VISVESVARAYATECHNOLOGICALUNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT
On

DATA STRUCTURES(23CS3PCDST)

Submitted by

Priyam Thakur

(1BM23CS252)

inpartialfulfillmentfortheawardofthedegreeof BACHELOR OF ENGINEERING
in
COMPUTERSCIENCEANDENGINEERING

B.M.S.COLLEGEOFENGINEERING
(AutonomousInstitutionunderVTU)
BENGALURU-560019
September2024-January2025

B. M. S. College of Engineering,
BullTempleRoad,Bangalore560019
(AffiliatedToVisvesvarayaTechnologicalUniversity,Belgaum) Department
of Computer Science and Engineering

This is to certify that the Lab work entitled**“DATASTRUCTURES”**carriedoutbyNAME **(USN)**, who is bonafide student of **B. M. S. College of Engineering**. It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering**oftheVisvesvarayaTechnologicalUniversity,Belgaumduringtheyear 2024-25. The Lab report has been approved as it satisfies the academic requirements in respect of Data structures Lab - **(23CS3PCDST)**work prescribed for the said degree.

Prof.LakshmiNeelimaM
Assistant Professor
Department of CSE
BMSCE, Bengaluru

Dr.KavithaSooda
Professorand Head
Department of CSE
BMSCE, Bengaluru

Index Sheet

Course outcomes:

CO1	Apply the concept of linear and nonlinear data structures.
CO2	Analyze data structure operations for a given problem
CO3	Designanddevelopsolutionsusingtheoperationsoflinearandnonlineardata structure for a given specification.
CO4	Conductpracticalexperimentsfordemonstratingtheoperationsofdifferent data structures.

1. Write a program to simulate the working of stack using an array with the following:

a) Push
b) Pop
c) Display

The program should print appropriate messages for stack overflow, stack underflow

```
#include<stdio.h>

#define MAX 100

int stack[MAX];

int top = -1;

voidpush(intvalue){

    if (top == MAX - 1) {

        printf("StackOverflow!Unabletopush%d.\n",value);

    }else{

        top++;

        stack[top]=value;

        printf("Pushed%dintothestack.\n",value);

    }

}

void pop() {

    if (top == -1) {

        printf("StackUnderflow!Unabletopopanelement.\n");

    }else{

        printf("Popped%dfromthestack.\n",stack[top]);

        top—

    }

}

void display()

    {if(top==-1){
        printf("Stackis empty.\n");
    }else{
        printf("Stackelementsare:"); for
        (int i = top; i >= 0; i--) {
            printf("%d",stack[i]);
        }
        printf("\n");
    }
}

intmain() {
    intchoice,value;

    do {
        printf("\nStackOperations:\n");
        printf("1. Push\n");
        printf("2. Pop\n");
        printf("3.Display\n");
        printf("4. Exit\n");
        printf("Enteryourchoice:");
        scanf("%d", &choice);

        switch(choice){
            case 1:
                printf("Enterthevaluetopush:");
                scanf("%d", &value);push(value);
                break;
            case 2:
                pop();
                break;
            case 3:
                display();
                break;
            case 4:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalidchoice!Pleasetry again.\n");
        }
    }while(choice!= 4);

    return0;
}OUTPUT
```

:

```
1. Push
2. Pop
3. Display
4. Exit
Enteryourchoice:1
Enterthevaluetopush:10 Enter
your choice: 1
Enterthevaluetopush:20 Enter
your choice: 3
Enteryourchoice:2
Enteryourchoice:3
```

```
}
```

```
Enter your choice: 4

Pushed 10 into the stack.

Pushed 20 into the stack.

Stack elements are: 20 10

Popped20fromthestack.

Stack elements are: 10

Exiting...
```

2. WAPtoconvertagivenvalidparenthesizedinfixarithmeticexpressiontopostfixexpression.The expression consists of single character operands and the binary operators + (plus), - (minus), * (multiply) and / (divide)

```
#include <stdio.h>

#include<stdlib.h>

#include<ctype.h>

#include<string.h>

#define MAX 100

char stack[MAX];

int top = -1;

voidpush(charc){

    if (top == MAX - 1) {

        printf("Stack Overflow\n");

        exit(1);

    }

    stack[++top]=c;

}


charpop(){

    if (top == -1) {
        printf("Stack Underflow\n");
        exit(1);
    }
    returnstack[top--];
}
charpeek(){
    if(top==-1){
        return '\0';
    }
    returnstack[top];
}
intprecedence(charop){ switch
    (op) {
        case'+':
        case'-':
            return1;
        case '*':
        case'/':
            return2;
        default:
            return0;
    }
}

intisOperator(charc){
    returnc== '+'|| c== '-'|| c== '*'|| c== '/';
}
voidinfixToPostfix(constchar*infix,char*postfix){
```

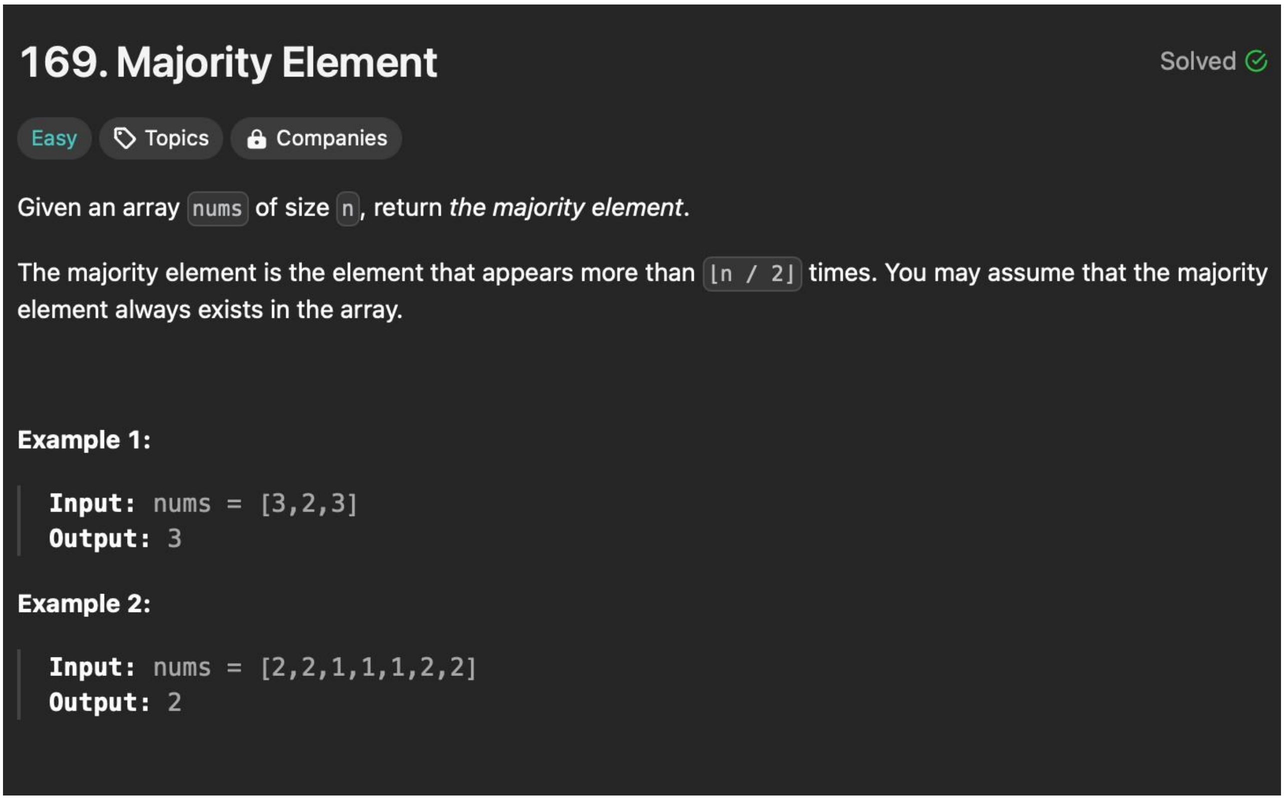
```
inti,j=0; char
c;
 for(i=0;infix[i]!='\0';i++){ c =
    infix[i];


    if (isalnum(c)) {
        postfix[j++]=c;
    }elseif(c=='('){ push(c);
    }elseif (c== ')') {
        while(top!=-1&&peek()!='('){
            postfix[j++] = pop();
        }
        if (top == -1 || peek() != '(') {
            printf("Invalidexpression\n");
            exit(1);
        }
        pop();
    }elseif(isOperator(c)){
       while(top!=-1&&precedence(peek())>=precedence(c)){
            postfix[j++] = pop();
        }
        push(c);
    }else{
        printf("Invalidcharacterinexpression\n");
        exit(1);
    }
}
```

```
while(top != -1) {
     if(peek()=='(') {
         printf("Invalidexpression\n");
         exit(1);
      }
      postfix[j++]=pop();
   }

   postfix[j]=‘\0';
}
intmain() {
   charinfix[MAX],postfix[MAX];
   printf("Enteravalidparenthesizedinfixexpression:"); scanf("%s",
   infix);

   infixToPostfix(infix,postfix);

   printf("Postfixexpression:%s\n",postfix);
return 0;
}OUTPUT
```

