Wydział Informatyki i Telekomunikacji
Politechnika Wrocławska

Analiza eksperymentalna nowych algorytmów sortowania w miejscu

Damian Baliński
Nr indeksu: 250332

Praca inżynierska napisana
pod kierunkiem
dr inż. Zbigniewa Gołębiewskiego

Politechnika
Wrocławska
Wrocław 2021

Spis treści

Implementacja systemu

1.1 Struktura systemu

Aplikacja składa się z dwóch niezależnych części: silnika testującego oraz silnika graficznego.

Silnik testujący to generycznych biblioteka algorytmów sortujących oraz narzędzie przetwarzające te algorytmy. Aplikacja w oparciu o konfigurację wejściową generuje zestaw testowy oraz utrwala wyniki przeprowadzonych testów na dysku. W zależności od konfiguracji, silnik testujący może sumować, zliczać lub uśredniać liczbę wykonywanych operacji takich jak: liczba porównań, liczba operacji zamiany miejsc, liczba operacji przypisania. Ta część aplikacji została napisana w języku C++ z wykorzystaniem technik programowania obiektowego.

Silnik graficzny to zbiór skryptów przetwarzających wyniki z silnika testującego. Na podstawie pliku konfiguracyjnego oraz danych wejściowych generowane są wizualizacje graficzne w postaci wykresów, dzięki czemu użytkownik końcowy może w łatwy sposób analizować oraz porównywać testowane algorytmy. Ta część systemu została napisana w języku Python przy użyciu biblioteki matplotlib.

1.2 Koncepcje architektury silnika testującego

1.2.1 Wstrzykiwanie zależności

Większość algorytmów sortujących składa się z kilku odrębnych kroków. Niektóre z tych kroków są na tyle złożone, że stanowią osobne algorytmy. Dla przykładu jednym etapów sortowania metodą Quick Sort jest partycjonowanie danych wejściowych na rozłączne zbiory. Aby w łatwy sposób umożliwić modyfikację testowanych algorytmów, bez konieczności ponownej implementacji całego procesu, zastosowano technikę wstrzykiwania zależności. Jeżeli algorytm testujący korzysta z innego algorytmu, to algorytm składowy jest wstrzykiwany w trakcie działania programu. Dzięki temu lekka modyfikacja testowanego algorytmu ogranicza się do podmiany jego algorytmów składowych, bez konieczności ingerowania w strukturę bazową.

1.2.2 Obiektowość

Aby uprościć organizację kodu zastosowano model obiektowy. Każdy z algorytmów wykorzystywanych w systemie został zamodelowany za pomocą odrębnej klasy. Dla każdej rodziny algorytmów tego samego typu istnieje nadrzędna klasa abstrakcyjna określająca interfejs dla tej rodziny. Korzyści wynikające z zastosowanego modelu, takie jak statyczny polimorfizm oraz dziedziczenie gwarantują bardziej wiarygodne działanie programu oraz umożliwiają wykrywanie błędów strukturalnych już na etapie kompilacji projektu.

1.2.3 Bezstanowość

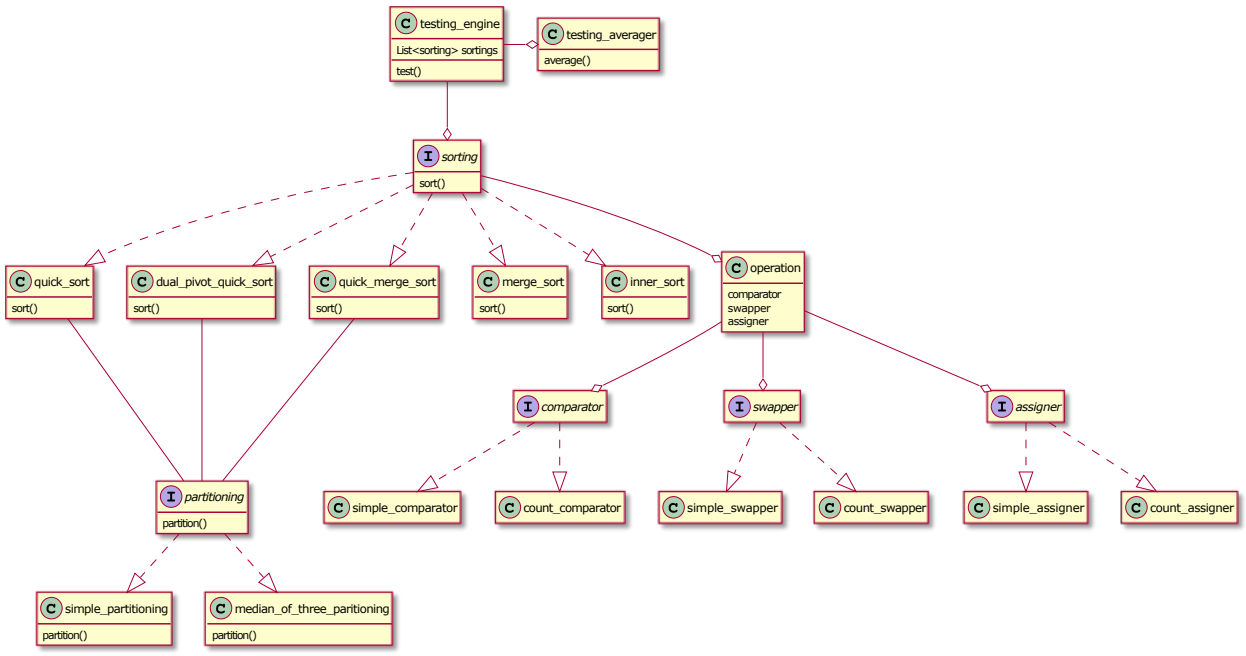
Powszechnym problemem w programowaniu obiektowym jest przechowywanie stanu. Problem ten wynika po części z praktyki hermetyzacji danych wewnątrz obiektowej abstrakcji. Użytkownik zewnętrzny korzystając z interfejsu danego komponentu nie ma dostępu do procesów zachodzących w jego wnętrzu. Może to prowadzić do tzw. efektów ubocznych (ang. side effects), przez co wyniki zwracane przez program stają się

