1. Jak uruchomić program?

Należy rozpakować dostarczony plik .zip, a następnie importować projekt *philosophers* do środowiska Intellij Idea 2025.2 (File->Open...). Do wykonania ćwiczenia użyto Javy z SDK Oracle OpenJDK 25.

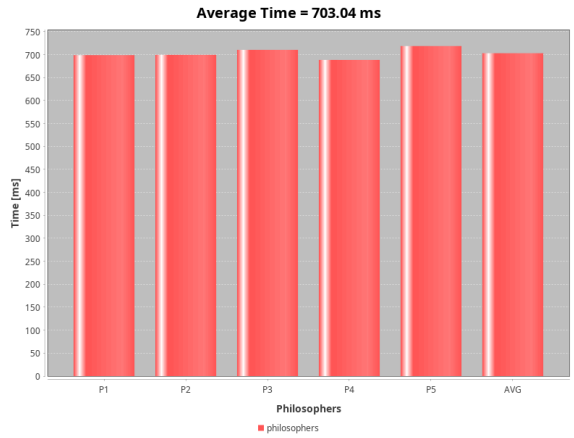
2. Symulacja

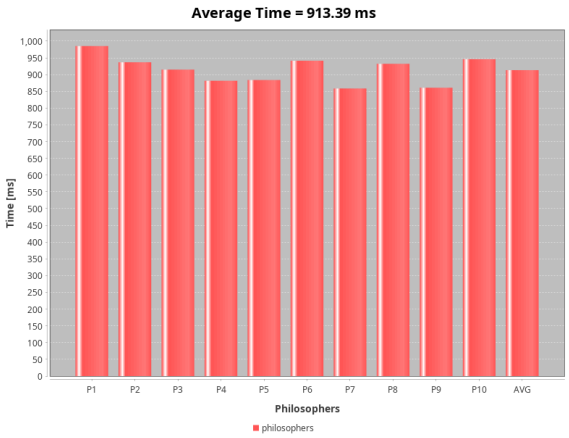
W celu rozwiązania zadania oraz prezentacji wyników przygotowano następujące klasy:

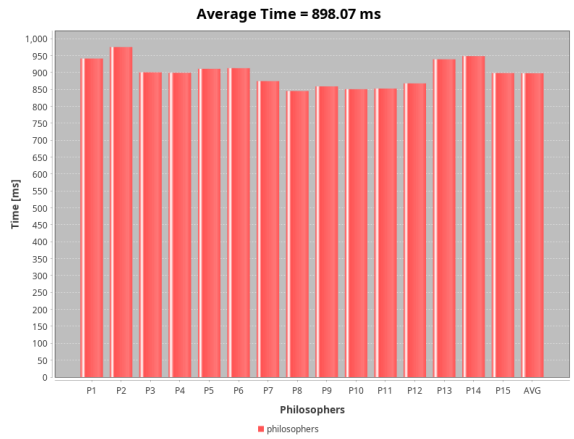
- `Main` - klasa główna.
- `Simulation<PhilosopherType extends PhilosopherBase>` - klasa odpowiada za przeprowadzenie symulacji. Wystarczy stworzyć jej obiekt oraz wywołać metodę `simulate()`.
- `PhilosopherBase` - abstrakcyjna klasa dostarczająca funkcjonalności wspólnych dla różnych wariantów filozofów (np. metody odpowiadającej za myślenie).
- `PhilosopherVX`s - 6 klas dziedziczących z `PhilosopherBase` implementujących różne warianty jedzenia.
- `Fork` - Klasa modelująca widelec oraz dostarczająca mechanizm kontroli dostępu do zasobów. W implementacji użyto semafora. Próba podniesienia widelca skutkuje wywołaniem na semaforze `acquire()`, a jego odłożenie - `release()`.
- `Waiter` - Klasa modelująca nadzorcę (kelnera) za pomocą semafora, który można uzyskać (`acquire()`) $N - 1$ razy.

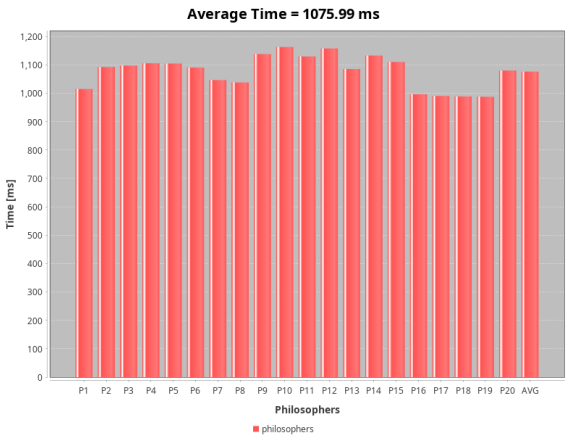
2.1. Wariant 1

Najprostszy wariant, najpierw próbujemy dostać semafor lewego widelca, a zaraz po nim prawego. W implementacji po prostu wywołujemy po sobie metody `lift()` lewego i prawego widelca. Na poniższych wykresach widać, że wraz z ilością filozofów rośnie średni czas oczekiwania na dostęp do zasobów. Wynika to z dłuższych cyklów zależności (filozof 3. oczekuje na widelec filozofa 4., który czeka na 5., itd.). W trakcie 5 minut wykonywania symulacji nie doszło do zakleszczenia.