:

```
Enteravalidparenthesizedinfixexpression:(a+b)*c
Postfix expression: ab+c*
Enteravalidparenthesizedinfixexpression:((a+b)*c-d)/e
Postfix expression: ab+c*d-e/
Enteravalidparenthesizedinfixexpression:(a+b)&c
Invalid character in expression
```

3.



169. Majority Element

Given an array `nums` of size `n`, return *the majority element*.

The majority element is the element that appears more than `⌊n / 2⌋` times. You may assume that the majority element always exists in the array.

Example 1:

```
Input: nums = [3,2,3]
Output: 3
```

Example 2:

```
Input: nums = [2,2,1,1,1,2,2]
Output: 2
```

```
intmajorityElement(int*nums,intnumsSize){ int
num;
intcount = 0;
for(inti=0;i<numsSize;i++){
if(nums[i] == num) {
continue;
}
num = nums[i];

for(intj=0;j<numsSize;j++){
if(nums[j] == num) {
count++;
}
}
if(count> (numsSize/ 2)) {
```

```
break;

}

count= 0;

}

returnnum;

}
```

OUTPUT:

```
Input: [3,2,3]

Output: 3
```

4a. WAPto simulate the working of a queue of integers using an array. Provide the following operations:Insert,Delete,DisplayTheprogramshouldprintappropriatemessagesforqueueempty and queue overflow conditions .

4b.WAPtosimulatetheworkingofacircularqueueofintegersusinganarray.Provide the following operations:
Insert, Delete &amp; Display The program should print appropriate messages for queue empty and queue overflow conditions.

```
4a. #include <stdio.h>
     #define MAX 5

intqueue[MAX];
intfront=-1,rear=-1;

void insert(int value)
   {if(rear==MAX-1){
       printf("QueueOverflow!Cannotinsert%d.\n",value);
   }else{
       if(front==-1)front=0;
       queue[++rear] = value;
       printf("Inserted%dintothequeue.\n",value);
   }
}

voiddelete(){
   if(front==-1||front>rear){
       printf("QueueUnderflow!Cannotdeleteelement.\n");
   }else{
       printf("Deleted %d from the queue.\n", queue[front++]);
       if (front > rear) {
           front = rear= -1;
       }
   }
}
```

```
voiddisplay(){
    if(front==-1){
        printf("Queueisempty.\n");
    }else{
        printf("Queue elements are: ");
        for(inti=front;i<=rear;i++){
            printf("%d",queue[i]);
        }
        printf("\n");
    }
}

intmain(){
    intchoice,value;

    do{
        printf("\nQueue Operations:\n");
        printf("1. Insert\n");
        printf("2. Delete\n");
        printf("3. Display\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch(choice){
            case 1:
                printf("Enterthevaluetoinsert:");
                scanf("%d", &value);
                insert(value);
                break;
            case2:
                delete();
                break;
            case3:
                display();
                break;
            case4:
                printf("Exiting Queue operations.\n");
                break;
            default:
                printf("Invalidchoice!Pleasetryagain.\n");
        }
    }while(choice!=4);

    return0;
}
```

OUTPUT:

```
ChooseQueueType:
1. LinearQueue
```

```
2. Circular Queue
Enteryourchoice:1

LinearQueueOperations:
1. Insert
2. Delete
3. Display
4. Exit
Enteryourchoice:1
Enterthevaluetoinsert:10
Enter your choice: 1
Enterthevaluetoinsert:20
Enter your choice: 3
Enteryourchoice:2
Enteryourchoice:3
Enteryourchoice:4
Inserted 10 into the queue.
Inserted 20 into the queue.
Queue elements: 10 20
Deleted10fromthequeue.
Queue elements: 20
ExitingLinearQueueoperations.
```

```
4b. #include <stdio.h>
    #define MAX 5

intcqueue[MAX];
intfront=-1,rear=-1;

voidinsert(intvalue){
   if((rear+1)%MAX==front){
      printf("CircularQueueOverflow!Cannotinsert%d.\n",value);
   }else{
      if (front == -1) front = 0;
      rear=(rear+1)%MAX;
      cqueue[rear] = value;
      printf("Inserted%dintothecircularqueue.\n",value);
   }
}

voiddelete(){
   if(front==-1){
      printf("CircularQueueUnderflow!Cannotdeleteelement.\n");
   }else{
      printf("Deleted %d from the circular queue.\n", cqueue[front]);
      if (front == rear) {
         front = rear= -1;
      }else{
         front=(front+1)%MAX;
      }
   }
}
```

```
voiddisplay(){
    if(front==-1){
        printf("CircularQueue isempty.\n");
    }else{
        printf("CircularQueueelementsare:"); int
        i = front;
        while(1){
            printf("%d ", cqueue[i]);
            if (i == rear) break;
            i=(i+1)%MAX;
        }
        printf("\n");
    }
}

intmain(){
    intchoice,value;

    do{
        printf("\nCircular Queue Operations:\n");
        printf("1. Insert\n");
        printf("2. Delete\n");
        printf("3. Display\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch(choice){
            case 1:
                printf("Enterthevaluetoinsert:");
                scanf("%d", &value);
                insert(value);
                break;
            case2:
                delete();
                break;
            case3:
                display();
                break;
            case4:
                printf("Exiting Circular Queue operations.\n");
                break;
            default:
                printf("Invalidchoice!Pleasetryagain.\n");
        }
    }while(choice!=4);

    return0;
}
```

