Sprawozdanie

November 19, 2021

1 Problem pięciu filozofów

Adrianna Łysik

Importy

[1]:
```
import csv
import matplotlib.pyplot as plt
import numpy as np
```

Urządzenie MacBook Air z procesorem M1, pamięć 8GB.

1.1 Implementacja - programowanie wielowątkowe

Zaimplementuj rozwiązanie problemu pięciu filozofów w Javie, wykorzystując mechanizm semaforów, w następujących wariantach: - Wariant z możliwością zagłodzenia - Wariant z arbitrem

1.1.1 Wykonanie zadania

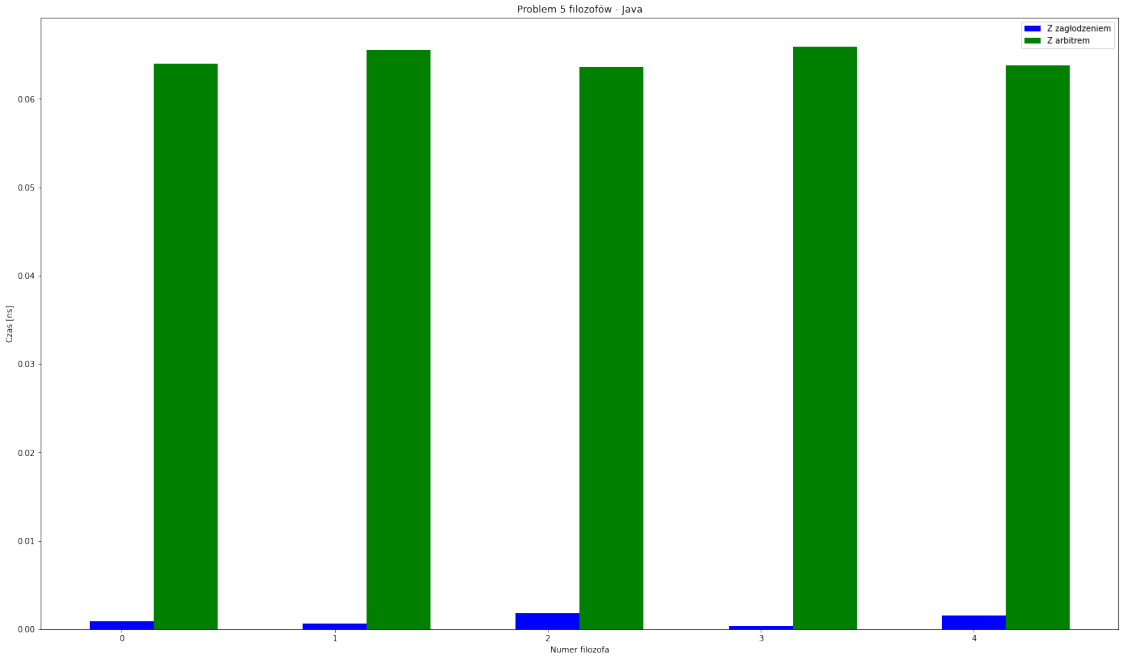
- Program został uruchomiony dla 5, 10 oraz 15 filozofów.
- Dla wszystkich wariantów, każdy filozof "zjada" 100 posiłków.
- Pliki z czasami wykonania posiadają budowę: numer filozofa ; średni czas oczekiwania.
- Czas zwrócony został w nanosekundach, a następnie przeskalowany na milisekundy.

[2]:
```
def open_and_return_data1(filename):
    num = []
    time = []
    with open(filename) as file:
        reader = csv.reader(file, delimiter = ';')
        for row in reader:
            num.append(int(row[0]))
            time.append(int(row[1])/10e6)
    return num, time
```

Java - problem 5 filozofów

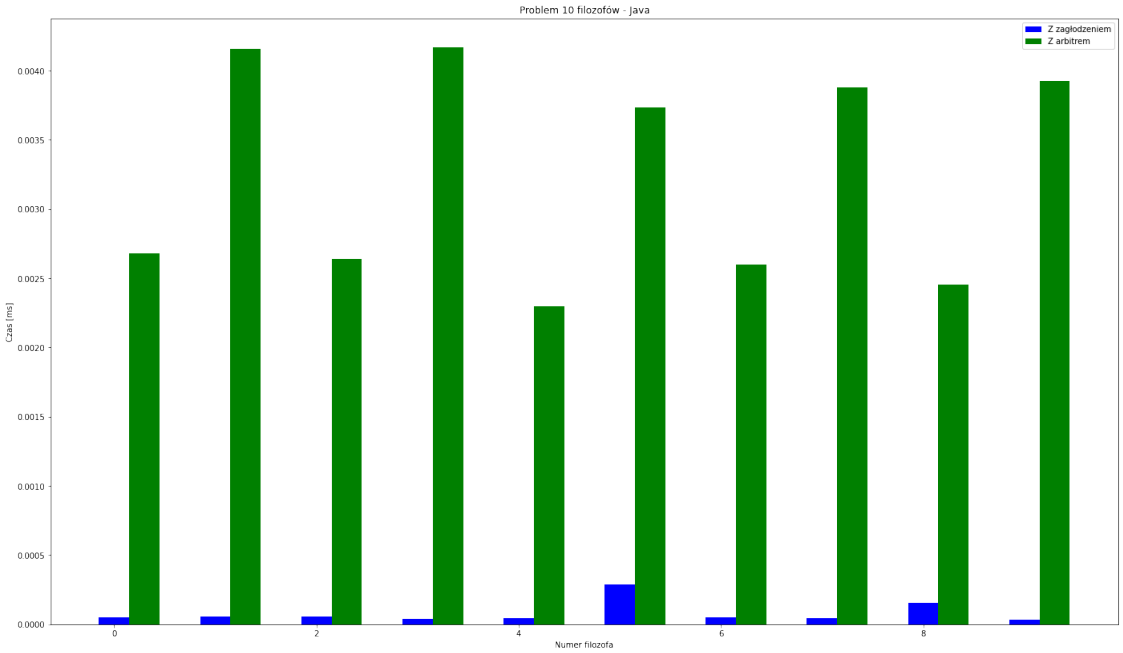
[3]:
```
number1, time1 = open_and_return_data1('Times/greedy5-java.txt')
number2, time2 = open_and_return_data1('Times/waiter5-java.txt')

fig = plt.figure()
```

```
fig.set_size_inches(18.5, 10.5)
ax = fig.add_axes([0,0,1,1])
ax.bar(np.array(number1) + 0.0, time1, color = 'b', width = 0.3)
ax.bar(np.array(number2) + 0.3, time2, color = 'g', width = 0.3)
ax.legend(labels=['Z zagłodzeniem', 'Z arbitrem'])
ax.set_title("Problem 5 filozofów - Java")
ax.set_ylabel("Czas [ns]")
ax.set_xlabel("Numer filozofa")
plt.show()
```



