**DEPARTMENT OF COMPUTER APPLICATION**

**TKM COLLEGE OF ENGINEERING**

**KOLLAM – 691005**

Logo

Description automatically generated

**20MCA135 – DATA STRUCTURES LAB**

PRACTICAL RECORD BOOK

First Semester MCA

2021-2022

**Submitted by:**

NAME : KANNAN R R

REG NO : TKM21MCA-2026

**DEPARTMENT OF COMPUTER APPLICATION TKM COLLEGE OF ENGINEERING**

**KOLLAM – 691005**

Logo

Description automatically generated with medium confidence

**Certificate**

This is a bonafide record of the work done by KANNAN R R (TKM21MCA-2026) in the First Semester in Data Structures Lab Course(20MCA135) towards the partial fulfillment of the degree of Master of Computer Applications during the academic year 2021-2022

Staff Member in-charge Examiner

……………………….. ……………………….

**INDEX**

|  |  |  |  |
| --- | --- | --- | --- |
| **Program No :** | **Programs** | | **Page No :** |
|  | **CO1** | |  |
| 1 | 1.1 | Write a C program to Merge two sorted array |  |
| 2 | 1.2 | Write a C program to implement Stack Operations |  |
| 3 | 1.3 | Write a C program to implement Queue Operations |  |
| 4 | 1.4 | Write a C program to implement Linked stack |  |
| 5 | 1.5 | Write a C program to implement Singly Linked List Operations |  |
| 6 | 1.6 | Write a C program to implement Doubly Linked List Operations. |  |
| 7 | 1.7 | Write a C program to implement Binary Search Tree. |  |
|  | **CO2** | |  |
| 8 | 2.1 | Write a C program to implement BitString Operations. |  |
|  | **CO3** | |  |
| 9 | 3.1 | Write a C program to implement Red Black Tree Operations. |  |
| 10 | 3.2 | Write a C program to implement B-Tree Operations. |  |
|  | **C04** | |  |
| 11 | 4.1 | Write a C program to implement Binomial Heap. |  |
|  | **CO5** | |  |
| 12 | 5.1 | Write a C program to implement Depth first Search. |  |
| 13 | 5.2 | Write a C program to implement Breadth first Search. |  |
| 14 | 5.3 | Write a C program to implement Kruskal’s algorithm. |  |
| 15 | 5.4 | Write a C program to implement Prim’s Algorithm |  |
| 16 | 5.5 | Write a C program to implement Topological Sort |  |
| 17 | 5.6 | Write a C program to implement Dijkstra’s Algorithm |  |

**PROGRAM NO : 1**

**AIM :** Write a C program to Merge two sorted array.

**CODE :**

#include<stdio.h>

int main()

{

int i,j,m,n,k;

int arr1[10],arr2[10],res[10];

printf("Enter size of first array : ");

scanf("%d",&m);

printf("Enter the size of second array : ");

scanf("%d",&n);

printf("Enter a sorted array\n");

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

{

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

}

printf("Enter a sorted array\n");

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

{

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

}

printf("First array : ");

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

{

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

}

printf("\n");

printf("Second array : ");

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

{

printf("%d\t",arr2[i]);

}

i=0;

j=0;

k=0;

printf("\n");

while((i<m)&&(j<n))

{

if(arr1[i]<arr2[j])

{

res[k]=arr1[i];

i++;k++;

}

else

{

res[k]=arr2[j];

j++;k++;

}

}

while(i<m)

{

res[k]=arr1[i++];

}

while(j<n)

{

res[k]=arr2[j++];

}

printf("Merged array : ");

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

{

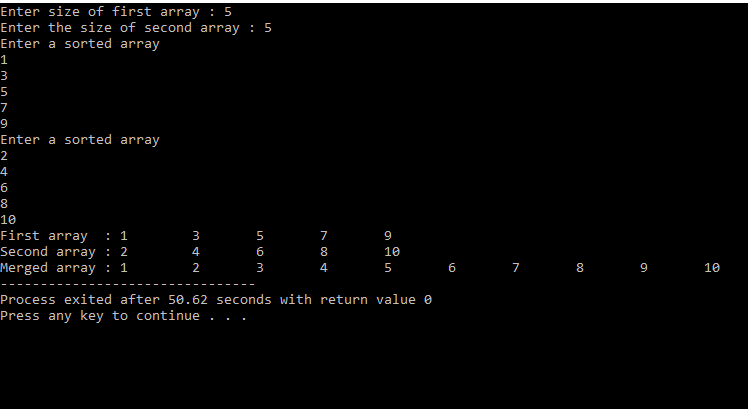
printf("%d\t",res[i]);

}

return 0;

}

**OUTPUT :**



**RESULT :** The program was executed successfully and output obtained

**PROGRAM NO : 2**

**AIM :** Write a C program to implement Stack Operations .

**CODE :**

#include<stdio.h>

#define n 5

int s[n],top=-1;

void push();

void pop();

void Top();

void display();

void main() {

int o,c=1;

while(c==1) {

printf("Enter any of the below option number\n1.push\n2.pop\n3.top\n4.display\n");

scanf("%d",&o);

switch(o) {

case 1 : push();

break;

case 2 : pop();

break;

case 3 : Top();

break;

case 4 : display();

break;

}

printf("Do you want to continue(0/1)\n");

scanf("%d",&c);

}

}

void push() {

int x;

printf("Enter an element to push\n");

scanf("%d",&x);

if(top==n-1) {

printf("\nOverflow\n");

} else {

top++;

s[top]=x;

}

}

void pop() {

if(top==-1) {

printf("\nUnderflow\n");

} else {

printf("Popped element is %d",s[top]);

top--;

}

}

void Top() {

if(top==-1) {

printf("\nUnderflow\n");

} else {

printf("Top element is %d",s[top]);

}

}

void display() {

if(top==-1) {

printf("\nUnderflow\n");

} else {

printf("Stack elements are\n");

for (int i=top; i>=0; i--) {

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

}

}

}

**OUTPUT :**

Text

Description automatically generated

**RESULT :** The program was executed successfully and output obtained

**PROGRAM NO : 3**

**AIM :** Write a C program to implement Queue Operations

**CODE :**

#include<stdio.h>

#define n 5

int q[n],front=-1,rear=-1;

void insert();

void delete();

void peak();

void display();

void main() {

int o,c=1;

while(c==1) {

printf("Enter any of the below option number\n1.Insert\n2.Delete\n3.Peak\n4.Display\n");

scanf("%d",&o);

switch(o) {

case 1 : insert();

break;

case 2 : delete();

break;

case 3 : peak();

break;

case 4 : display();

break;

default : printf("Invalid entry");

}

printf("Do you want to continue(0/1)\n");

scanf("%d",&c);

}

}

void insert() {

int x;

printf("Enter an element to insert\n");

scanf("%d",&x);

if(rear==n-1) {

printf("\nOverflow\n");

//return;

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

front=rear=0;

} else {

rear++;

}

q[rear]=x;

}

void delete() {

if(front==-1 || front>rear) {

printf("\nUnderflow\n");

} else {

printf("Deleted element is %d",q[front]);

front++;

if(front>rear) {

front=rear=-1;

}

}

}

void peak() {

if(front==-1 || front>rear) {

printf("\nUnderflow\n");

} else {

printf("Peak element is %d",q[front]);

}

}

void display() {

if(front==-1 || front>rear) {

printf("\nUnderflow\n");

} else {

printf("Queue elements are\n");

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

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

}

}}

**OUTPUT :**

Text

Description automatically generated

**RESULT :** The program was executed successfully and output obtained

**PROGRAM NO : 4**

**AIM :** Write a C program to implement Linked Stack.

**CODE :**

#include<stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node \*next;

};

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

void push()

{ printf("Enter the data ");

scanf("%d",&newnode->data);

newnode->next=head;

head=newnode; }

void pop()

