Amal Vijayan
MCA A
Roll No: 10

Data Structure Record

Program no:1 - Array Merging

Step 1 - start

Step 2 - Declare the variables

Step 3 - Enter the size of first array

step 4 - Read the Elements of first array

step 5 - Enter the size of the second array.

step 6 - Read the Elements of the second array.

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

Step 8 - if a[i] >= b[i] the 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<m

step 13 - c[k++] = b[j++]

Step 14 - Print the first, second and Merged array

Step 15 : End.

Output

Enter the size of first array,

3

Enter the elements

4

5

6

Enter size of second array

4

Enter the elements

2

3

7

8

Array A : 4 5 6

Array B : 2 3 7 8

Merged array: 2 3 4 5 6 7 8

Program No: 2 - Stack Operations

Step 1: start

Step 2: Declare the variables and functions for push, pop, display and search operations.

Step 3: Read the choice

Step 4: Call the function corresponding to users choice.

Step 5: Push operation

→ Read the element to be pushed.

→ Declare new node and allocate memory

→ ~~Set~~ Set newNode → data = value.

→ Check if top == null then newNode → next = null

→ set new node → next = top.

Step 6: Pop operation

→ Check if top == null. then print stack underflow

→ Else ~~ptr = top~~ temp = top

→ Set temp = temp → next

→ free temp.

Step 7: Display

→ Check top == null. print stack is underflow

→ else declare temp and set temp == top

→ Repeat steps below while temp→next != null

→ print temp→data

→ set temp = temp→next

step 8 : search operation

→ Declare pointer variable

→ Set ptr = top

→ If ptr = null, print stack underflow

→ Else read the key value

→ Repeat below step while ptr != null

→ If ptr→data == item, then print element found at loc.

→ Increment i by 1 and set ptr = ptr→next

step 9 : End.

Output

```
menu
1. push
2. pop
3. display
4. Search
5. Exit
Enter your choice : 1
Enter the Element to insert: 5
Menu
Enter your choice : 1
Enter the Element to insert : 10
Menu
Enter your choice : 3
10
5

Menu
Enter your choice
2.
Element deleted: 10
```

Program No: 3 - Circular Queue

Step 1: Start

Step 2: Declare queue, variables, functions for Enqueue, dequeue, search and display operations.

Step 3: Read choice

Step 4: Enqueue operation

Step 4.1: Read the Element for Enqueue

Step 4.2: if front == -1 && rear == -1, then

set Front = 0, rear = 0

queue[rear] = item

Step 4.3: Else if Front = rear + 1

Print queue is overflow

Step 4.4: Else set rear = rear + 1 % Max size

queue[rear] = item

Step 5: Dequeue operation

Step 5.1: check front = rear = -1, then Print queue is underflow

Step 5.2: Else if Front == rear, print Element is deleted, set Front = rear = -1

Step 5.3: Else Element dequeued

From ... Max size

Step 6: display

Step 6.1: If Front = -1 and rear = -1, Print queue is empty

Step 6.2: Else while i <= rear, Repeat

Print queue[i] and set i = i+1 % MaxSize

Step 7: Search

Step 7.1: Read the Element to be searched,

Step 7.2: If item == queue[i], then point Element found and i = i+1.

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

Step 8: End.

Output

```
Menu
1. Insert
2. Delete
3. Display
4. search
Enter your choice : 1
Enter the element: 10
Menu
Enter your choice : 1
Enter the Element : 30
Menu
Enter your choice: 3
10
30
Menu
Enter your choice : 2
10 was deleted.
```

Program no: 4 - Doubly linked list

Step 1: start

Step 2: declare structure and variables

Step 3: declare functions for operations

Step 4: Define function to create a node

Step 5: set memory allocated to node = temp

temp→ PREV = temp→ next = null

Step 6: Read the value to b inserted in node.

Set temp -> data = value and count = count+1

Step 7: Read the choice from the user

Step 8: Insertion

8.1: If head == null, call the function create node

8.2: Set head = temp and temp1 = head.

8.3: Else call the function to create node,

temp → next = head, head → prev = temp, ~~temp = b~~

head = temp.

step 9: Insertion at the end.