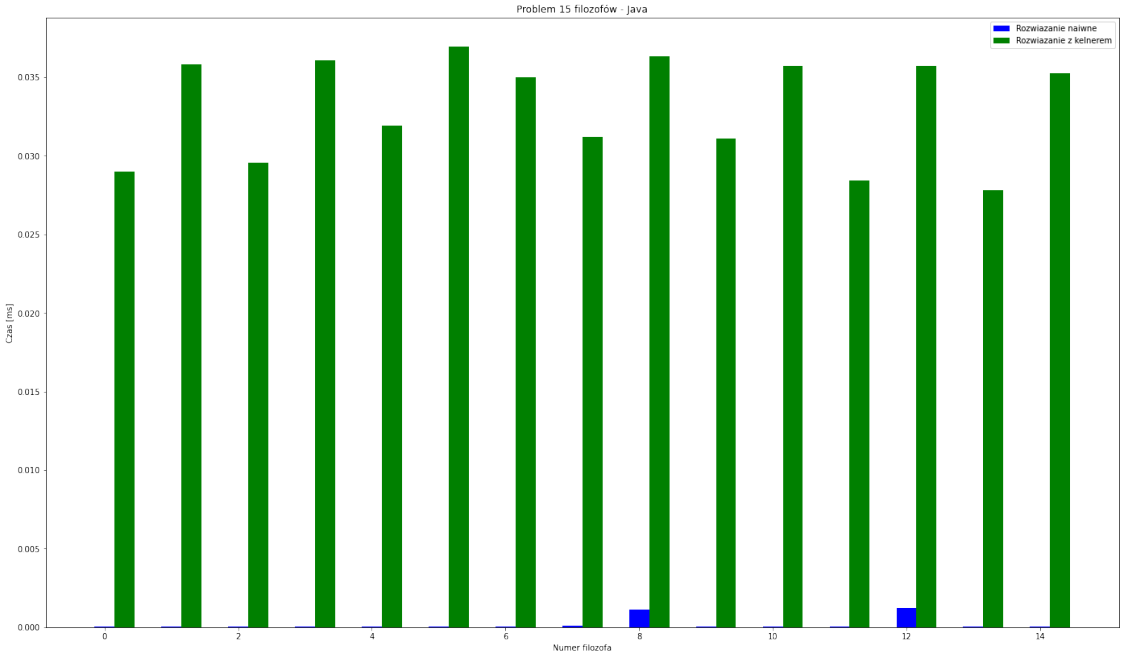
Java - problem 10 filozofów

[4]:
```
number1, time1 = open_and_return_data1('Times/greedy10-java.txt')
number2, time2 = open_and_return_data1('Times/waiter10-java.txt')

fig = plt.figure()
fig.set_size_inches(18.5, 10.5)
ax = fig.add_axes([0,0,1,1])
ax.bar(np.array(number1) + 0.0, time1, color = 'b', width = 0.3)
ax.bar(np.array(number2) + 0.3, time2, color = 'g', width = 0.3)
ax.legend(labels=['Z zagłodzeniem', 'Z arbitrem'])
ax.set_title("Problem 10 filozofów - Java")
ax.set_ylabel("Czas [ms]")
ax.set_xlabel("Numer filozofa")
plt.show()
```



Java - problem 15 filozofów

[5]:
```
number1, time1 = open_and_return_data1('Times/greedy15-java.txt')
number2, time2 = open_and_return_data1('Times/waiter15-java.txt')

fig = plt.figure()
fig.set_size_inches(18.5, 10.5)
ax = fig.add_axes([0,0,1,1])
ax.bar(np.array(number1) + 0.0, time1, color = 'b', width = 0.3)
ax.bar(np.array(number2) + 0.3, time2, color = 'g', width = 0.3)
ax.legend(labels=['Rozwiazanie naiwne', 'Rozwiazanie z kelnerem'])
ax.set_title("Problem 15 filozofów - Java")
ax.set_ylabel("Czas [ms]")
ax.set_xlabel("Numer filozofa")
plt.show()
```



1.1.2 Wnioski

W rozwiązaniu z arbitrem filozofowie pytają go o pozwolenie przed wzięciem widelca, ponieważ pilnuje on, aby jednocześnie konkurowało o widelec maksymalnie $N - 1$ filozofów. Dzięki temu unikamy zakleszczeń.

Czas oczekiwania na widelec jest jednak wysoki w porównaniu do rozwiązania z zagłodzeniem, które jest mniej "sprawiedliwe", gdyż jeśli któryś sąsiad będzie jadł, to nigdy oba widelce nie będą wolne.

Patrząc na wykresy można zauważyć, że żaden z filozofów nie czekał (średnio) dużą ilość czasu odbiegającą o innych, co świadczy o tym, że mógł nie zostać zagłodzony. Wpływ na to ma fakt, że chcemy, aby każdy filozof zjadł konkretną liczbę posiłków, więc jeśli jeden filozof zostanie głodzony przez pewnien okres czasu, to po skończeniu jedzenia przez pozostałych (lub po prostu sądsiadów), będzie mógł już jeść pozostałe posiłki bez czekania.

1.2 Implementacja - programowanie asynchroniczne

Korzystając z zadanego szkieletu programu w Node.js: - Zaimplementuj "naiwny" algorytm. - Zaimplementuj rozwiązanie asynchroniczne. - Zaimplementuj rozwiązanie z arbitrem. - Zaimplementuj rozwiązanie z jednoczesnym podnoszeniem widelców.

1.2.1 Wykonanie zadania

- Z uwagi na jednowątkowy sposób wykonania programu, skorzystałam z funkcji `setTimeout(handler, time)`, która wykonuje funkcję `handler` po czasie time.
- Program został uruchomiony dla 5 oraz 10 filozofów.

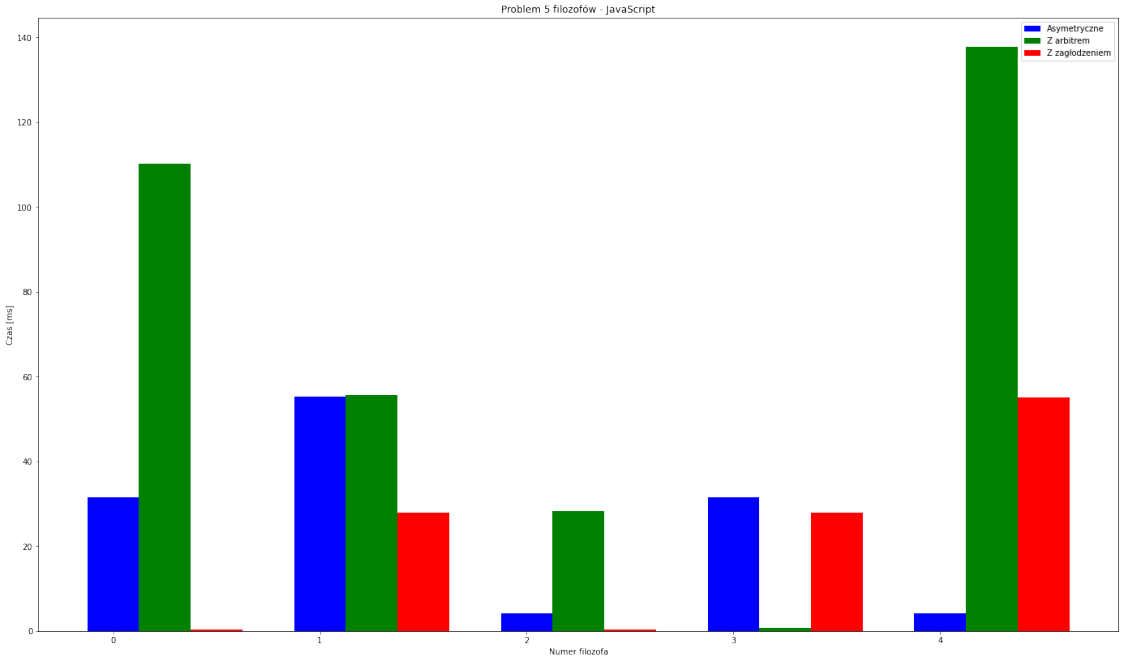
- W sprawozdaniu nie jest zawarty czas wykonania dla wersji naiwnej, ponieważ powoduje zakleszczenia.
- Dla ou wariantów, każdy filozof "zjada" 30 posiłków.
- Pliki z czasami wykonania posiadają budowę: numer filozofa ; czas czekania na jeden posiłek. Czasy następnie zostały zsumowane i obliczony został średni czas dla każdego filozofa.
- Czas zwrócony został w milisekundach.

[6]:
```
def open_and_return_data2(filename):
    num = []
    time = []
    data = dict()
    with open(filename) as file:
        reader = csv.reader(file, delimiter = ';')
        for row in reader:
            num.append(int(row[0]))
            time.append(int(row[1]))

    for i in num:
        if i not in data:
            data[i] = 0
    for i in range(len(num)):
        data[num[i]] += time[i]/len(num)

    result_number = []
    for i in data.keys():
        result_number.append(i)

    result_time = []
    for i in result_number:
        result_time.append(data[i])


    return result_number, result_time
```