{head=head->next; }

void display()

{

temp=head;

int count=0;

while(temp!=0)

{ printf("%d\n",temp->data);

count++;

temp=temp->next; }}

void addnode()

{

int o=1,c=0;

head=0;

while(o==1)

{

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

printf("Enter the data ");

scanf("%d",&newnode->data);

newnode->next=0;

if(head==0)

{head=temp=newnode; }

else

{ temp->next=newnode;

temp=newnode; }

printf("Do you want to continue insertion (0/1) ");

scanf("%d",&o); }}

void main()

{

int c=1,o,i,m;

while(c==1)

{

printf("Enter any of the below option number\n1.AddNode\n2.Push\n3.Pop\n4.Dispaly\n");

scanf("%d",&o);

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

switch(o)

{ case 1 : addnode();

break;

case 2 : push();

break;

case 3 : pop();

break;

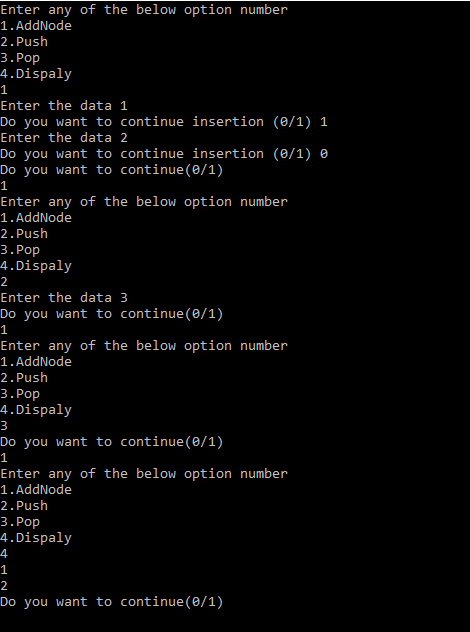
case 4 : display();

break; }

printf("Do you want to continue(0/1)\n");

scanf("%d",&c); }}

**OUTPUT :**



**RESULT :** The program was executed successfully and output obtained

**PROGRAM NO : 5**

**AIM :** Write a C program to implement Singly Linked List Operations

**CODE :**

#include<stdio.h>

#include<stdlib.h>

struct node {

int data;

struct node \*next;};

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

void InsertBeg() {

int m;

printf("Enter value to insert");

scanf("%d",&m);

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

newnode->data=m;

newnode->next=head;

head=newnode;

}

void InsertBetween() {

int m,x;

printf("Enter value to insert");

scanf("%d",&m);

printf("Enter after which value to insert");

scanf("%d",&x);

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

newnode->data=m;

temp=head;

while(temp->next!=0) {

if(temp->data==x) {

break;}

temp=temp->next;}

newnode->next=temp->next;

temp->next=newnode;}

void InsertEnd() {

int m;

printf("Enter value to insert");

scanf("%d",&m);

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

newnode->data=m;

while(temp->next!=0) {

temp=temp->next;}

temp->next=newnode;

newnode->next=0;}

void DeleteBeg() { head=head->next;}

void DeleteBetween() {

int x;

printf("Enter the node data for position");

scanf("%d",&x);

temp=head;

while(temp->data!=x) {

temp=temp->next;}

temp->next=temp->next->next;}

void DeleteEnd() {

temp=head;

while(temp->next!=0) {

prev=temp;

temp=temp->next;}

prev->next=NULL;}

void display() {

temp=head;

while(temp!=0) {

printf("%d\n",temp->data);

temp=temp->next;}}

void addnode() {

int o=1,c=0,m;

printf("Enter value to insert");

scanf("%d",&m);

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

newnode->data=m;

newnode->next=0;

if(head==0) {

head=temp=newnode;

} else {

temp->next=newnode;

temp=newnode;}}

void main() {

int c=1,o,i,m;

head=0;

while(c==1) {

printf("Enter any of the below option number\n1.AddNode\n2.Insert-Begining\n3.Insert-End\n4.Insert-Between\n5.Delete-Begining\n6.Delete-End\n7.Delete-Between\n8.Dispaly\n");

scanf("%d",&o);

switch(o) {

case 1 : addnode();break;

case 2 : InsertBeg();break;

case 3 : InsertBetween();break;

case 4 : InsertEnd();break;

case 5 : DeleteEnd();break;

case 6 : DeleteEnd();break;

case 7 : DeleteEnd();break;

case 8 : display();break;

}

printf("Do you want to continue(0/1)\n");

scanf("%d",&c);

}

}

**OUTPUT :**

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

**RESULT :** The program was executed successfully and output obtained

**PROGRAM NO : 6**

**AIM :** Write a C program to implement Doubly Linked List Operations .

**CODE :**

#include<stdio.h>

#include<stdlib.h>

struct node {

struct node \*prev;

int data;

struct node \*next;};

struct node \*head,\*newnode,\*temp,\*ptr;

void Beg() {

newnode->prev = head;

head=newnode;

newnode->next = head->prev;}

void Between() {

int x;

temp=head;

printf("Enter the node data for position");

scanf("%d",&x);

while(temp->data!=x) {

temp=temp->next;}

newnode->next=temp->next;

temp->next=newnode;

temp->next->prev=newnode;}

void End() {

temp=head;

while(temp->next!=NULL) {

temp=temp->next;}

temp->next=newnode;

newnode->prev=temp;}

void DelBeg() {head=head->next;}

void DelBetween() {

int x;

temp=head;

printf("Enter the node data for position");

scanf("%d",&x);

while(temp->data!=x) {

temp=temp->next;}

printf("asa");

ptr=temp;

temp->next->next->prev=ptr;

temp->next=ptr->next->next;}

void DelEnd() {

temp=head;

while(temp->next!=NULL) {

temp=temp->next;}

temp->next=newnode;

newnode->prev=temp;}

void display() {

temp=head;

int count=0;

while(temp!=0) {

printf("%d\n",temp->data);

count++;

temp=temp->next;}}

void addnode() {

int o=1,c=0;

head=0;

while(o==1) {

if(head==0) {

head=temp=newnode;

} else {

temp->next=newnode;

newnode->prev=temp;

temp=newnode;}

printf("Do you want to continue insertion (0/1) ");

scanf("%d",&o);}}

void CreateNode() {

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

printf("Enter the data ");

scanf("%d",&newnode->data);

newnode->next=NULL;

newnode->prev=NULL;}

void main() {

int c=1,o,i,m;

while(c==1) {

printf("Enter any of the below option number\n1.AddNode\n2.Insert-Begining\n3.Insert-End\n4.Insert-Between\n5.Delete-Begining\n6.Delete-End\n7.Delete-Between\n8.Dispaly\n");

scanf("%d",&o);

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

switch(o) {

case 1 : CreateNode();

addnode();

break;

case 2 : CreateNode();

Beg(m);

break;

case 3 : CreateNode();

End(m); break;

case 4 : CreateNode();

Between();break;

case 5 : DelBeg();break;

case 6 : DelEnd();break;

case 7 : DelBetween();break;

case 8 : display();break;

