Teoria Współbieżności

Laboratorium 4 - Problem producentów-konsumentów

Magdalena Pastuła

Data laboratorium: 27.10.2020

1. Zadanie do wykonania.

Zadaniem było zaimplementowanie problemu producentów-konsumentów przy założeniu, że producenci dodają, a konsumenci zdejmują, z bufora losową liczbę elementów. Liczba ta nie może być większa niż połowa rozmiaru bufora.

2. Koncepcja rozwiązania.

Zaimplementowany bufor posiada metody `putMany` i `getMany`, które odpowiednio pobierają i zwracaja ArrayListę elementów. Metody te wywołują określoną w argumencie ilość razy metody `put()` i `get()`, które korzystają z monitorów Javy. Dodatkowo, klasa Buffer wewnętrznie korzysta z LinkedListy, która implementuje interfejs Queue.

Klasy Producent i Consument pobierają w konstruktorze semafory, które indykują, czy program powinien zakończyć już swoje działanie, na przykład w sytuacji, gdy wszyscy producenci już dodali elementy do bufora, ale część konsumentów czeka jeszcze na następne. Dodatkowo, klasy te przyjmują w konstruktorze jako argumenty liczbę serii dodawań lub wyjmowań elementów oraz minimalną i maksymalną liczbę dodawanych lub odejmowanych elementów.

Natomiast główny program jest napisany w postaci funkcji, do której jest przekazywane ograniczenie losowej liczby dodawań i wymowań elementów dla producentów i konsumentów, liczba samych producentów i konsumentów oraz rozmiar samego bufora. Następnie wykonywana jest seria pomiarów dla różnych parametrów i kreślone wykresy. Każdy parametr był zmieniany oddzielnie.

3. Implementacja i wyniki.

Metody `putMany` i `getMany` klasy Buffer:

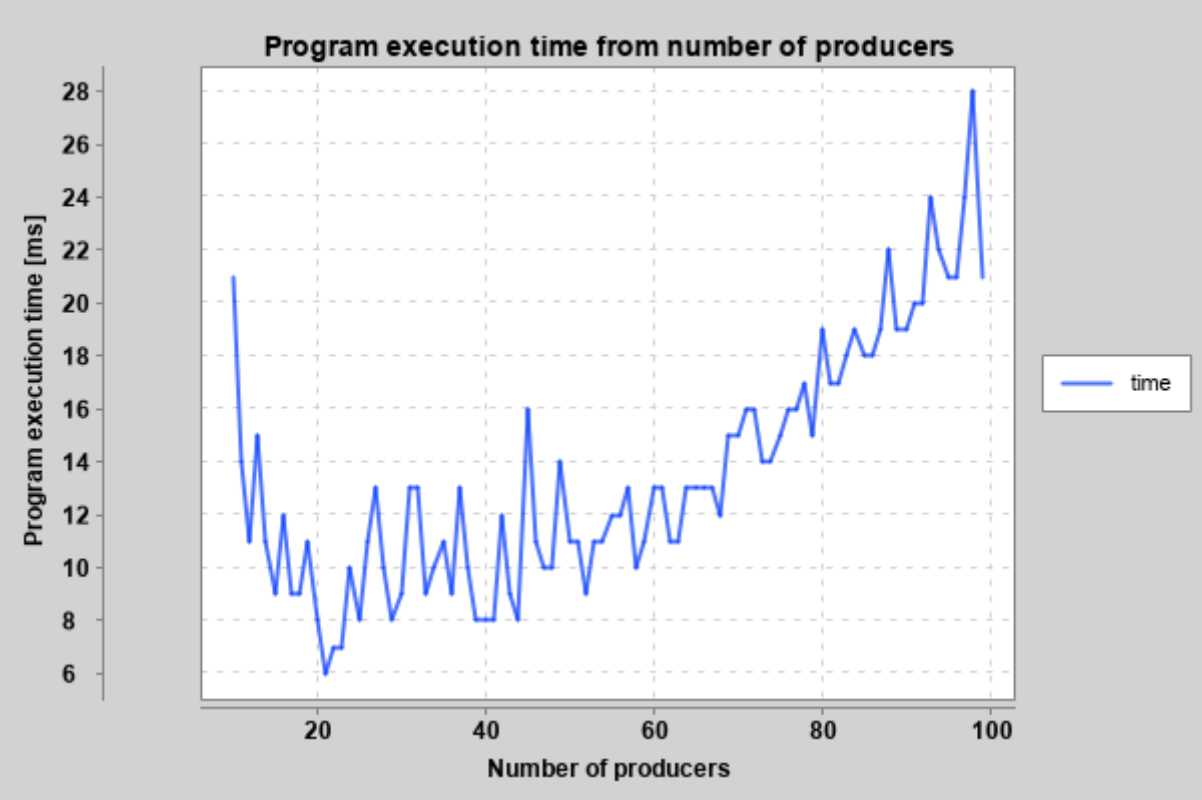
```
synchronized void putMany(ArrayList<Integer> list) {

        for (int i=0;i<list.size();i++) {
            put(list.get(i));
        }
    }

    synchronized ArrayList<Integer> getMany(int num) {
        ArrayList<Integer> to_return = new ArrayList<>(num);

        for (int i=0;i<num;i++) {
            to_return.add(this.get());
```

```
        }

        return to_return;
    }
```

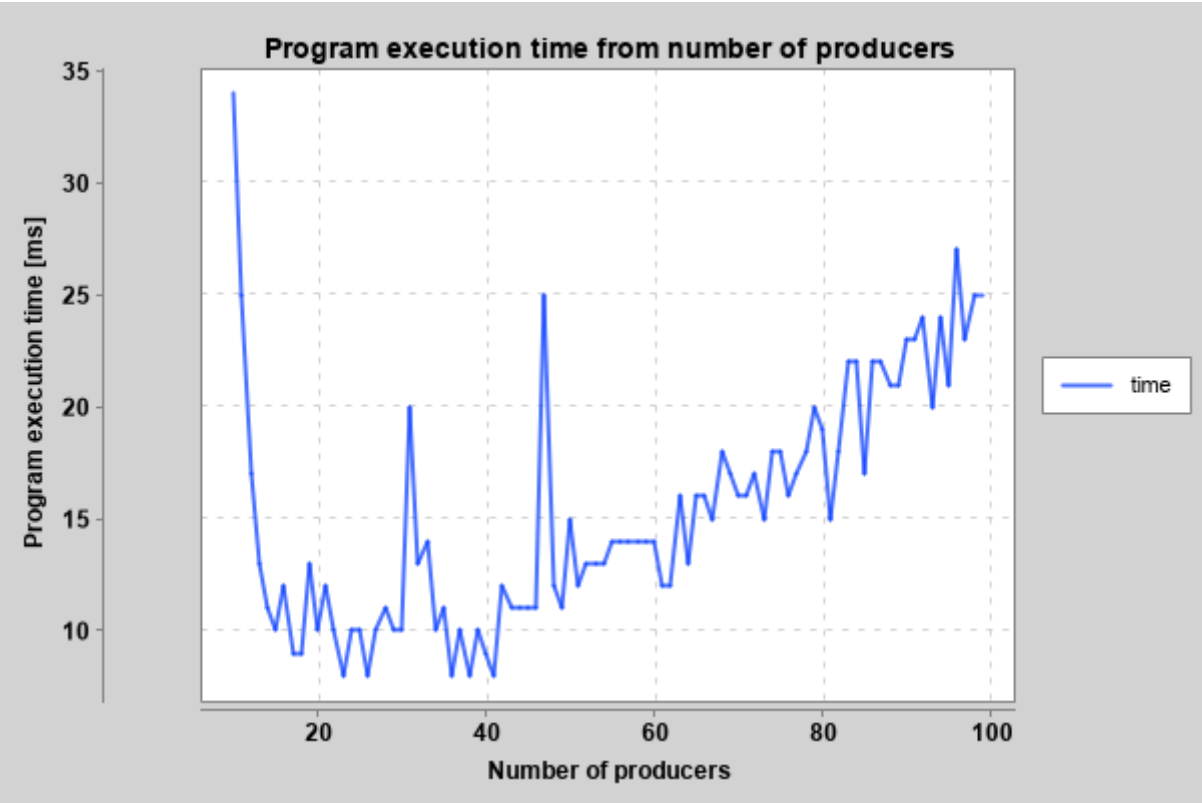
Funkcja main:

```
public static void main(String[] args) {
        int start_i = 10;
        int end_i = 100;

        double[] xData = new double[end_i - start_i];
        double[] yData = new double[end_i - start_i];

        for (int i=start_i;i<end_i;i++) {
            float elapsed_ms = program(49, 50, 10, i, 100);
            xData[i-start_i] = i;
            yData[i-start_i] = elapsed_ms;
            System.out.println("Elapsed time: " + elapsed_ms + " ms");
        }

        try {
            XYChart chart = QuickChart.getChart("Program execution time from
number of producers", "Number of producers", "Program execution time [ms]",
"time", xData, yData);
            new SwingWrapper(chart).displayChart();
            BitmapEncoder.saveBitmap(chart, "./prod_no_gen",
BitmapEncoder.BitmapFormat.PNG);
        } catch (Exception e) {}
    }
```

