**Lab Assignment 8**

**Data Structures and Algorithms**

**Diya Goyal 102215255 2NC11**

Note: Use C/C++ or JAVA programming language.

**Q1. Write a program to create a binary tree. The nodes of the tree are input from the user.**

**a) Find out the height of a given binary tree efficiently.**

**b) Write a recursive program to traverse through the binary tree elements.**

/\*

Roll Number: 102215255

Name: Diya Goyal

Description: Q1. Write a program to create a binary tree. The nodes of the tree are input from the user.

a) Find out the height of a given binary tree efficiently.

b) Write a recursive program to traverse through the binary tree elements.

Acknowledgement: GeeksforGeeks

\*/

#include <bits/stdc++.h>

using namespace std;

class TreeNode {

public:

int data;

TreeNode\* left;

TreeNode\* right;

};

int getHeight(TreeNode\* node) {

if (node == NULL)

return 0;

if (node->left || node->right)

return 1 + max(getHeight(node->left), getHeight(node->right));

}

TreeNode\* createNewNode(int data) {

TreeNode\* newNode = new TreeNode();

newNode->data = data;

newNode->left = NULL;

newNode->right = NULL;

return (newNode);

}

void inOrderTraversal(TreeNode\* root) {

if (!root) {

return;

}

inOrderTraversal(root->left);

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

inOrderTraversal(root->right);

}

int main() {

TreeNode\* root = createNewNode(4);

root->left = createNewNode(27);

root->right = createNewNode(60);

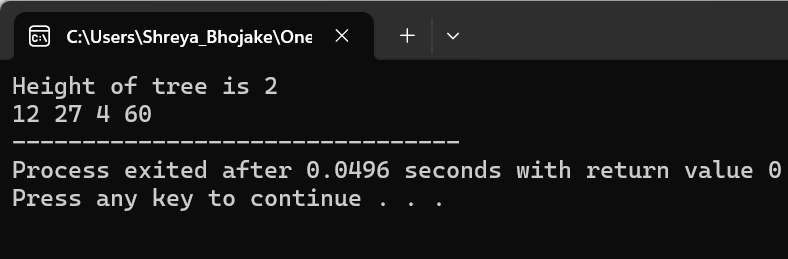
root->left->left = createNewNode(12);

cout << "Height of tree is " << getHeight(root) << endl;

inOrderTraversal(root);

return 0;

}



**Q2. Write a program for creating a binary search tree (BST) from a given array of elements.**

**a) Insert an element in a BST.**

**b) Write a recursive as well as iterative program for search in a BST.**

**c) Write a program to print the elements of a BST after performing In-order Traversal (Note that you get a sorted list of elements after in-order traversal).**

/\*

Roll Number: 102215255

Name: Diya Goyal

Description: Q2. Write a program for creating a binary search tree (BST) from a given array of elements.

a) Insert an element in a BST.

b) Write a recursive as well as iterative program for search in a BST.

c) Write a program to print the elements of a BST after performing In-order Traversal (Note that you get a sorted list of elements after in-order traversal).

Acknowledgement: GeeksforGeeks

\*/

#include <bits/stdc++.h>

using namespace std;

class TreeNode {

public:

int data;

TreeNode\* left;

TreeNode\* right;

TreeNode(int val) {

data = val;

left = NULL;

right = NULL;

}

};

TreeNode\* insertNode(TreeNode\* root, int value) {

if (!root) {

return new TreeNode(value);

}

if (value > root->data) {

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

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

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

}

return root;

}

bool searchRecursive(TreeNode\* root, int val) {

if (root == nullptr)

return false;

if (root->data == val)

return true;

if (val < root->data)

return searchRecursive(root->left, val);

else

return searchRecursive(root->right, val);

}

bool searchIterative(TreeNode\* root, int key) {

while (root != NULL) {

if (key > root->data)

root = root->right;

else if (key < root->data)

root = root->left;

else

return true;

}

return false;

}

void inOrderTraversal(TreeNode\* root) {

if (!root) {

return;

}

inOrderTraversal(root->left);

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

inOrderTraversal(root->right);

}

int main() {

TreeNode\* root = NULL;

root = insertNode(root, 10);

root = insertNode(root, 40);

root = insertNode(root, 30);

root = insertNode(root, 20);

root = insertNode(root, 60);

inOrderTraversal(root);

cout << endl;

int key;

cout << "Enter element to search" << endl;

cin >> key;

if (searchIterative(root, key))

cout << "Element is found (using iterative)" << endl;

else

cout << "Element is not found (using iterative)" << endl;

if (searchRecursive(root, key))

