-

1] create a new node, initialize the data and address to the (Null ^ address of tail)
2] Then check if the list is empty, return with that node
3] otherwise, assign the XOR of the tail node to the
XOR (XOR (tail -> both, nullPtr), new-node address)

2] Deletion :-

a] Deletion at Beginning in XOR Linked List :-

steps for delete an element at beginning in XOR Linked List :-

1] check if the head pointer is not null (i.e. the list is not empty)
2] Find the next node's address using XOR by Performing XOR (head -> both, nullPtr)
3] Delete the current head node to free up the memory and update the head pointer to Point to the calculated next node.

b] Deletion at End in XOR Linked List :-

steps for delete an element at beginning in XOR Linked List :-

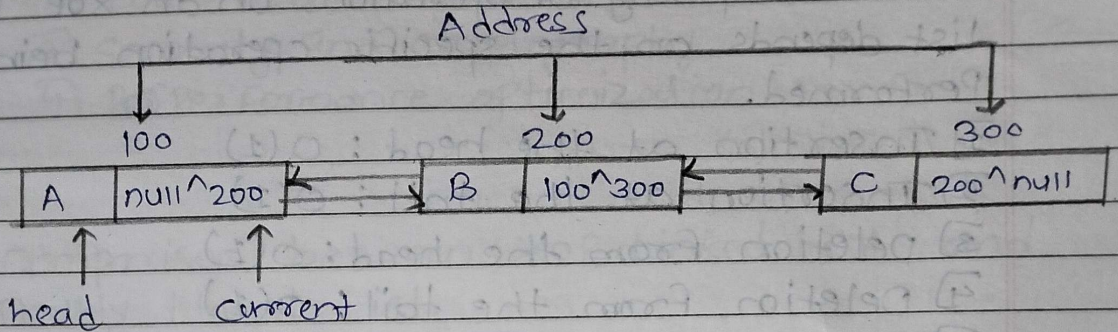
1] Check if the tail pointer is not null (i.e. the list is not empty)
2] If the list is not empty :-
a] Find the previous node's address using XOR by performing XOR (tail → both, null ptr)
This gives you the previous node in the list
b] Delete the current tail node to free up the memory.
c] update the tail pointer to point to the calculated previous node.

3] Traversal :-
Two types of traversal :-
1] Forward Traversal :-
2] Backward Traversal

1] Forward Traversal :-
-When traversing the list forward, it's important to always keep the memory address of the previous element.
-Address of previous element helps in calculating the address of the next element by the below formula:

address of next node = (address of Prev node) ^ (both

- Here, "both" is the XOR of address of previous node and address of next node



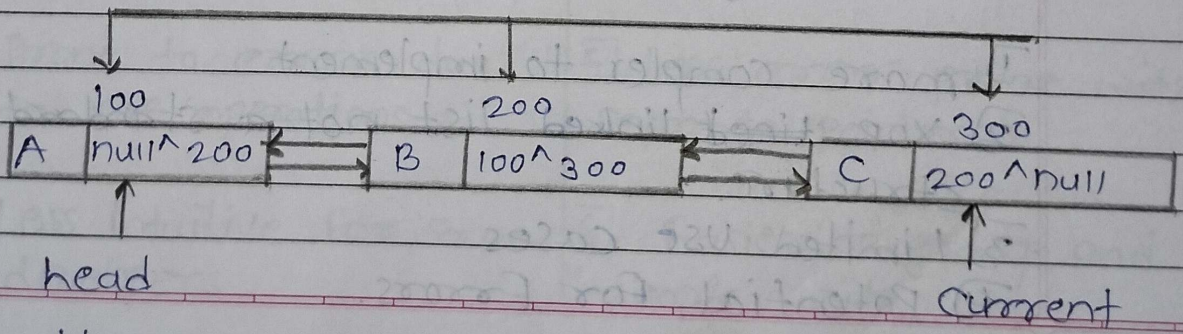
address of next = (prev)^(address of current)

2) Backward Traversal:

- When traversing the list backward, it's important to always keep the memory address of the next element.
- Address of next element helps in calculating the address of the previous element by the below formula:

address of previous Node = (address of next node)^(both)

- Here, "both" is the XOR of address of previous node and address of next node.



address of Prev = (next)^(address of current)

Page No.
Date

Complexity :-

- Time complexity of operations on an XOR linked list depends on the specific operation being performed.

