МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ государственное БЮДЖЕТНОЕ

образовательное учреждение

высшего образования

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кафедра защиты информации



**ОТЧЁТ**

**по лабораторной работе №7**

**«Рекурсивные функции»**

**по дисциплине: «***Программирование***»**

|  |  |
| --- | --- |
| Выполнил: | Проверил: |
| Студент гр. «АБс-322», «АВТФ» | Ассистент кафедры ЗИ |
| *Налитов Андрей Александрович* | Исаев Г. А. |
| «27» мая 2024 г. | «\_\_\_» \_\_\_\_\_\_ 2024 г. |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) |

Новосибирск 2024

# **Цели и задачи работы**

Изучение рекурсивного программирования, методов разработки эффективных алгоритмов.

# **Задания**

Написать рекурсивную программу решения поставленной задачи.

## **Задание 1**

На веревке висят прямоугольные скатерти и салфетки так, что они занимают всю веревку. Один предмет был украден. Можно ли перевесить оставшиеся предметы так, чтобы веревка снова стала занятой полностью? Длина веревки, количество предметов и размеры каждого предмета известны.

## **Задание 2**

По входным числам n и k, вернуть последовательность перестановок. Набор [1, 2, 3, ..., n] содержит в общей сложности n! уникальных перестановок. Перечислив и пометив все перестановки по порядку, мы получим следующую последовательность чисел 𝑛𝑘 для n = 3:

𝑛1 = "123";

𝑛2 = "132";

𝑛3 = "213";

𝑛4 = "231";

𝑛5 = "312";

𝑛6 = "321".

Ограничения:

1 <= n <= 9.

1 <= k <= n!

# **Задание 1 (вариант 4)**

На веревке висят прямоугольные скатерти и салфетки так, что они занимают всю веревку. Один предмет был украден. Можно ли перевесить оставшиеся предметы так, чтобы веревка снова стала занятой полностью? Длина веревки, количество предметов и размеры каждого предмета известны.

## **С++**

Текст программы:

#include <iostream>

#include <vector>

#include <algorithm> //для использования сортировки

#include <locale.h> //подключение библиотеки для задач связанных из локализацией

using namespace std;

// Функция для вычисления размера предмета как произведение длины и ширины (так как предметы прямоугольные)

int calculateItemSize(int length, int width) {

return length \* width;

}

// Рекурсивная функция для проверки возможности равномерного распределения предметов по обеим веревкам так, чтобы веревка снова стала занятой полностью

bool canBalance(int ropeLength, const vector<pair<int, int>>& items, int index = 0, int leftSum = 0, int rightSum = 0) {

// pair для хранения размеров предметов в виде длины и ширины

// вектор items содержит пары <length, width> для каждого предмета

// Базовый случай: если веревка полностью занята (после каждого перемещения предмета)

if (leftSum == rightSum || leftSum + rightSum == 2 \* ropeLength) {

return true; // если все условия выполняются (предметы могут быть равномерно распределены по обеим веревкам веревки)

}

// Базовый случай: если веревка не занята и предметы закончились

if (index == items.size()) { // если индекс текущего предмета равен размеру вектора предметов - все предметы были рассмотрены, но веревка не занята

return false; // функция возвращает false

}

// Попытка перевесить предмет на левой веревке

if (canBalance(ropeLength, items, index + 1, leftSum + calculateItemSize(items[index].first, items[index].second), rightSum)) {

// ropeLength - длина веревки

// items - вектор размеров предметов

// index + 1 - следующий индекс предмета в векторе, переходим к следующему предмету

// leftSum + calculateItemSize(items[index].first, items[index].second) - сумма размеров предметов на левой части веревки после добавления текущего предмета

// аргументы items[index].first и items[index].second -

// для текущего предмета (предмета с индексом index в векторе items)

// длина и ширина извлекаются из пары <length, width> и передаются в функцию calculateItemSize.