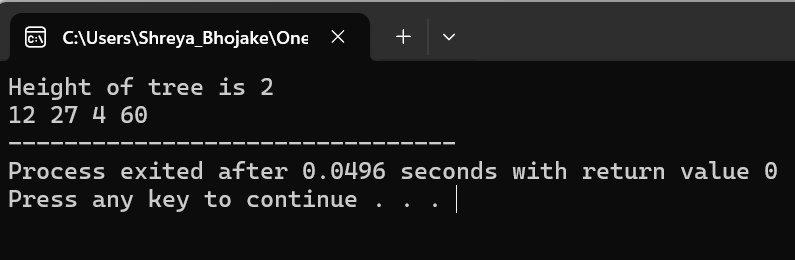
cout << "Element is found (using recursive)" << endl;

else

cout << "Element is not found (using recursive)" << endl;

return 0;

}



**Q3. Write a program to find the number of connected components in a given graph using depth first or breadth first traversal. Assume that the graph is given as an adjacency matrix.**

/\*

Roll Number: 102215255

Name: Diya Goyal

Description: Q3. Write a program to find the number of connected components in a given graph using depth first or breadth first traversal. Assume that the graph is given as an adjacency matrix.

Acknowledgement: GeeksforGeeks

\*/

#include <bits/stdc++.h>

using namespace std;

void depthFirstSearch(int node, vector<int> adjacencyList[], int visited[]) {

visited[node] = 1;

for (auto neighbor : adjacencyList[node]) {

if (!visited[neighbor]) {

depthFirstSearch(neighbor, adjacencyList, visited);

}

}

}

int numberOfConnectedComponents(vector<vector<int>> adjacencyMatrix, int V) {

vector<int> adjacencyList[V];

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

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

if (adjacencyMatrix[i][j] == 1 && i != j) {

adjacencyList[i].push\_back(j);

adjacencyList[j].push\_back(i);

}

}

}

int visited[V] = {0};

int count = 0;

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

if (!visited[i]) {

count++;

depthFirstSearch(i, adjacencyList, visited);

}

}

return count;

}

int main() {

vector<vector<int>> adjacencyMatrix{

{1, 0, 1},

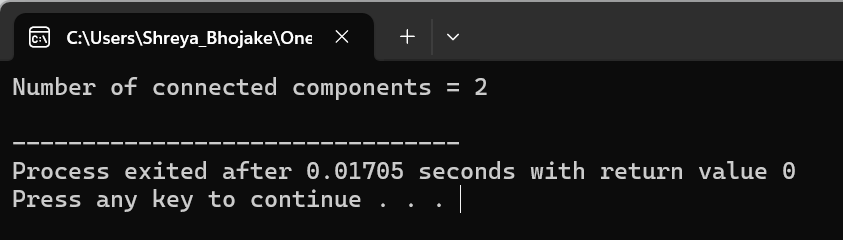
{0, 1, 0},

{1, 0, 1}};

cout << "Number of connected components = " << numberOfConnectedComponents(adjacencyMatrix, 3) << endl;

return 0;

}



**Q4. Implement Dijkstra's algorithm to find the shortest path between a single pair of source and destination in a given graph. You may represent the graph as you require.**

/\*

Roll Number: 102215255

Name: Diya Goyal

Description: Implement Dijkstra's algorithm to find the shortest path between a single pair of source and destination in a given graph. You may represent the graph as you require.

Acknowledgement: GeeksforGeeks

\*/

#include <iostream>

#include <vector>

#include <queue>

using namespace std;

struct Edge {

int destination;

int weight;

};

struct Vertex {

int vertexIndex;

int distance;

bool operator>(const Vertex& other) const {

return distance > other.distance;

}

};

vector<int> dijkstra(const vector<vector<Edge>>& graph, int source) {

int numVertices = graph.size();

vector<int> distances(numVertices, INT\_MAX);

vector<bool> visited(numVertices, false);

priority\_queue<Vertex, vector<Vertex>, greater<Vertex>> pq;

distances[source] = 0;

pq.push({source, 0});

while (!pq.empty()) {

Vertex currentVertex = pq.top();

pq.pop();

if (visited[currentVertex.vertexIndex]) continue;

visited[currentVertex.vertexIndex] = true;

for (const Edge& edge : graph[currentVertex.vertexIndex]) {

int neighbor = edge.destination;

int weight = edge.weight;

if (!visited[neighbor] && distances[currentVertex.vertexIndex] + weight < distances[neighbor]) {

distances[neighbor] = distances[currentVertex.vertexIndex] + weight;

pq.push({neighbor, distances[neighbor]});

