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

Wstęp

Ogólna historia algorytmów sortujących. Motywacja tworzenie algorytmów

1.1 Cel pracy

Motywem przewodnim pracy jest analiza nowoczesnych algorytmów sortujących w miejscu, takich jak koncepcja QuickMerge Sort. W tym celu przygotowano analizę porównawczą podstawowych algorytmów sortujących oraz dokonano przeglądu zmodyfikowanych wersji tych algorytmów oraz przeanalizowano nowoczesne algorytmy hybrydowe, będące połączeniem dwóch lub wielu algorytmów podstawowych.

1.2 Zakres pracy

Aby ułatwić analizę algorytmów przygotowany został silnik testujący oraz silnik graficzny, które w oparciu o plik konfiguracyjny przeprowadzają testy oraz tworzą wizualizację wyników tych testów. Wykorzystując podane narzędzia została przeprowadzona analiza podstawowych oraz hybrydowych algorytmów.

1.3 Przegląd literatury

Ogólny opis pracy Sebastiana Wilda. Pomysł na algorytm QuickMerge Sort.

1.4 Zawartości pracy

Ogólny sposób organizacji dokumentu. Rozdział pierwszy - analiza matematyczna problemu. Rozdział drugi - przegląd podstawowych algorytmów sortujących. Rozdział trzeci - przegląd hybrydowych algorytmów sortujących.

Analiza problemu

2.1 Model matematyczny przypadku średniego

2.2 Model matematyczny przypadku pesymistycznego

2.3 Założenia

2.3.1 Założenia odnośnie testowanych parametrów

Testowana będzie złożoność czasowa, w tym celu analizowane są takie parametry jak: liczba porównań, liczba swapów oraz liczba przypisań, z pominięciem operacji wykonywanych na iteratorach pętli - wyjaśnienie dlaczego.

2.3.2 Założenia odnośnie danych wejściowych

Wiele algorytmów sortujących bazuje na pewnych założeniach odnośnie wejściowego zbioru danych. Np. algorytm ... doskonale radzi sobie ze zbiorem danych prawie posortowanym, tzn. takim w którym ... . W tej pracy zakładamy że dane wejściowe będą losowym ciągiem liczb powtórzeniami.

Przegląd podstawowych algorytmów sortujących

3.1 Quick Sort

Ogólna koncepcja algorytmu, autor, rok powstania, bez pseudokodu, gdzie algorytm jest wykorzystywany (java)

3.1.1 Analiza algorytmu Quick Sort

Wykresy liczby operacji porównania, zamiany miejsc, przypisania. Eksperymentalna analiza wartości oczekiwanej liczby operacji.

3.1.2 Problemy związane z algorytmem Quick Sort

Wiele kosztownych operacji porównania, przeciwieństwo algorytmu Merge Sort. Słaba pesymistyczna złożoność czasowa.

3.2 Merge Sort

Ogólna koncepcja algorytmu, autor, rok powstania, bez pseudokodu.

3.2.1 Analiza algorytmu Merge Sort

Wykresy liczby operacji porównania, zamiany miejsc, przypisania. Eksperymentalna analiza wartości oczekiwanej liczby operacji.

3.2.2 Problemy związane z algorytmem Merge Sort

Konieczność alokacji dodatkowych zasobów.

Przegląd hybrydowych algorytmów sortujących

4.1 Główne sposoby modyfikacji algorytmów

1. Podmiana algorytmów składowych, np. inny algorytm partycjonowania.
2. Łączenie wielu algorytmów w jeden.

4.2 Quick Sort z różnymi algorytmami partycjonowania

Wykresy porównujące wydajność algorytmów rodziny quick sort w zależności od wyboru algorytmu partycjonowania: partycjonowanie zwykłe, median of three, median of medians.

4.3 QuickMerge Sort

Koncepcja algorytmu, mocne strony (Merge Sort bez konieczności alokacji pamięci)

4.3.1 Pseudokod

4.3.2 Analiza algorytmu

Wykresy liczby wykonywanych operacji w porównaniu do algorytmów bazowych. Wykresy gęstości liczby wykonywanych operacji dla stałej liczby n, np n = 10000.

4.3.3 Wnioski

Wyniki analizy porównawczej

4.4 Intro Sort

Ogólny opis algorytmu, gdzie jest wykorzystywany (std::sort w g++), zalety.

4.4.1 Pseudokod

4.4.2 Analiza algorytmu

Wykresy liczby wykonywanych operacji w porównaniu do algorytmów bazowych. Wykresy gęstości liczby wykonywanych operacji dla stałej liczby n, np n = 10000.

4.4.3 Wnioski

Wyniki analizy porównawczej

Implementacja systemu

5.1 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.

5.2 Koncepcje architektury silnika testującego

5.2.1 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.

5.2.2 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.