9.1: If head == null, call the function to create node.

Set temp = head, temp→ next = temp

temp→ prev = temp1. & temp1 = temp.

Step 10: Insertion at specified position

10.1: Read the position

Set temp2 = head

10.2: Check if pos < 1 or pos > Count+1, then

Print out of range.

10.3: check if head==null & pos = 1, print

Cannot insert other than 1st location

10.4: Check if head==null and pos = 1, then

call the function to create new node,

10.5: while i < pos, then set temp2 = temp2->next

then i = i+1

10.6: Create newnode, and set,

temp -> prev = temp2

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

temp2 -> next = temp.

Step 11: Deletion operation

11.1: Read the location where the node to be delete.

Set temp2 = head

11.2: check if pos < 1 or pos >= Count+1,

the print out of range.

11.3. Check if head == null then print

list is Empty

11.4 : while i<pos , then,

temp2 = temp2 -> next and i = i++

11.5 : check if i == 1,

temp2 -> next == null,

Print node deleted, Free(temp)

set temp2 = head = null.

11.6 : check if temp2 -> next == null, then temp2 -> prev -> next = null, Free(temp)

Print node deleted.

11.7 : temp2 -> next -> prev = temp2 -> prev, then check if i != 1, then temp2 -> prev -> next = temp2 -> next.

11.8 : check if i = 1, then

head = ~~head~~ temp2 -> next, print node deleted, free temp2 and decrement count by 1.

Step 12

~~8~~ 12 : Display operation

12.1 : set temp2 = n

12.2 : check if temp2 = null, then print list is Empty

12.3 : while temp2 -> next != null. then print temp2 -> n then temp2 = temp2 -> next.

step : 13 : Search operation

13.1 : Declare necessary variables

13.2 : set temp = head

13.3 : check if temp2 == null, then print the list is empty

13.4 : Read the value to be searched

13.5 : while temp2 != null check, if temp2 -> n == data, the print Element found and count = count+1

13.6 : Else set temp2 = temp2 -> next and count = count+1

13.7 : print Element found in the list

step 14 : End programs.

Output

Menu

1) Insert at beginning

2. Insert at end

3. insert at position

4) Delete

5. Display

6.) Search

7. Exit

Enter choice : 1

Enter the value to insert : 5

Menu

Enter choice

Enter the value : 10

Menu

Enter choice : 2

Enter the value : 2

Menu

Enter choice : 3

Enter the ~~value~~ position : 2

Enter value to insert : 13

Menu

Enter choice : 5

Element in linked list : 10 13 5 2

Menu

Enter choice : 4

Enter the position to delete : 2

Node deleted

Menu

Enter choice : 6

Enter the value to search : 10

Element found in 1 position

Menu.

Step 1: start

Step 2: Declare necessary variables

Step 3: Read the choice from user.

Step 4: Union operation

Step 4.1: Read the cardinality of the 2 sets

4.2: Check if m != n the print cannot perform union

4.3: Else Read the Elements in both sets,

4.4: Repeat 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

Step 5: Read the choice from user.

Insertion

Step 1: Read the cardinality of two sets

Step 2: Check if m != n print Cannot perform union

Step 3: Read the Elements in two sets,

Step 4: Repeat the step 4.1 to 4.2 until i < m

4.1: C[i] = A[] & B[]

4.2 = Print C[i]

4.3 : increment i by 1

step 5 : difference operation

step 5.1 : Read the cardinality of the set.

5.2 : check if m!=n then print cannot perform union

5.3 : else read the Elements of both sets

5.4 : repeat step 5.5 to 5.8 until i<n

5.5 : if A[i] == 0 then C[i] == 0

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

5.7 : Else c[i] = 1

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

1. union
2. intersection
3. subtraction
4. Exit

Enter your choice: 1

Enter the size of set 1

3

Enter the Elements

1
2
3

Enter the size of set 2

2

Enter the Elements

2,
3

union 1 2 3

Menu

Enter choice: 2

Enter the size of set 1

3

Enter the Elements:

1
2
3
Enter size of set 2:
2
Enter Elements
3
2
difference : 1

Program NO: 6 : Binary Search Tree,

Step 1: start

step 2: declare structure, structure pointer for insertion, deletion and search operations and declare function for inorder traversal

step 3: Declare a pointer as root and other required variables

step 4: Read the choice from user,

step 5: choice = Insertion

5.1 : Read the value to be inserted

5.2 : Pass the value to the insert pointer and also the root pointer

5.3 : Check if ! root then allocate mem for the root.

