Program No:1

Aim: Merge two sorted arrays and store in a third array.

Algorithm

1. Start
2. Read the size of two arrays, m and n
3. Read the element in both arrays arr1 and arr2.
4. Declare elements m,n,i,j,k, and array arr3
5. Initialize i=0, j=0, k=0
6. Repeat step 7 while (i<m and j<n)
7. if arr1[i]>arr2[j])

   then set arr3[k] = arr2[j]

   set j = j+1

   else

   set arr3[k] = arr1[i]

   set i = i+1

   (End of the loop)

8. Repeat step 9 while i<m

9. set arr3[k] = arr1[i], i++ = i+1
   (End of loop)

10. Repeat step 11 while j<n

11. set arr3[k] = arr2[j], j = j+1
   (End of loop)

12. Print ~~arr1[i]~~

12. Set i=0, repeat ~~st~~ step 13 while i<m,

13. print arr1[i]
   i++ = i+1

14 set i=0, repeat step 15 while i<n

15. print arr2[i]
   i++ = i+1

16. set i=0, repeat step 17 while i<m+n

17. print arr3[i]

18. stop

Program No: 2

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

Algorithm:-

1. Start
2. If user select push operation then
3. Create a new node with the given data.
4. If top == NULL then:
   (Check ~~wether~~ whether the stack is empty)
5. SET top = newnode
   SET newnode → next = NULL

ELSE:

SET newnode → next = top

top = new node

[End of if structure]

6. If user select pop operation then

7. If top == NULL then:

(Check whether stack is empty)

display "stack is empty".

ELSE:

SET temp = top

(Create a temporary node and set it to top).

display temp → data

8. SET top = temp → next

(make Top point to the next node).

9. free (temp)
(Delete the temporary node)

10. If user select search operation then.

11. Declare a pointer variable temp and the variable key that holds the value to be searched.

12. SET temp = Top
SET flag = 0

13. Repeat while temp != NULL
If temp -> data = key then:
display "element found"
SET flag = 1
[End of if structure]
Goto step 14

ELSE:

SET temp = temp → next

[End of while loop ~~if structure~~]

14. If flag == 0 then:

display "element not found".

[End of if structure]

15. If user select display operation then.

16. Declare a pointer node ptr.

SET node ptr = top.

17. If node ptr == NULL then:

display "stack is empty"

[End of if structure]

18. while node ptr != NULL then:

print node ptr → data

SET node ptr = node ptr -> next

if node ptr ! = NULL then:

print " --> "

[End of if structure]

[End of while]

19. Exit.

Program No: 3.

Aim: Program to perform operation in circular Queue.

Algorithm:

1. start

2. If user select the insertion operation then:

3. Declare a variable item with given value.

4. If front ==0 && rear == size -1 || front == rear+1 then:

Display "Queue overflow"

[End of if structure]

5. If front == -1 then:

SET front = 0
SET rear = 0
[End of if structure]

6. If rear == size - 1
SET rear = 0
else:
SET rear = rear+1
[End of if structure]

7. cq[SET cq[rear] = item.

8. If user select deletion operation then:

9. If front == -1 then:
Display "Queue overflow"
[End of if structure]

10. If front == rear then:
SET front = -1

SET rear = -1

[End of if structure]

11. If front == size-1 then:

SET front = 0

Else

SET front = front + 1

[End of if structure]

12. If user select the display operation

then

13. SET front_pos = front

SET rear_pos = rear.

14. If front == -1 then:

Display "Que is empty"

[End of if structure"]

15. If front_pos <= rear_pos then:
Repeat while front_pos <= rear_pos then:

print cq[front_pos]
SET front_pos = front_pos + 1
[End of while]
Else:
Repeat while front_pos <= size-1
print cq[front_pos]
SET front_pos = front_pos + 1
[End of while]

16. SET front_pos = 0
17. Repeat while front_pos <= rear_pos then:

cq[front_pos]
Set front_pos = front_pos + 1

[End of while]

[End of if]

18. If user select search operation then:

19. Declare a variable ser with value to be searched.

20. Declare a temporary variable temp the SET temp = ser.

21. SET i = front.

22. Repeat for i <= rear then:

If == cq[i] then:

print i+1

SET j = j+1

[End of if]

If j==0 then

Display "item not found"

[End of if structure]

SET $i = i + 1$

[End of for loop]

23. Exit.

Program No:4

Aim: Program to perform operation in doubly linked list.

