МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
"МИРЭА - Российский технологический университет"
РТУ МИРЭА

Отчет по выполнению практического задания №9
Тема: Рекурсивные алгоритмы и их реализация
Дисциплина: «Структуры и алгоритмы обработки данных»

Выполнил студент Антонов А.Д.

Группа ИКБО-01-20

Москва 2021

СОДЕРЖАНИЕ

1. Ответы на вопросы.

1.1. Определение рекурсивной функции.

Рекурсивная функция – числовая функция числового аргумента, которая в своем определении содержит себя же.

1.2. Шаг рекурсии

Шаг рекурсии – активизация очередного рекурсивного выполнения функции при других исходных данных.

1.3. Глубина рекурсии

Глубина рекурсии – наибольшее одновременное кол-во рекурсивных вызовов функции, определяющее максимальное кол-во слоев рекурсивного стека, в котором хранятся отложенные вычисления.

1.4. Условие завершения рекурсии

Условие завершения рекурсии – условие, останавливающее вызов рекурсивной функции и формирующее конкретное простейшее решение вычислительного процесса.

1.5. Виды рекурсий

Линейная рекурсия: каждый вызов порождает только один новый вызов. В этом случае дерево рекурсивных вызовов содержит только одну ветвь. Каскадная рекурсия: каждый вызов порождает несколько новых вызовов.

1.6. Прямая и косвенная рекурсии

Прямая рекурсия – непосредственный вызов функции из описания самой рекурсивной функции.

Косвенная рекурсия – циклическая последовательность вызовов нескольких алгоритмов друг друга.

1.7. Организация стека рекурсивных вызовов

Рекурсивный стек – область памяти, в которую заносятся значения всех локальных переменных алгоритма (программы) в момент рекурсивного обращения. Каждое такое обращение формирует один слой данных стека. При завершении вычислений по конкретному обращению а из стека считывается соответствующий ему слой данных, и локальные переменные восстанавливаются, снова принимая значения, которые они имели в момент обращения а.

2. Отчет по разработанной программе. Вариант 2.

2.1. Задание 1

2.1.1. Условие задачи

Найти n-ое число Фибоначчи: дана последовательность чисел, последующее из которых равно сумме двух предыдущих. Начинается последовательность с двух единиц (1 1 2 3 5 и т.д.). Необходимо найти n-ый член этой последовательности.

2.1.2. Постановка задачи

Разработать алгоритм для реализации задачи рекурсивным и итерационным способами для заданного с клавиатуры числа n.

2.1.3. Описание алгоритма

Алгоритм можно описать следующим рекуррентным соотношением:

$$F_n = \begin{cases} 0, & n = 0; \\ 1, & n = 1; \\ F_{n-1} + F_{n-2}, & n \geqslant 2. \end{cases}$$

2.1.4. Коды используемых функций

```
int FibRec (int n)
{
    if (n == 0) return 0;
    if (n == 1) return 1;
    return FibRec( n: n - 1) + FibRec( n: n - 2);
}

int main()
{
    int n;
    cout << "Enter n: ";
    cin >> n;
    cout << "Result (iterative): " << FibIter(n) << endl;
    cout << "Result (recursive): " << FibRec(n);
}
```

2.1.5. Ответы на задания по задаче 1

Итерационный алгоритм решения задачи:

Алгоритм: при помощи цикла for меняем значения a и b (равные 0 и 1 соответственно) таким образом, что a = b, b = a + b

Код:

```
int FibIter (int n)
{
    int a = 0, b = 1, temp;
    for (int i = 0; i < n; i++)
        { temp = a; a = b; b += temp; }
    return a;
}
```

Теоретическая сложность:

В данном алгоритмы задействован цикл for, вложенность отсутствует, откуда следует, что сложность алгоритма линейная.

Рекуррентная зависимость:

Рекуррентная зависимость была рассмотрена в пункте 2.1.3.

Реализация рекурсивной функции решения задачи:

Рекурсивная функция решения данной задачи приведена в пункте 2.1.4.

Глубина рекурсии:

Максимальное число одновременных рекурсивных вызовов – n, где n – порядковый номер числа, которое необходимо найти.

Сложность рекурсивного алгоритма методом подстановки и дерева рекурсии (при n = 5):

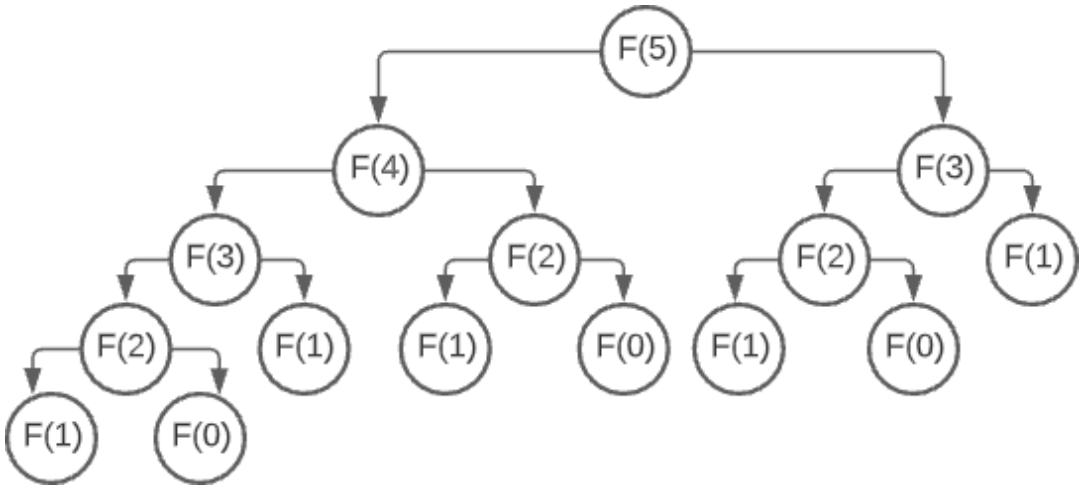

Рис. 1. Дерево рекурсии алгоритма

На рисунке 1 видно, что за каждым вызовом следует еще два

рекурсивных вызова, но с меньшим n. Поэтому теоретическая сложность $O(2^n)$.

