Sortowanie pliku z użyciem wielkich buforów

sprawozdanie

Michał Tarnacki s188627

19 listopada 2023

1 Wstęp

Celem projektu było zaimplementowanie algorytmu do sortowania plików przy użyciu jednej z wybranych metod. Wybrana została przeze mnie metoda wielkich buforów, korzystająca z sortowania metodą quick sort oraz scalania (n-1)+1 wykorzystującego kopiec w celu zmniejszenia liczby porównań. Dokładniejszy opis algorytmu znajduje się w sekcji Algorytm.

2 Algorytm

2.1 Opis

2.1.1 Ogólne

Do zaimplementowania algorytmu został wykorzystany język C ze względu na szybkość wykonywania obliczeń. Program składa z kilku części realizujących poszczególne etapy algorytmu m.in. z pliku FirstStage.C realizującego etap pierwszy algorytmu oraz SecondStage.C realizującego drugi etap algorytmu.

2.1.2 Rekord

Taśmy reprezentowane są przez pliki binarne zawierające rekordy stałej długości w formacie: ID (objętość: double), Radius (promień: unsigned char) oraz Height (wysokość: unsigned char). Rekord zawiera więc 10 B. Dostępne wartości promienia oraz wysokości mieszczą się w przedziale 1...254. Rekord Radius=255 i Height=255 został zarezerwowany jako wartość specjalna.

2.1.3 Bufory

Do obsługi buforów została stworzona struktura buffer:

```
typedef struct buffer_{
        void*startLoc;
        int currentPos;
        int noOfRec;
        int blockReadNo;
        int blockWriteNo;
} buffer;
```

z użyciem której wykonywane są operacje dyskowe (o czym niżej) oraz na której wykonywane jest sortowanie. Jak widać to bufor zapisuje liczbę operacji odczytów i zapisów blokowych a więc podpinając do bufora różne pliki, po szeregu operacji na nich, otrzymujemy sumę odczytów i zapisów blokowych wszystkich wykorzystywanych przez bufor plików. Do celów testów liczba buforów została ustalona na N=101 natomiast pojemność pojedynczego bufora na b=10 rekordów.

2.1.4 Obsługa plików

Zapis oraz odczyt blokowy realizowany jest w następujący sposób:

- Stworzona została struktura sequential_file

```
typedef struct sequential_file_{
        void*filePtr;
        char*name;
        buffer *buff;
} sequential_file;
```

- Gdy chcemy skorzystać z pliku otwieramy go metodą
  `InitFile(char *file, void*bufferLoc, int mode),`
  której argumentami są kolejno: łańcuch znaków z nazwą plików, wskaźnik na bufor który podłączamy do pliku oraz tryb w jakim ma zostać otwarty
- Można zainicjować plik bez bufora jednak aby skorzystać z operacji odczytu lub zapisu należy go wcześniej podłączyć przypisując jego adres do odpowiedniego wskaźnika w zmiennej strukturalnej sequential_file
- Gdy chcemy odczytać blok danych korzystamy z metody
  `size_t BlockRead(sequential_file*file),`
  zostanie wówczas odczytane b rekordów (lub mniej jeśli czytamy ostatnie rekordy z pliku) do podłączonego do pliku bufora
- Gdy chcemy odczytać pojedynczy rekord używany metody
  `void* ReadOneRecord(sequential_file *file),`
  zwracającej adres do pojednyczego rekordu znajdującego się w buforze
- Podczas pojedynczego odczytu automatycznie odczytywany jest nowy blok przy dotarciu do końca poprzedniego (pojedynczy odczyt z bufora zachowuje blokowy odczyt pliku)
- Do zapisu rekordu służy funkcja
  `void BlockInsertRecord(sequential_file *file, const void *rec)`
  automatycznie zapisująca blok do powiązanego pliku jeśli bufor się wypełni
- Jeśli chcemy zapisać bufor gdy nie jest pełny możemy skorzystać z funkcji
  `size_t ForceBlockWrite(sequential_file *file)`
  wymuszającej zapis niepełnego bloku

2.1.5 Sortowanie

W celu posortowania rekordów, N buforów traktowanych jest jako jeden. W tym celu stosowana jest warstwa translacji adresów która zapewnia odczyt i zapis z/do właściwych buforów. Sam quicksort widzi adresy liniowo.

