**DSA LAB PRACTICALS**

Title: Traversing in arrays

Introduction: Traversing is just scanning and printing every element of the array.

Code:

#include <stdio.h>

int main()

{

int i, arr1[10];

for(i=0;i<10;i+=1)

{

printf("Enter element for the array at index number %d=",i);

scanf("%d",&arr1[i]);

}

printf("printing the array=");

for(i=0;i<10;i+=1)

{

printf("%d",arr1[i]);

}

}

Output:

Conclusion: Thus we were able to get the desired output out of this program.

Title: Searching element in arrays

Introduction: This program searches for an element in the array and prints it’s position if found.

Code:

#include <stdio.h>

int main()

{

int i, n, arr1[5];

for(i=0;i<5;i+=1)

{

printf("Enter element for the array at index number %d=",i);

scanf("%d",&arr1[i]);

}

printf("Enter element to search=");

scanf("%d",&n);

for(i=0;i<5;i+=1)

{

if(arr1[i]==n)

{

printf("The element is at index number %d",i);

}

}

}

Output:

Conclusion: Thus we were able to get the desired output out of this program.

Title: Insertion in array

Introduction: It is inserting a value at any position in the array and arranging the rest of the values accordingly by changing their position to fit the value.

Code:

#include <stdio.h>

int main()

{

int i, j, n, arr1[20];

for(i=0;i<5;i+=1)

{

printf("Enter element for the array at index number %d=",i);

scanf("%d",&arr1[i]);

}

printf("Enter the element to indert=");

scanf("%d",&j);

printf("Enter the position at which it is to be inserted=");

scanf("%d",&n);

for(i=5;i>=n;i-=1)

{

arr1[i] = arr1[i-1];

}

arr1[n]=j;

printf("printing the array=");

for(i=0;i<5+1;i+=1)

{

printf("%d, ",arr1[i]);

}

}

Output:

Conclusion: Thus we were able to get the desired output out of this program

Title: Deleting Element from array

Introduction: Removing a value in the array and arranging the values after it accordingly.

Code:

#include <stdio.h>

int main()

{

int i, j, k, arr1[20];

for(i=0;i<5;i+=1)

{

printf("Enter element for the array at index number %d=",i);

scanf("%d",&arr1[i]);

}

printf("Enter the element to delete=");

scanf("%d",&j);

for(i=0;i<5;i+=1)

{

if(arr1[i]==j)

{

k=i;

}

}

for(i=k;i<5;i+=1)

{

arr1[i]=arr1[i+1];

}

printf("printing the array=");

for(i=0;i<5-1;i+=1)

{

printf("%d",arr1[i]);

}

}

Output:

Conclusion: Thus we were able to get the desired output out of this program.

Title: Merging arrays

Introduction: Combining the values of two separate arrays into one containing all values of previous.

Code:

#include <stdio.h>

int main()

{

int i, j, arr1[5],arr2[5],arr3[10];

printf("Enter elements for the first array=");

for(i=0;i<5;i+=1)

{

scanf("%d",&arr1[i]);

}

printf("Enter elements for the second array=");

for(i=0;i<5;i+=1)

{

scanf("%d",&arr2[i]);

}

for(i=0;i<5;i+=1)

{

arr3[i]=arr1[i];

}

for(i=0,j=5;i<5 && j<10;i+=1,j+=1)

{

arr3[j]=arr2[i];

}

printf("The merged array is=");

for(i=0;i<10;i+=1)

{

printf("%d, ",arr3[i]);

}

}

Output:

Conclusion: Thus we were able to get the desired output out of this program.

Title: Parentheses Counter

Introduction: Counts number of opening and closing parentheses.

