Wrocław, 07.12.2021r.

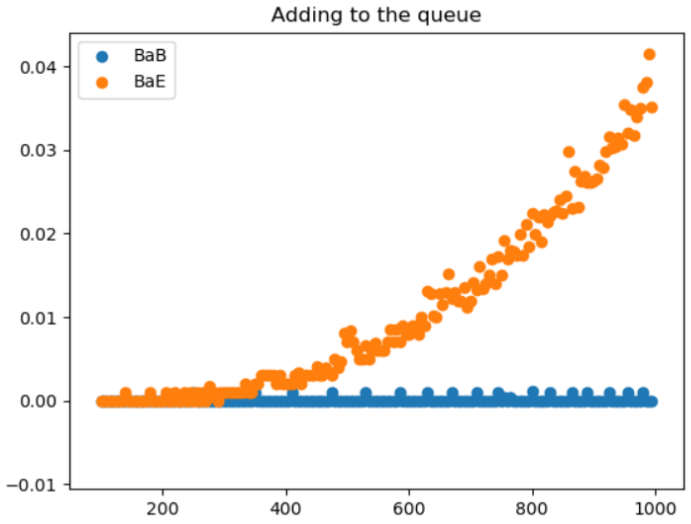
Lista zadań nr 4

Julia Mazur, 262296

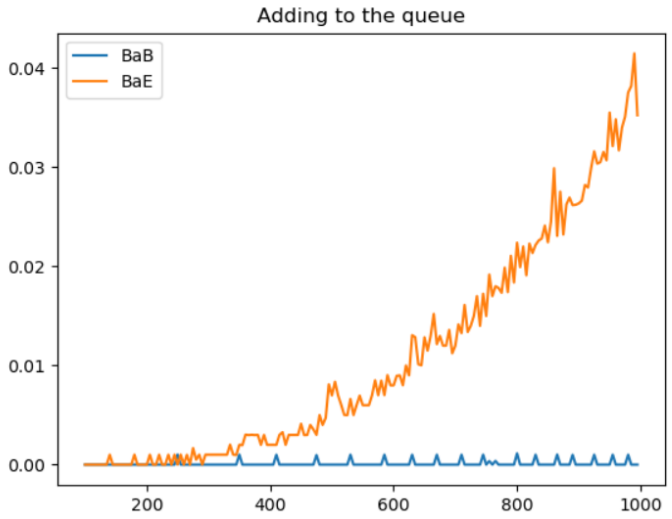
Zadanie 1. i 2.

Aby porównać wydajność obu implementacji postanowiłam porównać czasy wykonywania konkretnych działań na tych kolejkach:

1. Zaczęłam od sprawdzenia ile czasu zajmuje wpisanie różnej ilości elementów do kolejki:

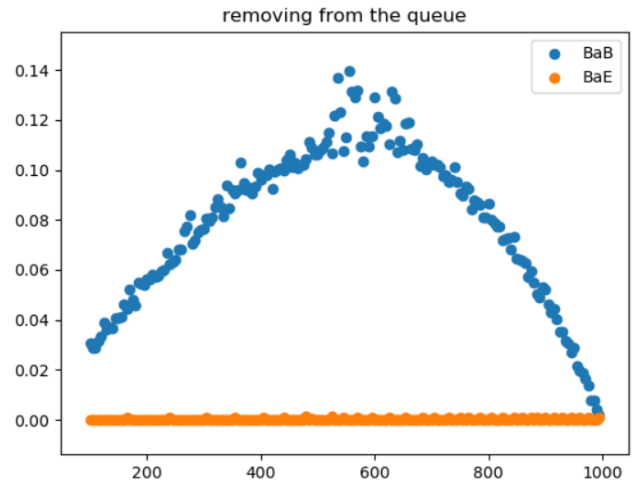


Skoro punkty są tak gęsto postanowiłam je połączyć



Widzimy, że kolejka BaB dla większej ilości jest zdecydowanie szybsza.

