Author: iceshaver@int.pl 30.12.2018

Struktury Baz Danych – sprawozdanie z projektu II

Wstęp

Zadaniem projektowym było zaprojektowanie i zaimplementowanie indeksowej organizacji pliku. W ramach jego realizacji wybrałem wariant z B+-drzewem. Rekordami przechowywanymi w tak skonstruowanej bazie danych są studenci i ich trzy oceny oraz unikalny klucz. Struktura rekordu wraz z rozmiarami przedstawia się następująco:

Klucz (8 B)	Id Studenta (5 B)	Ocena 1 (1 B)	Ocena 2 (1 B)	Ocena 3 (1 B)

Zaimplementowane operacje oraz algorytmy:

- wstawianie rekordu
- odczyt rekordu
- usuwanie rekordu
- aktualizacja rekordu
- przeglądanie całej zawartości pliku i indeksu zgodnie z kolejnością wartości klucza
- mechanizm pozwalający na ponowne wykorzystanie zwalnianego przy usuwaniu miejsca

Ponadto przeprowadziłem eksperyment dotyczący liczby dostępów do dysku podczas pracy z bazą danych, w zależności od stopni węzłów wewnętrznych i liści, którego wyniki znajdują się na końcu tego sprawozdania.

Działanie programu

Projekt został wykonany jako aplikacja CLI działająca w terminalu pod systemem Linux, napisana w języku C++. Używa biblioteki boost oraz Graphviz. Do wyświetlanie drzewa w formie graficznej wymaga zainstalowanej przeglądarki plików SVG. Po uruchomieniu programu i wpisaniu polecenia *help* otrzymujemy listę wszystkich dostępnych komend.

```
Available commands:
close               Save and close db file
create              Create new record
delete              Delete record
draw                Draw and display tree as svg picture
exit                Close DB file and Exit program
file                Print content of db file in human readable form to stdout
gentestfile         Generate random test file with specified size (if not size is random
help                Prints this help
lastop              Last operation statistics
load                Load test file
ls                  Print all records in order by key value
lsd                 Print all records in order by key value (descending)
new                 Create new db file at specified location
nodes               Print all nodes of tree to stdout
open                Open specified db file
read                Read record
stats               Print DB statistics
truncatetree        Remove all records, clean db file
update              Update record
> _
```

Fig 1: Dostępne polecenia

Zaraz po starcie programu, możemy wykonać następujące operacje:

- *new [nazwa pliku]* - stworzenie nowej bazy danych, jako argument tego polecenia musimy podać nazwę pliku, w którym będzie ona trzymana
- *open [nazwa pliku]*- otwarcie wcześniej stworzonego pliku bazy danych
- *gentestfile [nazwa pliku] [liczba operacji]*– wygenerowanie tekstowego pliku testowego zawierającego sekwencję operacji dodawania, aktualizacji oraz usuwania rekordów, który można wczytać do stworzonej bazy danych. Argumenty: [nazwa pliku, który będzie zawierał testowe polecenia] [ilość operacji (jeżeli nie podano zostanie wybrana losowo)]
- *exit lub ctrl+d* – wyjście z programu

Po stworzeniu nowej bazy danych lub otwarciu istniejącej, dostępne są następujące komendy:

- *create [klucz] [rekord]* – stworzenie nowego rekordu, rekord w tym przypadku jest trzema ocenami studenta w zakresie 0 – 100.
- *read [klucz]* – odczyt i wypisanie na standardowe wyjście rekordu o podanym kluczu
- *update [klucz] [rekord]* – aktualizacja rekordu o podanym kluczu do danych podanych jako [rekord]
- *delete [klucz]* – usunięcie rekordu o podanym kluczu
- *ls* oraz *lsd* – wypisanie na standardowe wyjście wszystkich rekordów z bazy danych wg klucza (rosnąco oraz malejąco)
- *nodes* – wypisanie na standardowe wyjście wszystkich węzłów składających się na B+-drzewo
- *file* – wyświetlenie struktury pliku na standardowym wyjściu w formie czytelnej dla człowieka
- *draw* – rysowanie B+-drzewa w formie graficznej i otwarcie go w domyślnej dla systemu przeglądarce plików SVG za pomocą xdg-open

