दिल्ली विश्वविद्यालय

University of Delhi

Practical file

(Algorithms and advance data structur)

NAME - AVIKA SINGH

CLASS ROLL NO - 2K22/CS/15

UNIVERSITY ROLL NO - 22013570010

SUBJECT - *Algorithms and advance data structurs*

DATE OF SUBMITTION - 12/21/2024

1. Write a program to sort the elements of an array using Randomized Quick Sort (the program should report the number of comparisons).

Code:

```
#include<iostream>
#include <cstdlib>
#include <ctime>

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 pivotindex = low + rand() % (high - low + 1);
    swap(arr[pivotindex], arr[high]);  // Move pivot to end

    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 if arr[j] is smaller than
pivot
        }
    }
    swap(arr[i + 1], arr[high]);  // Move pivot to its correct position
    return i + 1;  // Return the pivot index
}

void r_quicksort(int arr[], int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);  // Partition index
        r_quicksort(arr, low, pi - 1);  // Recursively sort left part
        r_quicksort(arr, pi + 1, high);  // Recursively sort right part
    }
```

```
}

void printArray(int arr[], int size) {
    for (int i = 0; i < size; ++i) {
        cout << arr[i] << " ";
    }
    cout << endl;
}

int main() {
    srand(static_cast<unsigned>(time(0)));  // Initialize random seed

    int n;
    cout << "Enter the number of elements in the array: ";
    cin >> n;

    int* arr = new int[n];  // Dynamically allocate array based on user
input

    cout << "Enter the elements of the array:" << endl;
    for (int i = 0; i < n; ++i) {
        cin >> arr[i];
    }

    cout << "Original array: ";
    printArray(arr, n);

    r_quicksort(arr, 0, n - 1);  // Call quicksort

    cout << "Sorted array: ";
    printArray(arr, n);

    delete[] arr;  // Free dynamically allocated memory

    return 0;
}
```

OUTPUT :

```
Enter the number of elements in the array: 5
Enter the elements of the array:
11
4
89
67
-23
Original array: 11 4 89 67 -23
Sorted array: -23 4 11 67 89


...Program finished with exit code 0
Press ENTER to exit console.
```

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

Code:

```
#include<iostream>
#include <cstdlib>
#include <ctime>

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 pivotindex = low + rand() % (high - low + 1);
    swap(arr[pivotindex], 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;
}

int randomizedSelect(int arr[], int low, int high, int i) {
    if (low == high) {
        return arr[low];
    }
    int pivotIndex = partition(arr, low, high);
    int numElementsInLeft = pivotIndex - low + 1;
    if (i == numElementsInLeft) {
        return arr[pivotIndex];
    }
    else if (i < numElementsInLeft) {
        return randomizedSelect(arr, low, pivotIndex - 1, i);
    }
    else {
        return randomizedSelect(arr, pivotIndex + 1, high, i -
numElementsInLeft);
    }
}

void printArray(int arr[], int size) {
    for (int i = 0; i < size; ++i) {
        cout << arr[i] << " ";
    }
    cout << endl;
}

int main() {
    srand(static_cast<unsigned>(time(0)));
    int n, i;
    cout << "Enter the number of elements in the array: ";
    cin >> n;
    int* arr = new int[n];
    cout << "Enter the elements of the array:" << endl;
    for (int j = 0; j < n; ++j) {
        cin >> arr[j];
    }
    cout << "Enter the value of i (the position of the element to
find): ";
    cin >> i;
    if (i < 1 || i > n) {
```

```
        cout << "Invalid value of i. It must be between 1 and " << n <<
endl;
    } else {
        int result = randomizedSelect(arr, 0, n - 1, i);
        cout << "The " << i << "-th smallest element is: " << result <<
endl;
    }
    delete[] arr;
    return 0;
}
```

OUTPUT :

```
Enter the number of elements in the array: 8
Enter the elements of the array:
34
89
3
14
98
67
88
67
Enter the value of i (the position of the element to find): 4
The 4-th smallest element is: 67


...Program finished with exit code 0
Press ENTER to exit console.
```

