**ASSIGNMENT 08**

Given sequence k = k1 <k2 <... <kn of n sorted keys, with a search probability pi for each key ki Build the Binary search tree that has the least search cost given the access probability for each key?

#include <iostream>

#include <vector>

#include<climits>

using namespace std;

// Forward declaration of Node structure

struct Node;

// Function to build the optimal binary search tree

Node\* buildOptimalBST(vector<int>& keys, vector<double>& probabilities);

// Function to recursively build the optimal binary search tree

Node\* buildSubtree(vector<int>& keys, vector<vector<int>>& root, int i, int j);

// Function to search for a key in the binary search tree

Node\* search(Node\* root, int key);

// Function to delete the binary search tree

void deleteBinarySearchTree(Node\* root);

// Binary search tree node structure

struct Node {

int key;

Node\* left;

Node\* right;

Node(int k) : key(k), left(nullptr), right(nullptr) {}

};

// Function to calculate the sum of probabilities from index i to j

double sum(vector<double>& probabilities, int i, int j) {

double s = 0.0;

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

s += probabilities[k];

}

return s;

}

Node\* buildSubtree(vector<int>& keys, vector<vector<int>>& root, int i, int j) {

if (i > j) {

return nullptr;

}

int r = root[i][j];

Node\* rootNode = new Node(keys[r]);

rootNode->left = buildSubtree(keys, root, i, r - 1);

rootNode->right = buildSubtree(keys, root, r + 1, j);

return rootNode;

}

Node\* buildOptimalBST(vector<int>& keys, vector<double>& probabilities) {

int n = keys.size();

vector<vector<double>> cost(n + 1, vector<double>(n + 1, 0.0));

vector<vector<int>> root(n + 1, vector<int>(n + 1, 0));

// Initialize base cases

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

cost[i][i] = probabilities[i - 1];

root[i][i] = i;

}

// Build the cost matrix

for (int len = 2; len <= n; len++) {

for (int i = 1; i <= n - len + 1; i++) {

int j = i + len - 1;

cost[i][j] = INT\_MAX;

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

double c = ((r > i) ? cost[i][r - 1] : 0) + ((r < j) ? cost[r + 1][j] : 0) + sum(probabilities, i, j);

if (c < cost[i][j]) {

cost[i][j] = c;

root[i][j] = r;

}

}

}

}

// Build the optimal binary search tree

return buildSubtree(keys, root, 1, n);

}

Node\* search(Node\* root, int key) {

if (root == nullptr || root->key == key) {

return root;

}

if (key < root->key) {

return search(root->left, key);

}

return search(root->right, key);

}

void deleteBinarySearchTree(Node\* root) {

if (root == nullptr) {

return;

}

// Delete the left and right subtrees

deleteBinarySearchTree(root->left);

deleteBinarySearchTree(root->right);

// Delete the current node

delete root;

}

int main() {

vector<int> keys = {1, 2, 3, 4, 5};

vector<double> probabilities = {0.1, 0.2, 0.15, 0.3, 0.25};// Build the optimal binary search tree

Node\* root = buildOptimalBST(keys, probabilities);

// Perform any desired operations on the constructed binary search tree

// For example, you can perform a search operation on the tree

int keyToSearch = 3;

Node\* result = search(root, keyToSearch);

if (result != nullptr) {

cout << "Key " << keyToSearch << " found in the binary search tree." << endl;

} else {

cout << "Key " << keyToSearch << " not found in the binary search tree." << endl;

}

// Cleanup: Free the memory allocated for the binary search tree

deleteBinarySearchTree(root);

return 0;

}

OUTPUT:

