TASK 2:

QUADRATIC PROBING:

#include<iostream>

using namespace std;

class Hashing{

private:

int\* arr;

int size;

public:

Hashing(){

size = 10;

arr = new int[size];

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

arr[i] = 0;

}

}

void insert(int value){

int copy = value;

if (!NotFull()){

cout << "Hash Table is Filled\n";

return;

}

else if (NotFull()){

int index = getindex(value);

if (arr[index] == 0){

arr[index] = value;

}

else if (arr[index] != 0){

findfreelocation(index, 0, value);

}

}

}

void findfreelocation(int index, int i, int value){

int getindex = ((index + (i \* i)) % size);

if (getindex == size){

return;

}

if (arr[getindex] == 0){

arr[getindex] = value;

return;

}

else{

findfreelocation(index, ++i, value);

}

}

int getindex(int key){

return ((key) % size);

}

bool NotFull(){

int count = 0;

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

if (arr[i] == 0){

count++;

}

}

if (count >= 1){

return true;

}

else

{

return false;

}

}

bool search(int value){

int index = getindex(value);

int s = 0;

if (arr[index] == value){

cout << "Element Founded Successfully at index " << index + 1 << "\n";

}

else if (arr[index] != value){

findsearch(index, 0, value, s);

if (s == 1){

return true;

}

else{

return false;

}

}

}

void findsearch(int index, int i, int value, int& s){

int getindex = ((index + (i \* i)) % size);

if (getindex == size){

return;

}

if (arr[getindex] == value){

cout << "Element Founded Successfully at index " << getindex + 1 << "\n";

s = 1;

return;

}

else{

findsearch(index, ++i, value, s);

}

}

bool delete\_element(int value){

int index = getindex(value);

if (arr[index] == value){

arr[index] = 0;

cout << "Element Founded Successfully at index " << index + 1 << " and deleted\n";

return true;

}

else if (arr[index] != value){

findndelete(index, 0, value);

}

return true;

}

void findndelete(int index, int i, int value){

int getindex = ((index + (i \* i)) % size);

if (arr[getindex] == value){

arr[getindex] = 0;

cout << "Element Founded Successfully at index " << getindex + 1 << "\n";

return;

}

else{

findndelete(index, ++i, value);

}

}

void display(){

cout << "Hash Table is: ";

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

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

}

cout << endl;

}

};

int main(){

Hashing obj;

int size;

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

int n;

cin >> n;

obj.insert(n);

}

obj.display();

cout << "Enter number to search: ";

cin >> size;

obj.search(size);

cout << "Enter number to delete: ";

cin >> size;

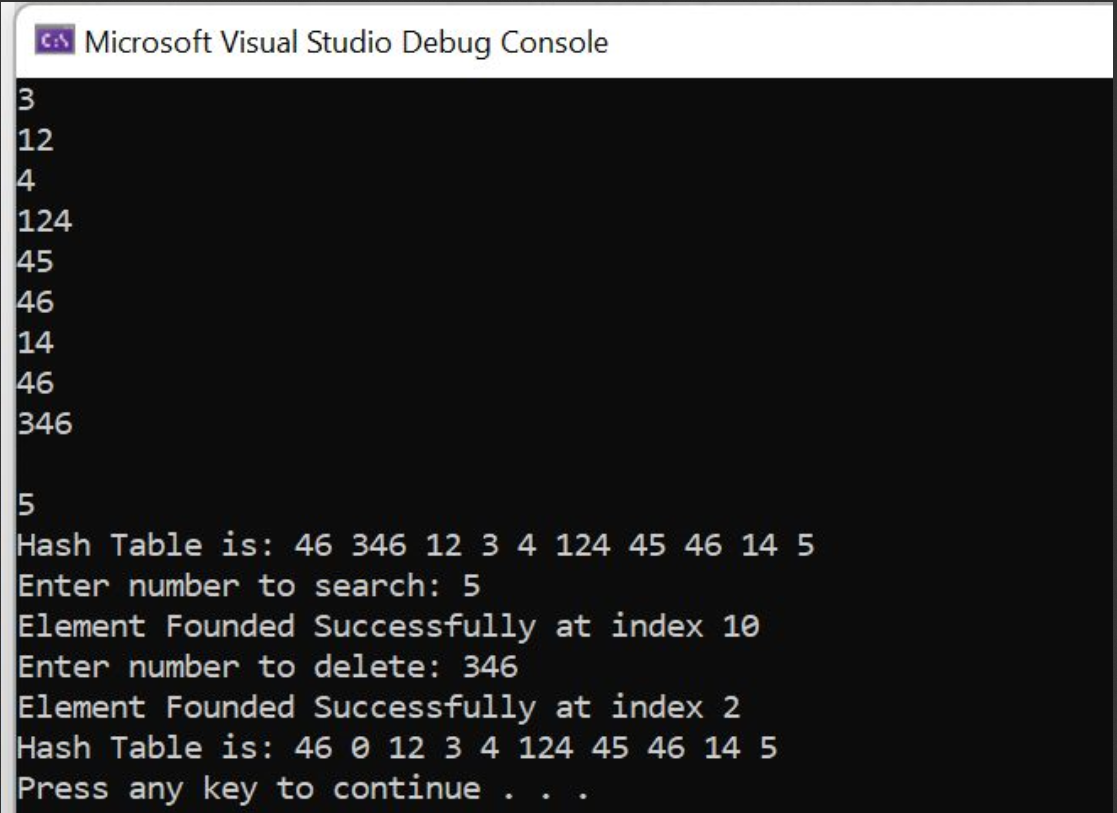
obj.delete\_element(size);