Code:

#include <stdio.h>

int main()

{

char arr1[5];

int i, c, sum1=0, sum2=0, sum3;

printf("Enter an expression containing parentheses= ");

scanf("%s",arr1);

for(i=0;i<15;i+=1)

{

if(arr1[i]=='(' || arr1[i]=='{' || arr1[i]=='[')

{

sum1+=1;

}

if(arr1[i]==')' || arr1[i]=='}' || arr1[i]==']')

{

sum2+=1;

}

}

printf("Number of opening parentheses=%d and number of closing parentheses=%d",sum1,sum2);

sum3=sum1+sum2;

if(sum3%2==0)

{

printf("\nThis expression is valid! (or you put brackets wrong)");

}

else

{

printf("\nDo it again now!!");

}

}

Output:

Conclusion: Thus we were able to get the desired output out of this program.

Title: Parantheses Checker

Introduction: Checks the validity of the equation using stacks.

Code:

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

#define MAX 30

int top=-1;

int stack[MAX];

void push(char);

char pop();

int match(char a,char b);

int check(char []);

int main()

{

char exp[MAX];

int valid;

printf("Enter an algebraic expression : ");

scanf("%s",exp);

valid=check(exp);

if(valid==1)

printf("Valid expression\n");

else

printf("Invalid expression\n");

return 0;

}

int check(char exp[] )

{

int i;

char temp;

for(i=0;i<strlen(exp);i++)

{

if(exp[i]=='(' || exp[i]=='{' || exp[i]=='[')

push(exp[i]);

if(exp[i]==')' || exp[i]=='}' || exp[i]==']')

if(top==-1)

{

printf("Right parentheses are more than left parentheses\n");

return 0;

}

else

{

temp=pop();

if(!match(temp, exp[i]))

{

printf("Mismatched parentheses are : ");

printf("%c and %c\n",temp,exp[i]);

return 0;

}

}

}

if(top==-1)

{

printf("Balanced Parentheses\n");

return 1;

}

else

{

printf("Left parentheses more than right parentheses\n");

return 0;

}

}

int match(char a,char b)

{

if(a=='[' && b==']')

return 1;

if(a=='{' && b=='}')

return 1;

if(a=='(' && b==')')

return 1;

return 0;

}

void push(char item)

{

if(top==(MAX-1))

{

printf("Stack Overflow\n");

return;

}

top=top+1;

stack[top]=item;

}

char pop()

{

if(top==-1)

{

printf("Stack Underflow\n");

exit(1);

}

return(stack[top--]);

}

Output:

Conclusion: Thus we were able to get the desired output out of this program

Title: Stack Operations

Introduction: A menu driven program on various operations we perform using the logic of stacks.

Code:

#include <stdio.h>

int stack[100],choice,n,top,x,i;

void push(void);

void pop(void);

void display(void);

int main()

{

//clrscr();

top=-1;

printf("\n Enter the size of STACK:");

scanf("%d",&n);

printf("\n\t STACK OPERATIONS");

printf("\n\t--------------------------------");

printf("\n\t 1.PUSH\n\t 2.POP\n\t 3.DISPLAY\n\t 4.EXIT");

do

{

printf("\n Enter the Choice:");

scanf("%d",&choice);

switch(choice)

{

case 1:

{

push();

break;

}

case 2:

{

pop();

break;

}

case 3:

{

display();

break;

}

case 4:

{

printf("\n\t EXIT POINT ");

break;

}

default:

{

printf ("\n\t Please Enter a Valid Choice(1/2/3/4)");

}

}

}

while(choice!=4);

return 0;

}

void push()

{

if(top>=n-1)

{

printf("\n\tSTACK is over flow");

}

else

{

printf(" Enter a value to be pushed:");

scanf("%d",&x);

top+=1;

stack[top]=x;

}

}

void pop()

{

if(top<=-1)

{

printf("\n\t Stack is under flow");

}

else

{

printf("\n\t The popped elements is %d",stack[top]);

top-=1;

}

}

void display()

{

if(top>=0)

{

printf("\n The elements in STACK \n");

for(i=top; i>=0; i-=1)

printf("\n%d",stack[i]);

printf("\n Press Next Choice");

}

else

{

printf("\n The STACK is empty");

}

}

Output:

Conclusion: Thus we were able to get the desired output out of this program.

Title: Singly Linked List

Introduction: Menu driven program on operations we can perform using the logic of linked list.

Code:

#include <stdio.h>

#include <stdlib.h>

#include <conio.h>

#include <malloc.h>

struct node

{

int data;

struct node \*next;

};

struct node \*start=NULL;

struct node \*create\_11(struct node \*);

struct node \*display(struct node \*);

struct node \*insert\_beg(struct node \*);

struct node \*insert\_end(struct node \*);

struct node \*insert\_after(struct node \*);

struct node \*insert\_before(struct node \*);

struct node \*delete\_beg(struct node \*);

struct node \*delete\_end(struct node \*);

struct node \*delete\_given(struct node \*);

int main()

{

int option;

do

{

printf("\n\n \*\*\*\*\*MAIN MENU \*\*\*\*\*");

printf("\n 1: Create a list");

printf("\n 2: Display the list");

printf("\n 3: Insert at beginning");

printf("\n 4: Insert at ending");

printf("\n 5: Insert after node");

printf("\n 6: Insert before node");

printf("\n 7: Delete at beginning");

printf("\n 8: Delete at ending");

printf("\n 9: Delete at given node");

printf("\n 10: Exit");

printf("\n\n Enter your option : ");

scanf("%d",&option);

switch(option)

{

case 1 : start=create\_11(start);

break;

case 2 : start=display(start);

break;

case 3 : start=insert\_beg(start);

break;

case 4 : start=insert\_end(start);

break;

case 5 : start=insert\_after(start);

break;

case 6 : start=insert\_before(start);

break;

case 7 : start=delete\_beg(start);

break;

case 8 : start=delete\_end(start);

break;

case 9 : start=delete\_given(start);

break;

}

}while(option!=10);

return 0;

}

struct node \*create\_11(struct node \*start)

{

struct node \*new\_node,\*ptr;

int num;

printf("\nEnter -1 to end");

printf("\nEnter the data :");

scanf("%d",&num);

while(num!=-1)

{

new\_node=(struct node \*)malloc(sizeof(struct node));

new\_node -> data=num;

if(start==NULL)

{

new\_node->next=NULL;

start=new\_node;

}

else

{

ptr=start;

while(ptr->next!=NULL)

{

ptr=ptr->next;

}

ptr->next=new\_node;

new\_node->next=NULL;

}

return start;

}

}

struct node \*display(struct node \*start)

{

struct node \*ptr;

ptr=start;

while(ptr!=NULL)

{

printf("%d\t",ptr->data);

ptr=ptr->next;

}

return start;

}

struct node \*insert\_beg(struct node \*start)

{

struct node \*new\_node,\*ptr;

int num;

printf("\nEnter the data :");

scanf("%d",&num);

new\_node=(struct node \*)malloc(sizeof(struct node));

new\_node -> data=num;

new\_node->next=start;

start=new\_node;

return start;

}

struct node \*insert\_end(struct node \*start)

{

struct node \*new\_node,\*ptr;

int num;

printf("\nEnter the data :");

scanf("%d",&num);

new\_node=(struct node \*)malloc(sizeof(struct node));

ptr=start;

while(ptr->next!=NULL)

{

ptr=ptr->next;

}

ptr->next=new\_node;

new\_node -> data=num;

new\_node->next=NULL;

return start;

}

struct node \*insert\_after(struct node \*start)

{

struct node \*new\_node,\*ptr,\*preptr;

int num,x;

printf("\nEnter the data :");

scanf("%d",&num);

printf("\nEnter the number after which it is to be added :");

scanf("%d",&x);

new\_node=(struct node \*)malloc(sizeof(struct node));

ptr=start;

while(preptr->data!=x)

{

preptr=ptr;

ptr=ptr->next;

}

preptr->next=new\_node;

new\_node -> data=num;

new\_node->next=ptr;

return start;

}

struct node \*insert\_before(struct node \*start)

{

struct node \*new\_node,\*ptr,\*preptr;

int num,x;

printf("\nEnter the data :");

scanf("%d",&num);

printf("\nEnter the number before which it is to be added :");

scanf("%d",&x);

new\_node=(struct node \*)malloc(sizeof(struct node));

ptr=start;

while(ptr->data!=x)

{

preptr=ptr;

ptr=ptr->next;

}

preptr->next=new\_node;

new\_node -> data=num;

new\_node->next=ptr;

return start;

}

struct node \*delete\_beg(struct node \*start)

{

struct node \*ptr;

start=start->next;

ptr=start;

return start;

}

struct node \*delete\_end(struct node \*start)

{

struct node \*ptr,\*preptr;

ptr=start;

while(ptr->next!=NULL)

{

preptr=ptr;

ptr=ptr->next;

}

preptr->next=NULL;

return start;

}

struct node \*delete\_given(struct node \*start)

{

struct node \*ptr,\*preptr;

int num;

printf("\nEnter the value to delete :");

scanf("%d",&num);

ptr=start;

while(ptr->data!=num)

{

preptr=ptr;

ptr=ptr->next;

}

preptr->next=ptr->next;

free(ptr);

return start;

}

Output:

Conclusion: Thus we were able to get the desired output out of this program.

Title: Doubly Linked List

Introduction: Menu driven program on operations we can perform using the logic of doubly linked list

Code:

#include <stdio.h>

#include <stdlib.h>

#include <conio.h>

#include <malloc.h>

struct node

{

struct node \*prev;

int data;

struct node \*next;

};

struct node \*start=NULL;

struct node \*create\_11(struct node \*);

struct node \*display(struct node \*);

struct node \*insert\_beg(struct node \*);

struct node \*insert\_end(struct node \*);

struct node \*insert\_after(struct node \*);

struct node \*insert\_before(struct node \*);

struct node \*delete\_beg(struct node \*);

struct node \*delete\_end(struct node \*);

struct node \*delete\_given(struct node \*);

int main()

{

int option;

do

{

printf("\n\n \*\*\*\*\*MAIN MENU \*\*\*\*\*");

printf("\n 1: Create a list");

printf("\n 2: Display the list");

printf("\n 3: Insert at beginning");

printf("\n 4: Insert at ending");

printf("\n 5: Insert after node");

printf("\n 6: Insert before node");

printf("\n 7: Delete at beginning");

printf("\n 8: Delete at ending");

printf("\n 9: Delete at given node");

printf("\n 10: Exit");

printf("\n\n Enter your option : ");

scanf("%d",&option);

switch(option)

{

case 1 : start=create\_11(start);

break;

case 2 : start=display(start);

break;

case 3 : start=insert\_beg(start);

break;

case 4 : start=insert\_end(start);

break;

case 5 : start=insert\_after(start);

break;

case 6 : start=insert\_before(start);

break;

case 7 : start=delete\_beg(start);

break;

case 8 : start=delete\_end(start);

break;

case 9 : start=delete\_given(start);

break;

}

}while(option!=10);

return 0;

}

struct node \*create\_11(struct node \*start)

{

struct node \*new\_node,\*ptr;

int num;

printf("\nEnter -1 to end");

printf("\nEnter the data :");

scanf("%d",&num);

while(num!=-1)

{

new\_node=(struct node \*)malloc(sizeof(struct node));

new\_node -> data=num;

if(start==NULL)

{

new\_node->next=NULL;

start=new\_node;

}

else

{

ptr=start;

while(ptr->next!=NULL)

{

ptr=ptr->next;

}

ptr->next=new\_node;

new\_node->next=NULL;

}

return start;

}

}

struct node \*display(struct node \*start)

{

struct node \*ptr;

ptr=start;

while(ptr!=NULL)

{

printf("%d\t",ptr->data);

ptr=ptr->next;

}

return start;

}

struct node \*insert\_beg(struct node \*start)

{

struct node \*new\_node,\*ptr;

int num;

printf("\nEnter the data :");

scanf("%d",&num);

new\_node=(struct node \*)malloc(sizeof(struct node));

new\_node -> data=num;

new\_node->next=start;

start->prev=new\_node;

start=new\_node;

return start;

}

struct node \*insert\_end(struct node \*start)

