

**RAMANUJAN COLLEGE**

**University Of Delhi**

**NAAC Grade A++ with CGPA 3.71**

**Algorithms and Advanced Data**

**Structure**

**ASSIGNMENT**

**NAME – Aditya Swaroop**

**ROLL NO – 20221405**

**EXAM ROLL NO - 22020570028**

**SEMESTER – 5th**

**SUBMITTED TO – ASMITA MA’AM**

## 1. Write a program to sort the elements of an array using Randomized Quick Sort (the program should

## report the number of comparisons).

## A. Source code

## B. Screenshot of the input and sorted array

## C. Number of comparisons reported

#include <iostream>

#include <cstdlib> // For rand() and srand()

#include <ctime> // For time()

using namespace std;

void swap(int& a, int& b) {

int temp = a;

a = b;

b = temp;

}

int partition(int arr[], int low, int high, int& comparisons) {

int randomPivotIndex = low + rand() % (high - low + 1);

swap(arr[randomPivotIndex], arr[high]); // Move the random pivot to the end

int pivot = arr[high];

int i = low - 1;

for (int j = low; j < high; ++j) {

comparisons++; // Count comparisons

if (arr[j] <= pivot) {

i++;

swap(arr[i], arr[j]);

} }

swap(arr[i + 1], arr[high]);

return i + 1;}

void randomizedQuickSort(int arr[], int low, int high, int& comparisons) {

if (low < high) {

int pivotIndex = partition(arr, low, high, comparisons);

randomizedQuickSort(arr, low, pivotIndex - 1, comparisons);

randomizedQuickSort(arr, pivotIndex + 1, high, comparisons);

}

}

int main() {

srand(time(0)); // Seed for random number generator

int n;

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

cin >> n;

int arr[n];

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

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

cin >> arr[i]; }

int comparisons = 0;

randomizedQuickSort(arr, 0, n - 1, comparisons);

cout << "Sorted array: ";

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

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

cout << endl;

cout << "Number of comparisons: " << comparisons << endl;

return 0;

}

A screenshot of a computer program

Description automatically generated

# 2. Write a program to find the ith smallest element of an array using Randomized Select.

#include <iostream>

#include <cstdlib> // For rand()

#include <ctime> // For srand()

using namespace std;

void swap(int& a, int& b) {

int temp = a;

a = b;

b = temp;

}

int partition(int arr[], int low, int high) {

int randomPivotIndex = low + rand() % (high - low + 1);

swap(arr[randomPivotIndex], arr[high]);

int pivot = arr[high];

int i = low - 1;

for (int j = low; j < high; ++j) {

if (arr[j] <= pivot) {

i++;

swap(arr[i], arr[j]);

} }

swap(arr[i + 1], arr[high]);

return i + 1;

}

// Randomized Select function to find the i-th smallest element

int randomizedSelect(int arr[], int low, int high, int i) {

if (low == high) {

return arr[low];

}

int pivotIndex = partition(arr, low, high);

int k = pivotIndex - low + 1; // Rank of the pivot element

if (i == k) {

return arr[pivotIndex]; // Found the i-th smallest element

} else if (i < k) {

return randomizedSelect(arr, low, pivotIndex - 1, i); // Search in the left subarray

} else {

return randomizedSelect(arr, pivotIndex + 1, high, i - k); // Search in the right subarray

}}

int main() {

srand(time(0)); // Seed for random number generator

int n, i;

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

cin >> n;

int arr[n];

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

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

cin >> arr[j]; }

cout << "Enter the value of i (1-based index): ";

cin >> i;

if (i < 1 || i > n) {

cout << "Invalid input. i must be between 1 and " << n << ".\n";

return 1;}

int result = randomizedSelect(arr, 0, n - 1, i);

cout << "The " << i << "-th smallest element is: " << result << endl;

return 0;}

A screenshot of a computer program

Description automatically generated

# 3. Write a program to determine the minimum spanning tree of a graph using Kruskal’s algorithm.

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

struct Edge {

int src, dest, weight;

};

struct Subset {

int parent;

int rank;

};

// Function to find the parent of a node (with path compression)

int find(Subset subsets[], int node) {

if (subsets[node].parent != node) {

subsets[node].parent = find(subsets, subsets[node].parent);

}

return subsets[node].parent;

}

// Function to perform union of two subsets

void unionSubsets(Subset subsets[], int u, int v) {

int rootU = find(subsets, u);

int rootV = find(subsets, v);

if (subsets[rootU].rank < subsets[rootV].rank) {

subsets[rootU].parent = rootV;

} else if (subsets[rootU].rank > subsets[rootV].rank) {

subsets[rootV].parent = rootU;

