**Module: - 01**

**Sorting Techniques**

* **Bubble and Insertion sort (menu driven):**

**Code:**

#include <iostream>

using namespace std;

class Sorting {

private:

static const int MAX\_SIZE = 100;

int arr[MAX\_SIZE];

int size;

public:

Sorting() {

size = 0;

}

void insertElement(int element) {

if (size < MAX\_SIZE) {

arr[size++] = element;

} else {

cout << "Array is full. Cannot insert more elements." << endl;

}

}

void displayArray() {

cout << "Array: ";

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

cout << arr[i] << " ";

}

cout << endl;

}

void bubbleSort() {

for (int i = 0; i < size - 1; ++i) {

bool swapped = false;

for (int j = 0; j < size - i - 1; ++j) {

if (arr[j] > arr[j + 1]) {

int temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

swapped = true;

}

}

if (!swapped) {

break; // If no swaps occur in an iteration, the array is already sorted

}

}

cout << "Sorted using Bubble Sort." << endl;

}

void insertionSort() {

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

int key = arr[i];

int j = i - 1;

while (j >= 0 && arr[j] > key) {

arr[j + 1] = arr[j];

j = j - 1;

}

arr[j + 1] = key;

}

cout << "Sorted using Insertion Sort." << endl;

}

void sort(int choice) {

switch (choice) {

case 1:

bubbleSort();

break;

case 2:

insertionSort();

break;

default:

cout << "Invalid choice! Please enter 1 for Bubble Sort or 2 for Insertion Sort." << endl;

}

}

};

int main() {

Sorting sortObj;

int choice, num, element;

cout << "Enter the number of elements in the array: ";

cin >> num;

cout << "Enter the elements of the array: ";

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

cin >> element;

sortObj.insertElement(element);

}

cout << "Choose sorting method:" << endl;

cout << "1. Bubble Sort" << endl;

cout << "2. Insertion Sort" << endl;

cin >> choice;

sortObj.sort(choice);

sortObj.displayArray(); // Displaying the sorted array

return 0;

}

**Output:**

Bubble Sort: -

A black screen with white text

Description automatically generated

Insertion Sort: -

A black screen with white text

Description automatically generated

* **Selection and Shell sort (menu driven):**

**Code:**

#include <iostream>

using namespace std;

class Sorting {

private:

static const int MAX\_SIZE = 100;

int arr[MAX\_SIZE];

int size;

public:

Sorting() {

size = 0;

}

void insertElement(int element) {

if (size < MAX\_SIZE) {

arr[size++] = element;

} else {

cout << "Array is full. Cannot insert more elements." << endl;

}

}

void displayArray() {

cout << "Array: ";

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

cout << arr[i] << " ";

}

cout << endl;

}

void selectionSort() {

for (int i = 0; i < size - 1; ++i) {

int minIndex = i;

for (int j = i + 1; j < size; ++j) {

if (arr[j] < arr[minIndex]) {

minIndex = j;

}

}

if (minIndex != i) {

int temp = arr[i];

arr[i] = arr[minIndex];

arr[minIndex] = temp;

}

}

cout << "Sorted using Selection Sort." << endl;

}

void shellSort() {

for (int gap = size / 2; gap > 0; gap /= 2) {

for (int i = gap; i < size; ++i) {

int temp = arr[i];

int j;

for (j = i; j >= gap && arr[j - gap] > temp; j -= gap) {

arr[j] = arr[j - gap];

}

arr[j] = temp;

}

}

cout << "Sorted using Shell Sort." << endl;

}

void sort(int choice) {

switch (choice) {

case 1:

selectionSort();

break;

case 2:

shellSort();

break;

default:

cout << "Invalid choice! Please enter 1 for Selection Sort or 2 for Shell Sort." << endl;

}

}

};

int main() {

Sorting sortObj;

int choice, num, element;

cout << "Enter the number of elements in the array: ";

cin >> num;

cout << "Enter the elements of the array: ";

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

cin >> element;

sortObj.insertElement(element);

}

cout << "Choose sorting method:" << endl;

cout << "1. Selection Sort" << endl;

cout << "2. Shell Sort" << endl;

cin >> choice;

sortObj.sort(choice);

sortObj.displayArray(); // Displaying the sorted array

return 0;

}

**Output:**

Selection Sort: -

A black screen with white text

Description automatically generated

Shell Sort: -

A black screen with white text

Description automatically generated

* **Implement Radix Sort:**

**Code:**

#include <iostream>

using namespace std;

class RadixSort {

private:

int arr[100];

int size;

int getMax() {

int max = arr[0];

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

if (arr[i] > max) {

max = arr[i];

}

}

return max;

}

void countSort(int exp) {

int output[100];

int count[10] = {0};

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

count[(arr[i] / exp) % 10]++;

}

for (int i = 1; i < 10; ++i) {

count[i] += count[i - 1];

}

for (int i = size - 1; i >= 0; --i) {

output[count[(arr[i] / exp) % 10] - 1] = arr[i];

count[(arr[i] / exp) % 10]--;

}

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

arr[i] = output[i];

}

}

public:

RadixSort() {

size = 0;

}

void insertElement(int element) {

if (size < 100) {

arr[size++] = element;

} else {

cout << "Array is full. Cannot insert more elements." << endl;

}

}

void radixSort() {

int max = getMax();

for (int exp = 1; max / exp > 0; exp \*= 10) {

countSort(exp);

}

}

void displayArray() {

cout << "Sorted Array: ";

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

cout << arr[i] << " ";

}

cout << endl;

}

};

int main() {

RadixSort sortObj;

int num, element;

cout << "Enter the number of elements in the array: ";

cin >> num;

cout << "Enter the elements of the array: ";

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

cin >> element;

sortObj.insertElement(element);

}

sortObj.radixSort();

sortObj.displayArray();

return 0;

}

**Output:**

**A black screen with white text

Description automatically generated**

**Module: - 02**

**Searching Techniques**

* **Linear Search, Binary Search (menu driven):**

**Code:**

#include <iostream>

using namespace std;

class Search {

private:

static const int MAX\_SIZE = 100;

int arr[MAX\_SIZE];

int size;

public:

Search() {

size = 0;

}

void insertElement(int element) {

if (size < MAX\_SIZE) {

arr[size++] = element;

} else {

cout << "Array is full. Cannot insert more elements." << endl;

}

}

void linearSearch(int key) {

bool found = false;

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

if (arr[i] == key) {

cout << "Linear Search: Element found at index " << i << endl;

found = true;

break;

}

}

if (!found) {

cout << "Linear Search: Element not found in the array." << endl;

}

}

void bubbleSort() {

for (int i = 0; i < size - 1; ++i) {

for (int j = 0; j < size - i - 1; ++j) {

if (arr[j] > arr[j + 1]) {

int temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

}

void binarySearch(int key) {

bubbleSort();

int left = 0;

int right = size - 1;

bool found = false;

while (left <= right) {

int mid = left + (right - left) / 2;

if (arr[mid] == key) {

cout << "Binary Search: Element found at index " << mid << endl;

found = true;

break;

