No:1 Date:31/12/2020

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

Step1: Start

Step2: Declare the variables, a[m], b[n], c[m+n]

Step3: Read the size of first array.

Step4: Read the elements of first array in sorted order.

Step5: Read the size of second array.

Step6: Read the elements of second array in sorted order.

Step7: Repeat the steps 8 and 9 while while (i<m && j<n).

Step8: check if a[i] >= b[j] then c[k++]=b[j++]

Step9: else c[k++]=a[i++]

Step10: Repeat step 11 while i<m.

Step11: c[k++]=a[i++]

Step12: Repeat step 13 while i<m

Step13: c[k++]=b[j++]

Step14: Print the first array

Step 15: Print second array

Step 16: Print merged array.

Step 17: End.

NO:2 Date: 3/1/2021

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

Step 1: Start

Step 2: Declare Variables, node.

Step 3: Declare functions for push, pop, search, display.

Step 4: Readd the choice from user using switch

Step 5: If the user choose to push an element, then call the function push(), and readd the value to be pushed.

Step 5.1: Declare new node, allocate memory for new node.

Step 5.2: Set newnode -> Data = Value

Step 5.3: If top == null then set newnode -> next = null.

Step 5.4: Set newnode -> next = top.

Step 5.5: Set top = newnode & print successful insertion.

Step 6: If user choose to pop an element, then call pop()

Step 6.1: If top == Null, Print stack is empty

Step 6:2: Declare *temp, initialize temp=top.

Step 6:3: print element that being deleted
set temp=temp->next.
free(temp).

Step 7: If selected choise is display then call display().

Step 7:1: If (top==Null) print empt stack.

step 7.2: else declare *temp, temp=top.

Step 7.3: Repeate steps bebow while(temp->next != null)

Step 7:4: print temp->data.
temp=temp->next.

Step 8: If selected choise is search, then call Search().

Step 8.1: Declare *ptr, ptr=top.

Step 8:2: check if ptr=null then print, empty stack

Step 8.3: else read the element to be searched.

Step 8.4: Repeat the steps 8.5 to 8.7.
while (ptr!=Null

Step 8:5: If (ptr->data == item)

then print element found, flag = 1

Step 8.6: else set flag = 0

Step 8.7: Increment i by 1 and set ptr = ptr -> next

Step 8.8: If flag = 0, print element not found.

Step 9: end.

NO:3 Date: 7/1/2021

Aim: Circular Queue: Add, Delete search.

Step 1: Start

Step 2: Declare the functions for enqueue and dequeue, Search and display, Queues.

Step 3: Read the choice from the user.

Step 4: If the user choose the choice enque, the read the element to be inserted, call the insert() function.

Step 4.1: Check if (front == -1 && rear == -1) then

Set front = rear = 0 and

Set Queue[rear] = element.

Step 4.2: else if (rear+1 % max == front

or front == rear + 1)

Print Queue is overflow.

Step 4.3: else set rear = rear + 1 % max and

Set Queue[rear] = element.

Step 5: If the user choice is deque then call delete()

Step 5.1: If (front == -1 && rear == -1), print

Queue is under-flow.

Step 5.2: else check if front == rear then print the element is to be dele-ted then set front = -1, rear = -1

Step 5.3: Else print element to be dequeued
Set front = front + 1 % max.

Step 6: If choice is display then call display()

Step 6.1: check if front = -1 and rear = -1 then.
print Queue is empty.

Step 6.2: else repeat step 6.3 while i <- rear.

Step 6.3 : print Queue[i] and set i = i + 1 % max

Step 7: If the choice is search then call search()

Step 7.1: Read the element to be searched.

Step 7.2: If item = Queue[i] then print item found and its position, i++

Step 7.3: If c == 0 print item not found.

Step 8. End.

No:4 Date:14/1/2021

Aim:- Doubly linked list - Insertion, Deletion, Search.

Step1: Start.

Step2: Declare structure.

Step3: Declare functions to create a node, insert a node in begining, at the end, given position, display, search.

Step4: Define function to create a node, declare the required variables.

step4.1: Set memory allocated to the node=temp. then set temp->prev=null and temp->next=null

Step4.2: Read the value to be inserted to the node

Step4.3: Set temp->n=data and increment count by 1.

Step4.3: Read the choice from the user.

Step6: If choice is insertion then call insertion operation at the begining then call the corresponding function.

Step6.1: check if head==null then call the function to create a node. Step4 to 4.3

Step 6.2 : Set head=temp & temp1=head.

Step 6.3 : Set head=1 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 choice is insertion at the end. Call the function.

Step 7.1 : If head==null then call function to create a neonode, set temp=head and 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 at any position then call the corresponding function.

Step 8.1 : Declare the necessary variables.

Step 8.2 : Read the position, element of the node to be inserted, Set temp2=head

Step 8.3 : If pos<1 or pos>=count+1 then print out of

Range.

Step8.4: If head==null, pos=1 then print "cannot insert other than 1st position.

Step8.5: If head==null and pos=1 then call the function to create new node, set. temp=head, and head=temp1.

Step8.6: While ~~pos~~ < i < pos. then set

temp2 = temp2->next.

Step8.7: call the function to create new node. then set temp->prev=temp2.

temp->next=temp2->next->prev=temp.

temp2->next=temp.

Step9: If the user choose to perform deletion operation. Then call the function for deletion.

Step9.1: Declare the Variables.

Step9.2: Read the position where the node need to be deleted set temp2=head.