2.1.6. Код программы и результаты тестирования

```cpp
#include <iostream>
using namespace std;

int FibIter (int n)
{
    int a = 0, b = 1, temp;
    for (int i = 0; i < n; i++)
        { temp = a; a = b; b += temp; }
    return a;
}

int FibRec (int n)
{
    if (n == 0) return 0;
    if (n == 1) return 1;
    return FibRec( n: n - 1) + FibRec( n: n - 2);
}

int main()
{
    int n;
    cout << "Enter n: ";
    cin >> n;
    cout << "Result (iterative): " << FibIter(n) << endl;
    cout << "Result (recursive): " << FibRec(n);
}
```

Таблица 1. Тестирование рекурсивной и итерационной функций

Номер теста	Входные данные	Ожидаемый результат	Результат выполнения программы
1	7	13	Enter n: 7 Result (iterative): 13 Result (recursive): 13
2	5	5	Enter n: 5 Result (iterative): 5 Result (recursive): 5

Из таблицы 1 видно, что обе функции отработали верно.

2.2. Задание 2

2.2.1. Условие задачи

В однонаправленном списке найти элемент с заданным значением и вернуть на него указатель.

2.2.2. Постановка задачи

Разработать рекурсивную функцию нахождения элемента однонаправленного списка.

2.2.3. Описание алгоритма

В данном задании используется однонаправленный список размера n. Структура узла списка содержит информационную часть (целое число) и указатель на следующий узел. Нахождение элемента будет осуществляться следующим образом:

1) Если главный узел равен nullptr, возвращаем nullptr.
2) Если значение главного узла равно искомому, возвращаем этот узел.
3) Иначе возвращаем результат вызова функции find(head->next, v)

2.2.4. Коды используемых функций

```
21  struct Node
22  {
23      int data = 0;
24      Node* next;
25      Node (int data, Node *nextNode = nullptr):
26          data(data), next(nextNode) {}
27  };
28
29  void insert (Node *&head, int data)
30  {
31      Node* newHead = new Node(data, head);
32      head = newHead;
33  }
```

```
35  Node* createList (int length)
36  {
37      Node *head = nullptr;
38      int v;
39      for (int i = 0; i < length; i++)
40      {
41          cout << "Enter value: ";
42          cin >> v;
43          //cout << endl;
44          insert(head, v);
45      }
46      return head;
47  }
48
49  void showList (Node *head)
50  {
51      Node *node = head;
52      while (node->next != nullptr)
53      {
54          cout << node->data << ' ';
55          node = node->next;
56      }
57      cout << endl;
58  }
```

2.2.5. Ответы на задания по задаче 2

Реализация рекурсивной функции решения задачи:

```
60  Node* find (Node *head, int v)
61  {
62      if (head == nullptr || head->data == v)
63          return head;
64      return find(head->next, v);
65  }
```

Глубина рекурсии:

Поиск узла списка осуществляется последовательно, отсюда следует, что глубина рекурсии для данного алгоритма равна length, где length – длина списка.

Теоретическая сложность алгоритма:

Каждый вызов функции вызывает вложенную функцию только один раз до тех пор, пока не выполнится одно из условий выхода. Поэтому теоретическая сложность O(n).

2.2.6. Код программы и результаты тестирования

```
67  int main()
68  {
69      int length, v;
70      cout << "List size: ";
71      cin >> length;
72      Node *list = createList(length);
73      cout << "Enter value to find: ";
74      cin >> v;
75      Node *node = find(list, v);
76      if (node != nullptr)
77          cout << "Value found: " << node->data << endl;
78      else
79          cout << "Value not found!" << endl;
80  }
```

Таблица 2. Тестирование программы

Номер теста	Входные данные	Ожидаемый результат	Результат выполнения программы
1	3 2 9 0 0	Value found: 0	List size: 3 Enter value: 2 Enter value: 9 Enter value: 0 Enter value to find: 0 Value found: 0
2	2 -5 5 -5	Wrong size	List size: 2 Enter value: -5 Enter value: 5 Enter value to find: -5 Value found: -5
3	2 -12 3 5	Value not found!	List size: 2 Enter value: -12 Enter value: 3 Enter value to find: 5 Value not found!

Из таблицы 2 видно, что алгоритм отработал верно.

ВЫВОДЫ

В ходе данной практической работы были получены знания и практические навыки по разработке и реализации рекурсивных функций на примере последовательности Фибоначчи. Для этой же задачи была разработана функция решения итерационным способом. Оба алгоритма успешно прошли проверку. Также была разработана рекурсивная функция для поиска узла односвязного списка по значению, которая тоже успешно прошла проверку. Для каждой рекурсивной функции были определены: рекурсивная зависимость, глубина рекурсии и теоретическая сложность алгоритма.

СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ

1. Лекционный материал по структуре и алгоритмам обработки данных Гданского Н.И.
2. https://code-live.ru/post/cpp-array-tutorial-part-2/
3. Википедия: Инвариант цикла [Электронный ресурс]: https://ru.wikipedia.org/wiki/Инвариант_цикла
4. Википедия: Вычислительная сложность [Электронный ресурс]: https://ru.wikipedia.org/wiki/Вычислительная_сложность