Sortowanie plików sekwencyjnych

Dominik Lau (188697)

7 października 2023

1 Wprowadzenie

Do zaimplementowania wybrałem algorytm sortowania przez scalanie w schemacie 2+1 (czyli z użyciem trzech taśm). Wylosowane przeze mnie typ rekordu to **numery rejestracyjne samochodów**, które miałem uporządkować **leksykograficznie**. Implementacji dokonałem w języku C++, rozmiar rekordu przyjąłem $R = 7$ [B], najpierw jako rozmiar strony ustaliłem $B = 70$ [B] a następnie $B = 700$ [B].

2 Sortowanie przez scalanie

Sortowanie to składa się z kolejnych iteracji (faz), z czego każda faza ma dwie części. Oto pseudokod

```
while(!sorted){
   [tape1, tape2] = distribute(tape)
   [tape, sorted] = merge(tape1, tape2)
}
```

Dystrybucja polega na rozkładaniu kolejnych podciągów niemalejących (serii) wartości w taśmie wejściowej na na zmianę na dwie taśmy pomocnicze. **Scalanie** to łączenie kolejnych serii i zapisywanie ich na taśmie. W ten sposób z każdą iteracją w sortowanej taśmie powstaje coraz więcej serii aż do momentu powstania pojedynczej. Wówczas jesteśmy w stanie uznać plik za posortowany.

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

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

gdzie $b = \frac{B}{R}$, pozostałe oznaczenia jak powyżej. Czas wykonania algorytmu liczony jest w ilości operacji dyskowych.

3 Specyfikacja pliku

3.1 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.

3.2 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.

3.3 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.

4 Prezentacja wyników programu

Efekt wywołania operacji help udostępnianej przez program

```
  manual - generate tape from user input
  random <n> - generate tape with n random records
  file - generate tape from file
  debug - enable/disable debug mode
  exit - exit
```

manual, file i random to trzy sposoby na wprowadzanie danych do programu (tworzenie taśmy). Debug umożliwia przełączanie trybu wypisywania wszystkiego i wypisywania tylko statystyk pomiaru (odczyty/zapisy/fazy). Oto przykładowe wywołanie random 5.

```
[TAPE] maintape
7DEP1CD
0OF2Q8J
M9E0CHA
L6VIHD8
QIWSW6Y

[TAPE] temptape1

[TAPE] temptape2

[SORTING] after #1 distribution
[TAPE] temptape1
7DEP1CD
L6VIHD8
QIWSW6Y

[TAPE] temptape2
0OF2Q8J
M9E0CHA

[SORTING] after #1 merge
[TAPE] maintape
0OF2Q8J
7DEP1CD
L6VIHD8
M9E0CHA
QIWSW6Y

[Measurement] r: 4 w: 3 io(r+w): 7 phases: 1
```

temptape1 i *temptape2* to taśmy pomocnicze a *maintape* to taśma główna czyli sortowany plik sekwencyjny.

5 Eksperyment

5.1 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.

5.2 Wyniki

(wyniki teoretyczne to przypadki średnie)
dla $b = 10$

N	odczyty	zapisy	r+w	liczba faz	teoretyczne r+w	teoretyczna liczba faz
10	6	5	11	2	12	3
100	125	124	249	6	240	6
1000	1809	1808	3617	9	3600	9
5000	12011	12010	24021	12	24000	12
10000	26011	26010	52021	13	52000	13
25000	70013	70012	140025	14	140000	14
50000	150013	150012	300025	15	300000	15
100000	320015	320014	640029	16	640000	16
200000	680014	680013	1360027	17	1360000	17

dla $b = 100$

N	odczyty	zapisy	r+w	liczba faz	teoretyczne r+w	teoretyczna liczba faz
10	9	8	17	3	1	3
100	18	17	35	6	24	6
1000	189	188	377	9	360	9
5000	1212	1211	2423	12	2400	12
10000	2613	2612	5225	13	5200	13
25000	7014	7013	14027	14	14000	14
50000	15015	15014	300025	15	300000	15
100000	32016	32015	64031	16	64000	16
200000	68017	68016	136033	17	136000	17

5.3 Omówienie

Powyższe wyniki można uznać za zgodne z rozważaniami teoretycznymi mimo drobnych odstępstw, które związane są ze szczegółami implementacyjnymi (np. pewnymi nadmiarowymi odczytami/zapisami) a także oscylacjami od wartości średniej. Warto zwrócić uwagę na fakt, że większa ilość rekordów skutkuje mniejszym odchyleniem od oczekiwanych ilości zapisu/odczytu - jest to uzasadnione **centralnym twierdzeniem granicznym** (rozkład wartości średniej dąży do szpilki prawdopodobieństwa). Pragnę zauważyć, że **liczba faz w zależności od N nie jest różna dla różnych współczynników blokowania** - zależy tylko od ilości serii, których w średnim przypadku jest $\frac{N}{2}$. Różni się natomiast liczba operacji wejścia-wyjścia, **zgodnie z intuicją czym większa strona tym rzadziej do niej sięgamy** (spadek liniowy względem $\frac{1}{b}$).

6 Wnioski