



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 № 1

по дисциплине «Структуры и алгоритмы обработки данных»

Тема: «Оценка сложности и определение эффективности алгоритма»

Выполнил:
Студент группы ИКБО-29-20

Хан Анастасия Александровна

Проверил:

Копылова А.В.

Москва 2021

СОДЕРЖАНИЕ

Отчет по заданию 1

Постановка задачи 1.....	3
Модель решения поставленной задачи	3
Реализация алгоритма в виде функции и отладка.	18
Реализация заполнения массива случайными числами, вывода массива	31
Результаты тестирования.....	31

Отчет по заданию 2

Постановка задачи 2.....	3
Модель решения поставленной задачи	3
Реализация алгоритма в виде функции и отладка.	18
Реализация заполнения массива случайными числами, вывода массива	31
Результаты тестирования.....	31
Выводы	32
Информационные источники	32

Отчет по заданию 1

1. Постановка задачи 1

x-массив, n – количество элементов в массиве, key – удаляемое значение	
Алгоритм 1	Алгоритм 2
<pre>delFirstMetod(x,n,key){ i←1 while (i<=n) do if x[i]=key then //удаление for j←i to n-1 do x[j] ←x[j+1] od n←n-1 else i←i+1 endif od }</pre>	<pre>delOtherMetod(x,n,key){ j←1 for i←1 to n do x[j]=x[i]; if x[i]!=key then j++ endif od n←j }</pre>

Определить эффективный алгоритм из двух предложенных, используя оценку теоретической сложности каждого из алгоритмов и емкостную сложность решения следующей задачи: дан массив из n элементов целого типа, удалить из массива все значения равные заданному.

2. Модель решения поставленной задачи

Описание выполнения алгоритма 1

1. В функцию передается массив x из n элементов и значение key , элементы массива равные key удаляются
2. Переменной i присваивается значение 1
3. Далее в цикле `while` с условием $i \leq n$, то есть пока номер i меньше количества элементов массива, выполняется:
 - 3.1. Если значение текущего элемента совпадает со значением key :
 - 3.1.1. В цикле `for` от текущего элемента i и до предпоследнего элемента массива происходит присвоение текущему элементу следующего
 - 3.1.2. Переменная n – количество элементов массива уменьшается на 1.
 - 3.2. Иначе значение i увеличивается на 1.

Определение инварианта для внешнего цикла – доказательство корректности цикла для алгоритма 1.

С каждым шагом алгоритма область неопределенности уменьшается – мы либо переходим к следующему элементу, пока не дойдем до n элемента, либо «удаляем» элемент, смещая весь массив, начиная с этого элемента, на 1, соответственно уменьшая n .

Инвариант - $1 \leq I \leq n$

Инвариант корректен, т.к. i перед началом цикла равно 1

Значение i увеличивается после каждого раза, когда элемент не равен key (иначе уменьшается n)

После завершения цикла значение I будет равно n , что значит, что цикл прошелся по всем элементам массива.

Определение вычислительной сложности алгоритма 1 с помощью теоретического подхода

Оператор	Количество выполнений оператора
$i \leftarrow 1$	1
while ($i \leq n$) do	$n + 1$
if $x[i] = key$ then	n
for $j \leftarrow i$ to $n-1$ do	$n * n$
$x[j] \leftarrow x[j+1]$	$n * n$
od	
$n \leftarrow n-1$	n
else	
$i \leftarrow i+1$	n
endif	
od	

Таким образом, $T_1(n) = 1 + n + 1 + n + n^2 + n^2 + n + n = 2n^2 + 4n + 2$

Описание выполнения алгоритма 2

1. В функцию передается массив x из n элементов и значение key , элементы массива равные key удаляются

2. Переменной j присваивается значение 1

3. В цикле for от элемента $i = 1$ и до последнего элемента массива происходит:

3.1. Присваивается элементу массива $x[j]$ значение $x[i]$

3.2. Если значение текущего элемента не равно key :

3.2.1. Значение j увеличивается на 1

4. Переменной n присваивается значение переменной j .