}

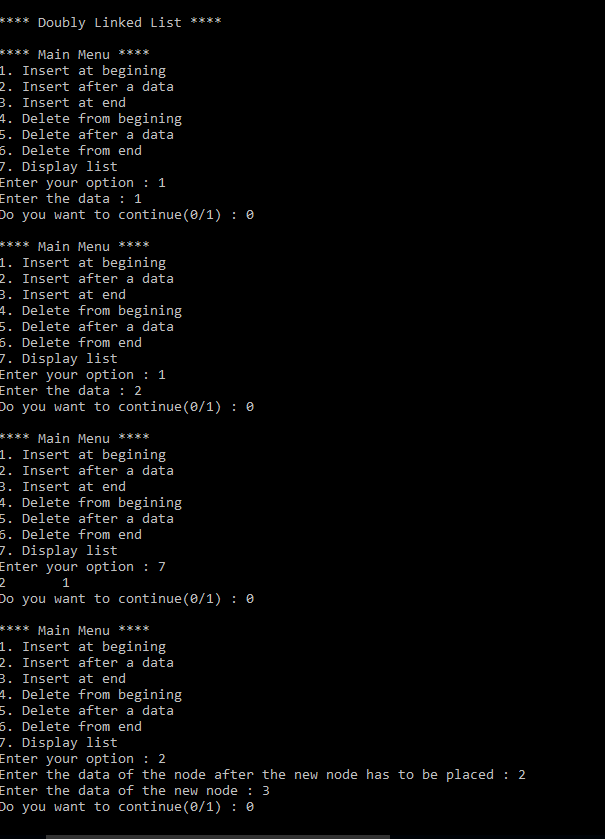
printf("Do you want to continue(0/1)\n");

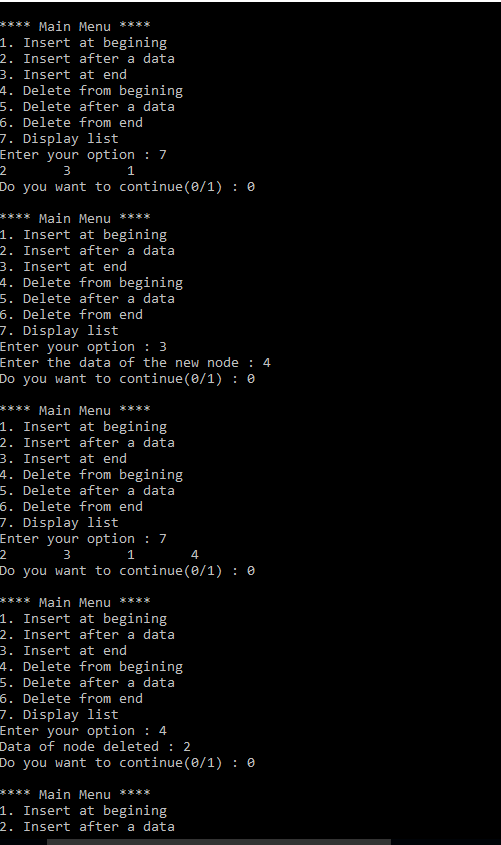
scanf("%d",&c);

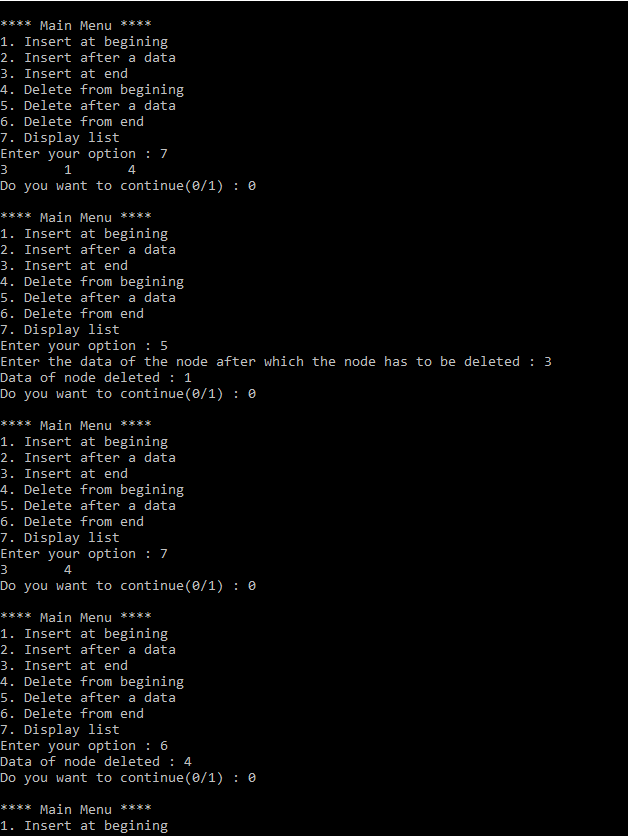
}

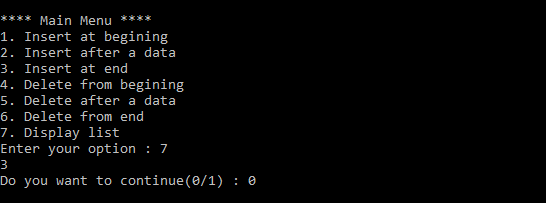
}

**OUTPUT :**

****

****

****

****

**RESULT :** The program was executed successfully and output obtained

**PROGRAM NO : 7**

**AIM :** Write a C program to implement Binary Search Tree .

**CODE :**

#include <stdio.h>

#include <stdlib.h>

struct node {

int data;

struct node \*left, \*right;};

void inorder(struct node \*r) {

if(r!=NULL) {

inorder(r->left);

printf("%d ",r->data);

inorder(r->right);}}

void min(struct node \*r) {

struct node \*pre;

while(pre!= NULL && r->left!=NULL) {

pre=r;

r=r->left;

}

printf("\nMin element is : %d",r->data);}

void max(struct node \*r) {

struct node \*pre;

while(pre!=NULL && r->right!=NULL) {

pre=r;

r=r->right;}

printf("\nMax element is : %d",r->data);}

void insuc(struct node \*r,int x) {

struct node \*temp=r,\*l;

struct node \*n;

while(temp->right!=NULL || temp->left!=NULL) {

if(x==temp->data) {

n=temp;

break;

} else if(x> temp->data) {

temp=temp->right;

} else {

l=temp;

temp=temp->left;}}

if(temp->right!=NULL) {

temp=temp->right;

while(temp->left!=NULL) {

temp=temp->left; }

printf("\nInorder succesor of %d is %d",x,temp->data);

} else {

printf("\nInorder succesor of %d is %d",x,l->data);}}

void search(struct node \*r,int x) {

int f=0;

struct node \*pre=r;

while(pre->right!=NULL || pre->left!=NULL) {

if(x==r->data) {

pre=r;

f=1;

break;

} else if(x> r->data) {

pre=r;

r=r->right;

} else {

pre=r;

r=r->left;}}

if(f==1) {

printf("Found");

} else {

printf("Not Found");

}}

void main() {

int n, i, item,x;

struct node \*new, \*temp, \*root;

printf("Enter the number of elements\n");

scanf("%d", &n);

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

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

new->right = NULL;

new->left = NULL;

printf("Enter the data \n");

scanf("%d", &item);

new->data = item;

if (i == 0) {root = new;

} else {

temp = root;

while (1) {

if (item > temp->data) {

if (temp->right != NULL) {

temp = temp->right;

} else {

temp->right = new;

break; }

} else {

if (temp->left != NULL) {

temp = temp->left;

} else {

temp->left = new;

break; }}}

}}

inorder(root);

printf("\nEnter the data to search \n");

scanf("%d",&x);

search(root,x);

min(root);

max(root);

int v=0;

printf("Enter an element to get inorder succesor\n");

scanf("%d",&v);

insuc(root,v);}

**OUTPUT:**

Text

Description automatically generated

**RESULT :** The program was executed successfully and output obtained

**PROGRAM NO : 8**

**AIM :** Write a C program to implement BitString Operations .

**CODE :**

#include<stdio.h>

int x,y,z,a[10],b[10],b1[10],b2[10],b2c[10],u[10],d=0,o=1;

void Union() {

printf("\nA Union B : ");

for (int i=0;i<10;i++) {

printf("%d",b1[i] | b2[i]);

}

}

void Intersection() {

printf("\nA Intersection B : ");

for (int i=0;i<10;i++) {

printf("%d",b1[i] \* b2[i]);

}

}

void Diff() {

for (int i=0;i<10;i++) {

if(b2[i]==0) {

b2c[i]=1;