1] Insertion at the head : O(1)
2] Insertion at the tail : O(1)
3] Deletion from the head : O(1)
4] Deletion from the tail : O(1)
5] Traversal : O(n)

- space complexity of XOR linked list is O(n).

Advantages :-

1] Memory Efficiency
2] Traversal Flexibility
3] XOR Linked lists is that we can move forward and backward along the list.
4] uses less memory
5] insertion and deletion done in constant time O(1).

Disadvantage :-

1] more complex to implement
2] XOR ~~tined~~ linked list not a standard data structure.
3] Limited use cases
4] Potential for Errors.

Application :-
1] Memory-constrained Environment
2] Low-level programming
3] Security and cryptography
4] Performance optimization.

Que- Comparison between XOR Linked List and Doubly Linked List.

⇒

	XOR Linked List	Doubly Linked List
1]	Consumes less memory	1] Requires more memory
2]	Requires bitwise XOR operation for traversal	2] straightforward next/prev pointers for Traversal
3]	Efficient due to adjacent node pointers for Insertion/Deletion	3] straightforward but may involve more adjustment for insertion/deletion
4]	More complex due to XOR operations	4] simpler to implement and understand
5]	Prone to errors due to XOR operations	5] Less error-prone with separate pointers
6]	Less intuitive for some developers	6] Easier to grasp and work with.

10/03/2024

Page No.
Date

Que- Explain skip List with example. Explain insertion and deletion operation with complexity, advantages, Disadvantage and Application.

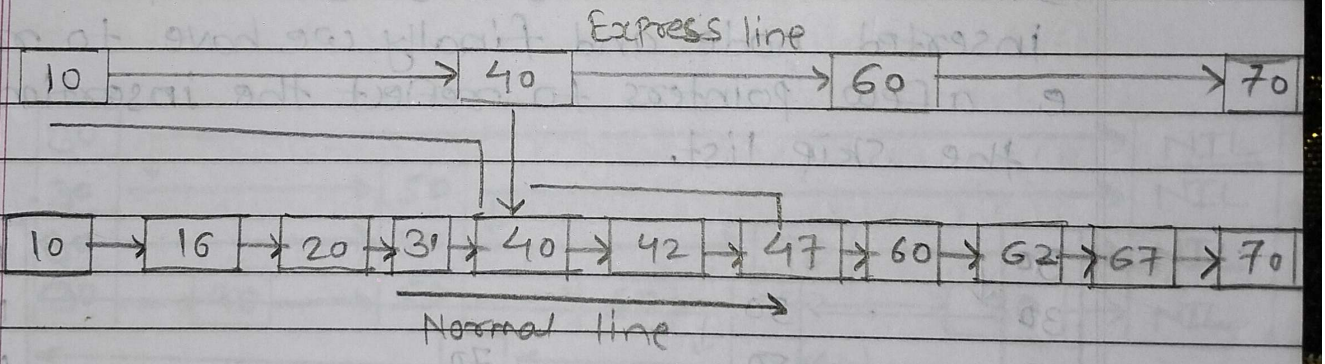
→

Skip List :-

- A skip list is a probabilistic data structure.
- The skip list is used to store a sorted list of elements or data with a linked list.
- It allows the process of the elements or data to view efficiently.
- In one single step, it skips several elements of the entire list, which is why it is known as skip list.
- It consists of a base list that includes a set of elements which maintains the link hierarchy of the subsequent elements.
- A skip list is a data structure that allows for efficient search, insertion and deletion of elements in a sorted list.
- In a skip list, elements are organized in layers, with each layer having a smaller number of elements than the one below it.
- The bottom layer is a regular linked list, while the layers above it contain "skipping" links that allow for fast navigation to elements that are far apart in the bottom layers.
- The idea behind this is to allow for quick traversal to the desired element, reducing the average number of steps needed to reach it.
- Skip lists are implemented using a technique called "coin flipping".