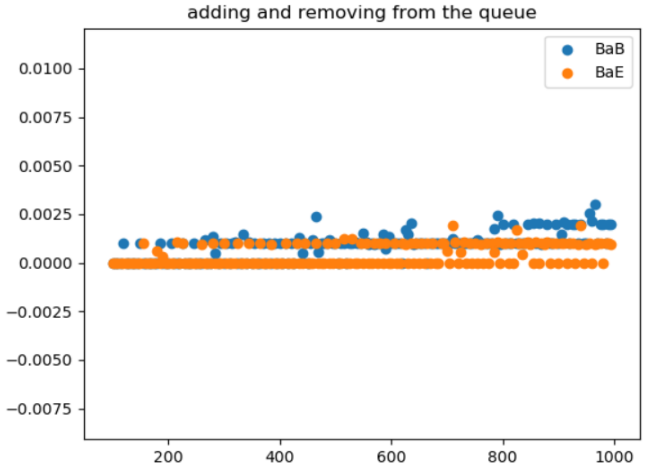
2. Następnie sprawdziłam ile czasu zajmuje usunięcie różnej ilości elementów z kolejki:



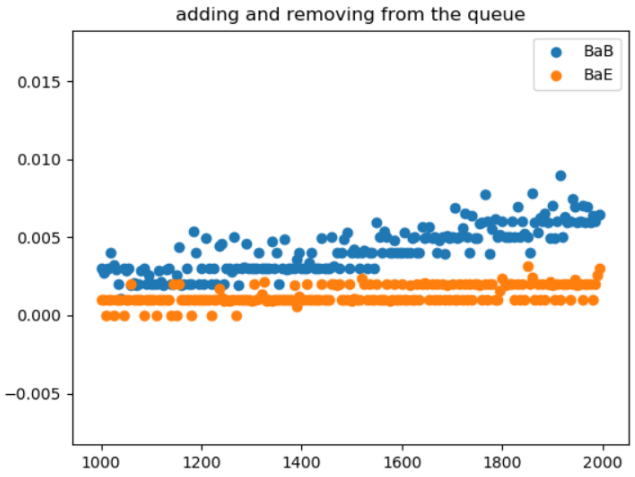
Ponownie jako że punkty są gęsto wygenerowałam jeszcze taki wykres:

Widzimy, że kolejka BaE dla ok 600 elementów jest zdecydowanie szybsza.

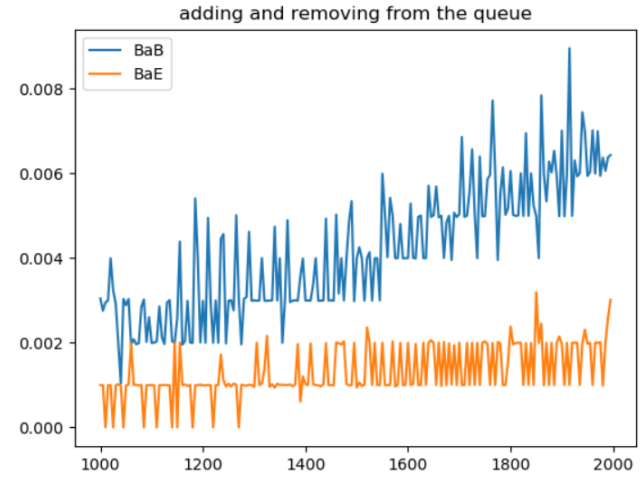
3. W kolejnym kroku porównałam dodawanie i odejmowanie z listy:



Wykres jest trochę bardziej czytelny dla większej ilości elementów:

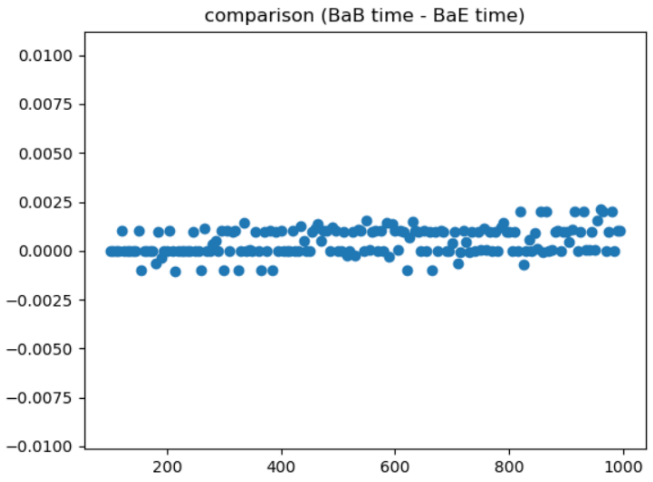


Skoro punkty są gęsto wygenerowałam jeszcze ciągły wykres:

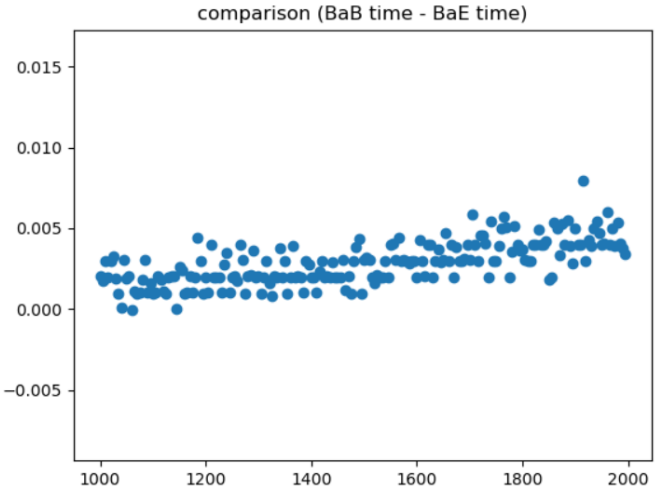


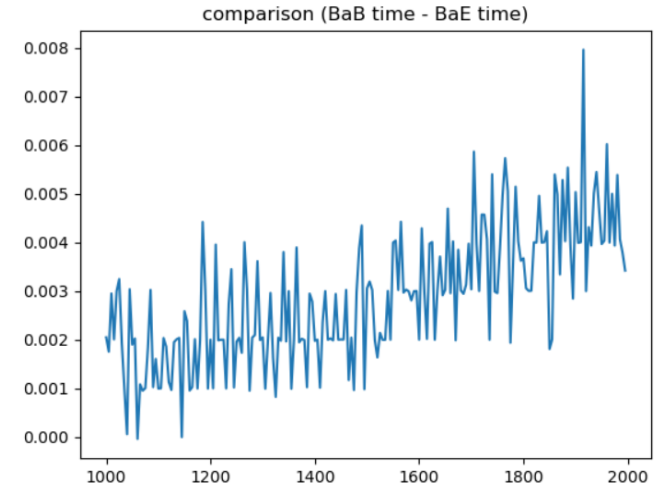
W tym miejscu możemy zauważyć, że BaE jest szybsza od BaB, ale szybkości obu kolejek są porównywalne.

4. Ostatecznie postanowiłam wygenerować wykres przedstawiający różnicę czasów wykonywania działań z punktu (3):



A dla większej ilości elementów:





Z tego widzimy, że kolejka BaE jest szybsza w swoim działaniu

Linki do kodu:

1. https://github.com/Swinkawkrawacie/algorytmy2021-22/blob/0b1a550fd99865306cbabe531afe57085187c6c6/lista4/L4_ZAD1.py
2. https://github.com/Swinkawkrawacie/algorytmy2021-22/blob/0b1a550fd99865306cbabe531afe57085187c6c6/lista4/L4_ZAD2.py

Zadanie 3.

W dobie pandemii często spotykamy się z sytuacją, że kolejki tworzą się nie tylko w sklepie, ale i na zewnątrz. Załóżmy, że mamy sklep, w którym jednocześnie może przebywać 5 osób. Dodatkowo, właściciele sklepu nie przewidzieli wybuchu pandemii i otworzyli bardzo wąski sklep. Po wstawieniu lad i rozłożeniu towaru, ledwo zostało miejsce dla klientów, dlatego po zakupach zakończonych przez jedną osobę wszystkie znajdujące się w sklepie muszą wyjść, aby zrobić miejsce dla tej na początku. Załóżmy, że każdy klient robi zakupy przez 60s, przemieszczenie się o jedno miejsce do tyłu (oraz wyjście ze sklepu) zajmuje 5s a przesunięcie się o jedno miejsce do przodu (oraz wejście do sklepu) zajmuje 3s (bo trzeba zachowywać odpowiednie odległości). Ile czasu zajmie nam zrobienie zakupów razem z koniecznym postojem (zakładamy, że stoimy w kolejce jako ostatni), jeśli przyszliśmy jako

a) trzeci?
b) siódmi?
c) dwunaści?