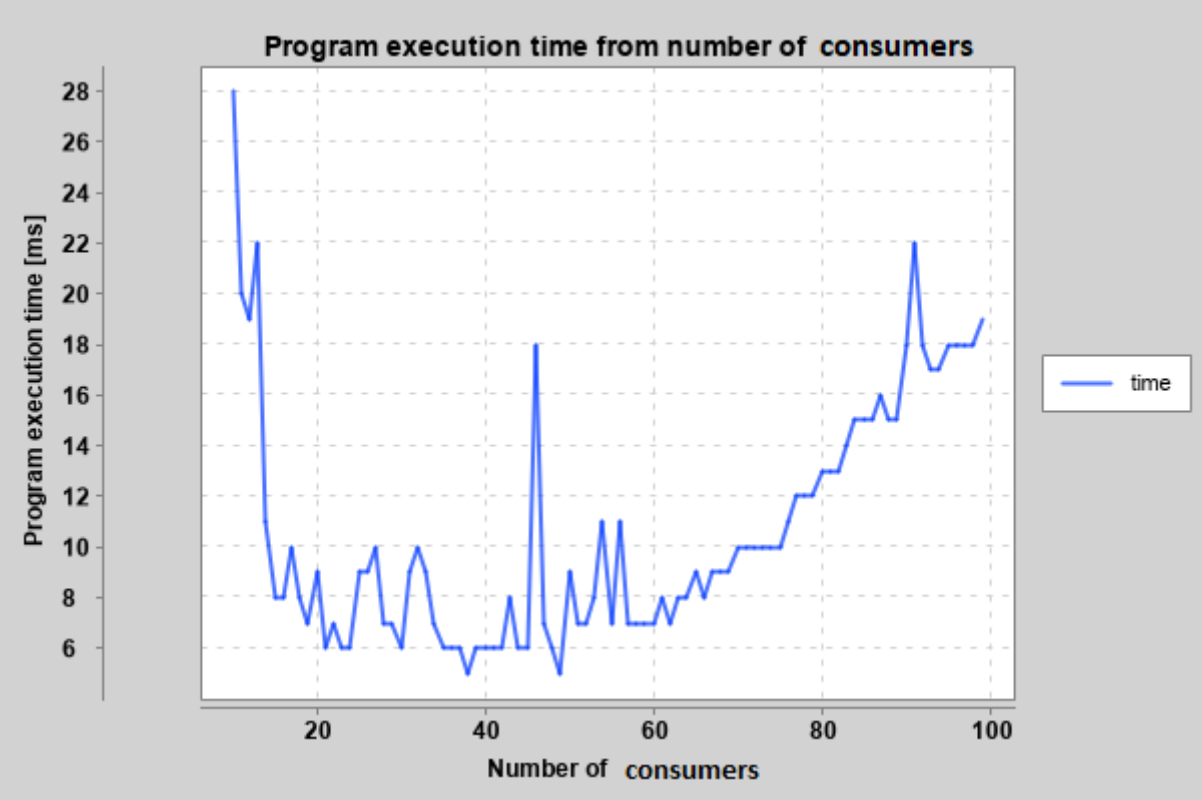
Liczba serii dla wszystkich testów była równa 10. Na początku sprawdzono czas wykonywania programu dla zmiennej liczby producentów (między 10 a 100) przy stałej liczbie konsumentów równej 10 i rozmiarze bufora równej 100. Losowość została ograniczona do przedziału (20, 50). Poniższy wykres ilustruje wyniki.



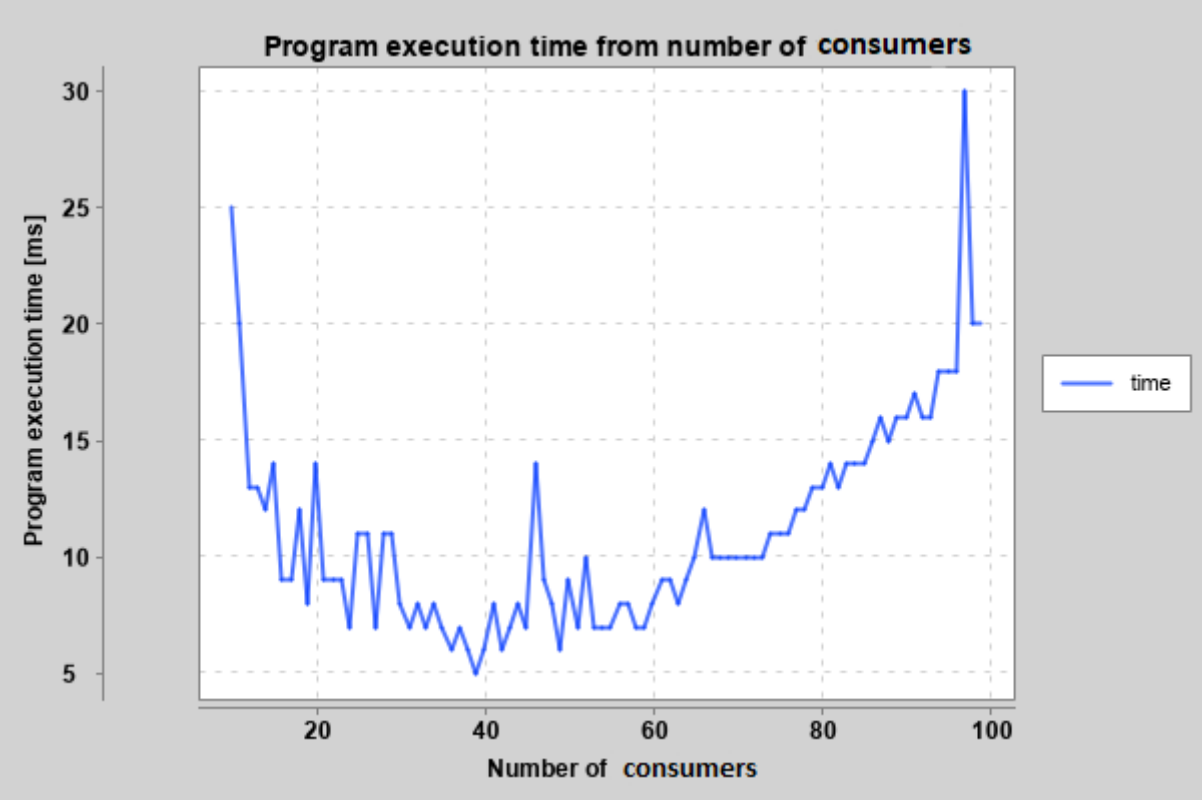
Dodatkowo, poniżej znajduje się wykres również dla zmiennej liczby producentów, ale bez losowości, dla liczby elementów w serii równej 50.



