Experiment No: 14 and 16

Title: singly linked list and Doubly linked list

Aim

the goal of this project is single linked list and double linked list and also using stack using linked list

Learning objectives

1 Students should able to design and analyze simple linear and non linear data structures.

2It strengthen the ability to the students to identify and apply the suitable data structure for the given real world problem.

3 It enables them to gain knowledge in practical applications of data structures .

Theory:

single linked list Single Linked List:

A single linked list is a linear data structure consisting of nodes where each node has two components: data and a reference (or pointer) to the next node in the sequence. The last node's reference points to null, indicating the end of the list. The first node is known as the head.

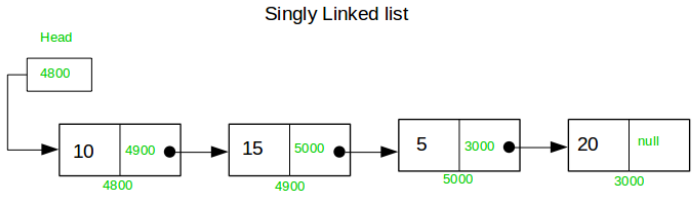

Figure 1: Enter Caption

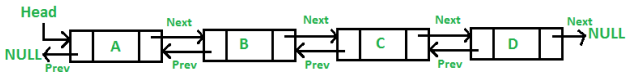

Figure 2: Enter Caption

double linked list

Double linked lists are used when you need to navigate the list in both directions or when you want to optimize certain operations, such as removal of nodes in the middle of the list. However, they come at the cost of increased complexity and memory usage compared to single linked lists.

Algorithm

Singly Linked List:

1Insertion at the beginning: To add a new node at the beginning of the list, you create a new node, set its next pointer to the current head, and update the head to point to the new node.

2Insertion at the end: To add a new node at the end, you traverse the list to find the last node, set its next pointer to the new node, and update the new node's next to point to null.

3Deletion of a node: To delete a specific node, you traverse the list to find the node and adjust the next pointer of the preceding node to skip the node you want to remove.

4Search: To search for a specific value in the list, you traverse the list, comparing the data in each node until you find a match.

5Traversal: To traverse the entire list, start from the head and follow the next pointers until you reach the end (i.e., next points to null).

Lab outcome:

LO1 : Write functions to implement linear and non-linear data structure operations **LO2** : Suggest appropriate linear / non-linear data structure operations for solving a given problem

Conclusion:

. Efficient for insertion and deletion at the beginning.

. Requires less memory due to only having a single link (next pointer).

. Traversal is efficient in one direction, but backward navigation is not possible without additional data structures.

. Doubly linked lists consist of nodes connected both forwards and backwards, with each node pointing to the next and previous nodes.

. Efficient for insertion and deletion at both the beginning and end.

. Offers bidirectional traversal, enabling forward and backward navigation.

Program :

Output:

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct node {
    int data;
    struct node *next;
} Node;

Node *head = NULL;

// Creates a new node and inserts it at the beginning of the list.
void push(int data) {
    Node *new_node = malloc(sizeof(Node));
    new_node->data = data;
    new_node->next = head;
    head = new_node;
}

// Prints the contents of the list.
void print_list() {
    Node *current = head;
    while (current != NULL) {
        printf("%d ", current->data);
        current = current->next;
    }
    printf("\n");
}
```

Figure 3: Enter Caption

```c
int main() {
    // Push some elements into the list.
    push(10);
    push(20);
    push(30);

    // Print the contents of the list.
    print_list();

    return 0;
}
```

Figure 4: Enter Caption

Output:

```
30 20 10
```

Figure 5: output of the programme