**Practical No : 6**

**Name :** Samrudhi Pawar

**Roll No :** 44

**Title :**Develop a program to implement Optimal Binary Search Tree using

Dynamic Programming.

**Program :**

#include <iostream>

#include <vector>

#include <iomanip>

#include <limits>

using namespace std;

void printMatrix(const vector<vector<double>>& matrix) {

for (const auto& row : matrix) {

for (double value : row) {

cout << setw(10) << setprecision(2) << fixed << value << " ";

}

cout << endl;

}

}

void optimalBST(const vector<string>& keys, const vector<double>& probabilities) {

int n = keys.size();

vector<vector<double>> cost(n, vector<double>(n, 0));

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

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

cost[i][i] = probabilities[i];

root[i][i] = i;

}

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

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

int j = i + chainLen - 1;

cost[i][j] = numeric\_limits<double>::infinity();

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

double leftCost = (r > i) ? cost[i][r - 1] : 0;

double rightCost = (r < j) ? cost[r + 1][j] : 0;

double totalCost = leftCost + rightCost + 0.0;

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

totalCost += probabilities[k];

}

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

cost[i][j] = totalCost;

root[i][j] = r;

}

}

}

}

cout << "\nCost Matrix:\n";

printMatrix(cost);

cout << "\nRoot Matrix:\n";

for (const auto& row : root) {

for (int value : row) {

cout << setw(5) << value << " ";

}

cout << endl;

}

}

int main() {

int n;

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

cin >> n;

vector<string> keys(n);

vector<double> probabilities(n);

cout << "Enter keys (one per line):\n";

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

cin >> keys[i];

}

cout << "Enter probabilities (one per line):\n";

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

cin >> probabilities[i];

}

optimalBST(keys, probabilities);

return 0;

}

**Output :**

Enter the number of keys: 4

Enter keys (one per line):

5

8

7

4

Enter probabilities (one per line):

1

2

3

4

Cost Matrix:

1.00 4.00 10.00 18.00

0.00 2.00 7.00 15.00

0.00 0.00 3.00 10.00

0.00 0.00 0.00 4.00

Root Matrix:

0 1 1 2

0 1 2 2

0 0 2 3

0 0 0 3