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

Code :

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

struct Edge {
    int u, v, weight;
```

```
    bool operator<(const Edge& e) const {
        return weight < e.weight;
    }
};

class DisjointSet {
public:
    vector<int> parent, rank;

    DisjointSet(int n) {
        parent.resize(n);
        rank.resize(n, 0);
        for (int i = 0; i < n; ++i) {
            parent[i] = i;
        }
    }

    int find(int u) {
        if (parent[u] != u) {
            parent[u] = find(parent[u]);
        }
        return parent[u];
    }

    void unionSets(int u, int v) {
        int root_u = find(u);
        int root_v = find(v);
        if (root_u != root_v) {
            if (rank[root_u] > rank[root_v]) {
                parent[root_v] = root_u;
            } else if (rank[root_u] < rank[root_v]) {
                parent[root_u] = root_v;
            } else {
                parent[root_v] = root_u;
                rank[root_u]++;
            }
        }
    }
};

void kruskal(int n, vector<Edge>& edges) {
    sort(edges.begin(), edges.end());
    DisjointSet ds(n);
```

```
    vector<Edge> mst;
    int mstWeight = 0;

    for (Edge& e : edges) {
        int u = e.u, v = e.v;
        if (ds.find(u) != ds.find(v)) {
            mst.push_back(e);
            mstWeight += e.weight;
            ds.unionSets(u, v);
        }
    }

    cout << "Edges in the Minimum Spanning Tree (MST):\n";
    for (Edge& e : mst) {
        cout << e.u << " - " << e.v << " : " << e.weight << endl;
    }
    cout << "Total weight of the MST: " << mstWeight << endl;
}

int main() {
    int n, m;
    cout << "Enter the number of vertices: ";
    cin >> n;
    cout << "Enter the number of edges: ";
    cin >> m;

    vector<Edge> edges(m);

    cout << "Enter the edges in the format (u v weight):\n";
    for (int i = 0; i < m; ++i) {
        cin >> edges[i].u >> edges[i].v >> edges[i].weight;
    }

    kruskal(n, edges);

    return 0;
}
```

OUTPUT :

```
Enter the number of vertices: 4
Enter the number of edges: 5
Enter the edges in the format (u v weight):
0 1 10
0 2 6
0 3 5
1 3 15
2 3 4
Edges in the Minimum Spanning Tree (MST):
2 - 3 : 4
0 - 3 : 5
0 - 1 : 10
Total weight of the MST: 19


...Program finished with exit code 0
Press ENTER to exit console.
```

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.

Code :

```
#include <iostream>
#include <vector>
#include <climits>
using namespace std;

struct Edge {
    int u, v, weight;
};

class Graph {
public:
    int V, E;
    vector<Edge> edges;

    Graph(int V, int E) {
        this->V = V;
        this->E = E;
    }

    void addEdge(int u, int v, int weight) {
```

```cpp
        edges.push_back({u, v, weight});
    }

    void BellmanFord(int src);
};

void Graph::BellmanFord(int src) {
    vector<int> dist(V, INT_MAX);
    dist[src] = 0;

    // Relax all edges |V| - 1 times
    for (int i = 1; i <= V - 1; ++i) {
        for (const auto& edge : edges) {
            if (dist[edge.u] != INT_MAX && dist[edge.u] + edge.weight <
dist[edge.v]) {
                dist[edge.v] = dist[edge.u] + edge.weight;
            }
        }
    }

    // Check for negative weight cycles
    for (const auto& edge : edges) {
        if (dist[edge.u] != INT_MAX && dist[edge.u] + edge.weight <
dist[edge.v]) {
            cout << "Graph contains negative weight cycle\n";
            return;
        }
    }