{

struct node \*new\_node,\*ptr;

int num;

printf("\nEnter the data :");

scanf("%d",&num);

new\_node=(struct node \*)malloc(sizeof(struct node));

ptr=start;

while(ptr->next!=NULL)

{

ptr=ptr->next;

}

ptr->next=new\_node;

new\_node->prev=ptr;

new\_node -> data=num;

new\_node->next=NULL;

return start;

}

struct node \*insert\_after(struct node \*start)

{

struct node \*new\_node,\*ptr,\*preptr;

int num,x;

printf("\nEnter the data :");

scanf("%d",&num);

printf("\nEnter the number after which it is to be added :");

scanf("%d",&x);

new\_node=(struct node \*)malloc(sizeof(struct node));

ptr=start;

while(preptr->data!=x)

{

preptr=ptr;

ptr=ptr->next;

}

preptr->next=new\_node;

new\_node->prev=preptr;

new\_node -> data=num;

new\_node->next=ptr;

return start;

}

struct node \*insert\_before(struct node \*start)

{

struct node \*new\_node,\*ptr,\*preptr;

int num,x;

printf("\nEnter the data :");

scanf("%d",&num);

printf("\nEnter the number before which it is to be added :");

scanf("%d",&x);

new\_node=(struct node \*)malloc(sizeof(struct node));

ptr=start;

while(ptr->data!=x)

{

preptr=ptr;

ptr=ptr->next;

}

preptr->next=new\_node;

new\_node->prev=preptr;

new\_node -> data=num;

new\_node->next=ptr;

return start;

}

struct node \*delete\_beg(struct node \*start)

{

struct node \*ptr;

start=start->next;

ptr=start;

return start;

}

struct node \*delete\_end(struct node \*start)

{

struct node \*ptr,\*preptr;

ptr=start;

while(ptr->next!=NULL)

{

preptr=ptr;

ptr=ptr->next;

}

preptr->next=NULL;

return start;

}

struct node \*delete\_given(struct node \*start)

{

struct node \*ptr,\*preptr;

int num;

printf("\nEnter the value to delete :");

scanf("%d",&num);

ptr=start;

while(ptr->data!=num)

{

preptr=ptr;

ptr=ptr->next;

}

preptr->next=ptr->next;

ptr->next->prev=preptr;

ptr->next=ptr;

return start;

}

Output:

Conclusion: Thus we were able to get the desired output out of this program.

Title: Linear Queue

Introduction: Menu driven program on the operations we can perform using logic of queues.

Code:

#include <stdio.h>

#define MAX 5

int A[MAX];

int front=-1,rear=-1;

int main()

{

int opt,val;

printf("\n\t\t\tMENU DRIVEN");

do{

printf("\n\n1.Insert an element\n2.Delete an element\n3.Display the queue\n4.Exit");

printf("\n\nEnter an option no. to proceed : ");

scanf("%d",&opt);

switch(opt){

case 1 : printf("Enter an element to insert : ");

scanf("%d",&val);

if(rear==MAX-1)

{

printf("Queue Overflow");

}

else if(front==-1&&rear==-1)

{

front=rear=0;

}

else

{

rear++;

}

A[rear]=val;

break;

case 2 : if(front==-1||front>rear){

printf("Queue Underflow")

}

else

{

val=A[front];

front++;

}

printf("\nThe deleted item is %d",val);

break

case 3 : if(front==-1||front>rear)

{

printf("Queue Underflow");

}

else

{

for(int i=front;i<=rear;i++)

{

printf("\n%d",A[i]);

}

}

break;

}

}while(opt!=4)

Output:

Conclusion: Thus we were able to get the desired output out of this program.

Title: Circular Queue

Introduction: Menu driven program on the operations we can perform using logic of circular queues which are like queues but with last value preceding the first value like in a circle.

