Date:

Program no: 1

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

Algorithm: Merging (array1, array2, merge, m, n)

let array1 and array2 be sorted arrays with m and n elements, respectively. This algorithm merges array1 and array2 into an array 'merge' with m+n elements.

1. set i=0, j=0, k=0
2. Repeat while i<m and j<n
   if array1[i] < array2[j]
   set merge[k] = array1[i]
   set k = k+1 and i = i+1
   else
   set merge[k] = array2[j]
   set k = k+1 and j = j+1
   [end if]
   [end loop]
3. Repeat while i<m, then:
   set merge[k] = array1[i]
   set i = i+1
   set k = k+1

[end loop)

4. Repeat while $j \leq n$, then

set merge[k] = array2[j]

set $j = j+1$ and $k = k+1$

[end loop]

5. stop

Output

```
Enter the size of first array:
3
Enter the elements
1
3
4
Enter the size of second array:
4
Enter the elements
2
5
7
8
Merged array is
1
2
3
4
5
7
8
```

Date:

Program no : 2

Aim: Singly linked stack - push, pop, linear search

Algorithm :-

1. start
2. If user select push operation then
3. Create a newnode with the given data
4. If top == NULL then:
5. set top = newnode
   set newnode → next = NULL

   else
   set newnode → next = top
   top = newnode
6. If user select pop operation then
7. If top == NULL then
   display "stack is empty"

   else
   set temp = top (create a temporary node and set it to top)
   display temp → data
8. set top = temp → next
9. free temp (delete the temporary node)

10. If user select search operation then

11. Declare a pointer variable temp and the variable key that holds the value to be searched

12. set temp = top
    set flag = 0

13. Repeat while temp != NULL
    If temp → data == key then
    display "element found"
    set flag = 1
    [end if]
    goto step 14
    else
    set temp = temp → next
    [end while]

14. If flag == 0 then
    display "element not found"
    [end if]

15. If user select display operation then

16. declare a pointer node ptr
    set nodeptr = top

17. If nodeptr == NULL then

display " stack is empty "

[end if]

18. while node ptr != NULL then

print node ptr → data

set nodeptr = nodeptr → next

if nodeptr != NULL then:

Print " —→ "

[end if]

[end while]

19. stop

Output

- top of stack : 3

stack as linked list

3 → 2 → 1

Enter the element to search

2

element found

poped element is 3

Poped element is 2

top of stack 1

stack as linked list

1

Date:

Program no : 3

Aim; Circular Queue - Add, Delete, Search

Algorithm :

1. start
3. If user select the insertion operation then
3. declare a variable item with given value
4. If front == 0 && rear == size -1 || front == rear +1 then :

   Display "Queue Overflow"

   [end if]
5. If front == -1 then :

   set front = 0

   set rear = 0

   [end if]
6. If rear == size -1

   set rear = 0

   else

   set rear = rear +1

   [end if]
7. set cq[rear] = item

8. If user select deletion operation then

9 If front == -1 then

display "Queue underflow"

[end if]

10. If front == rear then

set front = -1

set rear = -1

[end if]

11. If front == size-1 then

set front = 0

else

set front = front + 1

[end if]

13. If user select the display operation then

13 set front_pos = front

set rear_pos = rear.

14. If front == -1 then

Display "Queue is empty".

[end if]

15. If front_pos <= rear_pos then

Repeat while front_pos <= rear_pos then

print cq [front_pos]

set front_pos = front_pos + 1

[end while]

else

Repeat while front_pos <= size - 1

print cq [front_pos]

set front_pos = front_pos + 1

[end while]

16. set front_pos = 0

17. Repeat while front_pos <= rear_pos then:

display cq[front_pos]

set front_pos = front_pos + 1

[end while]

[end if]

18. If user select search operation then

19 Declare a variable search with value to be searched.

20. declare a temporary variable temp then

set temp = search

21. Set i = front

22. Repeat for i<=rear then

    If fi == cq[i] then

    Print i+1

    Set j = j+1

    [end if]

    If j == 0 then

        Display "Item not found"

    [end If]

    Set i = i+1

    (end of for loop]

23. Stop

Output

```
1. insert
2. Search
3 Delete
4. Display
5. Quit
Enter your choice : 1
Input the element : 6
1. insert
2. Search
3. Delete
4 Display
5. Quit
Enter your choice : 1
Input the element : 5
1. insert
2. Search
3. Delete
4 Display
5. Quit
Enter your choice : 2
Item found at location 2
```

1. insert
2. Search
3. Delete
4. Display
5. Quit

Enter your choice : 3

Element deleted from queue is : 6

1. insert
3. search
3. Delete
4. Display
5. Quit

Enter your choice : 4

Queue elements :

5

1. insert
2. search
3. Delete
4. Display
5. Quit

Enter your choice : 5

Date:

Program no : 4

Aim : Set Data structure and set operations (Union, Intersection, and Difference) using bit string.