} else {

subsets[rootV].parent = rootU;

subsets[rootU].rank++;

}

}

// Comparator function to sort edges by weight

bool compareEdges(Edge a, Edge b) {

return a.weight < b.weight;

}

// Kruskal's algorithm to find the MST

void kruskalMST(vector<Edge>& edges, int V) {

// Sort edges by weight

sort(edges.begin(), edges.end(), compareEdges);

// Create a subset array for union-find

Subset\* subsets = new Subset[V];

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

subsets[v].parent = v;

subsets[v].rank = 0;

}

vector<Edge> mst; // To store the MST edges

int mstWeight = 0;

for (const auto& edge : edges) {

int uRoot = find(subsets, edge.src);

int vRoot = find(subsets, edge.dest);

if (uRoot != vRoot) {

mst.push\_back(edge);

mstWeight += edge.weight;

unionSubsets(subsets, uRoot, vRoot);

}

}

cout << "Edges in the Minimum Spanning Tree:\n";

for (const auto& edge : mst) {

cout << edge.src << " -- " << edge.dest << " == " << edge.weight << endl;

}

cout << "Total weight of the Minimum Spanning Tree: " << mstWeight << endl;

delete[] subsets; // Clean up

}

int main() {

int V, E;

cout << "Enter the number of vertices: ";

cin >> V;

cout << "Enter the number of edges: ";

cin >> E;

vector<Edge> edges(E);

cout << "Enter the edges (source, destination, weight):\n";

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

cin >> edges[i].src >> edges[i].dest >> edges[i].weight; }

kruskalMST(edges, V);

return 0;}

A screenshot of a computer program

Description automatically generated

# 4. Write a program to implement the Bellman-Ford algorithm to find the shortest paths from a given source node to all other nodes in a graph.

#include <iostream>

#include <vector>

#include <climits>

using namespace std;

struct Edge {

int src, dest, weight;

};

// Function to implement Bellman-Ford algorithm

void bellmanFord(int V, int E, vector<Edge>& edges, int source) {

// Distance array to store the shortest distance from source to each vertex

vector<int> distance(V, INT\_MAX);

distance[source] = 0;

// Relax all edges V-1 times

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

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

int u = edges[j].src;

int v = edges[j].dest;

int weight = edges[j].weight;

if (distance[u] != INT\_MAX && distance[u] + weight < distance[v]) {

distance[v] = distance[u] + weight;

}

} }

// Check for negative weight cycles

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

int u = edges[j].src;

int v = edges[j].dest;

int weight = edges[j].weight;

if (distance[u] != INT\_MAX && distance[u] + weight < distance[v]) {

cout << "Graph contains a negative weight cycle.\n";

return;

}

}

// Print the distances

cout << "Vertex\tDistance from Source\n";

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

if (distance[i] == INT\_MAX) {

cout << i << "\tINF\n";

} else {

cout << i << "\t" << distance[i] << "\n";

}}}

int main() {

int V, E;

cout << "Enter the number of vertices: ";

cin >> V;

cout << "Enter the number of edges: ";

cin >> E;

vector<Edge> edges(E);

cout << "Enter the edges (source, destination, weight):\n";

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

cin >> edges[i].src >> edges[i].dest >> edges[i].weight;

}

int source;

cout << "Enter the source vertex: ";

cin >> source;

bellmanFord(V, E, edges, source);

return 0;

}

A screenshot of a computer program

Description automatically generated

# 5. Write a program to implement a B-Tree.

#include <iostream>

#include <vector>

using namespace std;

// B-Tree Node

class BTreeNode {

public:

vector<int> keys; // Keys in the node

vector<BTreeNode\*> children; // Child pointers

int t; // Minimum degree (defines range for number of keys)

bool leaf; // True if leaf node

// Constructor

BTreeNode(int t, bool leaf) {

this->t = t;

this->leaf = leaf;

}

// Function to traverse the tree

void traverse() {

for (size\_t i = 0; i < keys.size(); i++) {

if (!leaf)

children[i]->traverse();

cout << keys[i] << " ";

}

if (!leaf)

children[keys.size()]->traverse();

}

// Function to search for a key in the subtree rooted at this node

BTreeNode\* search(int k) {

size\_t i = 0;

while (i < keys.size() && k > keys[i])

i++;

if (i < keys.size() && keys[i] == k)

return this;

if (leaf)

return nullptr;

return children[i]->search(k);

}

// Splits a full child

void splitChild(int i, BTreeNode\* y) {

BTreeNode\* z = new BTreeNode(y->t, y->leaf);

z->keys.assign(y->keys.begin() + t, y->keys.end());

if (!y->leaf)

z->children.assign(y->children.begin() + t, y->children.end());

keys.insert(keys.begin() + i, y->keys[t - 1]);

children.insert(children.begin() + i + 1, z);

y->keys.resize(t - 1);

if (!y->leaf)

y->children.resize(t);