} else if (arr[mid] < key) {

left = mid + 1;

} else {

right = mid - 1;

}

}

if (!found) {

cout << "Binary Search: Element not found in the array." << endl;

}

}

void search(int key, int choice) {

switch (choice) {

case 1:

linearSearch(key);

break;

case 2:

binarySearch(key);

break;

default:

cout << "Invalid choice! Please enter 1 for Linear Search or 2 for Binary Search." << endl;

}

}

};

int main() {

Search searchObj;

int choice, key, num;

cout << "Enter the number of elements in the array: ";

cin >> num;

cout << "Enter the elements of the array: ";

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

int element;

cin >> element;

searchObj.insertElement(element);

}

cout << "Enter the element to search: ";

cin >> key;

cout << "Choose search method:" << endl;

cout << "1. Linear Search" << endl;

cout << "2. Binary Search" << endl;

cin >> choice;

searchObj.search(key, choice);

return 0;

}

**Output:**

Linear Search: -

A black screen with white text

Description automatically generated

Binary Search: -

A black screen with white text

Description automatically generated

**Module: - 03**

**Hashing**

* **To implement hashing technique: Modulo division, digit extraction, fold shift, fold boundary methods for collision resolution use linear probe:**

**Code:**

#include <iostream>

using namespace std;

const int TABLE\_SIZE = 10;

class HashTable {

private:

int table[TABLE\_SIZE];

int moduloDivisionHash(int key) {

return key % TABLE\_SIZE;

}

int digitExtractionHash(int key) {

return key % 10;

}

int foldShiftHash(int key) {

int sum = 0;

while (key > 0) {

sum += key % 100;

key /= 100;

}

return sum % TABLE\_SIZE;

}

int foldBoundaryHash(int key) {

int sum = 0;

while (key > 0) {

int digit = key % 10;

sum += digit;

key /= 10;

}

return sum % TABLE\_SIZE;

}

int linearProbe(int index, int key) {

int i = 1;

while (table[(index + i) % TABLE\_SIZE] != -1) {

i++;

}

return (index + i) % TABLE\_SIZE;

}

public:

HashTable() {

for (int i = 0; i < TABLE\_SIZE; i++) {

table[i] = -1;

}

}

void insert(int key) {

int index;

index = foldBoundaryHash(key);

if (table[index] == -1) {

table[index] = key;

} else {

index = linearProbe(index, key);

table[index] = key;

}

}

void display() {

cout << "Hash Table:" << endl;

for (int i = 0; i < TABLE\_SIZE; i++) {

cout << i << ": ";

if (table[i] != -1) {

cout << table[i];

}

cout << endl;

}

}

};

int main() {

HashTable ht;

int keys[] = {25, 35, 45, 55, 12, 22, 17, 30, 32};

int numKeys = sizeof(keys) / sizeof(keys[0]);

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

ht.insert(keys[i]);

}

ht.display();

return 0;

}

**Output:**

**A screen shot of a computer

Description automatically generated**

**Module: - 04**

**Linked Lists**

* **A menu driven program that implements singly linked list for the following operations: insert, display, count, delete, search:**

**Code:**

#include<iostream>

using namespace std;

class node {

private:

int data;

node\* address;

public:

void add(int);

void display(void);

int count(void);

void search(int);

void sort(void);

void insert(int,int);

void remove(int);

};

node\* p;

void node::add(int num) {

node\* q=p;

if(p==NULL) {

p=new node; p->data=num; p->address=NULL; }

else {

while(q

->address!=NULL)

{

q=q

->address;

}q

->address = new node;

q

->address

-> data = num;

q

->address

-> address = NULL;

}}

void node::display(void) {

node\* q=p;

if(p==NULL) {

cout<<"NO LINKLIST"; }

else {

while(q!=NULL) {

cout<<q

->data<<" ";

q=q

->address;

}}}

int node::count(void) {

node\* q=p;

int num;

int i=0;

if(p==NULL) {

return 0; }

else {

while(q!=NULL) {

q=q

->address;

i++; }

return i; }}

void node::search(int num) {

node\* q=p;

int flag=0;

int pos=1;

if(p==NULL) {

cout<<"NO LINKLIST AVAILABLE"; }

else {

for(q=p;q!=NULL;q=q

->address)

{

if(q

->data==num)

{

flag=1;

break; }

pos++; }

if(flag==1) {

cout<<"Number is available at position: " << pos; }

else {

cout<<"Number is not Available"; }}}

void node::sort(void) {

node\* q=p;

node\* i;

node\* j;

int temp;

if(p==NULL) {

cout<<"NO LINKLIST AVAILABLE"; }

else {

for(i=p;i!=NULL;i=i

->address)

{

for(j=i

->address;j!=NULL;j=j

->address)

{

if(i

->data>j

->data)

{

temp=i

->data;

i

->data=j

->data;

j

->data=temp; }}}}}

void node::insert(int num, int pos) {

node\* q=p;

node\* temp;

int i;

if(pos==1) {

p=new node; p->data = num; p->address=q;

return; }

if(pos==1+count()) {

add(num);

return; }

for(i=1;i<=(pos

-2);i++)

{

q=q

->address;

}

temp=q

->address;

q

->address=new node;

q

->address

->data = num;

q

->address

->address=temp;

}

void node::remove(int pos) {

node\* q=p;

node\* temp;

if(pos==1) {

p=p

->address;

delete(q); }

for(int i=1;i<=(pos

-2);i++)

{

q=q

->address;

}

temp=q

->address;

q

->address=q

->address

->address;

delete(temp); }

int main(void) {

int num,option,pos;

char ch='y';

p=NULL;

node n;

while(ch=='y') {

cout<<"\n 1.ADD";

cout<<"\n 2.DISPLAY";

cout<<"\n 3.COUNT";

cout<<"\n 4.SEARCH";

cout<<"\n 5.SORT";

cout<<"\n 6.INSERT";

cout<<"\n 7.REMOVE";

cout<<"\n Enter an Option (1-7): ";

cin>>option;

switch(option) {

case 1: {

cout<<"\n Enter a Number: ";

cin>>num;

n.add(num);

cout<<"\n Do you want to Continue (press y): ";

break; }

case 2: {

cout<<"\n Display List: ";

n.display();

cout<<"\n Do you want to Continue (press y): ";

break; }

case 3: {

num=n.count();

cout<<"\n Number of Nodes: " << num;

cout<<"\n Do you want to Continue (press y): ";

break; }

case 4: {

cout<<"\n Enter a Number to Search: ";

cin>>num;

n.search(num);

cout<<"\n Do you want to Continue (press y): ";

break;

}

case 5:

{

cout<<"\n Before Sorted List: ";

n.display();

n.sort();

cout<<"\n After Sorted List: ";

n.display();

cout<<"\n Do you want to Continue (press y): ";

break;

}

case 6:

{

cout<<"\n Enter a number you want to add: ";

cin>> num;

cout<<"\n Enter position where you want to add: ";

cin>> pos;

n.insert(num,pos);

cout<<"\n Do you want to Continue (press y): ";

break;

