



МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт Информационных технологий

Кафедра Математического обеспечения и стандартизации информационных
технологий

Отчет по выполнению индивидуального задания № 6

по дисциплине «Структуры и алгоритмы обработки данных»

Тема: «Рекурсивные алгоритмы и их реализация»

Выполнил:

Студент группы ИКБО-29-20

Хан Анастасия Александровна

Проверил:

Копылова А.В.

Москва 2021

СОДЕРЖАНИЕ

Ответы на вопросы.....	3
------------------------	---

Отчет по задаче 1

Условие задачи.....	4
Постановка задачи.....	4
Описание алгоритма – рекуррентная зависимость.....	4
Коды используемых функций... ..	4
Ответы на задания по задаче 1: список требований к задаче 1	5
Код программы и скриншоты результатов тестирования.....	15

Отчет по задаче 2

Условие задачи.....	18
Постановка задачи.....	18
Описание алгоритма – рекуррентная зависимость.....	18
Коды используемых функций... ..	18
Ответы на задания по задаче 1: список требований к задаче 1	18
Код программы и скриншоты результатов тестирования.....	20
Выводы	21
Информационные источники	21

Ответы на вопросы

1) Определение рекурсивной функции

Рекурсивная функция – функция, которая в своем теле содержит обращение к самой себе с измененным набором параметров.

2) Шаг рекурсии

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

3) Глубина рекурсии

Глубина рекурсивных вызовов – наибольшее одновременное количество рекурсивных вызовов функции, определяющее максимальное количество слоев рекурсивного стека, в котором осуществляется хранение отложенных вычислений.

4) Условие завершения рекурсии

Условие завершения рекурсии – условие, которое определяет завершение рекурсии и формирование конкретного простейшего действия вычислительного процесса.

5) Виды рекурсии

- **линейная** – рекурсия, в которой каждый вызов порождает ровно один новый вызов

- **каскадная** – рекурсия, в которой каждый вызов порождает несколько новых вызовов

6) Прямая и косвенная рекурсия

Прямая рекурсия – рекурсия, в которой вызов функции самой себя делается непосредственно в этой же функции.

Косвенная рекурсия - рекурсия, в которой данная функция вызывается из другой функции, которая вызывается из данной функции.

7) Организация стека рекурсивных вызовов

Вся иерархия вызовов хранится в стеке вызовов. Когда метод вызывает сам себя, новым локальным переменным и параметрам выделяется место в стеке и код метода выполняется с этими новыми начальными значениями. При каждом возврате из рекурсивного вызова старые локальные переменные и параметры удаляются из стека, и выполнение продолжается с момента вызова внутри метода.

Персональный вариант: $29\%16 + 1 = 15$.

Отчет по задаче 1

1. Условие задачи

Реализовать выполнения алгоритма “Ханойская башня”

2. Постановка задачи

Привести итерационные и рекурсивные алгоритмы решения задачи, описать рекурсивную зависимость. Написать две функции – рекурсивную и итерационную, которые должны выполнять алгоритм Ханойской башни. Отладить функции, оценить их теоретические сложности.

3. Описание алгоритма – рекуррентная зависимость

В начале работы у нас есть n дисков, три стержня – начальный (1), дополнительный (2) и конечный (3).

Рассмотрим башню при $n = 1, 2, 3$:

$n = 1, 1 \Rightarrow 3$.

$n = 2, 1 \Rightarrow 2, 1 \Rightarrow 3, 2 \Rightarrow 3$.

$n = 3, 1 \Rightarrow 3, 1 \Rightarrow 2, 3 \Rightarrow 2, 1 \Rightarrow 3, 2 \Rightarrow 1, 2 \Rightarrow 3, 1 \Rightarrow 3$.

Если $n = 4$, то мы должны свести задачу к известной – переносе 3-х дисков, но не на конечный (для пирамиды из 3-х дисков – он дополнительный), стержень, а на дополнительный (для пирамиды из 3-х дисков – он конечный), чтобы потом нижний и самый большой 4 диск перенести на конечный стержень. А далее пирамиду из 3-х дисков перенести с дополнительного стержня (он теперь будет начальный) на конечный с помощью начального (теперь он является дополнительным). Мы выявили, что для конкретной пирамиды размером k , появляется зависимость: мы перекладываем на стержень, который для нее является дополнительным, перемещаем целевой диск на конечный стержень для данной пирамиды; потом стержень, на котором лежат $k - 1$ дисков, является уже не дополнительным, а начальным, конечный остается конечным, а бывший начальный становится дополнительным. Таким образом, в чередовании стержней, на которые надо переносить, и заключается идея решения задачи.

