MERGING

Step 1 : Start
Step 2 : Declare the variables
Step 3 : Read the size of first array
Step 4 : Read elements of first array in sorted order
Step 5 : Read the size of second array
Step 6 : Read the element of second array in sorted array order
Step 7 : Repeat Step 8 and 9 while i<m and j<n
Step 8 : check if a[i] = b[j] then c[k++] = b[j++]
Step 9 : Else c[k++] = a[i++]
Step 10 : Repeat Step 11 while i<m
Step 11 : c[k++] = a[i++]
Step 12 : Repeat Step 13 while j<n
Step 13 : c[k++] = b[j++]
Step 14 : Print the first array
Step 15 : Print the second array
Step 16 : print the merged array
Step 17 : End

STACK OPERATIONS

Step 1 : Start

Step 2 : Declare the node and the required variables

Step 3 : Declare the function for push pop display and search

Step 4 : Read the choice from the user do push, pop display or search an element

Step 5 : if the user choose do push an element, then read the element to be pushed and call the function do push the element by passing the value to the function

Step 5.1 : Declare the new node and allocate memory for the new node

Step 5.2 : Set new node ⇒ data = value

Step 5.3 : Check if top == Null then Set newnode ⇒ next = null

Step 5.4 : Else Set new node ⇒ next = top

Step 5.5 : Set top = new node and then print insertion is Successful

Step 6 : if the user choose to pop an element

from the Stack then call the function to pop the element.

Step 6.1 : Check if top == Null then print Stack is Empty

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

Step 6.3 : Print the element that is being deleted

Step 6.4 : Set temp => next = temp

Step 6.5 : free the temp

Step 7 : if the user choose to display the element in the Stack then call the function to display the element in the Stack

Step 7.1 : Check if top == Null then Print Stack is empty.

Step 7.2 : Else declare a Pointer variable temp & initialize it to top

Step 7.3 : Repeat Steps below while temp -> next != null

Step 7.4 : Print temp => data

Step 7.5 : Set temp = temp -> next

Step 8 : if the user choose to Search an element from the Stack then call the function to Search an element

Step 8.1 : Declare a pointer variable ptr and other necessary variable

Step 8.2 : Initialize ptr = top

Step 8.3 : Check if ptr = null then Print Stack empty

Step 8.4 : Else read the element to be Searched

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

Step 8.6 : Check if ptr -> data == item then print element founded and to be located and Set flag = 1

Step 8.7 : Else Set flag = 0

Step 8.8 : increment i by 1 and Set ptr = ptr -> next

Step 8.9 : Check if flag = 0 then print the element not found

Step 9 : End

Circular Queue operations

Step 1 : Start

Step 2 : Declare the queue and other variables

Step 3 : Declare the functions for enqueue, dequeue Search and display

Step 4 : Read the choice from the user

Step 5 : if the user choose the choice enqueue then read the element to be inserted from the user and call the enqueue function by Passing the value.

Step 5.1 : Check if front == -1 && rear == -1 then Set front = 0, rear = 0 and Set queue[rear] = element.

Step 5.2 : Else if rear + 1 % max == front or front == rear + 1 then print queue is overflow

Step 5.3 : Else Set rear = rear + 1 % max and Set queue[rear] = element

Step 6 : if the 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

Step 6.2 : Else check if front == rear then print the element is to be deleted then set front = -1 and rear = -1

Step 6.3 : Else print the element to be dequed set front = front + 1 % max

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

Step 7.1 : check if front = -1 and rear = -1 the print Queue is empty

Step 7.2 : Else repeat the Step 7.3 while i <= rear

Step 7.3 : print queue[i] and set i = i + 1 % max

Step 8 : if the user choose the Search then call the function to Search an element in the Queue.

Step 8.1 : Read the element to be Searched in the Queue

Step 8.2 : Check if item == Queue[i] then print item found and its position and increment i by 1

Step 8.3 : Check if == 0 then print item not found

Step 9 : Stop

Doubly linked list operation.