// rightSum - сумма размеров предметов на правой части веревки без изменений

return true; // веревка может быть перевешена и предметы могут быть равномерно распределены по обеим веревкам

}

// Попытка перевесить предмет на правой веревке

if (canBalance(ropeLength, items, index + 1, leftSum, rightSum + calculateItemSize(items[index].first, items[index].second))) {

// ropeLength - длина веревки

// items - вектор размеров предметов

// index + 1 - следующий индекс предмета в векторе, переходим к следующему предмету

// leftSum - сумма размеров предметов на левой части веревки без изменений

// rightSum + calculateItemSize(items[index].first, items[index].second) - сумма размеров предметов на правой части веревки после добавления текущего предмета

// аргументы items[index].first и items[index].second -

// для текущего предмета (предмета с индексом index в векторе items)

// длина и ширина извлекаются из пары <length, width> и передаются в функцию calculateItemSize.

return true; // веревка может быть перевешена и предметы могут быть равномерно распределены по обеим веревкам

}

// Если предмет нельзя перевесить ни на одной веревке и равномерное распределение не удается

return canBalance(ropeLength, items, index + 1, leftSum, rightSum); // пытается перевесить предмет на одной из частей веревки, не влияя на сумму размеров предметов

// ropeLength - длина веревки

// items - вектор размеров предметов

// index + 1 - следующий индекс предмета в векторе, переходим к следующему предмету

// leftSum - сумма размеров предметов на левой части веревки без изменений

// rightSum - сумма размеров предметов на правой части веревки без изменений

}

int main() {

setlocale(LC\_ALL, "rus");

int ropeLength = 10;

vector<pair<int, int>> items = { {2, 2}, {3, 2}, {2, 3}, {2, 2}, {2, 3} }; // предметы заданы как длина и ширина и их количество(количество элементов в векторе)

// Сортируем предметы по убыванию размера, чтобы начать с самых больших

sort(items.begin(), items.end(), [](const pair<int, int>& a, const pair<int, int>& b) {

return calculateItemSize(a.first, a.second) > calculateItemSize(b.first, b.second);

});

// items.begin() и items.end() указывают на диапазон, который должен быть отсортирован - от начала вектора до конца

// [](const pair<int, int>& a, const pair<int, int>& b) - лямбда-функция, которая определяет критерий сортировки (a и b, которые являются элементами вектора items)

// return calculateItemSize(a.first, a.second) > calculateItemSize(b.first, b.second) - тело лямбда-функции, которая сравнивает размеры двух предметов a и b

// sort - сортирует элементы вектора items в соответствии с данным критерием, что означает, что самые большие предметы будут расположены в начале вектора

// Добавляем украденный предмет

pair<int, int> stolenItem = { 3, 2 }; // предмет украденный

items.push\_back(stolenItem);

// Удаляем украденный предмет из вектора предметов

items.erase(remove(items.begin(), items.end(), stolenItem), items.end());

// Проверяем, можно ли перевесить веревку и выводим результат

if (canBalance(ropeLength, items)) {

cout << "Да, веревку можно уравновесить." << endl;

}

else {

cout << "Нет, веревку нельзя уравновесить." << endl;

}

return 0;

**Результат работы программы:**

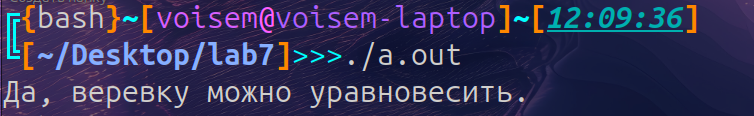


Рис 1. Результат работы программы на языке C++. Задание 1

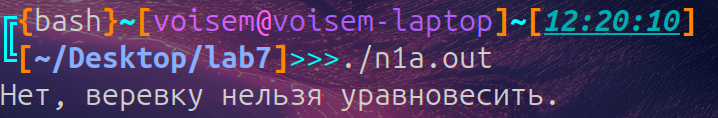


Рис 2. Результат работы программы на языке C++. Задание 1

**Python:**

# Функция для вычисления площади предмета

def calculate\_item\_size(length, width) :

return length \* width

# Функция для проверки возможности уравновешивания веревки

def can\_balance(rope\_length, items, index = 0, left\_sum = 0, right\_sum = 0) :

if left\_sum == right\_sum or left\_sum + right\_sum == 2 \* rope\_length :

return True

if index == len(items) :

return False

if can\_balance(rope\_length, items, index + 1, left\_sum + calculate\_item\_size(items[index][0], items[index][1]), right\_sum) :

return True

if can\_balance(rope\_length, items, index + 1, left\_sum, right\_sum + calculate\_item\_size(items[index][0], items[index][1])) :

return True

return can\_balance(rope\_length, items, index + 1, left\_sum, right\_sum)

