**1)**

#include <bits/stdc++.h>

using namespace std;

int fibonacci(int n)

{ if (n <= 1)

{ return n;

}

return fibonacci(n - 1) + fibonacci(n - 2);

}

int main()

{

int i, n, t1 = 0, t2 = 1, nT;

cout<< "Fibonacci Series in C++ Without Using Recursion ";

cout<< "\nEnter any number:";

cin >> n;

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

{

cout << t1 << " "; nT = t1 + t2; t1 = t2; t2 = nT;

}

cout<< "\n\nFibonacci Series in C++ Using Recursion";

cout<< "\nEnter any number:"; int p;

cin>>p;

cout << fibonacci(p);

}

**/\* OUTPUT:-**

Fibonacci Series in C++ Without Using Recursion

Enter any number:15

0 1 1 2 3 5 8 13 21 34 55 89 144 233 377

Fibonacci Series in C++ Using Recursion

Enter any number:15

377

**2)**

#include <iostream>

#include <vector>

#include <algorithm>

struct Item {

int weight;

int value;

double ratio; // value per unit weight

};

bool compareItems(Item item1, Item item2) {

return item1.ratio > item2.ratio;

}

double fractionalKnapsack(std::vector<Item>& items, int capacity) {

std::sort(items.begin(), items.end(), compareItems);

double totalValue = 0.0;

int currentWeight = 0;

for (const auto& item : items) {

if (currentWeight + item.weight <= capacity) {

totalValue += item.value;

currentWeight += item.weight;

} else {

int remainingCapacity = capacity - currentWeight;

totalValue += item.ratio \* remainingCapacity;

break;

}

}

return totalValue;

}

int main() {

int n, capacity;

std::cout << "Enter the number of items: ";

std::cin >> n;

std::vector<Item> items(n);

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

std::cout << "Enter weight and value for item " << i + 1 << ": ";

std::cin >> items[i].weight >> items[i].value;

items[i].ratio = static\_cast<double>(items[i].value) / items[i].weight;

}

std::cout << "Enter the maximum capacity of knapsack: ";

std::cin >> capacity;

double maxValue = fractionalKnapsack(items, capacity);

std::cout << "Maximum value in knapsack = " << maxValue << std::endl;

return 0;

}

**Output :**

**/tmp/PgPvSXPBz2.o**

**Enter the number of items: 1**

**Enter weight and value for item 1: 50**

**9**

**Enter the maximum capacity of knapsack: 100Maximum value in knapsack = 9**

**3)**

#include <iostream>

#include <queue>

#include <unordered\_map>

struct HuffmanNode {

char data;

int frequency;

HuffmanNode\* left;

HuffmanNode\* right;

HuffmanNode(char data, int frequency) : data(data), frequency(frequency), left(nullptr), right(nullptr) {}

};

struct CompareNodes {

bool operator()(HuffmanNode\* left, HuffmanNode\* right) {

return left->frequency > right->frequency;

}

};

void generateCodes(HuffmanNode\* root, std::string code, std::unordered\_map<char, std::string>& codes) {

if (root) {

if (!root->left && !root->right) {

codes[root->data] = code;

}

generateCodes(root->left, code + "0", codes);

generateCodes(root->right, code + "1", codes);

}

}

std::unordered\_map<char, std::string> buildHuffmanTree(std::unordered\_map<char, int>& frequencyMap) {

std::priority\_queue<HuffmanNode\*, std::vector<HuffmanNode\*>, CompareNodes> minHeap;

for (auto& pair : frequencyMap) {

minHeap.push(new HuffmanNode(pair.first, pair.second));

}

while (minHeap.size() != 1) {

HuffmanNode\* left = minHeap.top();

minHeap.pop();

HuffmanNode\* right = minHeap.top();

minHeap.pop();

HuffmanNode\* internalNode = new HuffmanNode('\0', left->frequency + right->frequency);

internalNode->left = left;

internalNode->right = right;

minHeap.push(internalNode);

} HuffmanNode\* root = minHeap.top();

std::unordered\_map<char, std::string> codes;

generateCodes(root, "", codes);

return codes;