}

case 7:

{

cout<<"Enter the position you want to delete: ";

cin>>pos;

n.remove(pos);

cout<<"\n Do you want to Continue (press y): ";

break;

}

}

}

}

**Output:**

Adding: -

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

Displaying: -

A screen shot of a computer

Description automatically generated

Counting: -

A screen shot of a computer

Description automatically generated

Searching: -

A screen shot of a computer

Description automatically generated

Sorting: -

A screen shot of a computer

Description automatically generated

Inserting: -

A screenshot of a computer program

Description automatically generated

Removing: -

A screenshot of a computer program

Description automatically generated

* **A menu driven program that implements doubly linked list for the following operations: insert, display, count, delete, search:**

**Code:**

#include<iostream>

using namespace std;

class dnode {

private:

int data;

dnode\* address;

dnode\* prev;

public:

void add(int);

void display(void);

int count(void);

void search(int);

void sort(void);

void insert(int,int);

void remove(int);

void reverse(void);

};

dnode\* p;

void dnode::add(int num) {

dnode\* q=p;

if(p==NULL) {

p=new dnode; p->prev=NULL; p->data=num; p->address=NULL; }

else {

while(q

->address!=NULL)

{

q=q

->address;

}q

->address = new dnode;

q

->address

-> prev = q;

q

->address

-> data = num;

q

->address

-> address = NULL;

}}

void dnode::display(void) {

dnode\* q=p;

if(p==NULL) {

cout<<"NO LINKLIST"; }

else {

while(q!=NULL) {

cout<<q

->data<<" ";

q=q

->address;

}}}

int dnode::count(void) {

dnode\* q=p;

int num;

int i=0;

if(p==NULL) {

return 0; }

else {

while(q!=NULL) {

q=q

->address;

i++; }

return i; }}

void dnode::search(int num) {

dnode\* q=p;

int flag=0;

int pos=1;

if(p==NULL) {

cout<<"NO LINKLIST AVAILABLE";

}

else {

for(q=p;q!=NULL;q=q

->address)

{

if(q

->data==num)

{

flag=1;

break; }

pos++; }

if(flag==1) {

cout<<"Number is available at position: " << pos; }

else {

cout<<"Number is not Available"; }}}

void dnode::sort(void) {

dnode\* q=p;

dnode\* i;

dnode\* j;

int temp;

if(p==NULL) {

cout<<"NO LINKLIST AVAILABLE"; }

else {

for(i=p;i!=NULL;i=i

->address)

{

for(j=i

->address;j!=NULL;j=j

->address)

{

if(i

->data>j

->data)

{

temp=i

->data;

i

->data=j

->data;

j

->data=temp; }}

}}}

void dnode::insert(int num, int pos) {

dnode\* q=p;

dnode\* temp;

int i;

if(pos==1) {

p=new dnode; p->data = num; p->address=q; p->prev = NULL;

if(q!=NULL) {q->prev = p; }

return; }

if(pos==1+count()) {

add(num);

return; }

for(i=1;i<=(pos

-2);i++)

{

q=q

->address;

}

temp=q

->address;

q

->address=new dnode;

q

->address

->data = num;

q

->address

->prev = q;

q

->address

->address=temp;

if(temp != NULL) {

temp

->prev = q

->address;

}}

void dnode::remove(int pos) {

dnode\* q=p;

dnode\* temp;

if(pos==1) {

p=p

->address;

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

delete(q);

return; }

for(int i=1;i<=(pos

-2);i++)

{

q=q

->address;

}

temp=q

->address;

q

->address=temp

->address;

if(temp

->address != NULL)

{

temp

->address

->prev = q;

}

delete(temp); }

void dnode::reverse(void) {

dnode \*q=p;

if(p==NULL) {

cout << "NO LINKLIST AVAILABLE"; }

else {

while(q

->address!=NULL)

{

q=q

->address;

}

while(q!=NULL) {

cout << q

->data << " ";

q=q

->prev;

}}}

int main(void) {

int num,option,pos;

char ch='y';

p=NULL;

dnode n;

while(ch=='y') {

cout<<"\n 1.ADD";

cout<<"\n 2.DISPLAY";

cout<<"\n 3.COUNT";

cout<<"\n 4.SEARCH";

cout<<"\n 5.SORT";

cout<<"\n 6.INSERT";

cout<<"\n 7.REMOVE";

cout<<"\n 8.REVERSE";

cout<<"\n Enter an Option (1-8): ";

cin>>option;

switch(option) {

case 1: {

cout<<"\n Enter a Number: ";

cin>>num;

n.add(num);

cout<<"\n Do you want to Continue (press y): ";

break; }

case 2: {

cout<<"\n Display List: ";

n.display();

cout<<"\n Do you want to Continue (press y): ";

break; }

case 3: {

num=n.count();

cout<<"\n Number of Nodes: " << num;

cout<<"\n Do you want to Continue (press y): ";

break; }

case 4: {

cout<<"\n Enter a Number to Search: ";

cin>>num;

n.search(num);

cout<<"\n Do you want to Continue (press y): ";

break; }

case 5: {

cout<<"\n Before Sorted List: ";

n.display();

n.sort();

cout<<"\n After Sorted List: ";

n.display();

cout<<"\n Do you want to Continue (press y): ";

break; }

case 6: {

cout<<"\n Enter a number you want to add: ";

cin>> num;

cout<<"\n Enter position where you want to add: ";

cin>> pos;

n.insert(num,pos);

cout<<"\n Do you want to Continue (press y): ";

break; }

case 7: {

cout<<"Enter the position you want to delete: ";

cin>>pos;

n.remove(pos);

cout<<"\n Do you want to Continue (press y): ";

break; }

case 8: {

cout << "\n Before List Reverse: ";

n.display();

cout << "\n After List Reverse: ";

n.reverse();

cout << "\n Do you want to Continue (press y): ";

break; }}

}}

**Output:**

Adding: -

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

Displaying: -

A screen shot of a computer

Description automatically generated

Counting: -

A screen shot of a computer

Description automatically generated

Searching: -

A screen shot of a computer

Description automatically generated

Sorting: -

A screen shot of a computer

Description automatically generated

Inserting: -

A screen shot of a computer program

Description automatically generated

A screen shot of a computer

Description automatically generated

Removing: -

A screenshot of a computer program

Description automatically generated

Reversing: -

A screen shot of a computer

Description automatically generated

* **A menu driven program that implements singly circular linked list for the following operations: insert, display, count, delete, search:**

**Code:**

#include<iostream>

using namespace std;

class cnode {

private:

int data;

cnode\* next;

public:

void add(int);

void display(void);

int count(void);

void search(int);

void sort(void);

void insert(int, int);

void remove(int);

};

cnode\* p;

void cnode::add(int num)

{

cnode\* q = p;

if (p == NULL) {

p = new cnode; p->data = num; p->next = p;

}

else {

while (q

->next != p)

{

q = q

->next;

}q

->next = new cnode;

q

->next

->data = num;

q

->next

->next = p;

}

}

void cnode::display(void)

{

cnode\* q = p;

if (p == NULL)

{

cout << "No LinkList Available";

}

else {

do {

cout << q

->data << " ";

q = q

->next;

} while (q != p);

}

}

int cnode::count(void)

{

cnode\* q = p;

int i = 0;

if (p == NULL) {

return i;

}

else {

do {

i++;

q = q

->next;

} while (q != p);

return i;

}

}

void cnode::search(int num)

{

cnode\* q = p;

int flag = 0;

int pos = 1;

if (p == NULL) {

cout << "No LinkList Available";

}

else {

do {

if (q

->data == num)

{

flag = 1;

break; }

q = q

->next;

pos++;

} while (q != p);

}

if (flag == 1) {

cout << "Data is available at position " << pos;

}

else {

cout << "Number is not available";

}

}

void cnode::sort(void) { cnode \*i; cnode \*j;

int temp;

if(p==NULL) {

cout << "NO LINKLIST";

}

else {

i=p;

do{

j=i

->next;

while(j!=p) {

if(i

->data>j

->data)

{

temp=i

->data;

i

->data=j

->data;

j

->data=temp; }

j=j

->next;

}

i=i

->next;

} while (i

->next!=p);

}

}

void cnode::insert(int pos, int num)

{

cnode\* q = p;

cnode\* temp;

if(pos==1) {

node \*i=p;

do{

i=i

->next;

} while (i

->next!=p);

p=new node; p->data=num; p->next=q; i->next=p; }

else {

for(int i=1;i<=pos

-2;i++)

{

q=q

->next;

}

temp = q

->next;

q

->next=new cnode;

q

->next

->data=num;

q

->next

->next=temp;

}

}

void cnode::remove(int pos) {cnode \*q=p; cnode \*temp; cnode \*i=p;

if(p==NULL) {

cout << "NO LINKEDLIST AVAILABLE"; }

else if(pos==1) {

do{

22

i=i

->next;

}while(i

->next!=p);

if(i==p) {

temp = p;

delete(temp);

p=NULL;

cout << "\n DATA REMOVED";

}

else {

temp = p;

p=p

->next;

i

->next=p;

delete(temp);

cout << "\n DATA REMOVED";

}}

else {

for(int i=1;i<=pos

-2;i++)

{

q=q

->next;

}

temp=q

->next;

q

->next=q

->next

->next;

delete(temp);

cout << "\n DATA REMOVED";

}}

int main(void) {

int num, option, pos;

char ch = 'y';

p = NULL;

cnode n;

int pos;

while (ch == 'y') {

cout << "\n 1.ADD";

cout << "\n 2.DISPLAY";

cout << "\n 3.COUNT";

cout << "\n 4.SORT";

cout << "\n 5.SEARCH";

cout << "\n 6.INSERT";

cout << "\n 7.REMOVE";

cout << "\n Enter an Option: ";

cin >> option;

switch (option) {

case 1: {

cout << "\n Enter a Number: ";

cin >> num;

n.add(num);

cout << "\n Do you want to continue? (press y): ";

break;

}

case 2: {

cout << "\n Display List: \n";

n.display();

cout << "\n Do you want to continue? (press y): ";

break;

}

case 3: {

num = n.count();

cout << "\n Number of Nodes: " << num;

cout << "\n Do you want to continue? (press y): ";

break;

}

case 4: {

cout << "\n Before Sorted List: ";

n.display();

n.sort();

cout << "\n After Sorted List: ";

n.display();

cout << "\n Do you want to continue? (press y): ";

break;

}

case 5: {

cout << "\n Enter number to search: ";

cin >> num;

n.search(num);

cout << "\n Do you want to continue? (press y): ";

break;

}

case 6: {

cout << "\n Enter number you want to add: ";

cin >> pos;

cout << "\n Enter position where you want to add: ";

cin >> num;

n.insert(pos, num);

cout << "\n Do you want to continue? (press y): ";

break;

}

case 7: {

cout << "\n Enter the position you want to Delete: ";

cin >> pos;

n.remove(pos);

cout << "\n Do you want to continue? (press y): ";

break;

}

}

}

}

**Output:**

Adding: -

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

Displaying: -

A screen shot of a computer

Description automatically generated

Counting: -

A screen shot of a computer

Description automatically generated

Searching: -

A screen shot of a computer

Description automatically generated

Sorting: -

A screen shot of a computer

Description automatically generated

Inserting: -

A screenshot of a computer program

Description automatically generated

Removing: -

A screenshot of a computer program

Description automatically generated

**Module: - 05**

**Stacks**

* **Program that implements stack using array:**

**Code:**

#include <iostream>

using namespace std;

#define MAX\_SIZE 100

class Stack {

private:

int top;

int arr[MAX\_SIZE];

public:

Stack() {

top = -1; // Initializing top to -1 when the stack is empty

}

bool isEmpty() {

return (top == -1);

}

bool isFull() {

return (top == MAX\_SIZE - 1);

}

void push(int value) {

if (isFull()) {

cout << "Stack Overflow: Cannot push element, stack is full." << endl;

return;

}

arr[++top] = value;

cout << value << " pushed into the stack." << endl;

}

void pop() {

if (isEmpty()) {

cout << "Stack Underflow: Cannot pop element, stack is empty." << endl;

return;

}

cout << arr[top] << " popped from the stack." << endl;

top--;

}

int peek() {

if (isEmpty()) {

cout << "Stack is empty." << endl;

return -1; // Return -1 for an empty stack

}

return arr[top];

}

};

int main() {

Stack stack;

int choice, value;

do {

cout << "\nStack Operations:" << endl;

cout << "1. Push" << endl;

cout << "2. Pop" << endl;

cout << "3. Peek" << endl;

cout << "4. Exit" << endl;

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1:

cout << "Enter value to push: ";

cin >> value;

stack.push(value);

break;

case 2:

stack.pop();

break;

case 3:

cout << "Top element: " << stack.peek() << endl;

break;

case 4:

cout << "Exiting the program." << endl;

break;

default:

cout << "Invalid choice. Please enter a valid option." << endl;

}

} while (choice != 4);

return 0;

}

**Output:**

Push Operation: -

A screen shot of a black screen

Description automatically generated

Pop Operation: -

A screen shot of a black screen

Description automatically generated

Peek Operation: -

A black screen with white text

Description automatically generated

* **Program that implements stack using linked list:**

**Code:**

#include <iostream>

using namespace std;

class Node {

public:

int data;

Node\* next;

Node(int value) {

data = value;

next = NULL;

}

};

class Stack {

public:

Node\* top;

public:

Stack() {

top = NULL; // Initializing top to NULL when the stack is empty

}

bool isEmpty() {

return (top == NULL);

}

void push(int value) {

Node\* newNode = new Node(value);

if (isEmpty()) {

top = newNode;

} else {

newNode->next = top;

top = newNode;

}

cout << value << " pushed into the stack." << endl;

}

void pop() {

if (isEmpty()) {

cout << "Stack Underflow: Cannot pop element, stack is empty." << endl;

return;

}

Node\* temp = top;

top = top->next;

cout << temp->data << " popped from the stack." << endl;

delete temp;

