МИНОБРНАУКИ РОССИИ
Санкт-Петербургский государственный
электротехнический университет «ЛЭТИ»
им. В.И. Ульянова (Ленина)
Кафедра ВТ

ОТЧЁТ
ЛАБОРАТОРНАЯ РАБОТА №2
по дисциплине «Параллельные алгоритмы и системы»
Тема: Линейные разделяемые структуры данных (Вариант 3)

Студент гр. 9892 _____ Лескин К.А.

Преподаватель _____ Пазников А.А.

Санкт-Петербург
2022

1 Цель работы

Изучение и реализация линейных разделяемых структур данных в многопоточных системах.

2 Задание

1. Реализовать структуру данных (языки: ‘C++/C‘ желательно, ‘Java‘ допускается, но хуже)
2. Разработать простой тест
3. Оценить эффективность, построить графы, проанализировать, сформулировать выводы.
4. Написать отчет с результатами экспериментов и выводами.

3 Выполнение задания

Работа выполнена на языке `C++14`.

В ходе работы был создан класс `Set`, реализующий множество на основе неблокириющего списка.

Для тестирования был написан класс `TestSet`. Данный класс на параметризуемом интервале потоков запускает функцию, случайно выбирающую операцию над множеством и выполняющую её. Вероятность выбора той или иной операции может быть задана. Функция работает до тех пор, пока не истечёт количество операций, которое тоже параметризуется.

Тестирование проводилось на `CPU` модели
`AMD Ryzen 3 3200U with Radeon Vega Mobile Gfx` с 4 `CPUs`.

Тест 1

Параметры тестирования:

- Количество операций: 32
- Вероятность выполнения операции `add`: 33.33 %
- Вероятность выполнения операции `remove`: 33.33 %
- Вероятность выполнения операции `contains`: 33.33 %

Результаты тестирования представленны в таблице 1

Кол-во потоков	Время выполнения (мс)
1	11
2	10
3	12
4	4

Таблица 1 – Результаты теста 1

График результатов тестирования представлен на рис.1

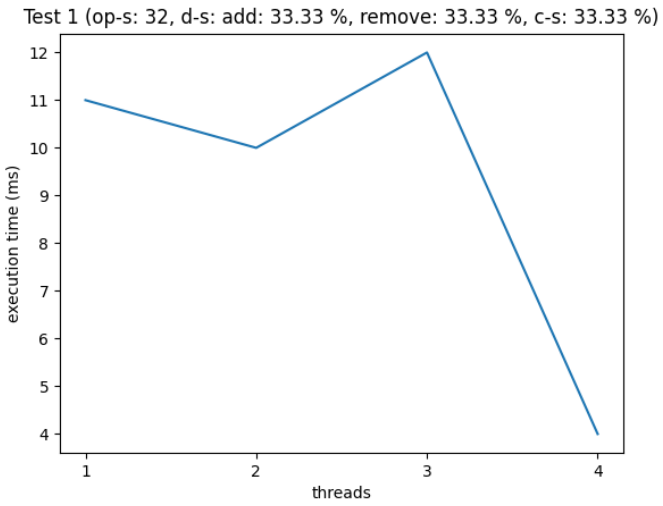

Рис. 1 – График результатов теста 1

Тест 2

Параметры тестирования:

- Количество операций: 32
- Вероятность выполнения операции `add`: 90 %
- Вероятность выполнения операции `remove`: 9 %
- Вероятность выполнения операции `contains`: 1 %

Результаты тестирования представленны в таблице 2

Кол-во потоков	Время выполнения (мс)
1	2
2	3
3	8
4	15

Таблица 2 – Результаты теста 2

График результатов тестирования представлен на рис.2

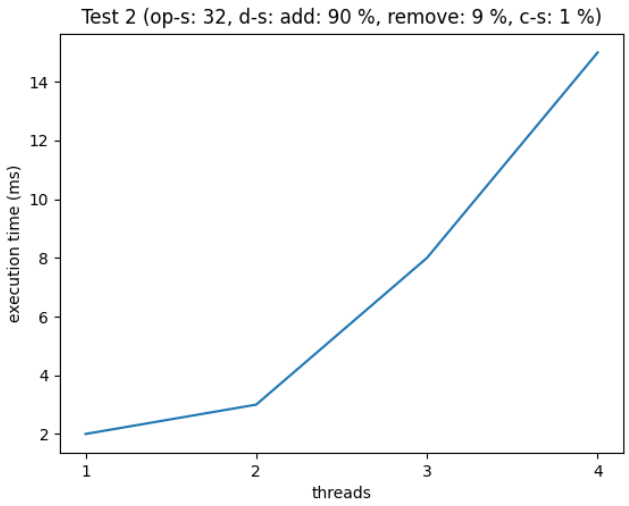

Рис. 2 – График результатов теста 2

Тест 3

Параметры тестирования:

- Количество операций: 32
- Вероятность выполнения операции `add`: 90 %
- Вероятность выполнения операции `remove`: 1 %
- Вероятность выполнения операции `contains`: 9 %

Результаты тестирования представленны в таблице 3

Кол-во потоков	Время выполнения (мс)
1	8
2	8
3	11
4	9

Таблица 3 – Результаты теста 3

График результатов тестирования представлен на рис.3

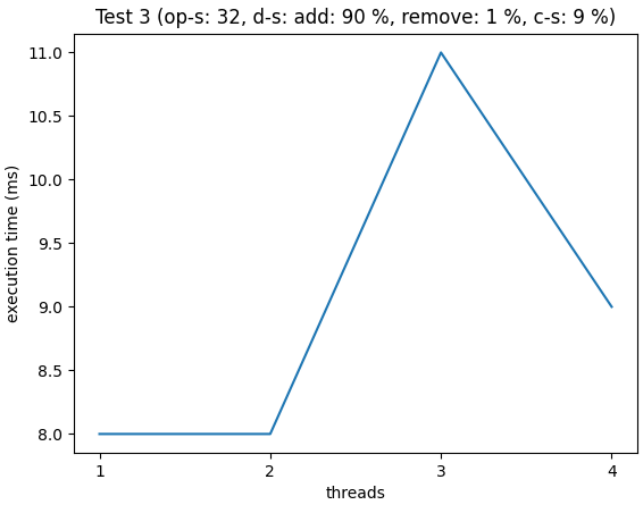

Рис. 3 – График результатов теста 3

Тест 4

Параметры тестирования:

- Количество операций: 32
- Вероятность выполнения операции `add`: 9 %
- Вероятность выполнения операции `remove`: 90 %
- Вероятность выполнения операции `contains`: 1 %

Результаты тестирования представленны в таблице 4

Кол-во потоков	Время выполнения (мс)
1	6
2	14
3	18
4	18

Таблица 4 – Результаты теста 4

График результатов тестирования представлен на рис.4

Рис. 4 – График результатов теста 4

Тест 5

Параметры тестирования:

- Количество операций: 32
- Вероятность выполнения операции `add`: 1 %
- Вероятность выполнения операции `remove`: 90 %
- Вероятность выполнения операции `contains`: 9 %

Результаты тестирования представленны в таблице 5

Кол-во потоков	Время выполнения (мс)
1	1
2	1
3	2
4	2

Таблица 5 – Результаты теста 5

График результатов тестирования представлен на рис.5

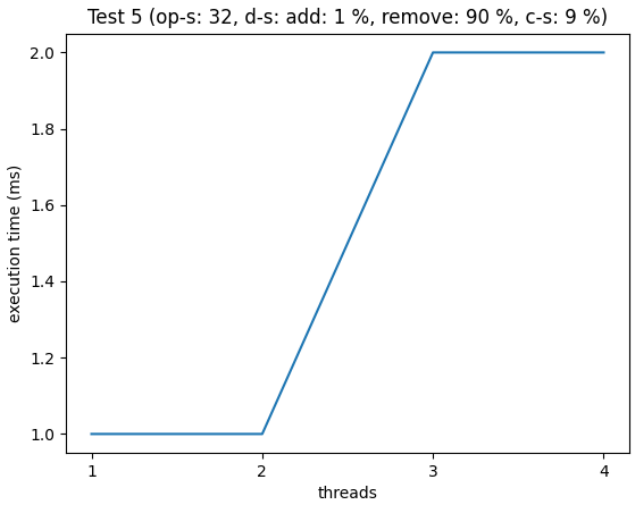

Рис. 5 – График результатов теста 5

Тест 6

Параметры тестирования:

- Количество операций: 32
- Вероятность выполнения операции `add`: 9 %
- Вероятность выполнения операции `remove`: 1 %
- Вероятность выполнения операции `contains`: 90 %

Результаты тестирования представленны в таблице 6

Кол-во потоков	Время выполнения (мс)
1	4
2	4
3	8
4	5

Таблица 6 – Результаты теста 6

График результатов тестирования представлен на рис.6

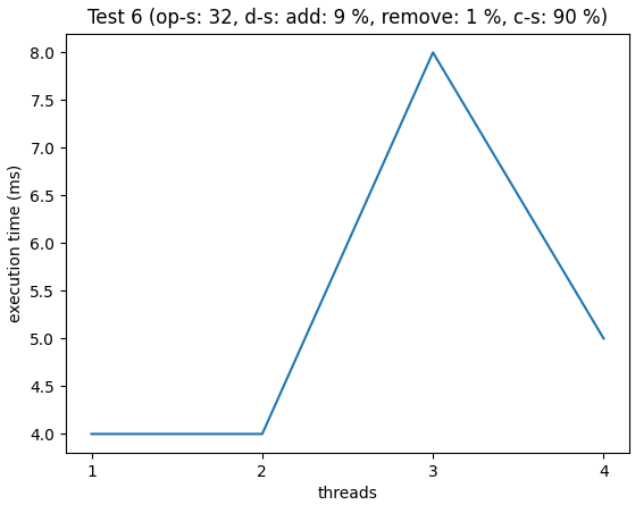

Рис. 6 – График результатов теста 6

Тест 7

Параметры тестирования:

- Количество операций: 32
- Вероятность выполнения операции `add`: 1 %
- Вероятность выполнения операции `remove`: 9 %
- Вероятность выполнения операции `contains`: 90 %

Результаты тестирования представленны в таблице 7

Кол-во потоков	Время выполнения (мс)
1	2
2	4
3	1
4	1

Таблица 7 – Результаты теста 7

График результатов тестирования представлен на рис.7

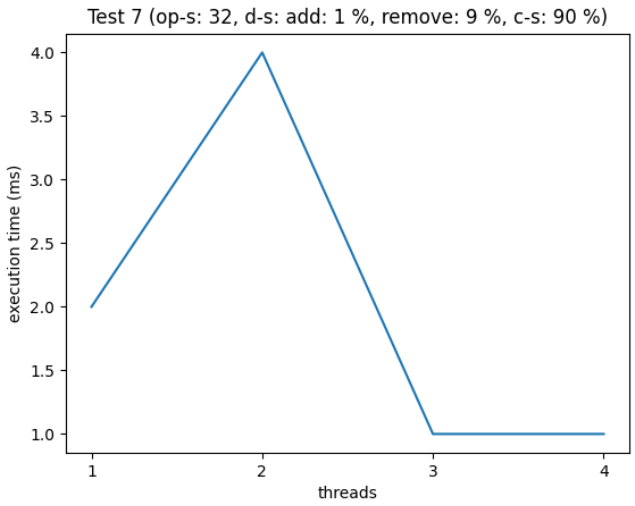

Рис. 7 – График результатов теста 7

Тест 8

Параметры тестирования:

- Количество операций: 128
- Вероятность выполнения операции `add`: 33.33 %
- Вероятность выполнения операции `remove`: 33.33 %
- Вероятность выполнения операции `contains`: 33.33 %

Результаты тестирования представленны в таблице 8

Кол-во потоков	Время выполнения (мс)
1	5
2	9
3	5
4	4

Таблица 8 – Результаты теста 8

График результатов тестирования представлен на рис.8

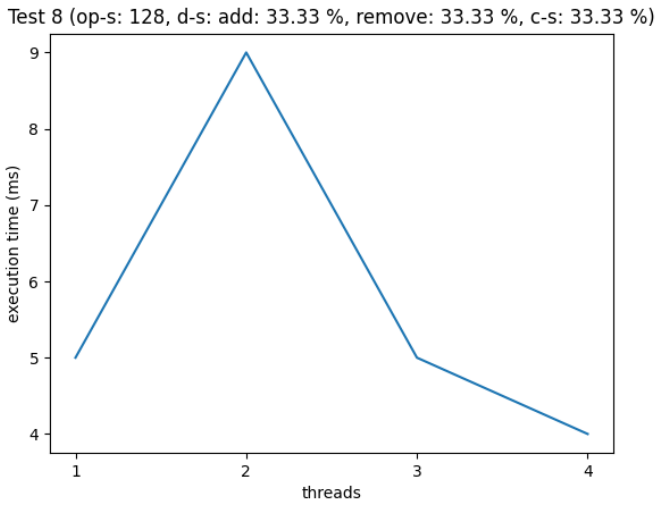

Рис. 8 – График результатов теста 8

Тест 9

Параметры тестирования:

- Количество операций: 128
- Вероятность выполнения операции `add`: 90 %
- Вероятность выполнения операции `remove`: 9 %
- Вероятность выполнения операции `contains`: 1 %

Результаты тестирования представленны в таблице 9

Кол-во потоков	Время выполнения (мс)
1	14
2	3
3	4
4	6

Таблица 9 – Результаты теста 9

График результатов тестирования представлен на рис.9

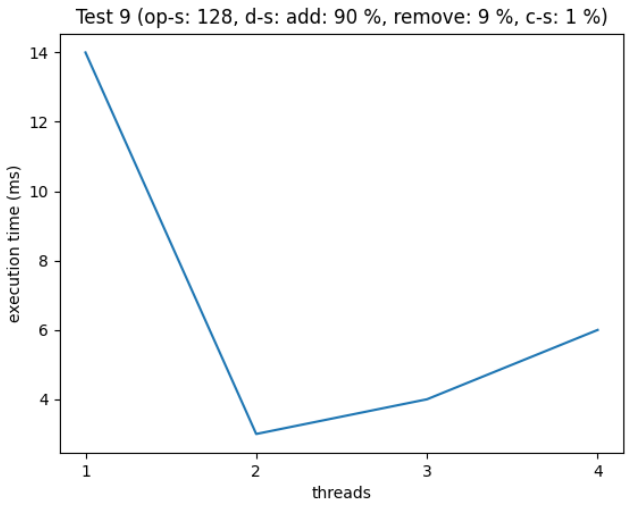

Рис. 9 – График результатов теста 9

Тест 10

Параметры тестирования:

- Количество операций: 128
- Вероятность выполнения операции `add`: 90 %
- Вероятность выполнения операции `remove`: 1 %
- Вероятность выполнения операции `contains`: 9 %

Результаты тестирования представленны в таблице 10

Кол-во потоков	Время выполнения (мс)
1	9
2	7
3	7
4	3

Таблица 10 – Результаты теста 10

График результатов тестирования представлен на рис.10

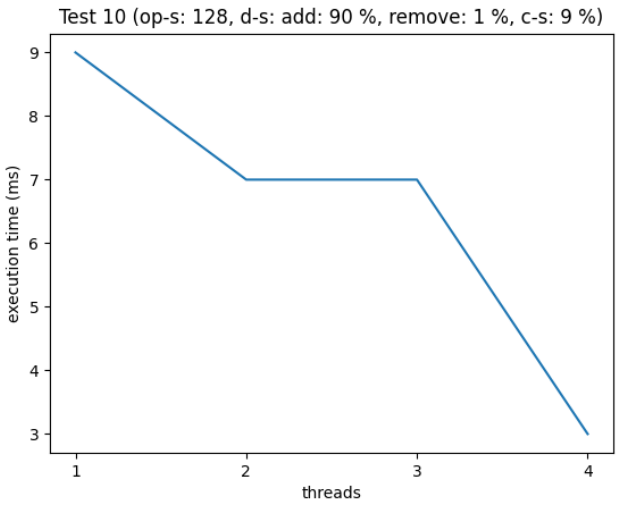

Рис. 10 – График результатов теста 10

Тест 11

Параметры тестирования:

- Количество операций: 128
- Вероятность выполнения операции `add`: 9 %
- Вероятность выполнения операции `remove`: 90 %
- Вероятность выполнения операции `contains`: 1 %