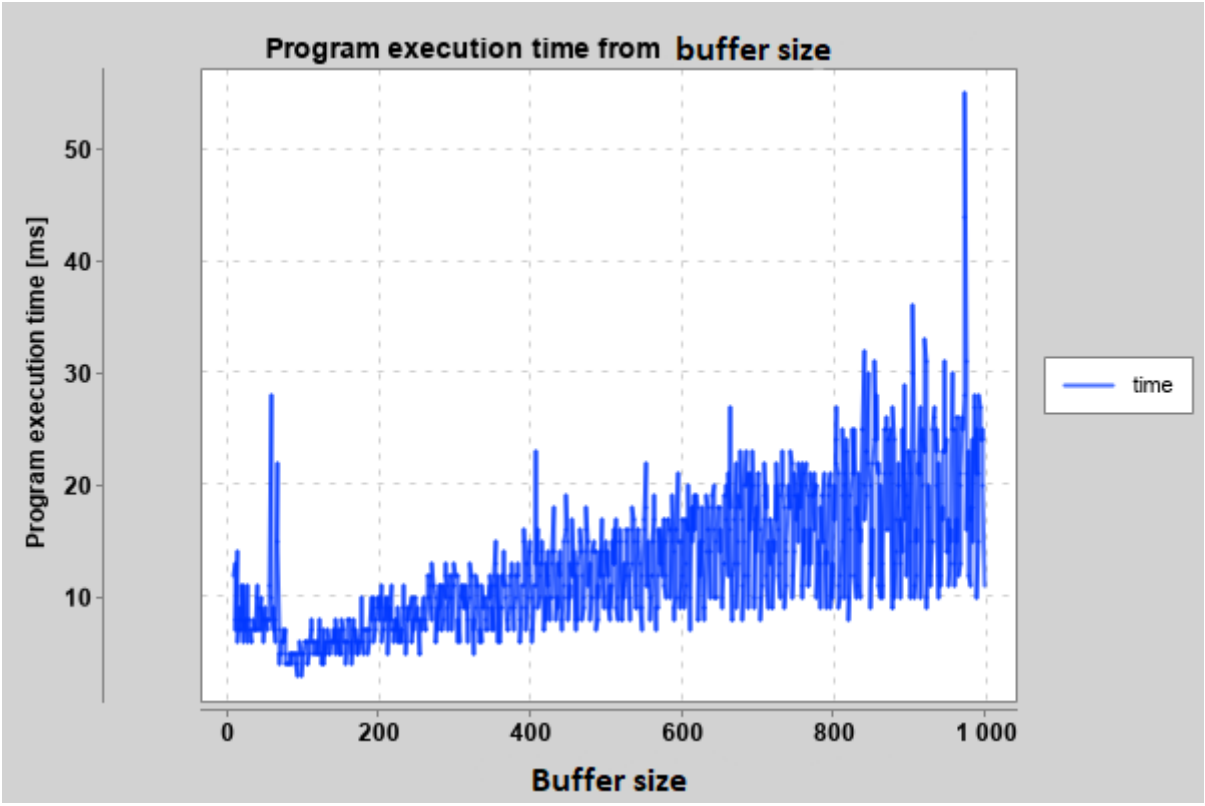
Podobnie przeprowadzono eksperyment dla zmiennej liczby konsumentów (między 10 a 100) przy stałych wartościach pozostałych parametrów i zakresu losowości jak poprzednio. Poniższy wykres przedstawia wyniki tego kroku.



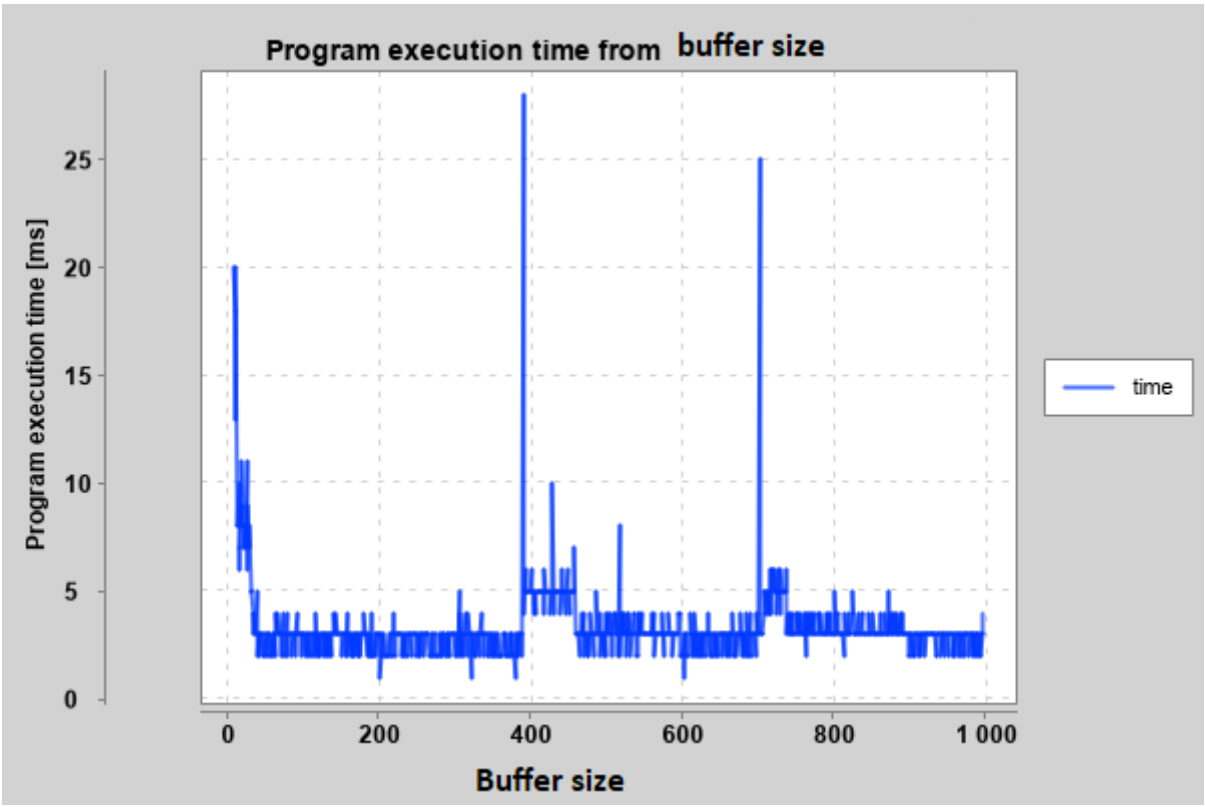
Również, jak poprzednio wykonano również pomiar dla stałej liczby elementów równej 50.



