SPRAWOZDANIE Z PROJEKTU: STRUKTURY BAZ DANYCH

Projekt nr 2: Indeksowa organizacja pliku (B-Drzewo)

Autor: Krzysztof Taraszkiewicz **Wydział:** ETI **Kierunek:** informatyka **Data:** 13.12.2025

---

1. Cel i treść zadania

Celem projektu było zaprojektowanie i zaimplementowanie indeksowej organizacji pliku w oparciu o strukturę **B-Drzewa**. Kluczowym aspektem zadania była symulacja obsługi pamięci zewnętrznej (dysku), realizacja operacji CRUD (Create, Read, Update, Delete) zgodnie z algorytmami przedstawionymi na wykładzie oraz przeprowadzenie eksperymentów wydajnościowych.

Zrealizowane podzadania:

1. **Symulacja dysku:** Implementacja zapisu blokowego na surowym pliku binarnym ze stałym rozmiarem strony (512 bajtów).
2. **B-Drzewo:** Implementacja struktury **B-drzewa** rzędu **d**, obsługująca operacje wstawiania, wyszukiwania i usuwania.
3. **Algorytmy Wykładowe:** Zastosowanie strategii **Bottom-Up** (wstępującej) z priorytetem **kompensacji** przed podziałem/scalaniem węzłów.
4. Implementacja mechanizmu **"Space Reuse"** (lista wolnych stron) dla pliku z danymi.
5. **Eksperymenty:** Analiza wpływu rzędu drzewa **d** na liczbę operacji I/O oraz rozmiar pliku indeksu.

---

2. Opis implementacji

2.1. Metoda indeksowania

Zastosowano klasyczne B-Drzewo, gdzie węzły przechowują klucze oraz wskaźniki (adresy stron) do rekordów w oddzielnym pliku danych.

- **Struktura strony:** Każda strona indeksu zawiera **m** kluczy, gdzie **d<=m<=2d** (za wyjątkiem korzenia).
- **Separacja danych:** Indeks (main_index.bin) jest oddzielony od danych (main_data.bin). Plik danych działa jak sterta (heap file) z możliwością ponownego wykorzystania zwolnionych stron.

2.2. Algorytmy operacji (Zgodność z wykładem)

W implementacji ściśle odwzorowano algorytmy przedstawione w materiałach dydaktycznych, stosując podejście **Bottom-Up**.

- **Wstawianie (INSERT):**
    1. **Nowy rekord** jest wstawiany fizycznie do liścia[1].
    2. Sprawdzany jest **warunek** przepełnienia (m > 2d).
    3. **Kompensacja (Priorytet):** W przypadku przepełnienia, algorytm najpierw próbuje przesunąć nadmiarowe klucze do sąsiada (lewego lub prawego), dokonując rotacji przez rodzica.
    4. **Podział (Split):** Dopiero gdy sąsiedzi są pełni, następuje podział węzła, a środkowy klucz jest promowany do rodzica. Proces może propagować się w górę aż do korzenia.
- **Usuwanie (DELETE):**
    1. Klucz jest **usuwany fizycznie** z liścia (w przypadku węzłów wewnętrznych następuje wcześniejsza zamiana z poprzednikiem).
    2. Sprawdzany jest warunek niedomiaru (m < d).
    3. **Kompensacja (Priorytet):** Algorytm sprawdza, czy można pożyczyć klucz od "bogatego" sąsiada (m > d).
    4. **Scalanie (Merge):** Jeśli kompensacja jest niemożliwa, następuje scalenie węzła z sąsiadem, pobierając klucz separatora z rodzica.

2.3. Symulacja pamięci zewnętrznej

Warstwa fizyczna (**DiskManager**) operuje na plikach binarnych:

- Plik jest **logicznie** podzielony na strony o rozmiarze PAGE_SIZE = 512 B.
- **Strona 0** jest zarezerwowana na metadane (ID korzenia, licznik stron).
- Obiekty są **serializowane** (pickle) i zapisywane bezpośrednio w wyznaczonych offsetach pliku, co symuluje dostęp swobodny (random access) do sektorów dysku.

---

3. Specyfikacja formatu pliku testowego

Program umożliwia wczytywanie komend z pliku tekstowego (tryb skryptowy). Każda linia zawiera jedną instrukcję.

Format komend:

- **Dodawanie:** ADD <Klucz> <Liczba1> <Liczba2> ...
    - Przykład: ADD 105 12 44 5
- **Usuwanie:** DEL <Klucz>
    - Przykład: DEL 105
- **Aktualizacja:** UPD <Klucz> <NowaLiczba1> <NowaLiczba2> ...
    - Przykład: UPD 105 99 88
- **Komentarze:** Linie zaczynające się od # są ignorowane.

---

4. Prezentacja wyników działania programu

Program działa w trybie interaktywnym oraz umożliwia wyświetlanie stanu struktur.

1. **Wyświetlanie Drzewa (print):** Prezentuje strukturę hierarchiczną indeksu, pokazując klucze w poszczególnych węzłach oraz ich relacje (rodzic-dziecko). Umożliwia weryfikację poprawności budowy B-Drzewa.

--- Struktura B-Drzewa ---

Strona 1: [30]

Strona 2: [10, 20]

Strona 3: [40, 50]

2. **Wyświetlanie Rekordów (scan):** Realizuje przegląd sekwencyjny (In-Order). Przechodzi przez indeks posortowany kluczami i dla każdego klucza pobiera odpowiedni rekord z pliku danych.

Klucz: 10 -> [ID: 10 | Liczby: [1, 2] | Suma: 3]

Klucz: 20 -> [ID: 20 | Liczby: [5, 5] | Suma: 10]

3. **Statystyki I/O:** Po każdej operacji program wyświetla liczbę wykonanych odczytów i zapisów stron dyskowych.
    - Przykład: IO: Odczyty: 4, Zapisy: 2

---

5. Eksperymenty

5.1. Metodyka

Przeprowadzono serię eksperymentów mających na celu zbadanie wpływu rzędu drzewa () oraz liczby rekordów () na wydajność operacji dyskowych i rozmiar plików.

- **Zmienne niezależne:**
    - Rząd drzewa .
    - Liczba rekordów .
- **Zmienne zależne (mierzone):**
    - Średnia liczba odczytów stron na jedną operację wstawiania.
    - Średnia liczba zapisów stron na jedną operację wstawiania.
    - Rozmiar pliku indeksu (w bajtach).
- **Przebieg:** Dla każdej pary wygenerowano losowe rekordy, wstawiono je do pustej bazy i zliczono operacje dyskowe.

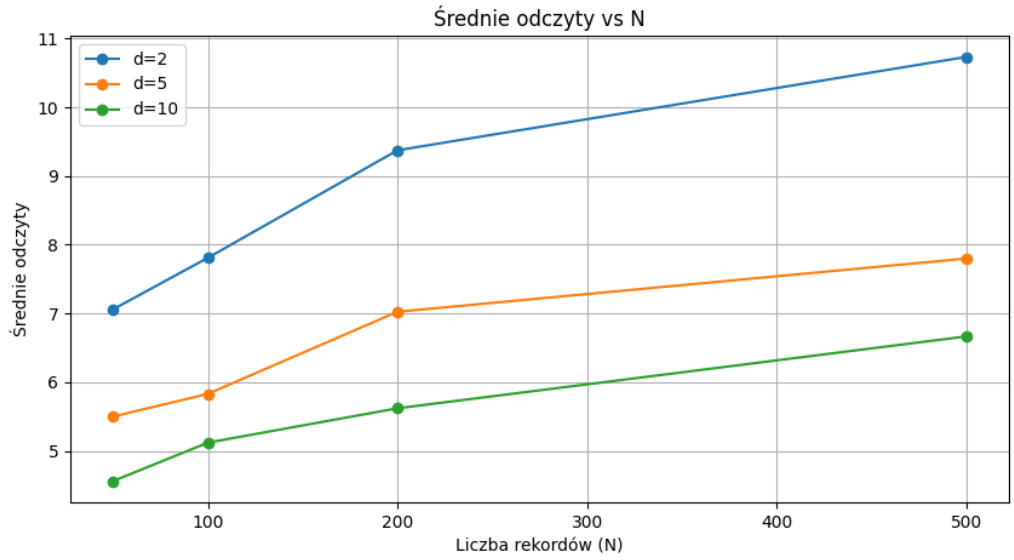
5.2. Wyniki eksperymentów

Tabela 1. Zestawienie wyników pomiarów (dla PAGE_SIZE = 512B)