Определение инварианта для внешнего цикла – доказательство корректности цикла для алгоритма 2.

С каждым шагом алгоритма область неопределенности уменьшается – мы либо переходим к следующему элементу, пока не дойдем до n элемента,

либо «удаляем» элемент, смещая весь массив, начиная с этого элемент, на 1, соответственно уменьшая n.

Инвариант - $1 \leq i \leq n$

Инвариант корректен, т.к. i перед началом цикла инициализируется со значением 1

Значение i увеличивается после каждой итерации

После завершения цикла значение i будет равно n, что значит, что цикл прошелся по всем элементам массива.

Определение вычислительной сложности алгоритма 2 с помощью теоритического подхода

Оператор	Количество выполнений оператора
$j \leftarrow 1$	1
for $i \leftarrow 1$ to n do	$n + 1$
$x[j] = x[i];$	n
if $x[i] \neq \text{key}$ then	n
$j++$	n
endif	
od	
$n \leftarrow j$	1

Таким образом, $T_2(n) = 1 + n + 1 + n + n + n + 1 = 4n + 3$

3.Реализация алгоритма в виде функции и отладка.

```
int delFirstMetod(int* a, int n, int key)
```

```
{
    int i = 0;
    while (i < n)
    {
        if (a[i] == key)
        {
            for (int j = i; j < n - 1; j++)
            {
                a[j] = a[j + 1];
            }
            n--;
        }
        else
        {
            i++;
        }
    }
    return n;
}
```

```
int delOtherMetod(int* a, int n, int key)
```

```

{
    int j = 0;
    for (int i = 0; i < n; i++)
    {
        a[j] = a[i];
        if (a[i] != key)
        {
            j++;
        }
    }
    n = j;
    return n;
}

```

4. Реализация заполнения массива случайными числами, вывода массива.

```

void input(int* a, int n)
{
    for (int i = 0; i < n; i++)
    {
        a[i] = rand()%100;
    }
}

void output(int* a, int n)
{
    cout << endl;
    for (int i = 0; i < n; i++)
    {
        cout << a[i] << ' ';
    }
}

```

5. Результаты тестирования

Чтобы было видно различие между алгоритмом 1 и 2, протестируем на данных при $n = 10000$ и $n = 100000$

Алгоритм 1 при $n = 10000$

```
10000  
  
33  
  
time: 2003000 nanos  
-----  
Process exited with return value 0  
Press any key to continue . . . _
```

Алгоритм 1 при $n = 100000$

```
100000  
  
33  
  
time: 258174000 nanos  
-----  
Process exited with return value 0  
Press any key to continue . . . _
```

Алгоритм 2 при $n = 10000$

```
10000  
  
33  
  
time: 0 nanos  
-----  
Process exited with return value 0  
Press any key to continue . . .
```

Алгоритм 2 при $n = 100000$

```
1000000
33
time: 0 nanos
-----
Process exited with return value 0
Press any key to continue . . .
```

Вывод: алгоритм 2 быстрее, в этом мы убедились при оценке теоритической сложности и по результатам тестирования.

Отчет по заданию 2

1. Постановка задачи

Коэффициенты системы линейных уравнений заданы в виде прямоугольной матрицы размером $n \times n$. С помощью допустимых преобразований привести систему к треугольному виду (коэффициенты должны быть только над главной диагональю). Примечание. Система состоит из n уравнений с n неизвестными. Матрица имеет размер $n \times (n+1)$. Т.е. i -ая строка матрицы хранит коэффициенты i -ого уравнения и свободный член.

