MERGING

Step 1 :- Start

Step 2 :- Declare the Variables

Step 3 :- Read the size of the first array

Step 4 :- Read elements of first array in sorted order

Step 5 :- Read the size of second array

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

Step 7 :- Repeat step 8 and 9 while i<m && 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.

output

Enter number of elements in array 1 : 4
Enter array 1 in sorted order : 1, 3, 5, 7
Enter number of elements in array 2 : 4
Enter array 2 in sorted order : 1, 4, 6, 7
first array is : 1, 3, 5, 7
second array is : 2, 4, 6, 7
merged array is : 1, 2, 3, 4, 5, 6, 7, 7

STACK OPERATIONS

Step 1 :- Start

Step 2 :- Declare the node and the required variables

Step 3 :- Declare the functions for push, pop display and search an element

Step 4 :- Read the choice from the user.

Step 5 :- If the user choose to push an element then read the element to be pushed & call the function to push the element by passing the value to the function

Step 5.1 :- Declare the newnode & allocate memory for the newnode

Step 5.2 :- Set Newnode -> data - value

Step 5.3 :- check if top == null then set Newnode -> next = null

Step 5.4 :- Set newnode -> Next = top

Step 5.5 :- Set top = newnode & then print insertion is success-full

Step 6 :- If 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 being deleted

Step 6.4 :- Set temp = temp -> next

Step 6.5 :- Free the temp

Step 7 :- If the user choose the display 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.

output

choices :-

1. push
2. pop
3. Display
4. Search
5. Exit

Enter your choice : 1

Enter the value to be insert : 7

insertion is successfull

choices :-

1. push
2. pop
3. Display
4. Search
5. Exit

Enter your choice : 2

Popped element : 14

choices

1. push
2. pop
3. Display
4. Search
5. Exit

Enter your choice : 3

7 ⟶ null

Enter your choice : 4

Enter the item to be searched : 2
Item not found

choices :-

1. push
2. pop
3. display
4. search
5. Exit

Enter your choice :- 5

Circular Queue operation

Step 1 :- start

Step 2 :- Declare the queue and other variable

Step 3 :- Declare the function 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 under-flow

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 dequeued 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 then 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 c==0 then print item not found

Step 9 :- end.

output

choices :-

1. Insert
2. Delete
3. Display
4. Search
5. Exit

Enter your choice :- 1

Enter the number to be inserted : 7

choices :-

1. Insert
2. Delete
3. Display
4. Search
5. Exit

Enter your choice : 1

Enter the number to be inserted : 14

choices :-

1. Insert
2. Delete
3. Display
4. Search
5. Exit

Enter your choice 2

7 was deleted

choices:-

1. Insert
2. Delete
3. Display
4. Exit
5. Search

Enter your choice : 3

14

choices :-

1. Insert
2. Delete
3. Display
4. Search
5. Exit

Enter your choice : 4

Enter the element to be search: 14

Item found at location : 2.

Doubly linked list operations

Step 1 :- Start

Step 2 :- Declare a structure and related variables

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 step 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 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 the call the function to perform the insertion operation

Step 8.1 :- Declare a 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 == null 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 New node then set temp = head and head = temp1

Step 8.6 :- while i < pos then set temp2 = temp2 -> next then 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 users choose to perform deletion operation is the list then call the function to perform the deletion operation

Step 9.1 :- Declare the necessary variables

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 -> next 1 = null then print temp2 -> n then temp2 = temp2 -> next

Step 11 :- If the user choose to perform the search operation then call the function to perform search operations

Step 11.1 :- Declare the necessary variables

Step 11.2 :- step tep2 = 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 then check IF temp2 -> n == data then print element found at position count + 1

Step 11.6 :- else set temp2 = temp2 -> next and increment count by 1

Step 11.7 :- print element not found in the list

Step 12 :- End.

output

1. Insert at beginning
2. Insert at end
3. Insert at position 1
4. Delete at i
5. Display from beginning
6. Search for element
7. Exit

Enter choice 1
Enter the value of node :- 3
Enter choice 1
Enter the value of node : 4
Enter choice 5
linked list element from beginning : 4, 3
Enter choice :3
Enter the position to be inserted : 2
Enter value to node 1
Enter choice : 2
Enter value to node : 6
Enter choice : 5
linked list element from beginning : 4, 1, 3, 6
Enter choice : 4
Enter position to be deleted 1
node deleted
Enter choice : 5
linked list element from beginning : 1, 3, 6
Enter choice : 4
Enter position to be deleted : 3
Node deleted from the list
Enter choice : 5

linked list element from beginning : 1, 3
Enter choice : 6
Enter value to be search : 7
Error : not found in list
Enter choice : 7

Set operations

Step 1 :- Start

Step 2 :- Declare the necessary variable

Step 3 :- Read the choice from the user to perform set operation

Step 4 :- If the user choose to perform Union

Step 4.1 :- Read the cardinality of a 2 sets

Step 4.2 :- check if m != n then print cannot perform Union

Step 4.3 :- else read the elements in both the sets

Step 4.4 :- Repeat the step 4.5 to 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 to 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[j]

Step 5.6 :- print C[i]

