Page Subhash

Date

DS LAB RECORD

Submitted By →

AKSHAY MURALI

MCA BATCH 'A'

ROLL NO - 07

Submitted to →

MERIN MISS

PROGRAM - 1 ALGORITHM

Page Subhash
Date

AIM ⇒ Program to perform merge operation in two sorted.

Step 1 → Start
Step 2 → Declare the Variables
Step 3 → Read the size of first array.
Step 4 → Read the 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 → Repeate 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 → Repeate Step 11 while i<m
Step 11 → c[k++] = a[i++]
Step 12 → Repeate 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

```
Size of first array
2
Enter 2 value in sorted order
3
6
Size of second array
4
Enter 4 value in sorted order
4
5
8
9
A List is :
   3   6
B List is :
   4   5   8   9
C List is :
   3   4   5   6   8   9
```

PROGRAM - 2 ALGORITHM

Page Subhash
Date

AIM → Program to perform operation in Singly linked Stack.

Step 1 → Start
Step 2 → Declare the node and the required variables.
Step 3 → Declare the functions for push pop display and search.
Step 4 → Read the Choice from the user to push pop display or search an element.
Step 5 → If the user Choose to 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.
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 new Node → 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 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 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.

Date

OUTPUT →

Stack using linked list

Menu.

1. Push
2. Pop
3. Display
4. Search
5. Exit

Enter your choice : 1

Enter the value to be inserted : 2

Insertion is successful !!!

Menu

1. Push
2. Pop
3. Display
4. Search
5. Exit

Enter your choice : 1

Enter the value to be inserted : 34

Insertion is successful.

Menu

1. Push
2. Pop
3. Display
4. Search
5. Exit

Enter your choice : 1

Enter the value to be inserted 45

Insertion is successful !!!

Date

OUTPUT →

1. Push
2. Pop
3. Display
4. Search
5. Exit

Enter your Choice : 2

Deleted element : 45

Menu

1. Push
2. Pop
3. Display
4. Search
5. Exit

Enter your Choice : 3

34 → 2 → NULL

Menu

1. Push
2. Pop
3. Display
4. Search
5. Exit

Enter your Choice : 4

Enter item which is to be searched : 2

Item found at location 2.

PROGRAM No - 3 ALGORITHM

Page Subhash
Date

AIM → Program to perform operations in Circular Queue

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 & 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 three 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 the 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] & 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 & its position and increment c by 1.

Step 8.3 → Check if c == 0 then print item not found.

Step 9 → End.

OUTPUT →

Press 1 → Insert an element.
Press 2 → Delete an element
Press 3 → Display the element
Press 4 → Search the element.
Enter your Choice 1
Enter the element which is to be inserted

Press 1 → Insert an element
Press 2 → Delete an element
Press 3 → Display the element
Press 4 → Search the element
Enter your Choice 1
Enter the element which is to be inserted

Press 1 → Insert an element
Press 2 → Delete an element
Press 3 → Diplay the element
Press 4 → Search the element
Enter your Choice 1
Enter the element which is to be inserted 3.

Press 1 : Insert an element
Press 2 : Delete an element
Press 3 : Display the element
Press 4 : Search the element
Enter your Choice 1
Enter the element which is to be inserted 4.

Subhash

Date

OUTPUT →

Press 1 : Insert an element
Press 2 : Delete an elen
Press 3 : Display the element
Press 4 : Search the element
Enter your Choice 2
The dequed element is 1.

Press 1 : Insert an element
Press 2 : Delete an element
Press 3 : Display the element
Press 4 : Search the element
Enter your Choice 3
Elements in a queue are : 2, 3, 4.

Press 1 : Insert an element
Press 2 : Delete an element
Press 3 : Display the element
Press 4 : Search the element.

Enter your Choice 4
Enter the element which is to be searched 4.
Item found at location 3.

PROGRAM NO-1 ALGORITHM

Page Subhash
Date

AIM → Program to perform operation in doubly linked list.

Step 1 → Start
Step 2 → Declare a structure & related variable
Step 3 → Declare functions to create a node, insert a node in the beginning at the end and given position, display the list & search an element in the list.
Step 4 → Define function to create a node, declare the required variable
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 ~~wa~~ 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 list.
Step 6 → If the user choose to perform Insertion operation at begining then call the function to perform the insertion.
Step 6.1 → Set head = temp & temp1 - head
Step 6.2 → Set head = temp & temp = head.
Step 6.3 → Else call the function to creat a node perform Step 4 to 4.3 then Set temp → next = head, Set head → prev = temp & 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 = temp 1

Step 7.2 → Else call the function to create a newnode then set temp 1 → next = temp, temp → prev = temp 1 and temp 1 = 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 head to the inserted, set temp 2 = 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 newnode. then set temp = head and head = temp 1

Step 8.6 → while i < pos then increment i by 1.

