Projekt 2: B-Drzewo

Jan Wisniewski 197662 Informatyka

Listopad 2025

Spis treści

1 Wstęp

Kluczową cechą B-drzewa jest jego optymalizacja pod kątem minimalizacji operacji wejścia-wyjścia na dysku. W tym celu, każdy węzeł B-drzewa jest przechowywany jako pojedynczy, spójny blok danych na dysku, często nazywany **stroną**. Ta stronicowana, **blokowa organizacja** pozwala na wczytanie całego węzła do pamięci operacyjnej za pomocą jednej, wydajnej operacji odczytu dysku, co jest kluczowe dla szybkości działania przy pracy z dużymi zbiorami danych.

W ramach projektu zaimplementowane zostały metody **wstawiania** i **usuwania**, **aktualizacji**, elementów w B-drzewie. Każda z tych operacji uwzględnia odpowiednią procedurę rozdzielania węzłów oraz łączenia węzłów, aby zapewnić równowagę drzewa.

2 Opis struktury

B-drzewo jest samobalansującą się strukturą danych, która umożliwia przechowywanie danych w formie drzewiastej, gdzie każdy węzeł może mieć wiele dzieci. Klucze w B-drzewie są przechowywane w węzłach w sposób posortowany, co umożliwia szybkie wyszukiwanie oraz modyfikację danych. B-drzewo zapewnia zrównoważony dostęp do danych w czasie logarytmicznym, co czyni je efektywnym w kontekście przechowywania dużych zbiorów danych.

B-drzewo charakteryzuje się kilkoma właściwościami:

1. Wszystkie liście znajdują się na tym samym poziomie, równym h;
2. Każda strona zawiera co najwyżej $2d$ kluczy;
3. Każda strona poza korzeniem zawiera co najmniej d kluczy (korzeń może zawierać 1 klucz);
4. Jeśli strona niebędąca liściem ma m kluczy, to ma $m + 1$ stron potomnych.

3 Struktura Plików i Węzłów

W implementacji B-drzewa zastosowano podział na dwa pliki dyskowe, co jest standardową praktyką w systemach zarządzania bazami danych:

- **Plik indeksowy (Index File)**: Przechowuje całą strukturę drzewa, czyli węzły. Każdy węzeł B-drzewa odpowiada jednej logicznej stronie (*page*) w tym pliku.
- **Plik rekordów (Record File)**: Przechowuje właściwe dane rekordów, do których odwołują się wskaźniki zawarte w kluczach w pliku indeksowym.

Struktura pliku indeksowego jest ściśle zdefiniowana w celu zapewnienia wydajnego dostępu blokowego:

1. **Strona Nagłówkowa (Header Page)**: Pierwsza strona pliku, zawierająca metadane, takie jak rząd drzewa (*tree order*), całkowita liczba stron (*page count*) oraz całkowita liczba rekordów (*record count*).

2. **Strony Węzłów (Node Pages)**: Wszystkie pozostałe strony to węzły. Każdy węzeł ma następującą wewnętrzną strukturę:

   - Liczba kluczy w węźle, k.
   - Następnie k elementów, z których każdy składa się z:
     - Klucza (*key-value*) i wskaźnika do rekordu (*record-pointer*) w pliku rekordów.
     - Wskaźnika potomnego (*pred-pointer*), wskazującego na węzeł zawierający klucze mniejsze niż ten klucz.
   - Na samym końcu węzła znajduje się dodatkowy wskaźnik potomny (*succ-pointer*), wskazujący na węzeł, który zawiera klucze większe niż wszystkie klucze w bieżącym węźle.

3.1 Operacje na B-drzewie

3.1.1 Wstawianie klucza

Podstawowa procedura wstawiania klucza do B-drzewa przebiega w kilku krokach:

1. **Inicjalizacja i Wczytanie**: Rozpocznij procedurę rekurencyjną od korzenia drzewa. W każdym kroku wczytaj węzeł z pliku indeksowego na podstawie jego adresu.

