Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

 Лабораторная работа 10

По дисциплине «Основы алгоритмизации и программирования»

На тему «Рекурсивные алгоритмы»

 Выполнила:

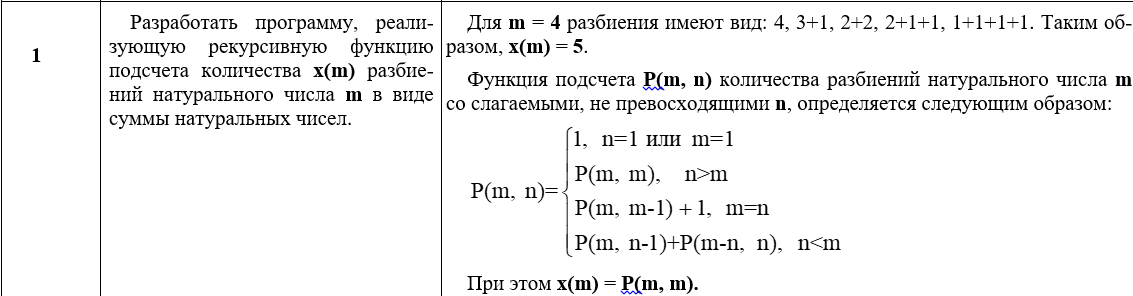
Студентка1 курса 6 группы

Альшевская Алина Михайловна

Преподаватель: асс. Андронова М.В.

2023, Минск

Вариант 1



#include <iostream>

#include <vector>

// Функция для подсчета количества разбиений

int P(int m, int n, std::vector<std::vector<int>>& dp) {

// Если m равно 0, то есть одно разбиение (пустое)

if (m == 0) {

return 1;

}

// Если m меньше 0 или если нет слагаемых, то нет разбиений

if (m < 0 || n == 0) {

return 0;

}

// Если значение уже было вычислено, то возвращаем его

if (dp[m][n] != -1) {

return dp[m][n];

}

// Разбиение числа m на слагаемые, не превосходящие n, равно сумме:

// 1) количества разбиений m на слагаемые, не превосходящие n-1,

// 2) количества разбиений m-n на слагаемые, не превосходящие n.

dp[m][n] = P(m, n - 1, dp) + P(m - n, n, dp);

return dp[m][n];

}

int main() {

setlocale(LC\_ALL, "Rissian");

int m;

std::cout << "input number m: ";

std::cin >> m;

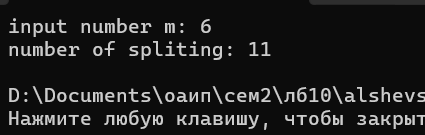
// Инициализируем таблицу для сохранения промежуточных результатов

std::vector<std::vector<int>> dp(m + 1, std::vector<int>(m + 1, -1));

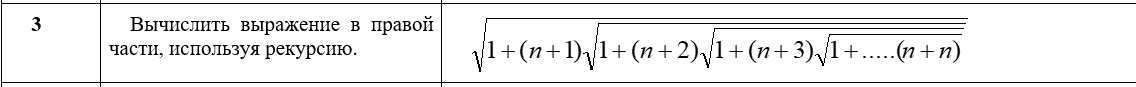
std::cout << "number of spliting: " << P(m, m, dp) << std::endl;

return 0;

}



Вариант 3



#include <cmath>

#include <iostream>

// Рекурсивная функция для вычисления выражения

double calculate\_expression(int n, int i = 1) {

// Базовый случай: если i равно n, возвращаем sqrt(1 + n\*n)

if (i == n) {

return std::sqrt(1 + n \* n);

}

// Рекурсивный случай: возвращаем sqrt(1 + i \* рекурсивный вызов с i+1)

else {

return std::sqrt(1 + i \* calculate\_expression(n, i + 1));

}

}

int main() {

int n;

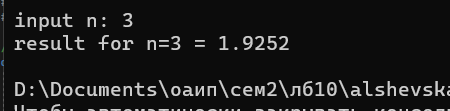
std::cout << "input n: ";

std::cin >> n;

std::cout << "result for n=" << n << " = " << calculate\_expression(n) << std::endl;

return 0;

}



Дополнительные задания

Задание1

Ввести цифру **А**, записать в файл все возможные числа, состоящие из цифр, не превышающих или равных **A**. Количество цифр в числах должно быть равно **А**.

Примечание: использовать дополнительный массив.