}

// Inserts a non-full node

void insertNonFull(int k) {

int i = keys.size() - 1;

if (leaf) {

while (i >= 0 && k < keys[i]) {

i--;

}

keys.insert(keys.begin() + i + 1, k);

} else {

while (i >= 0 && k < keys[i]) {

i--;

}

i++;

if (children[i]->keys.size() == 2 \* t - 1) {

splitChild(i, children[i]);

if (k > keys[i])

i++;

}

children[i]->insertNonFull(k);

}

}

};

// B-Tree

class BTree {

private:

BTreeNode\* root;

int t;

public:

// Constructor

BTree(int t) {

this->t = t;

root = nullptr;

}

// Traverse the tree

void traverse() {

if (root)

root->traverse();

cout << endl;

}

// Search a key in the tree

BTreeNode\* search(int k) {

return root ? root->search(k) : nullptr;

}

// Insert a new key into the tree

void insert(int k) {

if (!root) {

root = new BTreeNode(t, true);

root->keys.push\_back(k);

} else {

if (root->keys.size() == 2 \* t - 1) {

BTreeNode\* s = new BTreeNode(t, false);

s->children.push\_back(root);

s->splitChild(0, root);

int i = 0;

if (k > s->keys[0])

i++;

s->children[i]->insertNonFull(k);

root = s;

} else {

root->insertNonFull(k);

}

}

}

};

int main() {

int t;

cout << "Enter the minimum degree of the B-Tree: ";

cin >> t;

BTree btree(t);

int choice, key;

while (true) {

cout << "\n1. Insert Key\n2. Search Key\n3. Traverse Tree\n4. Exit\n";

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1:

cout << "Enter key to insert: ";

cin >> key;

btree.insert(key);

break;

case 2:

cout << "Enter key to search: ";

cin >> key;

if (btree.search(key))

cout << "Key found!\n";

else

cout << "Key not found.\n";

break;

case 3:

cout << "Tree traversal: ";

btree.traverse();

break;

case 4:

return 0;

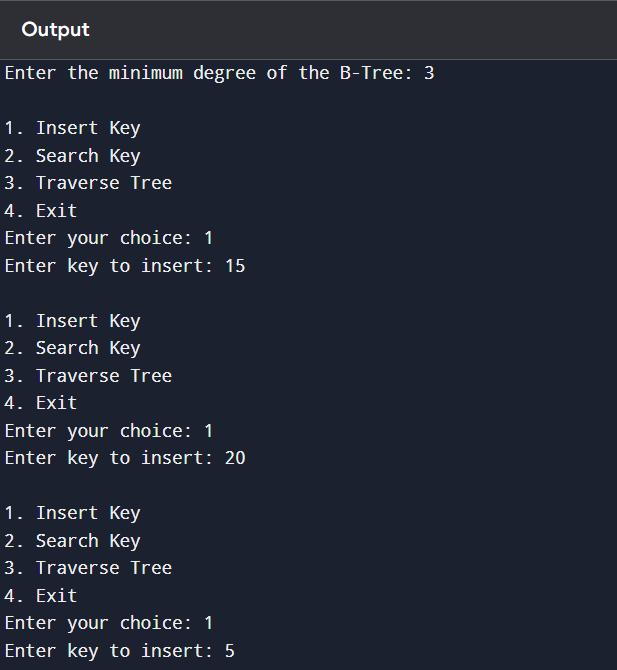
default:

cout << "Invalid choice. Please try again.\n";

}

}

}



A screenshot of a computer program

Description automatically generated

# 6. Write a program to implement the Tree Data structure, which supports the following operations:

a. Insert

b. Search

#include <iostream>

using namespace std;

// Define a Node structure

struct TreeNode {

int data; // Value of the node

TreeNode\* left; // Pointer to the left child

TreeNode\* right; // Pointer to the right child

TreeNode(int value) : data(value), left(nullptr), right(nullptr) {}

};

// Binary Search Tree class

class BinarySearchTree {

private:

TreeNode\* root;

// Helper function for insertion

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

if (!node)

return new TreeNode(value);

if (value < node->data)

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

else if (value > node->data)

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

return node;

}

// Helper function for searching

bool searchNode(TreeNode\* node, int value) {

if (!node)

return false;

if (node->data == value)

return true;

else if (value < node->data)

return searchNode(node->left, value);

else

return searchNode(node->right, value);

}

// Helper function for in-order traversal

void inOrderTraversal(TreeNode\* node) {

if (node) {

inOrderTraversal(node->left);

cout << node->data << " ";

inOrderTraversal(node->right);