5.4 : Set root → info = value, then,
root → left = root → right = NULL;

5.5 : Check if root → info > x. then call insert fn to insert left of the root

5.6 : Check info root → info < x. then insert at the right of the tree.

Step 6 : deletion.

Step 6:1 : Read the Element to delete bsmstree,

6.2 : Check if not ptr. then print node not found

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

6.4 : Else if ptr → info > x then call delete pointer by passing the left pointer and the item

6.5 : check ptr → info == item

ptr → left == ptr → right =

Free (ptr)

6.6 : Else if ptr → left == null, set,

P1. Ptr → right and free [ptr]

6.7 : Else if ptr → right == null, set P1.

ptr → left and free (ptr)

6.8 : Else set P1 = ptr → right and P2 = ptr → right

6.9 : while P1 → left ≠ null,

set P1 → left = ptr → left and free (ptr)

Step 7 : Search operation

7.1 : Read the Element to be searched

7.2 : If item > ptr → info then

ptr = ptr → right

7.3 : Else if item < ptr → info, then

ptr = ptr → left

7.4 : if ptr then, print Element is found,

7.5 : Else print Element not found.

Step 8 : Traversal

8.1 : If root != null, recursively call function by passing root → left

8.2 : print root → info

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

Step 9 : End program.

Output

1. insert
2. delete
3. inorder traversal
4. search
5. Exit

Enter choice : 1

Enter element : 20

Root is 20

Inorder traversal = 20

Menu

Enter your choice : 1

Enter Element : 30

Inorder traversal 10. 30

Menu

Enter your choice : 4

Enter the Element to search : 30

Element 30 was found in tree.

Program no: 7. Disjoint set operations

step 1 : start

step 2 : Declare structure and related variable

step 3: declare function Makeset

step 3.1 : Repeat 3.2 to 3.4 until i<n

3.2 : disparent [i] is set to i

3.3 : set dis. rank [i] is equal to

3.4 : increment i by 1

step 4 : Declare function Display

4.1 : Repeat 4.2 to 4.3 until i<n

4.2 : print dis.parent [i]

4.3 : increment i by 1

4.4 : Repeat 4.5 and 4.6 until i<n

4.5 : print dis.rank [i]

4.6 : increment i by 1

step 5: Declare the function find and pass x to the function

5.1 : if dis.parent [x] = x,

set return value to disparent (x)

5.2 : return disparent [x]

function union and pass a variable x and y

6.1: set xset to find(x)

6.2: set yset to find(y)

6.3: If y set = xset then return

6.4: If disrank[xset], < disrank[yset] then

6.5: set yset = disparent[yset]

6.6: set -1 to disrank[xset]

6.7: Else if check disrank[xset] > disrank[yset]

6.8: set xset to disparent[yset]

6.9: Set -1 to disrank[yset]

6.10: Else disparent[xset] = xset

6.11: set disrank[xset]+1 to disrank[xset]

6.12: set -1 to disparent (rank) [yset]

Step 7: Read the no. of elements

Step 8: Call Function Makeset

Step 9: Read the Choice.

Step 10: If union operation call function union

Step 11: If find operation call function Find and read the Elements to check they are Connected.

11.1 : If Find (x) = Find (y), then print they are connected

11.2 : else print not connected

step 12 : For display option call the display Function

step 13 : End.

Output

No. of Elements 4

Menu

1. Union
2. Find
3. Display

Enter choice : 1

Enter the Elements to perform union

3

4

Do you want to Continue (1/0)

1

Menu

Enter your choice : 1

Enter the Elements to Perform union

5

6

Do you want to Continue (1/0)

1

Menu

Enter choice : 3

Parent array

8 1 2 3

Rank array

-1 0 0 1