$$Hanoi(n, from, free, to) = \begin{cases} (1)n! = 0, Hanoi(n - 1, from, to, free) \\ (2)n = 0, \quad \text{переход к (3)} \\ (3)Hanoi(n, from, free, to) \end{cases}$$

4. Коды используемых функций

```
void hanoi(int number, int from, int free, int to)
{
    if (number != 0)
    {
        hanoi(number-1, from, to, free);
        cout << "Передвигаем " << number << "-й диск с " << from << "-го
стержня на " << to << "-ий" << endl;
        hanoi(number-1, free, from, to);
    }
}
```

5. Ответы на задания по задаче 1: список требований к задаче 1

1. Приведите итерационный алгоритм решения задачи

Итерационный алгоритм решения таков:

Необходимо создать стек, в котором будет храниться задача (структура) – с какого стержня на какой перекладывать, сколько дисков осталось переложить и какой шаг из трех будет выполняться (0, 1, 2). По позиции в цикле будет выбираться действие, которое надо сделать.

Изначально добавляем в стек задач перемещение пирамиды из n дисков с начального стержня на конечный с помощью вспомогательного, с позицией равной 0 (чтобы переместить верхние $n - 1$ дисков).

Далее, пока стек не пуст, выполняется цикл, из верхнего элемента стека берется номер действия (0, 1 или 2), выбирается нужный блок программы и выполняется:

- 0 – добавление задачи перемещения диска с начального стержня на вспомогательный (если количество дисков, которые надо переместить, равно 0, то удаляем задачу из стека);
- 1 – вывод какой диск переставляем с какого стержня на какой, добавление задачи перестановки диска с вспомогательного стержня на конечный;
- 2 – удаление задачи из стека.

2. Реализуйте алгоритм в виде функции и отладьте его

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

```
struct Condition
```

```

{
    int dNumber, from, free, to, position;
};
Condition* setNewTask(int n, int from, int free, int to, int position)
{
    Condition* cond = new Condition;
    cond->dNumber = n;
    cond->from = from;
    cond->free = free;
    cond->to = to;
    cond->position = position;
    return cond;
}

void hanoi_iteration(int n)
{
    vector<Condition*> stack;
    stack.push_back(setNewTask(n, 1, 2, 3, 0));
    while (stack.size() > 0)
    {
        Condition *cond = stack.back();
        switch (cond->position)
        {
            case 0:
            {
                if (cond->dNumber == 0)
                {
                    stack.pop_back();
                }
                else
                {
                    cond->position++;
                    stack.push_back(setNewTask(cond-
>dNumber - 1, cond->from, cond->to, cond->free, 0));
                }
                break;
            }
            case 1:
            {
                cond->position++;
                cout << "Передвигаем " << cond->dNumber
<< "-й диск с " << cond->from << "-го стержня на " << cond->to << "-ий" <<
endl;
                stack.push_back(setNewTask(cond->dNumber

```


else 1), блок case 1 исполнился 1 раз, блок case 2 исполнился 1 раз.

Случай для двух дисков, $n = 2$, $position = 0$. Вход в цикл, (сравнение, размер стека $1 > 0$), перемена состояния на $position = 1$ и создание элемента с $n = 1$ и $position = 0$; при $n = 1$ мы знаем, что количество операций сравнения в цикле равно 6 (6 сравнение – это уже сравнение стека только с элементом $n = 2$). Далее элемент $n = 2$ сменяет состояние $position$ на 2, опять порождается элемент $n = 1$ и $position = 0$, 6 сравнений. Далее элемент с $n = 2$ и $position = 2$ удаляется из стека. Происходит сравнение размера стека, $0 = 0$, выход.

Итого 14 операций сравнения при $n = 2$

Также: блок case 0 исполнился 7 раз (в нем if исполнился 4 раза, else 3), блок case 1 исполнился 3 раза, блок case 2 исполнился 3 раза.

Аналогично при $n = 3$, сравнение размера стека при входе + порождаются два элемента с $n = 2$ (Всего 28 сравнений) + сравнение размера стека при выходе = 30 операций сравнения.

Также: блок case 0 исполнился 15 раз (в нем if исполнился 8 раза, else 7), блок case 1 исполнился 7 раз, блок case 2 исполнился 7 раз.