- In this technique, a random number is generated for each insertion to determine the number of layers the new element will occupy.

Example :-
Suppose you want to find 47 in this example. You will start the search from the first node of the express line and continue running on the express line until you find a node that is equal a 47 or more than 47.
- You can see in the example that 47 does not exist in the express line, so you search for a node of less than 47, which is 40. Now you go to the normal line with the help of 40 and search the 47, as show in the diagram.



operation in the skip list :-

1) Insertion :-
- Insertion of a node in a skip list needs to be in such a manner that after insertion of the new node the list should still be in sorted order and there is a randomization method based on the probability to ensure that skewness will never happen and we will never lose the efficiency of operations on skip list.

- we first need to find a particular node in the bottom linked list, only then we will insert the new node at the appropriate location:-

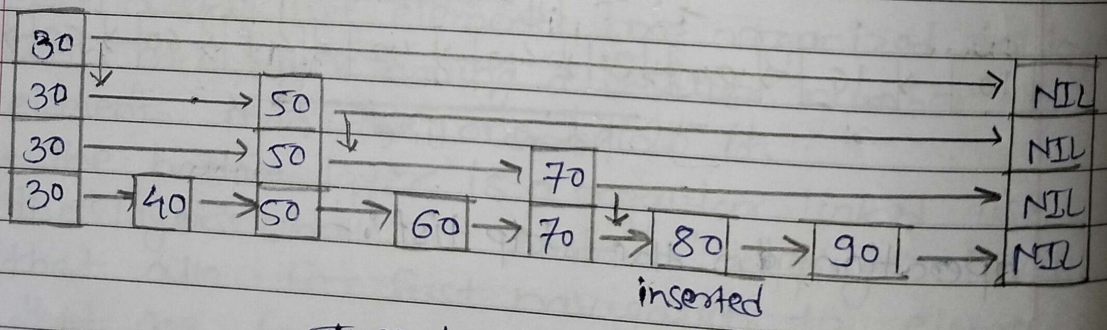
- insertion algorithm in steps:-

1] we will iterate through the layer till we find an element that is just greater than the new node we are trying to insert.

2] once we find such a node we will use its pointer to go down to the lower layer.

3] we will repeat steps 1 and 2 till we reach the bottom-most layer, once we reached the required node while performing step 1, do the insertion of new node.

4] A random level will be chosen for the inserted node and finally we have to rearrange a few pointers to reflect the insertion in the skip list.

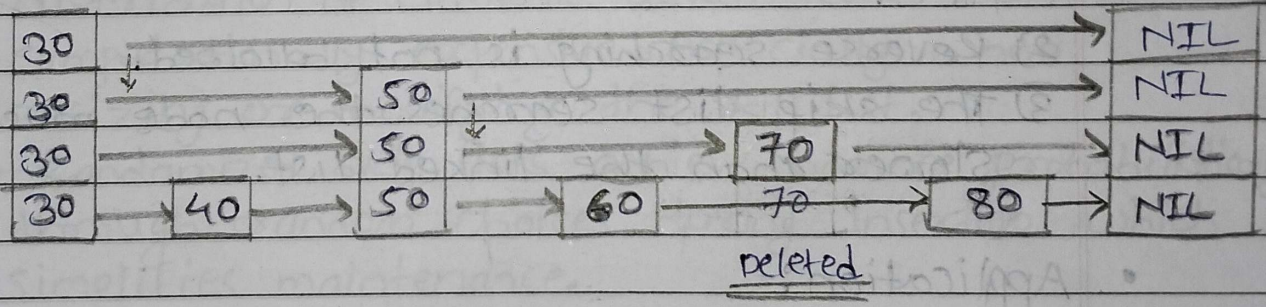


Insert element 80

Page No.
Date

2) Deletion :-
- For deletion operation, similar to insertion, we will first try to find the node to be deleted and then we will do some pointers rearrangement to remove the node from the skip list.

Steps for Deletion Algorithm :-
1) we will iterate through the layer till we find an element that is just greater than the node we are trying to delete.
2) once we find such a node we will use its pointer to go down to the lower layer.
3) we will repeat steps 1 and 2 till we reach the bottom-most layer, once we reached the required node while performing step 1, do remove that node.



deleted element 70

- Complexity :-
Time complexity :-
1) Insertion — O(logn)
2) Deletion — O(logn)
space complexity :-
O(nlogn)

- Advantages:-
1) Insertion of node is very fast because there are no rotations in the skip list.
2) Skip list is simple to implement as compared to the hash table.
3) It is very simple to find a node in the list because it stores the nodes in sorted form.
4) The skip list algorithm can be modified very easily in a more specific structure, such as indexable skip lists, trees or priority queues.
5) The skip list is a robust and reliable list.

- Disadvantage:-
1) It requires more memory than the balanced tree.
2) Reverse searching is not allowed.
3) The skip list searches the node much slower than the linked list.

- Application:-
1) It is used in distributed applications, and it represents the pointers and system in the distributed applications.
2) It is used to implement a dynamic elastic concurrent queue with low lock contention.
3) It is also used with the cmap template class.
4) The indexing of the skip list is used in running median problem.
5) The skip List is used for the delta-encoding Posting in the Lucene search.

Que.- ~~Explain perfect skip list with the help of example.~~ why random skip list is preferred over perfect skip list?

⇒

~~Perfect skip list~~ :-
Randomized skip lists are often preferred over perfect skip lists for several reasons:

1] Simplicity of Implementation :-
- Randomized skip lists are easier to implement compared to perfect skip lists.
- They do not require maintaining a perfect distribution of elements across levels, which simplifies the insertion and deletion.

2) Balancing :-
- In perfect skip lists, maintaining a perfect distribution of elements across levels can be computationally expensive, especially when resizing or modifying the skip list.
- Randomized skip lists achieve balance probabilistically through random choices during insertion, which simplifies maintenance.

3) Performance :-
- Randomized skip lists offer similar performance guarantees as perfect skip lists with average time complexities of $O(\log n)$ for search, insertion and deletion.
- However, the randomized nature of skip lists may lead to better performance in practice, especially in dynamic environments where the distribution of element changes frequently.

4) Flexibility :-
- Randomized skip lists are more flexible in terms of tuning parameters such as the probability of adding a level during insertion (p).
- This allows developers to adjust the skip list's behavior based on specific requirements and performance considerations.

5) Memory Efficiency :-
- Randomized skip lists may be more memory-efficient compared to perfect skip lists since they do not require maintaining a strick distribution of elements across levels.
- This can result in lower memory overhead especially for large skip lists.

Page No.
Date

Que- Explain self organizing linked lists with example, advantage, disadvantage and application

⇒

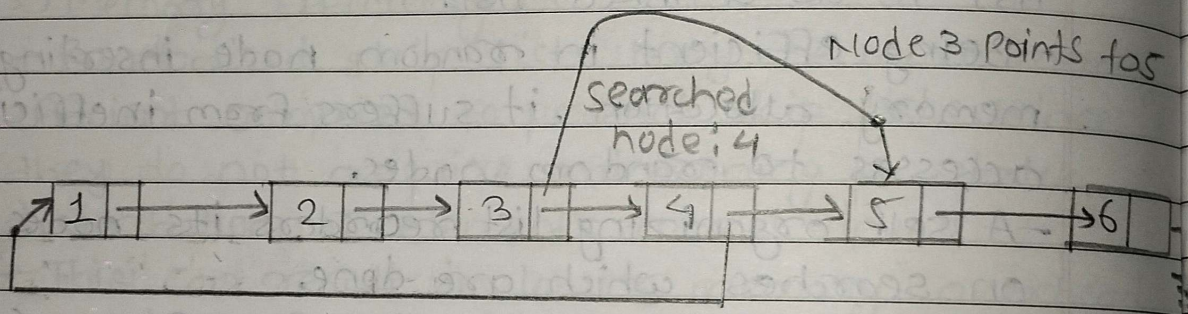
Self organizing linked list :-
- It is a list that reorders elements based on some self-organizing heuristics to improve average access time.
- The simplest implementation of self-organizing list is as a linked list.
- Being efficient in random node inserting and memory allocation, it suffers from inefficient accesses to random nodes.
- A self organizing list reorders its nodes based on searches which are done.
- The idea is to use locality of reference (In a typical database, 80% of the access are to 20% of the items).
- Following are different strategies used by self organizing lists :-