    // Print the shortest distances
    cout << "Vertex \t Distance from Source " << src << endl;
    for (int i = 0; i < V; ++i) {
        if (dist[i] == INT_MAX) {
            cout << i << " \t\t INF" << endl;
        } else {
            cout << i << " \t\t " << dist[i] << endl;
        }
    }
}

int main() {
    int V, E, u, v, weight, src;
```

```cpp
    cout << "Enter the number of vertices: ";
    cin >> V;

    cout << "Enter the number of edges: ";
    cin >> E;

    Graph g(V, E);

    cout << "Enter the edges (u v weight):\n";
    for (int i = 0; i < E; ++i) {
        cin >> u >> v >> weight;
        g.addEdge(u, v, weight);
    }

    cout << "Enter the source vertex: ";
    cin >> src;

    g.BellmanFord(src);

    return 0;
}
```

OUTPUT :

```
Enter the number of vertices: 5
Enter the number of edges: 8
Enter the edges (u v weight):
0 1 -1
0 2 4
1 2 3
1 3 2
1 4 2
2 3 5
3 4 -3


3 4 4
Enter the source vertex: 0
Vertex   Distance from Source 0
0               0
1               -1
2               2
3               1
4               -2


...Program finished with exit code 0
Press ENTER to exit console.
```

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

Code :

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

struct BTreeNode {
    vector<int> keys;
    vector<BTreeNode*> children;
    bool isLeaf;
    int t;

    BTreeNode(int t, bool isLeaf);
    void insertNonFull(int key);
    void splitChild(int i, BTreeNode* y);
};

class BTree {
public:
    BTreeNode* root;
    int t;

    BTree(int t);
    void insert(int key);
    void traverse(BTreeNode* node);
    BTreeNode* search(BTreeNode* node, int key);
};

BTreeNode::BTreeNode(int t, bool isLeaf) {
    this->t = t;
    this->isLeaf = isLeaf;
}

BTree::BTree(int t) {
    this->t = t;
    root = new BTreeNode(t, true);
}

void BTreeNode::insertNonFull(int key) {
    int i = keys.size() - 1;
```

```cpp
    if (isLeaf) {
        while (i >= 0 && keys[i] > key) {
            i--;
        }
        keys.insert(keys.begin() + i + 1, key);
    } else {
        while (i >= 0 && keys[i] > key) {
            i--;
        }
        i++;
        if (children[i]->keys.size() == 2 * t - 1) {
            splitChild(i, children[i]);
            if (keys[i] < key) {
                i++;
            }
        }
        children[i]->insertNonFull(key);
    }
}

void BTreeNode::splitChild(int i, BTreeNode* y) {
    BTreeNode* z = new BTreeNode(t, y->isLeaf);

    for (int j = 0; j < t - 1; j++) {
        z->keys.push_back(y->keys[j + t]);
    }

    if (!y->isLeaf) {
        for (int j = 0; j < t; j++) {
            z->children.push_back(y->children[j + t]);
        }
    }

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

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

void BTree::insert(int key) {
    if (root->keys.size() == 2 * t - 1) {
```

```cpp
        BTreeNode* newRoot = new BTreeNode(t, false);
        newRoot->children.push_back(root);
        newRoot->splitChild(0, root);
        root = newRoot;
    }
    root->insertNonFull(key);
}

BTreeNode* BTree::search(BTreeNode* node, int key) {
    int i = 0;
    while (i < node->keys.size() && key > node->keys[i]) {
        i++;
    }

    if (i < node->keys.size() && key == node->keys[i]) {
        return node;
    }

    if (node->isLeaf) {
        return nullptr;
    }

    return search(node->children[i], key);
}

void BTree::traverse(BTreeNode* node) {
    for (int i = 0; i < node->keys.size(); i++) {
        if (!node->isLeaf) {
            traverse(node->children[i]);
        }
        cout << node->keys[i] << " ";
    }
    if (!node->isLeaf) {
        traverse(node->children[node->keys.size()]);
    }
}

int main() {
    int t;
    cout << "Enter the minimum degree (t) of the B-Tree: ";
    cin >> t;

    BTree tree(t);
```

```cpp
    int n, key;
    cout << "Enter the number of keys to insert: ";
    cin >> n;
    cout << "Enter the keys to insert: ";
    for (int i = 0; i < n; i++) {
        cin >> key;
        tree.insert(key);
    }

    cout << "B-Tree Traversal (In-order): ";
    tree.traverse(tree.root);
    cout << endl;

    cout << "Enter a key to search: ";
    cin >> key;

    BTreeNode* result = tree.search(tree.root, key);
    if (result != nullptr) {
        cout << "Key " << key << " found in the B-Tree.\n";
    } else {
        cout << "Key " << key << " not found in the B-Tree.\n";
    }

    return 0;
}
```