При $n = 1$: количество операций сравнения в цикле: $6 = 1 + 2 \cdot (2^{n+1} - 2) + 1 = 2^{n+2} - 2$

При $n = 2$: количество операций сравнения в цикле: $6 = 1 + 2 \cdot (2^{n+1} - 2) + 1 = 2^{n+2} - 2$

При $n = k$: количество операций сравнения в цикле: $6 = 1 + 2 \cdot (2^{k+1} - 2) + 1 = 2^{k+2} - 2$

Количество входов в блок

- case 0: $2^{k+1} - 1$ (Блок if выполняется 2^k раз, Блок else выполняется $2^k - 1$ раз)
- case 1 : $2^k - 1$
- case 2: $2^k - 1$

Оператор	Время выполнения инструкции	Количество выполнений оператора
vector<Condition*> stack;	c1	1
stack.push_back(setNewTask(n, 1, 2, 3, 0));	c2	5
while (stack.size() > 0)	c3	$2^{n+2} - 2$
{		
Condition *cond = stack.back();	c4	$2^{n+2} - 2$
switch (cond->position)	c5	$2^{n+2} - 2$

case 0:	c6	$2^{n+2}-2$
{		
if (cond->dNumber == 0)	c7	$2^{n+1} - 1$
{		
stack.pop_back();	c8	2^n
}		
else		
{		
cond->position++;	c9	$2^n - 1$
stack.push_back(setNewTask(cond->dNumber - 1, cond->from, cond->to, cond->free, 0));	c10	$6*(2^n - 1)$
}		
break;	c11	$2^n - 1$
}		
case 1:	c12	$2^{n+2}-2 - 2^n - 1 = 3*2^n - 3$
{		
cond->position++;	c13	$2^n - 1$
cout << "Передвигаем " << cond->dNumber << "-й диск с " << cond->from << "-го стержня на " << cond->to << "-ий" << endl;	c14	$2^n - 1$
stack.push_back(setNewTask(cond->dNumber - 1, cond->free, cond->from, cond->to, 0));	c15	$6*(2^n - 1)$
break;	c16	$2^n - 1$
}		
case 2:	c17	$2^n - 1$
{		
stack.pop_back();	c18	$2^n - 1$
break;	c19	$2^n - 1$
}		
}		
}		

$$\begin{aligned}
T(n) = & c1 + 5*c2 + c3*(2^{n+2}-2) + c4*(2^{n+2}-2) + c5*(2^{n+2}-2) + c6*(2^{n+2}-2) + \\
& c7*(2^{n+1} - 1) + c8 * 2^n + c9*(2^n - 1) + c10*6*(2^n - 1) + c11*(2^n - 1) + c12*(3*2^n - 3) \\
& + c13*(2^n - 1) + c14*(2^n - 1) + c15*6*(2^n - 1) + c16*(2^n - 1) + c17*(2^n - 1) + \\
& c18*(2^n - 1) + c19*(2^n - 1) = 2^n*(4*c3 + 4*c4 + 4*c5 + 4*c6 + 2*c7 + c8 + c9 + \\
& 6*c10 + c11 + 3*c12 + c13 + c14 + 6*c15 + c16 + c17 + c18 + c19) + (c1 + 5*c2 - 4*c3 - \\
& 4*c4 - 4*c5 - 4*c6 - 2*c7 - c9 - 6*c10 - c11 - 3*c12 - c13 - c14 - 6*c15 - c16 - c17 - \\
& c18 - c19) \Rightarrow O(n) = 2^n
\end{aligned}$$