}

int peek() {

if (isEmpty()) {

cout << "Stack is empty." << endl;

return -1; // Return -1 for an empty stack

}

return top->data;

}

};

int main() {

Stack stack;

int choice, value;

do {

cout << "\nStack Operations:" << endl;

cout << "1. Push" << endl;

cout << "2. Pop" << endl;

cout << "3. Peek" << endl;

cout << "4. Exit" << endl;

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1:

cout << "Enter value to push: ";

cin >> value;

stack.push(value);

break;

case 2:

stack.pop();

break;

case 3:

cout << "Top element: " << stack.peek() << endl;

break;

case 4:

cout << "Exiting the program." << endl;

break;

default:

cout << "Invalid choice. Please enter a valid option." << endl;

}

} while (choice != 4);

return 0;

}

**Output:**

Push Operation: -

A screen shot of a computer program

Description automatically generated

Pop Operation: -

A black screen with white text

Description automatically generated

Peek Operation: -

A black screen with white text

Description automatically generated

* **Implement to check Balancing of Parenthesis:**

**Code:**

#include <iostream>

using namespace std;

class ParenthesisChecker {

public:

bool isBalanced(const char\* expr) {

int count = 0;

while (\*expr != '\0') {

if (\*expr == '(') {

count++;

} else if (\*expr == ')') {

count--;

}

if (count < 0) {

return false;

}

expr++;

}

return (count == 0);

}

};

int main() {

ParenthesisChecker checker;

string expression;

cout << "Enter an expression with parentheses: ";

cin >> expression;

if (checker.isBalanced(expression.c\_str())) {

cout << "The expression has balanced parentheses." << endl;

} else {

cout << "The expression does not have balanced parentheses." << endl;

}

return 0;

}

**Output:**

****

**A black background with white text

Description automatically generated**

* **Program that Evaluation of postfix expression:**

**Code:**

#include <iostream>

using namespace std;

class PostfixEvaluator {

private:

int stack[1000];

int top;

public:

PostfixEvaluator() {

top = -1;

}

void push(int value) {

if (top >= 999) {

cout << "Stack Overflow: Cannot push element, stack is full." << endl;

return;

}

stack[++top] = value;

}

int pop() {

if (top < 0) {

cout << "Stack Underflow: Cannot pop element, stack is empty." << endl;

return -1;

}

return stack[top--];

}

int evaluatePostfix(const char\* expr) {

int operand1, operand2;

int i = 0;

while (expr[i] != '\0') {

if (expr[i] >= '0' && expr[i] <= '9') {

push(expr[i] - '0');

} else if (expr[i] == ' ') {

// Ignore spaces

} else {

operand2 = pop();

operand1 = pop();

switch (expr[i]) {

case '+':

push(operand1 + operand2);

break;

case '-':

push(operand1 - operand2);

break;

case '\*':

push(operand1 \* operand2);

break;

case '/':

push(operand1 / operand2);

break;

}

}

i++;

}

return pop();

}

};

int main() {

PostfixEvaluator evaluator;

char expression[1000];

cout << "Enter a postfix expression (with spaces between elements): ";

cin.getline(expression, 1000);

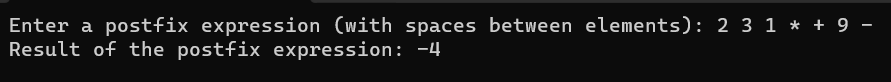
int result = evaluator.evaluatePostfix(expression);

cout << "Result of the postfix expression: " << result << endl;

return 0;

}

**Output:**

****

**Module: - 06**

**Queues:**

* **Program that implements ordinary Queue using array:**

**Code:**

#include<iostream>

using namespace std;

class queue {

private:

int data;

queue\* next;

public:

void add(int);

void display(void);

int remove(void);

};

queue\* front;

queue\* rear;

void queue::display(void) {

queue\* q=front;

if((front==NULL) && (rear==NULL)) {

cout<<"queue is empty"; }

else {

while(q!=NULL) {

cout<<q

->data<<" ";

q=q

->next;

}}}

void queue::add(int num) {

if((front==NULL) && (rear==NULL)) {

front = rear = new queue;

front

->data=num;

front

->next=NULL;

}

else {

rear

->next= new queue;

rear

->next

->data=num;

rear

->next

->next=NULL;

rear=rear

->next;

}}

int queue::remove(void) {

int num;

if((front==NULL) && (rear==NULL)) {

return

-1;

}

else {

queue\* temp = front;

num = front

->data;

front = front

->next;

if(front!=NULL) {

delete(temp); }

else {

delete(rear);

rear=NULL; }

return num; }}

int main() {

int num, option;

queue q;

char ch='y';

front = NULL;

rear = NULL;

while(ch == 'y') {

cout << "\n 1.ADD";

cout << "\n 2.REMOVE";

cout << "\n 3.DISPLAY";

cout << "\n Enter an Option(1-3): ";

cin >> option;

switch (option)

{

case 1:

{

cout << "Enter a Number to ADD: ";

cin >> num;

q.add(num);

cout << "\n Do you want to Continue (press y): ";

break;

}

case 2:

{

num = q.remove();

cout << num;

cout << "\n Do you want to Continue (press y): ";

break;

}

case 3:

{

q.display();

cout << "\n Do you want to Continue (press y): ";

break;

}

}

}

}

**Output:**

Adding: -

A screenshot of a computer program

Description automatically generated

Displaying: -

A black screen with white text

Description automatically generated

Removing: -

A screen shot of a computer program

Description automatically generated

* **Program that implements circular Queue using array:**

**Code:**

#include<iostream>

using namespace std;

class queue {

private:

int data;

queue\* next;

public:

void add(int);

void display(void);

int remove(void);

};

queue\* front;

queue\* rear;

void queue::display(void) {

queue\* q=front;

if((front==NULL) && (rear==NULL)) {

cout<<"queue is empty"; }

else {

do{

cout<<q->data<<" ";

q=q->next;

}while(q!=front); }}

void queue::add(int num) {

if((front==NULL) && (rear==NULL)) {

front = rear = new queue;

front->data=num;

front->next=front;

}

else {

rear->next= new queue;

rear->next->data=num;

rear->next->next=front;

rear=rear->next;

}}

int queue::remove(void) {

int num;

queue\* temp;

if((front==NULL) && (rear==NULL)) {

return

-1;

}

else {

if(front==rear) {

temp=front;

num=front->data;

delete(temp);

front=rear=NULL;

return num; }

else {

temp=front;

num=front->data;

front=front->next;

delete(temp);

return num; }}}

int main(void) {

int num, option;

queue q;

char ch='y';

front = NULL;

rear = NULL;

while(ch == 'y') {

cout << "\n 1.ADD";

cout << "\n 2.REMOVE";

cout << "\n 3.DISPLAY";

cout << "\n Enter an Option(1-3): ";

cin >> option;

switch (option)

{

case 1:

{

cout << "Enter a Number to ADD: ";

cin >> num;

q.add(num);

cout << "\n Do you want to Continue (press y): ";

break;