1) Move-to-Front method :-
- Any node searched is moved to the front.
This strategy is easy to implement, but it may over-reward infrequently accessed items as it always move the item to front.

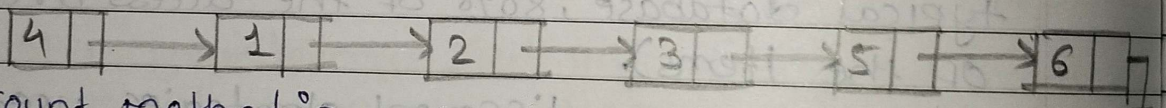
2) Count method :-
- Each node stores count of the number of times it was searched.
- Nodes are ordered by decreasing count.
- This strategy requires extra space for storing count.

3] Transpose method :-
- Any node searched is swapped with the preceding node.
- unlike move-to-front, this method does not adapt quickly to changing access patterns.

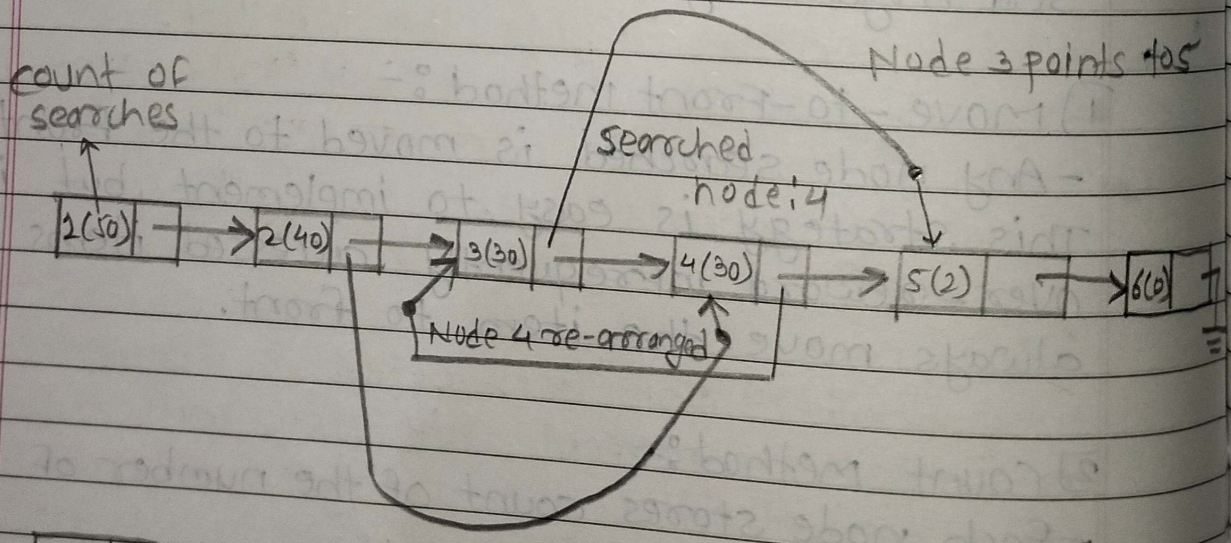
1] move-to-Front method :-

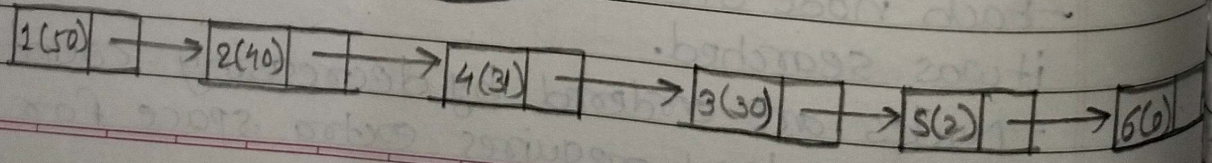


Node 4 is moved to front

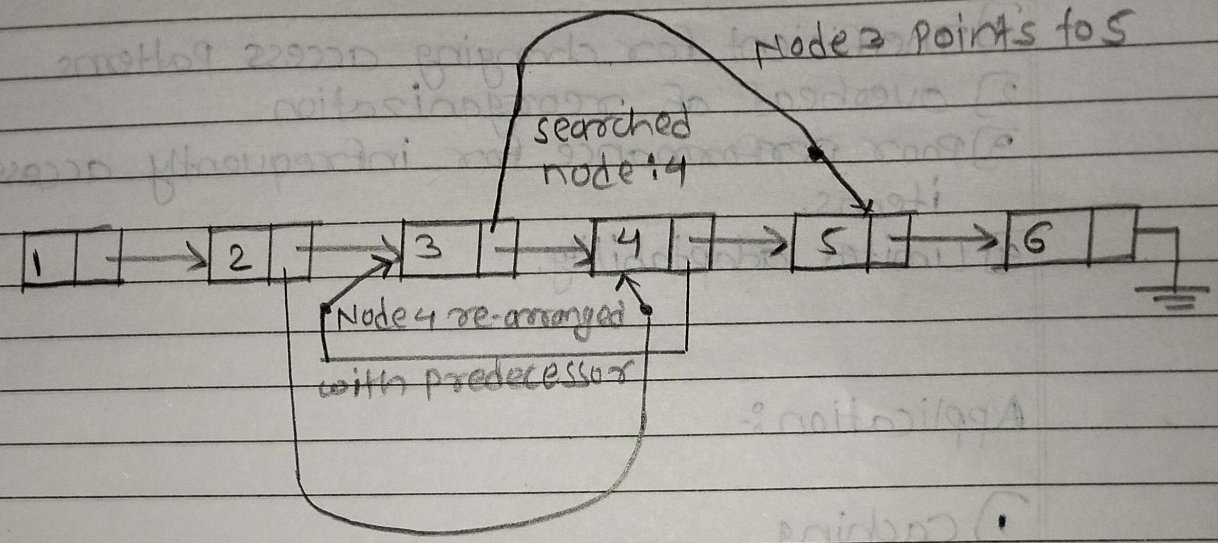


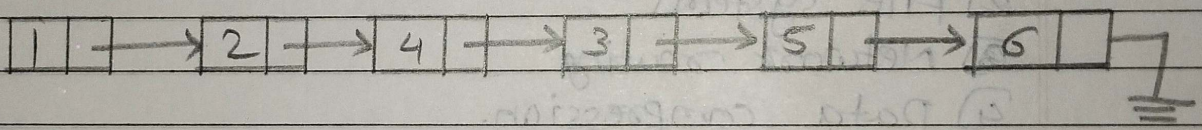
2) count method :-





4) Transpose method :-





- Advantages :-

1) Improved access time :- Frequently accessed items move towards the front of the list, reducing search time.
2) Simplified maintenance :- Reorganization during access removes the need for manual reordering, reducing complexity.
3) Adaptive Performance :- The list dynamically adapts to access patterns, optimizing performance over time.
4) Lower overhead :- Compared to other data structures like balanced trees, self-organizing linked lists often have lower memory and computational overhead.

Disadvantage:-

1) Inefficient for changing access patterns
2) overhead of reorganization
3) Poor performance for infrequently accessed items.
4) Limited adaptability.

Application:-

1) Caching
2) File systems
3) Network routing
4) Data compression.