Step 5.7 :- Increment i by 1

Step 6 :- If the user choose to perform det different 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<n
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 and 7.2 until i<m
Step 7.1 :- print C[i]
Step 7.2 :- increment i by 1

output

```
choice to perform
1. Union
2. Intersection
3. Difference
4. Exit
choice : 1
Enter first set : 3
Enter second set: 3
Enter first set (0/1): 1
1
Enter set (0/1) 0 0 1
Elements of set 1 Union set 2: 101
choice to perform
1. Union
2. Intersection
3. Difference
4. Exit
choice: 3
Enter first set : 4
Enter second set : 4
Enter first set :(0/1) 1
0
0
1
Enter second set :(0/1) 1
0
1
0
Elements of set 1 - set 2: 0 0 0 1
```

choice to perform
1. Union
2. Intersection
3. Difference
4. Exit
choice : 2
Enter first set : 3
Enter second set : 3
Enter first set (0 1) 1
0
0
Enter second set (0 1) 0
1
0
1
Intersection of set 1
Intersection set 2 100
choice to perform
1. Union
2. Insertion (Intersection)
3. Difference
4. Exit
choice : 4
Program Exit Successfully

Binary Search Tree

Step 1 :- Start

Step 2 :- Declare a structure and structure pointers for insertion deletion and search operations 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 to 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 6 :- pass the value to the insert 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 root and then set left and right part of the root to 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 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 then 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 and return null

Step 6.5 :- Else if ptr → left == null then set p1. ptr → left and free ptr. return p1

Step 6.6 :- Else if ptr → right == null then set p1. ptr → right 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 :- ~~If the user~~ Declare the necessary 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 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 to null recursively call the function by passing root → left

Step 8.2 :- print root → info

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

output

1. Insert in binary tree
2. Delete from binary tree
3. Inorder traversal of binary tree
4. Search
5. Exit

choice 1
Enter new element : 10
Root is 10
Inorder traversal of binary tree is : 10

1. Insert in binary tree
2. Delete from binary tree
3. Inorder traversal of binary tree
4. Search
5. Exit

Enter choice 1
Enter new element : 12
Root is 10
Inorder traversal of binary tree : 10, 12

1. Insert in binary tree
2. Delete from binary tree
3. Inorder traversal of binary tree
4. Search
5. Exit

Enter choice : 1
Enter new element : 16
Root is 10.
Inorder traversal of binary tree : 10, 12, 16

1. Insert in binary tree
2. Delete from binary tree
3. Inorder traversal of binary tree
4. Search
5. Exit

Enter choice : 1

Enter new element : 14

Inorder traversal of binary tree : 10, 12, 14, 14

1. Insert in binary tree
2. Delete from binary tree
3. Inorder traversal of binary tree
4. Search
5. Exit

Enter choice : 1

Enter new element : 15

Root is 10

Inorder traversal of binary tree :- 10, 12, 14, 15

1. Insert in binary tree
2. Delete in binary tree
3. Inorder traversal of binary tree
4. Search
5. Exit

Enter choice : 2

Enter element to be deleted :- 15

10, 12, 14, 16

1 Insert in binary tree
2. Delete from binary tree
3. Inorder traversal of binary tree
4 Search
5 Exit

Enter choice : 3

Inorder traversal of binary tree is

10, 12, 14, 16

1. Insert in binary tree
2. Delete from binary tree
3. Inorder traversal of binary tree
4. Search
5. Exit

Enter choice : 4

Search operation in, binary tree

Enter the element to be searched : 16

Element 16 which was searched is found

and 15-16

1. Insert in binary tree
2. Delete from binary tree
3. Inorder traversal of binary tree
4. Search
5. Exit

choice 5

Disjoin set [create, union, find]

Step 1 :- Start

Step 2 :- Read the number of elements from the user and store it in to the dis n

Step 3 :- Call function make set(n) then

Step 4 :- Set i = 0

Step 5 :- Repeate for i < dis. n then
Set dis. parent [i] = i
Set dis. rank [i] = 0
Set i = i + 1
[end for loop]

Step 6 :- user select the union operation then

Step 7 :- Read the elements to perform union and store in to x and y respectively

Step 8 :- perform find operation in with x and y store result in to x set and y set perform step 2

Step 9 :- If x set == y set then [end y]

Step 10 :- If dis rank [x set] < dis. rank [y set] then : set dis. parent [x set] = y set. set dis rank [x set] =
[end if]

Step 11 :- else if dis. rank [x set] > dis. rank [y set] then
SET dis. parent [y set] = x set
SET dis. rank [x set] = dis rank [x set] + 1
SET dis rank [y set] = -1

Step 14 :- If user choose find operation then

Step 15 :- Read the elements to check. If and store the value into the variables x and y respectively

Step 16 :- If find x == find y then
Display "Connected Components"

Step 17 :- else
Display "No connected Components"

Step 18 :- If user select the display operation then

Step 19 :- set i=0

Step 20 :- Repeate for i< dis n then
print dis.parent[i]
set i = i+1
[end of for loop]

Step 21 :- If dis.parent[x] != x
then
set dis.parent[x] = find (dis.par[x])
Return dis parent[x]

Step 22 : Exit

Output

How many elements : 5

-- menu --

1 Union
2 Find
3 Display

Enter choice : 1

Enter the elements to perform Union : 3

Do you wish to Continue ? (1/0) : 1

-- menu --

1. Union
2. Find
3. Display

Enter choice : 3

Parent array : 0 1 2 2 4

Rank array : 0 0 1 -1 0

Do you want to Continue ? (1/0)

1

-- menu --

1. Union
2. Find
3. Display

enter choice : 2

Enter the elements to check I-I Connected Components

2
8

Connected Components

1

-- menu --

1 Union

2 Find

3 Display

Enter choice

2

Enter the elements to check if Connected Components 1

3

Connected Components

Do you wish to continue ? (1/0)

0