Żeby rozwiązać ten problem, korzystamy ze stosu i kolejki jednostronnej. Załóżmy, że n>5, czyli utworzyła się kolejka na zewnątrz (zakładam ten model, bo sytuacja, że kolejki na zewnątrz nie będzie zawiera się w nim). Zatem tworzymy stos z ludzi wewnątrz i kolejkę z ludzi na zewnątrz. Za każdym razem gdy ktoś chce odejść od kasy musimy zdjąć wszystko ze stosu i wstawić na nowo bez pierwszej osoby czas opróżniania stosu to "$ilość\ ludzi\ w\ sklepie$" $*\ 5s$, a wejście do sklepu zajmuje $3s$ pierwszej osobie i nie musimy dodawać reszty, ponieważ ta osoba zaczyna zakupy, kiedy inni wchodzą do sklepu.

a) $n = 3$:

```
We are 3rd in the line
Finally, we got out
The time it took us:  0:03:36
```

b) $n = 7$:

```
We are 7th in the line
Finally, we got out
The time it took us:  0:09:23
```

c) $n = 12$:

```
We are 12th in the line
Finally, we got out
The time it took us:  0:16:43
```

Link do kodu:

3. https://github.com/Swinkawkrawacie/algorytmy2021-22/blob/0b1a550fd99865306cbabe531afe57085187c6c6/lista4/L4_ZAD3.py

Zadanie 4.

Program do sprawdzania składni dokumentu html pod względem brakujących znaczników zamykających.
Aby stworzyć ten program zaczęłam od znalezienia pojedynczych znaczników w htmlu (takich, które nie mają odpowiadających im znaczników zamykających). Następnie zaimplementowałam stos, na którym przechowywałam kolejne znaczniki otwierające.
Żeby ułatwić sobie pracę przed rozpoczęciem analizy znaczników usunęłam z tekstu komentarze, jako że to jedne z pojedynczych znaczników, które mogą zawierać dowolne znaki nie mające wpływu na dokument.
Podczas przeszukiwania tekstu, jeśli napotkałam znacznik otwierający, to wstawiałam go na stos, a jeśli był to znacznik zamykający, sprawdzałam ostatni element na stosie i odpowiednio go usuwałam jeśli był poprawny lub stwierdzałam, że dokument html jest niepoprawny. Ostatecznie, jeśli przeszłam przez cały dokument i stos na końcu był pusty to oznaczało, że składnia była poprawna.

Wywołanie programu dla podanych przykładów:

```
first text, correct: True
second text, correct: False
third text, correct: False
```

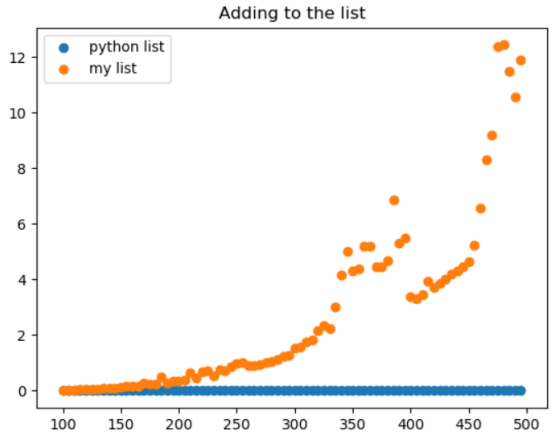
Link do kodu:

4. https://github.com/Swinkawkrawacie/algorytmy2021-22/blob/0b1a550fd99865306cbabe531afe57085187c6c6/lista4/L4_ZAD4.py

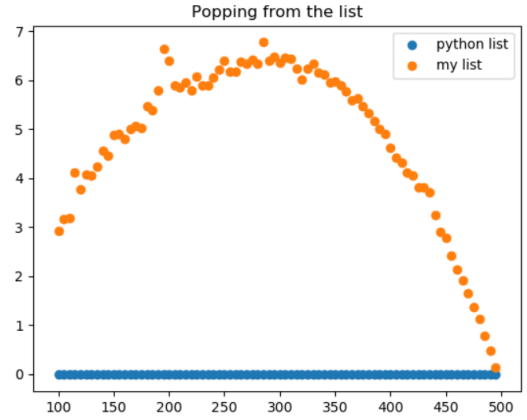
Zadanie 5., 6., 7. i 8.

Aby porównać działanie list, zmierzyłam czas wykonywania konkretnych operacji dla różnej ilości elementów na listach.