}

}

}

return distances;

}

int main() {

int numVertices = 5;

vector<vector<Edge>> graph(numVertices);

graph[0].push\_back({1, 10});

graph[0].push\_back({3, 5});

graph[1].push\_back({2, 1});

graph[1].push\_back({3, 2});

graph[2].push\_back({4, 4});

graph[3].push\_back({1, 3});

graph[3].push\_back({2, 9});

graph[3].push\_back({4, 2});

int source = 0;

int destination = 3;

vector<int> shortestDistances = dijkstra(graph, source);

cout << "Shortest distance from " << source << " to " << destination << ": ";

if (shortestDistances[destination] != INT\_MAX) {

cout << shortestDistances[destination] << endl;

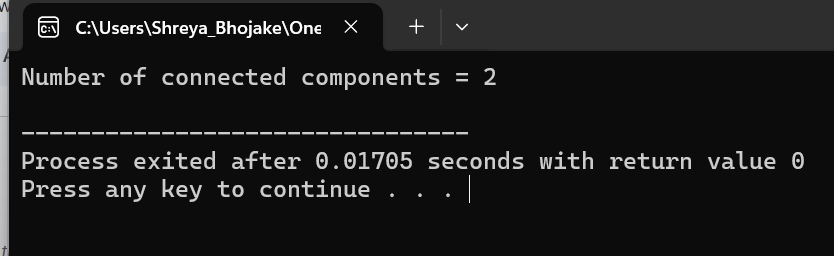
} else {

cout << "No path exists" << endl;

}

return 0;

}



**Q5. Write a program to demonstrate Heap Sort.**

/\*

Roll Number: 102215255

Name: Diya Goyal

Description: . Write a program to demonstrate Heap Sort.

Acknowledgement: GeeksforGeeks

\*/

#include <bits/stdc++.h>

using namespace std;

void heapify(vector<int>& array, int size, int index) {

int right = 2 \* index + 2;

int left = 2 \* index + 1;

int largestIndex = index;

if (left < size && array[left] > array[largestIndex]) {

largestIndex = left;

}

if (right < size && array[right] > array[largestIndex]) {

largestIndex = right;

}

if (largestIndex != index) {

swap(array[index], array[largestIndex]);

heapify(array, size, largestIndex);

}

}

void heapSort(vector<int>& array, int size) {

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

heapify(array, size, i);

}

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

swap(array[0], array[i]);

heapify(array, i, 0);

}

}

void printArray(vector<int>& array, int size) {

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

cout << array[i] << " ";

}

cout << "\n";

}

int main() {

vector<int> array = {1, 3, 5, 4, 6, 13, 10, 9, 17, 15, 8};

int size = array.size();

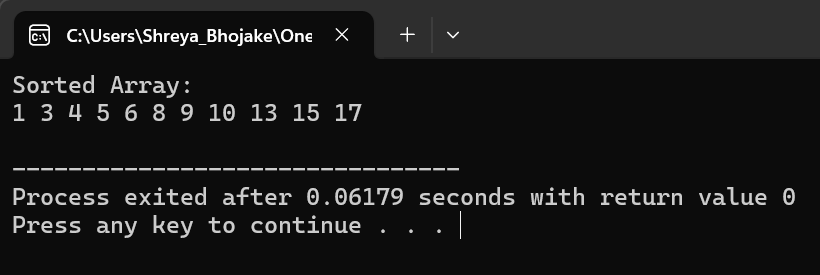
heapSort(array, size);

cout << "Sorted Array:\n";

printArray(array, size);

return 0;

}



**Q6. Write a program to implement Hash Table and chaining to avoid collisions and perform the following functions:**

* **Insert a Key**
* **Search a Key**
* **Delete a Key**

/\*

Roll Number: 102215255

Name: Diya Goyal

Description: Write a program to implement Hash Table and chaining to avoid collisions and perform the following functions:

* Insert a Key
* Search a Key
* Delete a Key

Acknowledgement: GeeksforGeeks\*/

#include <iostream>

#include <list>

#include <vector>

using namespace std;

class HashTable {

private:

int size;

vector<list<int>> table;

int hashFunction(int key) {

return key % size;

}

public:

HashTable(int tableSize) {

size = tableSize;

table.resize(size);

}

void insert(int key) {

int index = hashFunction(key);

table[index].push\_back(key);

}

bool search(int key) {

int index = hashFunction(key);

for (const auto& k : table[index]) {

if (k == key)

return true;

}

return false;

}

void remove(int key) {

int index = hashFunction(key);

table[index].remove(key);