} else {

b2c[i]=0;

}

}

printf("\nA - B : ");

for (int i=0;i<10;i++) {

printf("%d",b1[i] \* b2c[i]);

}

}

void main() {

printf("Enter the number of elements in set U\n");

scanf("%d",&x);

printf("Enter the elements in set U\n");

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

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

}

printf("Enter the number of elements in set A\n");

scanf("%d",&y);

printf("Enter the elements in set A\n");

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

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

}

printf("Enter the number of elements in set B\n");

scanf("%d",&z);

printf("Enter the elements in set B\n");

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

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

}

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

for (int j=0;j<y;j++) {

if(a[j]==u[i]) {

b1[i]=1;

break;

} else {

b1[i]=0;

}

}

}

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

for (int j=0;j<z;j++) {

if(b[j]==u[i]) {

b2[i]=1;

break;

} else {

b2[i]=0;

}

}

}

printf("\nA : ");

for (int i=0;i<10;i++) {

printf("%d",b1[i]);

}

printf("\nB : ");

for (int i=0;i<10;i++) {

printf("%d",b2[i]);

}

while(o==1) {

printf("\nEnter any option \n1.Union\n2.Intersection\n3.Difference\n");

scanf("%d",&d);

switch(d) {

case 1 : Union();

printf("\n");

break;

case 2 : Intersection();

printf("\n");

break;

case 3 : Diff();

printf("\n");

break;

}

printf("Continue(0/1)");

scanf("%d",&o);

}

}

**OUTPUT :**

Text

Description automatically generated

**RESULT :** The program was executed successfully and output obtained

**PROGRAM NO : 9**

**AIM :** Write a C program to implement Red Black Tree Operations .

**CODE :**

#include <stdio.h>

#include <stdlib.h>

enum nodeColor {

RED,

BLACK

};

struct rbNode {

int data, color;

struct rbNode \*link[2];

}

;

struct rbNode \*root = NULL;

// Create a red-black tree

struct rbNode \*createNode(int data) {

struct rbNode \*newnode;

newnode = (struct rbNode \*)malloc(sizeof(struct rbNode));

newnode->data = data;

newnode->color = RED;

newnode->link[0] = newnode->link[1] = NULL;

return newnode;

}

// Insert an node

void insertion(int data) {

struct rbNode \*stack[98], \*ptr, \*newnode, \*xPtr, \*yPtr;

int dir[98], ht = 0, index;

ptr = root;

if (!root) {

root = createNode(data);

return;

}

stack[ht] = root;

dir[ht++] = 0;

while (ptr != NULL) {

if (ptr->data == data) {

printf("Duplicates Not Allowed!!\n");

return;

}

index = (data - ptr->data) > 0 ? 1 : 0;

stack[ht] = ptr;

ptr = ptr->link[index];

dir[ht++] = index;

}

stack[ht - 1]->link[index] = newnode = createNode(data);

while ((ht >= 3) && (stack[ht - 1]->color == RED)) {

if (dir[ht - 2] == 0) {

yPtr = stack[ht - 2]->link[1];

if (yPtr != NULL && yPtr->color == RED) {

stack[ht - 2]->color = RED;

stack[ht - 1]->color = yPtr->color = BLACK;

ht = ht - 2;

} else {

if (dir[ht - 1] == 0) {

yPtr = stack[ht - 1];

} else {

xPtr = stack[ht - 1];

yPtr = xPtr->link[1];

xPtr->link[1] = yPtr->link[0];

yPtr->link[0] = xPtr;

stack[ht - 2]->link[0] = yPtr;

}

xPtr = stack[ht - 2];

xPtr->color = RED;

yPtr->color = BLACK;

xPtr->link[0] = yPtr->link[1];

yPtr->link[1] = xPtr;

if (xPtr == root) {

root = yPtr;

} else {

stack[ht - 3]->link[dir[ht - 3]] = yPtr;

}

break;

}

} else {

yPtr = stack[ht - 2]->link[0];

if ((yPtr != NULL) && (yPtr->color == RED)) {

stack[ht - 2]->color = RED;

stack[ht - 1]->color = yPtr->color = BLACK;

ht = ht - 2;

} else {

if (dir[ht - 1] == 1) {

yPtr = stack[ht - 1];

} else {

xPtr = stack[ht - 1];

yPtr = xPtr->link[0];

xPtr->link[0] = yPtr->link[1];

yPtr->link[1] = xPtr;

stack[ht - 2]->link[1] = yPtr;

}

xPtr = stack[ht - 2];

yPtr->color = BLACK;

xPtr->color = RED;

xPtr->link[1] = yPtr->link[0];

yPtr->link[0] = xPtr;

if (xPtr == root) {

root = yPtr;

} else {

stack[ht - 3]->link[dir[ht - 3]] = yPtr;

}

break;

}

}

}

root->color = BLACK;

}

// Delete a node

void deletion(int data) {

struct rbNode \*stack[98], \*ptr, \*xPtr, \*yPtr;

struct rbNode \*pPtr, \*qPtr, \*rPtr;

int dir[98], ht = 0, diff, i;

enum nodeColor color;

if (!root) {

printf("Tree not available\n");

return;

}

ptr = root;

while (ptr != NULL) {

if ((data - ptr->data) == 0)

break;

diff = (data - ptr->data) > 0 ? 1 : 0;

stack[ht] = ptr;

dir[ht++] = diff;

ptr = ptr->link[diff];

}

if (ptr->link[1] == NULL) {

if ((ptr == root) && (ptr->link[0] == NULL)) {

free(ptr);

root = NULL;

} else if (ptr == root) {

root = ptr->link[0];

free(ptr);

} else {

stack[ht - 1]->link[dir[ht - 1]] = ptr->link[0];

}

} else {

xPtr = ptr->link[1];

if (xPtr->link[0] == NULL) {

xPtr->link[0] = ptr->link[0];

color = xPtr->color;

xPtr->color = ptr->color;

ptr->color = color;

if (ptr == root) {

root = xPtr;

} else {

stack[ht - 1]->link[dir[ht - 1]] = xPtr;

}

dir[ht] = 1;

stack[ht++] = xPtr;

} else {

i = ht++;

while (1) {

dir[ht] = 0;

stack[ht++] = xPtr;

yPtr = xPtr->link[0];

if (!yPtr->link[0])

break;

xPtr = yPtr;

}

dir[i] = 1;

stack[i] = yPtr;

if (i > 0)

stack[i - 1]->link[dir[i - 1]] = yPtr;

yPtr->link[0] = ptr->link[0];

xPtr->link[0] = yPtr->link[1];

yPtr->link[1] = ptr->link[1];

if (ptr == root) {

root = yPtr;

}

color = yPtr->color;

yPtr->color = ptr->color;

ptr->color = color;

}

}

if (ht < 1)

return;

if (ptr->color == BLACK) {

while (1) {

pPtr = stack[ht - 1]->link[dir[ht - 1]];

if (pPtr && pPtr->color == RED) {

pPtr->color = BLACK;

break;

}

if (ht < 2)

break;

if (dir[ht - 2] == 0) {

rPtr = stack[ht - 1]->link[1];

if (!rPtr)

break;

if (rPtr->color == RED) {

stack[ht - 1]->color = RED;

rPtr->color = BLACK;

stack[ht - 1]->link[1] = rPtr->link[0];

rPtr->link[0] = stack[ht - 1];

if (stack[ht - 1] == root) {

root = rPtr;

} else {

stack[ht - 2]->link[dir[ht - 2]] = rPtr;

}

dir[ht] = 0;

stack[ht] = stack[ht - 1];

stack[ht - 1] = rPtr;

ht++;

rPtr = stack[ht - 1]->link[1];

}

if ((!rPtr->link[0] || rPtr->link[0]->color == BLACK) &&

(!rPtr->link[1] || rPtr->link[1]->color == BLACK)) {

rPtr->color = RED;

