OMKAR BHARITKAR

112016020 ECE

Omkar R. Bhavitkar
112016020 ECE.

DSA LAB: 5 Theory.

Q1. Define linked list. Comment on: The features of linked list over array.

--> Linked list is the linear data structure in which element are not stored at continguous memory location.
The linked list node contain two part head & tail. Head stores information while tail stores address of next node.

Features over array :-

i) It can be accessed in sequential manner.
ii) In linked ~~list~~ insertion & deletion can be done at any point in list at constant time.
iii) In linked list we can add any number of elements in list.
iv) In linked list we can extend list as we needed while in array extension of same declear array is not possible.

Q2. How many pointers are required for implementing a singly linked list?

→ Three types of pointers are required to implement singly linked list :-

i) A head pointer which is required for pointing to start of record.

ii) Info pointer which require to point last of node.

iii) Link pointer which require to point the ~~so~~ next node.

Q3. State and explain different type of linked list. How a linked list node can be represented?

→ following are the different types of linked list :-

i) Simply linked list :

It is simplest type of linked list in which every node contain some data and pointer to next data, it goes only in one direction.

2) Doubly linked list:-
In this type of linked list item can be traversed in both direction.

3) Circular linked list:-
In circular linked list last node caintains a pointer to first node of list.

4) Circular doubly linked list:-
In this item can be traversed in both direction as well as last node caintain address of 1st node.

Q4. Compare Singly linked list and doubly linked list.

→ ① Singly linked list can be traversed only in one direction while Doubly linked list can be traversed in both direction.

② Singly linked list caintain pointer to next node only while doubly caintain pointer to next as well as previous node.

③ Singly consist of two parts i.e. data & pointer to next node while doubly consist of three parts i) data, ii) pointer to next node iii) pointer to next node.

④ Singly uses less memory as it store pointer of only one node where as Doubly use more memory as it store pointer of both previous and next node..

Q5. Give application of linked list.

→ Application of linked list:

i) In implimentation of stack & queues.
ii) In implimentation of graphs.
iii) For representing sparse matrices.
iv) Manipulation of polynomial by storing coefficient.
v) For dynamic memory allocation.
vi) For maintaining directory of names.

Q6. Write down the detailed algorithms for the given problem statement.

→ @ Algorithm for question 1.

Insertion of node at the end of list.

step - 1 : START.

step - 2 : Creat a node pointer dynamically
node * ptr = new node ().

step - 3 : Assign the data to the data part
ptr → data = data;

step - 4 : Assign NULL pointer to the pointer of the node.
ptr → next = NULL

step - 5 : If head pointer is equal to NULL assign head painter to the created node painter

Step - 6 : Else Iterate through the nodes till one of the nodes do not point to NULL
curr = head.
while (curr → next != NULL)
curr = curr → next.

step 7 : Assign the pointer to the related node to the node pointer which points to NULL

Step 8 : END.

Deletation of Node from the starting pai

Step 1: Start.

Step 2: Assign two node pointer one to the head of the linked list and another to NULL
Node* Sub = Head
Node* last = ptr.

Step 3: Check if Head is not NULL and the key data is passed is equal to the data which is to be erased.
if (sub1 = NULL + Sub → data == data)
{ Head = sub → next;
delete sub;
}

Step 4: Else iterate through the list till the data is not found or the pointer is equal to NULL.

Step 5: If data not found print "Data not found"

Step 6: Else point the "last pointer to "sub pointer".
i.e. point the previous node pointer to the next node's pointer.
last → next = sub → next;
delete the current node.
delete sub;

Step 7: END.

Printing the data in the nodes in reverse order using recursion.

Step 1 : START.
Step 2 : check the current node pointer passed is NULL
Step 3 : If yes, then return.
Step 4 : else call the functions by sending the address of the pointer of next node.
Step 5 : Print the data
Step 6 : END

Pseudo Code :

```
void recur (Node* ptr)
{
    IF (ptr == NULL) return
    else recur (ptr → next)
        cout << ptr → data ;
}
```

✱ Algorithm of 2nd question.

Insertion in doubly linked list.

Step 1 :- Start.
Step 2 :- Assign a new node pointer.
Step 3 :- Assign data passed to the data variable in pointer.
Step 4 :- Assign the forward pointer to the Head.
Step 5 :- Assign the backward pointer to NULL
Step 6 :- Check if head pointer is equal to NULL.
Step 7 :- If no, point the backward pointer head to the new node pointer created.
Step 8 :- Point the head pointer to the new node created.
Step 9 :- END.

Pseudo Code :-

```
Node *ptr = new Node ();
ptr → data = data;
ptr → next = Head;
ptr → prev = NULL;
If (Head != NULL) (head → prev) = ptr
head = ptr;
```

Sorting of the doubly linked list

Step 1 - Start.
Step 2 - Assign two node pointers to NULL and declare a int variable
Step 3 - check if the list is empty.
Step 4 - make a do-while loop.
Step 5 - Assign int variable value zero and assign one pointer to head
flag = 0;
ptr = head
Step 6 :- Iterate ptr till it is not equal to
Step 7 :- While in loop check the condition if the ptr → data is greater than ptr → next → data
Step 8 :- If the condition is true,
swap (ptr → data ; ptr → next → data
flag = 1.
Step 9 :- Break the loop when the value of flag is zero.
Step 10 :- END.

Pseudo code:

```
node * ptr = NULL
node * ptr1 = NULL
int flag;
do
{
    flag = 0;
    ptr = head;
```

```
while (ptr -> next != ptr1)
{ if (ptr -> data > ptr -> next -> data)
  { swap (ptr -> data, ptr -> next -> data
      flag = 1;
  }
ptr = ptr -> next;
}
  ptr1 = ptr -> next;
} while (flag == 1);
```

Addition of two Binary Number

Pseudo code:

```
string add (string &s1, string s2)
{ int ln = s1.size();
 string t = "";  int carry = 0;
 for(int i = ln-1; i >= 0; i--)
  {
    if (si[i] == '1' && s2[i] == '1'
                    carry == 0)
  { t.append ('0');
      carry = 1;
  }
  else if (s1[i] == '1' && s2[i] == '1'
                    && carry == 1)
  {  t.appd ('1');
      carry = 1;
  }
    else if (si[i] == '0' && s2[i] == '1'
          (s1[i] == '1' && s2[i] == '0')
                    && carry == 0)
```

```
{   t.append('1');
    carry = 0;
}

else if((s1[i] == '1' && s2[i] == '0') ||
    (s1[i] == '0' && s2[i] == '1') &&
                        carry == 1)
{
    t.append('0');
    carry = 1;
}
else if (s1[i] == '0' && s2[i] == '0'
    carry == 0)
{
    t.append('0');
    carry = 0;
}
else if (s1[i] == '0' && s2[i] == '0'
            && carry == 1)
{   t.append('1')
    carry = 0;
}
}
  if(carry == 1) t.append('1');
  reverse(t.begin(), t.end());
  return t;
};
```