}

};

int main() {

HashTable ht(10);

ht.insert(10);

ht.insert(20);

ht.insert(30);

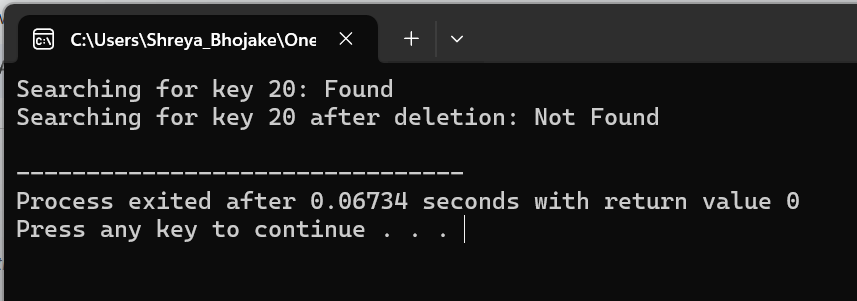
cout << "Searching for key 20: " << (ht.search(20) ? "Found" : "Not Found") << endl;

ht.remove(20);

cout << "Searching for key 20 after deletion: " << (ht.search(20) ? "Found" : "Not Found") << endl;

return 0;

}



**Q7. Write a program to demonstrate the Linear Probing for avoiding collisions in a Hash table.**

/\*

Roll Number: 102215255

Name: Diya Goyal

Description: . Write a program to demonstrate the Linear Probing for avoiding collisions in a Hash table.

Acknowledgement: GeeksforGeeks

\*/

#include <iostream>

using namespace std;

class HashTable {

private:

int tableSize;

int\* hashTable;

int hashFunction(int key) {

return key % tableSize;

}

public:

HashTable(int size) {

tableSize = size;

hashTable = new int[tableSize];

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

hashTable[i] = -1;

}

void insert(int key) {

int index = hashFunction(key);

int originalIndex = index;

while (hashTable[index] != -1) {

index = (index + 1) % tableSize;

if (index == originalIndex) {

cout << "Hash Table is full. Cannot insert key " << key << endl;

return;

}

}

hashTable[index] = key;

}

bool search(int key) {

int index = hashFunction(key);

int originalIndex = index;

while (hashTable[index] != -1 && hashTable[index] != key) {

index = (index + 1) % tableSize;

if (index == originalIndex)

return false;

}

if (hashTable[index] == key)

return true;

return false;

}

void remove(int key) {

int index = hashFunction(key);

int originalIndex = index;

while (hashTable[index] != -1 && hashTable[index] != key) {

index = (index + 1) % tableSize;

if (index == originalIndex) {

cout << "Key " << key << " not found in Hash Table." << endl;

return;

}

}

if (hashTable[index] == key)

hashTable[index] = -1;

else

cout << "Key " << key << " not found in Hash Table." << endl;

}

};

int main() {

HashTable hashTable(10);

hashTable.insert(10);

hashTable.insert(20);

hashTable.insert(30);

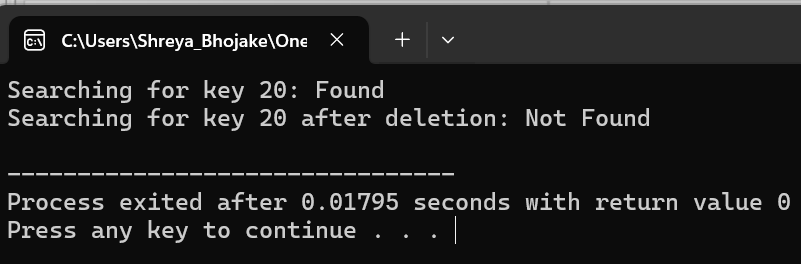
cout << "Searching for key 20: " << (hashTable.search(20) ? "Found" : "Not Found") << endl;

hashTable.remove(20);

cout << "Searching for key 20 after deletion: " << (hashTable.search(20) ? "Found" : "Not Found") << endl;

return 0;

}



**Q8. Write a program to demonstrate the Quadratic Probing for avoiding collisions in a Hash table.**

/\*

Roll Number: 102215255

Name: Diya Goyal

Description: Write a program to demonstrate the Quadratic Probing for avoiding collisions in a Hash table

Acknowledgement: GeeksforGeeks

\*/

#include <iostream>

using namespace std;

class HashTable {

private:

int tableSize;

int\* hashTable;

int hashFunction(int key) {

return key % tableSize;

}

public:

HashTable(int size) {

tableSize = size;

hashTable = new int[tableSize];