Algorithm :

1. Start
2. If user select the union operation then
3. declare two array set1[i] and set2[i], Declare 2 variable n_1, n_2 for holding the size of 2 arrays.
4. Read elements into the arrays set1[i] and set2[i]
5. If $n_1 == n_2$ then:
6. Set i=0
7. Repeat for $i < n_2$ then
   set set3[i] = set1[i] || set3[i]
8. Set i = i+1
   [end for loop]
9. Set i=0
10. Repeat for i<2 then

print set3[i]

11. Set i=i+1

[end for loop]

[end if]

else

print "size are not equal"

Exit

13. If user select insertion operation then

13. declare two array set1[i] and set2[i] with size n_1, n_2 respectively and read elements to the arrays.

14. If $n_1 == n_2$ then

15 Set i=0

16. Repeat for $i < n_2$ then :

17 Set set3[i] = set1[i] ff set2[i]

18 Set i = i+1

[end for loop]

19. Set i=0

20. Repeat for $i < n_2$ then :

print Set3 [i]

21. set i = i+1

[end for Loop]

(end if)

else

print ("size are not equal"

Exit

22. If user select the subtraction then

23. declare 2 array set1[i] and set2(i) with n_1, n_2, size respectively and input the elements to the array.

24. If $n_1 == n_2$ then

25. set i=0

26. Repeat for $i < n_2$ then

set Set3 [i] = set1[i] && Set2 [i]

27. i = i+1

[end for loop]

28. set i=0

29. Repeat for $i < n_2$ then

print Set3 [i]

30. Set i = i+1

[end for loop]

[end if]

else

"print size are not equal"

31. Exit.

Output

Press 1 for union

Press 2 for intersection

Press 3 for Subtraction

enter your choice !1

Enter the size of set-1

10

enter the element- of set-1

1
0
0
1
0
1
1
0
0
1

enter the size of set-2

10

Enter the elements of set2

0
0
1
1
0
0
1
0
0

10 1 1 1 1 1 0 1

want to continue : press 1 for union

press 2 for intersection

Press 3 for substraction

Enter your choice 3

enter the size of set1

10

Enter the element of set1

1
0
0
1
0
1
0
0
1

Enter the size of set2

10

Enter the element of set2

0
0
1
1
0
0
1
0
0
0

0 0 0 1 0 0 0 0 0 0

Want to continue. press 1 for union

Press 2 for intersection

Press 3 for substraction

Enter your choice : 3

enter the size of set1

10

Enter the element of set1

1
0
0
1
0
1
1
0
0
1

enter the size of set2

10

enter the elements of set2

0
0
1
1
1
0
0
1
0
0
1 0 0 0 0 1 1 0 0 1

Date:

Program no: 5

Aim: Binary Search trees - Insertion, Deletion, Search

Algorithm:

1. Start
2. If user select the insertion operation. Then
3. Create a new BST node and assign values to it.
4. Create tree (node, data) // call the creat-e tree function with root value and the data entered by user.
5. If root == NULL then
6. Declare a temporary variable temp.

   Set temp → data = data

   Set temp → left → right = NULL;

   return the new node temp to the calling fn.

   [end if]
7. If data < (node → data)
8. Call the Createnode function with node → left

and assign the return value in node → left.

node → left = create tree (node → left, data)

[end if]

9. If data > node → data

10. Call the create tree function with node → right and assign the return value in node → right

node → right = create node (node → right data)

[end if

11. return the original root pointer 'node' to the calling function.

12. If the user select the search element operation then.

13. Search (node, data) // call the search function with root value and the element to be searched.

14. If node == NULL

print "Element not found"

[end if]

15. If data < node → data then :

Call the search fn with node → left and assign the return value in node → left.

node → lef = Search (node → left, data)

[end if]

16. If data > node → data then.

Call search fn with node → right and assign the return value in node → right.

node → right = search (node → right, data)

[end IF]

else

print "Element ~~found~~ is" node → data

17. return the original root pointer 'node to the calling fn.