2.1.6 Scalanie

Aby zminimalizować liczbę porównań przy scalaniu wykorzystywany jest minimalny kopiec. Algorytm:

1. Do tablicy ładowane są po kolei rekordy z każdego z wczytanych plików oraz zapisywana jest informacja z którego bufora pochodzą.
2. Budowany jest kopiec, w ten sposób najmniejszy element znajduje się w jego korzeniu.
3. Wyjmujemy najmniejszy element i zapisujemy go w pliku wynikowym. Na jego miejsce natomiast wczytujemy kolejny element z bufora do którego przypisany był zapisany element. Od tego wczytania są jednak wyjątki. Bufor z którego mamy odczytać:
   - jest pusty
   - zawiera mniejszy element niż w korzeniu kopca

Wówczas wczytujemy rekord maksymalny (Radius=255, Height=255), niewystępujący w bazie danych (Po zrobieniu Heapify mamy pewność, że nie wystąpi on w korzeniu o ile w kopcu znajdują się jeszcze rekordy niemaksymalne)

4. Wykonujemy funkcję heapify w wyniku której najmniejszy rekord ponownie znajduje się w korzeniu

   Jeśli w korzeniu znajduje się rekord maksymalny wczytujemy dalszą zawartość wszystkich plików (wczytując rekord maksymalny w przypadku pustego pliku) i wracamy do pkt 2.

   Jeśli uda nam się wczytać żadnego pliku kończymy działanie algorytmu oraz zapisujemy zmiany

3 Implementacja

3.1 Faza 1

Podczas fazy 1. wczytywane jest N*b rekordów do buforów. Następnie bufory są sortowane i ostatecznie dystrybuowane na taśmy. Czynność powtarzana jest do rozdystrybuowania wszystkich rekordów z pliku wejściowego. Maksymalna liczba taśm wynosi N-1 ze względu na konieczność wczytania wszystkich w fazie 2.

3.1.1 Analiza

Ze względu na fakt że każdy blok jest raz wczytywany do bufora oraz raz zapisywany do pliku wyjściowego oczekiwana liczba operacji dyskowych wynosi:

$$2\frac{N}{b},$$

liczba przebiegów zapisanych na dysku natomiast wynosi:

$$\lceil\frac{N}{nb}\rceil,$$

ze względu na sortowanie sortowanie podczas jednego przebiegu nb rekordów.

3.2 Faza 2

Podczas fazy 2. wczytywane są wszystkie istniejące taśmy (powstałe w fazie 1 lub w wyniku dystrybucji w fazie 2.) a następnie scalane na taśmę wynikową. W kolejnym kroku stare taśmy są usuwane oraz następuje dystrybucja taśmy wynikowej na nowe taśmy. Algorytm powtarzany jest do momentu gdy wczytana zostanie tylko jedna taśma. Wówczas mamy pewność, że jest ona posortowana (podczas dystrybucji kolejne serie rozkładane są do kolejnych taśm a więc jeśli istnieje 1 plik to jest w nim tylko jedna seria)

3.2.1 Analiza

Ze względu na fakt, iż po każdym cyklu fazy 2. długość serii rośnie n-1 (dla dużych n w przybliżeniu n) krotnie, oczekiwana liczba cykli wynosi

$$log_n(\frac{N}{nb}).$$

Ponieważ podczas tej fazy najpierw wynik scalania zapisywany jest na jednej taśmie a następnie z niej dystrybuowany na nowe taśmy, oczekiwana liczba operacji dyskowych wynosi:

$$2 * 2\frac{N}{b}.$$

Na każdy blok wykonywane są 2 operacje odczytu oraz 2 zapisu.

4 Korzystanie z programu

Do obsługi programu wykorzystać można następujące polecenia:

- create [nazwa_pliku] - tworzy oraz otwiera nową bazę danych
- open [new|custom nazwa_pliku] otwiera istniejącą bazę danych (dla new - nowo stworzoną, dla custom - o podanej nazwie pliku)
- insert [n] [random|manual] dodaje do bazy danych n rekordów (random - losowo, manual - ręcznie wpisując)
- print - wypisuje aktualną bazę danych
- sort [normal|debug] - sortuje aktualną bazę danych
- replace - nadpisuje aktualną bazę danych przez nowo utworzoną
- exit - kończy działanie programu

Po każdym sortowaniu wypisywany jest na standardowe wyjście raport zawierający liczbę odczytów, zapisów, sortowań oraz cykli fazy 2.

