1. Министерство образования и науки Российской Федерации
2. Санкт-Петербургский Политехнический Университет Петра Великого
3. —
4. Институт кибербезопасности и защиты информации

**ЛАБОРАТОРНАЯ РАБОТА № 3**

1. «**Нахождение n-нного элемента последовательности**»
2. по дисциплине «Структуры данных»
3. Выполнил
4. студент гр.5151001/40001 Волошкевич М.А.

<*подпись*>

1. Преподаватель Семьянов П.В.

<*подпись*>

1. Санкт-Петербург
2. 2025 г.
3. **Цель работы**

Реализовать алгоритм для нахождения энного элемента в последовательности, состоящей из чисел, которые являются произведением степеней чисел 3, 5 и 7. Проверить корректность работы алгоритма и его эффективность.

**2. Постановка задачи**

Написать программу, которая для заданного числа n находит n-й элемент последовательности, где каждый элемент является произведением степеней чисел 3, 5 и 7. Последовательность должна быть упорядочена по возрастанию. Алгоритм должен быть реализован с использованием динамического массива для хранения промежуточных результатов.

**3. Теоретические исследования**

Для выполнения работы необходимо было изучить алгоритмы генерации последовательностей, а также методы динамического управления памятью для хранения промежуточных результатов. Основные теоретические аспекты:

* Последовательность формируется из чисел вида 3^a×5^b×7^c, где a, b, c — неотрицательные целые числа.
* Для эффективного нахождения следующего элемента используются указатели на текущие минимальные кандидаты для умножения на 3, 5 и 7.
* Динамический массив используется для хранения элементов последовательности с возможностью его компактификации для оптимизации использования памяти.

**4. Описание решения**

Алгоритм реализован на языке C и состоит из следующих этапов:

1. **Инициализация динамического массива**: Начальный массив содержит один элемент — 1.
2. **Генерация последовательности**:
   * Для каждого следующего элемента выбирается минимальное значение среди произведений текущих элементов на 3, 5 и 7.
   * Указатели для умножения на 3, 5 и 7 обновляются в зависимости от выбранного минимального значения.
3. **Сокращение рамзера массива**: Удаление уже неиспользуемых элементов для экономии памяти.
4. **Вывод результата**: Возвращается n-й элемент последовательности.

**Ключевые параметры:**

* Используется динамический массив для хранения элементов.
* Указатели i3, i5, i7 отслеживают текущие позиции для умножения на 3, 5 и 7 соответственно.
* Когда минимальное из указателей i3, i5 и i7 число отличается от предыдущего наименьшего из указателей, то массив удаляется до минимального индекса.

**5. Тестирование и результаты работы программы**

При вводе различных значений получались правильные варианты ответов, соответствующих базовому критерию деления на 3, 5 и 7. При вводе экстремальных значений, при которых n-нный элемент последовательности близок к максимальному размеру типа данных в котором он хранится, так же не наблюдается отклонений. Можно сделать вывод, что программа работает правильно.

**6. Вывод**

В ходе работы был успешно реализован алгоритм для нахождения n-го элемента в последовательности чисел, образованных произведениями степеней 3, 5 и 7. Алгоритм демонстрирует корректность работы и эффективность благодаря использованию динамического массива и компактификации. Для дальнейшего улучшения можно рассмотреть возможность параллелизации вычислений или оптимизации выбора минимального элемента.

**7. Приложения**

**Приложение1:**

#include <stdio.h>

#include <stdlib.h>

#define ll unsigned long long

typedef struct {

ll \*data;

int size;

} dynarr;

void init\_buffer(dynarr \*buf)

{

buf->size = 1;

buf->data = malloc(sizeof(ll));

buf->data[0] = 1;

}

void add\_element(dynarr \*buf, ll value)

{

buf->data = realloc(buf->data, (buf->size + 1) \* sizeof(ll));

buf->data[buf->size++] = value;

}

void compact\_buffer(dynarr \*buf, int new\_start)

{

int new\_size = buf->size - new\_start;

for (int i = 0; i < new\_size; i++)

{

buf->data[i] = buf->data[new\_start + i];

}

buf->size = new\_size;

buf->data = realloc(buf->data, buf->size \* sizeof(ll));

}

ll give\_n\_sequence(int n)

{

dynarr buf;

init\_buffer(&buf);

int i3 = 0, i5 = 0, i7 = 0;

int last\_compacted = 0;

int prev\_min\_used = 0;

for (int i = 1; i <= n; i++)

{

ll next3 = buf.data[i3] \* 3;

ll next5 = buf.data[i5] \* 5;

ll next7 = buf.data[i7] \* 7;

ll next = next3;

if (next5 < next) next = next5;

if (next7 < next) next = next7;

add\_element(&buf, next);

if (next == next3) i3++;

if (next == next5) i5++;

if (next == next7) i7++;

int min\_unused = (i3 < i5) ? i3 : i5;

min\_unused = (min\_unused < i7) ? min\_unused : i7;

if (min\_unused != prev\_min\_used)

{

compact\_buffer(&buf, min\_unused);

i3 -= min\_unused;

i5 -= min\_unused;

i7 -= min\_unused;

last\_compacted += min\_unused;

}

prev\_min\_used = min\_unused;

}

ll result = buf.data[n - last\_compacted];

free(buf.data);

return result;

}

int main() {

int n;

while (1) {

if (scanf("%d", &n) != 1) break;

printf("%llu\n", give\_n\_sequence(n));

}

return 0;

}