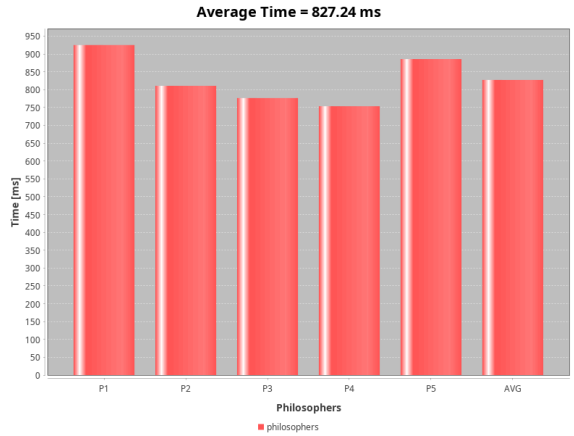


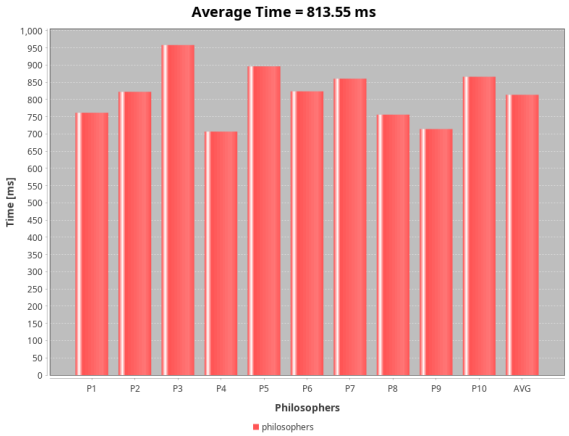


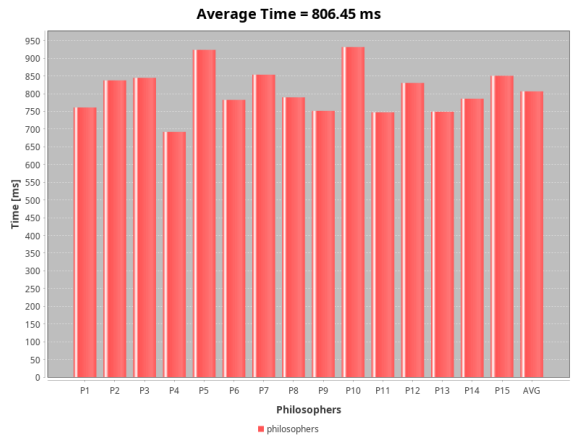


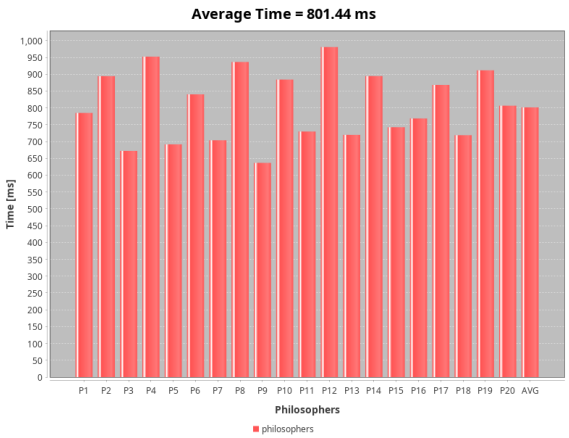
2.2. Wariant 2

W tym wariancie filozof podnosi widelce tylko jeśli oba są wolne. W implementacji metody jedzenia jeśli lewy lub prawy widelec są podniesione to filozof przechodzi do myślenia. W tym przypadku widzimy, że czas oczekiwania na zasoby skrócił się i nie rósł wraz z N. Wynika to zapewne z krótszych cyklów zależności.





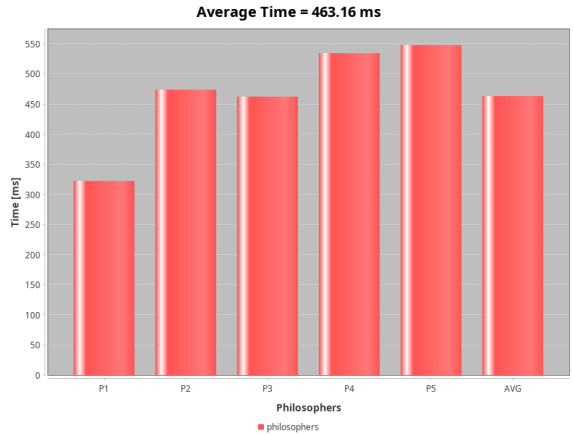


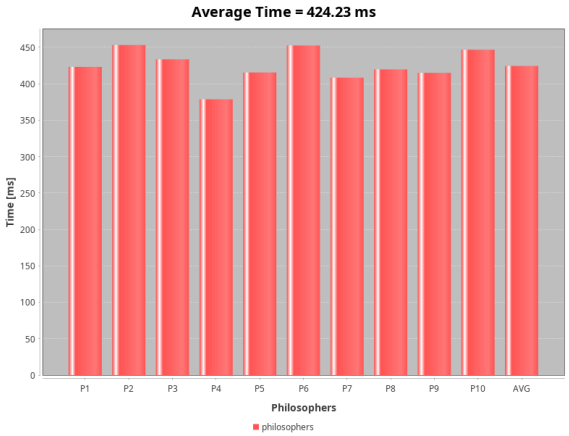


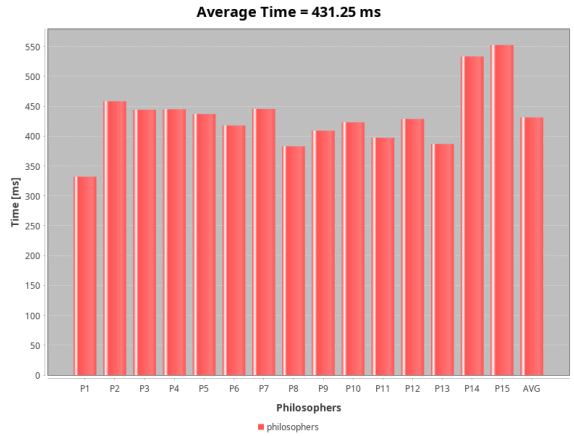
2.3. Wariant 3

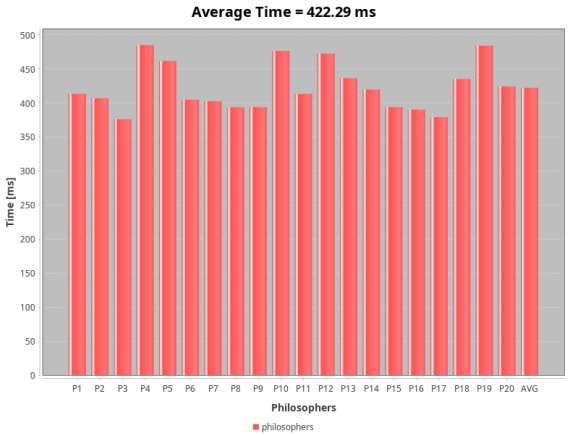
Filozof, który ma numer parzysty podnosi najpierw lewy widelec, a potem prawy. Jeśli ma numer nieparzysty, to na odwrót. W implementacji sprawdzamy czy `id % 2 == 0` dla danego filozofa i w zależności od wyniku najpierw próbujemy dostać semafor lewego lub prawego widelca. Jest to

znacznie lepsza implementacja od poprzednich - średni czas oczekiwania na zasoby zmniejszył się dwukrotnie.



