|  |
| --- |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования |
| **«МИРЭА – Российский технологический университет»** |
| **РТУ МИРЭА** |
|  |

| **Отчет по выполнению практического задания № 7** | |
| --- | --- |
| **Тема:** | |
| **«Рекурсивные алгоритмы и их реализация»** | |
| Дисциплина: «Структуры и алгоритмы обработки данных» | |
|  | Выполнил студент: Лелюхин Н.С. |
|  | Группа: ИКБО-74-23 |

Москва – 2024

СОДЕРЖАНИЕ

[1 ЦЕЛЬ 3](#_gjdgxs)

[2 ЗАДАНИЕ №1 3](#_30j0zll)

[2.1 Формулировка задачи 3](#_1fob9te)

[2.2 Итерационный алгоритм 3](#_2et92p0)

[2.3 Рекуррентная зависимость 6](#)

[2.4 Рекурсивная функция 6](#_17dp8vu)

[2.5 Объединение программ 9](#_1ci93xb)

[3 ЗАДАНИЕ №2 12](#_2bn6wsx)

[3.1 Формулировка задачи 12](#_qsh70q)

[3.2 Рекурсивная функция 12](#_u290pxmzmczz)

[4 ВЫВОДЫ 14](#_ihv636)

[5 ЛИТЕРАТУРА 15](#_32hioqz)

# **1 ЦЕЛЬ**

Получить знания и практические навыки по разработке и реализации рекурсивных процессов.

# **2 ЗАДАНИЕ №1**

## **2.1 Формулировка задачи (Вариант 2, в списке 18)**

Разработать и протестировать рекурсивные функции в соответствии с задачами варианта

Требования к выполнению первой задачи варианта:

• приведите итерационный алгоритм решения задачи

• реализуйте алгоритм в виде функции и отладьте его

• определите теоретическую сложность алгоритма

• опишите рекуррентную зависимость в решении задачи

• реализуйте и отладьте рекурсивную функцию решения задачи

• определите глубину рекурсии, изменяя исходные данные

• определите сложность рекурсивного алгоритма, используя метод подстановки и дерево рекурсии

• приведите для одного из значений схему рекурсивных вызовов

• разработайте программу, демонстрирующую выполнение обеих функций и покажите результаты тестирования.

Задание: Найти n-ое число Фибоначчи.

## **2.2 Итерационный алгоритм**

Для этого мы используем итерационный метод, который начинается с определения первых двух чисел Фибоначчи: 0 и 1. Затем мы последовательно вычисляем следующее число Фибоначчи путем сложения двух предыдущих чисел. Мы продолжаем этот процесс до тех пор, пока не достигнем n-го числа Фибоначчи. В конце мы возвращаем значение n-го числа Фибоначчи.

Реализуем алгоритм в функции fibonacciIterative в блок кода 1.

| #include <iostream> using namespace std;  int fibonacciIterative(int n) {  if (n <= 1)   return n;  int prev = 0;  int current = 1;  for (int i = 2; i <= n; ++i) {  int next = prev + current;  prev = current;  current = next;  }  return current; }  int main() {  int n;  cout << "Введите значение n: ";  cin >> n;  cout << "Число Фибоначчи в позиции" << n << " (итерация): " << fibonacciIterative(n) << endl;   return 0; } |
| --- |

Блок код 1 - Реализация итерационного алгоритма для задачи 1

Проведем отладку данной функции при n равной 10 (рис. 1).

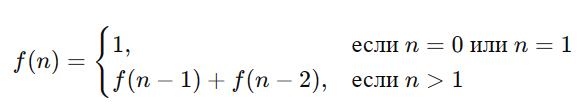


Рисунок 1 - Тестирование алгоритма

Теоретическая сложность итерационного алгоритма нахождения n-го числа Фибоначчи - O(n), где n - номер числа Фибоначчи, которое мы хотим найти. Это потому, что для вычисления n-го числа Фибоначчи мы должны выполнить n-1 итераций, каждая из которых требует постоянного времени. Таким образом, общее время выполнения зависит линейно от n.