Code:

#include <stdio.h>

#define MAX 5

int A[MAX];

int front=-1,rear=-1;

int main()

{

int opt,val;

printf("\n\t\t\tMENU DRIVEN");

do

{

printf("\n\n1.Insert an element\n2.Delete an element\n3.Display the queue\n4.Exit");

printf("\n\nEnter an option no. to proceed : ");

scanf("%d",&opt);

switch(opt)

{

case 1 : printf("Enter an element to insert : ");

scanf("%d",&val);

if(front==0&&rear==MAX-1)

{

printf("Queue Overflow");

}

else if(front==-1&&rear==-1)

{

front=rear=0;

}

else if(rear==MAX-1&&front!=0)

{

rear=0;

}

else

{

rear++;

}

A[rear]=val;

break;

case 2 : if(front==-1||front>rear)

{

printf("Queue Underflow");

}

else if(front==rear)

{

front=rear=-1;

}

else if(front==MAX-1)

{

front=0;

}

else

{

val=A[front];

front++;

}

printf("\nThe deleted item is %d",val);

break;

case 3 : if(front==-1||rear==-1)

{

printf("Queue Underflow");

}

else if(front<rear)

{

for(int i=front;i<=rear;i++)

{

printf("\n%d",A[i]);

}

}

else

{

for(int i=0;i<MAX;i++)

{

printf("\n%d",A[i]);

}

for(int i=0;i<=rear;i++)

{

printf("\n%d",A[i]);

}

}

break;

}

}while(opt!=4);

}

Output:

Conclusion: Thus we were able to get the desired output out of this program.

Title: Binary Tree

Introduction: A menu driven program on creation and traversal of binary tree logic.

Code:

#include <stdio.h>

#include <stdlib.h>

struct node

{

int a;

struct node \*left;

struct node \*right;

};

void generate(struct node \*\*, int);

void infix(struct node \*);

void postfix(struct node \*);

void prefix(struct node \*);

void delete(struct node \*\*);

int main()

{

struct node \*head = NULL;

int choice = 0, num, flag = 0, key;

do

{

printf("\nEnter your choice:\n1. Insert\n2. Traverse via infix\n3.Traverse via prefix\n4. Traverse via postfix\n5. Exit\nChoice: ");

scanf("%d", &choice);

switch(choice)

{

case 1:

printf("Enter element to insert: ");

scanf("%d", &num);

generate(&head, num);

break;

case 2:

infix(head);

break;

case 3:

prefix(head);

break;

case 4:

postfix(head);

break;

case 5:

delete(&head);

printf("Memory Cleared\nPROGRAM TERMINATED\n");

break;

default: printf("Not a valid input, try again\n");

}

} while (choice != 5);

return 0;

}

void generate(struct node \*\*head, int num)

{

struct node \*temp = \*head, \*prev = \*head;

if (\*head == NULL)

{

\*head = (struct node \*)malloc(sizeof(struct node));

(\*head)->a = num;

(\*head)->left = (\*head)->right = NULL;

}

else

{

while (temp != NULL)

{

if (num > temp->a)

{

prev = temp;

temp = temp->right;

}

else

{

prev = temp;

temp = temp->left;

}

}

temp = (struct node \*)malloc(sizeof(struct node));

temp->a = num;

if (num >= prev->a)

{

prev->right = temp;

}

else

{

prev->left = temp;

}

}

}

void infix(struct node \*head)

{

if (head)

{

infix(head->left);

printf("%d ", head->a);

infix(head->right);

}

}

void prefix(struct node \*head)

{

if (head)

{

printf("%d ", head->a);

prefix(head->left);

prefix(head->right);

}

}

void postfix(struct node \*head)

{

if (head)

{

postfix(head->left);

postfix(head->right);

printf("%d ", head->a);

}

}

void delete(struct node \*\*head)

{

if (\*head != NULL)

{

if ((\*head)->left)

{

delete(&(\*head)->left);

}

if ((\*head)->right)

{

delete(&(\*head)->right);

}

free(\*head);

}

}

Output:

Conclusion: Thus we were able to get the desired output out of this program.

Title: Insertion Sort

Introduction: Applying the method of insertion sort in C.

