SPRAWOZDANIE Z PROJEKTU: STRUKTURY BAZ DANYCH

Autor: Krzysztof Taraszkiewicz 197796

Temat: Sortowanie z użyciem wielkich buforów
Data wykonania: 23.11.2025
Typ rekordu: Zbiór liczb (kryterium sortowania: suma elementów zbioru)

---

1. Opis zastosowanej metody

1.1. Wybór algorytmu

Do realizacji projektu wybrano metodę **sortowania z użyciem wielkich buforów** (merge sort with large buffers), która jest najbardziej efektywną metodą sortowania dużych plików dyskowych spośród prezentowanych na wykładzie i jest odmianą sortowania zewnętrznego.

1.2. Liczba zastosowanych taśm

Implementacja wykorzystuje model **1-taśmowy** z zapisywaniem in-place:

Jedna taśma (plik) pełni wszystkie role:

- **Etap 1** (tworzenie serii):
    - Odczyt: wczytanie **n×b** rekordów do pamięci
    - Zapis: natychmiastowe zapisanie posortowanej serii w tym samym miejscu na taśmie
- **Etap 2** (scalanie):
    - Odczyt: wczytanie **n-1** serii do buforów wejściowych
    - Zapis: natychmiastowe zapisanie scalonej serii od początku pierwszej z scalanych serii

Brak plików tymczasowych - wszystkie operacje wykonywane są na oryginalnym pliku **self.input_file.**

1.3. Zasada działania algorytmu

Algorytm działa w dwóch głównych etapach:

ETAP 1: Tworzenie początkowych serii

1. Wczytaj **n×b** rekordów z pliku do pamięci (**n** buforów × **b** rekordów na bufor)
2. Posortuj te rekordy w pamięci RAM algorytmem efektywnym (wbudowany algorytm pythona)
3. Zapisz posortowaną serię **natychmiast** do pliku **wejściowego** w miejsce nieposortowanych pobranych rekordów
4. Powtarzaj kroki 1-3 aż do końca pliku

Koszt Etapu 1: 2N/b operacji dyskowych (każdy rekord raz odczytany, raz zapisany)

ETAP 2: Scalanie serii

4. Użyj **n-1 buforów** jako buforów wejściowych i **1 bufora** jako bufora wyjściowego
5. Scal pierwsze **n-1** serii używając kolejki priorytetowej (heap)
6. Zapisz scaloną serie natychmiast w miejsce pobranej grupy serii w pliku wejściowym
7. Powtarzaj krok **5-6** dla kolejnych grup **n-1** serii
8. Powtarzaj kroki **5-7** aż pozostanie jedna seria (plik posortowany)

Liczba faz scalania: $\log_n(N/(n \times b))$
Koszt jednej fazy: 2N/b operacji dyskowych
Całkowity koszt: $2N/b \times (1 + \log_n(N/(n \times b)))$

1.4. Wykorzystanie kolejki priorytetowej

Do efektywnego scalania **n-1** serii wykorzystano **kopiec binarny** (heap):

- W kopcu przechowywane są rekordy z **n-1** buforów wejściowych
- Operacja pobrania minimum: **O(log n)**
- Operacja wstawienia nowego elementu: **O(log n)**
- Pozwala to na efektywne scalanie wielu serii jednocześnie

2. Szczegóły implementacji

2.1. Struktura programu

Program składa się z następujących głównych komponentów:

Klasa Record:

- Reprezentuje **rekord** jako **zbiór** liczb naturalnych
- Implementuje **porównywanie** na podstawie **sumy** elementów
- Metody serializacji/deserializacji do/z formatu tekstowego

Klasa DiskSimulator:

- **Symuluje** operacje blokowe na **dysku**
- Implementuje odczyt i zapis stron dyskowych
- Śledzi liczniki operacji **read_count** i **write_count**
- Parametry: blocking factor (**b**), rozmiar rekordu (**R**), rozmiar strony (**B=b×R**)

Klasa RunInfo:

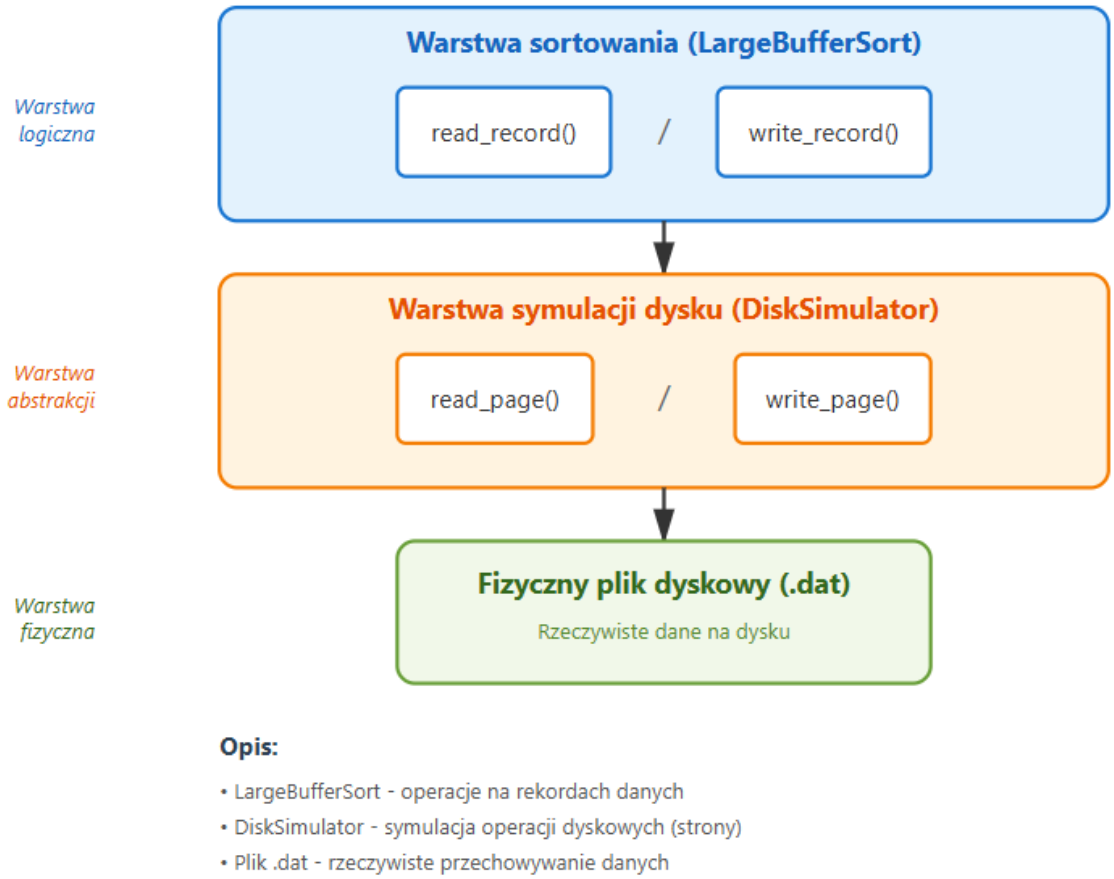
- Reprezentuje **logiczną serię** (ciąg posortowanych rekordów) w pliku
- Przechowuje pozycję serii: **start_page** (pierwsza strona) i **end_page** (strona za ostatnią)
- Pole **length** - długość serii w stronach (end_page - start_page)
- Używana do śledzenia granic serii podczas scalania

Klasa LargeBufferSort:

- Główna logika algorytmu sortowania
- Metoda **_create_runs()** - Etap 1
- Metoda **_merge_runs()** - Etap 2
- Metoda **_merge_multiple_runs()** - scalanie z użyciem heapa

2.2. Warstwa abstrakcji dyskowej

Symulacja operacji dyskowych została zaimplementowana jako oddzielna warstwa logiki:



Opis:

- LargeBufferSort - operacje na rekordach danych
- DiskSimulator - symulacja operacji dyskowych (strony)
- Plik .dat - rzeczywiste przechowywanie danych

2.3. Buforowanie

- Każdy bufor może pomieścić **b rekordów**
- W Etapie 1 używanych jest **n buforów** jednocześnie (n×b rekordów w RAM)
- W Etapie 2 używanych jest **n-1 buforów wejściowych + 1 bufor wyjściowy**
- Gdy bufor wyjściowy zapełni się, jego zawartość jest zapisywana na dysk

3. Specyfikacja formatów danych

3.1. Format pliku tekstowego (.txt)

Plik tekstowy zawiera rekordy w formacie:

liczba1,liczba2,liczba3,...

Przykład:

1,5,10,15

3,7,11

20,25,30,35,40

Każda linia reprezentuje jeden rekord (zbiór liczb oddzielonych przecinkami).