} else {

if (!rPtr->link[1] || rPtr->link[1]->color == BLACK) {

qPtr = rPtr->link[0];

rPtr->color = RED;

qPtr->color = BLACK;

rPtr->link[0] = qPtr->link[1];

qPtr->link[1] = rPtr;

rPtr = stack[ht - 1]->link[1] = qPtr;

}

rPtr->color = stack[ht - 1]->color;

stack[ht - 1]->color = BLACK;

rPtr->link[1]->color = BLACK;

stack[ht - 1]->link[1] = rPtr->link[0];

rPtr->link[0] = stack[ht - 1];

if (stack[ht - 1] == root) {

root = rPtr;

} else {

stack[ht - 2]->link[dir[ht - 2]] = rPtr;

}

break;

}

} else {

rPtr = stack[ht - 1]->link[0];

if (!rPtr)

break;

if (rPtr->color == RED) {

stack[ht - 1]->color = RED;

rPtr->color = BLACK;

stack[ht - 1]->link[0] = rPtr->link[1];

rPtr->link[1] = stack[ht - 1];

if (stack[ht - 1] == root) {

root = rPtr;

} else {

stack[ht - 2]->link[dir[ht - 2]] = rPtr;

}

dir[ht] = 1;

stack[ht] = stack[ht - 1];

stack[ht - 1] = rPtr;

ht++;

rPtr = stack[ht - 1]->link[0];

}

if ((!rPtr->link[0] || rPtr->link[0]->color == BLACK) &&

(!rPtr->link[1] || rPtr->link[1]->color == BLACK)) {

rPtr->color = RED;

} else {

if (!rPtr->link[0] || rPtr->link[0]->color == BLACK) {

qPtr = rPtr->link[1];

rPtr->color = RED;

qPtr->color = BLACK;

rPtr->link[1] = qPtr->link[0];

qPtr->link[0] = rPtr;

rPtr = stack[ht - 1]->link[0] = qPtr;

}

rPtr->color = stack[ht - 1]->color;

stack[ht - 1]->color = BLACK;

rPtr->link[0]->color = BLACK;

stack[ht - 1]->link[0] = rPtr->link[1];

rPtr->link[1] = stack[ht - 1];

if (stack[ht - 1] == root) {

root = rPtr;

} else {

stack[ht - 2]->link[dir[ht - 2]] = rPtr;

}

break;

}

}

ht--;

}

}

}

// Print the inorder traversal of the tree

void inorderTraversal(struct rbNode \*node) {

if (node) {

inorderTraversal(node->link[0]);

printf("%d ", node->data);

inorderTraversal(node->link[1]);

}

return;

}

int main() {

int ch, data;

while (1) {

printf("1. Insertion\t2. Deletion\n");

printf("3. Traverse\t4. Exit");

printf("\nEnter your choice:");

scanf("%d", &ch);

switch (ch) {

case 1:

printf("Enter the element to insert:");

scanf("%d", &data);

insertion(data);

break;

case 2:

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

scanf("%d", &data);

deletion(data);

break;

case 3:

inorderTraversal(root);

printf("\n");

break;

case 4:

exit(0);

default:

printf("Not available\n");

break;

}

printf("\n");

} return 0;}

**OUTPUT :**

Text

Description automatically generated

**RESULT :** The program was executed successfully and output obtained

**PROGRAM NO : 10**

**AIM :** Write a C program to implement B-Tree Operations .

**CODE :**

#include <stdio.h>

#include <stdlib.h>

#define MAX 3

#define MIN 2

struct BTreeNode {

int val[MAX + 1], count;

struct BTreeNode \*link[MAX + 1];

}

;

struct BTreeNode \*root;

struct BTreeNode \*createNode(int val, struct BTreeNode \*child) {

struct BTreeNode \*newNode;

newNode = (struct BTreeNode \*)malloc(sizeof(struct BTreeNode));

newNode->val[1] = val;

newNode->count = 1;

newNode->link[0] = root;

newNode->link[1] = child;

return newNode;

}

void insertNode(int val, int pos, struct BTreeNode \*node,

struct BTreeNode \*child) {

int j = node->count;

while (j > pos) {

node->val[j + 1] = node->val[j];

node->link[j + 1] = node->link[j];

j--;

}

node->val[j + 1] = val;

node->link[j + 1] = child;

node->count++;

}

void splitNode(int val, int \*pval, int pos, struct BTreeNode \*node,

struct BTreeNode \*child, struct BTreeNode \*\*newNode) {

int median, j;

if (pos > MIN)

median = MIN + 1; else

median = MIN;

\*newNode = (struct BTreeNode \*)malloc(sizeof(struct BTreeNode));

j = median + 1;

while (j <= MAX) {

(\*newNode)->val[j - median] = node->val[j];

(\*newNode)->link[j - median] = node->link[j];

j++;

}

node->count = median;

(\*newNode)->count = MAX - median;

if (pos <= MIN) {

insertNode(val, pos, node, child);

} else {

insertNode(val, pos - median, \*newNode, child);

}

\*pval = node->val[node->count];

(\*newNode)->link[0] = node->link[node->count];

node->count--;

}

int setValue(int val, int \*pval,

struct BTreeNode \*node, struct BTreeNode \*\*child) {

int pos;

if (!node) {

\*pval = val;

\*child = NULL;

return 1;

}

if (val < node->val[1]) {

pos = 0;

} else {

for (pos = node->count;

(val < node->val[pos] && pos > 1); pos--)

;

if (val == node->val[pos]) {

printf("Duplicates are not permitted\n");

return 0;

}

}

if (setValue(val, pval, node->link[pos], child)) {

if (node->count < MAX) {

insertNode(\*pval, pos, node, \*child);

} else {

splitNode(\*pval, pval, pos, node, \*child, child);

return 1;

}

}

return 0;

}

void insert(int val) {

int flag, i;

struct BTreeNode \*child;

flag = setValue(val, &i, root, &child);

if (flag)

root = createNode(i, child);

}

void search(int val, int \*pos, struct BTreeNode \*myNode) {

if (!myNode) {

return;

}

if (val < myNode->val[1]) {

\*pos = 0;

} else {

for (\*pos = myNode->count;

(val < myNode->val[\*pos] && \*pos > 1); (\*pos)--)

;

if (val == myNode->val[\*pos]) {

printf("%d is found", val);

return;

}

}

search(val, pos, myNode->link[\*pos]);

return;

}

void traversal(struct BTreeNode \*myNode) {

int i;

if (myNode) {

for (i = 0; i < myNode->count; i++) {

traversal(myNode->link[i]);

printf("%d ", myNode->val[i + 1]);

}

traversal(myNode->link[i]);

}

}

int main() {

int val, ch;

insert(8);

insert(9);

insert(10);

insert(11);

insert(15);

insert(16);

insert(17);

insert(18);

insert(20);

insert(23);

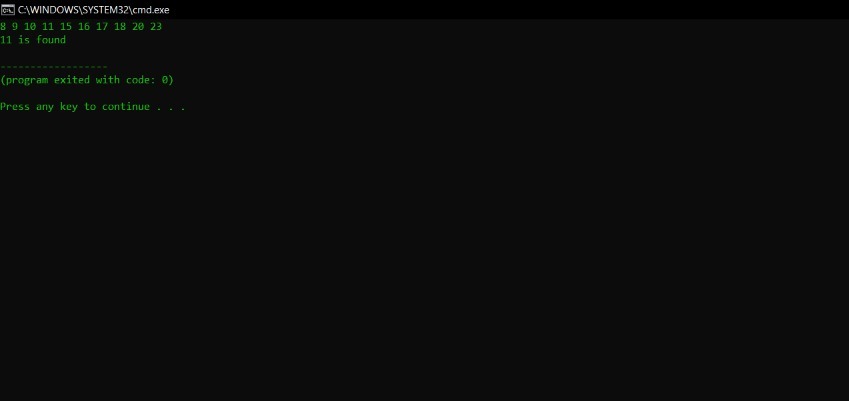
traversal(root);

printf("\n");

search(11, &ch, root);