}

case 2:

{

num = q.remove();

cout << num;

cout << "\n Do you want to Continue (press y): ";

break;

}

case 3:

{

q.display();

cout << "\n Do you want to Continue (press y): ";

break;

}

}

}

}

**Output:**

Adding: -

A screenshot of a computer program

Description automatically generated

Displaying: -

A black screen with white text

Description automatically generated

A black screen with white text

Description automatically generated

Removing: -

A screenshot of a computer program

Description automatically generated

* **Program that implements Priority Queue using Linked List:**

**Code:**

#include<iostream>

using namespace std;

class queue {

private:

int data;

int pr;

queue \*next;

public:

void add(int, int);

void display(void);

int remove(void);

void priority(void);

};

queue \*front;

queue \*rear;

void queue::display(void) {

queue \*q=front;

if((front==NULL)&&(rear==NULL)) {

cout << "Queue is Empty"; }

else {

while(q!=NULL) {

cout << "\n Entered No. is: " << q

->data << " "

<< "Priority is : " << q

->pr << " ";

q=q

->next;

}}}

void queue::add(int num, int pri) {

if((front==NULL)&&(rear==NULL)) {

front=rear=new queue;

front

->data=num;

front

->pr=pri;

front

->next=NULL;

}

else {

rear

->next=new queue;

rear

->next

->data=num;

rear

->next

->pr=pri;

rear

->next

->next=NULL;

rear=rear

->next;

}}

int queue::remove(void) {

int num;

queue \*temp;

if((front==NULL)&&(rear==NULL)) {

return

-1;

}

else {

temp = front;

num = front

->data;

if(front==rear) {

delete(temp);

front = rear = NULL;

return num; }

else {

front = front

->next;

delete(temp);

return num; }}}

void queue::priority(void) {

int num, pri;

if((front==NULL)&&(rear==NULL)) {

cout << "Queue is empty"; }

else

{

if(front

->pr > rear

->pr)

{

num=remove();

cout << num; }

else {

cout << "\n Enter a Number: ";

cin >> num;

cout << "Enter Priority: ";

cin >> pri;

add(num,pri); }}}

int main(void) {

int num,pri,option;

queue q;

char ch = 'y';

front=NULL;

rear=NULL;

while(ch=='y') {

cout << "\n 1.ADD";

cout << "\n 2.DISPLAY";

cout << "\n 3.REMOVE";

cout << "\n 4.PRIORITY";

cout << "\n Enter an Option(1-4): ";

cin >> option;

switch (option) {

case 1: {

cout << "\n Enter a Number: ";

cin >> num;

cout << "\n Enter Priority: ";

cin >> pri;

q.add(num,pri);

cout << "\n Do you want to Continue (press y): ";

break;

}

case 2:

{

cout << "\n DISPLAYED LIST: ";

q.display();

cout << "\n Do you want to Continue (press y): ";

break;

}

case 3:

{

num = q.remove();

cout << num;

cout << "\n Do you want to Continue (press y): ";

break;

}

case 4:

{

cout << "\n PRIORITY: ";

q.priority();

cout << "\n Do you want to Continue (press y): ";

break;

}

default:

{

cout << "\n Invalid Operation";

cout << "\n Do you want to Continue (press y): ";

break;

}

}

}

}

**Output:**

Adding: -

A screenshot of a computer program

Description automatically generated

A black background with white text

Description automatically generated

Displaying: -

A black screen with white text

Description automatically generated

Removing: -

A black screen with white text

Description automatically generated

Priority: -

A screenshot of a computer program

Description automatically generated

A screen shot of a computer

Description automatically generated

* **Program that implements Double ended Queue using Linked List:**

**Code:**

#include<iostream>

using namespace std;

class queue {

private:

int data;

queue\* next;

queue\* prev;

public:

void add(int);

void display(void);

int remove(void);

void addf(int);

int remover(void);

};

queue\* front;

queue\* rear;

void queue::display(void) {

queue\* q=front;

if((front==NULL) && (rear==NULL)) {

cout<<"queue is empty"; }

else {

while(q!=NULL) {

cout<<q

->data<<" ";

q=q

->next;

}}}

void queue::add(int num) {

if((front==NULL) && (rear==NULL)) {

front = rear = new queue;

front

->data=num;

front

->next=NULL;

front

->prev=NULL;

}

else {

rear

->next= new queue;

rear

->next

->data=num;

rear

->next

->next=NULL;

rear

->next

->prev=rear;

rear=rear

->next;

}}

int queue::remove(void) {

int num;

if((front==NULL) && (rear==NULL)) {

return

-1;

}

else {

queue\* temp = front;

num = front

->data;

front = front

->next;

if(front!=NULL) {

delete(temp);

front

->prev=NULL;

}

else {

delete(rear);

rear=NULL; }

return num; }}

void queue::addf(int num) {

queue\* temp;

if((front==NULL)&&(rear==NULL)) {

front = rear = new queue;

front

->data=num;

front

->next=NULL;

front

->prev=NULL;

}

else {

temp = front;

front = new queue;

front

->data = num;

front

->prev = NULL;

front

->next=temp;

front

->next

->prev=front;

}}

int queue::remover(void) {

int num;

queue\* temp;

if((front == NULL) && (rear == NULL)) {

return

-1;

}

else {

num = rear

->data;

rear = rear

->prev;

if(rear!=NULL) {

delete(rear

->next);

rear

->next=NULL;

}

else {

delete(front);

front=NULL; }

return num; }}

int main(void) {

int num, option;

queue q;

char ch='y';

front = NULL;

rear = NULL;

while(ch == 'y') {

cout << "\n 1.ADD";

cout << "\n 2.REMOVE";

cout << "\n 3.DISPLAY";

cout << "\n 4.ADD FROM FRONT";

cout << "\n 5.REMOVE FROM REAR";

cout << "\n Enter an Option(1-5): ";

cin >> option;

switch (option) {

case 1: {

cout << "Enter a Number to ADD: ";

cin >> num;

q.add(num);

cout << "\n Do you want to Continue (press y): ";

break; }

case 2: {

num = q.remove();

cout << num;

cout << "\n Do you want to Continue (press y): ";

break; }

case 3: {

q.display();

cout << "\n Do you want to Continue (press y): ";

break; }

case 4: {

cout << "Enter a Number to ADD to FRONT: ";

cin >> num;

q.addf(num);

cout << "\n Do you want to Continue (press y): ";

break; }

case 5: {

num = q.remover();

cout << num;

cout << "\n Do you want to Continue (press y): ";

break;

}

}

}

}

**Output:**

Adding: -

A screen shot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

Displaying: -

A screen shot of a computer

Description automatically generated

Removing: -

A screenshot of a computer program

Description automatically generated

Adding from front: -

A screenshot of a computer program

Description automatically generated

Removing from rear: -

A screen shot of a computer program

Description automatically generated

**Module: - 07**

**Binary search trees**

* **Program that implements Binary search tree for the following operations: insert, Recursive traversal: Preorder, Postorder, Inorder, search, largest node, smallest node, count numbers of nodes:**