}int main() {

std::string input;

std::cout << "Enter a string to encode: ";

std::cin >> input;

std::unordered\_map<char, int> frequencyMap;

for (char c : input) {

frequencyMap[c]++;

}

std::unordered\_map<char, std::string> huffmanCodes = buildHuffmanTree(frequencyMap);

std::cout << "Huffman Codes:\n";

for (auto& pair : huffmanCodes) {

std::cout << pair.first << ": " << pair.second << std::endl;

}

return 0;

}

**Output:**

**/tmp/PgPvSXPBz2.o**

**Enter a string to encode: 970446**

**Huffman Codes:**

**4: 11**

**0: 101**

**6: 100**

**7: 01**

**9: 00**

**00**

**dash: 2: 00: not found**

**9**

**dash: 3: 9: not found**

**4)**

#include <iostream>

#include <vector>

int knapsack(int capacity, std::vector<int>& weights, std::vector<int>& values) {

int n = weights.size();

std::vector<std::vector<int>> dp(n + 1, std::vector<int>(capacity + 1, 0));

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

for (int w = 1; w <= capacity; ++w) {

if (weights[i - 1] <= w) {

dp[i][w] = std::max(dp[i - 1][w], values[i - 1] + dp[i - 1][w - weights[i - 1]]);

} else {

dp[i][w] = dp[i - 1][w];

}

}

}

return dp[n][capacity];

}

int main() {

int n, capacity;

std::cout << "Enter the number of items: ";

std::cin >> n;

std::vector<int> weights(n);

std::vector<int> values(n);

std::cout << "Enter the weights of items: ";

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

std::cin >> weights[i];

}

std::cout << "Enter the values of items: ";

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

std::cin >> values[i];

}

std::cout << "Enter the maximum capacity of knapsack: ";

std::cin >> capacity;

int maxValue = knapsack(capacity, weights, values);

std::cout << "Maximum value in knapsack = " << maxValue << std::endl;

return 0;

}

**Output:**

**Enter the number of items: 2**

**Enter the weights of items: 45**

**12**

**Enter the values of items: 1**

**2**

**Enter the maximum capacity of knapsack: 20**

**Maximum value in knapsack = 2**

**5)**

#include <iostream>

#include <vector>

bool isSafe(std::vector<std::vector<int>>& board, int row, int col, int N) {

// Check if there is a queen in the same column

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

if (board[i][col] == 1) {

return false;

}

}

for (int i = row, j = col; i >= 0 && j >= 0; --i, --j) {

if (board[i][j] == 1) {

return false;

}

}

for (int i = row, j = col; i >= 0 && j < N; --i, ++j) {

if (board[i][j] == 1) {

return false;

}

}

return true;

}

bool solveNQueens(std::vector<std::vector<int>>& board, int row, int N) {

if (row == N) {

return true;

} for (int col = 0; col < N; ++col) {

if (isSafe(board, row, col, N)) {

board[row][col] = 1; // Place queen

if (solveNQueens(board, row + 1, N)) {

return true;

}

board[row][col] = 0;

}

}

return false;

}

int main() {

int N;

std::cout << "Enter the size of the chessboard (N): ";

std::cin >> N;

std::vector<std::vector<int>> board(N, std::vector<int>(N, 0));

board[0][0] = 1;

if (solveNQueens(board, 1, N)) {

std::cout << "N-Queens solution:\n";

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

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

std::cout << board[i][j] << " ";

}

std::cout << std::endl;

}

} else {

std::cout << "No solution exists for the given N.\n";

}

return 0;

}

**Output :**

**/tmp/PgPvSXPBz2.o**

**Enter the size of the chessboard (N): 8**

**N-Queens solution:**

**1 0 0 0 0 0 0 0**

**0 0 0 0 1 0 0 0**

**0 0 0 0 0 0 0 1**

**0 0 0 0 0 1 0 0**

**0 0 1 0 0 0 0 0**

**0 0 0 0 0 0 1 0**

**0 1 0 0 0 0 0 0**

**0 0 0 1 0 0 0 0**