}

**OUTPUT :**

****

**RESULT :** The program was executed successfully and output obtained

**PROGRAM NO : 11**

**AIM :** Write a C program to implement Binomial Heap.

**CODE :**

#include<stdio.h>

#include<stdlib.h>

struct node {

int n;

int degree;

struct node\* parent;

struct node\* child;

struct node\* sibling;

};

/\* struct node\* MAKE\_bin\_HEAP(); \*/

int bin\_LINK(struct node\*, struct node\*);

struct node\* CREATE\_NODE(int);

struct node\* bin\_HEAP\_UNION(struct node\*, struct node\*);

struct node\* bin\_HEAP\_INSERT(struct node\*, struct node\*);

struct node\* bin\_HEAP\_MERGE(struct node\*, struct node\*);

struct node\* bin\_HEAP\_EXTRACT\_MIN(struct node\*);

int REVERT\_LIST(struct node\*);

int DISPLAY(struct node\*);

struct node\* FIND\_NODE(struct node\*, int);

int bin\_HEAP\_DECREASE\_KEY(struct node\*, int, int);

int bin\_HEAP\_DELETE(struct node\*, int);

int count = 1;

struct node \* H = NULL;

struct node \*Hr = NULL;

int bin\_LINK(struct node\* y, struct node\* z) {

y->parent = z;

y->sibling = z->child;

z->child = y;

z->degree = z->degree + 1;

}

struct node\* CREATE\_NODE(int k) {

struct node\* p;

//new node;

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

p->n = k;

return p;

}

struct node\* bin\_HEAP\_UNION(struct node\* H1, struct node\* H2) {

struct node\* prev\_x;

struct node\* next\_x;

struct node\* x;

struct node\* H = NULL;

H = bin\_HEAP\_MERGE(H1, H2);

if (H == NULL)

return H;

prev\_x = NULL;

x = H;

next\_x = x->sibling;

while (next\_x != NULL) {

if ((x->degree != next\_x->degree) || ((next\_x->sibling != NULL)

&& (next\_x->sibling)->degree == x->degree)) {

prev\_x = x;

x = next\_x;

} else {

if (x->n <= next\_x->n) {

x->sibling = next\_x->sibling;

bin\_LINK(next\_x, x);

} else {

if (prev\_x == NULL)

H = next\_x; else

prev\_x->sibling = next\_x;

bin\_LINK(x, next\_x);

x = next\_x;

}

}

next\_x = x->sibling;

}

return H;

}

struct node\* bin\_HEAP\_INSERT(struct node\* H, struct node\* x) {

struct node\* H1 = NULL;

x->parent = NULL;

x->child = NULL;

x->sibling = NULL;

x->degree = 0;

H1 = x;

H = bin\_HEAP\_UNION(H, H1);

return H;

}

struct node\* bin\_HEAP\_MERGE(struct node\* H1, struct node\* H2) {

struct node\* H = NULL;

struct node\* y;

struct node\* z;

struct node\* a;

struct node\* b;

y = H1;

z = H2;

if (y != NULL) {

if (z != NULL && y->degree <= z->degree)

H = y; else if (z != NULL && y->degree > z->degree)

/\* need some modifications here Don look on it!!! \*/

H = z; else

H = y;

} else

H = z;

while (y != NULL && z != NULL) {

if (y->degree < z->degree) {

y = y->sibling;

} else if (y->degree == z->degree) {

a = y->sibling;

y->sibling = z;

y = a;

} else {

b = z->sibling;

z->sibling = y;

z = b;

}

}

return H;

}

int DISPLAY(struct node\* H) {

//work on display

struct node\* p;

if (H == NULL) {

printf("\nHEAP EMPTY");

return 0;

}

printf("\nTHE ROOT NODES ARE:-\n");

p = H;

while (p != NULL) {

printf("%d", p->n);

if (p->sibling != NULL)

printf("-->");

p = p->sibling;

}

printf("\n");

}

struct node\* bin\_HEAP\_EXTRACT\_MIN(struct node\* H1) {

int min;

struct node\* t = NULL;

struct node\* x = H1;

struct node \*Hr;

struct node\* p;

Hr = NULL;

if (x == NULL) {

printf("\nNOTHING TO EXTRACT");

return x;

}

// int min=x->n;

p = x;

while (p->sibling != NULL) {

if ((p->sibling)->n < min) {

min = (p->sibling)->n;

t = p;

x = p->sibling;

}

p = p->sibling;

}

if (t == NULL && x->sibling == NULL)

H1 = NULL; else if (t == NULL)

H1 = x->sibling; else if (t->sibling == NULL)

t = NULL; else

t->sibling = x->sibling;

if (x->child != NULL) {

REVERT\_LIST(x->child);

(x->child)->sibling = NULL;

}

H = bin\_HEAP\_UNION(H1, Hr);

return x;

}

int REVERT\_LIST(struct node\* y) {

if (y->sibling != NULL) {

REVERT\_LIST(y->sibling);

(y->sibling)->sibling = y;

} else {

Hr = y;

}

}

struct node\* FIND\_NODE(struct node\* H, int k) {

struct node\* x = H;

struct node\* p = NULL;

if (x->n == k) {

p = x;

return p;

}

if (x->child != NULL && p == NULL) {

p = FIND\_NODE(x->child, k);

}

if (x->sibling != NULL && p == NULL) {

p = FIND\_NODE(x->sibling, k);

}

return p;

}

int bin\_HEAP\_DECREASE\_KEY(struct node\* H, int i, int k) {

int temp;

struct node\* p;

struct node\* y;

struct node\* z;

p = FIND\_NODE(H, i);

if (p == NULL) {

printf("\nINVALID CHOICE OF KEY TO BE REDUCED");

return 0;

}

if (k > p->n) {

printf("\nSORY!THE NEW KEY IS GREATER THAN CURRENT ONE");

return 0;

}

p->n = k;

y = p;

z = p->parent;

while (z != NULL && y->n < z->n) {

temp = y->n;

y->n = z->n;

z->n = temp;

y = z;

z = z->parent;

}

printf("\nKEY REDUCED SUCCESSFULLY!");

}

int bin\_HEAP\_DELETE(struct node\* H, int k) {

struct node\* np;

if (H == NULL) {

printf("\nHEAP EMPTY");

return 0;

}

bin\_HEAP\_DECREASE\_KEY(H, k, -1000);

np = bin\_HEAP\_EXTRACT\_MIN(H);

if (np != NULL)

printf("\nNODE DELETED SUCCESSFULLY");

}

int main() {

int i, n, m, l;

struct node\* p;

struct node\* np;

char ch;

printf("\nENTER THE NUMBER OF ELEMENTS:");

scanf("%d", &n);

printf("\nENTER THE ELEMENTS:\n");

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

scanf("%d", &m);

np = CREATE\_NODE(m);

H = bin\_HEAP\_INSERT(H, np);

}

DISPLAY(H);

do {

printf("\nMENU:-\n");

printf(

"\n1)INSERT AN ELEMENT\n2)EXTRACT THE MINIMUM KEY NODE\n3)DECREASE A NODE KEY\n 4)DELETE A NODE\n5)QUIT\n");

scanf("%d", &l);

switch (l) {

case 1:

do {

printf("\nENTER THE ELEMENT TO BE INSERTED:");

scanf("%d", &m);

p = CREATE\_NODE(m);

H = bin\_HEAP\_INSERT(H, p);

printf("\nNOW THE HEAP IS:\n");

DISPLAY(H);

printf("\nINSERT MORE(y/Y)= \n");

fflush(stdin);

scanf("%c", &ch);

