**UNIT-2**

**LINKED LISTS**

**1. Singly linked lists: representation and operations**

A singly linked list is a data structure where each element (called a node) stores two things:

1. **Data:** The actual value or data that the node holds.
2. **Next:** A reference or a pointer to the next node in the list.

**The last node points to NULL (or None in Python) to indicate the end of the list.**

**Representation:**

**Each node in the list has the following structure:**

struct Node {

int data; // Stores data

Node\* next; // Pointer to the next node

};

* The head pointer points to the first node in the list.
* If the list is empty, the head is NULL.

**For example, a singly linked list representing the elements 1 -> 2 -> 3 -> NULL can be visualized like this:**

Head -> [1 | Next] -> [2 | Next] -> [3 | Next] -> NULL

**Where:**

* The first node contains the data 1 and a reference to the second node.
* The second node contains the data 2 and a reference to the third node.
* The third node contains the data 3 and a reference to NULL, signifying the end of the list.

**Operations on Singly Linked List:**

**1. Traversal:**

Traversal means visiting each node in the linked list. You start from the head node and follow the next pointers until you reach the end.

**void traverse(Node\* head) {**

**Node\* temp = head;**

**while (temp != NULL) {**

**cout << temp->data << " ";**

**temp = temp->next;**

**}**

**}**

**2. Insertion:**

There are three main types of insertion in a singly linked list:

* Insertion at the beginning (before the head): This involves creating a new node and pointing it to the current head, then updating the head to point to the new node.
* **void insertAtBeginning(Node\*\* head, int data) {**
* **Node\* newNode = new Node;**
* **newNode->data = data;**
* **newNode->next = \*head;**
* **\*head = newNode;**
* **}**
* Insertion at the end (after the last node): Traverse the list to the last node and update its next pointer to point to the new node.
* **void insertAtEnd(Node\* head, int data) {**
* **Node\* newNode = new Node;**
* **newNode->data = data;**
* **newNode->next = NULL;**
* **Node\* temp = head;**
* **while (temp->next != NULL) {**
* **temp = temp->next;**
* **}**
* **temp->next = newNode;**
* **}**
* Insertion at a specific position: Traverse to the given position and adjust the pointers accordingly.
* **void insertAtPosition(Node\* head, int data, int position) {**
* **Node\* newNode = new Node;**
* **newNode->data = data;**
* **Node\* temp = head;**
* **for (int i = 0; i < position - 1 && temp != NULL; i++) {**
* **temp = temp->next;**
* **}**
* **newNode->next = temp->next;**
* **temp->next = newNode;**
* **}**

**3. Deletion:**

**Deletion involves removing a node from the list, and there are three types of deletion:**

* Deletion from the beginning (removing the head node): Update the head to point to the second node.
* **void deleteAtBeginning(Node\*\* head) {**
* **if (\*head != NULL) {**
* **Node\* temp = \*head;**
* **\*head = (\*head)->next;**
* **delete temp;**
* **}**
* **}**
* Deletion from the end (removing the last node): Traverse to the second last node and set its next pointer to NULL.
* **void deleteAtEnd(Node\* head) {**
* **if (head == NULL) return;**
* **Node\* temp = head;**
* **Node\* prev = NULL;**
* **while (temp->next != NULL) {**
* **prev = temp;**
* **temp = temp->next;**
* **}**
* **prev->next = NULL;**
* **delete temp;**
* **}**
* Deletion at a specific position: Traverse to the desired position and adjust the pointers to remove the node.
* **void deleteAtPosition(Node\* head, int position) {**
* **if (head == NULL) return;**
* **Node\* temp = head;**
* **if (position == 0) {**
* **head = temp->next;**
* **delete temp;**
* **return;**
* **}**
* **for (int i = 0; temp != NULL && i < position - 1; i++) {**
* **temp = temp->next;**
* **}**
* **if (temp == NULL || temp->next == NULL) return;**
* **Node\* nextNode = temp->next->next;**
* **delete temp->next;**
* **temp->next = nextNode;**
* **}**

**4. Search:**

Searching for an element in a singly linked list involves traversing the list and checking each node's data.

**bool search(Node\* head, int data) {**

**Node\* temp = head;**

**while (temp != NULL) {**

**if (temp->data == data) {**

**return true;**

**}**

**temp = temp->next;**

**}**

**return false;**

**}**

**5. Reversing:**

Reversing a singly linked list means changing the direction of the next pointers so that the first node becomes the last.

**void reverse(Node\*\* head) {**

**Node \*prev = NULL, \*curr = \*head, \*next = NULL;**

**while (curr != NULL) {**

**next = curr->next;**

**curr->next = prev;**

**prev = curr;**

**curr = next;**

**}**

**\*head = prev;**

**}**

**6. Finding the Length:**

To find the length of a singly linked list, simply count the nodes by traversing the list.

**int length(Node\* head) {**

**int count = 0;**

**Node\* temp = head;**

**while (temp != NULL) {**

**count++;**

**temp = temp->next;**

**}**

**return count;**

**}**

**Time Complexity of Operations:**

* **Insertion:**
  + **At the beginning: O(1)O(1)**
  + **At the end: O(n)O(n) (because we need to traverse the entire list)**
  + **At a specific position: O(n)O(n)**
* **Deletion:**
  + **From the beginning: O(1)O(1)**
  + **From the end: O(n)O(n)**
  + **From a specific position: O(n)O(n)**
* **Search: O(n)O(n)**
* **Reversal: O(n)O(n)**
* **Traversal: O(n)O(n)**

