AIM : Program to Perform Merging operation.

Algorithm

Step 1 : Start

2 : Declare the variables

3 : Read the size of first array

4 : Read the elements of first array in sorted order.

5 : Read the size of second array.

6 : Read the elements of second array in sorted order.

7 : Repeat Step 8 and 9 while i<m & j<n

8 : Check if a[i] >= b[j] then c[k++] = b[j++]

9 : else c[k++] = a[i++]

10 : Repeat step 11 while i<m

11 : c[k++] = a[i++]

12 : Repeat Step 13 while j<n

13 : c[k++] = b[j++]

14 : Print the first array

```
15: Print the second array
16: Print the merged array
17: End.
```

Aim: Program to Perform Stack operations: Insert, delete, Search & display.

Algorithm

Step 1: Start

2: Declare the node and the required variables.

3: Declare the function for Push, Pop, display and search an element.

4: Read the choice from usee.

5: If the usee choose the push an element, then read the element to be pushed and call the function to push the element by passing the value to the function.

5.1: Declare the newnode and allocate memory for the newnode

5.2: Set newnode → data = value

5.3: check if top == null then set newnode → next = null

5.4: Set newnode → next = top

Step 5.5: Set top = new node and then Print insertion is successful

6: If user choose to POP an element from the stack then call the function to Pop the element.

6.1: Check if Pop == null then Print stack is empty.

6.2: Else declare a Pointer variable temp and initialize it to top

6.3: Print the element that being deleted

6.4: Set temp = temp → next.

6.5: Free the temp.

7: If user choose the display then call the function to the element in the stack.

7.1: check if top == null then Print Stack is empty.

7.2: else declare a pointer variable temp and initialize it to top.

7.3: Print temp → data.

7.4: Set temp = temp → next.

Step 8: If user choose the search an element from the stack then call the function to search an element.

8.1: Declare the pointer variable ptr and other necessary variable.

8.2: Initialize ptr = top

8.3: check if ptr = null then Print stack is empty.

8.4: else read the element to be searched.

8.5: Repeat step 8.6 to 8.8 while ptr != null

8.6: check if ptr -> data == item then Print element founded and to be located and set flag = 1

8.7: else get flag = 0

8.8: increement i by 1 and set ptr = ptr -> next

8.9: check if flag = 0 then Print the element not found.

Step 9: End.

Aim : Program to Perform Circular queue operations

Algorithm:

Step 1: Start

2: Declare the queue and three variables

3: Declare the functions for enqueue, dequeue, search and display.

4: Read the choice from the user.

5: If the user choose for choice enqueue, then read the element to be inserted from the user & call the enqueue function by passing the value.

5.1: check if front == -1 && rear == -1 then set front = 0, rear = 0 and set queue[rear] = element.

5.2: else if rear + 1 % max == front or front = rear + 1 then print Queue is overflow.

5.3: else set rear = rear + 1 % max and set queue[rear] = element.

6. If user choice is the option dequeue then call the function dequeue.

step 6.1: check if front == -1 and rear == -1 then Print queue is underflow.

6.2: else check if front == rear then Print the element is to be deleted. Then set front = -1 and rear = -1.

6.3: else Print the element to be dequeued Set front = front + 1 % max.

7: if user choice is to display the queue then call the function display.

7.1: check if front = -1 and rear = -1 then Print queue is empty.

7.2: else Repeat the step 7.3 while i <= rear.

7.3: Print queue[i] and set i = i+1 % max.

8: If user choose the search then call the function to search an element in the queue.

8.1: Read the element to be searched in the queue

8.2: check if item == queue[i] then Print item found & its position and increement i by 1

step 8.3: check c==0 then print item not found.

9: End

Aim: Program to perform operations of Doubly linked list: Insertion, Search and display.

Algorithm:

Step 1: Start

2: Declare a structure and related variables.

3: Declare functions to create a node, insert a node in the beginning, at the end and given position, display the list and search an element in the list.

4: Define function to create a node, declare the required variables.

4.1: Set memory allocated to the node= temp. then set temp → Prev = null and temp → next = null.

4.2: Read the value to be inserted to the node.

4.3: Set temp → n = data and increment count by 1.

5: Read the choice from the user to perform different operation on the list.

Step 6 : If the user choose to Perform insertion operation at the beginning then call the function to perform the insertion.

6.1 : check if head == null then call the function to create a node, Perform step 4 to 4.3 .

6.2 : Set head = temp & temp1 = head .

6.3 : else call the function to create a node, Perform step 4 - 4.3 then Set temp → next = head, Set head → Prev = temp and head = temp.

7 : If the user choise is to Perform insertion at the end of the list, then call the function to Perform the insertion at the end.

7.1 : check if head == null then call the function create a newnode then set temp = head & then Set head = temp1

7.2 : Else call the function to create a new node then set temp1 → next = temp, temp → prev = temp1 and temp1 = temp.

8 : If the user choose to Perform insertion in the list at any Position then call the function to Perform the insertion operation.

step 8.1: Declare the necessary variable.

8.2: Read the position where the node need to the inserted, set temp2 = head

8.3: check if pos < 1 OR pos >= count + 1 then Print the position is out of range.

8.4: check if head == null & pos = 1 then Print "Empty list cannot insert other than 1st position".

8.5: check if head == null & pos = 1 then call the function to create newnode, then set temp = head and head = temp1

8.6: while i < pos then set temp2 = temp2 -> next then increement i by 1.

8.7: call the function to create a new node and then the set temp -> prev = temp2, temp -> next = temp2 -> next -> prev = temp.

temp2 -> next = temp.

9: If the user choose to perform deletion operation in the list then call the function to perform the deletion operation

Step 9.1: Declare the necessary variables.