}

while (ch == 'Y' || ch == 'y');

break;

case 2:

do {

printf("\nEXTRACTING THE MINIMUM KEY NODE");

p = bin\_HEAP\_EXTRACT\_MIN(H);

if (p != NULL)

printf("\nTHE EXTRACTED NODE IS %d", p->n);

printf("\nNOW THE HEAP IS:\n");

DISPLAY(H);

printf("\nEXTRACT MORE(y/Y)\n");

fflush(stdin);

scanf("%c", &ch);

}

while (ch == 'Y' || ch == 'y');

break;

case 3:

do {

printf("\nENTER THE KEY OF THE NODE TO BE DECREASED:");

scanf("%d", &m);

printf("\nENTER THE NEW KEY : ");

scanf("%d", &l);

bin\_HEAP\_DECREASE\_KEY(H, m, l);

printf("\nNOW THE HEAP IS:\n");

DISPLAY(H);

printf("\nDECREASE MORE(y/Y)\n");

fflush(stdin);

scanf("%c", &ch);

}

while (ch == 'Y' || ch == 'y');

break;

case 4:

do {

printf("\nENTER THE KEY TO BE DELETED: ");

scanf("%d", &m);

bin\_HEAP\_DELETE(H, m);

printf("\nDELETE MORE(y/Y)\n");

fflush(stdin);

scanf("%c", &ch);

}

while (ch == 'y' || ch == 'Y');

break;

case 5:

printf("\nTHANK U SIR\n");

break;

default:

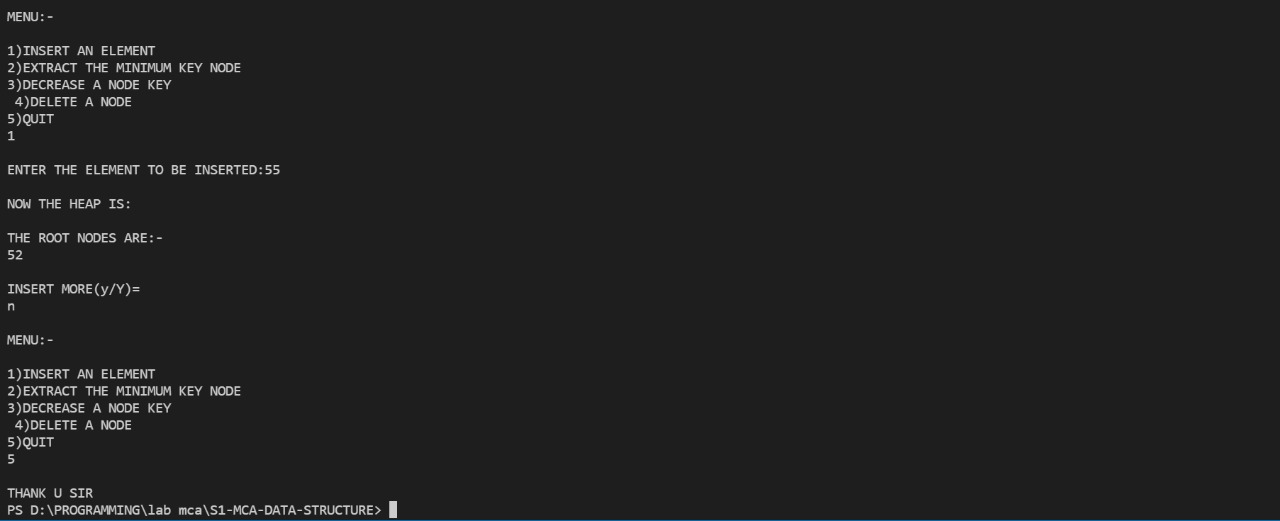
printf("\nINVALID ENTRY...TRY AGAIN....\n");}}

while (l != 5);

}

**OUTPUT:**

****

****

**RESULT:** The program was executed successfully and output obtained

**PROGRAM NO : 12**

**AIM :** Write a C program to implement Depth first Search.

**CODE :**

#include<stdio.h>

void dfs(int);

int g[10][10],visited[10], n;

void main()

{

int i, j;

printf ("enter the number of vertices:");

scanf ("%d", &n);

printf ("\n enter the adjacnecy matrix:");

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

{for(j = 0; j < n; ++j)

{printf("\n edge exist between vertices %d-%d :", i, j);

scanf("%d", &g[i][j]);}}

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

{visited[i] = 0; }

dfs(0);}

void dfs(int i)

{

int j;

printf ("\n %d", i);

visited[i] = 1;

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

{

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

{ dfs(j); } } }

**OUTPUT :**

Text

Description automatically generated

**RESULT :** The program was executed successfully and output obtained

**PROGRAM NO : 13**

**AIM :** Write a C program to implement Breadth first Search.

**CODE :**

#include<stdio.h>

int a[20][20],q[20],visited[20],n,i,j,f=0,r=-1;

void bfs(int v);

void main() {

int v;

printf("\n Enter the number of vertices:");

scanf("%d",&n);

printf("enter the adjecency matrix");

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

for (j=0;j<n;j++) {

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

printf("\n Enter the starting vertex:");

scanf("%d",&v);

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

q[i]=0;

visited[i]=0; }

bfs(v);

printf("\n The node which are reachable are:\n");

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

if(visited[i]) {

printf("%d\t",i);}}}

void bfs(int v) {

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

if(a[v][i] && !visited[i])

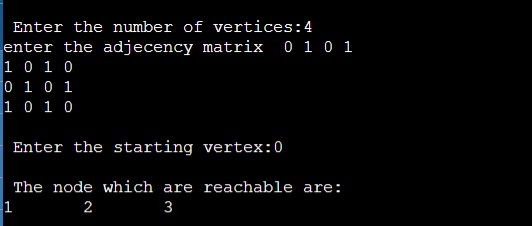
q[++r]=i;}

if(f<=r) {

visited[q[f]]=1;

bfs(q[f++]);}}

**OUTPUT :**

****

**RESULT:** The program was executed successfully and output obtained

**PROGRAM NO : 14**

**AIM :** Write a C program to implement Kruskal’s algorithm .

**CODE :**

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

int i,j,a,b,u,v,n,ne=1;

int min,cost=0,graph[9][9],parent[9];

int find(int);

int uni(int,int);

void main() {

printf("\nEnter the no. of vertices:");

scanf("%d",&n);

printf("\nEnter the cost adjacency matrix:\n");

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

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

printf("Enter the edge weight of %d to %d ",i,j);

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

if(graph[i][j]==0)

graph[i][j]=999;

}

}

printf("The edges of Minimum cost Spanning Tree are\n");

while(ne < n) {

min=999;

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

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

if(graph[i][j] < min) {

min=graph[i][j];

a=u=i;

b=v=j;

}

}

}

u=find(u);

v=find(v);

if(uni(u,v)) {

printf("edge (%d,%d) =%d\n",a,b,min);

cost +=min;

ne++;

}

graph[a][b]=graph[b][a]=999;

}

printf("\nMinimum cost = %d\n",cost);

}

int find(int i) {

while(parent[i]) {

i=parent[i];

}

return i;

}

int uni(int i,int j) {

if(i!=j) {

parent[j]=i;

