Linked Implementation of Queue And Stack

Bindu Singh
[1, 2] Dept of Computer Engineering Department
[1, 2] Vadodara Institute of Engineering

Abstract- *a linked list is a linear collection of data elements, in which linear order is not given by their physical placement in memory. Each pointing to the next node by means of a pointer. It is a data structure consisting of a group of nodes which together represent a sequence*

Keywords- Stack,Queue,Linked List

I. INTRODUCTION

Linked list is used to implement other data structures also. A stack is created using an array. This technique of creating a stack is easy, but the drawback is that the array must be declared to have some fixed size. In case the stack is a very small one or its maximum size is known in advance, then the array implementation of the stack gives an efficient implementation. But if the array size cannot be determined in advance, then the other alternative, i.e., linked representation, is used. The storage requirement of linked representation of the stack with n elements is O(n), and the typical time requirement for the operations is O(1).In a linked stack, every node has two parts—one that stores data and another that stores the address of the next node. The START pointer of the linked list is used as TOP. All insertions and deletions are done at the node pointed by TOP. If TOP = NULL, then it indicates that the stack is empty.

II. OPERATIONS ON A LINKED STACK

A linked stack supports all the three stack operations that is, push, pop.

PUSH OPERATION

The push operation is used to insert an element into the stack. The new element is added at the topmost position of the stack.

To insert an element we first check if TOP=NULL. If this is the case, then we allocate memory for a new node, store the value in its DATA part and NULL in its NEXT part. The new node will then be called TOP. However, if TOP!=NULL, then we insert the new node at the beginning of the linked stack and name this new node as TOP here is an algorithm to push an element into a linked stack.

Step1:Allocate memory for new node and name it as NEW_NODE
Step 2:new_node->data=val
Step 3:if top==NULL
Set new_node->next=NULL
Set top=new_node
Else
Set new_node->next=top
Set top=new_node
[end of if]
Step 4:End

In Step 1, memory is allocated for the new node. In Step 2, the DATA part of the new node is initialized with the value to be stored in the node. In Step 3, we check if the new node is the first node of the linked list. is done by checking if TOP = NULL. In case the IF statement evaluates to true, then NULL is stored in the NEXT part of the node and the new node is called TOP. However, if the new node is not the first node in the list, then it is

1) added before the first node of the list (that is, the TOP node) and termed as TOP.
2) POP OPERATION
3) The pop operation is used to delete the topmost element from a
4) stack. However, before deleting the value, we must first check
5) if TOP=NULL, because if this is the case, then it means that the
6) stack is empty and no more deletions can be done. If an attempt is made to delete a value from a stack that is already empty, an UNDERFLOW message is printed. In case TOP!=NULL, then we will delete the node pointed by TOP, and make TOP point to the second element of the linked stack.
7) Step1:if top=NULL
8) Print"underflow" goto step5
9) Step 2:set ptr=top
10) Step 3:set top=top->next
11) Step 4: free ptr
12) Step 5:end

13) In Step 1, we first check for the UNDERFLOW condition.

14) In Step 2, we use a pointer PTR that points to TOP. In Step 3, TOP is made to point to the next node in sequence. In Step 4 In Step 4, the memory occupied by PTR is given back to the free pool.

III. LINKED REPRESENTATION OF QUEUES

We have seen how a queue is created using an array. Although this technique of creating a queue is easy, its drawback is that the array must be declared to have some fixed size. If we allocate space for 50 elements in the queue and it hardly uses 20–25 locations, then half of the space will be wasted. And in case we allocate less memory locations for a queue that might end up growing large and large, then a lot of re-allocations will have to be done, thereby creating a lot of overhead and consuming a lot of time. In case the queue is a very small one or its maximum size is known in advance, then the array implementation of the queue gives an efficient implementation. But if the array size cannot be determined in advance, the other alternative, i.e., the linked representation is used. The storage requirement of linked representation of a queue with n elements is O(n) and the typical time requirement for operations is O(1).In a linked queue, every element has two parts, one that stores the data and another that stores the address of the next element. The START pointer of the linked list is used as FRONT. Here, we will also use another pointer called REAR, which will store the address of the last element in the queue. All insertions will be done at the rear end and all the deletions will be done at the front end. If FRONT = REAR = NULL, then it indicates that the queue is empty.