2. **Sprawdzenie Przepełnienia**: Sprawdź, czy bieżący węzeł jest pełny (`node-fullp`).

3. **Kompensacja (Optymalizacja)**: Jeśli węzeł jest pełny oraz jest liściem, spróbuj wykonać kompensację (`b-tree-compensation`)

4. **Podział**: Jeśli węzeł jest pełny, a kompensacja nie powiodła się lub nie jest możliwa, dokonaj **podziału węzła** (`b-tree-split-node`).

5. **Restart Ścieżki**: Jeśli nastąpił podział lub kompensacja, struktura drzewa uległa zmianie. Przeładuj referencję do węzła (węzła rodzica/nowego korzenia) i **powtórz proces szukania** miejsca w nowej strukturze.

6. **Znajdowanie Pozycji**: W aktualnym węźle znajdź pozycję dla klucza oraz wskaźnik do odpowiedniego węzła potomnego.

7. **Decyzja o Działaniu**:

   - **Węzeł Liść**: Jeśli węzeł jest liściem, wstaw klucz w znalezionej pozycji i zakończ rekurencję.

- **Węzeł Wewnętrzny**: Jeśli węzeł jest wewnętrzny, kontynuuj rekurencyjnie procedurę dla węzła potomnego (powrót do kroku 2).

3.1.2 Usuwanie klucza

Poniższy algorytm opisuje procedurę usuwania klucza x z B-drzewa.

1. **Wyszukiwanie**: Zlokalizuj klucz x w drzewie (wykorzystując algorytm wyszukiwania).

2. **Klucz Nieistniejący**: Jeśli klucz nie został znaleziony, zwróć `Not_Found`.

3. **Redukcja do Liścia**: Jeśli klucz x jest w węźle wewnętrznym, zastąp go kluczem **z liścia** (największym z lewego lub najmniejszym z prawego poddrzewa). Następnie usuń klucz zastępujący z liścia.

4. **Weryfikacja Liścia**: Po usunięciu, jeśli w liściu pozostaje $m \geq d$ kluczy, własność drzewa jest zachowana; zwróć `OK`.

5. **Kompensacja**: Jeśli nastąpił **UNDERFLOW** ($m < d$), spróbuj wykonać **kompensację** z sąsiednim węzłem. Jeśli się powiedzie, zwróć `OK`.

6. **Scalanie**: Jeśli kompensacja jest niemożliwa, wykonaj **scalenie** węzła z sąsiadem.

7. **Propagacja Niedopełnienia**: Jeśli po scaleniu niedopełnienie wystąpi w węźle przodka, przejdź do kroku 5, traktując węzeł przodka jako bieżący. W przeciwnym razie, zwróć `OK`.

3.1.3 Aktualizacja klucza

Procedura aktualizacji wskaźnika klucza w B-drzewie przebiega następująco:

1. **Wyszukiwanie Węzła**: Rozpocznij od korzenia drzewa i wykonaj standardowe wyszukiwanie, aby zlokalizować konkretną stronę (węzeł) zawierającą klucz przeznaczony do aktualizacji.

2. **Podmiana Wskaźnika**: Zmodyfikuj bezpośrednio wartość wskaźnika do rekordu stowarzyszonego z tym kluczem, zastępując stary adres nowym

3. **Zapis Zmian**: Zapisz zmodyfikowaną stronę węzła z powrotem do pliku indeksowego, utrwalając zaktualizowany wskaźnik na dysku.

4 Testowanie kodu i prezentacja implementacji

Implementacja B-drzewa została stworzona w języku programowania wysokiego poziomu **Common Lisp**, który oferuje dynamiczne środowisko pracy. W przeciwieństwie do tradycyjnych języków kompilowanych, charakteryzujących się cyklem **kompiluj-uruchom** (***compile-run cycle***), co znacząco wydłuża czas testowania, użyte środowisko umożliwia szybszą weryfikację.

Do testowania kodu oraz prezentacji jego działania zostana wykorzystane następujące narzędzia wbudowane w środowisko programowania:

- **REPL**: Interaktywna konsola pozwalająca na natychmiastowe wykonywanie pojedynczych instrukcji, co jest kluczowe dla szybkiej weryfikacji operacji wstawiania, usuwania i wyszukiwania.
- **Inspektor obiektów**: Umożliwia dynamiczne badanie wewnętrznego stanu węzłów i struktury drzewa w czasie rzeczywistym.

5 Eksperyment i Metodyka Testowania

Eksperyment polega na przetestowaniu algorytmu **wstawiania kluczy** w zaimplementowanym B-drzewie. Badanie zostanie przeprowadzone na zbiorach danych składających się z **100 000 rekordów**, co pozwoli na realistyczną ocenę wydajności struktury w warunkach dużego obciążenia dyskowego.

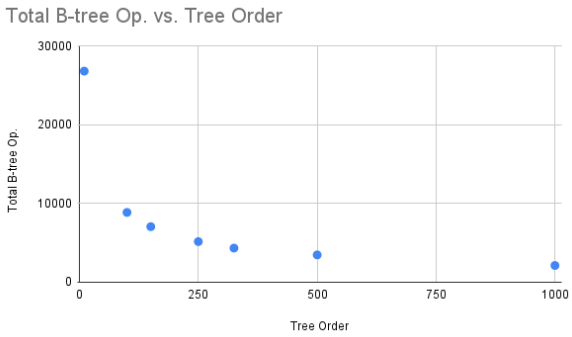
Głównym celem eksperymentu jest ocena **efektywności B-drzewa** w zależności od jego **rzędu** Pomiar efektywności będzie opierał się na analizie czasu wstawiania i liczby opracji io.

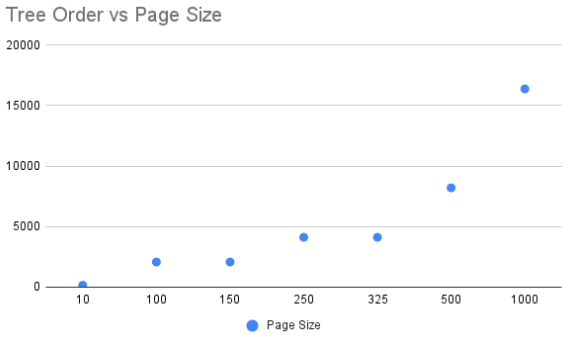
5.0.1 Zestawy Danych i Parametry

Testy zostaną przeprowadzone dla wartości rzędu drzewa $m = 10, 100, 150, 250, 325, 500, 1000$), co pozwoli na określenie optymalnej wielkości bloku dyskowego dla danego środowiska i zestawu danych.

Tree Order	Page Size	Split Count	Compensation Count	Read Count	Write Count	Time [s]
10	128	12708	14129	632512	175685	18.773
100	2048	1118	7752	313806	121859	11.515
150	2048	731	6328	297065	116401	12.434
250	4096	435	4728	255681	110600	14.441
325	4096	333	4005	211804	108211	15.318
500	8192	213	3259	207710	105626	18.490
1000	16384	107	2018	203990	101589	28.361

Tabela 1: Wyniki testu INSERTION-TEST dla różnych rzędu B-drzewa i rozmiarów strony





5.1 Analiza Wyników Eksperymentu

5.1.1 Wpływ Rzędu na Operacje Dyskowe i Strukturalne

- **Operacje I/O**: Wzrost Rzędu Drzewa i Rozmiaru Strony powoduje drastyczny spadek całkowitej liczby operacji IO, co potwierdza teoretyczne założenia B-drzewa. Największa redukcja IO nastąpiła przy przejściu z $m = 10$ na $m = 100$.
- **Operacje Strukturalne**: Liczba podziałów i kompensacji systematycznie maleje wraz ze wzrostem rzędu drzewa. Przy $m = 1000$ liczba podziałów jest minimalna (107), co świadczy o stabilności struktury.