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

hashTable[i] = -1;

}

void insert(int key) {

int index = hashFunction(key);

int originalIndex = index;

int i = 1;

while (hashTable[index] != -1) {

index = (originalIndex + i \* i) % tableSize;

++i;

if (i == tableSize) {

cout << "Hash Table is full. Cannot insert key " << key << endl;

return;

}

}

hashTable[index] = key;

}

bool search(int key) {

int index = hashFunction(key);

int originalIndex = index;

int i = 1;

while (hashTable[index] != -1 && hashTable[index] != key) {

index = (originalIndex + i \* i) % tableSize;

++i;

if (i == tableSize || index == originalIndex)

return false;

}

if (hashTable[index] == key)

return true;

return false;

}

void remove(int key) {

int index = hashFunction(key);

int originalIndex = index;

int i = 1;

while (hashTable[index] != -1 && hashTable[index] != key) {

index = (originalIndex + i \* i) % tableSize;

++i;

if (i == tableSize || index == originalIndex) {

cout << "Key " << key << " not found in Hash Table." << endl;

return;

}

}

if (hashTable[index] == key)

hashTable[index] = -1;

else

cout << "Key " << key << " not found in Hash Table." << endl;

}

};

int main() {

HashTable hashTable(10);

hashTable.insert(10);

hashTable.insert(20);

hashTable.insert(30);

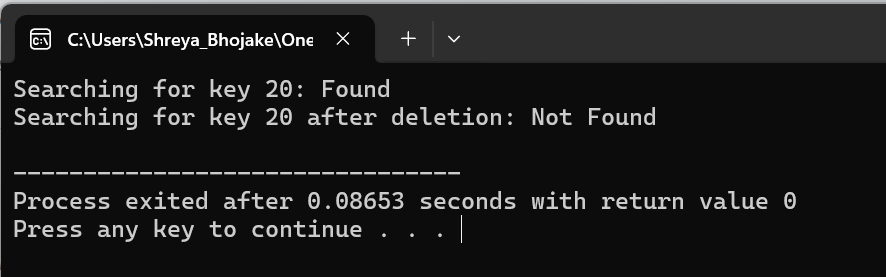
cout << "Searching for key 20: " << (hashTable.search(20) ? "Found" : "Not Found") << endl;

hashTable.remove(20);

cout << "Searching for key 20 after deletion: " << (hashTable.search(20) ? "Found" : "Not Found") << endl;

return 0;

}



**Q9. Write a program to demonstrate Double Hashing.**

/\*

Roll Number: 102215255

Name: Diya Goyal

Description: Write a program to demonstrate Double Hashing.

Acknowledgement: GeeksforGeeks

\*/

#include <iostream>

using namespace std;

class HashTable {

private:

int tableSize;

int\* hashTable;

int hashFunction(int key) {

return key % tableSize;

}

int secondHashFunction(int key) {

int prime = 7;

return prime - (key % prime);

}

public:

HashTable(int size) {

tableSize = size;

hashTable = new int[tableSize];

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

hashTable[i] = -1;

}

void insert(int key) {

int index = hashFunction(key);

if (hashTable[index] != -1) {

int step = secondHashFunction(key);

int i = 1;

while (hashTable[(index + i \* step) % tableSize] != -1)

++i;

index = (index + i \* step) % tableSize;

}

hashTable[index] = key;

}

bool search(int key) {

int index = hashFunction(key);

if (hashTable[index] != key) {

int step = secondHashFunction(key);

int i = 1;

while (hashTable[(index + i \* step) % tableSize] != key && hashTable[(index + i \* step) % tableSize] != -1)

++i;

if (hashTable[(index + i \* step) % tableSize] == key)

return true;

return false;

}

return true;

}

void remove(int key) {

int index = hashFunction(key);

if (hashTable[index] != key) {

int step = secondHashFunction(key);

int i = 1;

while (hashTable[(index + i \* step) % tableSize] != key && hashTable[(index + i \* step) % tableSize] != -1)

++i;

if (hashTable[(index + i \* step) % tableSize] == key)

hashTable[(index + i \* step) % tableSize] = -1;

return;

}

hashTable[index] = -1;

}

};

int main() {

HashTable hashTable(10);

hashTable.insert(10);

hashTable.insert(20);

hashTable.insert(30);

cout << "Searching for key 20: " << (hashTable.search(20) ? "Found" : "Not Found") << endl;

hashTable.remove(20);

cout << "Searching for key 20 after deletion: " << (hashTable.search(20) ? "Found" : "Not Found") << endl;

return 0;

}