Результаты тестирования представленны в таблице 11

Кол-во потоков	Время выполнения (мс)
1	7
2	5
3	3
4	39

Таблица 11 – Результаты теста 11

График результатов тестирования представлен на рис.11

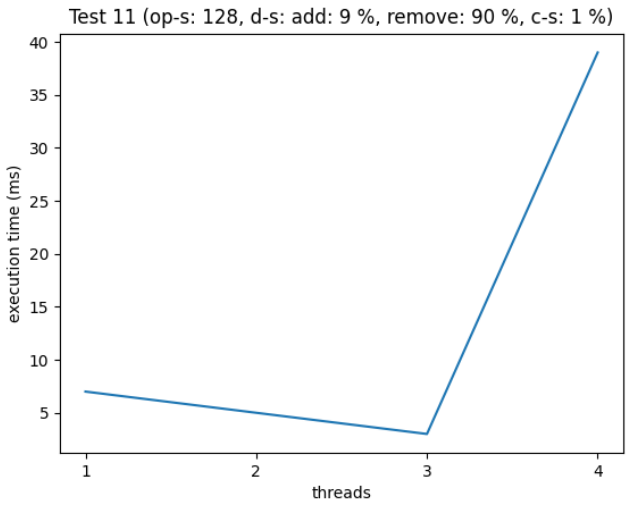

Рис. 11 – График результатов теста 11

Тест 12

Параметры тестирования:

- Количество операций: 128
- Вероятность выполнения операции `add`: 1 %
- Вероятность выполнения операции `remove`: 90 %
- Вероятность выполнения операции `contains`: 9 %

Результаты тестирования представленны в таблице 12

Кол-во потоков	Время выполнения (мс)
1	11
2	4
3	5
4	4

Таблица 12 – Результаты теста 12

График результатов тестирования представлен на рис.12

Рис. 12 – График результатов теста 12

Тест 13

Параметры тестирования:

- Количество операций: 128
- Вероятность выполнения операции `add`: 9 %
- Вероятность выполнения операции `remove`: 1 %
- Вероятность выполнения операции `contains`: 90 %

Результаты тестирования представленны в таблице 13

Кол-во потоков	Время выполнения (мс)
1	5
2	13
3	7
4	6

Таблица 13 – Результаты теста 13

График результатов тестирования представлен на рис.13

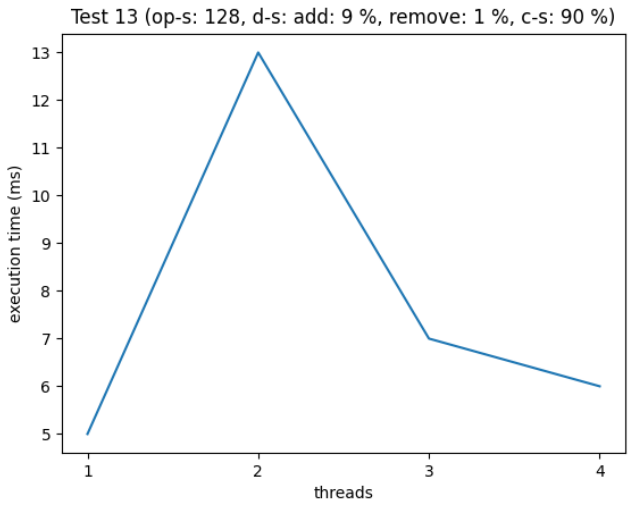

Рис. 13 – График результатов теста 13

Тест 14

Параметры тестирования:

- Количество операций: 128
- Вероятность выполнения операции `add`: 1 %
- Вероятность выполнения операции `remove`: 9 %
- Вероятность выполнения операции `contains`: 90 %

Результаты тестирования представленны в таблице 14

Кол-во потоков	Время выполнения (мс)
1	5
2	2
3	3
4	12

Таблица 14 – Результаты теста 14

График результатов тестирования представлен на рис.14

Рис. 14 – График результатов теста 14

Тест 15

Параметры тестирования:

- Количество операций: 1024
- Вероятность выполнения операции `add`: 33.33 %
- Вероятность выполнения операции `remove`: 33.33 %
- Вероятность выполнения операции `contains`: 33.33 %

Результаты тестирования представленны в таблице 15

Кол-во потоков	Время выполнения (мс)
1	49
2	35
3	46
4	42

Таблица 15 – Результаты теста 15

График результатов тестирования представлен на рис.15

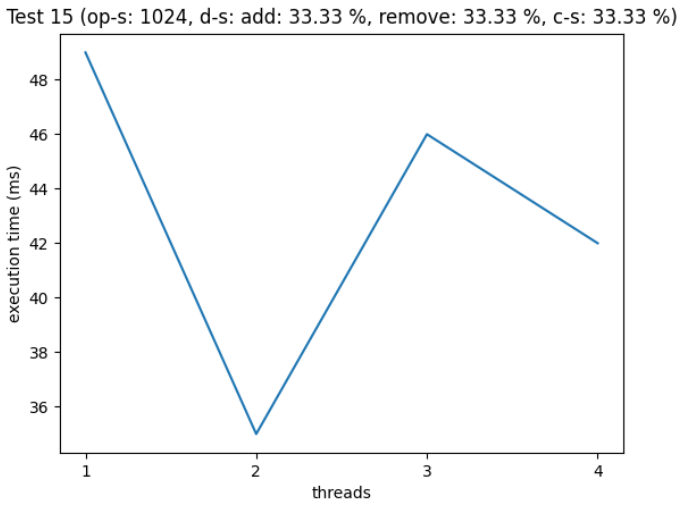

Рис. 15 – График результатов теста 15

Тест 16

Параметры тестирования:

- Количество операций: 1024
- Вероятность выполнения операции `add`: 90 %
- Вероятность выполнения операции `remove`: 9 %
- Вероятность выполнения операции `contains`: 1 %

Результаты тестирования представленны в таблице 16

Кол-во потоков	Время выполнения (мс)
1	80
2	29
3	24
4	23

Таблица 16 – Результаты теста 16

График результатов тестирования представлен на рис.16

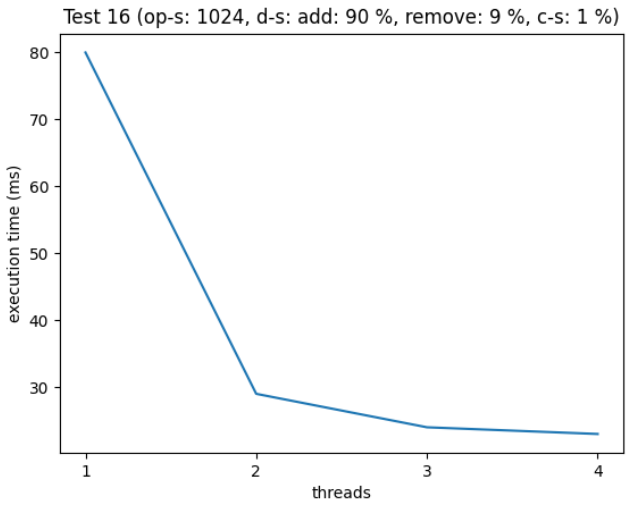

Рис. 16 – График результатов теста 16

Тест 17

Параметры тестирования:

- Количество операций: 1024
- Вероятность выполнения операции `add`: 90 %
- Вероятность выполнения операции `remove`: 1 %
- Вероятность выполнения операции `contains`: 9 %

Результаты тестирования представленны в таблице 17

Кол-во потоков	Время выполнения (мс)
1	46
2	30
3	22
4	19

Таблица 17 – Результаты теста 17

График результатов тестирования представлен на рис.17

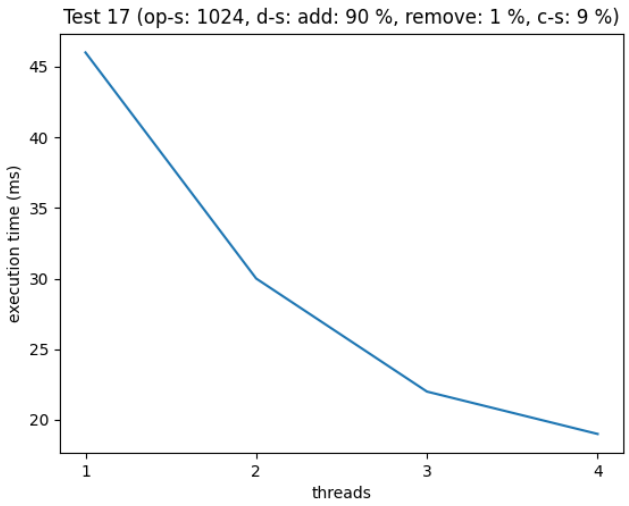

Рис. 17 – График результатов теста 17

Тест 18

Параметры тестирования:

- Количество операций: 1024
- Вероятность выполнения операции `add`: 9 %
- Вероятность выполнения операции `remove`: 90 %
- Вероятность выполнения операции `contains`: 1 %

Результаты тестирования представленны в таблице 18

Кол-во потоков	Время выполнения (мс)
1	43
2	27
3	22
4	30

Таблица 18 – Результаты теста 18

График результатов тестирования представлен на рис.18

Рис. 18 – График результатов теста 18

Тест 19

Параметры тестирования:

- Количество операций: 1024
- Вероятность выполнения операции `add`: 1 %
- Вероятность выполнения операции `remove`: 90 %
- Вероятность выполнения операции `contains`: 9 %

Результаты тестирования представленны в таблице 19

Кол-во потоков	Время выполнения (мс)
1	50
2	39
3	22
4	23

Таблица 19 – Результаты теста 19

График результатов тестирования представлен на рис.19

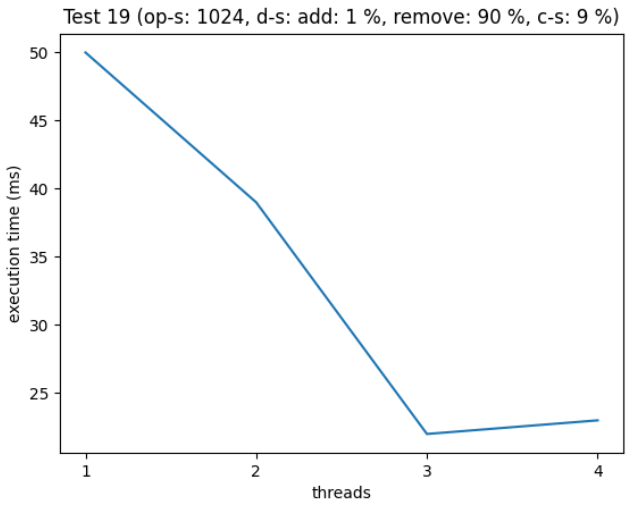

Рис. 19 – График результатов теста 19

Тест 20

Параметры тестирования:

- Количество операций: 1024
- Вероятность выполнения операции `add`: 9 %
- Вероятность выполнения операции `remove`: 1 %
- Вероятность выполнения операции `contains`: 90 %

Результаты тестирования представленны в таблице 20

Кол-во потоков	Время выполнения (мс)
1	44
2	37
3	21
4	34

Таблица 20 – Результаты теста 20

График результатов тестирования представлен на рис.20

Рис. 20 – График результатов теста 20

Тест 21

Параметры тестирования:

- Количество операций: 1024
- Вероятность выполнения операции `add`: 1 %
- Вероятность выполнения операции `remove`: 9 %
- Вероятность выполнения операции `contains`: 90 %

Результаты тестирования представленны в таблице 21

Кол-во потоков	Время выполнения (мс)
1	43
2	26
3	24
4	21

Таблица 21 – Результаты теста 21

График результатов тестирования представлен на рис.21

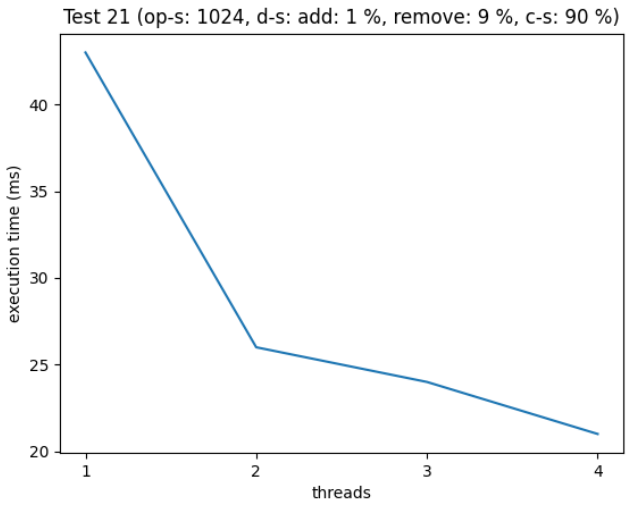

Рис. 21 – График результатов теста 21

Тест 22

Параметры тестирования:

- Количество операций: 32768
- Вероятность выполнения операции `add`: 33.33 %
- Вероятность выполнения операции `remove`: 33.33 %
- Вероятность выполнения операции `contains`: 33.33 %

Результаты тестирования представленны в таблице 22

Кол-во потоков	Время выполнения (мс)
1	1350
2	828
3	599
4	605

Таблица 22 – Результаты теста 22

График результатов тестирования представлен на рис.22

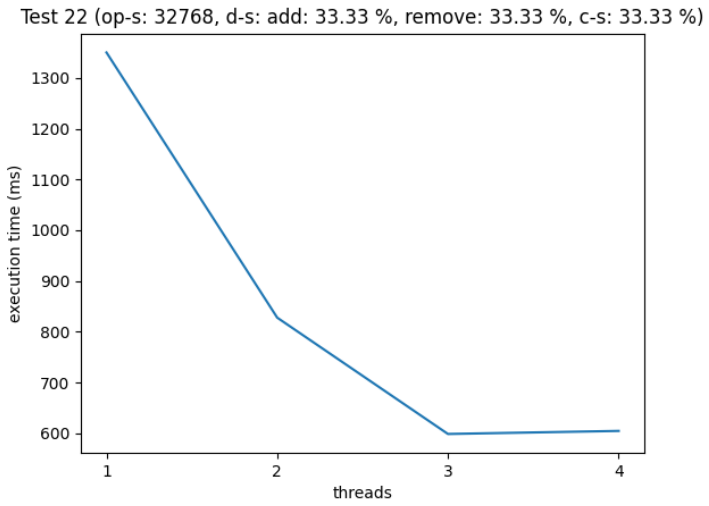

Рис. 22 – График результатов теста 22

Тест 23

Параметры тестирования:

- Количество операций: 32768
- Вероятность выполнения операции `add`: 90 %
- Вероятность выполнения операции `remove`: 9 %
- Вероятность выполнения операции `contains`: 1 %

Результаты тестирования представленны в таблице 23

Кол-во потоков	Время выполнения (мс)
1	1399
2	724
3	582
4	567

Таблица 23 – Результаты теста 23

График результатов тестирования представлен на рис.23

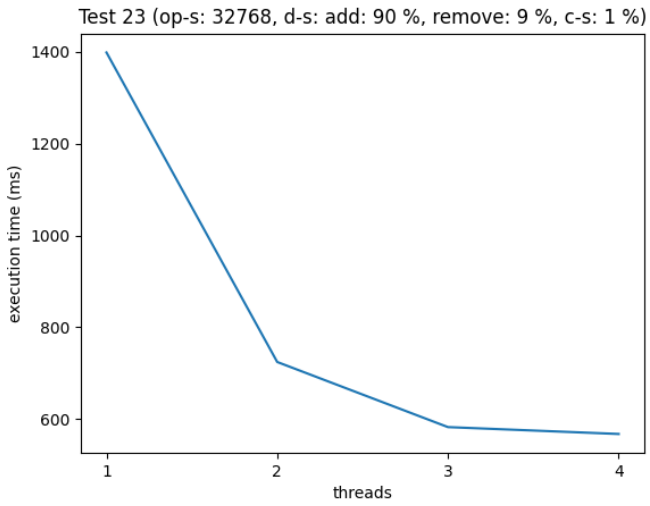

Рис. 23 – График результатов теста 23

Тест 24

Параметры тестирования:

- Количество операций: 32768
- Вероятность выполнения операции `add`: 90 %
- Вероятность выполнения операции `remove`: 1 %
- Вероятность выполнения операции `contains`: 9 %

Результаты тестирования представленны в таблице 24

Кол-во потоков	Время выполнения (мс)
1	1414
2	825
3	596
4	580

Таблица 24 – Результаты теста 24

График результатов тестирования представлен на рис.24

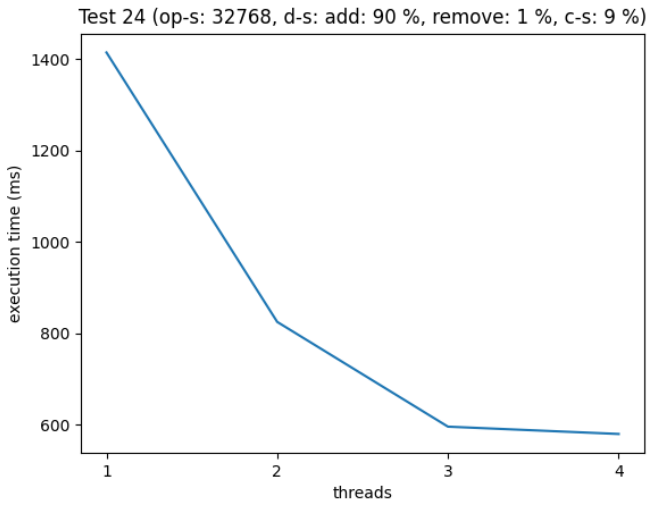

Рис. 24 – График результатов теста 24

Тест 25

Параметры тестирования:

- Количество операций: 32768
- Вероятность выполнения операции `add`: 9 %
- Вероятность выполнения операции `remove`: 90 %
- Вероятность выполнения операции `contains`: 1 %

Результаты тестирования представленны в таблице 25

Кол-во потоков	Время выполнения (мс)
1	1319
2	675
3	562
4	554

Таблица 25 – Результаты теста 25

График результатов тестирования представлен на рис.25

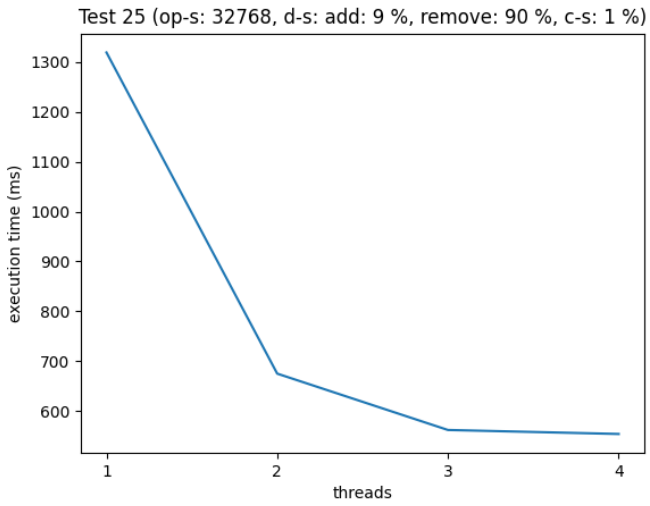

Рис. 25 – График результатов теста 25

Тест 26

Параметры тестирования:

- Количество операций: 32768
- Вероятность выполнения операции `add`: 1 %
- Вероятность выполнения операции `remove`: 90 %
- Вероятность выполнения операции `contains`: 9 %

Результаты тестирования представленны в таблице 26

Кол-во потоков	Время выполнения (мс)
1	1304
2	681
3	554
4	554

Таблица 26 – Результаты теста 26

График результатов тестирования представлен на рис.26

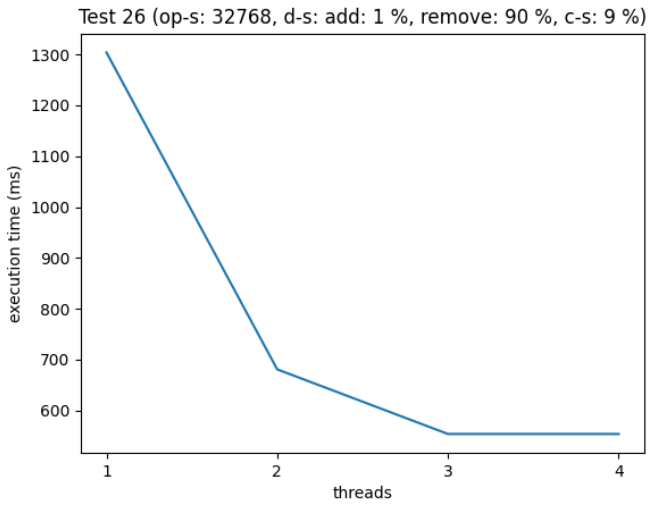

Рис. 26 – График результатов теста 26

Тест 27

Параметры тестирования:

- Количество операций: 32768
- Вероятность выполнения операции `add`: 9 %
- Вероятность выполнения операции `remove`: 1 %
- Вероятность выполнения операции `contains`: 90 %

Результаты тестирования представленны в таблице 27

Кол-во потоков	Время выполнения (мс)
1	1360
2	775
3	623
4	553

Таблица 27 – Результаты теста 27

График результатов тестирования представлен на рис.27

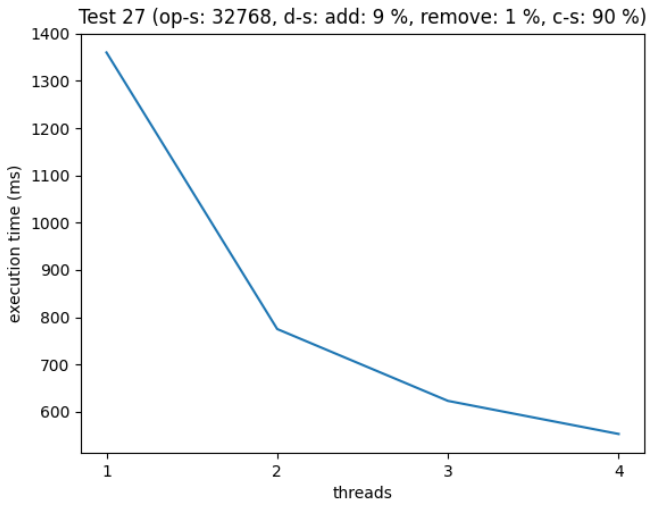

Рис. 27 – График результатов теста 27

Тест 28

Параметры тестирования:

- Количество операций: 32768
- Вероятность выполнения операции `add`: 1 %
- Вероятность выполнения операции `remove`: 9 %
- Вероятность выполнения операции `contains`: 90 %

Результаты тестирования представленны в таблице 28

Кол-во потоков	Время выполнения (мс)
1	1312
2	671
3	549
4	533

Таблица 28 – Результаты теста 28

График результатов тестирования представлен на рис.28

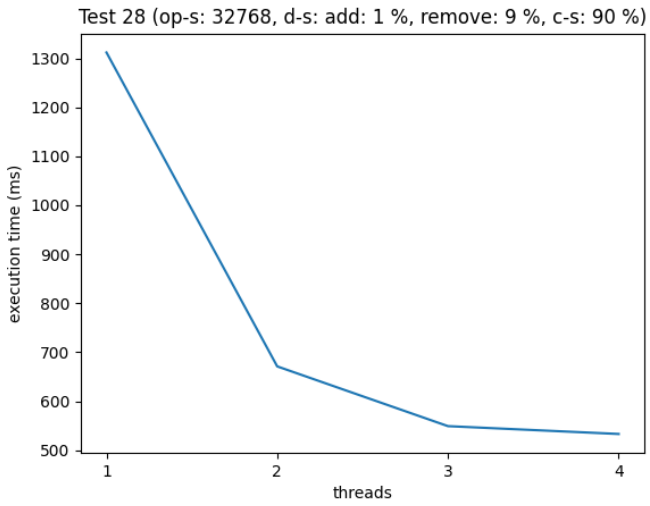

Рис. 28 – График результатов теста 28

Тест 29

Параметры тестирования:

- Количество операций: 131072
- Вероятность выполнения операции `add`: 33.33 %
- Вероятность выполнения операции `remove`: 33.33 %
- Вероятность выполнения операции `contains`: 33.33 %

Результаты тестирования представленны в таблице 29

Кол-во потоков	Время выполнения (мс)
1	5351
2	2721
3	2149
4	2094

Таблица 29 – Результаты теста 29

График результатов тестирования представлен на рис.29

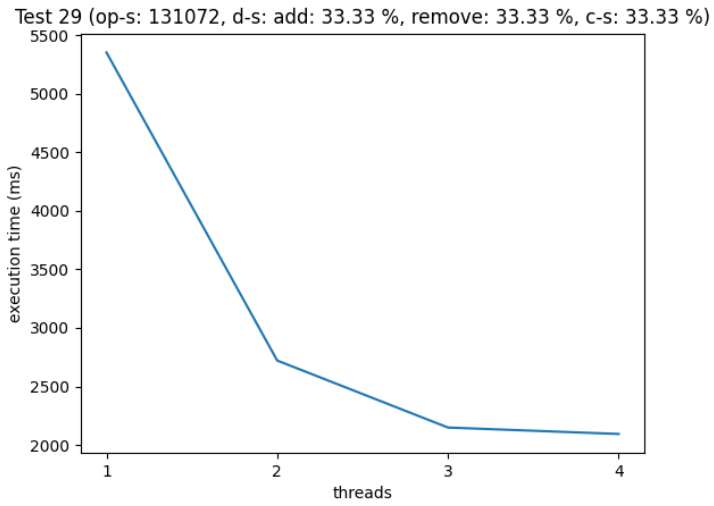

Рис. 29 – График результатов теста 29

Тест 30

Параметры тестирования:

- Количество операций: 131072
- Вероятность выполнения операции `add`: 90 %
- Вероятность выполнения операции `remove`: 9 %
- Вероятность выполнения операции `contains`: 1 %

Результаты тестирования представленны в таблице 30

Кол-во потоков	Время выполнения (мс)
1	5531
2	2834
3	2198
4	2122

Таблица 30 – Результаты теста 30

График результатов тестирования представлен на рис.30

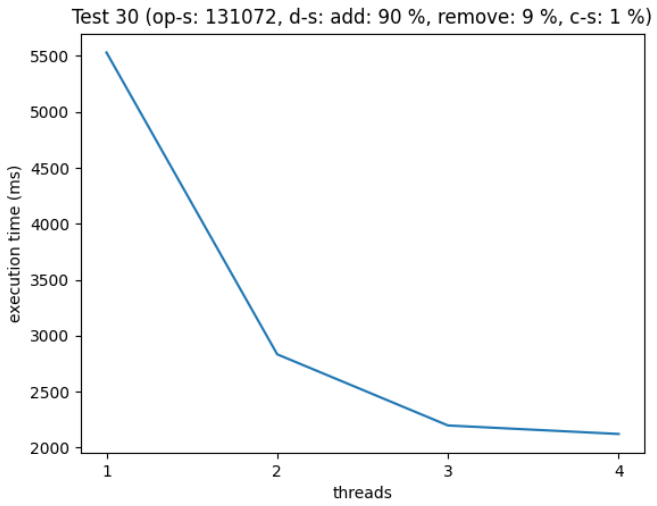

Рис. 30 – График результатов теста 30

Тест 31

Параметры тестирования:

- Количество операций: 131072
- Вероятность выполнения операции `add`: 90 %
- Вероятность выполнения операции `remove`: 1 %
- Вероятность выполнения операции `contains`: 9 %

Результаты тестирования представленны в таблице 31

Кол-во потоков	Время выполнения (мс)
1	5546
2	2816
3	2204
4	2125

Таблица 31 – Результаты теста 31

График результатов тестирования представлен на рис.31

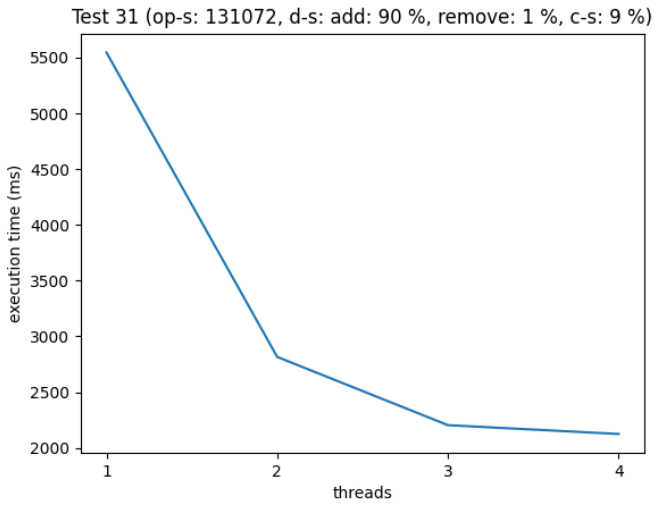

Рис. 31 – График результатов теста 31

Тест 32

Параметры тестирования:

- Количество операций: 131072
- Вероятность выполнения операции `add`: 9 %
- Вероятность выполнения операции `remove`: 90 %
- Вероятность выполнения операции `contains`: 1 %

Результаты тестирования представленны в таблице 32

Кол-во потоков	Время выполнения (мс)
1	5211
2	2671
3	2128
4	2091

Таблица 32 – Результаты теста 32

График результатов тестирования представлен на рис.32

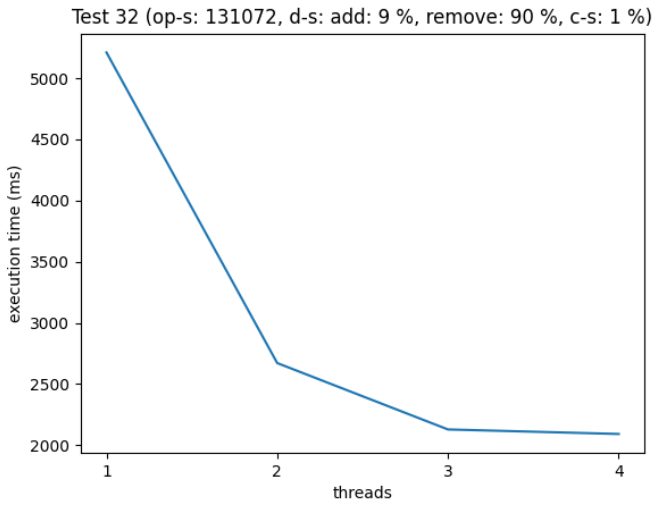

Рис. 32 – График результатов теста 32

Тест 33

Параметры тестирования:

- Количество операций: 131072
- Вероятность выполнения операции `add`: 1 %
- Вероятность выполнения операции `remove`: 90 %
- Вероятность выполнения операции `contains`: 9 %

Результаты тестирования представленны в таблице 33

Кол-во потоков	Время выполнения (мс)
1	5191
2	2630
3	2119
4	2100

Таблица 33 – Результаты теста 33

График результатов тестирования представлен на рис.33

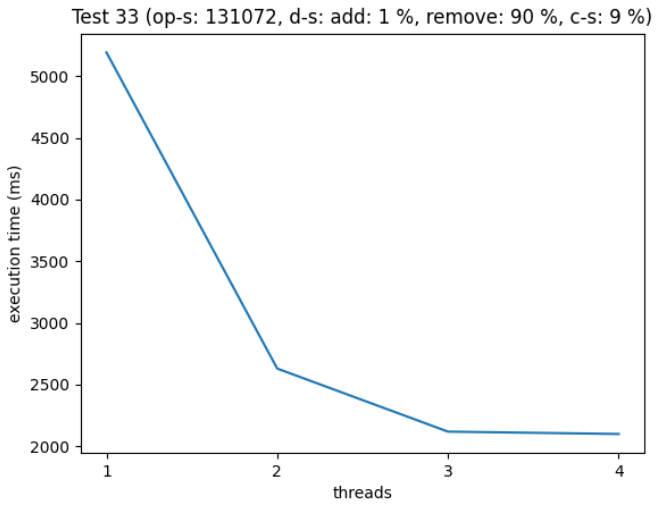

Рис. 33 – График результатов теста 33

Тест 34

Параметры тестирования:

- Количество операций: 131072
- Вероятность выполнения операции `add`: 9 %
- Вероятность выполнения операции `remove`: 1 %
- Вероятность выполнения операции `contains`: 90 %