2. Модель решения поставленной задачи

Описание выполнения алгоритма

1. На вход алгоритму подается матрица, заполненная случайными числами, размером $n \times (n + 1)$.
2. Инициализируется переменная $I = 0$;
3. В цикле `for` с переменной-счетчиком j от нулевого и до последнего элемента (это проход по строкам) выполняется:
 - 3.1. Инициализируется переменная $noNullEl = -1$, которая в последствии будет либо хранить номер ненулевого элемента столбца, либо иметь значение -1 (например, первая строка может попасться с нулевым первым элементом, и занулить нижестоящие в столбце элементы не получится; для решения проблемы надо поменять строку с нулевым элементом на первую строку, в которой первый элемент не равен 0).
 - 3.2. В цикле `for` с счетчиком $k = i$ идет проход по столбцу, в котором нужно найти первый ненулевой элемент, если элемент найден, то значению $noNullEl$ присваивается номер такого элемента и данный цикл прекращается.
 - 3.3. Проверка: Если $noNullEl = -1$, то выполняется команда `continue`
 - 3.4. Проверка: Если номер первого ненулевого элемента $noNullEl$ не совпадает с текущим номером элемента i , то нужно поменять эти строки:
 - 3.4.1. В цикле `for` с счетчиком $k = j$ до $n + 1$ выполняется перестановка соответственных элементов строки с ненулевым первым элементом и текущей строки.
 - 3.5. Инициализируется переменная вещественного типа $coef$;
 - 3.6. В цикле `for` с переменной $k = I + 1$ до n (проход по столбцам) выполняется:
 - 3.6.1. Рассчитывается коэффициент, на который будет умножаться вычитаемая строка: $coef = a[k][j] / a[i][j]$ (числитель – первый элемент уменьшаемой строки, знаменатель – первый элемент вычитаемой строки;

3.6.2. В цикле for с переменной-счетчиком $l = j$ до $n + 1$ выполняется вычитание из текущей строки первой текущей строки с номером строки I , умноженной на $coef$: $a[k][l] -= coef * a[i][l]$.

3.7. Значение переменной I увеличивается на 1.

Определение инвариантов для циклов – доказательство корректности алгоритма

Для внешнего цикла: инвариант $0 \leq j < n + 1$

Он корректен, т.к. перед первой итерацией j инициализируется со значением 0;

С каждой итерацией j увеличивается; после выполнения алгоритма j будет равно $n + 1$, что значит, что $j \leq n$

Для внутреннего цикла, в котором идет поиск ненулевого элемента: инвариант $i \leq k < n$

Он корректен, т.к. перед первой итерацией k инициализируется со значением i , которое в свою очередь инициализировалось до внешнего цикла;

С каждой итерацией k увеличивается; после выполнения алгоритма k будет равно n , что значит, что k принимало все значения от 0 до n , ненулевого элемента нет, либо $i \leq k < n$, что значит, что в цикле был найден ненулевой элемент и цикл завершился.

Для внутреннего цикла, в котором идет обмен элементами строк: инвариант $j \leq k < n + 1$

Он корректен, т.к. перед первой итерацией k инициализируется со значением j , которое в свою очередь инициализировалось в начале внешнего цикла;

С каждой итерацией k увеличивается; после выполнения алгоритма k будет равно $n + 1$, что значит, что k принимало все значения от j до $n + 1$.

Для внутреннего цикла, в котором происходит вычитание строк: инвариант $I + 1 \leq k < n$

Он корректен, т.к. перед первой итерацией k инициализируется со значением $I + 1$, которое в свою очередь инициализировалось в начале внешнего цикла;

С каждой итерацией k увеличивается; после выполнения алгоритма k будет равно n , что значит, что k принимало все значения от $I + 1$ до n (прошелся цикл по всем строкам, начиная с $I + 1$).

Для внутреннего цикла, который находится внутри предыдущего цикла: инвариант $j \leq l < n + 1$

Он корректен, т.к. перед первой итерацией l инициализируется со значением j , которое в свою очередь инициализировалось в начале внешнего

цикла;

С каждой итерацией l увеличивается; после выполнения алгоритма l будет равно $n + 1$, что значит, что l принимало все значения от j до $n + 1$ (цикл прошелся по всем столбцам).

Определение вычислительной сложности алгоритма 2 с помощью теоретического подхода

Оператор	Количество выполнений оператора
$i \leftarrow 0$	1
for $j \leftarrow 0$ to n do	$n+1$
noNullEl $\leftarrow -1$	n
for $k \leftarrow i$ to n do	$(1 + n)/2 * n = n/2 + n^2/2$
if $a[k][j] \neq 0$ then	$n/2 + n^2/2$
noNullEl = k	$n/2 + n^2/2$
break	$n/2 + n^2/2$
endif	
od	
if noNullEl == -1 then	n
continue	0
endif	
if noNullEl != i then	n
for $k \leftarrow j$ to $n + 1$ do	$(2 + n)/2 * (n+1) = n^2/2 + 3*n/2 + 1$
buff $\leftarrow a[i][k]$	$n^2/2 + 3*n/2 + 1$
$a[i][k] \leftarrow a[\text{noNullEl}][k]$	$n^2/2 + 3*n/2 + 1$
$a[\text{noNullEl}][k] \leftarrow \text{buff}$	$n^2/2 + 3*n/2 + 1$
od	
endif	
for $k \leftarrow i + 1$ to n do	$n/2 + n^2/2 - 1$
koef $\leftarrow a[k][j] / a[i][j]$	$n/2 + n^2/2 - 1$
for $l \leftarrow j$ to $n + 1$ do	$(n/2 + n^2/2 - 1) * (n^2/2 + 3*n/2 + 1) = 4n^4 + 16n^3 + 8n^2 - 20n - 8$
$a[k][l] \leftarrow \text{koef} * a[i][l];$	$4n^4 + 16n^3 + 8n^2 - 20n - 8$
od	
od	
$i++$	n
od	

Теоретическая сложность: $T(n) = 8n^4 + 16n^3 + 8n^2 - 20n - 8$

3. Реализация алгоритма в виде функции и отладка

Алгоритм в виде функции:

```
void makeTriangle(int n, vector<vector<float>> &a)
```

```
{
    int i = 0;

    for (int j = 0; j < n + 1; j++)
    {
        int noNullEl = - 1;
        for (int k = i; k < n; k++)
        {
            if (a[k][j] != 0)
            {
                noNullEl = k;
                break;
            }
        }
        if (noNullEl == -1)
        {
            continue;
        }
        if (i != noNullEl)
        {
            for (int k = j; k < n + 1; k++)
            {
                int buff = a[i][k];
                a[i][k] = a[noNullEl][k];
                a[noNullEl][k] = buff;
            }
        }
        float koef;
        for (int k = i + 1; k < n; k++)
        {
            koef = a[k][j] / a[i][j];
            for (int l = j; l < n + 1; l++)
            {
                a[k][l] -= koef * a[i][l];
            }
        }
    }
}
```

```

        i++;
    }
}

```

Тесты для makeTriangle(int n, vector<vector<float> > &a)			
Но ме р те ст а	Входные данные	Ожидаемый результат (при выводе)	Результат программы (при выводе)
1	N = 4 1 1 2 3 1 1 2 3 -1 -4 3 -1 -1 -2 -4 2 3 -1 -1 -6	1.00 1.00 2.00 3.00 1.00 0.00 1.00 1.00 -4.00 -5.00 0.00 0.00 -3.00 -27.00 -27.00 0.00 0.00 0.00 51.00 51.00	1.00 1.00 2.00 3.00 1.00 0.00 1.00 1.00 -4.00 - 5.00 0.00 0.00 -3.00 -27.00 -27.00 0.00 0.00 0.00 51.00 51.00

4. Реализация заполнения массива случайными числами, вывода массива

```

void fillArr(int n, vector<vector<float> > &vectorColumn)
{
    for (int i = 0; i < n; i++)

    {

        vector <float> vectorLine;

        for (int j = 0; j < n + 1; j++)
        {

            vectorLine.push_back(rand()%10 * 1.0);

        }
        vectorColumn.push_back(vectorLine);
    }
}