5.2.3 Bezstanowość

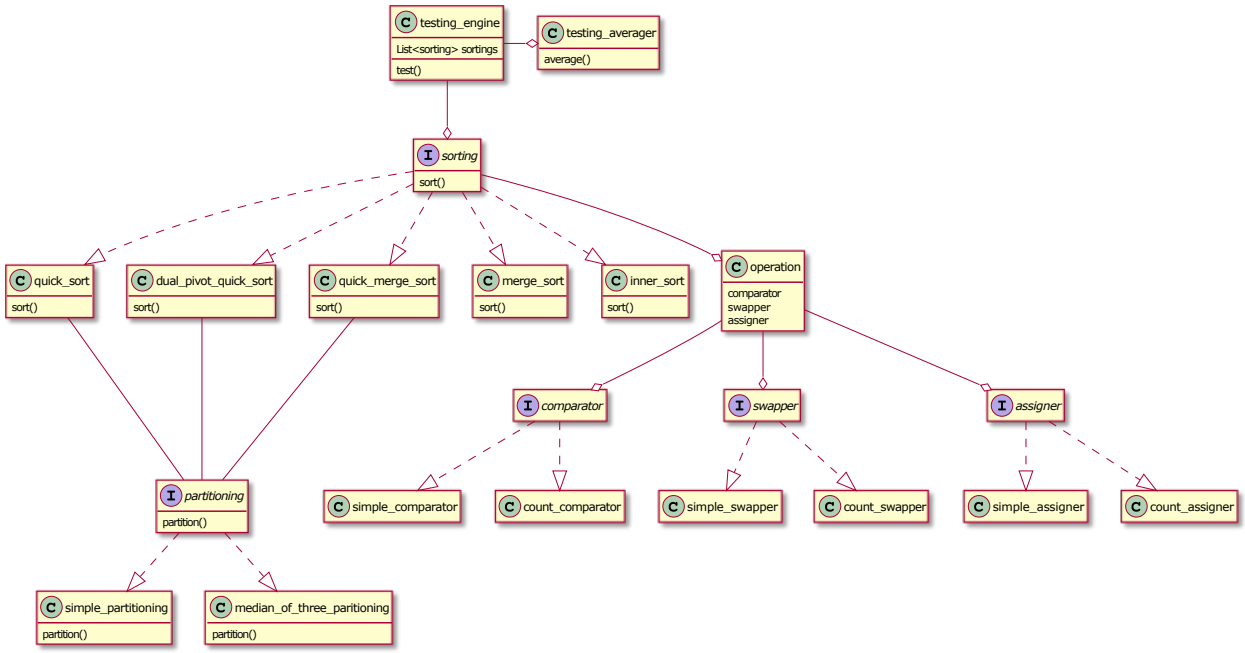
Powszechnym problemem w programowaniu obiektowym jest przechowywanie stanu. Problem ten wynika po części z praktyki hermetyzacji danych wewnątrz obiektowej abstrakcji. Użytkownik zewnętrzny korzystając z interfejsu danego komponentu nie ma dostępu do procesów zachodzących w jego wnętrzu. Może to prowadzić 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 w zaimplementowanym systemie 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.

5.3 Model aplikacji

5.3.1 Diagram klas

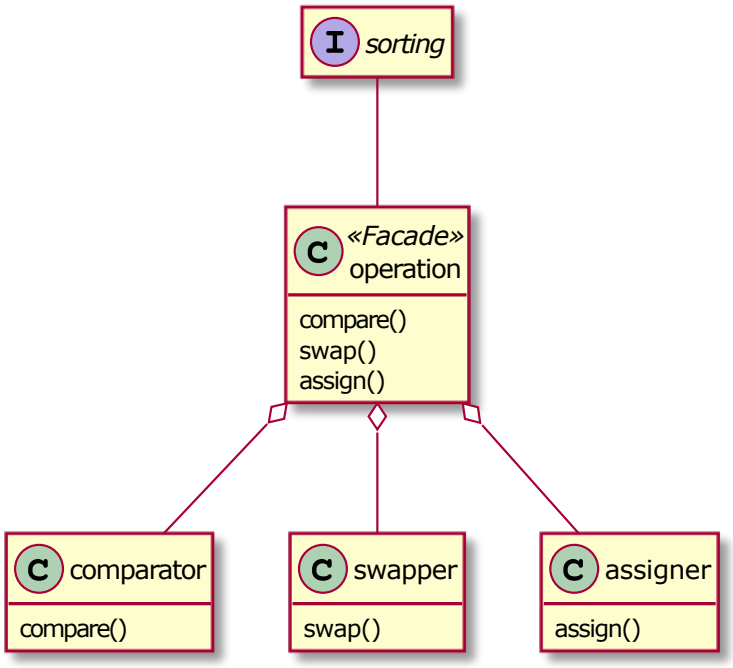


5.3.2 Diagram aktywności

5.4 Wzorce projektowe

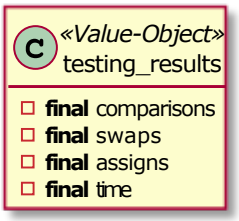
5.4.1 Fasada

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.



5.4.2 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 testów 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.



5.4.3 Strategia

Podsumowanie

Podsumowanie wyników testowania algorytmów. Wnioski z analizy algorytmów hybrydowych.

Bibliografia

Słownik pojęć

A.1 Notacja O())

A.2 Algorytm działający w miejscu

A.3 Algorytm stabilny

Algorytm nie zamienia kolejnością elementów o tej samej wartości.

Środowisko uruchomieniowe aplikacji

Platforma uruchomieniowa aplikacji - Windows. Wykorzystane języki programowania - C++, Python.

B.1 Zmienne środowiskowe

Opis zmiennych środowiskowych TEST-DIRECTORY, CONFIG-DIRECTORY, PLOT-DIRECTORY.

B.2 Biblioteki zewnętrzne

Biblioteki w C++ oraz Pythonie potrzebne do uruchomienia aplikacji wraz z numerami wersji.

B.3 Instalowanie aplikacji

Uruchamianie skryptu zaciągającego potrzebne zależności. Uruchamianie pliku makefile kompilującego i instalującego aplikację. Cykl pracy programu - tworzenie konfiguracji, testowanie silnikiem testującym, wizualizacja wyników przy użycia silnika graficznego.