Program 1

AIM: Program to Merge two sorted arrays

Algorithm:-

step 1 : START

step 2 : Initialize arr1[] arr2[], res[], m, n, i, k

step 3 : check while i<m && j<n
and arr1 > arr2[j] then set-
result, res[k++] = arr2[j++]
otherwise
arr1[i++].

step 4 : Repeat while i<m then set
res[k++] = arr1[j++]
set i = i+1
else
set k = k+1

step 5 : Repeat while j<n then set
res[k++] = arr2[j++]

step 5 : print the resultant array

step 6 : STOP.

Program NO : 2

Aim :- Singly linked stack - push, pop, linear search.

Algorithm

step 1 : START

step 2 : create a newnode with given data when user select a push operation.

step 3 : check whether the stack is empty or not

If top == NULL

Then set top = newnode

else:

newnode → next = top

top = newnode.

(end if)

step 4 : User select pop operation then check whether the stack is empty

else

set temp = top

display . temp - data

set top = temp → next

step 5 : check whether the item is present or not

while ptr != NULL
then ptr → data == item
set flag = 1
else
flag == 0
(item not found)

step 6 : If the stack is ~~not~~ empty
then
temp → next != NULL
temp = temp → next

step 7 : Display the list

step 8 : STOP.

program No: 3

AIM : Circular queue - ADD, delete, search operation using array implementation.

Step 1 : START

Step 2 : check whether the queue is empty

```
If (front == -1 && rear == -1)
    set
        queue[rear] = element
    else
        rear = (rear + 1) % max;
        queue[rear] = element
```

Step 3 : If the queue is not empty then can be delete element

```
front = (front + 1) % max
```

step 4 : Display the elements

```
If (front == -1 && rear == -1)
set queue is empty
```

else

repeat while(i<=rear)

set queue[i]

i=(i+1) % max

Step 5 : Choose the choice and get the result

Step 6 : STOP.

Program No : 4

AIM : Doubly Linked List - Insertion, Deletion Search

Algorithm

Step 1 : START

Step 2 : Check whether the list is overflow then print overflow

If (Ptr == NULL)
(Overflow condition)
else
{ Read item value

Step 3 : check head == NULL

SET Ptr → next = NULL

SET Ptr → prev = NULL

Ptr → data = item

head = Ptr.

Step 4 : else

SET Ptr → data = item

Ptr → prev = NULL

Ptr → next = head

head → prev = Ptr

head = Ptr

then node inserted.

Step 5 : If the node inserted at the last node then
check the overflow condition
otherwise

SET
temp = head
Repeat while temp->next != NULL
then SET temp = temp->next
ptr->prev = temp
ptr->next = NULL

step 6 : check the overflow condition
other wise inserted at the specified location.

Repeat for i=0; i<loc, and i++
then SET
temp = temp->next

Step 7 : Display inserted queue.

Step 8 : check head == NULL
Then print underflow

step 9 : check head->next == NULL

Then SET Head = NULL

Deleted node.

step 10 : check head -> next == NULL

Then

head = NULL

otherwise

SET

Ptr = head and check the condition:

If(Ptr->next != NULL)

Ptr => Ptr->next.

Ptr->prev->next = NULL

Deleted node.

step 11 : check if the queue is empty

otherwise

Repeat while(Ptr != NULL)

check Ptr->data == item

SET flag = 0

step 12 : Display the final queue

step 13 : STOP.

program No : 5

AIM : set Data structure and set operations (union, intersection and difference) using bit string.

Algorithm

Step 1 : START

Step 2 : If user select the union operation then declare two array set1[i] and set2 [i]

Step 3 : If $n_1 == n_2$ then
SET i = 0

Step 4 : Repeat for $i < n_2$ then
SET set3 [i] = set1[i] || set2[i]

Step 5 : Repeat for i < 2 then
Display set3[i]

else
Print size are not equal

Step 6 : If user select intersection operation then

Step 7 : declare two array set1[i] and set2[i] with sibe n_1, n_2.

step 8 : If $n_1 == n_2$ then

step 9 : Repeat for $i < n_2$ then

Step 10 : SET set3[i] = set1[i] && set2[i]

Step 11 : Repeat for $i < n_2$ then print set3[i]

Step 12 : If user select the Substraction then

step 13 : declare two array set1[i] and set2[i] with n_1, n_2 size

Step 14 : Repeat for $i < n_2$ then
SET set3[i] = set1[i] && !set2[i]

Step 15 : Repeat for, $i < n_2$
Display set3[i]

Step 16 : set i = i+1

else:
print size are not equal

Step 17 : Stop.

program No: 6

AIM: Binary search trees - Insertion, deletion search.

Algorithm:

step1 : START

step 2 : If user select the insertion operation then
Create a new BST node and assign value to it.

Step 3 : If root == NULL then
SET temp -> data = data
SET temp -> left -> right = NULL

step 4 : If data < node -> data.
SET
node -> left and assign the return value node -> left.

step 5 : data > node -> data
SET node -> right

step 6 : If the user seled the search element operation then:

step 7 : If node == NULL
Then element not found.

step 8 : If data < node → data then
node → left and assign the deleted
value in node → left.

step 9 : If data > node → data
then SET
node → right

step 10 : If the user select the deletion

step 11 : check node == NULL
Then element not found

step 12 : check data < node → data
then SET
node → left

step 13 : check data > node → data
then SET node → right
check node → right && node → left
Then
// replace with minimum element
in the right subtree.

step 14 : call function del with value
node -> right
temp -> data
otherwise
SET temp = node

step 15 : If node -> right == NULL then
SET node = node -> left
free (temp)

step 16 : If node != NULL then
Inorder (node -> left)
Display node -> data

step 17 : SET node -> right

step 18 : STOP.

Program No : 7

AIM :- Disjoint sets and the associated operations

Algorithm :

step 1 : START

step 2 : Declare the variables of the set

step 3 : store the user's data and call function makeset() then SET i=0

step 4 : Repeat for i<dis.n then
SET dis.parent[i] = i
SET dis.rank[i] = 0
SET i = i+1

step 5 : User select union operation then then read the elements to be perform and store to x set and y

step 6 : Perform find operation with x and store result into x set and y set perform step

step 7 : If xset == yset then
End of [loop IF]

Step 8 : If rank[xset] < dis rank [yset]
then
SET dis.parent [xset] = y set
SET dis. rank [x set] = -1
else dis.rank [xset] > dis. rank [yset]
then.
SET dis. parent [yset] = x set
SET dis. rank [y set] = -1
otherwise
SET dis parent [yset] = x set
SET dis. rank [xset] = dis. rank [xset]
SET dis. rank [yset] = -1

Step 9 : If user choose find operation
then :
If find x == find y then
Display the set

Step 10 : If user select the display
Operation then
Set i = 0

Step 11 : Repeat for i < dis. n then
Print dis. parent [i]
SET i = i+1

Step 12 : Repeat for i < dis. n then
print dis. rank [i]
SET i = i+1

Step 13 : If dis.parent[x] != x

then

SET dis.parent[x] = find (dis.parent[x]

Step 14 : Display elements

Step 15 : Exit.