3.2. Format pliku binarnego (.dat)

Plik binarny zawiera strony dyskowe o stałym rozmiarze:

- **Rozmiar strony:** B = b × R bajtów
- **Rozmiar rekordu:** R = 128 bajtów (z paddingiem)
- **Blocking factor:** b = liczba rekordów na stronę

Każdy rekord jest zapisany jako string UTF-8 o długości **R** bajtów:

{1,5,10,15}\0\0\0\0...\0

3.3. Generator danych testowych

Program zawiera generator tworzący duże pliki testowe:

- Generuje n rekordów z losową liczbą elementów (**1-15**)
- Wartości elementów z zakresu **1-100**
- Możliwość generowania plików dowolnej wielkości

4. Sposób prezentacji wyników działania programu

4.1. Wyświetlanie zawartości pliku

Program pozwala wyświetlić zawartość pliku przed i po sortowaniu:

==========

Zawartość pliku: data.dat

==========

1. {5,10,15} (suma=30)

2. {1,2,3} (suma=6)

3. {20,25} (suma=45)

...

4.2. Informacje o przebiegu sortowania

Podczas sortowania wyświetlane są:

- Liczba utworzonych serii początkowych
- Numer aktualnej fazy scalania
- Liczba serii do scalenia w każdej fazie

4.3. Tryb szczegółowy (z fazami)

W trybie szczegółowym wyświetlana jest zawartość plików po każdej fazie (wyświetlane dane w trybie szczegółowym zapisywane są również w pliku tekstowym):

Etap 1: Tworzenie posortowanych serii...

Utworzono 20 serii początkowych

Serie początkowe:

Seria 1:

1. {1,2} (suma=3)

2. {5,7} (suma=12)

...

Etap 2: Scalanie serii...

Faza scalania 1: Scalanie 20 serii...

Po fazie 1:

Seria 1:

4.4. Statystyki końcowe

Po zakończeniu sortowania wyświetlane są:

Liczba faz scalania: 4

Liczba odczytów stron: 5000

Liczba zapisów stron: 5000

Łączna liczba operacji dyskowych: 10000

4.5. Export do pliku tekstowego

Program:

- Zapisuje dane wejściowe automatycznie do pliku data_input.txt
- Pozwala na zapisanie danych posortowanych do pliku data_sorted.txt

Format eksportu:

Liczba rekordów: 1000

==========

1. {1,2,3} (suma=6)

2. {4,5} (suma=9)

...

5. Opis eksperymentu

5.1. Cel eksperymentu

Eksperyment miał na celu:

1. Zbadanie zależności liczby faz sortowania od liczby rekordów
2. Zbadanie zależności liczby operacji dyskowych od liczby rekordów
3. Porównanie wyników praktycznych z teoretycznymi
4. Zbadanie wpływu liczby buforów (n) na efektywność sortowania

5.2. Parametry eksperymentu

Stałe parametry:

- Blocking factor: **b = 10** rekordów na stronę
- Rozmiar rekordu: **R = 128** bajtów
- Rozmiar strony: **B = 1280** bajtów

Zmienne parametry:

- Liczba rekordów: **N ∈ {100, 500, 1000, 5000, 10000, 20000, 50000, 100000}**
- Liczba buforów: **n ∈ {5, 50}**

5.3. Sposób przeprowadzenia eksperymentu

Dla każdej kombinacji parametrów (N, n):

1. **Generowanie danych**
    - Wygenerowano **N** losowych rekordów
    - Każdy rekord zawierał **1-15** losowych liczb z zakresu **1-100**
2. **Sortowanie**
    - Uruchomiono algorytm sortowania z wieloma buforami
    - Zmierzono liczbę faz scalania
    - Zliczono operacje odczytu i zapisu stron
3. **Obliczenia teoretyczne**
    - Liczba serii początkowych: $\mathbf{r = \lceil N/(n \times b) \rceil}$
    - Liczba faz: $\mathbf{\lceil \log_n(r) \rceil}$ (jeśli r > 1, inaczej 0)

- Liczba operacji: **2N/b × (1 + liczba_faz)**

4. **Zapis wyników**
    - Wyniki zapisano do pliku tekstowego
    - Wygenerowano wykresy porównawcze

5.4. Wzory teoretyczne

Liczba początkowych serii:

$r = \lceil N / (n \times b) \rceil$

Liczba faz scalania:

$fazy = \lceil \log_n(r) \rceil$ dla $r > 1$

$fazy = 0$ dla $r \leq 1$

Całkowita liczba operacji dyskowych:

$operacje = 2N/b \times (1 + fazy)$

gdzie:

- Etap 1: 2N/b (odczyt + zapis wszystkich rekordów)
- Etap 2: 2N/b × fazy (każda faza odczytuje i zapisuje wszystkie rekordy)

6. Wyniki eksperymentu

6.1. Wyniki dla n=5 buforów i b=10

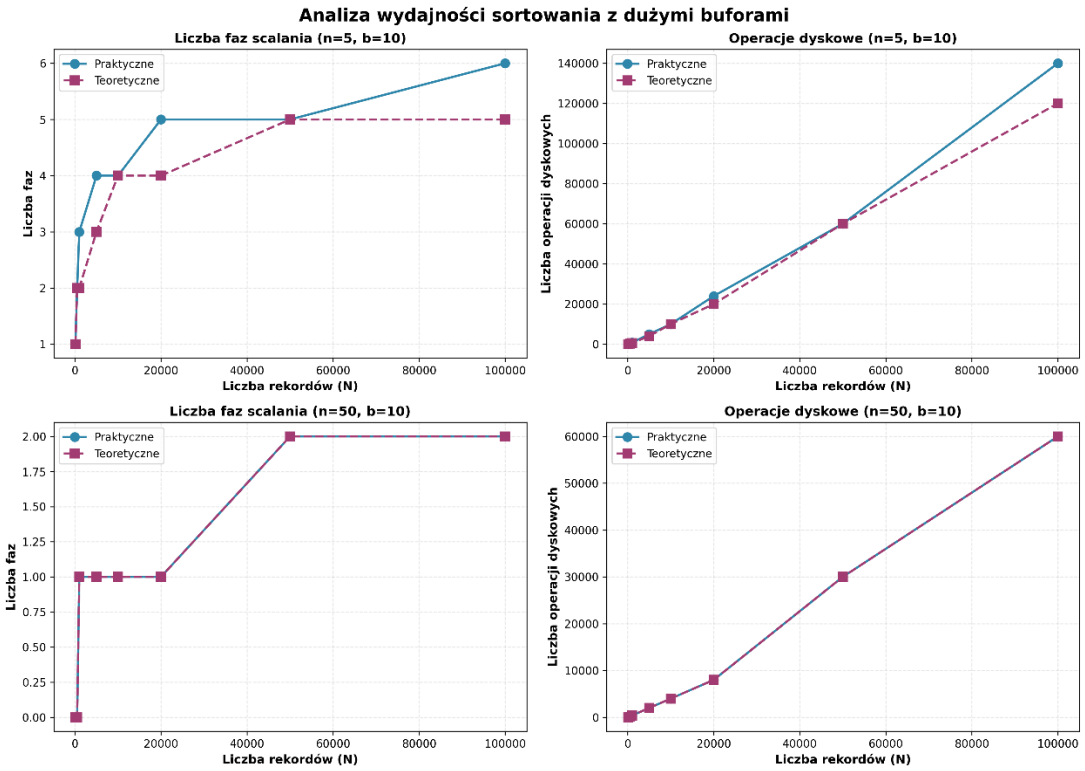
N	r	Fazy (P)	Fazy (T)	Operacje (P)	Operacje (T)
100	2	1	1	40	40
500	10	2	2	300	300
1000	20	3	2	800	600
5000	100	4	3	5000	4000
10000	200	4	4	10000	10000
20000	400	5	4	24000	20000
50000	1000	5	5	60000	60000
100000	2000	6	5	140000	120000

P = wyniki praktyczne, **T** = wyniki teoretyczne

6.2. Wyniki dla n=50 buforów i b=10

N	r	Fazy (P)	Fazy (T)	Operacje (P)	Operacje (T)
100	1	0	0	20	20
500	1	0	0	100	100
1000	2	1	1	400	400
5000	10	1	1	2000	2000
10000	20	1	1	4000	4000
20000	40	1	1	8000	8000
50000	100	2	2	30000	30000
100000	200	2	2	60000	60000

6.3. Wykres zbiorczy



6.4. Analiza wykresów

Wykres 1 i 2: Liczba faz scalania

- Dla małej liczby buforów (n=5): liczba faz rośnie **logarytmicznie**
- Dla dużej liczby buforów (n=50): liczba faz znacząco mniejsza
- Zgodność z przewidywaniami teoretycznymi (**$\log_n(r)$**)

Wykres 3 i 4: Liczba operacji dyskowych

- Zależność **liniowa** od liczby rekordów N
- Dla n=50 liczba operacji prawie dwukrotnie mniejsza niż dla n=5
- Znaczące rozbieżności dla n=5, idealnie nachodzące się wykresy dla n=50