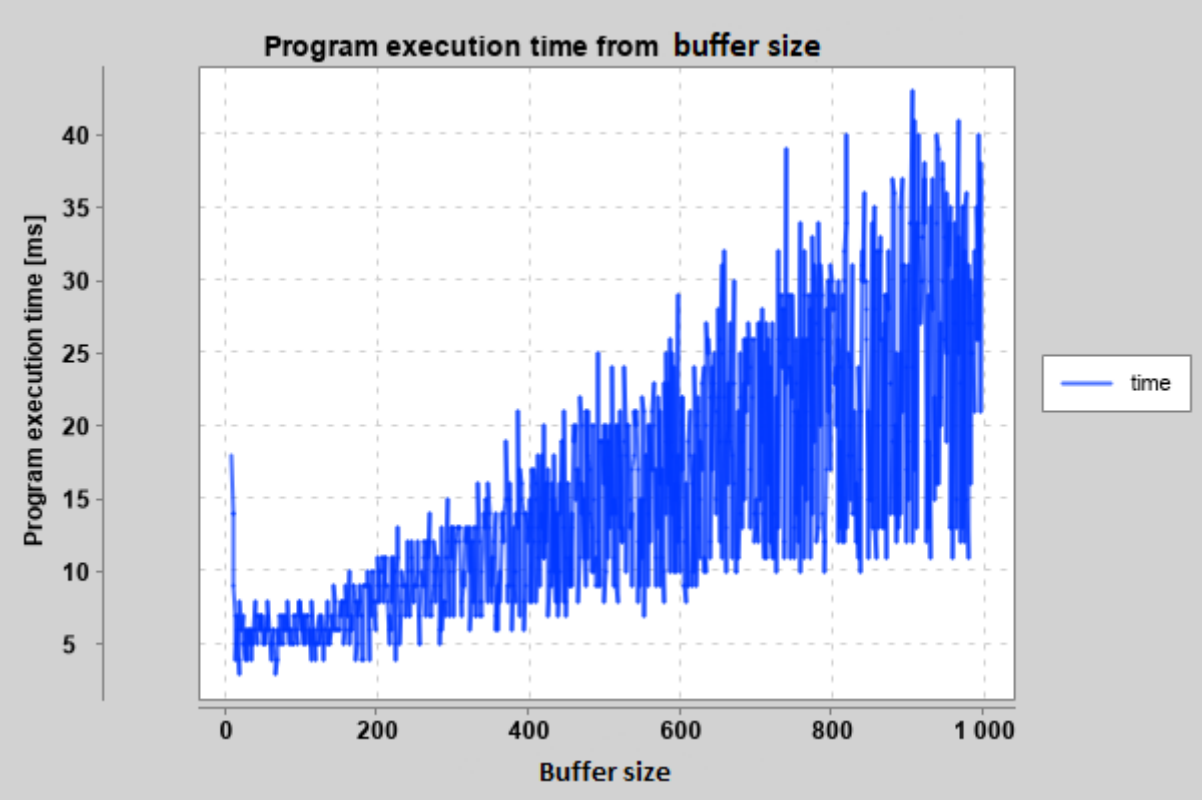
Następnie przeprowadzono eksperyment dla zmiennego rozmiaru bufora (między 10 a 1 000) przy stałej liczbie pozostałych parametrów. Losowość została ograniczona przedziału (i/5, i/2), gdzie i to długość bufora. Poniższy wykres przedstawia wyniki tego kroku.



Jak w poprzednich przypadkach, wykonano również pomiary przy braku losowości. Liczba dodawanych i wyjmowanych elementów z bufora wyniosła wtedy 5, czyli połowę najmniejszego rozmiaru bufora.



Na koniec wykonano również pomiary dla liczby elementów równej połowie rozmiaru każdego bufora.



4. Wnioski z ćwiczenia.

Czas wykonania programu definitywnie zależy od liczby producentów i konsumentów, a także od rozmiaru bufora. W przypadku zmiennej liczby producentów widać, że dla małych oraz dużych liczb czasy wykonania się zwiększają, zarówno przy ograniczonej losowości jak i jej braku. Minimum tych wykresów wydaje się być gdzieś w okolicy 20, czyli liczbie dwa razy większej niż liczba konsumentów.

Sytuacja wygląda bardzo podobnie dla zmiennej liczby konsumentów z tą różnicą, że minimum wydaje się być bardziej około 40, czyli liczbie cztery razy większej niż liczba producentów.

W przypadku zmiennego rozmiaru bufora, praktycznie dla wszystkich przypadków, im większy bufor, tym większy czas wykonania. Wiąże się to z faktem, że dla tych przypadków, wraz ze wzrostem rozmiaru bufora, rośnie liczba elementów do niego wstawianych i wyciąganych. Natomiast w momencie, gdy liczba elementów jest stała i względnie mała, co ma miejscie w drugim przypadku, czas wykonania jest praktycznie stały, pomijając pojedyncze przypadki. Można również zauważyć większe czasy na początku wykresu, które wynikają z faktu, że dla nich liczba dodawanych elemetów ma podobny rząd wielkości jak rozmiar samego bufora.