Step 1 : Start

Step 2 : Declare a structure and related variable

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

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

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

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

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

Step 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 begining then call the function to perform the insertion

Step 6.1 : Check if head == null then call the function to create a node perform Step 4 to 4.3

Step 6.2 : Set head = temp and temp1 = head

Step 6.3 : Else call the function to create a node Perform Step 4 to 4.3 then Set temp -> next = head, Set head -> prev = temp and head = temp

Step 7 : if the user choice is to perform insertion at the end of the list. then call the function to perform the insertion at the end.

Step 7.1 : Check if head == null then call the function to create a newnode then Set temp = head and then Set head = temp1

Step 7.2 : Else call the function to create a new node then Set temp1 -> next = temp, temp -> prev = temp1 and temp1 = temp

Step 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.

Step 8.2 : Read the position where the node need to be inserted, Set temp2 = head

Step 8.3 : check if pos < 1 or pos > = count + 1 then print the position is out of range

Step 8.4 : check if head and pos = 1 then print "Empty list cannot insert other than 1st Position."

Step 8.5 : check if head = = null and pos = 1 then call the function to create newnode then Set temp = head and head = temp1

Step 8.6 : while i < pos then Set temp2 = temp2 -> next increment i by 1

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

Step 9 : if the user choose to perform deletion operation in the list then all the function to perform the deletion operation.

Step 9.1 : Declare the necessary variable.

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

Step 9.3 : check if pos<1 or pos >= count+1 then print Position out of range

Step 9.4 : check if head == null then print the list is empty

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

Step 9.6 : check if i==1 then check if temp2 -> next == null then print node deleted free (temp2) Set temp2 = head = null

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

Step 9.8 : temp2 -> next -> prev = temp2 -> prev then check if i!=1 then temp2 -> prev -> next = temp2 -> next

Step 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

Step 10.1 : Set temp2 = n

Step 10.2 : check if temp2 = null then print list is empty

Step 10.3 : while temp2 ⇒ next1 = null then print temp2 ⇒ u then temp2 = temp2 ⇒ next

Step 11 : if the user choose to perform the Search Operation then call the function to Search operation.

Step 11.1 : Declare the necessary variable

Step 11.2 : Set temp2 = head

Step 11.3 : check if temp2 == null then print the list is empty

Step 11.4 : Read the value to be Searched

Step 11.5 : while temp2 != null the check if temp2 ⇒ n == data then print element found at Position count+1

Step 11.6 : else Set temp2 = temp2 ⇒ next and increment count

Step 11.7 : print element not found in the list

Step 12 : Stop

Set Operations

Step 1: Start

Step 2: Declare the necessary variable

Step 3: Read the choice from the user do perform Set operation.

Step 4: if the user choose do perform union.

Step 4.1: Read the cardinality of 2 Sets

Step 4.2: check if m1 = n then print cannot perform union.

Step 4.3: Ehen read the elements in both the Sets

Step 4.4: Repeat the Step 4.5 do 4.7 until i<m

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

Step 4.6: print C[i]

Step 4:7: increment i by 1

Step 5: Read the choice from the user do perform intersection.

Step 5.1: Read the cardinality of 2 Sets

Step 5.2: Check if m|=n then print cannot perform intersection.

Step 5.3 : Else read the elements in both the sets

Step 5.4 : Repeat the Step 5.5 to 5.7 until i<m

Step 5.5 : C[i] = A[i] & B[i]

Step 5.6 : Print C[i]

Step 5.7 : increment i by 1

Step 6 : if the user choose to perform set difference operation.

Step 6.1 : Read the cardinality of 2 sets

Step 6.2 : Check if m!=n then print cannot perform Set difference operation.

Step 6.3 : Else read the element in both sets

Step 6.4 : Repeat the Step 6.5 to 6.8 until i<m

Step 6.5 : Check if A[i] == 0 then C[i]=0

Step 6.6 : Else if B[i] == 1 then C[i]=0

Step 6.7 : Else C[i]=1

Step 6.8 : increment i by 1

Step 7 : Repeat the Step 7.1 & 7.2 until i<m

Step 7.1 : Print C[i]

Step 7.2 : increment i by 1

Binary Search Tree

Step 1 : Start

Step 2 : Declare a structure and structure pointers for insertion deletion and search operation and also declare a function for inorder traversal.

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

Step 4 : Read the choice from the user do perform insertion, deletion, searching and inorder traversal

Step 5 : if the user choose to perform insertion operation then read the value which is to be inserted to the tree from the user

Step 5.1 : pass the value to be inserted pointer and also the root Pointer

Step 5.2 : Check if ! root then allocate memory for the root.

Step 5.3 : Set the value to the info part of the

and then Set left and right part of the root do the null and return root

Step 5.4 : Check if root → info > x then call the insert Pointer to insert to left of the root

Step 5.5 : check if root → info < x then call the insert Pointer to insert to the right of the root

Step 5.6 : Return the root

Step 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

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

Step 6.2 : Else if ptr → info < x the call delete pointer by passing the right pointer and the item

Step 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 & return null

Step 6.5 : Else if ptr → left == null then Set p1. ptr → right and free ptr, return p,

Step 6.6 : Else if ptr → right == null then Set p1 = ptr → left and free ptr, return p1

Step 6.7 : Else Set p1 = ptr → right and p2 = ptr → right

Step 6.8 : while p1 → left not equal to null, Set p1 → left ptr → left and free ptr, return p2

Step 6.9 : Return ptr

Step 7 : if the user choose to perform Search operation the call the pointer to perform Search operation

Step 7.1 : Declare the necessary pointers and variable

Step 7.2 : Read the element to be Searched

Step 7.3 : while ptr check if item > ptr → info then ptr = ptr → right

Step 7.4 : Else if item < ptr → info then ptr = ptr → left

Step 7.5 : The break

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

Step 7.7 : The print element not found in tree and return root

Step 8 : if the user choose to perform traversal then call the traversal function and pass the root pointers.

Step 8.1 : if root not equals do null recursively call function by paring root ⇒ left

Step 8.2 : point root ⇒ info

Step 8.3 : call the traversal function recursively by paring root ⇒ right

Disjoint Sets

Step 1 : Start

Step 2 : Declare the structure and related structure variable

Step 3 : Declare a function makeset c)

Step 3.1 : Repeat Step 3.2 to 3.4 until i<n

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

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

Step 3.4 : increment i by 1

Step 4 : Declare a function display set

Step 4.1 : Repeat Step 4.2 and 4.3 until i<n

Step 4.2 : print dis parent [i]

Step 4.3 : increment i by 1

Step 4.4 : Repeat Step 4.5 and 4.6 until i<n

Step 4.5 : print dis rank [i]

Step 4.6 : increment i by 1

Step 5 : Declare a function find and pass x to the function

Step 5.1 : check if dis parent (x) != x then set the return value to the dis parent (x)

Step 5.2 : return disparent (x)

Step 6 : Declare a function union and pass two variables x and y

Step 6.1 : Set x Set do find (x)

Step 6.2 : Set y Set do find (y)

Step 6.3 : check if x Set = = y Set then return

Step 6.4 : check if dis rank [x Set] < dis rank [y Set] then

Step 6.5 : Set yset yset : dis.parent (y Set)

Step 6.6 : Set -1 to dis rank [x Set]

Step 6.7 : Else if check dis rank (x Set) > dis. rank (y Set]

Step 6.8 : Set x Set do dis parent (y Set)

Step 6.9 : Set -1 to dis rank [y set]

Step 6.10 : Else dis.parent [yset] = x Set

Step 6.11 : Set dis rank [x Set) + 1 do dis rank [x Set]

Step 6.12 : Set -1 do dis rank [y Set]

Step 7 : Read the number of elements.

Step 8 : Call the function makeset

Step 9 : Read the cochoice from user do ~~prgm~~ Perform union find unet display operation

Step 10 : if the user choice to perform union operation Read the element to perform union then call the function to Perform union operation.

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

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

Step 11.2 : Else print not connected component

Step 12 : if the user choose to perform display operation call the function display &d

Step 13 : end.