Code:

#include <stdio.h>

void main()

{

int a[100],pos,n,i,j;

printf("Enter the array size :");

scanf("%d",&n);

printf("\nEnter elements of array :\n");

for(i=0;i<n;i++)

{ scanf("%d",&a[i]);

}

for(i=0;i<n;i++)

{

pos=a[i];

j=i-1;

while(j>=0&&pos<=a[j])

{

a[j+1]=a[j];

j=j-1;

}

a[j+1]=pos;

}

printf("\n\nThe sorted array is :\n");

for(i=0;i<n;i++)

{

printf("%d\t",a[i]);

}

Output:

Conclusion: Thus we were able to get the desired output out of this program.

Title: Selection Sort

Introduction: Applying the method of insertion sort in C.

Code:

#include <stdio.h>

void main()

{

int a[100],pos,n,i,j,flag;

printf("Enter the array size :");

scanf("%d",&n);

printf("\nEnter elements of array :\n");

for(i=0;i<n;i++)

{

scanf("%d",&a[i]);

}

for(i=0;i<n;i++)

{

pos=i;

for(j=i+1;j<n;j++)

{

if(a[pos]>a[j])

{

pos=j;

}

}

if(pos!=i)

{

flag=a[i];

a[i]=a[pos];

a[pos]=flag;

}

}

printf("\n\nThe sorted array is :\n");

for(i=0;i<n;i++)

{

printf("%d\t",a[i]);

}

}

Output:

Conclusion: Thus we were able to get the desired output out of this program.

Title: DFS

Introduction: Depth-first search is an algorithm for traversing or searching tree or graph data structures.

Code:

#include <stdio.h>

#include <stdlib.h>

int visited[7] = {0,0,0,0,0,0,0};

int A [7][7]={

{0,1,1,1,0,0,0},

{1,0,1,0,0,0,0},

{1,1,0,1,1,0,0},

{1,0,1,0,1,0,0},

{0,0,1,1,0,1,1},

{0,0,0,0,1,0,0},

{0,0,0,0,1,0,0}

};

void DFS(int i)

{

printf("%d ", i);

visited[i] = 1;

for (int j = 0; j < 7; j++)

{

if(A[i][j]==1 && !visited[j])

{

DFS(j);

}

}

}

int main()

{

DFS(0);

}

Output:

Conclusion: Thus we were able to get the desired output out of this program.

Title: BFS

Introduction: Breadth-first search is an algorithm for traversing or searching tree or graph data structures.

Code:

#include<stdio.h>

#include<stdlib.h>

struct queue

{

int size;

int f;

int r;

int\* arr;

};

int isEmpty(struct queue \*q){

if(q->r==q->f){

return 1;

}

return 0;

}

int isFull(struct queue \*q){

if(q->r==q->size-1){

return 1;

}

return 0;

}

void enqueue(struct queue \*q, int val){

if(isFull(q)){

printf("This Queue is full\n");

}

else{

q->r++;

q->arr[q->r] = val;

// printf("Enqued element: %d\n", val);

}

}

int dequeue(struct queue \*q){

int a = -1;

if(isEmpty(q)){

printf("This Queue is empty\n");

}

else{

q->f++;

a = q->arr[q->f];

}

return a;

}

int main(){

// Initializing Queue (Array Implementation)

struct queue q;

q.size = 400;

q.f = q.r = 0;

q.arr = (int\*) malloc(q.size\*sizeof(int));

// BFS Implementation

int node;

int i = 1;

int visited[7] = {0,0,0,0,0,0,0};

int a [7][7] = {

{0,1,1,1,0,0,0},

{1,0,1,0,0,0,0},

{1,1,0,1,1,0,0},

{1,0,1,0,1,0,0},

{0,0,1,1,0,1,1},

{0,0,0,0,1,0,0},

{0,0,0,0,1,0,0}

};

printf("%d", i);

visited[i] = 1;

enqueue(&q, i); // Enqueue i for exploration

while (!isEmpty(&q))

{

int node = dequeue(&q);

for (int j = 0; j < 7; j++)

{

if(a[node][j] ==1 && visited[j] == 0){

printf("%d", j);

visited[j] = 1;

enqueue(&q, j);

}

}

}

}

Output:

Conclusion: Thus we were able to get the desired output out of this program.