How to implement a queue using two stacks?

This method makes sure that newly entered element is always at the top of stack 1,
so that deQueue operation just pops from stack1. To put the element at top of stack1, stack2 is used.

enQueue(q, x)

1) While stack1 is not empty, push everything from satck1 to stack2.
2) Push x to stack1 (assuming size of stacks is unlimited).
3) Push everything back to stack1.

deQueue(q)

1) If stack1 is empty then error
2) Pop an item from stack1 and return it.

Double link list operation:

Inserting At Beginning of the list

We can use the following steps to insert a new node at beginning of the double linked list...

Step 1: Create a newNode with given value and newNode → previous as NULL.

Step 2: Check whether list is Empty (head == NULL)

Step 3: If it is Empty then, assign NULL to newNode → next and newNode to head.

Step 4: If it is not Empty then, assign head to newNode → next and newNode to head.

Inserting At End of the list

We can use the following steps to insert a new node at end of the double linked list...

Step 1: Create a newNode with given value and newNode → next as NULL.

Step 2: Check whether list is Empty (head == NULL)

Step 3: If it is Empty, then assign NULL to newNode → previous and newNode to head.

Step 4: If it is not Empty, then, define a node pointer temp and initialize with head.

Step 5: Keep moving the temp to its next node until it reaches to the last node in the list (until temp → next is equal to NULL).

Step 6: Assign newNode to temp → next and temp to newNode → previous.

Inserting At Specific location in the list (After a Node)

We can use the following steps to insert a new node after a node in the double linked list...

Step 1: Create a newNode with given value.

Step 2: Check whether list is Empty (head == NULL)

Step 3: If it is Empty then, assign NULL to newNode → previous & newNode → next and newNode to head.

Step 4: If it is not Empty then, define two node pointers temp1 & temp2 and initialize temp1 with head.

Step 5: Keep moving the temp1 to its next node until it reaches to the node after which we want to insert the newNode (until temp1 → data is equal to location, here location is the node value after which we want to insert the newNode).

Step 6: Every time check whether temp1 is reached to the last node. If it is reached to the last node then display 'Given node is not found in the list!!! Insertion not possible!!!' and terminate the function. Otherwise move the temp1 to next node.

Step 7: Assign temp1 → next to temp2, newNode to temp1 → next, temp1 to newNode → previous, temp2 to newNode → next and newNode to temp2 → previous.

Deletion

In a double linked list, the deletion operation can be performed in three ways as follows...

Deleting from Beginning of the list

Deleting from End of the list

Deleting a Specific Node

Deleting from Beginning of the list

We can use the following steps to delete a node from beginning of the double linked list...

Step 1: Check whether list is Empty (head == NULL)

Step 2: If it is Empty then, display 'List is Empty!!! Deletion is not possible' and terminate the function.

Step 3: If it is not Empty then, define a Node pointer 'temp' and initialize with head.

Step 4: Check whether list is having only one node (temp → previous is equal to temp → next)

Step 5: If it is TRUE, then set head to NULL and delete temp (Setting Empty list conditions)

Step 6: If it is FALSE, then assign temp → next to head, NULL to head → previous and delete temp.

Deleting from End of the list

We can use the following steps to delete a node from end of the double linked list...

Step 1: Check whether list is Empty (head == NULL)

Step 2: If it is Empty, then display 'List is Empty!!! Deletion is not possible' and terminate the function.

Step 3: If it is not Empty then, define a Node pointer 'temp' and initialize with head.

Step 4: Check whether list has only one Node (temp → previous and temp → next both are NULL)

Step 5: If it is TRUE, then assign NULL to head and delete temp. And terminate from the function. (Setting Empty list condition)

Step 6: If it is FALSE, then keep moving temp until it reaches to the last node in the list. (until temp → next is equal to NULL)

Step 7: Assign NULL to temp → previous → next and delete temp.

Deleting a Specific Node from the list

We can use the following steps to delete a specific node from the double linked list...

Step 1: Check whether list is Empty (head == NULL)

Step 2: If it is Empty then, display 'List is Empty!!! Deletion is not possible' and terminate the function.

Step 3: If it is not Empty, then define a Node pointer 'temp' and initialize with head.

Step 4: Keep moving the temp until it reaches to the exact node to be deleted or to the last node.

Step 5: If it is reached to the last node, then display 'Given node not found in the list! Deletion not possible!!!' and terminate the fuction.

Step 6: If it is reached to the exact node which we want to delete, then check whether list is having only one node or not

Step 7: If list has only one node and that is the node which is to be deleted then set head to NULL and delete temp (free(temp)).

Step 8: If list contains multiple nodes, then check whether temp is the first node in the list (temp == head).

Step 9: If temp is the first node, then move the head to the next node (head = head → next), set head of previous to NULL (head → previous = NULL) and delete temp.

Step 10: If temp is not the first node, then check whether it is the last node in the list (temp → next == NULL).

Step 11: If temp is the last node then set temp of previous of next to NULL (temp → previous → next = NULL) and delete temp (free(temp)).

Step 12: If temp is not the first node and not the last node, then set temp of previous of next to temp of next (temp → previous → next = temp → next), temp of next of previous to temp of previous (temp → next → previous = temp → previous) and delete temp (free(temp)).

Displaying a Double Linked List

We can use the following steps to display the elements of a double linked list...

Step 1: Check whether list is Empty (head == NULL)

Step 2: If it is Empty, then display 'List is Empty!!!' and terminate the function.

Step 3: If it is not Empty, then define a Node pointer 'temp' and initialize with head.

Step 4: Display 'NULL <--- '.

Step 5: Keep displaying temp → data with an arrow (<===>) until temp reaches to the last node

Step 6: Finally, display temp → data with arrow pointing to NULL (temp → data ---> NULL).

Algorithm to merge sorted arrays

Merge algorithm

Assume, that both arrays are sorted in ascending order and we want resulting array to maintain the same order. Algorithm to merge two arrays A[0..m-1] and B[0..n-1] into an array C[0..m+n-1] is as following:

1. Introduce read-indices **i**, **j** to traverse arrays A and B, accordingly. Introduce write-index **k** to store position of the first free cell in the resulting array. By default **i** = **j** = **k** = 0.
2. At each step: if both indices are in range (**i** < m and **j** < n), choose minimum of (A[**i**], B[**j**]) and write it to C[**k**]. Otherwise go to step 4.
3. Increase **k** and index of the array, algorithm located minimal value at, by one. Repeat step 2.
4. Copy the rest values from the array, which index is still in range, to the resulting array.