4. Опишите рекуррентную зависимость в решении задачи

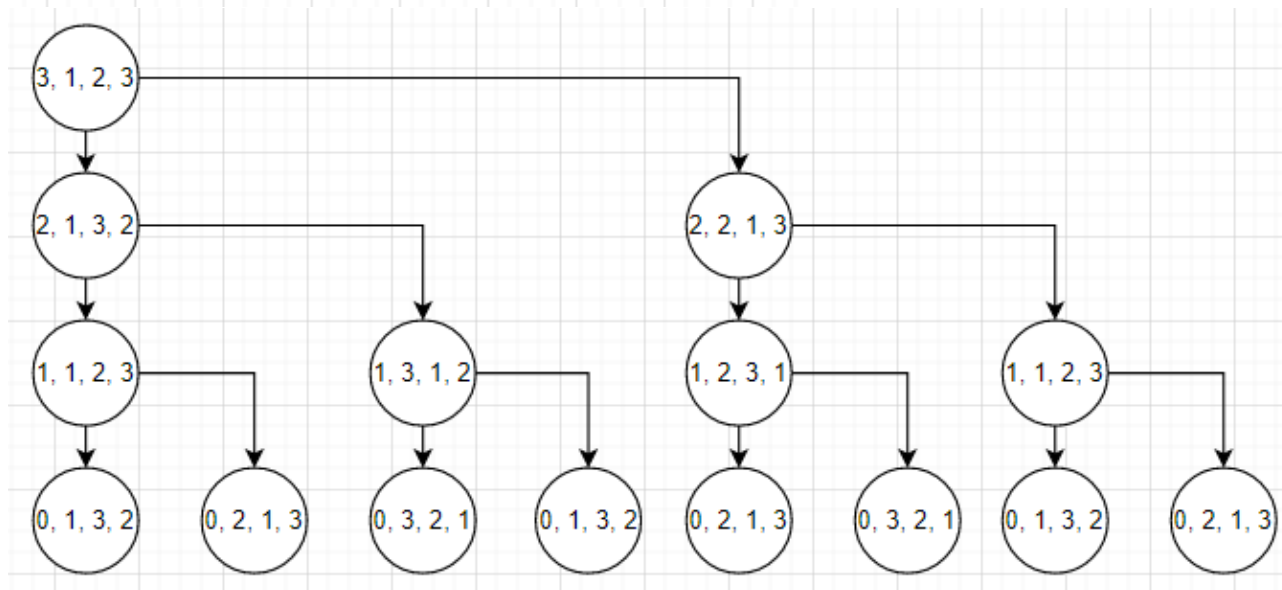
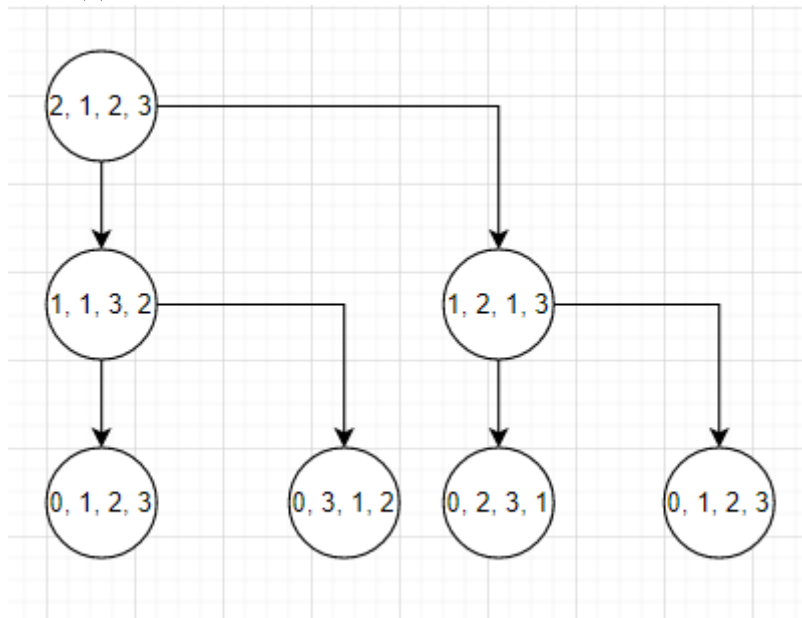
Рекуррентная зависимость описана в разделе 3 задания 1 “Описание алгоритма – рекуррентная зависимость”.

5. Реализуйте и отладьте рекурсивную функцию решения задачи

Рекурсивная функция реализована и отлажена в разделе 6 задания 1 “Код программы и скриншоты результатов тестирования”

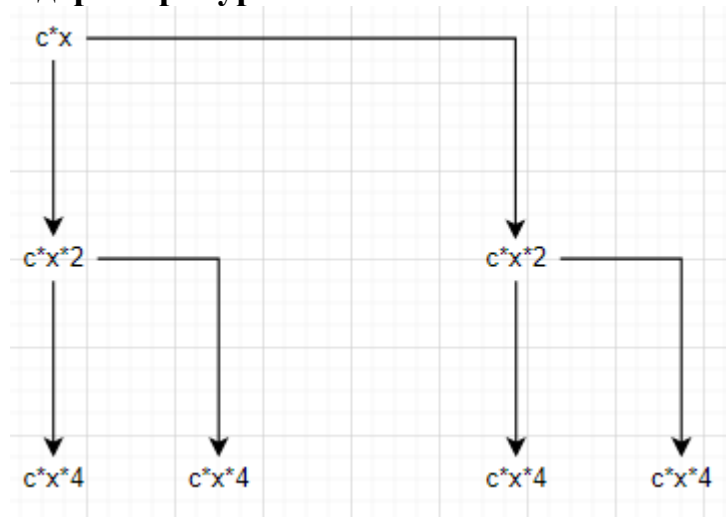
6. Определите глубину рекурсии, изменяя исходные данные

Из рисунков видно, что глубина рекурсии определяется как $n + 1$, n – количество дисков.