7. Porównanie wyników praktycznych i teoretycznych

7.1. Zgodność wyników

Dla n=50 buforów: Wyniki praktyczne są **identyczne** z teoretycznymi:

- **Różnice** w liczbie operacji **nie** występują.
- Liczba faz: zgodność **100%**

Dla n=5 buforów: Występują większe rozbieżności:

- Dla N ≥ 1000: liczba faz praktyczna potrafi być większa o 1
- Liczba operacji wyższa o 20-33% dla dużych N

7.2. Przyczyny rozbieżności

1. **Nierówny podział serii**
    - Ostatnia seria może być niepełna
    - Teoretyczne wzory zakładają pełne serie
2. **Zaokrąglenia matematyczne**
    - Wzór $\lceil \log_n(r) \rceil$ może niedoszacowywać dla niektórych wartości r
    - Przykład: dla r=20 i n=5, $\log_5(20)=1.861$, $\lceil 1.861 \rceil=2$, ale praktycznie potrzeba 3 faz
3. **Dodatkowe operacje**
    - Operacje zapisu ostatniego niepełnego bufora
    - Operacje związane z zarządzaniem plikami tymczasowymi

7.3. Wnioski z porównania

1. **Dla dużej liczby buforów (n=50):**
    - Wyniki zgodne z teorią
    - Przewidywalność algorytmu bardzo wysoka
    - Mniejsze n/r (n>>r) skutkuje lepszą zgodnością
2. **Dla małej liczby buforów (n=5):**
    - Większe rozbieżności dla dużych N a małych ilości i wielkości buforów
    - Dłuższy czas wykonywania algorytmów niż w przypadku n=50
    - Znaczna liczba operacji dyskowych

3. **Ogólnie:**
    - Teoretyczne wzory dają dolne ograniczenie (optimistic estimate)
    - Praktyczne wyniki **nigdy** nie są gorsze niż **O(N log N)**
    - Wzrost liczby buforów drastycznie redukuje liczbę faz i operacji dyskowych

8. Wnioski końcowe

8.1. Efektywność algorytmu

1. **Sortowanie z wieloma buforami jest bardzo efektywne:**
    - Dla N=100 000, n=50 i b=10: tylko 2 fazy, 60 000 operacji
    - Dla porównania natural merge: około 320 000 operacji
2. **Kluczowy wpływ liczby buforów:**
    - Zwiększenie n z 5 do 50 daje 10-krotną redukcję liczby początkowych serii
    - Dla N=100 000: redukcja z 6 do 2 faz
    - Redukcja operacji: z 140000 do 60000
3. **Skalowalność:**
    - Algorytm dobrze skaluje się dla dużych plików
    - Złożoność $O(N \log_n N)$ jest osiągana w praktyce

8.2. Zalety implementacji

- Przejrzysta struktura kodu
- Dokładna symulacja operacji dyskowych
- Możliwość debugowania z wyświetlaniem faz
- Generator danych testowych i narzędzia do eksperymentów
- Export wyników do formatu tekstowego i graficznego

8.3. Obserwacje dodatkowe

1. **Optymalna liczba buforów:**
    - **Większe n** zawsze **lepsze** pod względem danych **liczbowych** (mniej faz)
    - W praktyce ograniczone dostępną pamięcią RAM
    - Trade-off: pamięć vs liczba faz
2. **Znaczenie blocking factor:**
    - **Większe b redukuje** liczbę operacji dyskowych
    - **b=10** w eksperymencie to wartość **demonstracyjna**
    - W praktyce b=100-1000 (strony 4KB-128KB)
3. **Zastosowania praktyczne:**

- Sortowanie bardzo dużych plików (setki GB)
- Systemy bazodanowe (operacje ORDER BY na dużych tabelach)
- Przetwarzanie danych typu big data

8.4. Podsumowanie

Implementacja algorytmu sortowania z użyciem wielkich buforów została wykonana zgodnie z wersją wykładową. Przeprowadzone eksperymenty potwierdziły teoretyczne przewidywania dotyczące złożoności algorytmu. Metoda ta jest znacząco bardziej efektywna od prostszych metod scalania (natural merge) i stanowi podstawę algorytmów sortowania w nowoczesnych systemach bazodanowych, np. PostgreSQL, Oracle Database, Microsoft SQL Server. Zwiększenie liczby buforów o rząd wielkości (z 5 do 50) skutkuje wielokrotną redukcją liczby faz i operacji dyskowych, co potwierdza fundamentalną rolę pamięci operacyjnej w efektywnym sortowaniu dużych plików.