Sprawozdanie z projektu nr.2 ze Struktur Baz Danych

I. WSTĘP

Projekt zrealizowałem implementując indeksową organizację pliku jaką jest struktura indeksowo - sekwencyjna.

Organizacja ta składa się z primary area - część podstawowa, która jest podzielona na strony o rozmiarze PAGESIZE - współczynnik blokowania oraz z overflow area - część nadmiarowa. Trafiają tam rekordy, które nie zmieszczą się na danej stronie w primary area.

Wszystkie rekordy są posortowane rosnąco wg. klucza.

Typem mojego rekordu są równoległoboki - długości boków i jeden z kątów. Dlatego też za klucz przyjąłem wartości pól, to znaczy iloczyn dwóch boków (side1 i side2 oraz kąta - angle).

Każdy rekord składa się z klucza (key), boku (side1), boku (side2) oraz kąta (angle) oraz wskaźnika (z primary area wskazuje na rekord w overflow area, natomiast wsaźnik w overflow area wskazuje na kolejny rekord w overflow area). Jeśli wskaźnik jest równy -1 oznacza, ze nie wskazuje na żaden rekord z primary area na overflow lub w overflow nie wksazuje na żaden inny rekord lub jest ostatnim rekordem w overflow.

Primary area i overflow area to plik danych (file), natomiast Index area zawiera indeks rzadki stworzony z pierwszego klucza na danej stronie i numeru danej strony.

II. Zaimplementowane zmienne, struktury i funkcje

FILEINDEX - plik indeksu, jako “index”

TEMPDATA - plik tymczasowy, jako “temp”

FILEDATA - plik danych ,jako “data”

PAGESIZE - rozmiar strony / współczynnik blokowania

ALPHA - współczynnik alpha

INDEX_SIZE - rozmiar Index w bajtach (liczba bajtów potrzebna do przechowywania obiektu typu Index)

RECORD_SIZE - rozmiar Record w bajtach (liczba bajtów potrzebna do przechowywania obiektu typu Record)

struct GlobalVariables - konstruktor struktury ustawia wartości 4 zmiennych typu int:

recordsPrimaryArea - ilość rekordów w primary area = 0

recordsOverflowArea - ilość rekordów w overflow area = 0

maxNumberRecordsPrimary = 4 oraz ***maxNumberRecordsOverflow*** = połowa z maxNumberRecordsPrimary

struct Index - reprezentuje indeks, który zawiera klucz i numer strony typu int.

struct Record - inicializuje klucz (key), 2 boki (side1 i side2), kąt (angle) i wskaźnik (pointer). Klucz i wskaźnik na -1, pozostałe na wartość 0.

void init() - inicjalizuje pierwszy rekord o kluczu najmniejszym z możliwych (bo pole, nie może być równe 0) - 1 (bok, bok, kat odpowiednio - 1, 1, 90).

void addRecord() - dodanie rekordu. W przypadku, gdy liczba rekordów w overflow jest równa maksymalnej liczbie rekrdów w overflow area wykonuje reorganizację.

Wartość klucza przyjąłem jako iloczyn boku, boku oraz kąta.

Tworzę dynamicznie alokowaną tablice rekordów o rozmiarze PAGESIZE, gdzie każdy element tablicy jest obiektem typu Record. Na samym końcu zwalniam pamięć poprzez delete[], aby zapobiec wycieku pamięci.

Otwieram plik FILEDATA w trybie binarnym i wskaźnik fileRead umożliwia czytanie danych binarnych z tego pliku.

Otwieram plik tymczasowy (TEMPDATA) w trybie binarnych, wskaźnik fileWrite umożliwia zapis zmodyfikowanych danych do pliku tymczasowego.

Przeszukuje strony, celem znalezienia strony, na której ma być wstawiony dany rekord, dzięki funkcji searchIndex. Następnie sprawdzam, czy na stronie jest wolne miejsce (key == -1), jesli jest to dodaje rekord i przepisuje overflow area. W przypadku braku miejsce dodaje rekord do overflow area. Tam weryfikuje, czy mój klucz na stronie jest większy od klucza, który chce wstawić, jeśli tak to oznacza, że klucz, który chce wstawić musi być tuż przed tym kluczem. Zmniejszam i o jeden żeby wskazać na rekord, z którego będę pointował na overflow i dzięki temu będę mógł ustawić wskaźnik na ten rekord w overflow.

Rozpatruje przypadek, gdy mam np. w primary: 1 3 8 9 i w overflow 15. Chce dodać 5. Zatem dzięki zmniejszeniu i klucz będzie wskazywał na 3 i wskaznik jego to -1. Zatem dla tego klucza ten wskaznik muszę ustawić na nowo wstawiany rekord, który jest mniejszy niż ten, który obecnie się tam znajduje (15).

Jeszcze sprawdzam czy obecny pointer jest większy od klucza, dlaczego?

```
if (buffer[i].pointer == -1 || buffer[i].pointer > key)
```

Przykład: w primary mam 1 3 8 9 i w overflow 6. Wskaźnik na rekordzie o kluczu 3 jest ustawiony na 6 i chce wstawić 5. Dlatego wskaznik ten powinien byc ustawiony na 5 (bo klucz 5 będzie przed 6 w overflow area). Zatem wskaźnik nie

jest rowny -1, wiec rozpatruje przypadek lub, gdzie wskaznik jest wiekszy niż ten, ktory wstawiam zatem ustawiam, ten wskaźnik na klucz który chce wstawić (5).

Kolejno przechodzę przez rekordy w overflow i weryfikuje czy jest wolne miejsce (buffer[j].key == -1) lub czy klucz, który chce dodać jest mniejszy już od obecnego (key < buffer[j].key). Gdy jest to spełnione to przesuwam rekordy w góre od indeksu j, tak żeby wstawić w odpowiednie miejsce ten nowy rekord. Następnie aktualizuje wskaźniki.

```
if (buffer[j].key == -1 || key < buffer[j].key) {
    // przesuwam rekordy w prawo, aby zrobić miejsce dla nowego rekordu
    for (int k = globalVariable.maxNumberRecordsOverflow - 1; k > j; k--)
    {
        // robie miejsce dla rekordu o pozycji j, przesuwajac wszystkie
        // ktore sa od niego wieksze w gore w overflow (albo w prawo :)
        // jak kto woli nazywac :)) )
        buffer[k] = buffer[k - 1];
    }
```

void createIndex() - funkcja przechodzi przez każdą stron i po odczytaniu rekordu przypisuje klucz pierwszego rekordu na stronie do INDEX, jak i również numer strony.

void indexSearch() - przechodzę przez wszystkie strony (page = -1, bo nie znalazłem jeszcze strony). Dla każdego rekordu sprawdzam, czy jego klucz jest większy od szukanego klucza. Jeśli tak to oznacza, że rekord, którego szukam jest na tej stronie i ustawiam odpowiednio stronę.

void indexShow() - funkcja iteruje po wszystkich rekordach i wyświetla klucz i numer strony dla każdego z nich.

void fileShow() - funkcja wyświetla obszar primary area -strony i rekordy na tych stronach oraz obszar overflow area.

void find() - funkcja przeszukuje przez wszystkie strony i po znalezieniu tej, w której znajduje się dany klucz iteruje przez klucze na znalezionej stronie i klucz na stronie jest równy szukanemu kluczowi następuje znalezienie. Gdy nie znajdę rekordu w primary area, muszę szukać rekordu w overflow area. Tam przechodzę przez wszystkie (maksymalną liczbę rekordów w overflow area) rekordy w overflow area i sprawdzam dopasowanie obecnych tam kluczy wraz z szukanym kluczem.

void reorg() - funkcja przechodzi przez strony w primary area i przechodzi przez rekordy na stronach. Jeśli liczba rekordów załadowanych do bufora przenośnego (bufferMove - służy do tymczasowego przeniesienia rekordów) osiągnie wartość zapełnienia (wartość alpha * rozmiar strony) dla każdego rekordu pozbywam się jego wskaźnika (w buforze przenośnym), tak aby rekordy nie miały już powiązań do rekordów w obszarze przepełnienia (bo rekordy z primary area pointują na rekordy w overflow area). Następnie zapisuje rekordy z bufora przenośnego do tymczasowego. Muszę też usunąć rekordy z przenośnego po zapisie, tak aby zrobić miejsce dla kolejne (dzięki funkcji removeFromBuffer).

Załadowanie rekordów z primary area wygląda następująco:

Sprawdzam, czy klucz jest różny od -1, czyli czy rekord w ogóle znajduje się na stronie. Jeśli tak to zapisuje go na odpowiednim miejscu w buforze przenośnym (bufferMove) i ustawiam wskaźnik na wartość aktualnego rekordu (gdy pointer jest rozny od -1 to muszę przejść do overflow, bo oznacza to, że rekord pointuje na overflow).

Teraz jedna z najważniejszych aspektów - visit. Męczyłem się z tym bardzo długo, jak rozwiązać problem tego, że w przypadku, gdy mam 2 pointery lub więcej na jednej stronie to po pierwszym przejściu / odwiedzeniu overflow area pobierałem/przenosiłem do bufora (bufferMove) po kolei wszystkie rekordy (potem i tak będę je sortował) i skutkowało to tym, że jak już je przeniosłem to nic tam nie zostało. W momencie, gdy ponownie pointowałem do overflow to odczytywałem -1, jako klucz. Dlatego też muszę tam wejść tylko za 1 razem i już więcej tam nie wchodzić i dzięki visit = 1 to następuje, bo po pierwszym wejściu zmniejszam te wartość o 1 i już nie będzie równe 1. Z każdym odczytywanym rekordem czy to w primary area, czy w overflow muszę zweryfikować czy liczba rekordów w buforze przenośnym (recordsCounter) jest równa współczynnikowi

wypełnienia. Jeśli tego wypełnienia nie ma to wpisuje odpowiednie (idx_ov dba o indeks pozycji w overflow area) do bufora przenośnego rekordy z obszaru przepełnienia. Następnie ustawiam wskaźnik (bo warunkiem pętli jest miedzy innymi to, ze wykonuje sie dopoki wskaznik jest różny od -1), aby dowiedzieć się czy jest tam jeszcze jakiś rekord (w overflow każdy rekord wskazuje na kolejny). W przypadku, gdy zostały jeszcze jakieś rekordy w buforze przenośnym to muszę je zapisać do pliku.

Maksymalną liczbę rekordów w primary area ustalam na liczbę stron po reorganizacji razy rozmiar strony.

W przypadku, gdy maksymalna liczba rekordów w primary area podzielona przez 2 jest większa od 4 (czyli mam więcej niż 2 strony, bo strona ma 4 rekordy (2 strony dadzą 8 rekordów)) to ustalam maksymalną liczbę rekordów w overflow area na 4. