**Gdańsk, 11.03.2024**

**JĘZYKI SKRYPTOWE I ICH ZASTOSOWANIA**

*Zadanie 1: Test wydajności*

**Michał Sternik,**

201987

Testowana funkcja: ***statistics.median\_low()***

Funkcja median\_low() zwraca mniejszą z dwóch wartości środkowych (median) w przypadku, gdy liczba danych jest parzysta.

Poniżej znajduje się kod źródłowy implementacji testowanej funkcji zarówno w pythonie, jak I c++.:

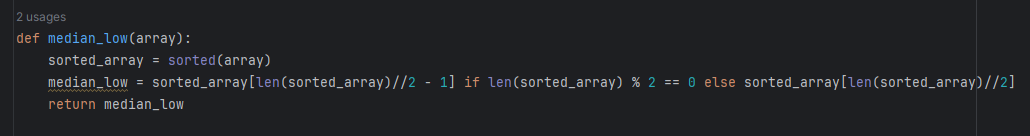


Figure 1 Implementacja metody równoważnej statistics.median\_low() w pythonie

Obraz zawierający tekst, zrzut ekranu, Czcionka, oprogramowanie

Opis wygenerowany automatycznie

Figure 2 Implementacja metody równoważnej statistics.median\_low() w c++

Teraz omówiony zostanie sposób generowania zestawów oraz ich ilość.

W tym celu przygotowany został osobny plik ***generate\_and\_save\_data.py***, który obsługuje generowanie oraz zapisywanie wygenerowanych danych do pliku tekstowego .txt. Jego kod znajduje się poniżej:

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

Figure 3 Kod metod do generowania i zapisywania danych do pliku txt

Dane generowane są w postaci mnogiej liczby list. Każda lista zawiera losową liczbę wartości z zakresu **od 1000 do 10000.** Wartości liczb w liście również są generowane losowo z zakresu **od 1 do 1000.** W kodzie ustalono ilość wierszy na **k = 1000.**

W związku z tym otrzymujemy następujący przykładowy zestaw danych:

Obraz zawierający czarne, ciemność

Opis wygenerowany automatycznie812,72,177,408,563,859, …..n,

81,457,562,697,689,120, …..n,

5,764,511,771,579,183, …….n,

.

.

.

526,21,257,92,384, …… n,

Dane zapisano w pliku test\_data.txt, do którego istnieje łatwy dostęp zarówno z poziomu projektu c++ jak i pythona.

Do pomiaru czasu działania zaimplementowanych funkcji wykorzystano dwie biblioteki, po jednej dla każdego projektu:

* Python

Użyto bibliotekę *time*.

Z tego modułu posłużono się metodą *time()* obliczając moment przed wywołaniem funkcji i moment zaraz po zakończeniu wykonywania funkcji.

*start\_time = time.time()*

*end\_time = time.time()*

Różnica pomiędzy dwoma momentami daje nam czas trwania funkcji.

Przyjęto dokładność funkcji mierzącej czas wynoszący **1 sekundę.**

* C++

Użyto bibliotekę *chrono*.

Posługujemy się funkcją *now().* Tutaj wygląda to podobnie jak w przypadku biblioteki *time* z pythona – obliczamy różnicę w momentach przed wywołaniem funkcji i momentum zaraz po ukończeniu jej wywołania:

start\_time = clock::now();

end\_time = clock::now();

Różnica pomiędzy dwoma momentami daje nam czas trwania funkcji.

Przyjęto dokładność funkcji mierzącej czas wynoszący **1 sekundę.**

Testy dały następujące rezultaty:

Dla przeprowadzenia testów przyjęto **wartość n** (różną dla pythona I c++ ale identyczną w obrębie funkcji wbudowanej i implementowanej samodzielnie w pythonie)

* Python: , błąd względny wyniósł **0.98%**
* C++: , błąd względny wyniósł **0.95%**

Wyniki pomiarów:

|  |  |  |
| --- | --- | --- |
| Funkcja | Wbudowana/implementowana samodzielnie | Czas działania |
| statistics.median\_low (python) | wbudowana |  |
| median\_low (python) | Implementowana samodzielnie |  |
| median\_low (C++) | Implementowana samodzielnie |  |