TEORIA WSPÓŁBIEŻNOŚCI
LABORATORIUM 4

Problem pięciu filozofów.

Bartosz Konieczny

Wydział Informatyki
Grupa 5
Wtorek 8:00

data oddania:
06.11.2025

Spis treści

Ćwiczenie

Celem ćwiczenia było porównanie różnych rozwiązań problemu pięciu filozofów w języku Java. Dokładniej przyjrzeliśmy się rozwiązaniom:

- naiwne - filozof czeka, aż zwolni się lewa pałeczka, podnosi ją i postępuje tak samo z prawą,
- z możliwością zagłodzenia - filozof podnosi pałeczki jednocześnie, jeśli obie są wolne,
- asymetryczne - „parzysty" filozof podnosi prawą pałeczkę, a „nieparzysty" lewą,
- z arbitrem - arbiter pilnuje, aby co najwyżej $n - 1$ filozofów jednocześnie konkurowało.

Implementacja problemu

Zaimplementowałem problem tworząc klasy odpowiadające pałeczką (dalej jako widelce), filozofom oraz odpowiednim rozwiązaniom.

2.1 Klasa Fork

Obiekty klasy `fork`, to tak naprawdę semafory, które przechowywane są w jednej liście o długości równej liczbie filozofów.

```
public class Fork {
    private boolean isUsed;

    public Fork(){
        isUsed = false;
    }

    public boolean isUsed() {
        return isUsed;
    }

    public synchronized void useFork(){
        while (isUsed){
            try {
                wait();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        isUsed = true;
    }

    public synchronized void relaseFork(){
        isUsed = false;
        notifyAll();
    }
}
```

2.2 Klasa Philosopher

W tej klasie najważniejsze są metody `run`, `think` i `eat`, które odpowiadają za przebieg symulacji (naprzemienne myślenie i jedzenie), myślenie przez stały lub losowy okres czasu oraz jedzenie wykorzystujące odpowiednie rozwiązanie.

2.3 Rozwiązanie naiwne

Metoda `startEating` zwraca wartość logiczną w zależności, czy udało się zająć widelce, szczególne znaczenie będzie to miało w przypadku rozwiązania z możliwym zagłodzeniem.

```
public boolean startEating(Philosopher philosopher) {
      Fork leftFork = forks.getFork(philosopher.getLeftForkIndex());
      leftFork.useFork();

      Fork rightFork = forks.getFork(philosopher.getRightForkIndex());
      rightFork.useFork();

      return true;
  }
```

2.4 Rozwiązanie z możliwością zagłodzenia

W tym przypadku jeśli nie jest możliwe zajęcie prawego widelca metoda zwróci `false`, aby możliwe było wznowienie mierzenia czasu w kolejnej próbie.

```
public boolean startEating(Philosopher philosopher) {
      Fork leftFork = forks.getFork(philosopher.getLeftForkIndex());
      Fork rightFork = forks.getFork(philosopher.getRightForkIndex());

      leftFork.useFork();
      if (!rightFork.isUsed()){
          rightFork.useFork();
      } else {
          leftFork.relaseFork();
          return false;
      }

      return true;
  }
```

2.5 Rozwiązanie asymetryczne

W zależności od indeksu filozofa wykonujemy akcje w odwrotnej kolejności.

```
  public boolean startEating(Philosopher philosopher) {
      Fork leftFork = forks.getFork(philosopher.getLeftForkIndex());
      Fork rightFork = forks.getFork(philosopher.getRightForkIndex());

      if(philosopher.getIndex() % 2 == 0){
          rightFork.useFork();

          leftFork.useFork();
      } else {
          leftFork.useFork();

          rightFork.useFork();
      }
      return true;
  }
```