- *load [nazwa pliku]* – ładuje plik testowy
- *stats* – wypisuje statystyki związane z bazą danych i działającym programem, tj.: ścieżka do pliku bazy danych, rozmiar tego pliku, typ przechowywanych danych oraz klucza, stopień i liczbę węzłów wewnętrznych oraz liści, liczbę rekordów, wysokość drzewa, liczbę przechowywanych węzłów w pamięci operacyjnej, liczbę odczytów i zapisów na dysk twardy (liczonych od otworzenia bazy danych) oraz procent użycia przestrzeni dyskowej w pliku bazy danych.
- *lastop* – Wypisuje liczbę odczytów i zapisów na dysk twardy wykonanych przez ostatnią operację
- *truncatetree* – usuwa zawartość bazy danych
- *close* – zapisuje i zamyka bazę danych
- *exit lub ctrl+d* – zapisuje, zamyka bazę danych oraz cały program

Szczegóły implementacji

Odczytywanie i zapisywanie węzłów

Odczytywanie i zapisywanie węzłów na dysk twardy odbywa się w sposób prawie niewidoczny dla operujących na nich algorytmów dzięki wykorzystaniu mechanizmu sprytnych wskaźników. Węzeł jest ładowany do pamięci za pomocą funkcji *readNode*, która jako argument pobiera offset węzła w pliku i zwraca wskaźnik na załadowany węzeł, a gdy taki węzeł zniknie z zakresu widoczności (wskaźnik zostanie usunięty), węzeł zostanie automatycznie zapisany z powrotem na dysku twardym (o ile był zmieniony). W razie potrzeby użycia już wcześniej załadowanych do pamięci węzłów, np. przy schodzeniu w głąb drzewa podczas dodawania nowego rekordu, gdzie możemy być zmuszeni w pewnym momencie do cofnięcia się po przodkach aż do korzenia, w węźle zapisywany jest wskaźnik na jego bezpośredniego rodzica, co tworzy listę węzłów idącą ku górze, aż do korzenia. Korzeń rezyduje w pamięci operacyjnej przez cały czas działania programu (stale istniejący wskaźnik na węzeł korzenia) na danej bazie danych, gdyż i tak jest on wykorzystywany przy każdej operacji. Jest on odczytywany zaraz po otworzeniu bazy danych i zapisywany w trakcie jej zamykania lub zamykania całego programu, dlatego ważne jest, aby mając otwartą bazę danych wychodzić z programu poleceniem *exit* lub kombinacja klawiszy ctrl+d, a nie np. ctrl+c, gdyż wtedy prawie na pewno uszkodzimy plik bazy danych. Takie zabiegi sprawiają, że w pamięci musi być miejsce na h + 1 węzłów (gdzie h to wysokość drzewa), aby można było bezproblemowo przeprowadzać wszystkie operacje (maksymalną liczbę węzłów załadowanych do pamięci w czasie działania programu można podejrzeć wykorzystując polecenie *stats*).

Plik bazy danych

Plik bazy danych zawierający B+-drzewo składa się z nagłówka konfiguracyjnego (zawierającego informację o położeniu korzenia w pliku i stopniach węzłów) oraz następujących po sobie zserializowanych węzłów, które to z kolei składają się z nagłówka, który określa jakiego typu jest węzeł i czy jest to miejsce wolne czy zajęte oraz tablicy kluczy i potomków (w przypadku węzłów wewnętrznych) lub tablicy kluczy i rekordów (w przypadku liści). Strukturę pliku najlepiej oddaje wynik polecenia *file.*

Zastosowano tutaj następujące konwencje:

- po lewej stronie jest offset pod jakim dany węzeł w pliku się znajduje
- LNode oznacza liść (LeafNode), a INode oznacza węzeł wewnętrzny (InnerNode)