OUTPUT:

```
ChooseQueueType:
```

```
1. LinearQueue
2. Circular Queue
Enteryourchoice:2

CircularQueueOperations:
1. Insert
2. Delete
3. Display
4. Exit
Enteryourchoice:1
Enterthevaluetoinsert:30
Enter your choice: 1
Enterthevaluetoinsert:40
Enter your choice: 1
Enterthevaluetoinsert:50
Enter your choice: 2
Enteryourchoice:3
Enteryourchoice:4
Inserted 30 into the circular queue.
Inserted 40 into the circular queue.
Inserted 50 into the circular queue.
Deleted30fromthecircularqueue.
Circular Queue elements: 40 50
Exiting Circular Queue operations.
```

5. Game oftwo stacks(Hackerrank)

```
#include <assert.h>

#include <ctype.h>

#include <limits.h>

#include <math.h>

#include<stdbool.h>

#include <stddef.h>

#include <stdint.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

char* readline();

char* ltrim(char*);

char* rtrim(char*);
```

```
char**split_string(char*);

int parse_int(char*);

/*

* Completethe'twoStacks'functionbelow.

*

* ThefunctionisexpectedtoreturnanINTEGER.

* Thefunctionacceptsfollowingparameters:

* 1.INTEGER maxSum

* 2.INTEGER_ARRAYa

* 3.INTEGER_ARRAYb

*/

inttwoStacks(intmaxSum,inta_count,int*a,intb_count,int*b){ int

sumStackA[a_count + 1];

intsumStackB[b_count+1];

int max = 0;

sumStackA[0]=0;

for(int i = 0; i < a_count; i++) {

sumStackA[i+1]=sumStackA[i]+a[i];

}

sumStackB[0]=0;

for(int i = 0; i < b_count; i++) {

sumStackB[i+1]=sumStackB[i]+b[i];

}

intj = b_count;

for(inti=0;i<=a_count;i++){

if(sumStackA[i] > maxSum) {

break;

}
```

```
while(j>0&&sumStackA[i]+sumStackB[j]>maxSum){ j--;

}

max= (max > i+ j) ?max: (i + j);

}

returnmax;

}

int main()

{

FILE*fptr=fopen(getenv("OUTPUT_PATH"),"w");

int g = parse_int(ltrim(rtrim(readline())));

for (int g_itr = 0; g_itr < g; g_itr++) {

char**first_multiple_input=split_string(rtrim(readline())); int

n = parse_int(*(first_multiple_input + 0));

intm=parse_int(*(first_multiple_input+ 1));

intmaxSum=parse_int(*(first_multiple_input+2));

char** a_temp = split_string(rtrim(readline()));

int*a=malloc(n*sizeof(int)); for

(int i = 0; i < n; i++) {

inta_item=parse_int(*(a_temp+i));

*(a+ i) = a_item;

}

char**b_temp=split_string(rtrim(readline()));

int* b = malloc(m * sizeof(int));

for (int i = 0; i < m; i++) {

intb_item=parse_int(*(b_temp+i));

*(b + i) = b_item;

}
```

```
}

intresult=twoStacks(maxSum,n,a,m,b);

fprintf(fptr, "%d\n", result);

}

fclose(fptr);

return 0;

}

char*readline(){

size_talloc_length=1024;

size_t data_length = 0;

char*data=malloc(alloc_length);

while (true) {

char*cursor=data+data_length;

char*line=fgets(cursor,alloc_length-data_length,stdin); if

(!line) {

break;

}

data_length+=strlen(cursor);

if(data_length<alloc_length-1||data[data_length-1]=='\n'){ break;

}

alloc_length<<=1;

data=realloc(data,alloc_length); if

(!data) {

data='\0';

break;

}

}

if(data[data_length-1]=='\n'){
```

```
data[data_length-1]='\0';
data=realloc(data,data_length);
if (!data) {
data= '\0';
}
}else{
data=realloc(data,data_length+1); if
(!data) {
data= '\0';
} else {
data[data_length]='\0';
}
}
returndata;
}
char*ltrim(char*str){ if
(!str) {
return'\0';
}
if(!*str){
returnstr;
}
while(*str!='\0'&&isspace(*str)){ str++;
}
returnstr;
}
char*rtrim(char*str){
```

```
if (!str) {
return'\0';
}
if(!*str){
returnstr;
}
char*end=str+strlen(str)- 1;
while(end>=str&&isspace(*end)){
end--;
}
*(end+1)='\0';
return str;
}
char**split_string(char*str){ char**
splits = NULL;
char*token=strtok(str,""); int
spaces = 0;
while(token) {
splits=realloc(splits,sizeof(char*)*++spaces); if
(!splits) {
returnsplits;
}
splits[spaces - 1] = token;
token=strtok(NULL,"");
}
returnsplits;
}
intparse_int(char*str){
```

```
char* endptr;

intvalue=strtol(str,&endptr,10);

if(endptr==str||*endptr!='\0'){

exit(EXIT_FAILURE);

}

returnvalue;

}OUTPUT

:

Input : 1

5 4 10

4 2 4 6 1

2 1 8 5

Output :

4
```

6. WAPtoImplementSingleLinkListwithfollowingoperations:Sortthelinkedlist,Reversethe linked list, Concatenation of two linked lists.

```
#include  <stdio.h>

#include<stdlib.h>

struct Node {

   int data;

   structNode*next;

};


structNode*createNode(intdata){

   structNode*newNode=(structNode*)malloc(sizeof(structNode));

   newNode->data = data;

   newNode->next=NULL;

   return newNode;
```

```
}
voidinsertNode(structNode**head,intdata){
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        *head=newNode;
    }else{
        struct Node* temp = *head;
        while(temp->next!=NULL){
            temp=temp->next;
        }
        temp->next=newNode;
    }
}

voiddisplayList(structNode*head){ if
    (head == NULL) {
        printf("Thelistisempty.\n");
        return;
    }
    structNode*temp=head;
    printf("Linked list: ");
    while (temp != NULL) {
        printf("%d",temp->data);
        temp = temp->next;
    }
    printf("\n");
}
voidsortList(structNode**head){
    if(*head==NULL||(*head)->next==NULL)return; struct
    Node* current = *head;
    structNode*index=NULL; int
    temp;

    while (current != NULL) {
        index = current->next;
        while(index!=NULL){
            if(current->data>index->data){
                temp = current->data;
                current->data=index->data;
                index->data = temp;
            }
            index=index->next;
        }
        current=current->next;
    }
    printf("Linkedlistsorted.\n");
}

voidreverseList(structNode**head){ struct
    Node* prev = NULL;
    structNode*current=*head; struct
    Node* next = NULL;

    while(current!=NULL){
        next = current->next;
        current->next = prev;
        prev=current;
        current=next;
    }
    *head=prev;
    printf("Linkedlistreversed.\n");
}

voidconcatenateLists(structNode**head1,structNode**head2){ if
    (*head1 == NULL) {
        *head1=*head2;
    }else{
        struct Node* temp = *head1;
        while(temp->next!=NULL){
            temp=temp->next;
        }
        temp->next=*head2;
    }
    printf("Lists concatenated.\n");
}

intmain() {
    structNode*list1=NULL;
    structNode*list2=NULL;
    int choice, value;

    do {
        printf("\nSingleLinkedListOperations:\n");
        printf("1. Insert into List 1\n");
        printf("2.InsertintoList2\n");

        printf("3. Display List 1\n");

        printf("4. Display List 2\n");

        printf("5. Sort List 1\n");

        printf("6. Reverse List 1\n");

        printf("7.ConcatenateList2intoList1\n"); printf("8.

        Exit\n");

        printf("Enteryourchoice:");

        scanf("%d", &choice);


        switch(choice){

            case 1:

               printf("EntervaluetoinsertintoList1:");

               scanf("%d", &value);

               insertNode(&list1,value);

               break;

            case 2:

               printf("EntervaluetoinsertintoList2:");

               scanf("%d", &value);

               insertNode(&list2,value);

               break;

            case 3:

               displayList(list1);

               break;

            case 4:

               displayList(list2);

               break;

            case 5:
```

```
            sortList(&list1);
            break;
        case 6:
            reverseList(&list1);
            break;
        case 7:
            concatenateLists(&list1,&list2);
            break;
        case 8:
            printf("Exitingprogram.\n");
            break;
        default:
            printf("Invalidchoice.Pleasetry again.\n");
      }
    }while(choice!= 8);

    return0;
}OUTPUT
```

:

```
SingleLinked List Operations:
1. InsertintoList1
2. InsertintoList2
3. DisplayList 1
4. DisplayList 2
5. Sort List 1
6. ReverseList1
7. ConcatenateList2intoList 1
8. Exit
```

```
Enteryourchoice:1
Entervalueto insertinto List1: 5

SingleLinked List Operations:
1. InsertintoList1
2. InsertintoList2
3. DisplayList 1
4. DisplayList 2
5. Sort List 1
6. ReverseList1
7. ConcatenateList2intoList 1
8. Exit
Enteryourchoice:1
Entervalueto insertinto List1: 10

SingleLinked List Operations:
1. InsertintoList1
2. InsertintoList2
3. DisplayList 1
4. DisplayList 2
5. Sort List 1
6. ReverseList1
7. ConcatenateList2intoList 1
8. Exit
Enteryourchoice:3
Linked list: 5 10

SingleLinked List Operations:
```

```
1. InsertintoList1
2. InsertintoList2
3. DisplayList 1
4. DisplayList 2
5. Sort List 1
6. ReverseList1
7. ConcatenateList2intoList 1
8. Exit
Enteryourchoice:2
Entervalueto insertinto List2: 2

SingleLinked List Operations:
1. InsertintoList1
2. InsertintoList2
3. DisplayList 1
4. DisplayList 2
5. Sort List 1
6. ReverseList1
7. ConcatenateList2intoList 1
8. Exit
Enteryourchoice:4
Linked list: 2

SingleLinked List Operations:
1. InsertintoList1
2. InsertintoList2
3. DisplayList 1
4. DisplayList 2
```

```
5. Sort List 1
6. ReverseList1
7. ConcatenateList2intoList 1
8. Exit
Enteryourchoice:7 Lists
concatenated.

SingleLinked List Operations:
1. InsertintoList1
2. InsertintoList2
3. DisplayList 1
4. DisplayList 2
5. Sort List 1
6. ReverseList1
7. ConcatenateList2intoList 1
8. Exit
Enteryourchoice:3
Linked list: 5 10 2

SingleLinked List Operations:
1. InsertintoList1
2. InsertintoList2
3. DisplayList 1
4. DisplayList 2
5. Sort List 1
6. ReverseList1
7. ConcatenateList2intoList 1
8. Exit
```

```
Enteryourchoice:5
Linked list sorted.

SingleLinked List Operations:
1. InsertintoList1
2. InsertintoList2
3. DisplayList 1
4. DisplayList 2
5. Sort List 1
6. ReverseList1
7. ConcatenateList2intoList 1
8. Exit
Enteryourchoice:3
Linked list: 2 5 10

SingleLinked List Operations:
1. InsertintoList1
2. InsertintoList2
3. DisplayList 1
4. DisplayList 2
5. Sort List 1
6. ReverseList1
7. ConcatenateList2intoList 1
8. Exit
Enteryourchoice:6
Linkedlist reversed.

SingleLinked List Operations:
```

```
1. InsertintoList1
2. InsertintoList2
3. DisplayList 1
4. DisplayList 2
5. Sort List 1
6. ReverseList1
7. ConcatenateList2intoList 1
8. Exit
Enteryourchoice:3
Linked list: 10 5 2

SingleLinked List Operations:
1. InsertintoList1
2. InsertintoList2
3. DisplayList 1
4. DisplayList 2
5. Sort List 1
6. ReverseList1
7. ConcatenateList2intoList 1
8. Exit
Enteryourchoice:8
Exiting program.
```

7.WAPtoImplementSingleLinkListtosimulateStack&QueueOperation

```
#include <stdio.h>
#include<stdlib.h>

structNode{
    int data;

    structNode*next;

};


structNode*createNode(intdata){

    structNode*newNode=(structNode*)malloc(sizeof(structNode));

    newNode->data = data;

    newNode->next=NULL;

    return newNode;

}


voidpush(structNode**top,intdata){

    structNode*newNode=createNode(data);

    newNode->next = *top;

    *top = newNode;

    printf("%dpushedtostack.\n", data);

}


intpop(structNode**top){ if

    (*top == NULL) {

        printf("Stackisempty!\n");

        return -1;

    }

    structNode*temp= *top;

    intpoppedData=temp->data;

    *top=(*top)->next;

    free(temp);

    returnpoppedData;
```

```
}


voidenqueue(structNode**front,structNode**rear,intdata){ struct
   Node* newNode = createNode(data);
   if(*rear ==NULL) {
      *front= *rear= newNode;
      printf("%denqueuedtoqueue.\n",data);
      return;
   }
   (*rear)->next=newNode;
   *rear=newNode;
   printf("%denqueuedtoqueue.\n", data);
}


intdequeue(structNode**front,structNode**rear){ if
   (*front == NULL) {
      printf("Queueisempty!\n");
      return -1;
   }
   structNode*temp= *front;
   intdequeuedData=temp->data;
   *front=(*front)->next; if
   (*front == NULL) {
      *rear=NULL;
   }
   free(temp);
   returndequeuedData;
}
```

```
voiddisplay(structNode*head){ if
   (head == NULL) {
      printf("Thelistisempty.\n");
      return;
   }
   structNode*temp=head;
   printf("The list: ");
   while (temp != NULL) {
      printf("%d",temp->data);
      temp = temp->next;
   }
   printf("\n");
}

intmain() {
   struct Node* stackTop = NULL;
    structNode*queueFront=NULL;
   struct Node* queueRear = NULL;

   intchoice,value;

   do {
      printf("\nChooseanoperation:\n");
      printf("1. Stack Push\n");printf("2.
      Stack Pop\n");
      printf("3. Display Stack\n");
      printf("4.QueueEnqueue\n");
      printf("5.Queue Dequeue\n");
      printf("6.DisplayQueue\n");
      printf("7. Exit\n");
      printf("Enteryourchoice:");
      scanf("%d", &choice);

      switch(choice){
         case 1:
            printf("Entervaluetopushontothestack:");
            scanf("%d", &value);
            push(&stackTop,value);
            break;
         case 2:
            value=pop(&stackTop); if
            (value != -1) {
               printf("Poppedfromstack:%d\n",value);
            }
            break;
         case 3:
            display(stackTop);
            break;
         case 4:
            printf("Entervaluetoenqueuetothequeue:"); scanf("%d",
            &value);
            enqueue(&queueFront,&queueRear,value);
            break;
         case 5:
            value=dequeue(&queueFront,&queueRear); if
            (value != -1) {
```

```
                    printf("Dequeuedfromqueue:%d\n", value);
                }
                break;
            case 6:
                display(queueFront);
                break;
            case 7:
                printf("Exitingtheprogram.\n");
                break;
            default:
                printf("Invalidchoice!Pleasetry again.\n");
        }
    }while(choice!= 7);

    return0;
}OUTPUT
```