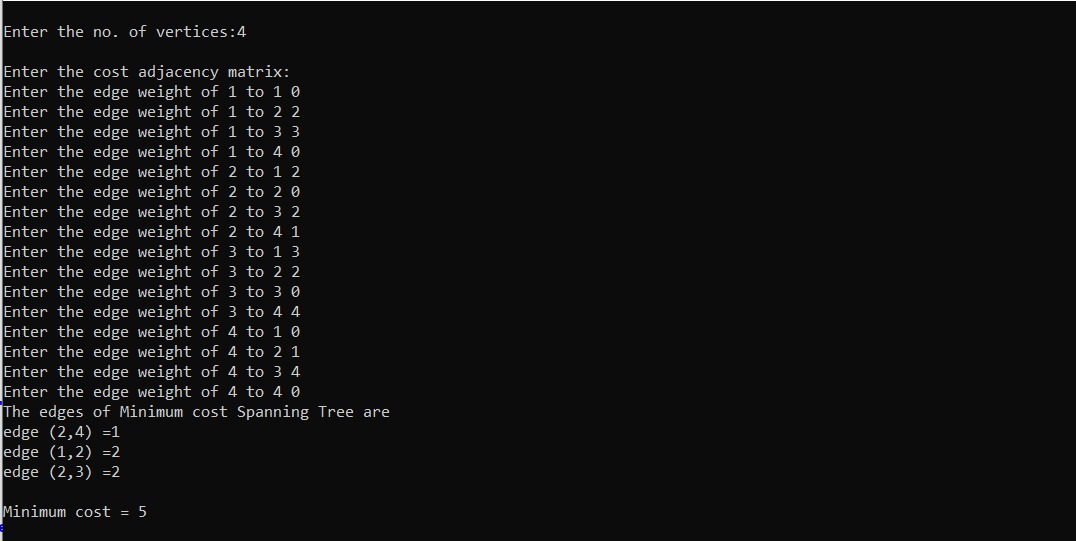
return 1;

}

return 0;

}

**OUTPUT :**

****

**RESULT :** The program was executed successfully and output obtained

**PROGRAM NO : 15**

**AIM :** Write a C program to implement Prim’s Algorithm

**CODE :**

#include<stdio.h>

#include<stdbool.h>

#define infinity 1000

//#define v 5

int graph[20][20];

int v;

/\*int graph[v][v] = {

{0, 9, 75, 0, 0},

{9, 0, 95, 19, 42},

{75, 95, 0, 51, 66},

{0, 19, 51, 0, 31},

{0, 42, 66, 31, 0}};

\*/

/\*void display(){

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

for(int j=0;j<v;j++){

printf("%d",graph[i][j]);

}

}

}\*/

void mst(bool span[]) {

int edge\_count=0,total=0,x,y;

span[0]=1;

printf("\nEdge : Weight\n");

while(edge\_count<v-1) {

int cost=infinity;

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

if(span[i]) {

for (int j=0;j<v;j++) {

if(!span[j] && graph[i][j]) {

if(graph[i][j] < cost) {

cost=graph[i][j];

x=i;

y=j; }}}}

printf("%d - %d : %d\n", x, y, graph[x][y]);

total+=graph[x][y];

span[y]=1;

edge\_count++;

printf("\nTotal Cost=%d\n",total); }

void main() {

printf("\nEnter the number of vertices ");

scanf("%d",&v);

printf("\nEnter the Adjacency Matrix \n");

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

for (int j=0;j<v;j++) {

scanf("%d",&graph[i][j])} }

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

graph[i][i]=0;

}

bool span[v];

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

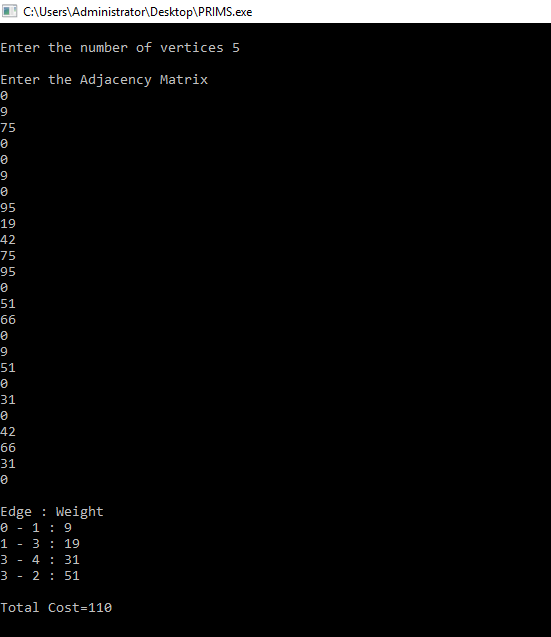
span[i]=0;

}

mst(span);

}

**OUTPUT :**



**RESULT :** The program was executed successfully and output obtained

**PROGRAM NO : 16**

**AIM :** Write a C program to implement Topological Sort

**CODE :**

#include <stdio.h>

void main() {

int n = 0;

printf("enter how many vertex are there - ");

scanf("%d", & n);

int a[n][n], tp[n], f[n], x = 0;

//considering the vertices to be numbers

printf("\nEnter 1 if an edge exits or otherwise\n");

for (int i = 1; i <= n; i++) {

f[i - 1] = 0;

for (int j = 1; j <= n; j++) {

printf("Does an edge exists from %d to %d - ", i, j);

scanf("%d", & a[i - 1][j - 1]);}}

printf("Topological Sort : - \n");

while (x < n) {

//finding indegree of all vertices

int in = 0, ind[n];

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

for (int j = 0; j < n; j++) {

if (a[j][i] == 1) {

in ++;}

}

ind[i] = in ;

in = 0; }

//Actual sorting

int t = 0;

for (t = 0; t < n; t++) {

if (ind[t] == 0 && f[t] == 0) {

f[t] = 1;

printf("%d ", t + 1);

break;

}

}

printf("\n");

//updating matrix with new values

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

if (a[t][i] == 1) {

a[t][i] = 0;

}

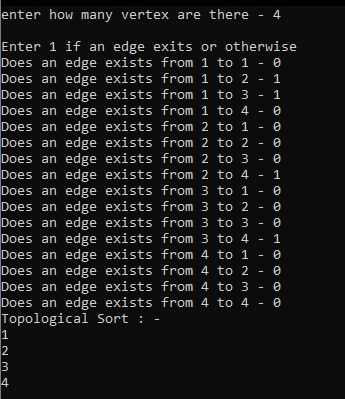
}

x++;

}

}

**OUTPUT :**

****

**RESULT :** The program was executed successfully and output obtained

**PROGRAM NO : 17**

**AIM :** Write a C program to implement Dijkstra’s Algorithm

**CODE :**

#include<stdio.h>

#include<conio.h>

#define INFINITY 9999

void dijkstra(int G[10][10], int n, int start);

int main() {

int G[10][10], i, j, n, u;

printf("Enter no. of vertices:");

scanf("%d", & n);

printf("\nEnter the adjacency matrix:\n");

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

for (j = 0; j < n; j++) {

printf("enter the distance between %d and %d :", i, j);

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

}

printf("\n");

}

printf("\nEnter the starting node:");

scanf("%d", & u);

dijkstra(G, n, u);

return 0;

}

void dijkstra(int G[10][10], int n, int start) {

int cost[10][10], distance[10], pred[10];

int visited[10], count, min, nextnode, i, j;

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

for (j = 0; j < n; j++) {

if (G[i][j] == 0) {

cost[i][j] = INFINITY;

} else {

cost[i][j] = G[i][j]; }}}

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

distance[i] = cost[start][i];

pred[i] = start;

visited[i] = 0;

}

distance[start] = 0;

visited[start] = 1;

count = 1;

while (count < n) {

min = INFINITY;

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

if (distance[i] < min && !visited[i]) {

min = distance[i];

nextnode = i; }}

visited[nextnode] = 1;

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

if (!visited[i]) {

if (min + cost[nextnode][i] < distance[i]) {

distance[i] = min + cost[nextnode][i];

pred[i] = nextnode;

}

count++;}}}

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

if (i != start) {

printf("\nDistance of node%d=%d", i, distance[i]);

printf("\nPath=%d", i);

j = i;

do {

j = pred[j];

printf("<-%d", j);

} while (j != start);

}

}

}

**OUTPUT :**

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

**RESULT :** The program was executed successfully and output obtained