- Zawartość danego węzła znajduje się w klamerkach { dane węzła }
- Klucze znajdują się wewnątrz trójkątnych nawiasów <klucz>
- Rekordy znajdują się wewnątrz okrągłych nawiasów (id_studenta ocena1 ocena2 ocena3)
- Liczby bez nawiasów to offsety wskazujące na potomków

```
> open db
db> file
0:      ConfigHeader {rootOffset: 3080, innerNodeDegree: 1, leafNodeDegree: 4}
24:     header, LNode: 24 { <9>(89 26 43 87)<167>(70 90 87 86)<171>(50 74 22 23)<172>(48 80 15 68)<187>(59 20 17 88)<200>(3 79 26 68)}
281:    header, LNode: 281 { <605>(6  5 83 89)<616>(58 49 10  8)<622>(51 84 31 47)<635>(78 31 79 61)<642>(26 15 99 88)<650>(40 76 13 16)<652>(52 50 77 37)}
538:    header, INode: 538 { 24<200>1809<276>2661}
619:    header, LNode: 619 { <497>(45  9 18  5)<505>(88 25 33 55)<538>(19 88 92 88)<556>(54 22 99 69)<559>(13 64 23  9)<566>(37  3 67 78)<581>(8 67 82 36)<596>(36 22  5  7)}
876:    header, LNode: empty
1133:   header, INode: 1133 { 1295<749>3675}
1214:   header, INode: 1214 { 538<486>2918}
1295:   header, LNode: 1295 { <719>(41 33 43 64)<723>(35 13 76 18)<732>(98 89 30100)<741>(86 16 93 71)<749>(39 32 68 54)}
1552:   header, LNode: 1552 { <832>(80 14 76 55)<849>(42 86 57 60)<854>(12 50 49 69)<855>(44 21 28 21)}
1809:   header, LNode: 1809 { <251>(24 98 74  8)<263>(47 78 38 97)<267>(64 91 38 75)<276>(46 84 97 64)}
2066:   header, INode: empty
2147:   header, LNode: empty
2404:   header, LNode: 2404 { <656>(91 70 56 69)<658>(94 47 56  0)<660>(29 13 55 74)<665>(79 79 73 82)<666>(21 29 62  9)<667>(67 80  8  1)<678>(5 53 67 49)<686>(38 91 59 23)}
2661:   header, LNode: 2661 { <286>(73 35 44  0)<355>(90 68 65 90)<395>(65 94 79 91)<438>(63 84 36 44)<456>(74 83 90 80)<463>(49 32  2 21)<469>(72 32 40 90)<486>(16 29 54 55)}
2918:   header, INode: 2918 { 619<596>281<652>2404}
2999:   header, INode: 2999 { 1133<827>3932}
3080:   header, INode: 3080 { 1214<686>2999}
3161:   header, LNode: 3161 { <896>(75  2 26 11)<918>(1  0 11 82)<951>(81 78  0 77)<967>(61 95 70 33)}
3418:   header, LNode: empty
3675:   header, LNode: 3675 { <763>(10 70 54 62)<792>(99 82  5  1)<793>(53 97 86 34)<794>(17 28 97 84)<827>(4 15 79 61)}
3932:   header, INode: 3932 { 1552<855>3161}
db> _
```

Fig 2: Struktura pliku bazy danych

Ponowne wykorzystanie miejsca po usuniętych węzłach

Zajmuje się tym funkcja *AllocateDiskMemory,* która na podstawie podanego typu węzła, szuka dla niego miejsca w pliku przeglądając go od początku do końca, co robi skacząc po nagłówkach węzłów i sprawdzając czy dane miejsce jest wolne i czy zgadza się typ węzła. Jeżeli tak to zwraca offset wolnego miejsca do wykorzystania, a jeżeli takie miejsce się nie znajdzie, rezerwuje nowa przestrzeń na końcu pliku, powiększając tym samym jego rozmiar.

Struktura pliku testowego

Plik testowy jest zwykłym plikiem tekstowym zawierającym dowolną ilość linii, z których każda może zawierać jedno z następujących poleceń:

- *create [klucz] [rekord]*
- *update [klucz] [rekord]*
- *delete [klucz]*

Istnieje także możliwość podania tutaj dowolnych innych komend prawidłowych dla tej aplikacji, ale na potrzeby realizacji tego projektu najważniejsze są powyższe trzy. Polecenia można komentować znakiem #.

Prezentacja wyników działania programu

Aplikacja umożliwia prezentowanie wyników na 7 sposobów:

- wypisanie pojedynczego rekordu, polecenie *read [klucz]*
- wypisanie wszystkich rekordów za pomocą polecenia *ls* lub *lsd*
- wypisanie wszystkich węzłów składających się na drzewo poleceniem *nodes*

- wypisanie zawartości pliku bazy danych w czytelny dla człowieka sposób poleceniem *file*
- narysowanie drzewa poleceniem *draw*
- wypisanie statystyk pracy programu oraz informacji o drzewie poleceniem *stats*
- wypisanie liczby operacji dyskowych wykonanych podczas realizacji ostatniej operacji, polecenie *lastop*

Eksperyment

Badanie liczby dostępów do dysku podczas pracy z bazą danych, w zależności od stopni węzłów wewnętrznych i liści.