5 Eksperyment

W celu przeprowadzenia eksperymentu zostały zachowane ustawione wcześniej parametry programu:

- b = 10
- n = 101

a sam eksperyment został przeprowadzony dla liczby rekordów:

$$N \in \{500, 1500, 15000, 150000, 1500000, 15000000\}.$$

W tym celu przygotowany został plik tekstowy (test) z poleceniami:

```
create db insert 500 random sort normal
create db insert 1500 random sort normal
create db insert 15000 random sort normal
create db insert 150000 random sort normal
create db insert 1500000 random sort normal
create db insert 15000000 random sort normal
exit
```

a następnie przekazany do programu którego standardowe wyście zostało zapisane w pliku

```
cat ./test | ./proj > output
```

Otrzymano następujący plik

```
Insert "help"for menu options
Sort successful
Read no: 50, write no: 50, sort no: 1, stage2 cycles: 0
Sort successful
Read no: 450, write no: 450, sort no: 2, stage2 cycles: 1
Sort successful
Read no: 4500, write no: 4500, sort no: 15, stage2 cycles: 1
Sort successful
Read no: 75004, write no: 75004, sort no: 149, stage2 cycles: 2
Sort successful
Read no: 750026, write no: 750026, sort no: 1486, stage2 cycles: 2
Sort successful
Read no: 10500062, write no: 10500062, sort no: 14852, stage2 cycles: 3
```

5.1 Porównanie wartości faktycznych z teoretycznymi

Do porównania otrzymanych wyników z oczekiwanymi, dla każdej liczby N obliczono teoretyczną liczbę odczytów, zapisów, sortowań, cykli w drugiej fazie oraz całkowitą liczbę operacji dyskowych, na podstawie wzorów zawartych w sekcji "Implementacja". Otrzymano następujące rezultaty:

n	b
101	10

N	500			
	Wart.teoretyczna	Wart.faktyczna	Różnica	%
Odczyty	50	50	0	0,00%
Zapisy	50	50	0	0,00%
Sortowania	1	1	0	0,00%
Cykle fazy 2.	0	0	0	0,00%
Operacje dyskowe	100	100	0	0,00%

N	1500			
	Wart.teoretyczna	Wart.faktyczna	Różnica	%
Odczyty	450	450	0	0,00%
Zapisy	450	450	0	0,00%
Sortowania	2	2	0	0,00%
Cykle fazy 2.	1	1	0	0,00%
Operacje dyskowe	900	900	0	0,00%

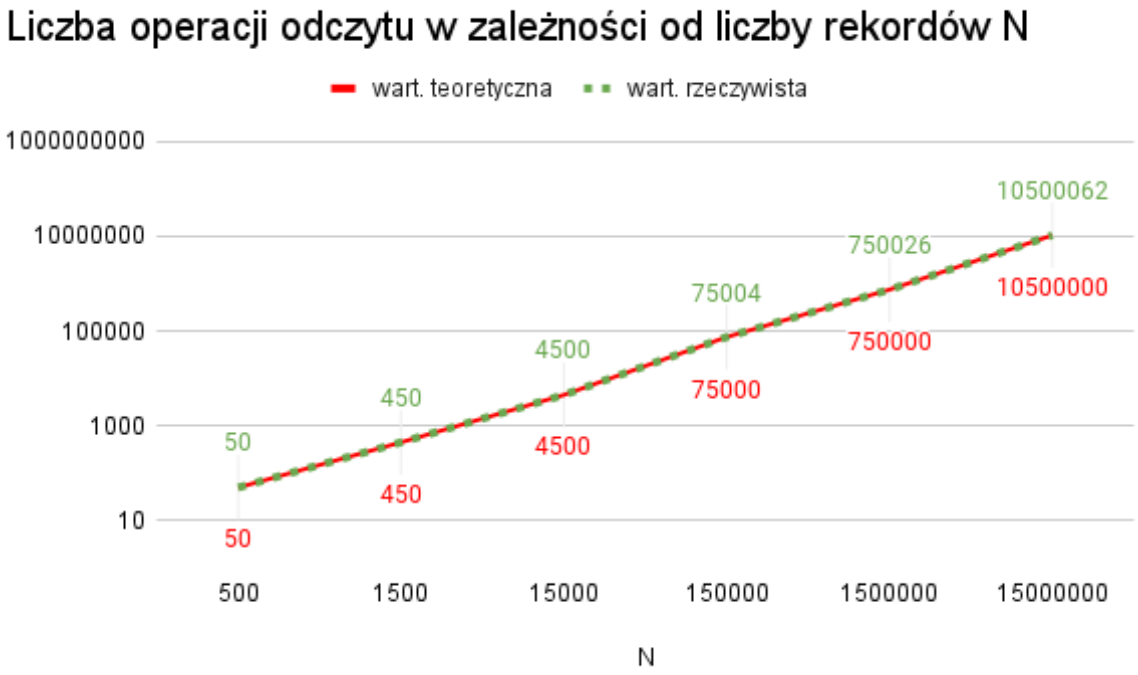
N	15000			
	Wart.teoretyczna	Wart.faktyczna	Różnica	%
Odczyty	4500	4500	0	0,00%
Zapisy	4500	4500	0	0,00%
Sortowania	15	15	0	0,00%
Cykle fazy 2.	1	1	0	0,00%
Operacje dyskowe	9000	9000	0	0,00%