7. Определите сложность рекурсивного алгоритма, используя

метод подстановки и дерево рекурсии



Глубина рекурсии (из задачи 6) равна $n + 1$;

Предположим, что значение каждого из узлов на уровне x : $T(x \cdot 2^{(n+1)})$

Время выполнения на каждом из уровней: $c \cdot x$

Просуммируем время на каждом из уровней и время всех уровней:

$$S = cx + 2cx + 2cx + 4cx + 4cx + 4cx + 4cx + \dots 2^{(n+1)}cx = cx (1 + 4 + 16 + \dots 2^{(n+1)}) =$$

$$2^{(n+1)} = [\text{воспользуемся формулой суммы геометрической прогрессии}] =$$

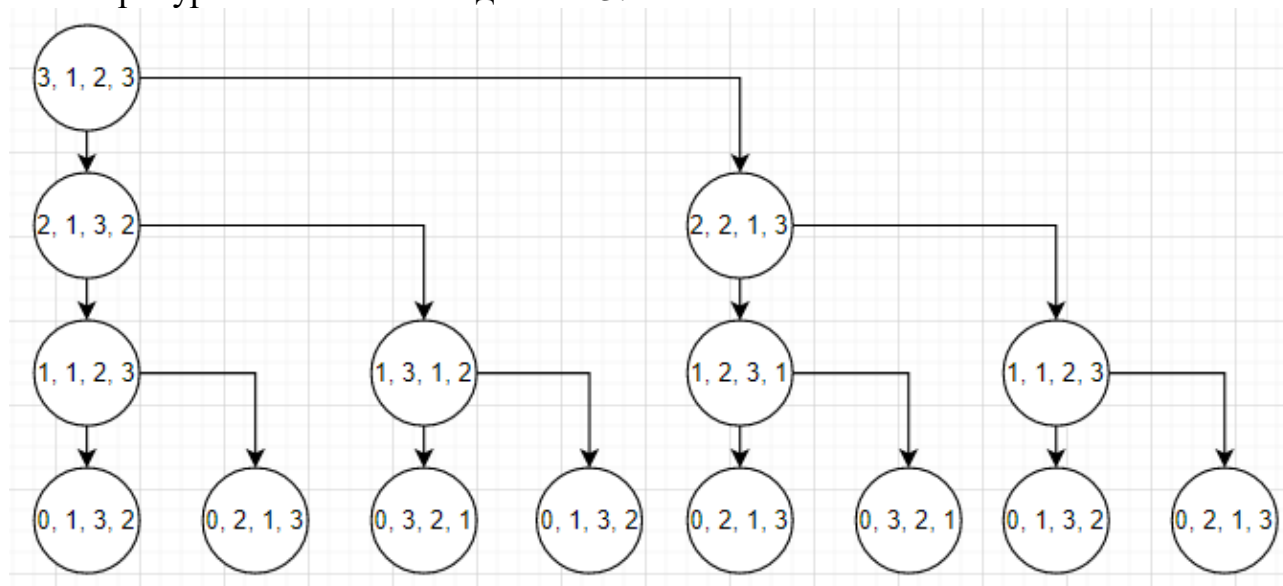
$$cx \cdot (1 \cdot (1 - 2^{(n+1)}) / (1 - 2)) = (2^{(n+1)} - 1) \cdot cx$$

Таким образом, сложность написанного рекурсивного алгоритма:

$$O(n) = 2^{(n+1)} - 1$$

8. Приведите для одного из значений схему рекурсивных вызовов

Схема рекурсивных вызовов для $n = 3$:



9. Разработайте программу демонстрирующую выполнение обеих функций и покажите результаты тестирования

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

struct Condition
{
    int dNumber, from, free, to, position;
};

void hanoi(int number, int from, int free, int to)
{
    if (number != 0)
    {
        hanoi(number-1, from, to, free);
        cout << "Передвигаем " << number << "-й диск с " << from << "-го
стержня на " << to << "-ий" << endl;
        hanoi(number-1, free, from, to);
    }
}

Condition* setNewTask(int n, int from, int free, int to, int position)
{
    Condition* cond = new Condition;
    cond->dNumber = n;
    cond->from = from;
    cond->free = free;
    cond->to = to;
    cond->position = position;
    return cond;
}

void hanoi_iteration(int n)
{
    vector<Condition*> stack;
    stack.push_back(setNewTask(n, 1, 2, 3, 0));
    while (stack.size() > 0)
    {
        Condition *cond = stack.back();
        switch (cond->position)
        {
            case 0:
```