Metodologia:

Użytkowanie bazy danych jest symulowane wczytaniem pliku testowego z sekwencją poleceń dodawania, aktualizacji, usuwania oraz odczytu zarówno pojedynczych rekordów jak i wszystkich rekordów sekwencyjnie.

Liczba poleceń: 100 000

Na początku dodanie 10 000 rekordów

W dalszej części większość poleceń stanowią odczyt losowy oraz sekwencyjny (>80%), reszta to aktualizacja, usuwanie oraz dodawanie nowych rekordów

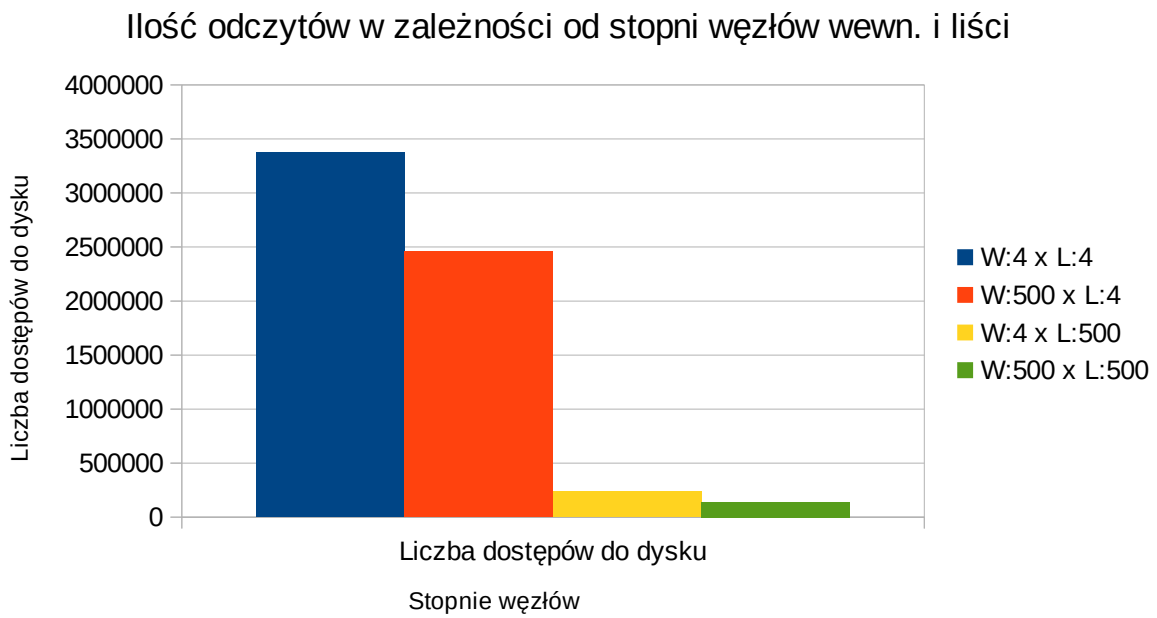
Stopnie węzłów zostają ustalone dla każdego pomiaru w poniższej konfiguracji:

- węzeł wewn.: 4, liść: 4
- węzeł wewn.: 500, liść: 4
- węzeł wewn.: 4, liść: 500
- węzeł wewn.: 500, liść: 500

Wyniki:

Stopień węzła wewn. x st. liścia	Liczba dostępów do dysku
W:4 x L:4	3376444
W:500 x L:4	2460152
W:4 x L:500	233609
W:500 x L:500	140944

Tabela 1: Liczba operacji dyskowych w zależności od stopnia węzła



Wnioski

Powyższy wykres pokazuje przede wszystkim wyraźną różnicę (o rząd wielkości) pomiędzy liczbą dostępów do dysku przy stopniu liścia 4 oraz 500. Jest ona znacznie większa, niż różnica pomiędzy tą liczbą przy zmianie stopnia węzła wewnętrznego z 4 na 500, co wskazuje na to, że w celu zminimalizowania narzutu opóźnienia dotyczącego operacji dyskowych należy przede wszystkim zwiększać stopień liścia. Oczywiście musi to być dokonane w ramach dostępnej pamięci operacyjnej, gdyż musi ona pomieścić co najmniej h węzłów wewnętrznych i co najmniej dwa liście.