OPERATIONS ON LINKED QUEUES

A queue has two basic operations: insert and delete. The insert operation adds an element to the end of the queue, and the delete operation removes an element from the front or the start of the queue. Apart from this, there is another operation peek which returns the value of the first element of the queue.

INSERT OPERATION

The insert operation is used to insert an element into a queue. The new element is added as the last element of the queue. To insert an element , we first check if FRONT=NULL. If the condition holds, then the queue is empty. So, we allocate memory for a new node, store the value in its data part and NULL in its next part. The new node will then be called both FRONT and rear. However, if FRONT!= NULL, then we will insert the new node at the rear end of the linked queue and name this new node as rear.

Step 1:Allocate memeory for new node and name it as PTR
Step 2:setr ptr->data=val
Sterp 3:if front=NULL
Set front=rear=ptr
Set front->next=rear->next=NULL
Else
Set rear->n0065xt=ptr
Set rear=ptr
Set rear->next=NULL
[end of if]
Step 4:end

In Step 1, the memory is allocated for the new node. In Step2, the DATA part of the new node is initialized with the value to be stored in the node. In Step 3, we check if the new node is the first node of the linked queue. This is done by checking if RONT = NULL. If this is the case, then the new node is tagged as FRONT as well as REAR. Also NULL is stored in the NEXT part of the node (which is also the FRONT and the REAR node). However, if the new node is not the first node in the list, thenit is added at the REAR end of the linked queue (or the last node of the queue.

DELETE OPERATION

The delete operation is used to delete the element that is first inserted in a queue, i.e., the element whose address is stored in FRONT. However, before deleting the value, we must first check if FRONT=NULL because if this is the case, then the queue is empty and no more deletions can be done. If an attempt is made to delete a value from a queue that is already empty, an underflow message is printed. To delete an element, we first check if FRONT=NULL. If the condition is false, then we delete the first node pointed by FRONT. The FRONT will now point to the second element of the linked queue. In Step 1, we first check for the underflow condition. If the condition is true, then an appropriate message is displayed, otherwise in Step 2, we use a pointer PTR that points to FRONT. In Step 3, FRONT is made to point to the next node in sequence. In Step 4In Step 4, the memory occupied by PTR is given back to the free pool.

Step 1:if front=NULL
Write “overflow” goto step 5
[end f if]
Step 2:set ptr=front
Step 3:front=front->next
Step 4:free ptr
Step 5:end

IV. CONCLUSION

This paper gives brief introduction about linked list can be used to implement other data structures like stack and queue and how to perform various operations in stack and queue.

REFERENCES

[1] G. O. Young, ―Synthetic structure of industrial plastics (Book style with paper title and editor),‖ in *Plastics*, 2nd ed. vol. 3, J. Peters, Ed. New York: McGraw-Hill, 1964, pp. 15–64.
[2] W.-K. Chen, *Linear Networks and Systems* (Book style). Belmont, CA:
Wadsworth, 1993, pp. 123–135.
[3] H. Poor, *An Introduction to Signal Detection and Estimation*. New York: Springer-Verlag, 1985, ch. 4.
[4] B. Smith, ―An approach to graphs of linear forms (Unpublished work style),‖ unpublished.
[5] E. H. Miller, ―A note on reflector arrays (Periodical style—Accepted for publication),‖ *IEEE Trans. Antennas Propagat.*, to be published.
[6] J. Wang, ―Fundamentals of erbium-doped fiber amplifiers arrays (Periodical style—Submitted for publication),‖ *IEEE J. Quantum Electron.*, submitted for publication.