```

```

void outArr(int n, vector<vector<float> > &a)
{
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n + 1; j++)
        {
            printf("%7.2f", a[i][j]);

        }
        cout << endl;
    }
}

```

5. Результаты тестирования

Код всей программы:

```

#include <iostream>
#include <vector>
#include <cstdlib>
#include <chrono>
using namespace std;
using namespace chrono;
void fillArr(int n, vector<vector<float> > &vectorColumn)
{
    for (int i = 0; i < n; i++)
    {
        vector <float> vectorLine;

        for (int j = 0; j < n + 1; j++)
        {
            vectorLine.push_back(rand()%10 * 1.0);

        }
        vectorColumn.push_back(vectorLine);
    }
}

```

```

void outArr(int n, vector<vector<float> > &a)
{
    for (int i = 0; i < n; i++)

        {

            for (int j = 0; j < n + 1; j++)
            {

                printf("%7.2f", a[i][j]);

            }
            cout << endl;
        }
}

```

```

void makeTriangle(int n, vector<vector<float> > &a)
{
    int i = 0;

    for (int j = 0; j < n + 1; j++)
    {
        int noNullEl = - 1;
        for (int k = i; k < n; k++)
        {
            if (a[k][j] != 0)
            {
                noNullEl = k;
                break;
            }
        }
        if (noNullEl == -1)
        {
            continue;
        }
        if (i != noNullEl)
        {
            for (int k = j; k < n + 1; k++)
            {
                int buff = a[i][k];
                a[i][k] = a[noNullEl][k];
                a[noNullEl][k] = buff;
            }
        }
    }
}

```

```

        }
    }
    float koef;
    for (int k = i + 1; k < n; k++)
    {
        koef = a[k][j] / a[i][j];
        for (int l = j; l < n + 1; l++)
        {
            a[k][l] -= koef * a[i][l];
        }
    }
    i++;
}

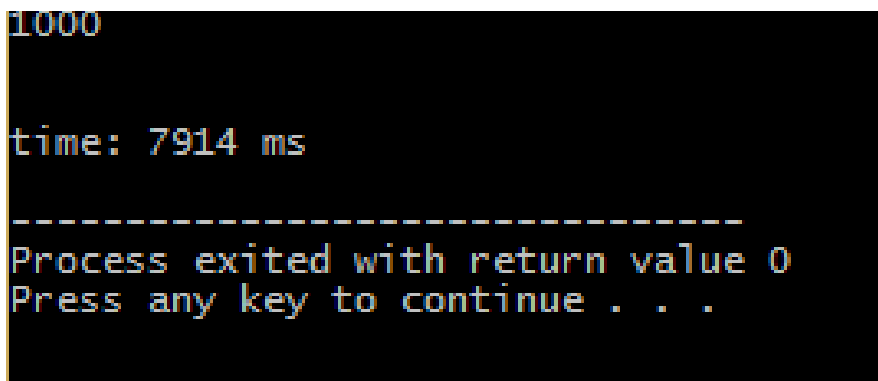
int main()
{
    srand(time(0));
    vector<vector<float>> arr;
    int n;
    cin >> n;
    fillArr(n, arr);
    cout << endl;
    auto begin = high_resolution_clock::now();
    makeTriangle(n, arr);
    auto end = high_resolution_clock::now();
    auto dur = duration_cast<milliseconds>(end - begin);
    cout << endl << "time: " << dur.count() << " ms" << endl;
}

```

Результаты тестирования:

Результат программы при n = 1000 (в матрице будет 1001000 элементов)

Лучший случай:



```

1000

time: 7914 ms

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

```

Худший случай:


```
1000

time: 11112 ms

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

Выводы

В ходе практической работы я научилась определять сложность алгоритмы на теоретическом уровне, замерять время выполнения работы алгоритма, оценивать сложности алгоритмов и выбирать наиболее оптимальный.

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) <https://pro-prof.com/archives/4275>