Step9.3: If pos<1 or pos>=count+1 then ring out of range

Step9.4: If head==null print empty

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

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

Step 9.7: If (-temp2 -> nex == null) -then.
-temp2 -> prev -> next=null.
free (-temp2). print deleted.

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

Step a.a: If i= 1 -then head=temp->next -then.
print. node deleted free temp.

Step 10: If the user chose display.

Step 10.1 : Set -temp2 =head.

Step 10.2 = check if temp2=null -then -then print.
is empty.

Step 10.3 : While -temp2 ->next =null -then print.
temp2 ->n -then
temp2 = temp2-> next.

Step 11 : If the user chose search.

Step 11.1 : Declare the necessary Variables.

Step 11.2 : Set -temp2=head.

No: 5 Date: 20/1/2021

Aim: Set operations (Union, Intersection and difference) Using bitstring

Step1: Start.

Step2: Declare the neccessary variables.

Step3: Read the choice from the user.

Step4: IF the user chose to perform union.

Step4.1: Read the cardinality of two sets.

Step4.2: check if m!=n print cannot perform union

Step4.3: else read the elements in both the sets.

Step4.4: Repeat the steps 4.5 to 4.7 until i<m.

Step4.5: C[i] = A[i]|B[i]

Step4.6: increment i.

Step 4.7: print

Step5: If the user coants to perform intersection.

Step5.1: Read the cardinality of two sets.

Step5.2: If m!=n. print cannot perform intersection.

Step5.3: else read the elements

Step5.4: Repeat steps 5.5 to 5.7 until i<m

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

Step5.6: increment i

Stepp5.7: Print [i].

Step6: If the user choose to perform difference operation

Step6.1: Read the cardinality.

Step6.2: if(m!=n) Print cannot perform difference operation.

Step6.3: else read the elements

Step6.4: Repeat the steps 6.5 to 6.8 until i<n.

Step6.5: if A[ ]==0 then c[i]=0.

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

step 6.7 else if C[i]=1.

step 6.8 increment i.

step7: Repeat the steps 7.1, 7.2 until i<m.

Step7.1: print C[i]

Step 7.2: i++.

Step8: Stop.

NO: 6 Date: 2/02/2021

Aim: Binary Search trees - Insertion, Deletion, Search

Step1: Start.

Step2: Declare a structure and structure pointers, functions for.

Step3: Declare a pointer as root and also the required variables

Step4: Read the choice from the user.

Step5: If the user chose to perform insertion operation then read the value which is to be inserted.

Step5.1. Pass the value to the insert pointer and also the root.

Step5.2: If root then allocate memory for the root

Step5.3: 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.

Step5.4: If root->info>x then call the insert pointer to insert to left of the root.

Step5.5: If root->info<x, then call insert() to the right of root.

Step 5.6: Return root.

Step6: If the user choose to perform deletion operation. Then read element to be deleted, pass the root and item to delete.

Step6.1: check if not ptr. then print node not found.

Step6.2: else if ptr->info<x. call delete() at the right

Step6.3 else if ptr->info>x. then call delete() at the left.

Step6.4: if ptr->info==item. check ptr->left==ptr->right.
free(ptr)
return Null.

Step6.5: else if ptr->left==null then set P1=ptr->right and free(ptr) return (P1)

Step6.6: else if ptr->right==null.
P1=Ptr->lef.
free(ptr), return(P1)

Step6.7: else set P1=ptr->right and P2=ptr->right.

Step6.8: while (ptr->left != Null),

~~Write~~

Set p1->left=>ptr=>left.

and free ptr, return p2.

Step 6.9 : return ptr.

Step 7 : If the user choose search operation.

Step 7.1 : Declare the pointers and variables.

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 . else break.

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

Step 7.7 . Else print element not found in tree and return true.

Step 8 . If the user choose to preform traversal

Step 8.1 : If root not equals to null, recursively call the functions by passing root->left

Step 8.2 print root->info.

Step 8.3: call the traversal function recursively by passing root->right

Step 9: stop

No: 7 Date: 19/02/2021

Aim:- Disjoint sets and the associated operations (Create, Union, find).

Step1: Start.

Step2: Declare the structure and related structure variable

Step3: Declare a function makeset().

Step3.1: Repeat step 3.2 to 3.4 until i<n.

Step3.2: dis.parent[i] is set to i

Step3.3: Set dis.rank[i] is equal to 0.

Step3.4: Increment i by 1.

Step4: Declare a function display set.

Step4.1: Repeat step 4.2 and 4.3 until i<n.

Step4.2: Print dis.parent[i]

Step4.3: Increment i by 1.

Step4.4: Repeat step 4.5 and 4.6 until i<n

Step4.5: Print dis.rank[i].

Step4.6: increment i by 1.

Step5: Declare a function find and pass x to the function

Step5.1: check if dis.parent[n]!=x then set

the return value to dis.parent [x].

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

Step 6.1: set X set to find (x).

Step 6.2: Set Y set to 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 Y set = 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 to dis.parent [Y set]

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

Step 6.10: Else dis.parent [Y set] = X set

Step 6.11: Set dis.rank [X set] +1 to dis.rank [Y set]

Step 6.12: Set -1 to dis.rank [Y set]

Step 7: Read the number of elements.

Step 8: call the function makeset()

Step 9: Red the choice from user.

Step 10: If the choice is union, then read the elements to perform union, if it

Step 11 is find operation, ~~check~~ Read the elements to check if they are connected.

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

Step 11.2: Else print not connected

Step 12: If ~~you~~ user choose disply then call display().

Step 13: Stop.