Step 8.7 → Call the function to create a new node & then set temp → prev = temp 2. temp → next = temp 2 → next

temp 2 → next → prev = temp
temp 2 → next = temp
Step 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 num any variables.
Step 9.2 → Read the position where node need to be deleted set temp 2 = 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 temp 2 = temp 2 → next and increment i by 1
Step 9.6 → Check if i == 1 then Check if temp 2 → next == null then print node deleted. free (temp 2). Set temp 2 = head = null.
Step 9.7 → Check if temp 2 → next == null then temp 2 → prev → next = null then free (temp 2). then print node delete.
Step 9.8 → temp 2 → next → prev = temp 2 → prev. then Click if i != 1 then temp 2 → prev → next = temp 2 → next.
Step 9.9 → Check if i == 1 then head = temp 2 → next then print node deleted then free temp 2 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 temp 2 = h

Step 10.2 → Check if temp 2 = null then print List is empty.

Step 10.3 → while temp 2 → next != null then print temp 2 → n then temp 2 = temp 2 → next.

Step 11 → If the user Choose to perform the Search operation then call function to perform Search operation.

Step 11.1 → Declare neccessary variables.

Step 11.2 - Step temp 2 = head.

Step 11.3 - Check if temp 2 == null then print the list is empty.

Step 11.4 → Read the value to be searched.

Step 11.5 → while temp 2 != null the check if temp 2 → n == data then print element found at position Count + 1.

Step 11.6 → Else, Set temp 2 . temp 2 → next and increment Count by 1

Step 11.7 → Print element not found in the list.

OUTPUT →

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

Enter Choice : 1
Enter value to node : 1
Enter Choice : 1
Enter value to node 2
Enter Choice : 1
Enter value to node 3
Enter Choice : 2
Enter Value to node : 7
Enter Choic : 5

Linked List elements from beging
: 3 2 1 7

PROGRAM NO - 5

Page Subhash
Date

AIM → Program to perform set operation.

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 2 sets
Step 4.2 → Check if m1 = 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 m1 = 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 $m_1 = n$ then print Cannot perform set difference operation.

Step 6.3 → Else read the elements in both sets.

Step 6.4 → Repeat the 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 → 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.

OUTPUT →

Input Choice to perform:

1. Union 2. Intersection 3. Difference 4. Exit

Enter the Cardinality of first set: 3
Enter the Cardinality of 2nd set: 3
Enter the element to first set: (0/1)
0
1
Enter the element of 2nd set (0/1)0
1
0
Element of set 1 union set 2: 111

PROGRAM NO - 6

Page Subhash
Date

AIM → Program to perform tree operations.

Step 1 → Start
Step 2 → Declare a structure and structure pointers for insertion deletion and search operation and also declare a function for in order traversal.
Step 3 → Declare a pointer as root and also the required variables.
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 5.1 → Ran the value be the insert pointer and 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 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 < n the call delete pointer by paring the right pointers & the item.
Step 6.3 → Else if ptr → info > n then call delete pointer by passing the left pointer & 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 P1
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 & 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 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 & pass the root pointer.
Step 8.1 → If the root not equal 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.

Enter Choice : 1
Enter new element : 50
root is 50.
Inorder traversal of binary tree is
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 : 25
root is 50,
Inorder traversal of binary tree is :

PROGRAM No - 7

Page Subhash
Date

AIM → Program to perform operation on disjoined set.

Step 1 → Start
Step 2 → Declare the Structure & related structure variable.
Step 3 → Declare a function makeset()
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 & 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 & 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 & pass x to the function.
Step 5.1 → Check if dis.parent[x]!=x then Set the return value to dis.parent[x]
Step 5.2 → return dis parent[x]
Step 6 → Declare a function Union & pass two variables x & 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 [xset] < dis.rank [y set] then.
Step 6.5 → Set yset = dis.parent[yset]
Step 6.6 → Set -1 to dis rank [x set]
Step 6.7 → Else if check dis rank [xset] > 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 [x set].
Step 6.12 → Set -1 to dis rank [y set]
Step 7 → Read the number of elements.
Step 8 → Call the function make set.
Step 9 → Read the choice from user to perform union find & display operation.
Step 10 → If the user Choose 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 set.
Step 13 → End.

OUT PUT →

How many element ? 4

Menu

1. Union
2. Find
3. Display

Enter Choice

1.

Enter the element to perform union

3

4

Do you want to Continue (1/0)

1.