2.6 Rozwiązanie z arbitrem

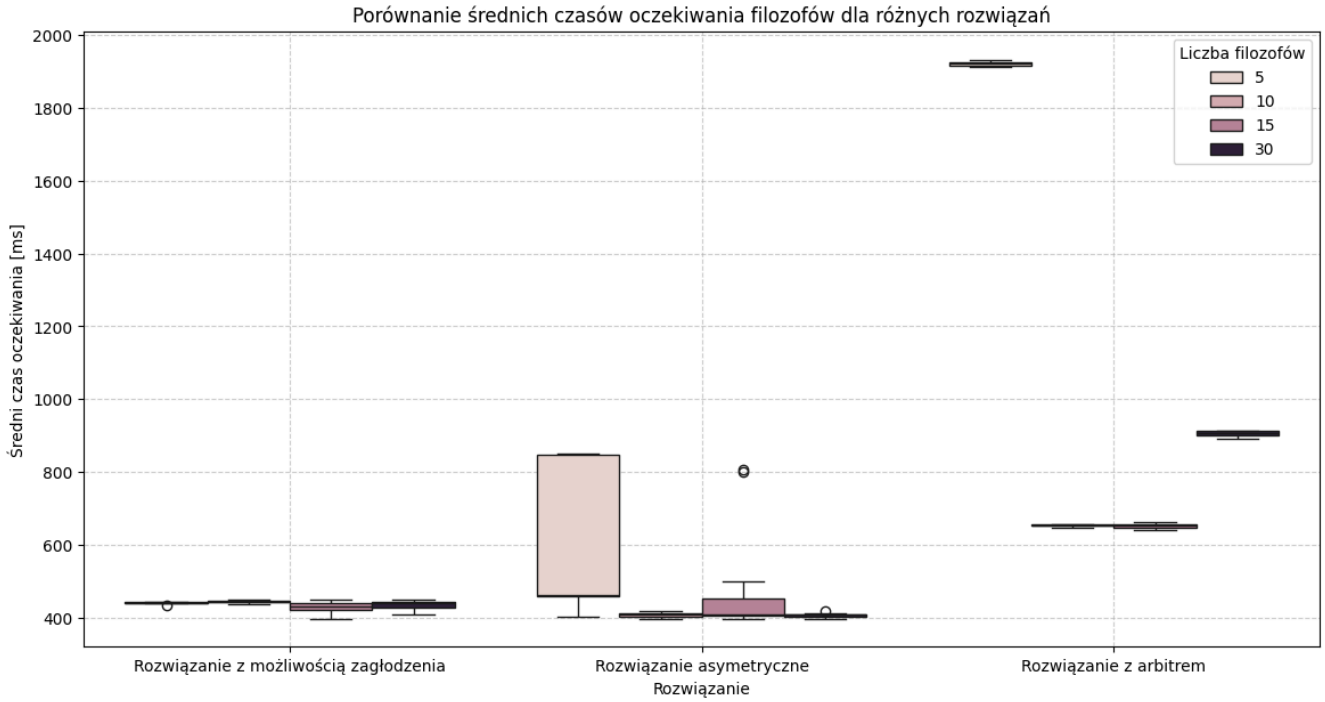
W tym przypadku zakładamy istnienie arbitra, który zlicza ilu filozofów próbuje użyć widelców, a następnie wykonujemy zwykłe naiwne rozwiązanie. Arbitrem jest `AtomicInteger` o nazwie `numberOfEatingPhilosophers` (typów prostych nie można zsynchronizować).

```
public boolean startEating(Philosopher philosopher) {
    Fork leftFork = forks.getFork(philosopher.getLeftForkIndex());
    Fork rightFork = forks.getFork(philosopher.getRightForkIndex());

    synchronized (numberOfEatingPhilosophers) {
        while (numberOfEatingPhilosophers.get() >= maximumNumberOfEatingPhilosophers) {
            try {
                numberOfEatingPhilosophers.wait();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        numberOfEatingPhilosophers.incrementAndGet();
    }

    leftFork.useFork();

    rightFork.useFork();

    return true;
}
```