# Длина веревки

rope\_length = 10

# Список предметов с их размерами

items = [(1, 1), (9, 9), (5, 5)]

# Сортировка предметов по убыванию площади

items.sort(key = lambda x : calculate\_item\_size(x[0], x[1]), reverse = True)

# Предмет, который был украден

stolen\_item = (3, 2)

# Добавление украденного предмета в список и удаление его из списка

items.append(stolen\_item)

items.remove(stolen\_item)

# Проверка возможности уравновешивания веревки с учетом украденного предмета

if can\_balance(rope\_length, items) :

print("Да, веревку можно уравновесить.")

else:

print("Нет, веревку нельзя уравновесить.")

**Результат работы программы:**

# 

Рис 3. Результат работы программы на языке Python. Задание 1

# **Задание 2 (вариант 3)**

По входным числам n и k, вернуть последовательность перестановок. Набор [1, 2, 3, ..., n] содержит в общей сложности n! уникальных перестановок. Перечислив и пометив все перестановки по порядку, мы получим следующую последовательность чисел 𝑛𝑘 для n = 3:

𝑛1 = "123";

𝑛2 = "132";

𝑛3 = "213";

𝑛4 = "231";

𝑛5 = "312";

𝑛6 = "321".

Ограничения:

1 <= n <= 9.

1 <= k <= n!

## **С++**

Текст программы:

#include <iostream>

#include <vector>

#include <string>

using namespace std;

string getKthPermutation(int n, int k, vector<int>& numbers) { // Объявление функции getKthPermutation с параметрами n, k и numbers

if (n == 1) {

return to\_string(numbers[0]); // Возвращаем строковое представление первого элемента вектора numbers

}

int fact = 1;

for (int i = 1; i <= n - 1; ++i) { // Цикл для вычисления факториала

fact \*= i; // Умножение fact на i

}

int index = (k - 1) / fact; // Вычисление индекса

int number = numbers[index]; // Получение числа по индексу

numbers.erase(numbers.begin() + index); // Удаление элемента из вектора

return to\_string(number) + getKthPermutation(n - 1, k - index \* fact, numbers); // Рекурсивный вызов функции с новыми параметрами

}

int main() {

setlocale(LC\_ALL, "ru");

int n, k; // Объявление переменных n и k

cout << "Введите n: ";

cin >> n;

cout << "Введите k: ";

cin >> k;

vector<int> numbers(n); // Создание вектора numbers размером n

for (int i = 0; i < n; ++i) { // Цикл для заполнения вектора числами от 1 до n

numbers[i] = i + 1;

}

cout << "Выход: " << getKthPermutation(n, k, numbers) << endl; // Вывод результата работы функции getKthPermutation

return 0;

}

**Результат работы программы:**

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рис 4. Результат работы программы(Задание 2). С++. Тест 1.

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

Рис 5. Результат работы программы(Задание 2).С++. Тест 2.

**Python:**

def get\_kth\_permutation(n, k, numbers): //Объявление функции getKthPermutation с параметрами n, k и numbers

if n == 1:

return str(numbers[0]) //Возвращаем строковое представление первого элемента вектора numbers

fact = 1

for i in range(1, n): //Цикл для вычисления факториала.

fact \*= i //Умножение fact на i

index = (k - 1) // fact

number = numbers[index]

numbers.pop(index)

return str(number) + get\_kth\_permutation(n - 1, k - index \* fact, numbers) // Рекурсивный вызов функции с новыми параметрами.

if \_\_name\_\_ == "\_\_main\_\_":

n = int(input("Введите n: "))

k = int(input("Введите k: "))

numbers = [i + 1 for i in range(n)]

print("Выход:", get\_kth\_permutation(n, k, numbers))

**Результат работы программы:**

Изображение выглядит как текст, снимок экрана, Шрифт, дизайн

Автоматически созданное описание

Рис 6. Результат работы программы(Задание 2). Python. Тест 1.

Изображение выглядит как текст, снимок экрана, Шрифт, дизайн

Автоматически созданное описание

Рис 7. Результат работы программы(Задание 2). Python. Тест 2.

**Вывод:**

Проделанный ход работы позволил изучить рекурсивное программирование, методы разработки эффективных алгоритмов.

**Ссылка на репозиторий в GitHub:**  git@github.com:IlyaShum/Laba\_7.git