Результаты тестирования представленны в таблице 34

Кол-во потоков	Время выполнения (мс)
1	5486
2	2833
3	2194
4	2103

Таблица 34 – Результаты теста 34

График результатов тестирования представлен на рис.34

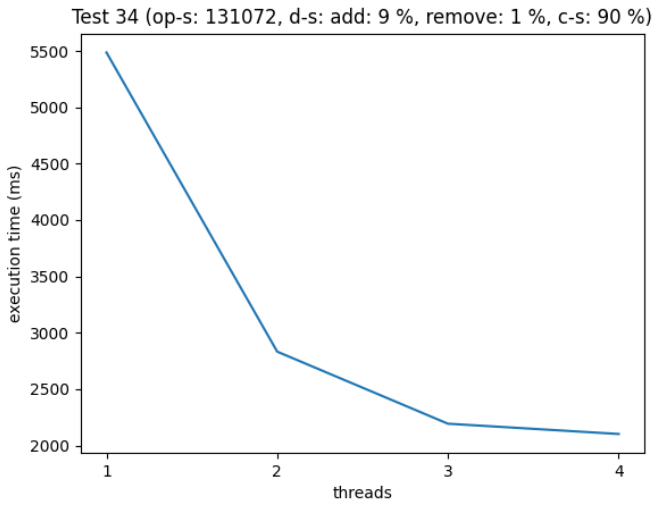

Рис. 34 – График результатов теста 34

Тест 35

Параметры тестирования:

- Количество операций: 131072
- Вероятность выполнения операции `add`: 1 %
- Вероятность выполнения операции `remove`: 9 %
- Вероятность выполнения операции `contains`: 90 %

Результаты тестирования представленны в таблице 35

Кол-во потоков	Время выполнения (мс)
1	5209
2	2688
3	2157
4	2083

Таблица 35 – Результаты теста 35

График результатов тестирования представлен на рис.35

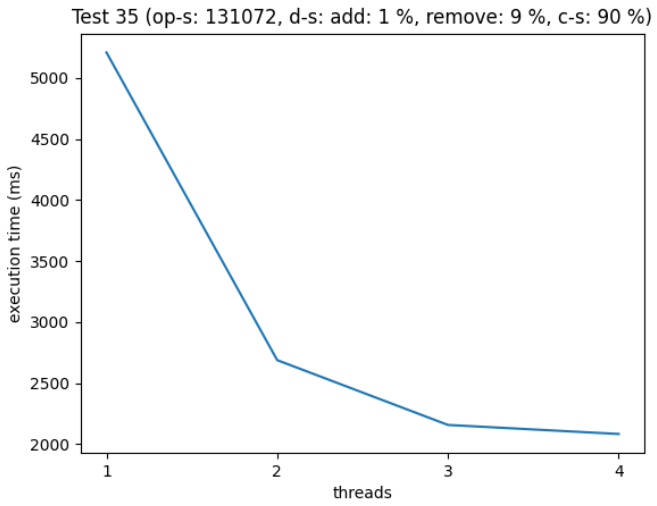

Рис. 35 – График результатов теста 35

4 Выводы

В результате выполнения лабораторной работы был создан класс `Set`, реализующий множество на основе неблокириющего списка – линейная разделяемая структура данных – который может использоваться в многопоточных системах.

По результатам тестов можно сделать вывод, что использование данаго класса возможно как в не многопоточных системах, так и в многопоточных, однако эффективность будет маленькой при небольших размерах данных, количестве операций и потоков и становиться заметной только при увеличении трёх параметров.

5 Приложение А – Листинг

Листинг 1 – src/log.h

```
#ifndef INC_LOG_H
#define INC_LOG_H

#include <iostream>

#define DBG

#define LOG(msg) {std::cout << msg;} while(0)
#define LOGLN(msg) {LOG(msg << '\n');} while(0)

#ifdef DBG
#define DEBUG(msg) {LOG(msg);} while(0)
#define DEBUGLN(msg) {LOGLN(msg);} while(0)
#else
#define DEBUG(msg) {;} while(0)
#define DEBUGLN(msg) {;} while(0)
#endif

#endif //INC_LOG_H
```

Листинг 2 – src/main.cpp

```
#include <vector>
#include "set/test_set/test_set.h"
#include "set/test_set/set_operations_distribution.h"
#include "log.h"

int main()
{
    DEBUGLN("This computer has " << getThreadsNumber() << "
   threads");
    std::vector<SetOperationsDistribution> distributions{
        SetOperationsDistribution(1,1,1),
        SetOperationsDistribution(90,9,1),
        SetOperationsDistribution(90,1,9),
        SetOperationsDistribution(9,90,1),
        SetOperationsDistribution(1,90,9),
        SetOperationsDistribution(9,1,90),
        SetOperationsDistribution(1,9,90)
    };
    std::vector<int> operations_num{1 << 5, 1 << 7, 1 << 10,
   1 << 15, 1 << 17};
    TestSet tester = TestSet();
    tester.run(distributions, operations_num, "../test_data.
   yaml");
    return 0;
}
```

Листинг 3 – src/timer.h

```
#ifndef INC_TIMER_H
#define INC_TIMER_H

#include <chrono>


template<typename Clock=std::chrono::high_resolution_clock>
class Timer {
public:
    Timer();
    Timer(const Timer&) = delete;
    Timer& operator=(const Timer&) = delete;
    int elapsed_ms();
private:
    std::chrono::time_point<Clock> _start_time;
};

template<class Clock>
Timer<Clock>::Timer() :
        _start_time(Clock::now()) {}

template<class Clock>
int Timer<Clock>::elapsed_ms()
{
    return std::chrono::duration_cast<std::chrono::
   milliseconds>(Clock::now() - _start_time).count();
}

#endif //INC_TIMER_H
```

Листинг 4 – src/rand/rand.cpp

```
#include "rand.h"

Rand::Rand()
{
    std::random_device rd;
    _generator = std::mt19937(rd());
}

uint Rand::operator()()
{
    return _generator();
}

uint Rand::range(uint start, uint end)
{
    return start + std::abs(static_cast<int>((*this)())) % (
    end + 1);
}
```

Листинг 5 – src/rand/rand.h

```
#ifndef INC_RAND_H
#define INC_RAND_H

#include <random>


typedef unsigned int uint;

class Rand
{
public:
    Rand();
    uint operator()();
    uint range(uint start = 0, uint end = 9);

private:
    std::mt19937 _generator;
};


#endif //INC_RAND_H
```

Листинг 6 – src/set/set.h

```
#ifndef INC_SET_H
#define INC_SET_H


#include <mutex>
#include <climits>

#include "../log.h"


struct Node
{
    Node(int n = 0, Node *xt = nullptr, bool m = false);

    int value;
    Node *next;
    bool marked;
    std::mutex lock;
};

class Set
{
public:
    Set();
    ~Set();

    bool add(const int &v);
    bool remove(const int &v);
    bool contains(const int &v);

private:
    Node *_head, *_tail;

    void _init();
    bool _check(const Node *a, const Node *b);
    void _clear();
};


#endif //INC_SET_H
```

Листинг 7 – src/set/set.cpp

```
#include "set.h"

Node::Node(int n, Node *xt, bool m)
          :value{n}, next{xt}, marked{m} {}

Set::Set()
{
    _init();
}

Set::~Set()
{
    _clear();
}

bool Set::add(const int &v)
{
    while (true) {
        Node *prev = _head;
        Node *next = _head->next;

        while (next->value < v)
        {
            prev = next; next = next->next;
        }

        std::unique_lock<std::mutex> prev_lock(prev->lock);
        std::unique_lock<std::mutex> curr_lock(next->lock);

        if(_check(prev, next))
        {
            if (next->value == v)
            {
                return false;
            }
            else
            {
                Node *newn = new Node(v, next);
                prev->next = newn;
                return true;
            }
        }
    }
}

bool Set::remove(const int &v)
{
    while (true) {
```

```
49          Node *prev = _head;
50          Node *tdel = _head->next;
51
52          while (tdel->value < v)
53          {
54              prev = tdel; tdel = tdel->next;
55          }
56
57          std::unique_lock<std::mutex> prev_lock(prev->lock);
58          std::unique_lock<std::mutex> curr_lock(tdel->lock);
59
60          if(_check(prev, tdel))
61          {
62              if (tdel->value != v)
63              {
64                  return false;
65              }
66              else
67              {
68                  tdel->marked = true;
69                  prev->next = tdel->next;
70                  //delete tdel;
71                  return true;
72              }
73          }
74      }
75  }
76
77  bool Set::contains(const int &v)
78  {
79      Node *node = _head;
80      while(node->value < v && node != _tail)
81      {
82          node = node->next;
83      }
84      return node->value == v and !node->marked;
85  }
86
87  void Set::_init()
88  {
89      _head = new Node(INT_MIN);
90      _tail = new Node(INT_MAX);
91      _head->next = _tail;
92  }
93
94  bool Set::_check(const Node *a, const Node *b)
95  {
96      return !a->marked && !b->marked && a->next == b;
97  }
```

```
98
99  void Set::_clear()
100 {
101     Node *p = _head->next;
102     while(p != _tail)
103     {
104         Node *next = p->next;
105         remove(p->value);
106         p = next;
107     }
108 }
```