:

```
Choosean operation:

1. StackPush

2. StackPop

3. DisplayStack

4. Queue Enqueue

5. Queue Dequeue

6. DisplayQueue

7. Exit

Enteryourchoice:1

Entervaluetopushontothestack:10 10

pushed to stack.
```

```
Choosean operation:
1. StackPush
2. StackPop
3. DisplayStack
4. Queue Enqueue
5. Queue Dequeue
6. DisplayQueue
7. Exit
Enteryourchoice:1
Entervaluetopushontothestack:20 20
pushed to stack.

Choosean operation:
1. StackPush
2. StackPop
3. DisplayStack
4. Queue Enqueue
5. Queue Dequeue
6. DisplayQueue
7. Exit
Enteryourchoice:3
The list: 20 10

Choosean operation:
1. StackPush
2. StackPop
3. DisplayStack
```

```
4. Queue Enqueue
5. Queue Dequeue
6. DisplayQueue
7. Exit
Enteryourchoice:4
Entervaluetoenqueuetothequeue:30 30
enqueued to queue.

Choosean operation:
1. StackPush
2. StackPop
3. DisplayStack
4. Queue Enqueue
5. Queue Dequeue
6. DisplayQueue
7. Exit
Enteryourchoice:4
Entervaluetoenqueuetothequeue:40 40
enqueued to queue.

Choosean operation:
1. StackPush
2. StackPop
3. DisplayStack
4. Queue Enqueue
5. Queue Dequeue
6. DisplayQueue
7. Exit
```

```
Enteryourchoice:6

The list: 30 40


Choosean operation:

1. StackPush

2. StackPop

3. DisplayStack

4. Queue Enqueue

5. Queue Dequeue

6. DisplayQueue

7. Exit

Enter your choice: 5

Dequeuedfromqueue:30


Choosean operation:

1. StackPush

2. StackPop

3. DisplayStack

4. Queue Enqueue

5. Queue Dequeue

6. DisplayQueue

7. Exit

Enteryourchoice:6

The list: 40


Choosean operation:

1. StackPush

2. StackPop
```

```
3. DisplayStack

4. Queue Enqueue

5. Queue Dequeue

6. DisplayQueue

7. Exit

Enter your choice: 7

Exiting theprogram.
```

8. WAPto Implement doubly link list with primitive operations Create a doubly linked list.Insert a newnodetotheleftofthenode.Deletethenodebasedonaspecificvalue Displaythecontentsofthe list2.

```
#include  <stdio.h>

#include<stdlib.h>

struct Node {

    int data;

    structNode*prev;

    structNode*next;

};

structNode*createNode(intdata){

    structNode*newNode=(structNode*)malloc(sizeof(structNode));

    newNode->data = data;

    newNode->prev=NULL;

    newNode->next=NULL;

    return newNode;

}

voidinsertNode(structNode**head,intdata){

    struct Node* newNode = createNode(data);

    if (*head == NULL) {

        *head=newNode;
        return;
    }
    struct Node* temp = *head;
    while(temp->next!=NULL){
        temp=temp->next;
    }
    temp->next=newNode;
    newNode->prev=temp;
    printf("Nodewithvalue%dinserted.\n",data);
}
voidinsertLeft(structNode**head,intnewData,intleftOfValue){
    struct Node* newNode = createNode(newData);
    structNode*temp= *head;
    while(temp!=NULL&&temp->data!=leftOfValue){ temp
        = temp->next;
    }

    if (temp != NULL) {
        newNode->next=temp;
        newNode->prev=temp->prev;

        if(temp->prev !=NULL) {
            temp->prev->next=newNode;
        }else{
            *head=newNode;//Ifit'sthefirstnode
        }
        temp->prev=newNode;
```

```
  printf("Nodewithvalue%dinsertedtotheleftofnodewithvalue%d.\n",newData, leftOfValue);
    }else{
        printf("Nodewithvalue%dnotfound.\n", leftOfValue);
    }
}
voiddeleteNode(structNode**head,intvalue){
    struct Node* temp = *head;

     while(temp!=NULL&&temp->data!=value){ temp
        = temp->next;
    }

      if(temp != NULL) {
           if (temp->prev != NULL) {
          temp->prev->next=temp->next;
       }else{
          *head=temp->next;//Ifit'sthefirstnode,changehead
       }

      if(temp->next !=NULL) {
          temp->next->prev=temp->prev;
       }

       free(temp);
       printf("Nodewithvalue%ddeleted.\n",value);
    }else{
       printf("Nodewithvalue%dnotfound.\n", value);
    }
```

```
}


voiddisplayList(structNode*head){ if
    (head == NULL) {
        printf("Thelistisempty.\n");
        return;
    }
    struct Node* temp = head;
    printf("DoublyLinkedList:");
    while (temp != NULL) {
        printf("%d",temp->data);
        temp = temp->next;
    }
    printf("\n");
}


intmain() {
    struct Node* head = NULL;
    intchoice,value,leftOfValue;


    do {
        printf("\nDoubly Linked List Operations:\n");
        printf("1. Create a new node (Insert)\n");
        printf("2.Insertanodetotheleftofanode\n");
        printf("3. Delete a node based on value\n");
        printf("4. Display the list\n");
        printf("5. Exit\n");
        printf("Enteryourchoice:");
```

```
scanf("%d",&choice);


switch(choice){
    case 1:
        printf("Entervaluetoinsertintothelist:");
        scanf("%d", &value);
        insertNode(&head,value);
        break;
    case 2:
        printf("Entervaluetoinsert:");
        scanf("%d", &value);
        printf("Enterthevaluetoinsertleftof:");
        scanf("%d", &leftOfValue);
        insertLeft(&head, value, leftOfValue);
        break;
    case 3:
        printf("Entervaluetodeletefromthelist:");
        scanf("%d", &value);
        deleteNode(&head,value);
        break;
    case 4:
        displayList(head);
        break;
    case 5:
        printf("Exitingtheprogram.\n");
        break;
    default:
        printf("Invalidchoice!Pleasetry again.\n");
        }
    }while(choice!= 5);


    return0;
}
```

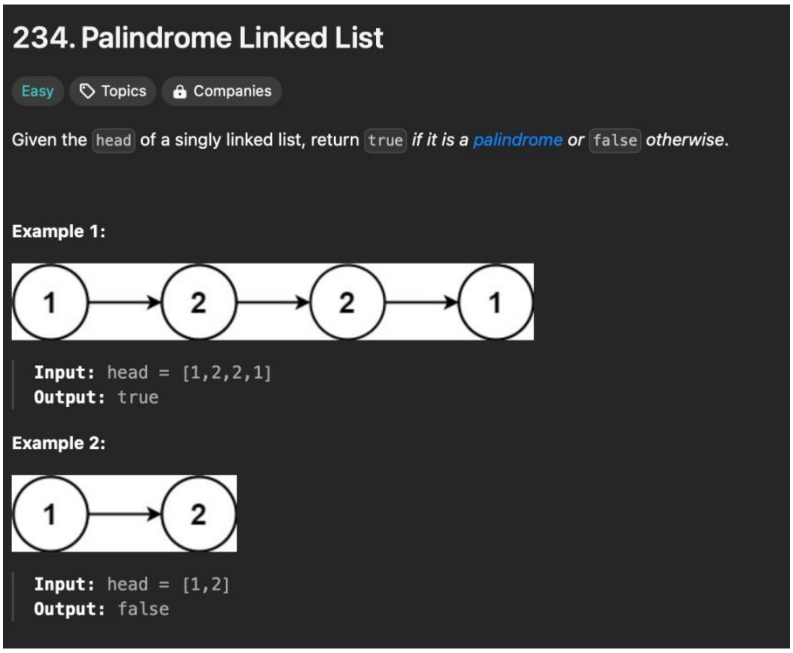
OUTPUT:

```
DoublyLinked List Operations:
1. Createanewnode(Insert)
2. Insertanodeto theleft ofanode
3. Deleteanodebasedon value
4. Displaythelist
5. Exit
Enteryourchoice:1
Entervaluetoinsertintothelist:10
Node with value 10 inserted.

DoublyLinked List Operations:
1. Createanewnode(Insert)
2. Insertanodeto theleft ofanode
3. Deleteanodebasedon value
4. Displaythelist
5. Exit
Enteryourchoice:1
Entervaluetoinsertintothelist:20
Node with value 20 inserted.

DoublyLinked List Operations:
1. Createanewnode(Insert)
```

```
2. Insertanodeto theleft ofanode

3. Deleteanodebasedon value

4. Displaythelist

5. Exit

Enter your choice: 4

DoublyLinkedList:1020


DoublyLinked List Operations:

1. Createanewnode(Insert)

2. Insertanodeto theleft ofanode

3. Deleteanodebasedon value

4. Displaythelist

5. Exit

Enter your choice: 2

Entervaluetoinsert:15

Enterthevalueto insertleftof: 20

Nodewith value15 insertedto theleft ofnodewithvalue20.


DoublyLinked List Operations:

1. Createanewnode(Insert)

2. Insertanodeto theleft ofanode

3. Deleteanodebasedon value

4. Displaythelist

5. Exit

Enteryourchoice:4

DoublyLinked List: 10 15 20


DoublyLinked List Operations:
```

```
1. Createanewnode(Insert)

2. Insertanodeto theleft ofanode

3. Deleteanodebasedon value

4. Displaythelist

5. Exit

Enteryourchoice:3

Entervaluetodeletefromthelist:15 Node

with value 15 deleted.


DoublyLinked List Operations:

1. Createanewnode(Insert)

2. Insertanodeto theleft ofanode

3. Deleteanodebasedon value

4. Displaythelist

5. Exit

Enter your choice: 4

DoublyLinkedList:1020
```

9.



234. Palindrome Linked List

Given the `head` of a singly linked list, return `true` *if it is a palindrome or* `false` *otherwise.*

Example 1:

```
Input: head = [1,2,2,1]
Output: true
```

Example 2:

```
Input: head = [1,2]
Output: false
```

/**

```
 * Definitionforsingly-linkedlist.
 * structListNode{
 *    int val;
 *    structListNode*next;
 * };
 */
boolisPalindrome(structListNode*head){
    if(head==NULL||head->next==NULL){ return
        true;
    }
    struct ListNode *fast = head;
    structListNode*slow= head;
    while(fast!=NULL&&fast->next!=NULL){ slow =
        slow->next;
        fast=fast->next->next;
    }
    struct ListNode *prev = NULL;
    structListNode*current=slow;
    struct ListNode *next = NULL;

    while(current!=NULL){
        next = current->next;
        current->next = prev;
        prev = current;
        current=next;
    }

       structListNode*firstHalf=head;
```

```
   structListNode*secondHalf=prev;


     while(secondHalf!=NULL){

       if(firstHalf->val!=secondHalf->val){ return

           false;

       }

       firstHalf = firstHalf->next;

       secondHalf=secondHalf->next;

   }

  returntrue;

}
```

OUTPUT:

CASE1:

INPUT:head= [1,2,2,1]

OUTPUT: true

CASE2 :

INPUT:head= [1,2]

OUTPUT: false

10. .Writeaprogram

- ToconstructabinarySearchtree.
- Totraversethetreeusingallthemethodsi.e.,in-order,preorderandpost order
- Todisplaytheelementsinthetree.

```
#include  <stdio.h>
#include  <stdlib.h>
struct Node {
   intdata;
   struct  Node*  left;
   structNode*right;
};

structNode*createNode(intdata){
   structNode*newNode=(structNode*)malloc(sizeof(structNode));
   newNode->data = data;
   newNode->left = NULL;
   newNode->right = NULL;
   return newNode;
}

struct Node* insertNode(struct Node* root, int data) {
       if (root == NULL) {
        returncreateNode(data);
   }

       if(data<root->data){
        root->left=insertNode(root->left,data);
   }else{
        root->right=insertNode(root->right,data);
   }

   returnroot;
}

voidinorderTraversal(structNode*root){ if
   (root != NULL) {
        inorderTraversal(root->left);
        printf("%d ", root->data);
        inorderTraversal(root->right);
   }
}

voidpreorderTraversal(structNode*root){ if
   (root != NULL) {
        printf("%d ", root->data);
        preorderTraversal(root->left);
        preorderTraversal(root->right);
   }
}

voidpostorderTraversal(structNode*root){ if
   (root != NULL) {
        postorderTraversal(root->left);
        postorderTraversal(root->right);
        printf("%d ", root->data);
   }
}

voiddisplayTree(structNode*root){
   printf("In-order traversal: ");
   inorderTraversal(root);
   printf("\n");

   printf("Pre-ordertraversal:");
   preorderTraversal(root);
   printf("\n");

   printf("Post-order traversal: ");
   postorderTraversal(root);
   printf("\n");
}

intmain(){
   struct Node* root = NULL;
   int choice, value;

   do{
        printf("\nBinarySearchTreeOperations:\n");
        printf("1. Insert a node into the BST\n");
        printf("2. Display the tree elements\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch(choice){
            case 1:
                printf("EntervaluetoinsertintotheBST:"); scanf("%d",
                &value);
                root=insertNode(root,value);
                break;
            case2:
                if(root==NULL){
                    printf("Thetreeisempty.\n");
                }else{
                    displayTree(root);
                }
                break;
            case 3:
                printf("Exiting program.\n");
                break;
            default:
                printf("Invalidchoice!Pleasetryagain.\n");
        }
   }while(choice!=3);

   return0;
}
```

OUTPUT:

```
BinarySearchTreeOperations:
1. InsertanodeintotheBST
2. Displaythetreeelements
3. Exit
Enteryourchoice:1
EntervaluetoinsertintotheBST: 50
```

```
BinarySearchTreeOperations:
1. InsertanodeintotheBST
2. Displaythetreeelements
3. Exit
Enteryourchoice:1
EntervaluetoinsertintotheBST: 30

BinarySearchTreeOperations:
1. InsertanodeintotheBST
2. Displaythetreeelements
3. Exit
Enteryourchoice:1
EntervaluetoinsertintotheBST: 70

BinarySearchTreeOperations:
1. InsertanodeintotheBST
2. Displaythetreeelements
3. Exit
Enteryourchoice:1
EntervaluetoinsertintotheBST: 20

BinarySearchTreeOperations:
1. InsertanodeintotheBST
2. Displaythetreeelements
3. Exit
Enteryourchoice:2
In-ordertraversal:20305070
Pre-ordertraversal:50302070
Post-order traversal:203070 50
```

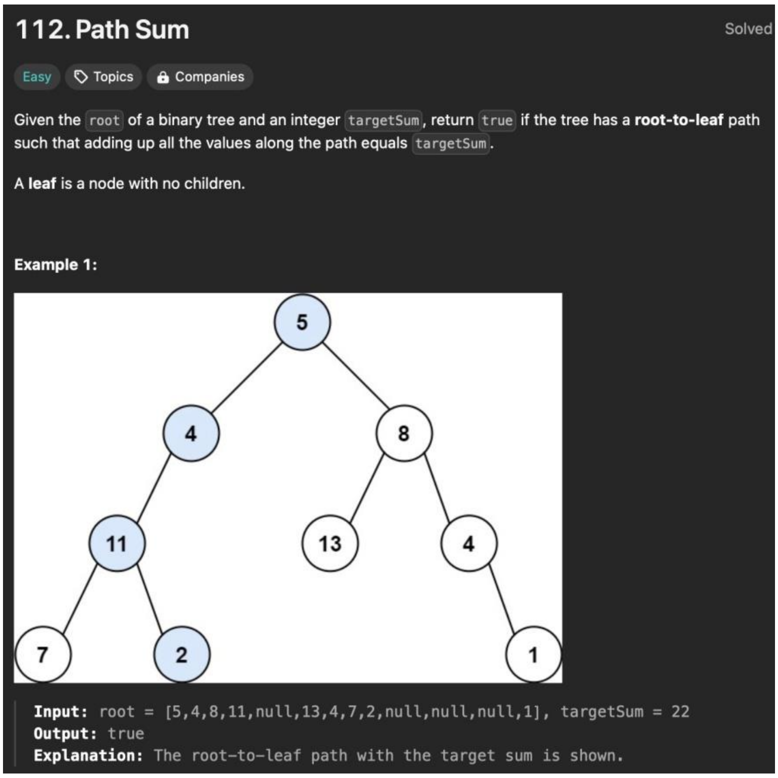
11. Write a program to traverse a graph using BFS method

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

#defineMAX_VERTICES50

voidBFS(intgraph[MAX_VERTICES][MAX_VERTICES],intvertices,intstartVertex){ bool
    visited[MAX_VERTICES] = {false};
   intqueue[MAX_VERTICES];
    int front = -1, rear = -1;

    visited[startVertex] = true;
    queue[++rear]=startVertex;

    printf("BFSTraversalOrder:");

    while (front != rear) {
        intcurrentVertex=queue[++front];
        printf("%d",currentVertex);//Printthevisitedvertex
        for(inti=0;i<vertices;i++){
            if(graph[currentVertex][i]==1&&!visited[i]){//Checkforadjacencyandvisitation visited[i] =
                true; // Mark as visited
                queue[++rear]= i; // Enqueue the adjacent vertex
            }
        }
    }
}

intmain(){
    intvertices,edges;
    intgraph[MAX_VERTICES][MAX_VERTICES]={0};

    printf("Input the number of vertices: ");
    scanf("%d", &vertices);

    printf("Input the number of edges: ");
    scanf("%d", &edges);

    printf("Input edges (format: start end):\n");
    for (int i = 0; i < edges; i++) {
        intstart,end;
        scanf("%d %d", &start, &end);
        graph[start][end] = 1;
        graph[end][start]=1;
    }

    intstartVertex;
    printf("Input the starting vertex for BFS: ");
    scanf("%d", &startVertex);

    BFS(graph,vertices,startVertex);

    return0;
}
```

OUTPUT:

```
Input the number of vertices: 5
Input the number of edges: 4
Inputedges(format:startend): 0
1
02
13
14
Input the starting vertex for BFS: 0
BFS Traversal Order: 0 1 2 3 4
```

12.



112. Path Sum

Solved

Given the `root` of a binary tree and an integer `targetSum`, return `true` if the tree has a **root-to-leaf** path such that adding up all the values along the path equals `targetSum`.

A **leaf** is a node with no children.

Example 1:

```
Input: root = [5,4,8,11,null,13,4,7,2,null,null,null,1], targetSum = 22
Output: true
Explanation: The root-to-leaf path with the target sum is shown.
```