```

        {
            if (cond->dNumber == 0)
            {
                stack.pop_back();
            }
            else
            {
                cond->position++;
                stack.push_back(setNewTask(cond-
>dNumber - 1, cond->from, cond->to, cond->free, 0));
            }
            break;
        }
        case 1:
        {
            cond->position++;
            cout << "Передвигаем " << cond->dNumber
<< "-й диск с " << cond->from << "-го стержня на " << cond->to << "-ий" << endl;
            stack.push_back(setNewTask(cond->dNumber
- 1, cond->free, cond->from, cond->to, 0));
            break;
        }
        case 2:
        {
            stack.pop_back();
            break;
        }
    }
}
}
int main()

```

```

{
    setlocale(LC_ALL, "Russian");
    int n;
    cout << "Количество дисков n: ";
    cin >> n;
    cout << "===Рекурсивная функция===" << endl;
    hanoi(n, 1, 2, 3);
    cout << "===Итерационная функция===" << endl;
    hanoi_iteration(n);
}

```

Результаты тестирования (одинаковы для двух функций):

```
Количество дисков n: 3
===Рекурсивная функция===
Передвигаем 1-й диск с 1-го стержня на 3-ий
Передвигаем 2-й диск с 1-го стержня на 2-ий
Передвигаем 1-й диск с 3-го стержня на 2-ий
Передвигаем 3-й диск с 1-го стержня на 3-ий
Передвигаем 1-й диск с 2-го стержня на 1-ий
Передвигаем 2-й диск с 2-го стержня на 3-ий
Передвигаем 1-й диск с 1-го стержня на 3-ий
===Итерационная функция===
Передвигаем 1-й диск с 1-го стержня на 3-ий
Передвигаем 2-й диск с 1-го стержня на 2-ий
Передвигаем 1-й диск с 3-го стержня на 2-ий
Передвигаем 3-й диск с 1-го стержня на 3-ий
Передвигаем 1-й диск с 2-го стержня на 1-ий
Передвигаем 2-й диск с 2-го стержня на 3-ий
Передвигаем 1-й диск с 1-го стержня на 3-ий
-----
Process exited with return value 0
Press any key to continue . . .
```

```

Количество дисков n: 4
===Рекурсивная функция===
Передвигаем 1-й диск с 1-го стержня на 2-ий
Передвигаем 2-й диск с 1-го стержня на 3-ий
Передвигаем 1-й диск с 2-го стержня на 3-ий
Передвигаем 3-й диск с 1-го стержня на 2-ий
Передвигаем 1-й диск с 3-го стержня на 1-ий
Передвигаем 2-й диск с 3-го стержня на 2-ий
Передвигаем 1-й диск с 1-го стержня на 2-ий
Передвигаем 4-й диск с 1-го стержня на 3-ий
Передвигаем 1-й диск с 2-го стержня на 3-ий
Передвигаем 2-й диск с 2-го стержня на 1-ий
Передвигаем 1-й диск с 3-го стержня на 1-ий
Передвигаем 3-й диск с 2-го стержня на 3-ий
Передвигаем 1-й диск с 1-го стержня на 2-ий
Передвигаем 2-й диск с 1-го стержня на 3-ий
Передвигаем 1-й диск с 2-го стержня на 3-ий
===Итерационная функция===
Передвигаем 1-й диск с 1-го стержня на 2-ий
Передвигаем 2-й диск с 1-го стержня на 3-ий
Передвигаем 1-й диск с 2-го стержня на 3-ий
Передвигаем 3-й диск с 1-го стержня на 2-ий
Передвигаем 1-й диск с 3-го стержня на 1-ий
Передвигаем 2-й диск с 3-го стержня на 2-ий
Передвигаем 1-й диск с 1-го стержня на 2-ий
Передвигаем 4-й диск с 1-го стержня на 3-ий
Передвигаем 1-й диск с 2-го стержня на 3-ий
Передвигаем 2-й диск с 2-го стержня на 1-ий
Передвигаем 1-й диск с 3-го стержня на 1-ий
Передвигаем 3-й диск с 2-го стержня на 3-ий
Передвигаем 1-й диск с 1-го стержня на 2-ий
Передвигаем 2-й диск с 1-го стержня на 3-ий
Передвигаем 1-й диск с 2-го стержня на 3-ий

-----
Process exited with return value 0
Press any key to continue . . .

```

6. Код программы и скриншоты результатов тестирования

```

#include <iostream>
using namespace std;

void hanoi(int number, int from, int free, int to)
{
    if (number != 0)
    {
        hanoi(number-1, from, to, free);
        cout << "Передвигаем " << number << "-й диск с " << from << "-го
стержня на " << to << "-ий" << endl;
        hanoi(number-1, free, from, to);
    }
}

```

```

    }
}
int main()
{
    setlocale(LC_ALL, "Russian");
    int n;
    cout << "Количество дисков n: ";
    cin >> n;
    cout << endl;
    hanoi(n, 1, 2, 3);
}

```

Результаты тестирования:

```

Количество дисков n: 1
Передвигаем 1-й диск с 1-го стержня на 3-ий
-----
Process exited with return value 0
Press any key to continue . . .

```

```

Количество дисков n: 2
Передвигаем 1-й диск с 1-го стержня на 2-ий
Передвигаем 2-й диск с 1-го стержня на 3-ий
Передвигаем 1-й диск с 2-го стержня на 3-ий
-----
Process exited with return value 0
Press any key to continue . . .

```

```

Количество дисков n: 3
Передвигаем 1-й диск с 1-го стержня на 3-ий
Передвигаем 2-й диск с 1-го стержня на 2-ий
Передвигаем 1-й диск с 3-го стержня на 2-ий
Передвигаем 3-й диск с 1-го стержня на 3-ий
Передвигаем 1-й диск с 2-го стержня на 1-ий
Передвигаем 2-й диск с 2-го стержня на 3-ий
Передвигаем 1-й диск с 1-го стержня на 3-ий
-----
Process exited with return value 0
Press any key to continue . . .

```



```
Количество дисков n: 4
Передвигаем 1-й диск с 1-го стержня на 2-ий
Передвигаем 2-й диск с 1-го стержня на 3-ий
Передвигаем 1-й диск с 2-го стержня на 3-ий
Передвигаем 3-й диск с 1-го стержня на 2-ий
Передвигаем 1-й диск с 3-го стержня на 1-ий
Передвигаем 2-й диск с 3-го стержня на 2-ий
Передвигаем 1-й диск с 1-го стержня на 2-ий
Передвигаем 4-й диск с 1-го стержня на 3-ий
Передвигаем 1-й диск с 2-го стержня на 3-ий
Передвигаем 2-й диск с 2-го стержня на 1-ий
Передвигаем 1-й диск с 3-го стержня на 1-ий
Передвигаем 3-й диск с 2-го стержня на 3-ий
Передвигаем 1-й диск с 1-го стержня на 2-ий
Передвигаем 2-й диск с 1-го стержня на 3-ий
Передвигаем 1-й диск с 2-го стержня на 3-ий

-----
Process exited with return value 0
Press any key to continue . . .
```

Отчет по задаче 2

1. Условие задачи

Удалить рекурсивно однонаправленный список.

2. Постановка задачи

Реализовать рекурсивную функцию, которая должна удалять линейный список.

3. Описание алгоритма – рекуррентная зависимость

$$\text{deleteList}(n) = \begin{cases} \text{deleteList}(n \rightarrow \text{next}), & n \neq 0 \\ \text{return}, & n == 0 \end{cases}$$

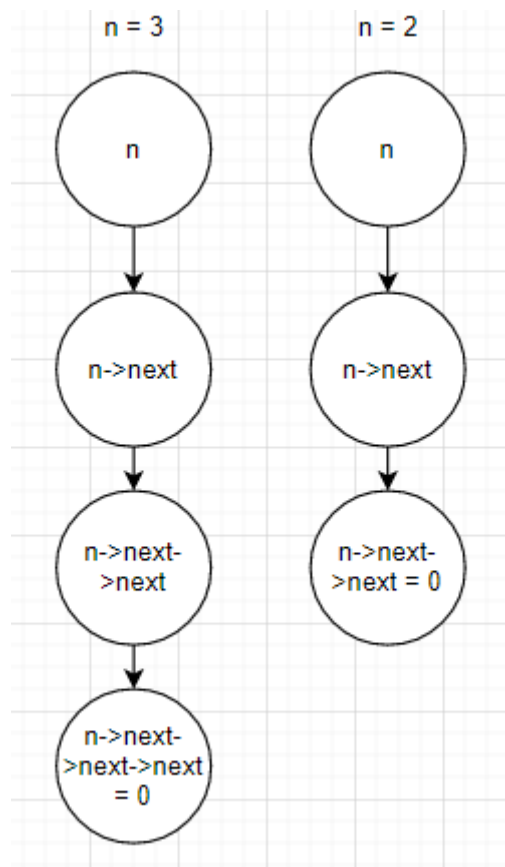
4. Коды используемых функций