18. If the user select the deletion operation then

19. del (node, data) // call the del fn with root value and the element to be deleted

20. declare a temporary variable temp.

21. If node == NULL then:

print "element not found"

[end if]

22. If data < node -> data then:

call the del fn with node -> left and

assign the return value to node -> left.

node -> left = del (node -> left . data)

[end if]

23. If data > node -> data then

call the del fn with node -> right

and assign the return value to node -> right

[end if)

24. Else:

// delete this node and replace with

either minimum element in the right subtree

or maximum element in the left subtree

25. If node -> right && node -> left

// replace with minimum element in the

right sub tree.

26. Call findmin function with node->right then return value assign in temp
go to step 32

set temp = findmin(node->right)
set node->data = temp->data
// replaced it with some other node.

27. Call fn del with value node->right, temp->data and return value assigned in node->right
else,

28. set temp = node
// if there is only one or zero children then we can directly remove it from the tree and connect its parent to its child.

29. If node->left == NULL then:
set node = node->right
Else:

30. If node->right == NULL then
set node = node->left

31. free(temp)

[end if]

[end if]

[end if]

32. findmin (node)

33. If node == NULL then

return null

goto step 26

[end if]

34 If node → left then

call the fn findmin with value (node→left)

Then return the value to calling function

return findmin (node → left)

else

return node

Goto step 26

(end If)

35. If the user select the display option

then

36. Inorder (node)

call the inorder fn with root value.

37. If root != NULL then

inorder (node -> left

call the fn inorder with value node -> left.

38. print node -> data

inorder (node -> right)

call the fn inorder with value node -> right

(end if)

39. Stop.

Output

1. Insertion in Binary search tree.
2. Search element in BST
3. Delete element in BST
4. Inorder
5. Exit

enter your choice : 1

enter N value : 4

Enter the value to create BST

1
2
5
6

1. insertion in Binary search tree
2. Search element in BST
3. Delete element in BST
4. Inorder
5. Exit

enter your choice 2

enter the value to search : 5

element founnt is : 5

1. insertion in Binary search tree.

2. ~~Delete~~ search element in BST

3. Delete element in BST

4. Inorder

5. Exit

enter your choice : 3

enter the element to delete : 5

1. Insert in binary search tree

2. search element in BST

3. Delete element in BST

4. Inorder

5. Exit

enter your choice : 5

Date

Program no : 6

Aim: Doubly linked list - insertion, Deletion, search.

Algorithm:

1. start
2. If user select the insert operation at beginning then
3. If head == NULL then
4. perform step 5 to 6
   (call the fn create())
5. set found = temp
   temp1 = head.
6. Else
7. perform step 56 to 59
   (call the fn create)
8. set temp → next = head
   set head → prev = temp
   set head = temp
   [end if]

If user choose the operation insert node at end then

10. If head == NULL then.

11 perform 58 to 61

12. set head = temp

set temp1 = head

13. Else

14. perform step 58 to 61

15. set temp1 → next = temp

set temp → prev = temp1

set temp1 = temp

(end if)

16. If user choose insert at any position then

17 Read the position and store it in to the variable pos

18. set temp2 = head

19. If pos < 1 || pos >= count+1 then:

Display "position out of range to insert"

Exit

[end if]

20. If head == NULL && pos != 1 Then

Display "empty list cannot insert other than 1st position"

exit

[end if]

21. If head == NULL && pos == 1 then:

22. perform step 58 to 64

(call fn create())

23. set head = temp

set temp1 = head

[end if]

exit.

24. Else

repeat.

25. while i < pos Then

26 Set temp2 = temp2 -> next

set i = i+1

[end while]

27 : Perform step 56 to 59

28 : set temp → prev = temp1

set temp → next = temp2 → next

set temp2 → next → prev = temp

set temp2 → next = temp

29. If user choose the operation deletion then

30 : Read the position & it store in to the variable pos.

31 . set . temp2 = head

32 . If pos < 1 || pos >= count + 1 then:

Display "position out of range to delete"

[end if]

exit

33 . If head == NULL then

Display "empty list no elements to delete"

(end if]

34. else

35. repeat while IZ pos.

set temp2 = temp2 → next

set i = i+1

[end of while]

36. If i == 1 then

if temp2 → next == NULL then :

Display "node deleted from first"

37. free temp2

set temp2 = head = NULL

Exit [end if]

38. If temp2 → next = NULL then

set temp2 → prev → next = NULL

free temp2

Display "node deleted from last"

[end if]

Exit

39. Set temp2 → next → prev = temp2 → prev

40. if i != 1 then

temp2 → prev → next = temp2 → next

[end if]

41. If i == 1 then

set head = temp2 → next

Display "node deleted"

42. free temp2

[end if]

43. Set Count = Count - 1

44. If user select display operation then

45. set temp2 = head

46. If temp2 == null

display "List empty to display"

[end if]

exit

47. while temp2 → next1 = null then

print temp2 → n

Set temp2 = temp2 → next

(end while)

48. print temp 2 -> n

49. If user select search operation then

50. set temp 2 = head

51. If temp 2 == NULL then

Display "list empty to search for data"

[end if]

exit

52. Read the value to be search and store it in to the variable data

53. While temp 2 != NULL then

54. If temp 2 -> n == data then

print count+1

Exit

55. Else

set temp 2 = temp 2 -> next

set count = count+1

[end if]

[end while]

56. Display " not found "

57. Exit

58. // when call create function, set temp → prev
= NULL // create()

59. set temp → next = NULL

Display " enter value to node "

60. Read data and assign it into

temp → n

temp → n = data

61. count = count + 1

62. Exit

Output

1 - Insert at beginning
2 - Insert at end
3 - Insert at position i
4 - Delete at i
5 - Display from begining
6 - search for element
7 - Exit

Enter choice : 1
Enter value to node : 3
Enter choice : 1
Enter value to node : 4
Enter choice : 5
Linked list elements from begining : 4 3
Enter choice : 5
Enter value to node : 6
Enter choice : 5
Linked list elements from begining : 4 3 6

Enter choice : 3

Enter position to be inserted : 2

Enter value to node : 1

Enter choice : 5

Linked list elements from begining : 4 1 3 6

Enter choice : 4

Enter position to be deleted : 1

Node deleted

Enter choice : 5

Linked list elements from begning : 1 3 6

Enter choice : 4

Enter position to be deleted : 3

Node deleted from list

Enter choice : 5

Linked list elements from begining : 13

Enter choice : 6

Enter value to search : 3

Data found in 2 position.

Enter choice : 6

Enter value to search : 7

Error : 7 not found in list

Enter choice :

Date:

Program no: 7

Aim: Disjoint-sets and the associated operations (create, union, find)

Algorithm:

1. start
3. Read the no of elements from user and store it in to the dis.n
3. Call fn makeset-() then
4. set i=0
5. Repeat for i<dis.n then
   set dis.parent[i] = i
   set dis.rank[i] = 0
   set i = i+1
   [end for loop]
6. User select the union operation then
7. Read the elements to perform union and store into x and y respectively
8. // perform find operation with x and y

store result into xset and yset perform step 23

9. If xset == yset then
   [end if]
10. If dis.rank[xset] < dis.rank[yset] then
   set dis.parent[xset] = yset
   set dis.rank[xset) = -1
   [end if]
11. else if dis.rank[xset] > dis.rank[yset] then
   set dis.parent[yset] = xset
   set dis.rank[yset] = -1
   [end if]
12. else
13. set dis.parent[yset] = xset
   set dis.rank[xset] = dis..rank[xset] + 1
   set dis.rank[yset] = -1

14. If user choose find operation then

15 Read the elements to check and store the value into the variables x and y respectively.

16. If find x == find y then:

Display "Connected components"

17. else

Display "not connected components"

18. If user select the display operation then.

19. set i = 0

20. Repeat for i < dis.n then

print dis.parent[i]

set i = i+1

[end for loop]

21. set i=0

24. Repeat for i < dis.n then

print dis.rank[i]

set i = i+1

[end for loop]

23. If dis.parent[x] != x

Then

set dis.parent[x] = find(dis.parent[x])

return dis.parent[x]

24. Stop

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 wish to continue ? (1/0)

1

--menu--

1. union

2. find

3. display

Enter choice

2

Enter the element to check if connected

Components.

2

3

Connected Components

do you wish to continue ? (1/0)

1

--menu--

1. union

2. find

3. display

Enter choice

3

Connected components.

do you wish to continue ? (1/0)

1

-- menu --

1. union
2. find
3. display

Enter choice.

2

Enter the elements check if connected components

1

3

not connected components.

do you wish to continue.