```
boolhasPathSum(structTreeNode*root,inttargetSum){ if
    (root == NULL) {
       returnfalse;
    }
    targetSum-=root->val;
     if(root->left==NULL&&root->right==NULL){ return
       targetSum == 0;
    }
     returnhasPathSum(root->left,targetSum)||hasPathSum(root->right,targetSum);
}OUTPUT
```

:

CASE1 :

Root =

[5,4,8,11,null,13,4,7,2,null,null,null,1]

targetsum=

22

output =
True

Expected=
True

CASE2:

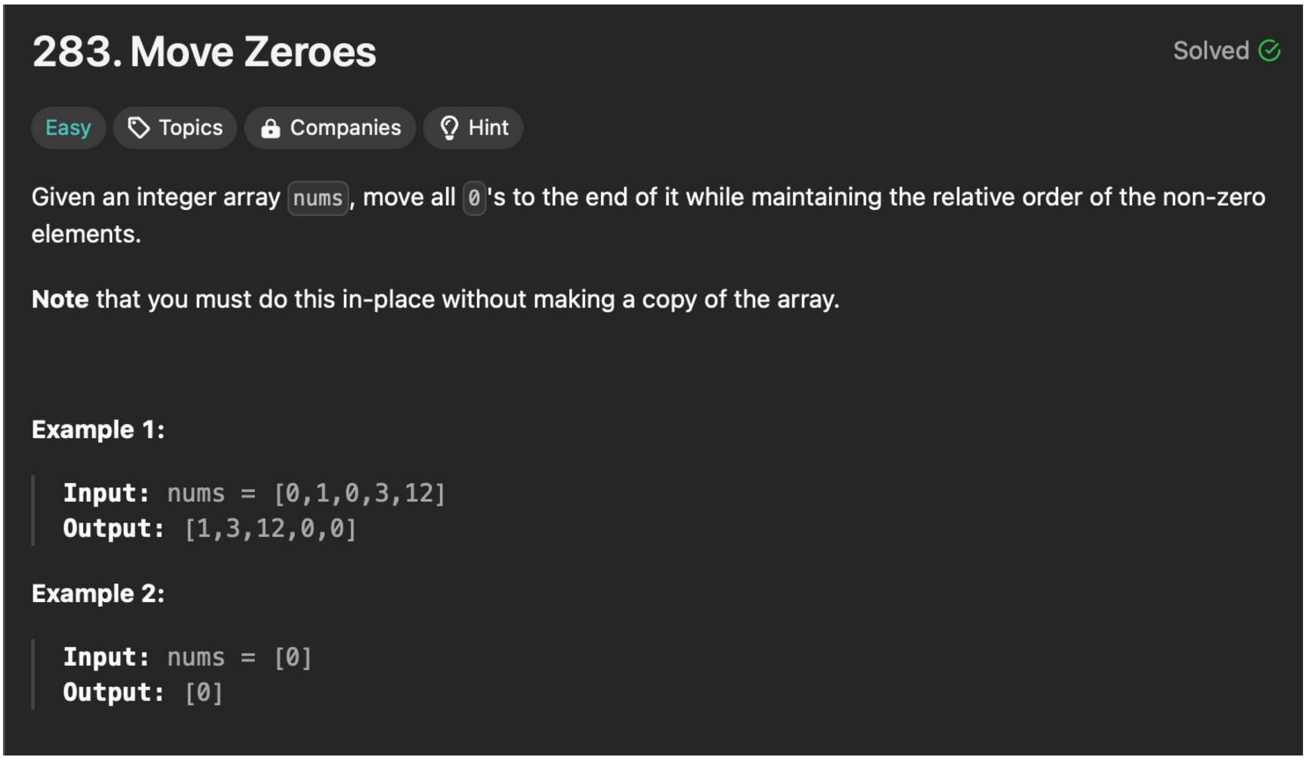
Root=
[1,2,3]

Targetsum=
5

Output=
False

Expected=
False

13.



283. Move Zeroes

Solved

Given an integer array `nums`, move all `0`'s to the end of it while maintaining the relative order of the non-zero elements.

Note that you must do this in-place without making a copy of the array.

Example 1:

```
Input: nums = [0,1,0,3,12]
Output: [1,3,12,0,0]
```

Example 2:

```
Input: nums = [0]
Output: [0]
```

voidmoveZeroes(int*nums,intnumsSize){

```
    int count=0;
    int j=0;
    for(inti=0;i<numsSize;i++){
        if(nums[i]==0){
            count++;
        }
        else{
            nums[j]=nums[i];
            j++;
        }
    }
    for(inti=0;i<count;i++){
        nums[j]=0;
        j++;
    }
}
```

OUTPUT:

CASE 1

Input

nums=

[0,1,0,3,12]

Output

[1,3,12,0,0]

Expected

[1,3,12,0,0]

CASE2 :

Input

nums=

[0]

Output

[0]

Expected

[0]

14. WriteaprogramforDepthfirstsearch

```
#include<stdio.h>
#include<stdlib.h>
inta[20][20], s[20], n;
voiddfs(intv)
{
int i;
s[v]=1;
for(i=1;i<=n;i++)
if(a[v][i]&&!s[i])
{
printf("\n%d->%d",v,i);
dfs(i);
}
}
int main()
{
int i, j, count=0;
printf("\nEnternumberofvertices:");
scanf("%d", &n);
for(i=1; i<=n; i++)
{
s[i]=0;
for(j=1;j<=n;j++)
a[i][j]=0;
}
printf("Entertheadjacencymatrix:\n");
for(i=1; i<=n; i++)
for(j=1; j<=n; j++)
scanf("%d",&a[i][j]);
dfs(1);
printf("\n");
for(i=1;i<=n;i++)
{
if(s[i])
count++;
}
if(count==n)
printf("Graphisconnected");
else
printf("Graphisnotconnected");
return 0;
}
/*
OUTPUT:
Enter number of vertices:5
Entertheadjacencymatrix: 0
1
1
1
0
0
1
0
1
```

```
0

1

1

1

0

1

0

1

0

0

1

0

1

0

1

1

1

0

1->2

2->3

3->4

4->5

Graphis connected

*/
```

15. Writeaprogramonhashingusinglinearprobing

```
#include <stdio.h>

#include<stdlib.h>

#defineTABLE_SIZE10
```

```
int h[TABLE_SIZE]={NULL};
void insert()
{
int key,index,i,flag=0,hkey;
printf("\nenteravaluetoinsertintohashtable\n");
scanf("%d",&key);
hkey=key%TABLE_SIZE;
for(i=0;i<TABLE_SIZE;i++)
{
index=(hkey+i)%TABLE_SIZE;
if(h[index] == NULL)
{
h[index]=key;
break;
}

}
printf("Noofprobesfor%dis%d",key,i+1); if(i
== TABLE_SIZE)
printf("\nelementcannotbeinserted\n");

}
void search()
{
int key,index,i,flag=0,hkey;
printf("\nentersearchelement\n");
scanf("%d",&key);
hkey=key%TABLE_SIZE;
```

```
for(i=0;i<TABLE_SIZE;i++)
{
index=(hkey+i)%TABLE_SIZE;
if(h[index]==key)
{
printf("valueisfoundatindex%d",index);
break;
}
}
if(i ==TABLE_SIZE)
printf("\nvalueisnot found\n");
}
void display()
{
int i;
printf("\nelementsinthehashtableare\n");
for(i=0;i< TABLE_SIZE; i++)
printf("\natindex%d\tvalue= %d",i,h[i]);
}
main()
{intopt,i;
while(1)
{printf("\nPress1.Insert\t2.Display\t3.Search\t4.Exit\n");
scanf("%d",&opt);
switch(opt)
{
case1:insert();
break;
case2:display();

break;

case3:search();

break;

case4:exit(0);

}

}

}OUTPUT

:

Press1. Insert     2.Display3. Search4.Exit

1

Enteravaluetoinsertintohashtable 15

Noof probes for 15 is 1


Press1. Insert     2.Display3. Search4.Exit

1

Enteravaluetoinsertintohashtable 25

Noof probes for 25 is 1


Press1. Insert     2.Display3. Search4.Exit

2


Elementsinthehashtableare At

index 0         value =15

Atindex 1      value=25

Atindex 2      value=0
```

```
 Atindex 3     value=   0

 Atindex 4     value=   0

 Atindex 5     value=   0

 Atindex 6     value=   0

 Atindex 7     value=   0

 Atindex 8     value=   0

 Atindex 9     value=   0


Press1. Insert     2.Display3. Search4.Exit

3

Entersearchelement

25

Valueisfoundatindex1


Press1. Insert     2.Display3. Search4.Exit

4

Exiting…
```