JavaScript - problem 5 filozofów

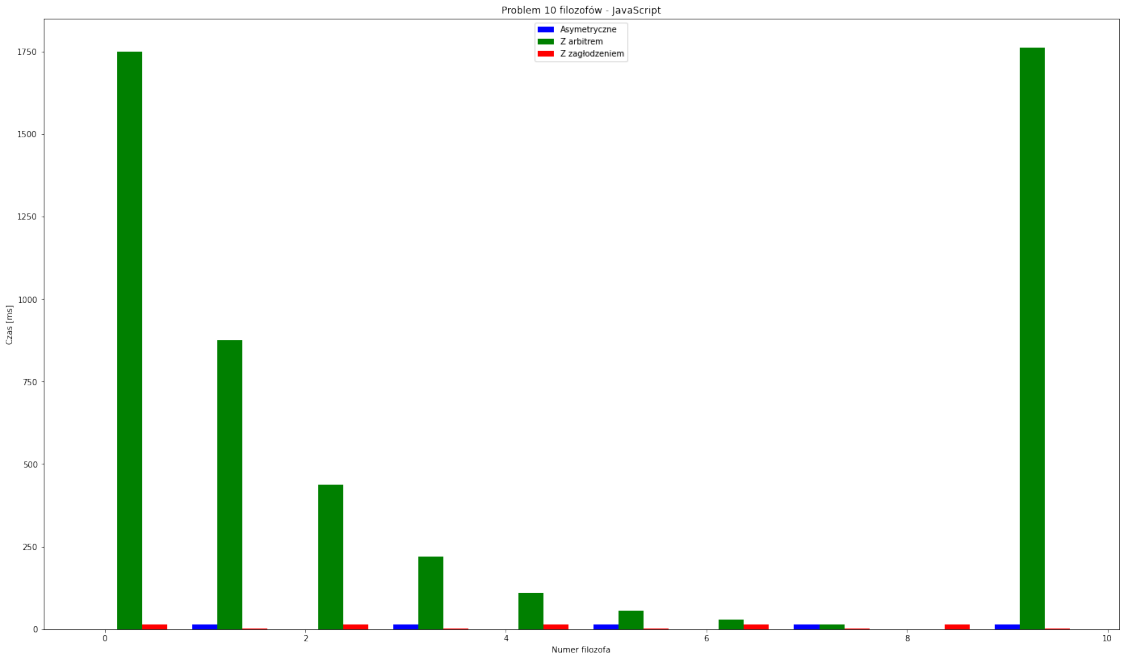
[7]:
```
number1, time1 = open_and_return_data2('Times/asyn5-js.txt')
number2, time2 = open_and_return_data2('Times/waiter5-js.txt')
number3, time3 = open_and_return_data2('Times/twoforks5-js.txt')

fig = plt.figure()
fig.set_size_inches(18.5, 10.5)
ax = fig.add_axes([0,0,1,1])
ax.bar(np.array(number1) + 0.0, time1, color = 'b', width = 0.25)
ax.bar(np.array(number2) + 0.25, time2, color = 'g', width = 0.25)
ax.bar(np.array(number3) + 0.5, time3, color = 'r', width = 0.25)
ax.legend(labels=['Asymetryczne', 'Z arbitrem', 'Z zagłodzeniem'])
ax.set_title("Problem 5 filozofów - JavaScript")
```

```
ax.set_ylabel("Czas [ms]")
ax.set_xlabel("Numer filozofa")
plt.show()
```



JavaScript - problem 10 filozofów

[10]:
```
number1, time1 = open_and_return_data2('Times/asyn10-js.txt')
number2, time2 = open_and_return_data2('Times/waiter10-js.txt')
number3, time3 = open_and_return_data2('Times/twoforks10-js.txt')

fig = plt.figure()
fig.set_size_inches(18.5, 10.5)
ax = fig.add_axes([0,0,1,1])
ax.bar(np.array(number1) + 0.0, time1, color = 'b', width = 0.25)
ax.bar(np.array(number2) + 0.25, time2, color = 'g', width = 0.25)
ax.bar(np.array(number3) + 0.5, time3, color = 'r', width = 0.25)
ax.legend(labels=['Asymetryczne', 'Z arbitrem', 'Z zagłodzeniem'])
ax.set_title("Problem 10 filozofów - JavaScript")
ax.set_ylabel("Czas [ms]")
ax.set_xlabel("Numer filozofa")
plt.show()
```



1.2.2 Wnioski

Rozwiązanie z arbitrem ponownie daje najwyższe czasy czekania. Przewaga dłuższego czasu rośnie dla tego rozwiązania dla większej ilości filozofów, czego nie dostrzegam dla pozostałych rozwiązań. Wpływ na to ma fakt, że ilość filozofów nie zmienia szans na to czy uda nam się dostać widelec, ponieważ jest to zależne od sąsiadów, których dalej jest dwóch.

Można również zauważyć, że dla większej liczby filozofów najdłuszszy średni czas oczekiwania ma filozof pierwszy i ostatni.

1.3 Krótkie podsumowanie

Zarówno dla podejścia wielowątkowego jak i asynchroniczego najdłuszszy średni czas oczekiwania ma rozwiązanie z arbitrem.

Dla podejścia wielokątkowego uzyskiwane są zdecydowanie krótsze czasy niż dla podejścia asynchronicznego.

W podejściu Javowym mamy do czynienia z synchronizacją wielu wątków, natomiast dla JavaScriptu wszystko odbywa się w ramach jednego wątku.

Dla pierwszego podejścia mamy do czynienia z wieloma wątkami, natomiast dla drugiego funkcje czekają w kolejce i wykonywane są jedna po drugiej. Co pokazuje nam, że więcej mechanizmów synchronizacji niekoniecznie oznacza dłuższy czas wykonania programu.