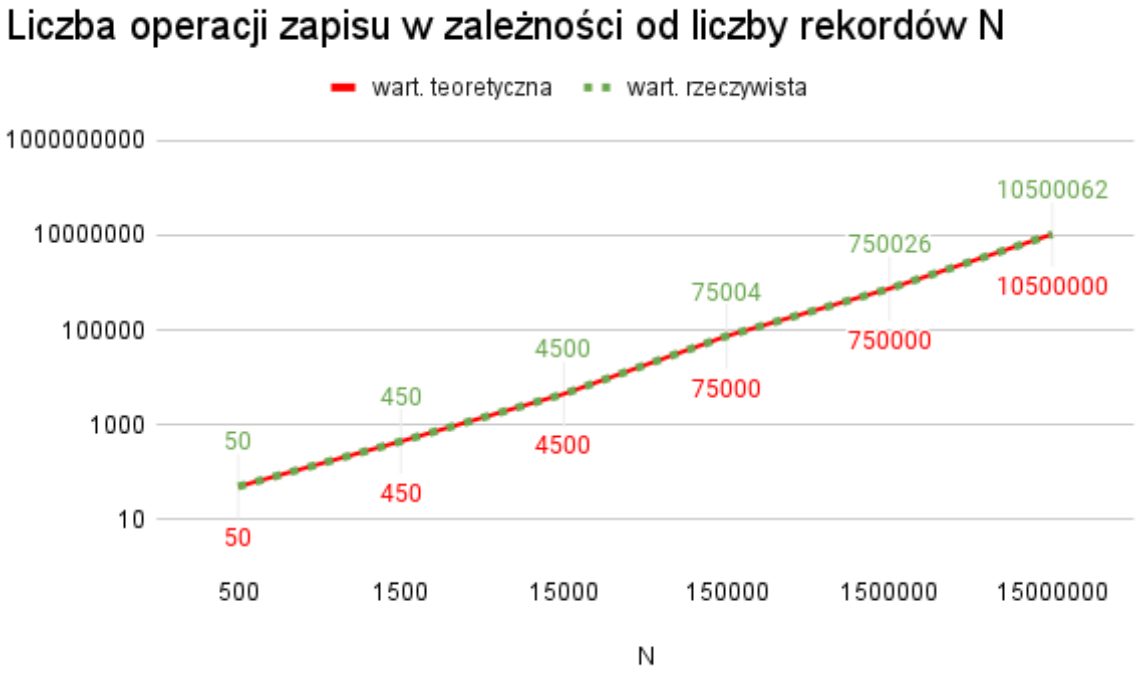
N	150000			
	Wart.teoretyczna	Wart.faktyczna	Różnica	%
Odczyty	75000	75004	4	0,005%
Zapisy	75000	75004	4	0,005%
Sortowania	149	149	0	0,000%
Cykle fazy 2.	2	2	0	0,000%
Operacje dyskowe	150000	150008	8	0,005%