obj.display();

system("pause");

return 0;

}



LINEAR PROBING:

#include<iostream>

using namespace std;

class Hashing{

private:

int\* arr;

int size;

public:

Hashing(){

size = 10;

arr = new int[size];

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

arr[i] = 0;

}

}

void insert(int value){

int copy = value;

if (!NotFull()){

cout << "Hash Table is Filled\n";

return;

}

else if (NotFull()){

int index = getindex(value);

while (true){

if (index == size){

index = wrapround();

continue;

}

else if (arr[index] == 0){

arr[index] = value;

break;

}

index++;

}

}

}

int wrapround(){

return 0;

}

int getindex(int key){

return key % size;

}

bool NotFull(){

int count = 0;

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

if (arr[i] == 0){

count++;

}

}

if (count >= 1){

return true;

}

else{

return false;

}

}

bool search(int value){

int index = getindex(value);

while (true){

if (index == size){

index = wrapround();

continue;

}

else if (arr[index] == value){

cout << "Element Found successfully\n";

return true;

break;

}

index++;

}

return false;

}

bool delete\_element(int value){

int index = getindex(value);

while (true){

if (index == size){

index = wrapround();

continue;

}

else if (arr[index] == value){

arr[index] = 0;

cout << "Element Deleted successfully\n";

return true;

break;

}

index++;

}

return false;

}

void display(){

cout << "Hash Table is: ";

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

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

}

cout << endl;

}

};

int main(){

Hashing obj;

int n;

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

cin >> n;

obj.insert(n);

}

obj.display();

obj.search(22);

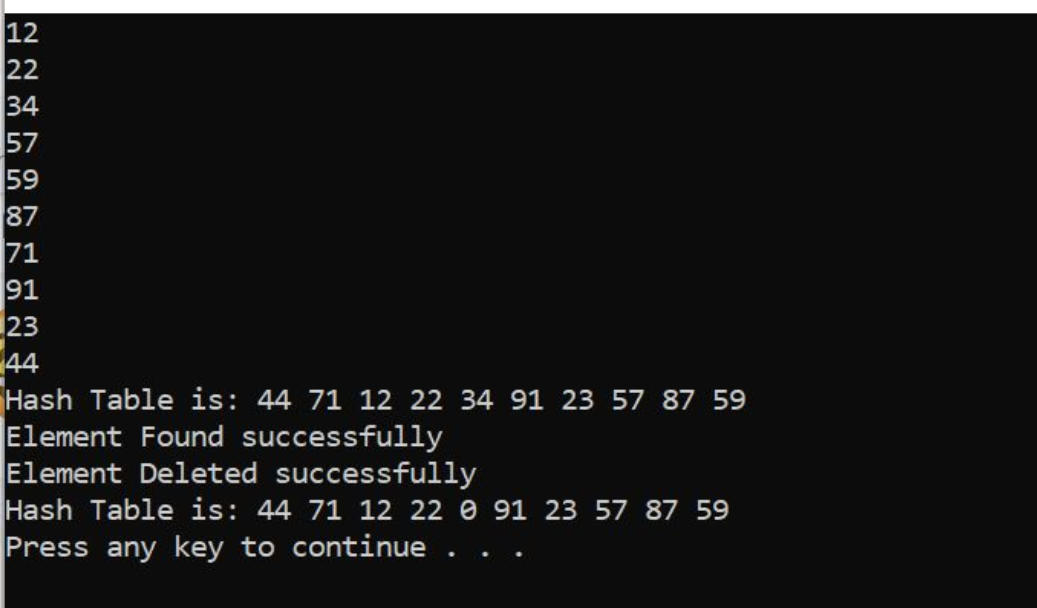
obj.delete\_element(34);

obj.display();

system("pause");

return 0;