Листинг 8 – src/set/test_set/test_set.cpp

```cpp
#include "test_set.h"

int getThreadsNumber()
{
    return static_cast<int>(std::thread::hardware_concurrency());
}

TestResult::TestResult(int n, int ops_n, SetOperationsDistribution o, std::map<int, int> r)
    :test_number{n},ops_num{ops_n},op{o},results{r}{}

bool operator!= (const TestResult &a, const TestResult &b)
{
    return a.test_number != b.test_number;
}

TestSet::TestSet(int start, int end)
{
    if(1 <= start && start <= getThreadsNumber()) _start = start; else _start = 1;
    if(_start <= end && end <= getThreadsNumber()) _end = end; else _end = getThreadsNumber();
}

void TestSet::run(const std::vector<SetOperationsDistribution> &sod,
                  const std::vector<int> &ops,
                  const std::string &output_path)
{
    _output_path = output_path;
    std::vector<TestResult> test_results;
    int test_number = 1;
    for(int operations_num : ops)
    {
        for(auto ops_distribution : sod)
        {
            auto results = _runTest(ops_distribution, operations_num);
            test_results.emplace_back(test_number, operations_num, ops_distribution, results);
            ++test_number;
        }
    }
    _writeResults(test_results);
}

std::map<int, int> TestSet::_runTest(SodLinkT p, int
```

```
    operations_num)
{
    std::map<int, int> results;
    LOGLN("operations: " << operations_num);
    for (int i = _start; i <= _end; ++i)
    {
        LOG(p << ": Running test with " << i << " number of
    threads (");
        int t = _testSet(i, p, operations_num);
        LOG("took: " << t << "ms)\n");
        results[i] = t;
    }
    return results;
}

int TestSet::_testSet(std::size_t threads_number, SodLinkT p,
    int operations_num)
{
    std::vector<std::thread> threads;
    auto set = Set();
    std::atomic<int> ops; ops.store(operations_num);
    Timer<std::chrono::high_resolution_clock> t;
    for(int thread_num = 0; thread_num < threads_number; ++
    thread_num)
    {
        threads.emplace_back(_job, std::ref(set), std::ref(
    ops), std::ref(p));
    }
    std::for_each(threads.begin(), threads.end(), std::mem_fn
    (&std::thread::join));
    return t.elapsed_ms();
}

void TestSet::_job(Set &set, std::atomic<int> &operations_num
    , SodLinkT op)
{
    Rand rand;
    while(operations_num > 0)
    {
        char o = op.pickRandom();
        uint value = rand.range(0, 1023);
        _perform(o, set, value);
        --operations_num;
    }
}

bool TestSet::_perform(char op, Set &set, uint value)
{
    switch(op)
```

```
84      {
85          case 'a': return set.add(value);
86          case 'c': return set.contains(value);
87          case 'r': return set.remove(value);
88          default: throw std::runtime_error("Unknown operation"
    );
89      }
90      return false;
91  }
92
93  void TestSet::_writeResults(std::vector<TestResult>
    test_results)
94  {
95      std::ofstream fout(_output_path, std::ios_base::trunc);
96      for(const auto &t : test_results)
97      {
98          fout <<
99          t.test_number << ":\n" <<
100         "  operations: " << t.ops_num << '\n' <<
101         "  distributions: \n" <<
102         "    add: " << t.op.addPercent() << '\n' <<
103         "    remove: " << t.op.removePercent() << '\n' <<
104         "    contains: " << t.op.containsPercent() << '\n' <<
105         "  results:\n";
106         for(auto r : t.results)
107         {
108         fout <<
109         "    " << r.first << ": " << r.second << '\n';
110         }
111     }
112     fout.close();
113 }
```

Листинг 9 – src/set/test_set/test_set.h

```
#ifndef INC_TEST_SET_H
#define INC_TEST_SET_H

#include <vector>
#include <map>
#include <fstream>
#include <thread>
#include <atomic>
#include <algorithm>
#include <functional>

#include "set_operations_distribution.h"
#include "../set.h"
#include "../../log.h"
#include "../../timer.h"
#include "../../rand/rand.h"


int getThreadsNumber();

struct TestResult
{
    TestResult(int n, int ops_n, SetOperationsDistribution o,
    std::map<int, int> r);
    friend bool operator!= (const TestResult &a, const
   TestResult &b);

    int test_number;
    int ops_num;
    SetOperationsDistribution op;
    std::map<int, int> results;
};

class TestSet
{
public:
    TestSet(int start = 0, int end = 0);
    void run(const std::vector<SetOperationsDistribution> &
   sod,
             const std::vector<int> &ops,
             const std::string &output_path);

private:
    int _start, _end;
    std::string _output_path;

    std::map<int, int> _runTest(SodLinkT p, int
   operations_num);
```

```
45      int _testSet(std::size_t threads_number, SodLinkT p, int
        operations_num);
46      static void _job(Set &set, std::atomic<int> &
        operations_num, SodLinkT op);
47      static bool _perform(char op, Set &set, uint value);
48      void _writeResults(std::vector<TestResult> test_results);
49  };
50
51
52  #endif //INC_TEST_SET_H
```

Листинг 10 – src/set/test_set/set_operations_distribution.cpp

```
#include "set_operations_distribution.h"

typedef unsigned int uint;

SetOperationsDistribution::SetOperationsDistribution(uint a,
    uint r, uint c)
     :add{a},remove{r},contains{c}{}

uint SetOperationsDistribution::total() const
{
     return add + remove + contains;
}

char SetOperationsDistribution::pickRandom() const
{
     Rand rand;
     double y = (static_cast<double>(total()) / 100.0) *
    static_cast<double>(rand.range(0, 100));
     uint a = add;
     uint r = add + remove;
     uint c = r + contains;
     char op;
     if(0 <= y && y < a) {op = 'a';}
     else if(a <= y && y < r) {op = 'r';}
     else {op = 'c';}
     //DEBUGLN(y << " >> " << 0 << "-a-" << a << "-r-" << r <<
     "-c-" << c << " --> " << op);
     return op;
}

float SetOperationsDistribution::addPercent() const
{
     return roundf((static_cast<float>(add) / total()) * 100.0
     * 100.0) / 100;;
}

float SetOperationsDistribution::removePercent() const
{
     return roundf((static_cast<float>(remove) / total()) *
    100.0 * 100.0) / 100;;
}

float SetOperationsDistribution::containsPercent() const
{
     return roundf((static_cast<float>(contains) / total()) *
    100.0 * 100.0) / 100;;
}

```

```
43 std::ostream& operator<< (std::ostream &out, const
       SetOperationsDistribution &sod)
44 {
45     out << "(add: " << sod.addPercent() << "%,"
46         << "remove: " << sod.removePercent() << "%,"
47         << "contains: " << sod.containsPercent() << "%)";
48     return out;
49 }
```

Листинг 11 – src/set/test_set/set_operations_distribution.h

```
#ifndef INC_SET_OPERATIONS_DISTRIBUTION_H
#define INC_SET_OPERATIONS_DISTRIBUTION_H

#include <string>
#include <sstream>
#include <cmath>

#include "../../log.h"
#include "../../rand/rand.h"


struct SetOperationsDistribution
{
    SetOperationsDistribution(uint a = 1, uint r = 1, uint c = 1);

    uint total() const;
    char pickRandom() const;

    float addPercent() const;
    float removePercent() const;
    float containsPercent() const;

    friend std::ostream& operator<< (std::ostream &out, const SetOperationsDistribution &sod);

    uint add;
    uint remove;
    uint contains;
};


typedef const SetOperationsDistribution& SodLinkT;

#endif //INC_SET_OPERATIONS_DISTRIBUTION_H
```