9.2: Read the position where node need to be deleted set temp2 = head

9.3: check if pos<1 or pos >= count+1. then Print position out of range.

9.4: check if head == null then Print the list is empty.

9.5: while i< pos then temp2 = temp -> next and increment i by 1

9.6: check if i == 1 then check if temp2 -> next == null then Print node deleted free(temp2) set temp2 = head = null.

9.7: check if temp2 -> next == null then temp2 -> Prev -> next = null then free(temp2) then Print node deleted.

9.8: temp2 -> next -> Prev = temp2 -> Prev then check if i! = 1 then temp2 -> Prev -> next = temp2 -> next.

9.9: check if i=1 then head = temp2 -> next then Print node deleted then free temp2 and decrement count by 1.

Step 10 : If the user choose to perform the display operation then call the function to display the list

10.1 : Set temp2 = n

10.2 : Check if temp2 = null then Print list is empty.

10.3 : while temp2 -> next != null then Print temp2 -> n then temp2 = temp2 -> next.

11 : If the user choose to perform the Search operation then call the function to perform Search operation.

11.1 : Declare the neccessary variables.

11.2 : Set temp2 = head.

11.3 : check if temp2 == null then Print the list is empty.

11.4 : Read the value to be searched.

11.5 : while temp2 != null then check if temp2 -> n == data then Print element found at Position count+1

11.6 : Else Set temp2 = temp2 -> next and increment count by 1

Step 11.7 : Print element not found in the list

12 : End

Aim : Program to Perform Set operations

Algorithm :

Step 1 : Start

2 : Declare the neccessary variable

3 : Read the choice from users to Perform Set operation.

4 : If the user choose to Perform union.

4.1 : Read the cardinality of 2 sets.

4.2 : Check if m1 = n then Print Cannot Perform union.

4.3 : else read the elements in both the sets.

4.4 : Repeat the step 4.5 to 4.7 until i<m

4.5 : C[i] = A[i] | B[i]

4.6 : Print c[i]

4.7 : Increment i by 1

5 : Read the choice from the user to Perform intersection.

5.1 : Read the cardinality of 2 sets.

step 5.2: check if m1=n then Print Cannot Peeform intersection.

5.3: else read the elements is both the sets

5.4: Repeat the step 5.5 - 5.7 until i<m

5.5: c[i] = A[i] & B[i]

5.6: Print c[i]

5.7: increment i by 1

6: If the cese choose to Peeform set difference opeeation

6.1: Read the cardinality of 2 sets.

6.2: check if m1=0 then Print Cannot Peeform set difference opeeation.

6.3: else read the element in both sets.

6.4: Repeat the step 6.5 - 6.8 until i<n

6.5: check if A[i] ==0 then c[i] = 0

6.6: else if B[i] ==1 then c[i]=0

6.7: else c[i]=1

6.8: Increment i by 1

Step 7 : Repeat the step 7.1 and 7.2 until i<m

7.1 : Print i<m

7.2 : Increment i by 1

8 : End

Aim : Program to implement Binary Search Tree.

Algorithm:

step 1 : Start

2 : Declare a structure and structure ~~variable~~ Pointer for insertion, deletion and search operations, and also declare a function for inorder traversal.

3 : Declare a pointer as root and also the required variable.

4 : Read the choice from the user to Perform insertion, deletion, searching and inorder traversal.

5 : If the user choose to Perform insertion operation then read the value which is to be inserted to the tree from the user.

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.

step 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 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 the delete pointer.

6.1: check if not ptr then print node not found.

6.2: else if ptr → info < x then call delete pointer by passing the right pointer and the item.

6.3: else if ptr → info > x then call delete pointer by passing the left pointer and the item.

step 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. then set p1 = ptr -> left and free ptr, return p1.

6.7: else set p1 = ptr -> Right and p2 = ptr -> right.

6.8: while p1 -> left not equal to null, Set p1 -> left ptr -> left and free ptr, return p2.

6.9: Return ptr.

7: If the user choose ~~the~~ to Perform Search operation then call the pointer to Perform Search operation.

7.1: Declare the neccessary pointer & variable

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 -> left.

step 7.5: else break

7.6 : check if ptr then print that the element is found.

7.7 : else print element not found in tree and return root.

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

8.1 : If root not equals to null recursively call the functions by passing root → left.

8.2: Print root → info

8.3: call the traversal function recursively by passing root → right

9 : End.

Aim : Program to perform dis joint.

Algorithm:

Step 1: start

2: Declare the structure and related structure variable.

3: Declare a function makeset()

3.1: Repeat step 3.2 - 3.4 until i<n

3.2: dis.parent[c] is set to i

3.3: Set dis.rank[c] is equal to 0

3.4: Increment i by 1

4: Declare a function display set

4.1: Repeate step 4.2 and 4.3 until i<n

4.2: Print dis.parent[i]

4.3: increment i by 1

4.4: Repeat step 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

Step 5.1: Check if dis.parent[x] != x then Set the return to dis.parent[x]

5.2: Return dis.parent[x]

6: Declare a function union & pass two variable x & 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

6.4: Check if dis.rank[x set] < dis.rank[y set]

6.5: Set y set = dis.parent[y set]

6.6: Set -1 to dis.rank[x set]

6.7: Else if check dis.rank[x set] > dis.rank[y set].

6.8: Set x set to dis.parent[y set]

6.9: set -1 to dis.Rank[y set]

6.10: else dis.parent[y set] = x set

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

6.12: Set -1 to dis.parent[y set]

7: Read the number of elements

8: call the function makeset()

Step 9 : Read the 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 operations.

11 : If the users 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 choose to perform display operation call the function display set.