5.1.2 Zależność Czasu Wstawiania

- **Minimalny Czas**: Najkrótszy całkowity czas wstawiania 11.515 s uzyskano dla **Rzędu Drzewa** $m = 100, 150$ dla strony 2048 B.
- **Wzrost Czasu**: Mimo dalszej minimalizacji operacji I/O, całkowity czas wstawiania zaczyna rosnąć dla rzędu powyżej $m = 150$, 28.361 s dla $m = 1000$.

5.1.3 Wniosek

Wzrost czasu przy bardzo dużych rzędach (np. $m = 1000$) sugeruje, że operacja stała się **związana z czasem procesora (CPU-bound)**. Procesor spędza więcej czasu na

wewnętrznym przeszukiwaniu dużej tablicy kluczy w pamięci RAM, niwelując zyski wynikające z mniejszej liczby odczytów dysku.

Optymalny Rząd Drzewa: Najbardziej efektywny dla tej implementacji i zestawu danych okazał się **Rząd Drzewa** $m = 100$, stanowiący najlepszy kompromis między minimalizacją kosztownych operacji dyskowych a wydajnością przetwarzania danych w pamięci. Sugeruje to też strone że strona na testowanym systemie plików ma 2048 B.

5.2 Raw Data

```
------------------------------
Running test INSERTION-TEST .
STAT:
 TREE-ORDER: 10
 PAGE-SIZE: 128
 COMPENSATION-COUNT: 14129
 READ-COUNT: 632512
 WRITE-COUNT: 175685
 SPLIT-COUNT: 12708
 MERGE-COUNT: 0
 Did 1 check.
    Pass: 1 (100%)
    Skip: 0 ( 0%)
    Fail: 0 ( 0%)

Evaluation took:
  18.773 seconds of real time
  18.578125 seconds of total run time (4.281250 user, 14.296875 system)
  [ Real times consist of 0.035 seconds GC time, and 18.738 seconds non-GC time. ]
  [ Run times consist of 0.031 seconds GC time, and 18.548 seconds non-GC time. ]
  98.96% CPU
  65,584,055,915 processor cycles
  1,039,445,840 bytes consed
------------------------------
Running test INSERTION-TEST .
STAT:
 TREE-ORDER: 100
 PAGE-SIZE: 2048
 COMPENSATION-COUNT: 7752
 READ-COUNT: 313806
 WRITE-COUNT: 121859
 SPLIT-COUNT: 1118
 MERGE-COUNT: 0
 Did 1 check.
    Pass: 1 (100%)
    Skip: 0 ( 0%)
    Fail: 0 ( 0%)

Evaluation took:
```

```
  11.515 seconds of real time
  11.406250 seconds of total run time (4.593750 user, 6.812500 system)
  [ Real times consist of 0.075 seconds GC time, and 11.440 seconds non-GC time. ]
  [ Run times consist of 0.046 seconds GC time, and 11.361 seconds non-GC time. ]
  99.05% CPU
  40,227,423,128 processor cycles
  2,684,440,976 bytes consed
--------------------------------
Running test INSERTION-TEST .
STAT:
 TREE-ORDER: 150
 PAGE-SIZE: 2048
 COMPENSATION-COUNT: 6328
 READ-COUNT: 297065
 WRITE-COUNT: 116401
 SPLIT-COUNT: 731
 MERGE-COUNT: 0
 Did 1 check.
    Pass: 1 (100%)
    Skip: 0 ( 0%)
    Fail: 0 ( 0%)

Evaluation took:
  12.434 seconds of real time
  12.421875 seconds of total run time (6.468750 user, 5.953125 system)
  [ Real times consist of 0.088 seconds GC time, and 12.346 seconds non-GC time. ]