}



TASK 1:

BFS:

BFS:

#include<iostream>

using namespace std;

class Queue {

int size;

int front, rear;

int\* arr;

public:

//Constructor for Queue class

Queue(int size) {

this->size = size;

front = rear = 0;

arr = new int[size];

}

// Function to check if Queue is Empty or Not

bool isEmpty() {

if (front == rear) {

return 1;

}

return 0;

}

// Function to check if Queue is Full or Not

bool isFull() {

if (rear == size - 1) {

return 1;

}

return 0;

}

// Funtion to insert elements in Queue

void enqueue(int val) {

if (isFull()) {

cout << "Queue is full!" << endl;

}

else {

rear++;

arr[rear] = val;

}

}

// Funtion to delete(pop) elements from Queue

int dequeue() {

if (isEmpty()) {

cout << "Queue is Empty!" << endl;

return -1;

}

else {

front++;

int val = arr[front];

return val;

}

}

// To display elements of the Queue

void display() {

if (isEmpty()) {

cout << "Queue is empty!" << endl;

}

else {

cout << "The elements of stack are as follows: - " << endl;

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

if (i == -1) {

continue;

}

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

}

cout << endl;

}

}

};

int main() {

bool chk = 0;

// Initializing Queue (Array Implementation)

Queue q(400);

// BFS

int node = 0;

int i = 0;

int visited[13] = { 0,0,0,0,0,0,0,0,0,0,0,0,0 };

int a[13][13] = {

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

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

{ 1,0,0,0,0,1,0,0,0,0,0,0,0 }, //2

{ 0,1,0,0,0,0,1,1,0,0,0,0,0 }, //3

{ 0,1,0,0,0,0,0,0,1,0,0,0,0 }, //4

{ 0,0,1,0,0,0,0,0,0,1,0,0,0 }, //5

{ 0,0,0,1,0,0,0,0,0,0,1,1,0 }, //6

{ 0,0,0,1,0,0,0,0,1,0,0,0,0 }, //7

{ 0,0,0,1,0,0,0,0,0,1,0,0,1 }, //8

{ 0,0,0,0,0,1,0,0,0,0,0,0,0 }, //9

{ 0,0,0,0,0,0,1,0,0,0,0,0,0 }, //10

{ 0,0,0,0,0,0,1,0,0,0,0,0,0 }, //11

{ 0,0,0,0,0,0,0,0,1,0,0,0,0 }, //12

};

cout << i << " ";

visited[i] = 1;

q.enqueue(i); // Enqueue i for exploration

while (!q.isEmpty()) {

int node = q.dequeue();

for (int j = 0; j < 13; j++) {

if (chk == 1) {

break;

}

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

cout << j << " ";

visited[j] = 1;

q.enqueue(j);

if (j == 6) {

chk = 1;

}

}

}

}

cout << endl;

system("pause");

return 0;

}

DFS:

#include<iostream>

using namespace std;

void DFS(int &i, int\* visited, int a[][13], int nodes, int path, bool& check) {

if (check == 0) {

if (visited[i] == 0) {

cout << i << " ";

visited[i] = 1;

if (i == path) {

check = 1;

return;

}

else if (i != path) {

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

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

DFS(j, visited, a, nodes, path, check);

}

}

}

}

}

}

int main() {

bool check = 0;

// DFS

int i = 0;

int visited[13] = { 0,0,0,0,0,0,0,0,0,0,0,0,0 };

int a[13][13] = {

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

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

{ 1,0,0,0,0,1,0,0,0,0,0,0,0 }, //2

{ 0,1,0,0,0,0,1,1,0,0,0,0,0 }, //3

{ 0,1,0,0,0,0,0,0,1,0,0,0,0 }, //4

{ 0,0,1,0,0,0,0,0,0,1,0,0,0 }, //5

{ 0,0,0,1,0,0,0,0,0,0,1,1,0 }, //6

{ 0,0,0,1,0,0,0,0,1,0,0,0,0 }, //7

{ 0,0,0,1,0,0,0,0,0,1,0,0,1 }, //8

{ 0,0,0,0,0,1,0,0,0,0,0,0,0 }, //9

{ 0,0,0,0,0,0,1,0,0,0,0,0,0 }, //10

{ 0,0,0,0,0,0,1,0,0,0,0,0,0 }, //11

{ 0,0,0,0,0,0,0,0,1,0,0,0,0 }, //12

};

DFS(i, visited, a, 13, 6, check);

cout << endl;

system("pause");

return 0;

}