Algorithm:

1 Start

2. If user select the union operation then:

3. Declare two array set1[i] and set2[i], Declare two variable n1,n2, for holding the size of two arrays.

4. Read elements into the arrays set1[i] and set2[i].

5. If n1==n2 then:

6 SET i=0

7 Repeat for i<n2 then

set set3[i] = set1[i]||set2[i].

8. SET i = i + 1

[End of for loop].

9. SET i = 0

10. Repeat for i < 2 then

print set3[i]

11. SET i = i + 1

[End of for loop]

[End of if]

Else:

print "size are not equal"

Exit.

12. If user select insertion operation then.

13. Declare two array set1[i] and set2[i] with size n1, n2 respectively

and Read elements to the arrays.

14. If n1 == n2. then

15. SET i = 0

16. Repeat for i<n2 then:

17. SET set3[i] = set1[i] && set2[i]

18. SET i = i+1

[End of for loop]

19. SET i = 0

20. Repeat for i<n2 then:

print set3[i].

21 set i = i+1

[End of for loop]

[End of if].

Else:

print "size are not equal" then.

Exit.

22. If user select the substraction then

23. Declare two array set1[i] and set2[i] with n1, n2 size respectively and input the elements to the array.

24. If n1 == n2 then

25. set i=0

26. Repeat for i<n2 then:
    SET set3[i] = set1[i] && ! set2[i]

27. i = i+1
    [End of for loop]

28. SET i=0

29. Repeat for i<n2 then:
    print set3[i]

30 Set i = i+1

[End of for loop]

[End of if]

Else:

"print size are not equal"

31 Exit.

Program No: 8

Aim: Program to perform set operation.

Algorithm:

1. Start
2. If user select the insertion operation then
3. Create a new BST node and assign values to it.
4. Create tree (node, data) // call the create tree function with the root value and the data entered by user.
5. If root == NULL then:
6. Declare a temporary variable temp
   SET temp -> data = data
   SET temp -> left -> right = NULL.

return the new node temp to the calling function.

[End of if]

7. If data < (node -> data)
8. Call the create node function with node -> left and assign the return value in node -> left
node -> left = create tree (node -> left, data)

[End of if structure]

9. If data > node -> data
10. Call the create tree function with node -> right and assign the return value in node -> right
node -> right = create tree (node -> right - data).

(End of if structure)

11 return the original root pointer 'no
to the calling function.

12. If the user select the search elem
operation then:

13. search (node, data)//call the search
function with root value and the
element to be searched.

14. If node = = NULL
print "element not found"
[End of if]

15. If data < node = = NULL
print "element not found"
~~[End of if]~~

16. If data < node -> data then:
call the search function with
node -> left = search and assign

node->left = search (node->left, data

[End of if structure]

16. If data>node->data then

call search function with node->right and assign the return value to node->right.

node->right = search(node->right, data

[End of if structure]

Else:

print "Element found is" node->data.

17. Return the original root pointer 'node' to the calling function.

18. If the user select the deletion operation then:

19. del(node, data) // call the del function with root value and the element to be deleted.

20. declare a temporary variable temp

21. If node = = NULL then:

print "Element found".

[end of if]

22. If data < node -> data then:

call the del function with node -> left and assign the return value to nod -> left

node -> left = del(node -> left, data)

[end of if].

23. If data > node -> data then

call the del function with node -> right and assign the return value to n node -> right.

[end of if]

24. Else if:

//delete this node and replace with either minimum element in the right sub tree or maximum element in the left subtree.

25. If node -> right && node -> left

//replace with minimum element in the right sub tree

26. call findmin function with node -> right then return value assign in temp, Go to step 30.

SET temp = find min(node -> right)

SET node -> data = temp -> data

//replaced it with some other node

27. call function del with value node -> right, temp -> data and return value assign n node -> right.

Else:

28: SET temp = node

// If there is only one or zero children then we can directly remove it from the tree and connect its parent to its child.

29. If node -> left == NULL then:

SET node = node -> right

Else:

30: If node -> right == NULL then.