}

}

public:

// Constructor

BinarySearchTree() : root(nullptr) {}

// Insert a value into the tree

void insert(int value) {

root = insertNode(root, value);

}

// Search for a value in the tree

bool search(int value) {

return searchNode(root, value);

}

// Display the tree in in-order traversal

void display() {

cout << "In-order Traversal: ";

inOrderTraversal(root);

cout << endl;

}

};

int main() {

BinarySearchTree bst;

int choice, value;

while (true) {

cout << "\n1. Insert\n2. Search\n3. Display\n4. Exit\n";

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1:

cout << "Enter the value to insert: ";

cin >> value;

bst.insert(value);

break;

case 2:

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

cin >> value;

if (bst.search(value))

cout << "Value found in the tree.\n";

else

cout << "Value not found in the tree.\n";

break;

case 3:

bst.display();

break;

case 4:

return 0;

default:

cout << "Invalid choice. Please try again.\n";

}

}

}

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

# 7. Write a program to search a pattern in a given text using the KMP algorithm.

#include <iostream>

#include <vector>

#include <string>

using namespace std;

// Function to create the Longest Prefix Suffix (LPS) array

void computeLPSArray(const string& pattern, vector<int>& lps) {

int length = 0; // Length of the previous longest prefix suffix

lps[0] = 0; // LPS for the first character is always 0

int i = 1;

while (i < pattern.length()) {

if (pattern[i] == pattern[length]) {

length++;

lps[i] = length;

i++;

} else {

if (length != 0) {

length = lps[length - 1];

} else {

lps[i] = 0;

i++;

}

}

}

}

// KMP Pattern Searching Function

void KMPSearch(const string& text, const string& pattern) {

int n = text.length();

int m = pattern.length();

// Create the LPS array

vector<int> lps(m);

computeLPSArray(pattern, lps);

int i = 0; // Index for text

int j = 0; // Index for pattern

bool found = false;

while (i < n) {

if (pattern[j] == text[i]) {

i++;

j++;

}

if (j == m) {

cout << "Pattern found at index " << i - j << endl;

found = true;

j = lps[j - 1];

} else if (i < n && pattern[j] != text[i]) {

if (j != 0) {

j = lps[j - 1];

} else {

i++;

}

}

}

if (!found) {

cout << "Pattern not found in the text." << endl;

}

}

int main() {

string text, pattern;

cout << "Enter the text: ";

getline(cin, text)

cout << "Enter the pattern: ";

getline(cin, pattern);

KMPSearch(text, pattern);

return 0;

}

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

# 8. Write a program to implement a Suffix tree.

#include <iostream>

#include <map>

#include <vector>

#include <string>

using namespace std;

// Node structure for the suffix tree

struct SuffixTreeNode {

map<char, SuffixTreeNode\*> children; // Edges to child nodes

int start;

int\* end;

int suffixIndex;

SuffixTreeNode(int start, int\* end) {

this->start = start;

this->end = end;

this->suffixIndex = -1;

}

};

// Suffix Tree class

class SuffixTree {

private:

string text;

SuffixTreeNode\* root;

int size;

// Recursive function to delete the tree

void freeTree(SuffixTreeNode\* node) {

for (auto& child : node->children) {

freeTree(child.second);

}

delete node;

}

// Helper function to print the suffix tree

void printTree(SuffixTreeNode\* node, int height, const string& text) {

if (!node)

return

for (auto& child : node->children) {

int start = child.second->start;

int end = \*(child.second->end);

cout << string(height, ' ') << text.substr(start, end - start + 1) << endl;

printTree(child.second, height + 2, text);

}

}

public:

// Constructor

SuffixTree(const string& txt) {

text = txt;

size = text.size();

root = buildSuffixTree();

}

// Destructor

~SuffixTree() {

freeTree(root);

}

SuffixTreeNode\* buildSuffixTree() {

SuffixTreeNode\* root = new SuffixTreeNode(-1, new int(-1));

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

SuffixTreeNode\* currentNode = root;

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

char currentChar = text[j];

if (currentNode->children.find(currentChar) == currentNode->children.end()) {

int\* end = new int(size - 1);

currentNode->children[currentChar] = new SuffixTreeNode(j, end);

}

currentNode = currentNode->children[currentChar];

}

currentNode->suffixIndex = i;

}

return root;

// Print the suffix tree

void printTree() {

cout << "Suffix Tree:" << endl;

printTree(root, 0, text);

}

}

int main() {

string inputText;

cout << "Enter the text: ";

cin >> inputText;

inputText += "$";

SuffixTree suffixTree(inputText);

suffixTree.printTree();

return 0;

}

A screenshot of a computer screen

Description automatically generated