```
void deleteList(Node* &N)
{
    if (N != 0)
    {
        deleteList(N->next);
        cout << endl << "I'm going to delete the element of the list with
number: " << N->a;
        delete N;
    }
    else
    {
        return;
    }
}
```

5. Ответы на задания по задаче 2: список требований к задаче 2

1. Определите глубину рекурсии

Глубина рекурсии: $n + 1$



2. Определите теоретическую сложность алгоритма

Оператор	Время выполнения инструкции	Количество выполнений оператора
if (N != 0)	c1	n + 1
{		
deleteList(N->next);	c2	n
cout << endl << "I will delete the element of the list with number: " << N->a;	c3	n
delete N;	c4	n
}		
else		
{		
return;	c5	1
}		

$$T(n) = c1*(n+1) + c2*n + c3*n + c4*n + c5 = n*(c1 + c2 + c3 + 4) + (c1 + c5) \Rightarrow O(n) = n$$

3. Разработайте программу, демонстрирующую работу функций и покажите результаты тестов

Разработаны в пункте 6.

6. Код программы и скриншоты результатов тестирования

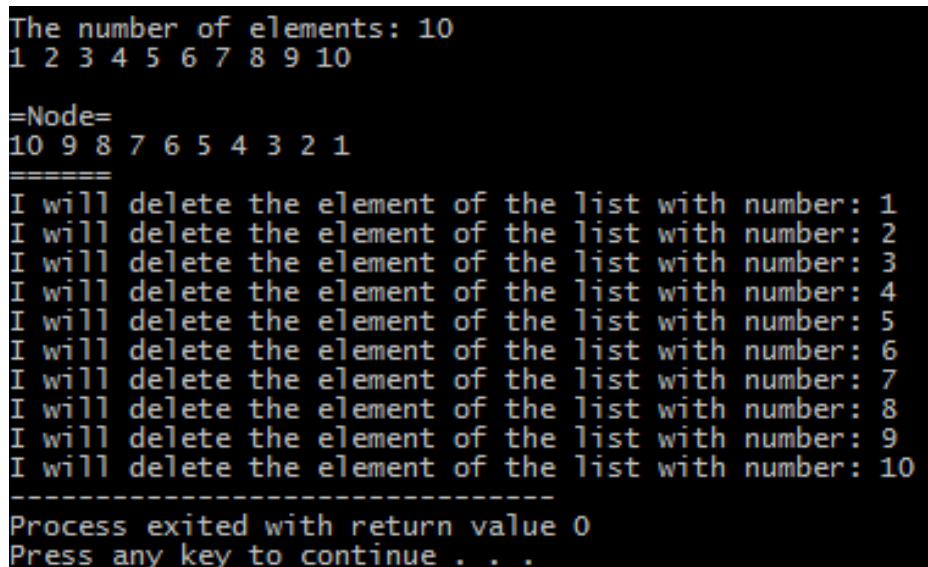
```
#include <iostream>
using namespace std;
struct Node
{
    int a;
    Node* next;
};
Node* insertNode(Node* N, int a)
{
    Node *newNode = new Node;
    newNode->a = a;
    newNode->next = N;
    return newNode;
}
void printNode(Node* N)
{
    cout << endl << "=Node=" << endl;
    while (N != 0)
    {
        cout << N->a << ' ';
        N = N->next;
    }
    cout << endl << "=====";
}
void createNewNode(Node* &N)
{
    int n, a;
    cout << endl << "The number of elements: ";
    cin >> n;
    for (int i = 0; i < n; i++)
    {
        cin >> a;
        N = insertNode(N, a);
    }
}
void deleteList(Node* &N)
{
    if (N != 0)
    {
        deleteList(N->next);
        cout << endl << "I'm going to delete the element of the list with
```

```

number: " << N->a;
        delete N;
    }
    else
    {
        return;
    }
}
int main()
{
    Node* N = 0;
    createNewNode(N);
    printNode(N);
    deleteList(N);
}

```

Результат тестирования:



```

The number of elements: 10
1 2 3 4 5 6 7 8 9 10

=Node=
10 9 8 7 6 5 4 3 2 1
=====
I will delete the element of the list with number: 1
I will delete the element of the list with number: 2
I will delete the element of the list with number: 3
I will delete the element of the list with number: 4
I will delete the element of the list with number: 5
I will delete the element of the list with number: 6
I will delete the element of the list with number: 7
I will delete the element of the list with number: 8
I will delete the element of the list with number: 9
I will delete the element of the list with number: 10
-----
Process exited with return value 0
Press any key to continue . . .

```

Выводы

В ходе практической работы я получила навыки работы с рекурсивными функциями, выполнила реализацию 2-х задач своего варианта с помощью рекурсивных функций, оценила их сложность.

Информационные источники

1. Лекции по Структурам и алгоритмам обработки данных Скворцова Л. А. – МИРЭА – Российский технологический университет 2021 год
2. Искусство программирования Т. 1 / Д. Кнут – М.: Вильямс, 2014. – 712 с.
3. draw.io [Электронный ресурс]. (URL: <https://app.diagrams.net>)
Последнее обращение 26.05.2021