## **2.3 Рекуррентная зависимость**

Рекуррентная зависимость для нахождения n-го числа Фибоначчи может быть описана следующим образом:



Это означает, что для нахождения n-го числа Фибоначчи мы суммируем два предыдущих числа Фибоначчи (n-1 и n-2), когда n больше 1. Если n равно 0 или 1, мы возвращаем 0 или 1 соответственно

## **2.4 Рекурсивная функция**

Реализуем рекурсивную функцию для данной задачи (блок кода 2).

| #include <iostream> using namespace std;  int fibonacciIterative(int n) {  if (n <= 1)   return n;   int prev = 0;  int current = 1;   for (int i = 2; i <= n; ++i) {  int next = prev + current;  prev = current;  current = next;  }   return current; }    int fibonacciRecursive(int n) {  if (n <= 1)   return n;   return fibonacciRecursive(n - 1) + fibonacciRecursive(n - 2); }  int main() {  int n;  cout << "Введите значение n: ";  cin >> n;   cout << "Число Фибоначчи в позици " << n << " (рекурсия): " << fibonacciRecursive(n) << endl;   return 0; } |
| --- |

Блок кода 2 - Рекурсивная функция для задачи 1

Проведем отладку данной функции на тех же значениях, что и при отладке итерационного алгоритма (рис. 2).



Рисунок 2 - Тестирование адаптированной программы

Глубина рекурсии для нахождения n-го числа Фибоначчи будет равна n. Это потому, что каждый раз, когда мы вызываем функцию для вычисления n-го числа Фибоначчи, мы делаем два рекурсивных вызова для (n-1) и (n-2), и эти вызовы продолжаются до достижения базовых случаев (n=0 или n=1). Таким образом, мы будем иметь дерево рекурсии глубиной n, где каждый узел имеет два потомка.

Рассмотрим пример схемы рекурсивных вызовов для нахождения n-го числа Фибоначчи. Для простоты возьмем n=5 (рис. 3).

| F(5)  |  +--> F(4)  | |  | +--> F(3)  | | |  | | +--> F(2)  | | | |  | | | +--> F(1)  | | | |  | | | +--> F(0)  | | |  | | +--> F(1)  | |  | +--> F(2)  |  +--> F(3) |
| --- |

Рисунок 3 - Вызов рекурсии для массива значений

## **2.5 Объединение программ**

Объединим программы итерационного алгоритма и рекурсивной функции в блок коде 3 с выполнением обоих алгоритмов. Продемонстрируем результаты работы программы на рисунке 4.

| #include <iostream> using namespace std;  int fibonacciIterative(int n) {  if (n <= 1)   return n;   int prev = 0;  int current = 1;   for (int i = 2; i <= n; ++i) {  int next = prev + current;  prev = current;  current = next;  }   return current; } int fibonacciRecursive(int n) {  if (n <= 1)   return n;   return fibonacciRecursive(n - 1) + fibonacciRecursive(n - 2); } int main() {  int n;  cout << "Введите значение n: ";  cin >> n;   cout << "Число Фибоначчи в позиции " << n << " (итерация): " << fibonacciIterative(n) << endl;   cout << "Число Фибоначчи в позиции " << n << " (рекурсия): " << fibonacciRecursive(n) << endl;   return 0; } |
| --- |

Блок кода 3 - Объединение программ

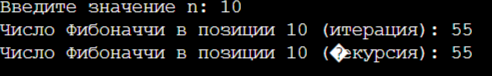


Рисунок 4 - Тестирование программы

# 

# **3 ЗАДАНИЕ №2**

## **3.1 Формулировка задачи**

Требования к выполнению второй задачи варианта:

• рекурсивную функцию для обработки списковой структуры согласно варианту. Информационная часть узла – простого типа – целого;

• для создания списка может быть разработана простая или рекурсивная функция по желанию (в тех вариантах, где не требуется рекурсивное создание списка);

• определите глубину рекурсии

• определите теоретическую сложность алгоритма

• разработайте программу, демонстрирующую работу функций и покажите результаты тестов.

Задание: В однонаправленном списке из n элементов найти элемент с заданным значением и вернуть на него указатель

## **3.2 Рекурсивная функция**

Реализуем задачу с помощью рекурсивной функции(блок кода 4).

| #include <iostream> using namespace std;  struct Node {  int data;  Node\* next;   Node(int value) : data(value), next(nullptr) {} }; Node\* findElementRecursive(Node\* head, int value) {  // Если достигнут конец списка или список пуст, вернуть nullptr  if (head == nullptr)  return nullptr;   // Если значение текущего узла совпадает с заданным, вернуть указатель на текущий узел  if (head->data == value)  return head;   // Рекурсивно искать в оставшейся части списка  return findElementRecursive(head->next, value); } int main() {  // Создаем список: 1 -> 2 -> 3 -> 4 -> 5  Node\* head = new Node(1);  head->next = new Node(2);  head->next->next = new Node(3);  head->next->next->next = new Node(4);  head->next->next->next->next = new Node(5);   int valueToFind = 3;   // Ищем элемент в списке  Node\* result = findElementRecursive(head, valueToFind);   if (result != nullptr) {  cout << "Найден элемент: " << result->data << endl;  } else {  cout << "Элемент не найден" << endl;  }   return 0; } |
| --- |

Блок кода 4 - Программа для задания 2 с рекурсивной функцией

Продемонстрируем результаты работы программы на рисунке 4.



Рисунок 4 - Тестирование программы

В данной функции, каждый раз, когда она вызывается рекурсивно, мы перемещаемся на следующий узел списка. Таким образом, глубина рекурсии будет равна количеству узлов в списке, пока мы не достигнем элемента с заданным значением или не достигнем конца списка.

В данном случае, глубина рекурсии будет равна количеству узлов в списке до того момента, когда мы найдем элемент с заданным значением или дойдем до конца списка. Для приведенного кода с функцией findElementRecursive и списком из 5 узлов, глубина рекурсии будет равна 5.

Теоретическая сложность рекурсивного поиска элемента в односвязном списке зависит от числа узлов в списке и положения искомого элемента.

Если искомый элемент находится в самом начале списка или отсутствует в списке, то время выполнения будет пропорционально числу узлов в списке, то есть O(n), где n - количество узлов в списке.

Однако, если искомый элемент находится в конце списка или отсутствует в списке, алгоритм также должен пройти по всем узлам списка, чтобы установить это. В этом случае также будет O(n) операций.

Таким образом, теоретическая сложность алгоритма рекурсивного поиска элемента в односвязном списке - O(n), где n - количество узлов в списке.

# **4 ВЫВОДЫ**

В процессе выполнения заданий "Найти n-ое число Фибоначчи" и "В однонаправленном списке из n элементов найти элемент с заданным значением и вернуть на него указатель, реализованные с помощью рекурсии", мы приобрели понимание и практические навыки по разработке и реализации рекурсивных процессов.

Для задачи "Найти n-ое число Фибоначчи" мы изучили алгоритм рекурсивного поиска числа Фибоначчи, определили глубину рекурсии и теоретическую сложность алгоритма. Это помогло нам углубить понимание работы рекурсивных функций и оценить их эффективность в решении задач.

Для задачи "В однонаправленном списке из n элементов найти элемент с заданным значением и вернуть на него указатель" мы разработали рекурсивную функцию, реализующую поиск элемента в однонаправленном списке. Мы определили глубину рекурсии, теоретическую сложность алгоритма и изучили рекурсивную зависимость в решении задачи. Это позволило нам научиться применять рекурсивные подходы к решению задач с использованием однонаправленных списков.

Таким образом, выполнение данных заданий позволило нам углубить знания и получить практические навыки в области разработки и реализации рекурсивных процессов, что является целью данной работы.

# 

# **5 ЛИТЕРАТУРА**

1. Бхаргава А. Грокаем алгоритмы. Иллюстрированное пособие для программистов и любопытствующих. – СПб: Питер, 2017. – 288 с.

2. Вирт Н. Алгоритмы + структуры данных = программы. – М.: Мир, 1985. – 406 с.

3. Кнут Д.Э. Искусство программирования, том 3. Сортировка и поиск, 2-е изд. – М.: ООО «И.Д. Вильямс», 2018. – 832 с.

4. Кораблин Ю.П. Структуры и алгоритмы обработки данных: учебно-методическое пособие / Ю.П. Кораблин, В.П. Сыромятников, Л.А. Скворцова. – М.: РТУ МИРЭА, 2020. — 219 с.

5. Кормен Т.Х. и др. Алгоритмы: построение и анализ, 3-е изд. – М.: ООО «И.Д.Вильямс», 2013. – 1328 с.

6. Макконнелл Дж. Основы современных алгоритмов. Активный обучающий метод. 3-е доп. изд., - М.: Техносфера, 2018. – 416 с.

7. Седжвик Р. Фундаментальные алгоритмы на C++. Анализ/Структуры данных/Сортировка/Поиск. – К.: Издательство «Диасофт», 2001. – 688 с.

8. Скиена С. Алгоритмы. Руководство по разработке, - 2-е изд. – СПб: БХВ-Петербург, 2011. – 720 с.

9. Хайнеман Д. и др. Алгоритмы. Справочник с примерами на C, C++, Java и Python, 2-е изд. – СПб: ООО «Альфа-книга», 2017. – 432 с.

10. AlgoList – алгоритмы, методы, исходники [Электронный ресурс]. URL: http://algolist.manual.ru/ (дата обращения 15.03.2022).

11. Алгоритмы – всё об алгоритмах / Хабр [Электронный ресурс]. URL: https://habr.com/ru/hub/algorithms/ (дата обращения 15.03.2022).