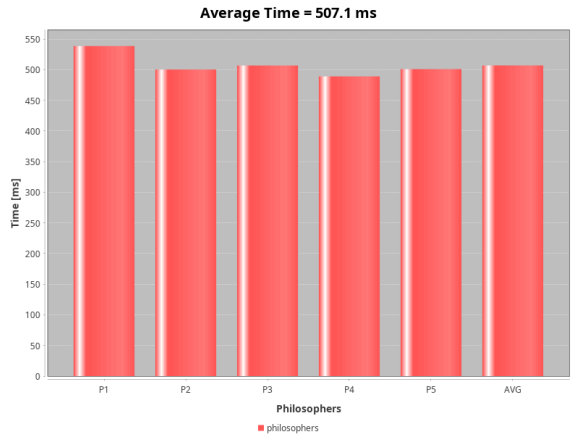


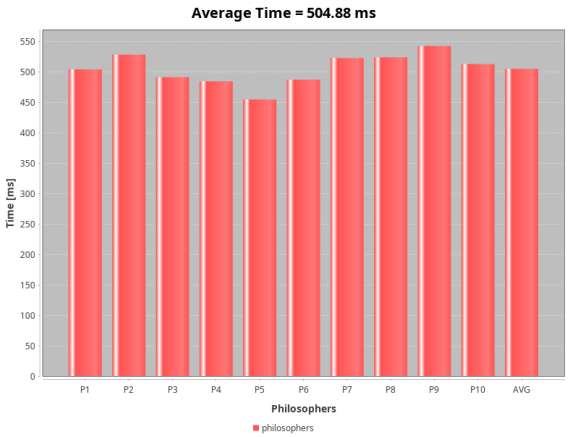


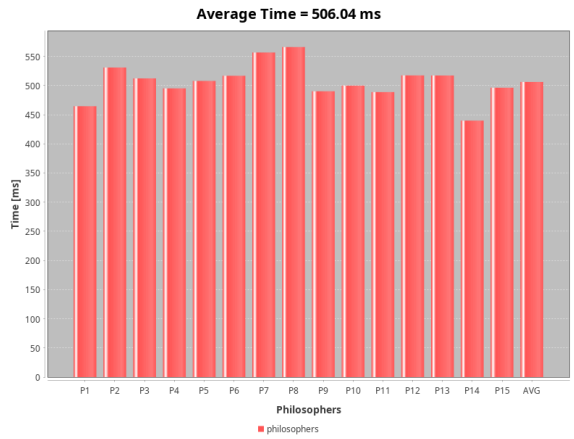


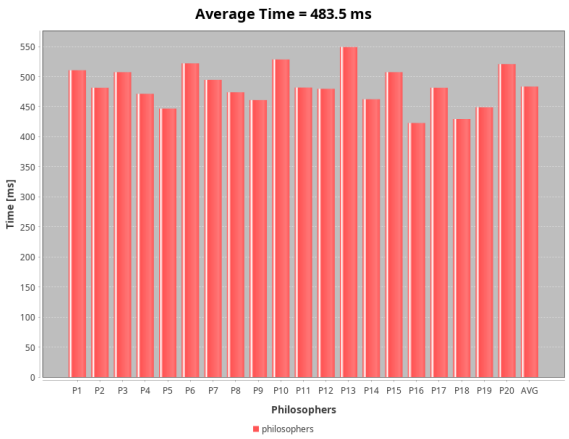
2.4. Wariant 4

W tej wersji to który widelec filozof podnosi najpierw jest losowane za pomocą warunku `Math.random() < 0.5`. Podejście stochastyczne dało lekko gorsze wyniki niż wariant 3. (średni czas oczekiwania na zasoby wydłużył się o kilkadziesiąt milisekund).