Wyniki eksperymentu

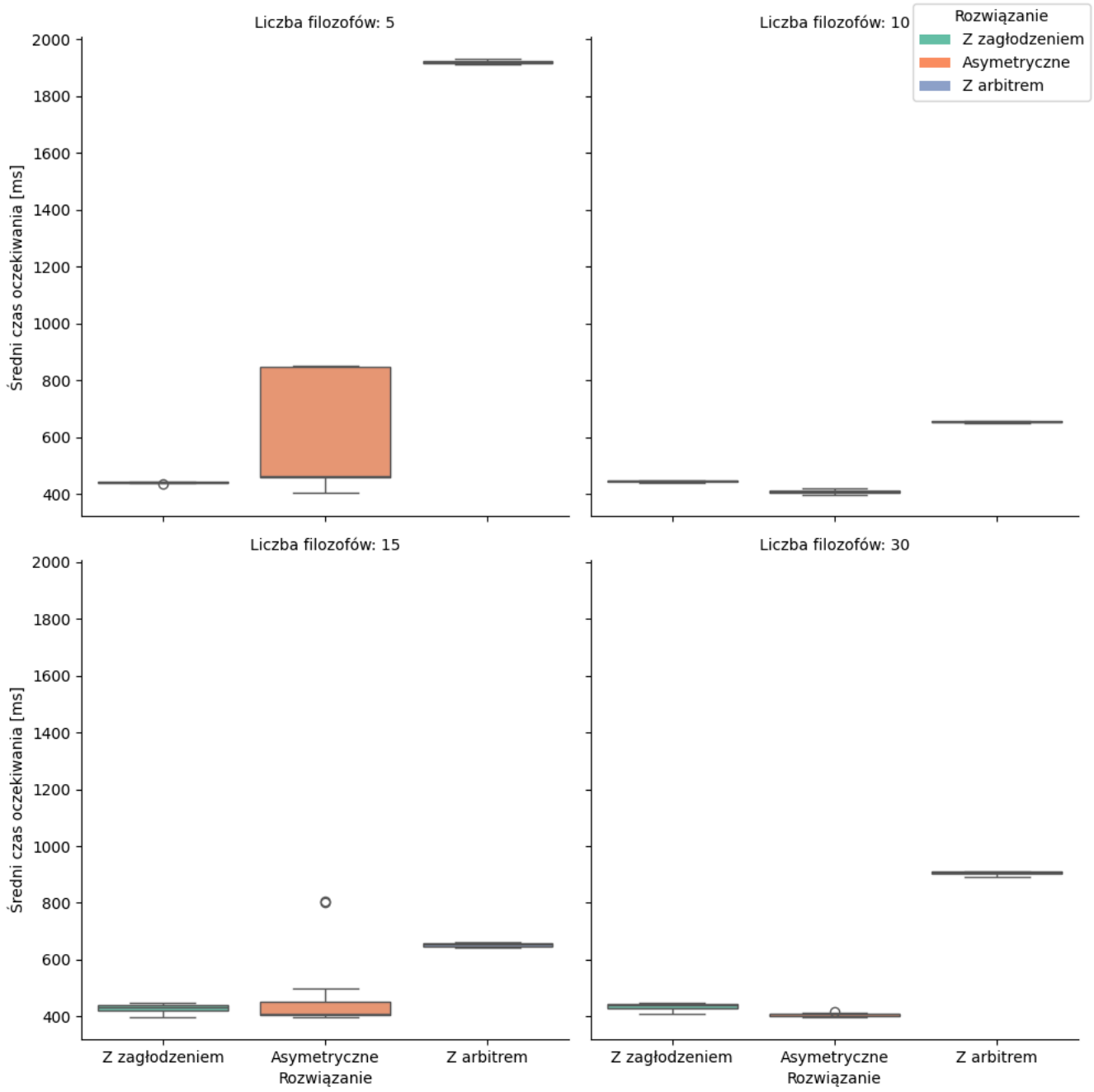
Wyniki działań programu zostały zapisane w pliku `data.csv` i opracowane w języku Python z wykorzystaniem bibliotek pandas, matplotlib i seaborn.

Eksperyment przeprowadziłem na wszystkich rozwiązaniach, pomijając rozwiązanie naiwne, które prowadziło do zakleszczenia się (zawsze w przypadku stałego czasu jedzenia, często w przypadku zmiennego czasu jedzenia).



Rys. 1: Porównanie średnich czasów oczekiwania filozofów dla różnych rozwiązań

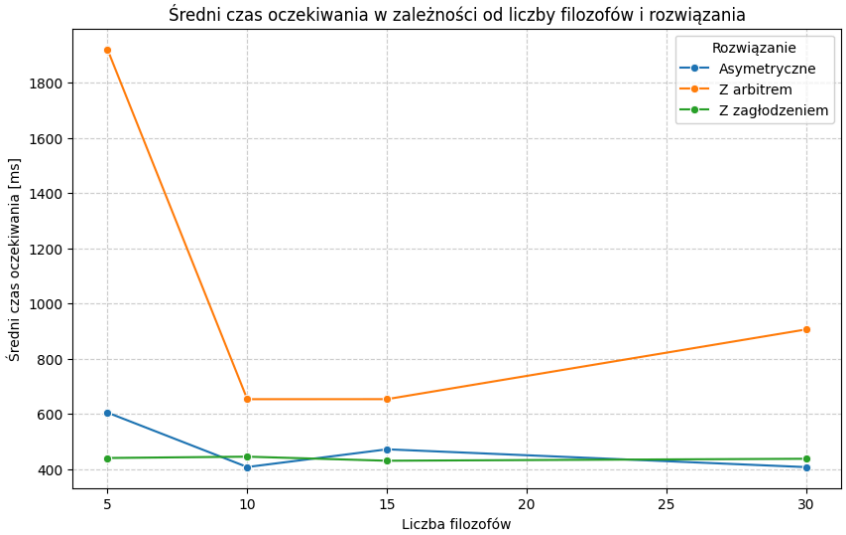
To co rzuca się w oczy na pierwszym zestawie wykresów (Rys. 1) to fakt, że dla rozwiązania z możliwością zagłodzenia zagłodzenie wystąpiło jedynie w najmniejszym przypadku, co sugerują wartości odstające. Rozwiązanie asymetryczne w pierwszym przypadku doprowadziło do bardzo rozrzuconych wartości dla jednych filozofów, co oznacza, że niektórzy czekali zdecydowanie dłużej niż inni, co ciekawe dla 15 filozofów paru z nich czekało znacznie dłużej od reszty, co sugeruje, że mogli być głodzeni. W przypadku rozwiązania z arbitrem zauważymy, że to rozwiązanie jest zdecydowanie lepsze dla dużej grupy filozofów oraz gwarantuje, że każdy filozof będzie czekał tyle samo.