#include <fstream>

#include <vector>

#include <iostream>

using namespace std;

void generateNumbers(int A, vector<int>& currentNumber, int currentLength, ofstream& file) {

if (currentLength == A) {

for (int digit : currentNumber) {

file << digit;

}

file << endl;

return;

}

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

currentNumber.push\_back(i);

generateNumbers(A, currentNumber, currentLength + 1, file);

currentNumber.pop\_back(); // Remove the last digit for the next call

}

}

int main() {

int A;

cout << "Enter the number A: ";

cin >> A;

ofstream file("numbers.txt");

if (!file) {

cerr << "Failed to open the file for writing." << endl;

return 1;

}

vector<int> currentNumber;

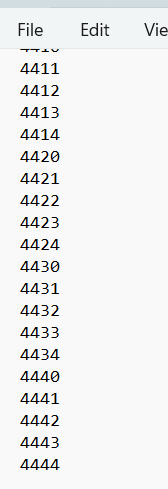
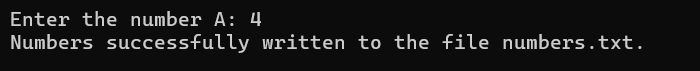
generateNumbers(A, currentNumber, 0, file);

file.close();

cout << "Numbers successfully written to the file numbers.txt." << endl;

return 0;

}



Задание 2

Задача проведения границы на карте («создание военных блоков»). Страны на карте заданы матрицей смежности. Если страны **i**, **j** имеют на карте общую границу, то элемент матрицы **A[i, j]** равен 1, иначе 0.

Необходимо разбить страны на две группы так, чтобы количество пар смежных стран из противоположных групп было минимальным.

#include <vector>

#include <algorithm>

#include <iostream>

using namespace std;

// Function to check if the remaining countries can be divided into two groups

bool canDivide(vector<vector<int>>& adjMatrix, int groupSize, int currentGroup, int currentGroupSize) {

if (currentGroupSize == groupSize) {

// If the current group has reached the required size, check if the remaining countries can be divided

for (int i = 0; i < adjMatrix.size(); i++) {

if (i != currentGroup && any\_of(adjMatrix[i].begin(), adjMatrix[i].end(), [&](int j) { return adjMatrix[currentGroup][j] == 1; })) {

return false; // Found a pair of adjacent countries from opposite groups

}

}

return true; // All countries have been successfully divided

}

for (int i = 0; i < adjMatrix.size(); i++) {

if (i != currentGroup && adjMatrix[currentGroup][i] == 1) {

// Check if country i can be added to the current group

if (canDivide(adjMatrix, groupSize, i, currentGroupSize + 1)) {

return true;

}

}

}

return false;

}

int main() {

int n; // Number of countries

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

cin >> n;

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

// Fill the adjacency matrix adjMatrix here

// Check if the countries can be divided into two groups

bool canBeDivided = canDivide(adjMatrix, n / 2, 0, 1);

if (canBeDivided) {

cout << "The countries can be divided into two groups with the minimum number of adjacent country pairs." << endl;

}

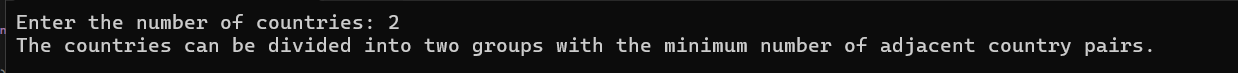
else {

cout << "The countries cannot be divided into two groups with the minimum number of adjacent country pairs." << endl;

}

return 0;

}



Задание 3

3. Дано **n** различных натуральных чисел (**n = 5**). Напечатать все перестановки этих чисел.

#include <iostream>

#include <vector>

using namespace std;

// Function to generate permutations

void generatePermutations(int n, int index, vector<int>& currentPermutation, vector<vector<int>>& allPermutations) {

if (index == n) {

// If the end of the permutation is reached, add the current permutation to the list of all permutations

allPermutations.push\_back(currentPermutation);

return;

}

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

// Check if number i has already been used in the current permutation

if (find(currentPermutation.begin(), currentPermutation.end(), i) == currentPermutation.end()) {

currentPermutation[index] = i; // Add number i to the current position

generatePermutations(n, index + 1, currentPermutation, allPermutations); // Generate permutations for the remaining numbers

}

}

}

int main() {

int n = 5; // Number of numbers

vector<vector<int>> allPermutations;

vector<int> currentPermutation(n);

generatePermutations(n, 0, currentPermutation, allPermutations);

// Print all permutations

for (const auto& permutation : allPermutations) {

for (int num : permutation) {

cout << num << " ";

}

cout << endl;

}

return 0;

}