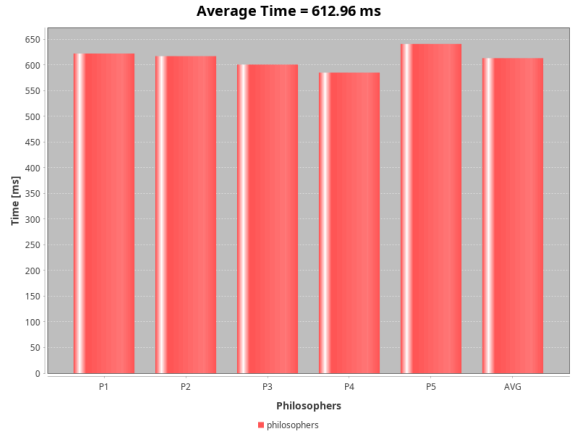


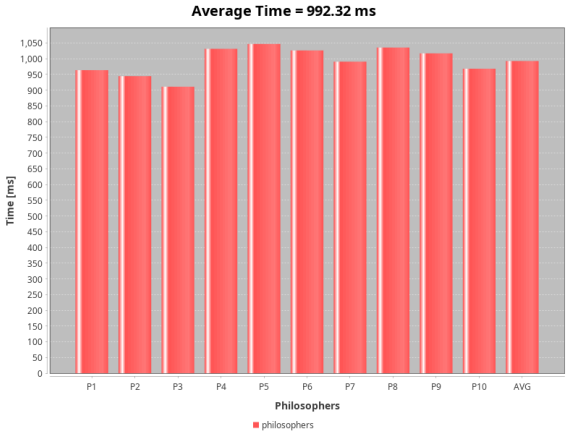


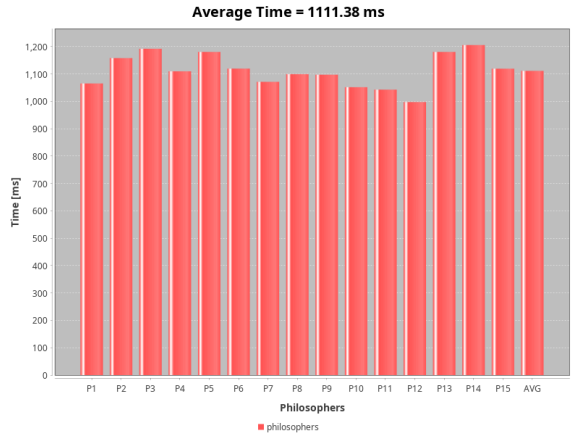


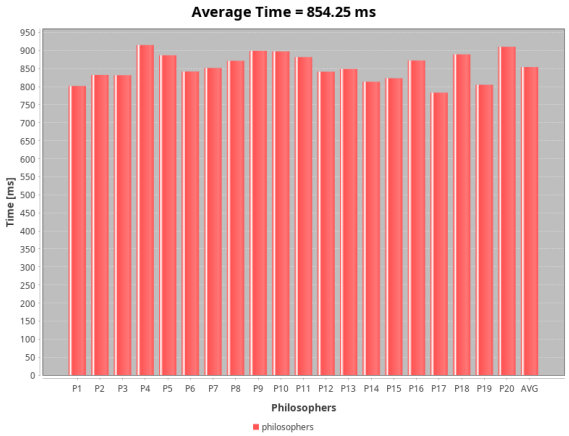
2.5. Wariant 5

Wykorzystujemy tu po raz pierwszy klasę `Waiter`. Przed jedzeniem każdy filozof próbuje dostać semafor kelnera, a po zakończonym posiłku go oddaje. W ten sposób jako że semafor kelnera można dostać $N - 1$ razy uzyskujemy sytuację gdy kelner pilnuje, aby zawsze jeden filozof nie jadł. Ceną za to zabezpieczenie jest jednak dłuższy czas oczekiwania na zasoby jak widać poniżej.