**Code:**

#include <iostream>

using namespace std;

class Node {

public:

int data;

Node\* left;

Node\* right;

Node(int value) {

data = value;

left = NULL;

right = NULL;

}

};

class BST {

private:

Node\* root;

Node\* insertRecursive(Node\* root, int value) {

if (root == NULL) {

return new Node(value);

}

if (value < root->data) {

root->left = insertRecursive(root->left, value);

} else if (value > root->data) {

root->right = insertRecursive(root->right, value);

}

return root;

}

void preorderRecursive(Node\* root) {

if (root != NULL) {

cout << root->data << " ";

preorderRecursive(root->left);

preorderRecursive(root->right);

}

}

void inorderRecursive(Node\* root) {

if (root != NULL) {

inorderRecursive(root->left);

cout << root->data << " ";

inorderRecursive(root->right);

}

}

void postorderRecursive(Node\* root) {

if (root != NULL) {

postorderRecursive(root->left);

postorderRecursive(root->right);

cout << root->data << " ";

}

}

Node\* searchRecursive(Node\* root, int value) {

if (root == NULL || root->data == value) {

return root;

}

if (value < root->data) {

return searchRecursive(root->left, value);

}

return searchRecursive(root->right, value);

}

Node\* findLargestNode(Node\* root) {

if (root == NULL || root->right == NULL) {

return root;

}

return findLargestNode(root->right);

}

Node\* findSmallestNode(Node\* root) {

if (root == NULL || root->left == NULL) {

return root;

}

return findSmallestNode(root->left);

}

int countNodes(Node\* root) {

if (root == NULL) {

return 0;

}

return 1 + countNodes(root->left) + countNodes(root->right);

}

public:

BST() {

root = NULL;

}

void insert(int value) {

root = insertRecursive(root, value);

}

void preorderTraversal() {

cout << "Preorder Traversal: ";

preorderRecursive(root);

cout << endl;

}

void inorderTraversal() {

cout << "Inorder Traversal: ";

inorderRecursive(root);

cout << endl;

}

void postorderTraversal() {

cout << "Postorder Traversal: ";

postorderRecursive(root);

cout << endl;

}

bool search(int value) {

return searchRecursive(root, value) != NULL;

}

int largestNode() {

Node\* largest = findLargestNode(root);

return largest ? largest->data : -1; // Return -1 if the tree is empty

}

int smallestNode() {

Node\* smallest = findSmallestNode(root);

return smallest ? smallest->data : -1; // Return -1 if the tree is empty

}

int countNodesInTree() {

return countNodes(root);

}

};

int main() {

BST bst;

int numNodes;

cout << "Enter the number of nodes in the BST: ";

cin >> numNodes;

cout << "Enter the values of the nodes:" << endl;

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

int val;

cin >> val;

bst.insert(val);

}

bst.preorderTraversal();

bst.inorderTraversal();

bst.postorderTraversal();

int searchValue;

cout << "Enter a value to search in the BST: ";

cin >> searchValue;

cout << "Search for value " << searchValue << ": " << (bst.search(searchValue) ? "Found" : "Not found") << endl;

cout << "Largest Node: " << bst.largestNode() << endl;

cout << "Smallest Node: " << bst.smallestNode() << endl;

cout << "Number of nodes in the tree: " << bst.countNodesInTree() << endl;

return 0;

}

**Output:**

**A screenshot of a computer program

Description automatically generated**

**Module: - 08**

**Heap**

* **Program that implements heap tree with methods: Build, Display, Delete:**

**Code:**

#include<iostream>

using namespace std;

class heap {

int arr[7];

int size;

public:

heap() {

size = 0;

}

void insertheap();

void buildheap();

void deleteheap();

void reheapup(int);

void reheapdown(int, int);

void display();

};

void heap::insertheap() {

for (int i = 0; i < 7; i++) {

cout << "Enter the element in heap: ";

cin >> arr[i];

size++;

reheapup(size - 1);

}

}

void heap::reheapup(int i) {

int p;

if (i != 0) {

p = (i - 1) / 2;

if (arr[i] > arr[p]) {

int temp = arr[i];

arr[i] = arr[p];

arr[p] = temp;

reheapup(p);

}

}

}

void heap::buildheap() {

for (int i = 0; i < 7; i++) {

cout << "Enter the element in heap: ";

cin >> arr[i];

size++;

}

for (int w = size / 2 - 1; w >= 0; w--) {

reheapdown(w, size - 1);

}

}

void heap::deleteheap() {

if (size == 0) {

cout << "Heap is empty.\n";

return;

}

int last = size - 1;

arr[0] = arr[last];

size--;

reheapdown(0, last);

cout << "After deletion: ";

display();

}

void heap::reheapdown(int r, int last) {

int lk, rk, lck, lci;

if (r \* 2 + 1 <= last) {

lk = arr[r \* 2 + 1];

if (r \* 2 + 2 <= last)

rk = arr[r \* 2 + 2];

else

rk = 0; // assumption right child is not there

if (lk > rk) {

lck = lk;

lci = r \* 2 + 1;

} else {

lck = rk;

lci = r \* 2 + 2;

}

if (arr[r] < arr[lci]) {

int temp = arr[r];

arr[r] = arr[lci];

arr[lci] = temp;

reheapdown(lci, last);

}

}

}

void heap::display(){

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

cout << "Heap element: " << arr[i] << endl;

}

}

int main() {

heap h;

h.buildheap();

h.insertheap();

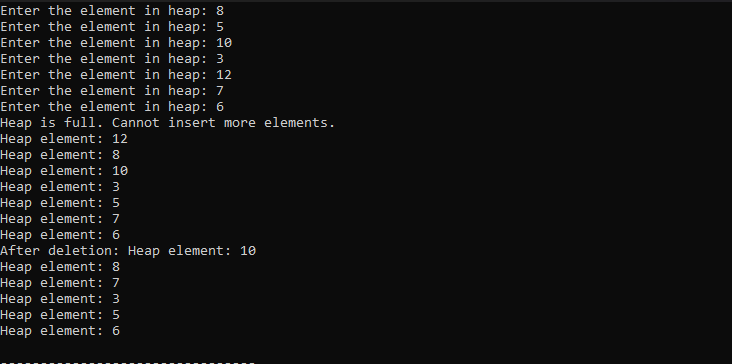
h.display();

h.deleteheap();

return 0;

}

**Output:**



**Module: - 09**

**Graphs**

* **Find a minimum spanning tree using any method Kruskal’s Algorithm or Prim’s Algorithm:**

**Code:**

#include<iostream>

using namespace std;

#define ROW 7

#define COL 7

#define infi 99 //infi for infinity

class prims

{

int graph[ROW][COL],nodes;

public:

prims();

void createGraph();

void primsAlgo();

};

prims :: prims()

{

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

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

graph[i][j]=0;

}

void prims :: createGraph()

{

int i,j;

cout<<"Enter Total Nodes : ";

cin>>nodes;

cout<<"\n\nEnter Adjacency Matrix : \n";

for(i=1;i<=nodes;i++)

for(j=1;j<=nodes;j++)

cin>>graph[i][j];

