22.Optimal Binary Search Tree

#include <stdio.h>

#include <limits.h>

#define MAX\_N 100

// Function to construct the cost and root tables for the optimal BST

void constructOptimalBST(int keys[], int freq[], int n, int cost[][MAX\_N], int root[][MAX\_N]) {

// Initialize diagonal elements of cost table to 0

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

cost[i][i] = freq[i];

root[i][i] = i;

}

// Fill the cost and root tables using dynamic programming

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

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

int j = i + l - 1;

int min\_cost = INT\_MAX;

int best\_root = -1;

// Consider each key as the root and find the minimum cost

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

int left\_cost = (k > i) ? cost[i][k - 1] : 0;

int right\_cost = (k < j) ? cost[k + 1][j] : 0;

int potential\_cost = left\_cost + right\_cost + freq[k];

if (potential\_cost < min\_cost) {

min\_cost = potential\_cost;

best\_root = k;

}

}

cost[i][j] = min\_cost;

root[i][j] = best\_root;

}

}

}

// Function to print the optimal BST in pre-order traversal

void printOptimalBST(int keys[], int root[][MAX\_N], int i, int j) {

if (i > j) {

return;

}

printf("%d ", keys[root[i][j]]);

printOptimalBST(keys, root, i, root[i][j] - 1);

printOptimalBST(keys, root, root[i][j] + 1, j);

}

int main() {

int keys[] = {10, 20, 30};

int freq[] = {4, 2, 6};

int n = sizeof(keys) / sizeof(keys[0]);

int cost[MAX\_N][MAX\_N], root[MAX\_N][MAX\_N];

constructOptimalBST(keys, freq, n, cost, root);

printf("Optimal Binary Search Tree: ");

printOptimalBST(keys, root, 0, n - 1);

printf("\n");

return 0;

} 