OUTPUT :

```
Enter the minimum degree (t) of the B-Tree: 3
Enter the number of keys to insert: 10
Enter the keys to insert: 10 20 5 6 30 40 50 35 15 25
B-Tree Traversal (In-order): 5 6 10 15 20 25 30 35 40 50
Enter a key to search: 25
Key 25 found in the B-Tree.


...Program finished with exit code 0
Press ENTER to exit console.
```

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

Code :

```
#include <iostream>
using namespace std;

struct Node {
    int data;
    Node* left;
    Node* right;

    Node(int value) {
        data = value;
        left = nullptr;
        right = nullptr;
    }
};

class BinarySearchTree {
private:
    Node* root;

    Node* insert(Node* node, int key) {
        if (node == nullptr) {
            return new Node(key);
        }

        if (key < node->data) {
            node->left = insert(node->left, key);
        } else if (key > node->data) {
            node->right = insert(node->right, key);
        }

        return node;
    }

    Node* search(Node* node, int key) {
        if (node == nullptr || node->data == key) {
            return node;
```

```
        }

        if (key < node->data) {
            return search(node->left, key);
        }

        return search(node->right, key);
    }

    void inorderTraversal(Node* node) {
        if (node == nullptr) {
            return;
        }

        inorderTraversal(node->left);
        cout << node->data << " ";
        inorderTraversal(node->right);
    }

public:
    BinarySearchTree() {
        root = nullptr;
    }

    void insert(int key) {
        root = insert(root, key);
    }

    Node* search(int key) {
        return search(root, key);
    }

    void inorderTraversal() {
        inorderTraversal(root);
        cout << endl;
    }
};

int main() {
    BinarySearchTree tree;

    int n, key;
    cout << "Enter the number of elements to insert into the tree: ";
```

```
    cin >> n;

    cout << "Enter the keys to insert: ";
    for (int i = 0; i < n; i++) {
        cin >> key;
        tree.insert(key);
    }

    cout << "In-order traversal of the tree: ";
    tree.inorderTraversal();

    cout << "Enter a key to search in the tree: ";
    cin >> key;

    Node* result = tree.search(key);
    if (result != nullptr) {
        cout << "Key " << key << " found in the tree." << endl;
    } else {
        cout << "Key " << key << " not found in the tree." << endl;
    }

    return 0;
}
```

OUTPUT :

```
Enter the number of elements to insert into the tree: 5
Enter the keys to insert: 50 30 20 40 70
In-order traversal of the tree: 20 30 40 50 70
Enter a key to search in the tree: 40
Key 40 found in the tree.


...Program finished with exit code 0
Press ENTER to exit console.
```

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

CODE :

```
#include <iostream>
#include <vector>
using namespace std;

void buildLPSArray(const string& pattern, vector<int>& lps) {
    int length = 0;
    int i = 1;
    lps[0] = 0;

    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++;
            }
        }
    }
}

void KMPSearch(const string& text, const string& pattern) {
    int n = text.length();
    int m = pattern.length();

    vector<int> lps(m, 0);
    buildLPSArray(pattern, lps);

    int i = 0;
    int j = 0;

    while (i < n) {
        if (pattern[j] == text[i]) {
```

```
            i++;
            j++;
        }

        if (j == m) {
            cout << "Pattern found at index " << i - j << endl;
            j = lps[j - 1];
        } else if (i < n && pattern[j] != text[i]) {
            if (j != 0) {
                j = lps[j - 1];
            } else {
                i++;
            }
        }
    }
}

int main() {
    string text, pattern;

    cout << "Enter the text: ";
    getline(cin, text);

    cout << "Enter the pattern to search: ";
    getline(cin, pattern);

    KMPSearch(text, pattern);

    return 0;
}
```

Output :

```
Enter the text: AABAACAADAABAABA
Enter the pattern to search: AABA
Pattern found at index 0
Pattern found at index 9
Pattern found at index 12


...Program finished with exit code 0
Press ENTER to exit console.
```

8. Write a program to implement a Suffix tree (bruteForceSuffixTrie method given in Ref. 4 at page 712).

Code :

```
#include <iostream>
#include <vector>
#include <string>
#include <unordered_map>
using namespace std;

class SuffixTreeNode {
public:
    unordered_map<char, SuffixTreeNode*> children;
    int start, end;
    SuffixTreeNode* suffixLink;

    SuffixTreeNode(int start, int end) : start(start), end(end),
suffixLink(nullptr) {}
};

class SuffixTree {
public:
    string text;
    SuffixTreeNode* root;
    SuffixTreeNode* activeNode;
    int activeEdge, activeLength, remainder;

    SuffixTree(string T) : text(T), root(new SuffixTreeNode(-1, -1)),
activeNode(root), activeEdge(-1), activeLength(0), remainder(0) {
        buildSuffixTree();
    }

    void buildSuffixTree() {
        int n = text.length();
        for (int i = 0; i < n; i++) {
            extendSuffixTree(i, n);
        }
    }

    void extendSuffixTree(int i, int n) {
        remainder++;
        while (remainder > 0) {
```

```
            if (activeLength == 0) {
                activeEdge = i;
            }

            if (activeNode->children.find(text[activeEdge]) ==
activeNode->children.end()) {
                activeNode->children[text[activeEdge]] = new
SuffixTreeNode(i, n - 1);
                remainder--;
            } else {
                SuffixTreeNode* nextNode =
activeNode->children[text[activeEdge]];
                if (activeLength >= nextNode->end - nextNode->start +
1) {
                    activeEdge += nextNode->end - nextNode->start + 1;
                    activeLength -= nextNode->end - nextNode->start +
1;
                    activeNode = nextNode;
                    continue;
                }

                if (text[nextNode->start + activeLength] == text[i]) {
                    activeLength++;
                    remainder--;
                    break;
                }

                int splitStart = nextNode->start;
                int splitEnd = nextNode->start + activeLength - 1;

                SuffixTreeNode* splitNode = new
SuffixTreeNode(splitStart, splitEnd);
                activeNode->children[text[activeEdge]] = splitNode;
                splitNode->children[text[i]] = new SuffixTreeNode(i, n
- 1);
                nextNode->start += activeLength;
                splitNode->children[text[nextNode->start]] = nextNode;

                remainder--;

                if (activeNode == root) {
                    activeLength--;
                    activeEdge++;
```

```
            } else {
                activeNode = activeNode->suffixLink;
            }
        }
    }
}

void printTree(SuffixTreeNode* node, const string& str) {
    if (node == nullptr) return;

    if (node->start != -1) {
        cout << str.substr(node->start, node->end - node->start +
1) << endl;
    }

    for (auto& child : node->children) {
        printTree(child.second, str);
    }
}

void printSuffixTree() {
    printTree(root, text);
}
};

int main() {
    string text;
    cout << "Enter the text: ";
    cin >> text;

    SuffixTree suffixTree(text);
    cout << "Suffix Tree:" << endl;
    suffixTree.printSuffixTree();

    return 0;
}
```

OUTPUT :

```
Enter the text: banana
Suffix Tree:
nana
anana
banana


...Program finished with exit code 0
Press ENTER to exit console.
```