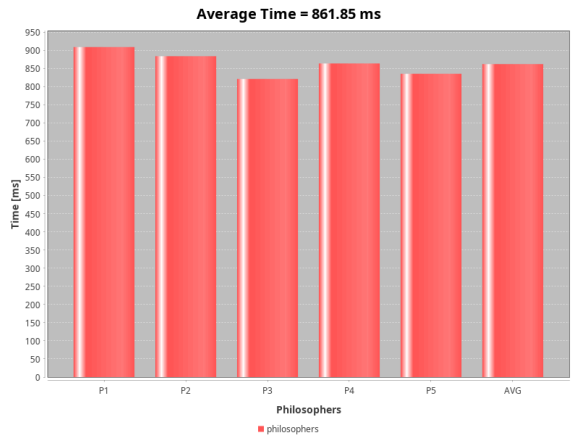


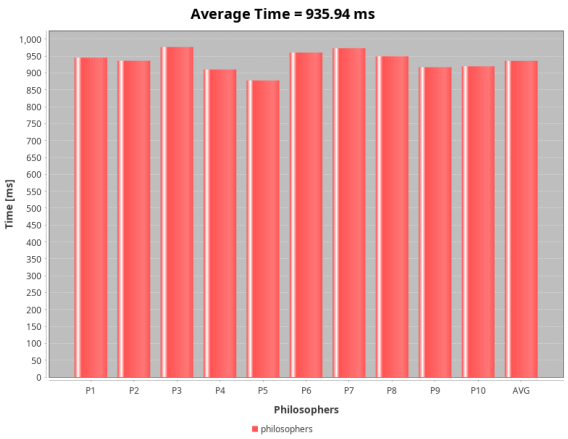


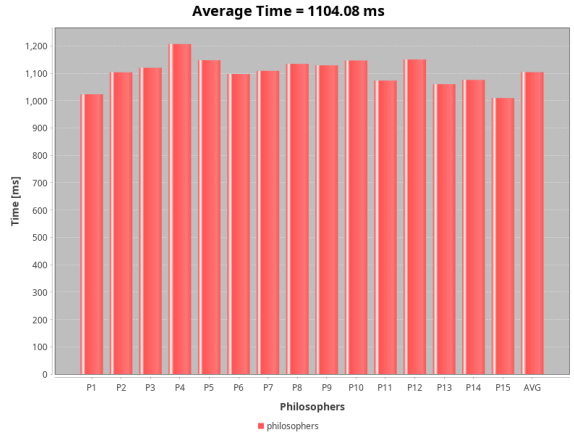


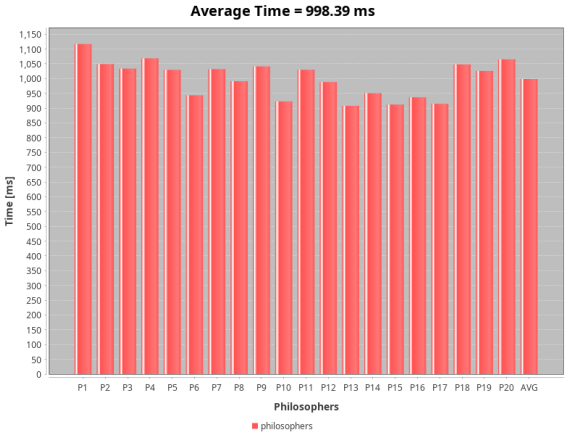
2.6. Wariant 6

Wariant bardzo podobny do poprzedniego, ale jeśli filozof nie może dostać semafora kelnera to wychodzi na korytarz.

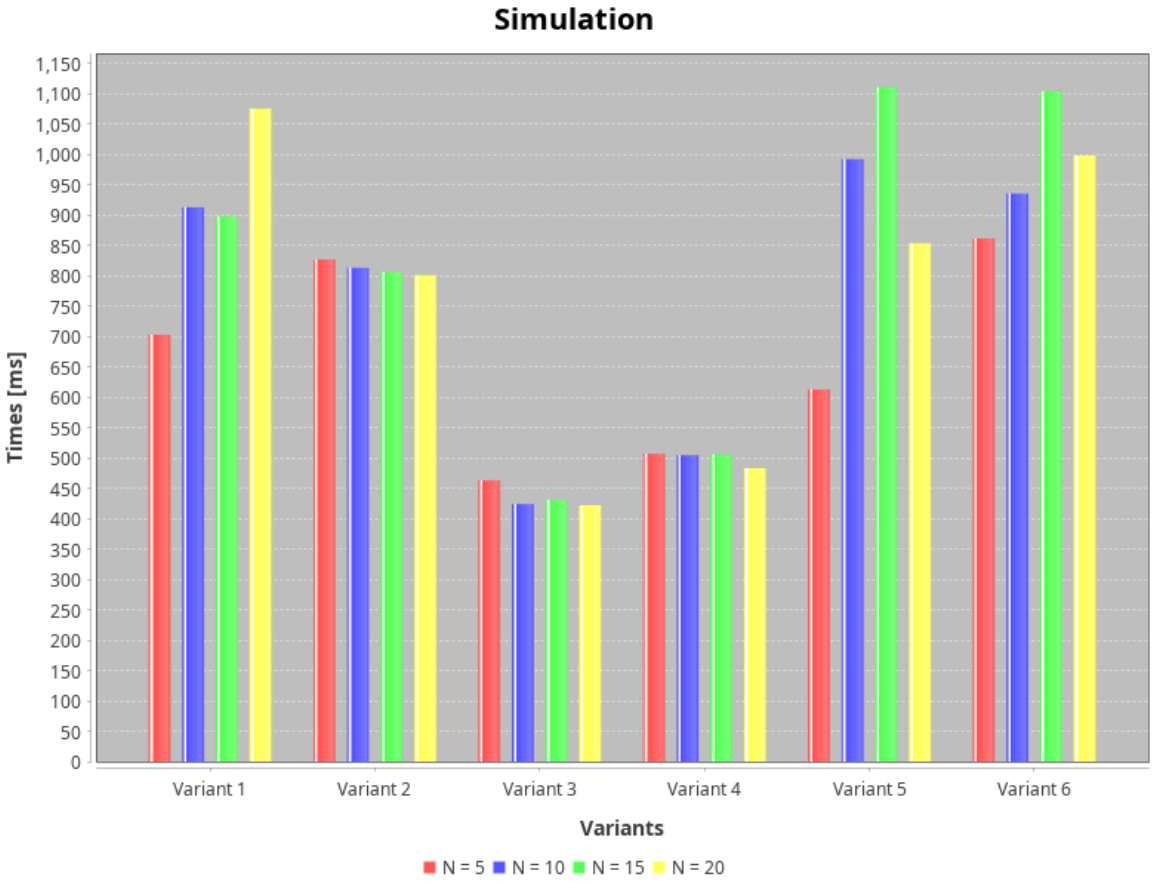

Average Time = 861.85 ms
Time [ms]
950
900
850
800
750
700
650
600
550
500
450
400
350
300
250
200
150
100
50
0
P1
P2
P3
P4
P5
AVG
Philosophers
philosophers


Average Time = 935.94 ms
Time [ms]
1,000
950
900
850
800
750
700
650
600
550
500
450
400
350
300
250
200
150
100
50
0
P1
P2
P3
P4
P5
P6
P7
P8
P9
P10
AVG
Philosophers
philosophers


Average Time = 1104.08 ms
Time [ms]
1,200
1,100
1,000
900
800
700
600
500
400
300
200
100
0
P1
P2
P3
P4
P5
P6
P7
P8
P9
P10
P11
P12
P13
P14
P15
AVG
Philosophers
philosophers


Average Time = 998.39 ms
Time [ms]
1,150
1,100
1,050
1,000
950
900
850
800
750
700
650
600
550
500
450
400
350
300
250
200
150
100
50
0
P1
P2
P3
P4
P5
P6
P7
P8
P9
P10
P11
P12
P13
P14
P15
P16
P17
P18
P19
P20
AVG
Philosophers
philosophers

3. Wnioski



Na podstawie końcowego wykresu możemy zobaczyć, że im lepsze zabezpieczenia przed zagłodzeniem programu tym krótszy średni czas oczekiwania na dostęp do zasobów. Warianty 3 i 4 mają również dużo lepsze czasy od wariantów 5 i 6 z czego wynika, że wprowadzenie arbitrów do logiki programu wydłużyło średni czas oczekiwania na zasoby.