N	1500000			
	Wart.teoretyczna	Wart.faktyczna	Różnica	
Odczyty	750000	750026	26	0,003%
Zapisy	750000	750026	26	0,003%
Sortowania	1486	1486	0	0,000%
Cykle	2	2	0	0,000%
Operacje dyskowe	1500000	1500052	52	0,003%

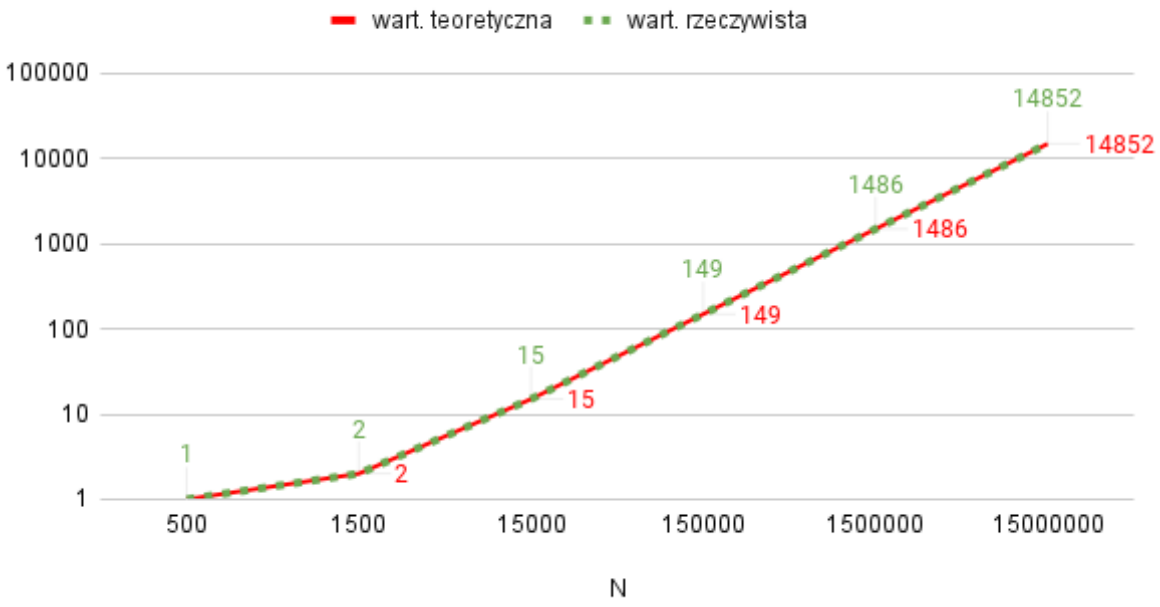
N	15000000			
	Wart.teoretyczna	Wart.faktyczna	Różnica	
Odczyty	10500000	10500062	62	0,001%
Zapisy	10500000	10500062	62	0,001%
Sortowania	14852	14852	0	0,000%
Cykle fazy 2.	3	3	0	0,000%
Operacje dyskowe	21000000	21000124	124	0,001%

Możemy zauważyć że gdy liczba cykli fazy 2. jest mniejsza niż 2, wyniki praktyczne w pełni pokrywają się z teoretycznymi. Dla większej liczby cykli wykonywane są dodatkowe operacje dyskowe. Jednak błąd względny nie przekracza 0,005%. Wynika to z faktu że przy dystrybucji w fazie 2 może zajść konieczność zapisania części bloku gdy rekordy w scalonym pliku są niekorzystnie ułożone. Mogą być niekorzystnie ułożone natomiast dlatego, że serie nie muszą być równej długości (gdy w etapie 1. mamy 150000000 rekordów i do 101 buforów ładujemy maksymalnie 10 rekordów otrzymamy 1470 serii zawierających 101*10 rekodów oraz 1 serię zawierającą 860 rekordów). Przy zapisywaniu oraz odczytywaniu części bloku wciąż używana jest operacja zapisu blokowego zatem faktyczna liczba operacji jest większa od przewidywanej.





Liczba sortowań w zależności od liczby rekordów N



Liczba cykli w zależności od liczby rekordów N