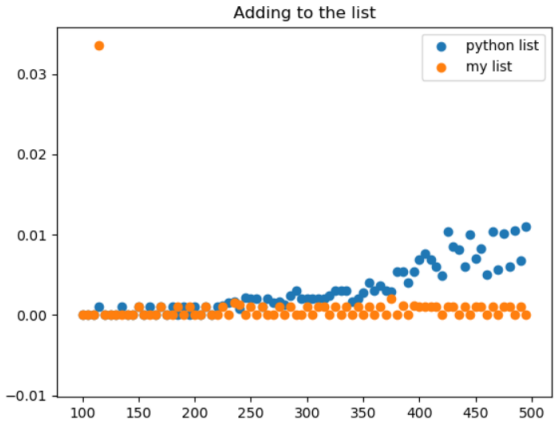
1. Zaczęłam od dopisywania do listy z prawej strony:



   Możemy zauważyć, że stworzona przeze mnie lista jest zdecydowanie wolniejsza.
2. Podobna sytuacja jest przy usuwaniu ostatniego elementu listy.

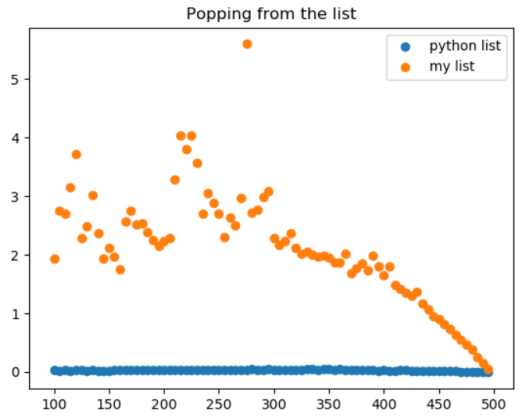


3. Sytuacja zmienia się dla dodawania elementów na początku listy:

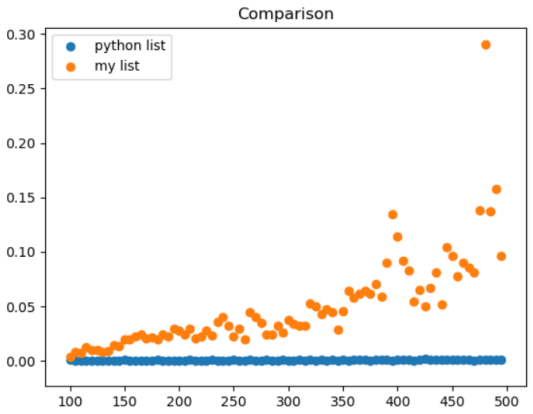


   Tutaj, może nieznacznie, ale moja lista jest szybsza.

4. Jednak przy usuwaniu z początku listy wracamy do sytuacji początkowej:

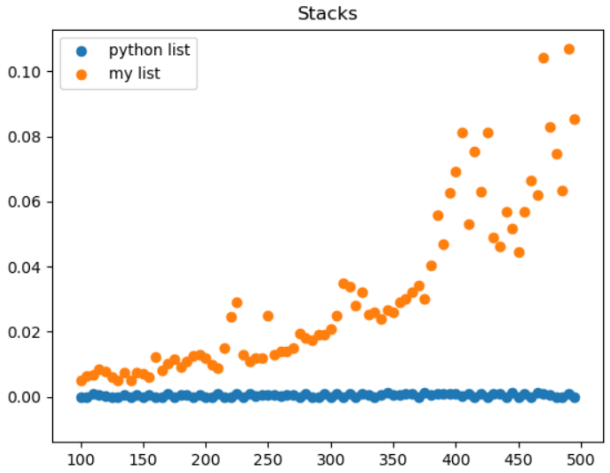


5. Ostatecznie zdecydowałam się sprawdzić dodawanie i usuwanie z początku listy.



Mimo że różnica jest znacznie mniejsza, moja lista wciąż jest wolniejsza od listy pythonowej.

6. Nawet stos, który jest dość optymalny przy użyciu listy jednokierunkowej, jest wolniejszy od tego na liście pythonowej:



Linki do kodu:

5. https://github.com/Swinkawkrawacie/algorytmy2021-22/blob/0b1a550fd99865306cbabe531afe57085187c6c6/lista4/L4_ZAD5.py
6. https://github.com/Swinkawkrawacie/algorytmy2021-22/blob/0b1a550fd99865306cbabe531afe57085187c6c6/lista4/L4_ZAD6.py
7. https://github.com/Swinkawkrawacie/algorytmy2021-22/blob/0b1a550fd99865306cbabe531afe57085187c6c6/lista4/L4_ZAD7.py
8. https://github.com/Swinkawkrawacie/algorytmy2021-22/blob/259fedef92fe8b82fd7cc523abff84f2c6051eb2/lista4/L4_ZAD8.py