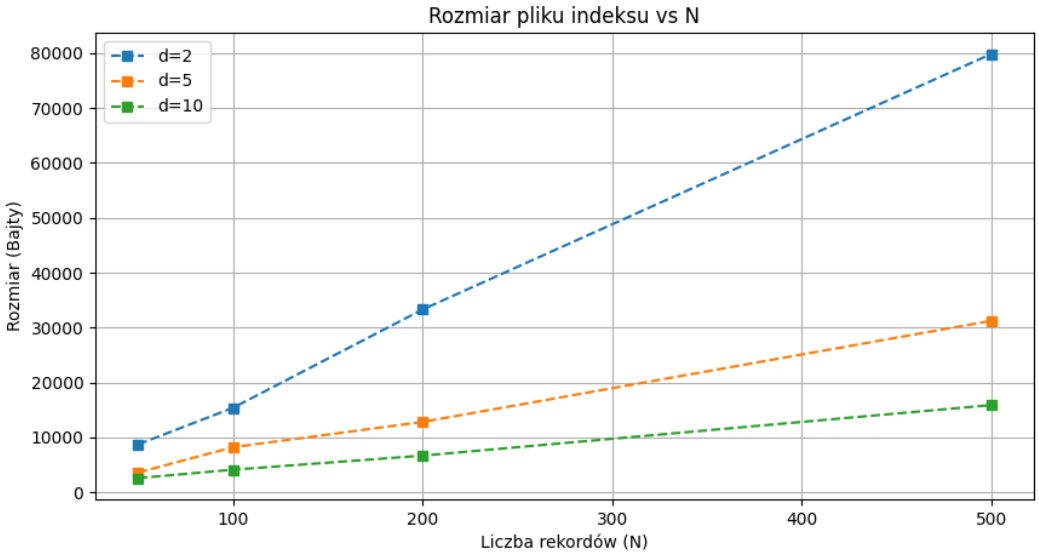
Rząd (d)	Liczba rekordów (N)	Śr. Odczyty	Śr. Zapisy	Rozmiar Indeksu (B)
2	50	7.06	3.80	8704
2	100	7.81	3.95	15360
2	200	9.38	4.08	33280
2	500	10.73	4.35	79872
5	50	5.50	3.38	3584
5	100	5.83	3.35	8192
5	200	7.03	3.62	12800
5	500	7.80	3.68	31232
10	50	4.56	2.66	2560
10	100	5.12	2.75	4096
10	200	5.62	3.17	6656
10	500	6.67	3.52	15872

5.3. Wykresy

Wykres 1. Średnia liczba odczytów dyskowych w funkcji liczby rekordów (N) dla różnych rzędów (d).



Wykres 2. Rozmiar pliku indeksu w funkcji liczby rekordów (N).



5.4. Analiza wyników i wnioski

1. **Wpływ rzędu** drzewa (d) na **liczbę odczytów**:
    - Zaobserwowano **wyraźną** zależność: im **większy** rząd drzewa, tym **mniej** odczytów dyskowych.
    - Dla d=2 (małe strony), drzewo jest wysokie i wąskie, co przy N=500 skutkuje średnio 10.73 odczytami na operację.

    - Dla d=10 (szerokie strony), drzewo jest niskie i płaskie, co redukuje liczbę odczytów do 6.67.
    - Potwierdza to teoretyczną złożoność **$O(\log_d N)$** – podstawa logarytmu rośnie wraz z **d**, zmniejszając wysokość drzewa.
2. **Kompensacja a Zapisy:** Mechanizm **kompensacji** pozwala utrzymać **zrównoważenie** drzewa bez częstych podziałów (split). Liczba zapisów jest **stabilna**, ale rośnie **wolniej** niż liczba odczytów, ponieważ **modyfikacje** często dotyczą tylko **liścia** i ewentualnie jego bezpośredniego sąsiada/rodzica, nie propagując zmian na całe drzewo.
3. **Utylizacja pamięci (Rozmiar pliku):**
    - Dla d=2 rozmiar indeksu przy 500 rekordach wynosi prawie 80 KB.
    - Dla d=10 ten sam zestaw danych zajmuje w indeksie tylko ~16 KB.
    - Wniosek: Wyższy **rząd** drzewa **drastycznie poprawia** efektywność wykorzystania miejsca na dysku. Węzły przy **małym d** mają **duży narzut** metadanych w stosunku do przechowywanych kluczy. Przy większym **d**, **stosunek** użytecznych danych do nagłówków strony jest **znacznie korzystniejszy**.

---

6. Podsumowanie

Projekt pozwolił na praktyczne zapoznanie się z problematyką organizacji danych na dysku. Kluczowe wnioski z realizacji to:

- Zastosowanie **B-Drzewa** znacząco **redukuje** liczbę dostępów do dysku w **porównaniu** do **struktur liniowych**, zwłaszcza dla dużych .
- Struktury o **wyższym** rzędzie **(d)** są znacznie **wydajniejsze** w środowisku pamięci zewnętrznej, minimalizując liczbę kosztownych operacji I/O oraz lepiej wykorzystując przestrzeń dyskową
- Mechanizm zwalniania miejsca **("Space Reuse")** zapobiega niekontrolowanemu rozrostowi pliku danych przy intensywnych operacjach usuwania i dodawania rekordów.
- Zastosowanie mechanizmów **kompensacji** przed podziałem/scalaniem węzłów skutecznie opóźnia wzrost wysokości drzewa.