Rys. 2: Porównanie średnich czasów oczekiwania filozofów dla rozwiązań z daną liczbą filozofów.

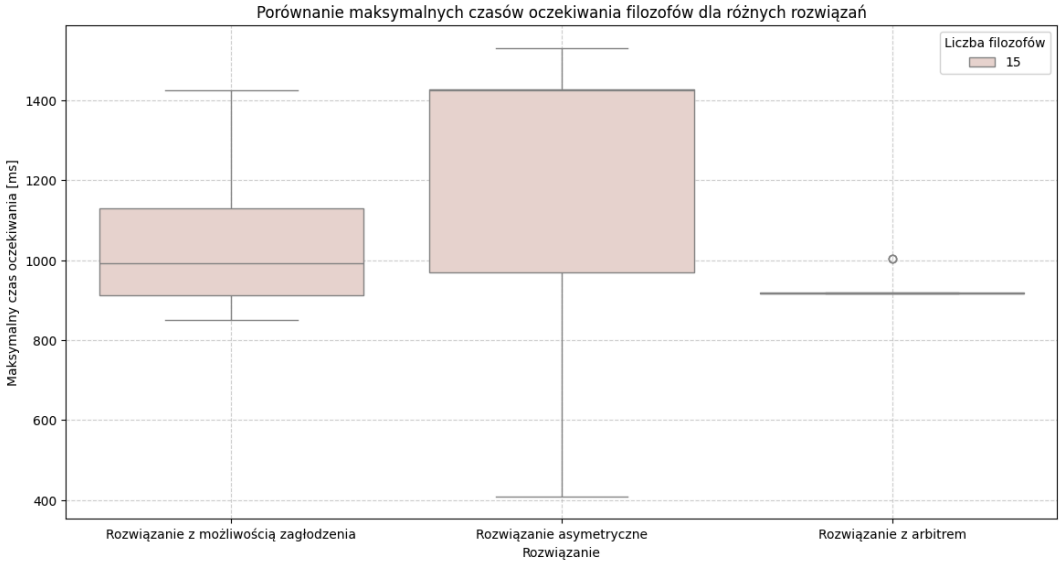
Na kolejnym wykresie (Rys. 2) widzimy, że rozwiązanie z arbitrem zawsze daje „najcieńsze" pudełko, a więc jest najbardziej sprawiedliwe ale niekoniecznie najbardziej efektywne. Przez użycie średniej ciężko zaobserwować zagłodzenie, tutaj lepiej sprawdziłaby się miara

maksimum, ponieważ gdy inni filozofowie się najedzą, ci którzy byli głodzeni mogą znacznie przyspieszyć swoje oczekiwanie (a eksperyment był przerywany po n iteracjach).



Rys. 3: Średni czas oczekiwania w zależności od liczby filozofów i rozwiązania

Wykres (Rys. 3) pokazuje nam, że mimo wzrostu liczby filozofów średni czas oczekiwania różni się bardzo niewiele, a nawet potrafi się zmniejszyć dla rozwiązania z arbitrem, co wynika z faktu, że dużo więcej osób może jeść równocześnie przy większej liczbie filozofów.



Rys. 4: Porównanie maksymalnych czasów oczekiwania filozofów dla różnych rozwiązań

Na koniec przygotowałem test prezentujący maksymalny czas i tutaj (Rys. 4) dokładnie widać wspomniane wcześniej wnioski. Dla rozwiązania z arbitrem czas w prawie każdym wypadku jest taki sam (pomijając jedną wartość odstającą). W przypadku rozwiązania asymetrycznego mamy bardzo duże robicie, które najprawdopodobniej wynika z „wnętrza" tego rozwiązania (po ustaleniu, który widelec weźmiemy pierwszy wykonujemy zwykłe naiwne rozwiązanie z lepszymi parametrami, gwarantuje to brak zakleszczenia, ale zezwala na długie oczekiwanie na zwolnienie). Ostatnim rozpatrywanym rozwiązaniem było to z możliwością zagłodzenia, co widać po rozproszeniu danych, wąsy w górę znacznie przewyższają średnią znajdującą się wewnątrz pudełka.