Sortowanie plików sekwencyjnych

Dominik Lau (188697)

5 października 2023

1 Wprowadzenie

Do zaimplementowania wybrałem algorytm sortowania przez scalanie w schemacie 2+1 (czyli z użyciem trzech taśm). Algorytm ten pozwala na posortowanie danych o rozmiarze większym niż rozmiar pamięci operacyjnej. W wypadku tego algorytmu zużywamy

$$S_{\text{DYSK}} = NR \text{ [B]}$$
$$S_{\text{RAM}} = B \text{ [B]}$$

gdzie B - rozmiar strony (bloku) pamięci w bajtach, N - ilość rekordów, R - rozmiar rekordów w bajtach. Warto zwrócić uwagę, że

$$S_{\text{RAM}} = O(1)$$

Oto statystyki prezentowanego algorytmu

$$T_{\text{Pes}} = \frac{4N\lceil log_2(N)\rceil}{b}$$
$$T_{\text{Opt}} = \frac{4N\lceil log_2(N) - 1\rceil}{b}$$

gdzie $b = \frac{B}{R}$, pozostałe oznaczenia jak powyżej.

Wylosowany przeze mnie typ rekordu to numer rejestracyjny samochodu, które miałem uporządkować leksykograficznie. Implementacji dokonałem w języku C++, jako rozmiar strony przyjąłem $B = 70$ B a rozmiar rekordu $R = 7$ B.

2 Specyfikacja pliku

2.1 Szczegóły implementacyjne

Plik sekwencyjny w kodzie reprezentuje klasa *Tape*, która jest generyczna i może być rozszerzona na inne typy rekordów. W ramach publicznego interfejsu udostępnia ona metody do odczytu aktualnego rekordu, pobrania następnego

rekordu (w trybie odczytu) oraz dodawania rekordu (w trybie zapisu). Szablonowa klasa bazowa rekordu zdefiniowana jest w pliku *RecordIfc.h*. Folder *impl* zawiera implementację wyżej wymienionych interfejsów do rekordu numeru rejestracyjnego. Klasa rekordu udostępnia operacje **serializacji i deserializacji** oraz porównywania.

2.2 Zapis

Bloki z taśmy są zapisywane bezpośrednio do pliku po wcześniejszej serializacji rekordów. **Serializacja rekordów** polega na przedstawieniu znaków wchodzących w skład numeru rejestracji w formie siedmiobajtowej tablicy. Wszystkie rekordy strony są w ten sposób transformowane i scalane w jeden wektor, który następnie jest zapisywany do pliku.

2.3 Odczyt

Odczyt działa podobnie do zapisu, z pliku pobierany jest blok, następuje **deserializacja** kolejnych siódemek bajtów do rekordów, które dodawane są do wektora w pamięci operacyjnej.

3 Prezentacja wyników programu

TODO: opis cli

4 Eksperyment

4.1 Szczegóły implementacyjne

Kod przeprowadzonego eksperymentu umieściłem w pliku *perf1.cpp* jako część biblioteki *sbd_test*. Test uruchamiany jest za pomocą frameworka do testowania gtest. W celu zliczania ilości operacji wejścia-wyjścia oraz liczby cykli algorytmu w bibliotece *libsbd* zdefiniowałem trzy zegary: *writeClock*, *readClock* oraz *phaseClock*. W pomiarach wykorzystuję również klasę *Measurement*, która zbiera pomiary na wzór paradygmatu RAII - w konstruktorze zapisywany jest aktualny stan zegara a w destruktorze nowy stan zegara jest odejmowany od starego, w ten sposób otrzymuję liczbę wywołań funkcji *tick* danego zegara.

4.2 Wyniki

TODO: wyniki w tabelce

5 Wnioski

TODO: przeanalizować i zapisać wnioski