SET node = node -> left

31: free(temp)

(End of if)

[End of if]
[End of if]

32 find min(node)

33 If node == NULL then
    return NULL.
    Go to step 24
  [End of if]

34 If node -> left then
  call the function find min with value (node -> left) then return the value to calling function return find min(node -> left)
  Else
    return node
  Goto step 24.
  [End of if]

35. If the user select the display option then:

36. Inorder (node)
call the inorder function with root value.

37. If nod != NULL then
Inorder (node -> left)
call the function inorder with value node -> left.

38. Print node -> data
Inorder(node -> right)
call the function inorder with value node -> right
[End of if]

39. Exit.

Program No:6

Aim: Program to perform binary tree operation

1. Start
2. Declare a structure and structure pointer for insertion, deletion and search operation and also declare a function for inorder traversal.
3. Declare a pointer as root and also the required variables.
4. Read the choice from the user to perform insertion, deletion, searching and inorder traversal
5. If the user choice to perform

insertion operation then read the value which is to be inserted to the tree from the user.

5.+ 5.1 Pass the value to the insert pointer and also the root pointer.

5.2. Check if root then allocate memory for the root.

5.3. Set the value to the info part of the root and then set left and right part of the root to null and return root.

5.4. Check if root → info > x then call the insert pointer to insert to left of the root.

5.5 Check if root → info < x then

call the insert pointer to insert to the right of the root.

5.6. Return the root.

6. If the user choose to perform deletion operation then read the element to be deleted from the tree. Pass the root pointer and the item to tree deletion pointer.

6.1. Check if root ptr then print node not found.

6.2. Else if ptr->info < x the call deletion pointer by passing the root right pointer and the item.

6.3. Else if ptr-> info > x then call

delete pointer by passing the left pointer and the item.

6.3. ~~Else if ptr->info>x then call deletion~~

6.4. Check if ptr->info == item then check if ptr->left == ptr->right then free ptr and return null.

6.5. Else if ptr->left == null then set P1. ptr->right and free ptr, return P1.

6.6. Else if ptr->right == null set P1 = ptr->left and free ptr, return P1.

6.7. Else set P1 = ptr->right and P2 = ptr->right

6.8 While P2- left not equal to

null, set p1 -> left = ptr -> left and free ptr, return p2.

6.9. Return ptr.

7. If the user choice to perform search operation then call the pointer to perform search operation

7.1. Declare the necessary pointers and variables.

7.2. Read the element to be searched.

7.3 While ptr check if item > ptr -> info then ptr = ptr -> right.

7.4. Else if item < ptr -> info then ptr = ptr -> right

7.5 Else break.

7.6 Check if ptr then print that

the element found

7.7. Else print element not found in tree and return root.

8. If the user choose to perform traversal then call the traversal function and pass the root pointer

8.1. If root not equal to null recursively call the function by passing root -> left.

8.2. Print root -> info

8.3. Call the traversal function recursively by passing root -> right.

9. End.

Program No 7.

Aim: Program to perform operation on disjoint set.

Algorithm:-

1. Start
2. Declare the structure and related structure variable.
3. Declare a function makeSet()
   3.1 Repeat step 3.2 to 3.4 until i<n.
   3.2 dis.parent[i] is set to i.
   3.3. Set dis.rank[i] is equal to 0
   3.4. Increment i by 1.
4. Declare a function display set.

4.1. Repeat step 4.2 and 4.3 until
i<n

4.2. Print dis.parent[i]

4.3. increment i by 1.

4.4. Repeat set 4.5 and 4.6 until
i<n.

4.5. Print dis.rank[i]

4.6. Increment i by 1.

5. Declare a function find and pass
x to the function.

5.1 check if dis.parent[x]! = x
then set the return value to
dis.parent[x]

5.2 return dis.parent[x]

6. Declare a function union and
pass two variables x and y.

6.1 set x set to find(x)

6.2. set y set to find(y)

6.3. check if x set == y set then return.

6.4. check if ds.rank[rank] < dis.rank [xset] > dis. rank[y]

6.8 set x set to dis.parent [yset]

~~6.9. set + to dis.rank[x.set]+1~~

to

6.9. set -1 to dis.rank [yset].

6.10 Else disparent (yset). x set.

6.11 set dis.rank[x.set]+1 to dis.rank [x set]

6.12. set -1 to dis. rank [y.set].

7 Read the number of elements.

8. call the function of method.

9. Read the number of elements choice from user to perform union find and display operation.

10. If the user choose to perform union operation. Read the element to perform union then all the function to perform union operation.

11. If the user choose to perform find operation read the element to check if connected.

11.1. check if find (x) = = find (y) then print connected component

11.2. Else print Not connected component.

12. If the user chooses to perform display operations call function display ()

13. End