//Assigning infinity to all graph[i][j] where weight is 0

for(i=1;i<=nodes;i++)

for(j=1;j<=nodes;j++)

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

graph[i][j]=infi;

//Printing graph in matrix form

cout<<"Matrix is:"<<endl;

for(i=1;i<=nodes;i++)

{

for(j=1;j<=nodes;j++)

{

cout<<" "<<graph[i][j];

}

cout<<endl;

}

}

void prims :: primsAlgo()

{

cout<<"Minimum spanning tree is:" ;

int selected[ROW],i,j,ne;

int f,t ,min,x,y;

f=0;

t=1;

for(i=1;i<=nodes;i++)

selected[i]=f;

selected[1]=t;

ne=1;

while(ne <= nodes-1)

{

min=infi;

for(i=1;i<=nodes;i++)

{

if(selected[i]==t)

{

for(j=1;j<=nodes;j++)

{

if(selected[j]==f)

{

if(min > graph[i][j])

{

min=graph[i][j];

x=i;

y=j;

}

}

}

}

}

selected[y]=t;

cout<<"\n"<<x<<" --> "<<y;

ne=ne+1;

}

}

int main(){

prims MST;

cout<<"\nPrims Algorithm to find Minimum Spanning Tree\n";

MST.createGraph();

MST.primsAlgo();

return 0;

}

**Output:**

**A screenshot of a computer program

Description automatically generated**

* **Create a graph (Using Adjacency Matrix):**

**Code:**

#include<iostream>

using namespace std;

class Graph{

int a[10][10];

public:

Graph(){

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

for(int j=0;j<10;j++)

{

a[i][j]=0;

}

}

}

void addEdge(int s,int d){

a[s][d]=1;

a[d][s]=1;

}

void addEdge\_D(int s,int d){

a[s][d]=1;

}

void showEdge(int no){

for(int i=0;i<no;i++){

for(int j=0;j<no;j++){

cout<<a[i][j]<<"\t";

}

cout<<endl;

}

}

};

int main(){

int no,start,destination;

Graph g;

int s;

cout<<"Which type of Graph you implement: \n";

cout<<"1.Undirected Graph\n2.directed Graph\n";

cin>>s;

switch(s){

case 1:

cout<<"How Many Edges Add in Graph :\n";

cin>>no;

cout<<"Enter Edge Start and Destination Point :\n";

for(int i=0;i<no;i++)

{

cin>>start;

cin>>destination;

g.addEdge(start,destination);

}

g.showEdge(no);

return 0;

break;

case 2:

cout<<"How Many Edges Add in Graph :\n";

cin>>no;

cout<<"Enter Edge Start and Destination Point :\n";

for(int i=0;i<no;i++)

{

cin>>start;

cin>>destination;

g.addEdge\_D(start,destination);

}

g.showEdge(no);

return 0;

break;

}

}

**Output:**

**A screenshot of a computer program

Description automatically generated**

**A screenshot of a computer program

Description automatically generated**

* **Implementation of Graph traversal (DFS and BFS):**

**Code:**

#include <iostream>

#define MAX 100

using namespace std;

class Graph {

int V;

int\*\* adjacencyMatrix;

public:

Graph(int vertices) : V(vertices) {

adjacencyMatrix = new int\*[V];

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

adjacencyMatrix[i] = new int[V];

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

adjacencyMatrix[i][j] = 0;

}

}

}

void addEdge(int v, int w) {

adjacencyMatrix[v][w] = 1;

adjacencyMatrix[w][v] = 1;

}

void DFSUtil(int v, bool visited[]) {

visited[v] = true;

cout << v << " ";

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

if (adjacencyMatrix[v][i] && !visited[i]) {

DFSUtil(i, visited);

}

}

}

void DFS(int start) {

bool\* visited = new bool[V];

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

visited[i] = false;

}

DFSUtil(start, visited);

}

};

int main() {

int vertices = 6;

Graph g(vertices);

g.addEdge(0, 1);

g.addEdge(0, 2);

g.addEdge(1, 3);

g.addEdge(2, 4);

g.addEdge(2, 5);

cout << "Depth-First Traversal starting from vertex 0: \n";

g.DFS(0);

return 0;

}

**Output:**

**A screen shot of a black background

Description automatically generated**

**Group Project**

**Sudoku Solver**

* **Implementation of sudoku solver:**

**Code:**

#include <iostream>

using namespace std;

const int N = 9;

class SudokuSolver {

public:

bool solveSudoku(int grid[N][N]) {

int row, col;

if (!findEmptyLocation(grid, row, col))

return true;

for (int num = 1; num <= 9; num++) {

if (isSafe(grid, row, col, num)) {

grid[row][col] = num;

if (solveSudoku(grid))

return true;

grid[row][col] = 0;

}

}

return false;

}

bool findEmptyLocation(int grid[N][N], int &row, int &col) {

for (row = 0; row < N; row++) {

for (col = 0; col < N; col++) {

if (grid[row][col] == 0)

return true;

}

}

return false;

}

bool usedInRow(int grid[N][N], int row, int num) {

for (int col = 0; col < N; col++) {

if (grid[row][col] == num)

return true;

}

return false;

}

bool usedInCol(int grid[N][N], int col, int num) {

for (int row = 0; row < N; row++) {

if (grid[row][col] == num)

return true;

}

return false;

}

bool usedInBox(int grid[N][N], int boxStartRow, int boxStartCol, int num) {

for (int row = 0; row < 3; row++) {

for (int col = 0; col < 3; col++) {

if (grid[row + boxStartRow][col + boxStartCol] == num)

return true;

}

}

return false;

}

bool isSafe(int grid[N][N], int row, int col, int num) {

return !usedInRow(grid, row, num) &&

!usedInCol(grid, col, num) &&

!usedInBox(grid, row - row % 3, col - col % 3, num) &&

grid[row][col] == 0;

}

void printGrid(int grid[N][N]) {

for (int row = 0; row < N; row++) {

for (int col = 0; col < N; col++) {

cout << grid[row][col] << " ";

}

cout << endl;

}

}

};

int main() {

int grid[N][N] = {

{0, 0, 5, 0, 2, 4, 0, 1, 3},

{0, 0, 6, 0, 3, 1, 0, 0, 0},

{0, 0, 1, 0, 8, 9, 5, 0, 7},

{1, 6, 0, 0, 9, 7, 0, 0, 5},

{7, 5, 8, 3, 0, 0, 0, 9, 0},

{0, 0, 9, 8, 0, 5, 0, 0, 0},

{5, 0, 7, 0, 6, 0, 3, 2, 4},

{0, 1, 0, 4, 5, 0, 9, 7, 0},

{0, 4, 3, 0, 0, 2, 0, 0, 0}

};

SudokuSolver solver;

if (solver.solveSudoku(grid)) {

cout << "Sudoku solved:\n";

solver.printGrid(grid);

} else {

cout << "No solution exists";

}

return 0;

}

**Output:**

**A screenshot of a computer

Description automatically generated**