  [ Run times consist of 0.125 seconds GC time, and 12.297 seconds non-GC time. ]
  99.90% CPU
  43,438,406,865 processor cycles
  3,222,396,192 bytes consed
--------------------------------
Running test INSERTION-TEST .
STAT:
 TREE-ORDER: 250
 PAGE-SIZE: 4096
 COMPENSATION-COUNT: 4728
 READ-COUNT: 255681
 WRITE-COUNT: 110600
 SPLIT-COUNT: 435
 MERGE-COUNT: 0
```

```
 Did 1 check.
    Pass: 1 (100%)
    Skip: 0 ( 0%)
    Fail: 0 ( 0%)

Evaluation took:
  14.441 seconds of real time
  14.375000 seconds of total run time (8.609375 user, 5.765625 system)
  [ Real times consist of 0.126 seconds GC time, and 14.315 seconds non-GC time. ]
  [ Run times consist of 0.078 seconds GC time, and 14.297 seconds non-GC time. ]
  99.54% CPU
  50,449,529,732 processor cycles
  4,879,049,360 bytes consed
--------------------------------
Running test INSERTION-TEST .
STAT:
 TREE-ORDER: 325
 PAGE-SIZE: 4096
 COMPENSATION-COUNT: 4005
 READ-COUNT: 211804
 WRITE-COUNT: 108211
 SPLIT-COUNT: 333
 MERGE-COUNT: 0
 Did 1 check.
    Pass: 1 (100%)
    Skip: 0 ( 0%)
    Fail: 0 ( 0%)

Evaluation took:
  15.318 seconds of real time
  15.140625 seconds of total run time (9.812500 user, 5.328125 system)
  [ Real times consist of 0.144 seconds GC time, and 15.174 seconds non-GC time. ]
  [ Run times consist of 0.171 seconds GC time, and 14.970 seconds non-GC time. ]
  98.84% CPU
  53,513,679,870 processor cycles
  5,267,848,512 bytes consed
--------------------------------
Running test INSERTION-TEST .
STAT:
 TREE-ORDER: 500
```

```
 PAGE-SIZE: 8192
 COMPENSATION-COUNT: 3259
 READ-COUNT: 207710
 WRITE-COUNT: 105626
 SPLIT-COUNT: 213
 MERGE-COUNT: 0
 Did 1 check.
    Pass: 1 (100%)
    Skip: 0 ( 0%)
    Fail: 0 ( 0%)

Evaluation took:
  18.490 seconds of real time
  18.312500 seconds of total run time (12.187500 user, 6.125000 system)
  [ Real times consist of 0.227 seconds GC time, and 18.263 seconds non-GC time. ]
  [ Run times consist of 0.250 seconds GC time, and 18.063 seconds non-GC time. ]
  99.04% CPU
  64,593,839,485 processor cycles
  7,504,893,824 bytes consed
--------------------------------
Running test INSERTION-TEST .
STAT:
 TREE-ORDER: 1000
 PAGE-SIZE: 16384
 COMPENSATION-COUNT: 2018
 READ-COUNT: 203990
 WRITE-COUNT: 101589
 SPLIT-COUNT: 107
 MERGE-COUNT: 0
 Did 1 check.
    Pass: 1 (100%)
    Skip: 0 ( 0%)
    Fail: 0 ( 0%)

Evaluation took:
  28.361 seconds of real time
  27.984375 seconds of total run time (20.906250 user, 7.078125 system)
  [ Real times consist of 0.409 seconds GC time, and 27.952 seconds non-GC time. ]
  [ Run times consist of 0.281 seconds GC time, and 27.704 seconds non-GC time. ]
  98.67% CPU
```

```
99,078,138,029 processor cycles
13,302,930,560 bytes consed
```