niewiarygodne.

Aby tego uniknąć zastosowano model bezstanowy. Żaden z algorytmów sortujących nie posiada zmiennych składowych, które mogły by zostać zmodyfikowane w trakcie działania programu. Podczas testowania dane są przekazywane poprzez sygnatury metod, wzorując się na technice programowania funkcyjnego. Dzięki temu wszystkie algorytmy sortujące wykorzystane w implementacji są oznaczone jako niemodyfikowalne, nie mogą zmienić stanu aktualnie testowanego algorytmu. Gwarantuje to całkowitą separację poszczególnych testów.

1.3 Model aplikacji

1.3.1 Diagram klas

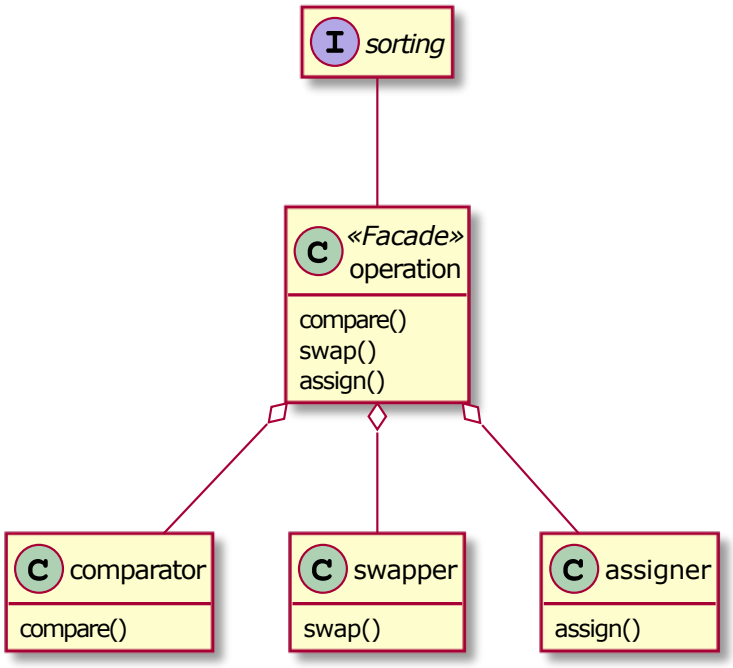


1.3.2 Diagram przepływu

1.4 Wzorce projektowe

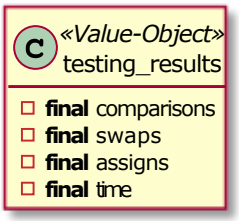
1.4.1 Fasada

W trakcie działania algorytm sortujący wykonuje wiele operacji atomowych, takich jak porównywanie elementów, zamiana elementów miejscami oraz operacje przypisania. Aby uniknąć nadmiaru odpowiedzialności dla klas sortujących zastosowano obiekt pośredniczący **operation** będący równocześnie **fasadą**. Fasada zapewnia jednolity interfejs dla wszystkich operacji atomowych oraz przekierowuje ich działanie do obiektów bezpośrednio odpowiedzialnych za ich wykonanie.



1.4.2 Obiekt-Wartość

Proces testowania algorytmu składa się z wielu iteracji. Każdy z atomowych testów wchodzących w skład iteracji powinien być całkowicie niezależny i odseparowana od innych testów. Aby to zapewnić, dane pochodzące z osobnych pojedynczego testu są przekazywane za pomocą **obiektów-wartości**. Pola w takim obiekcie po inicjalizacji stają się niemodyfikowalne. Użytkownik może jedynie odczytać ich wartość, bez możliwości ich modyfikacji. **Obiekt-wartość** jest gwarancją, że wyniki pochodzące z testu są rzetelne oraz nie zostały zmodyfikowane w trakcie przepływu danych pomiędzy procesami.



1.4.3 Budowniczy

1.4.4 Strategia

1.5 Pseudokod algorytmu Quick Merge Sort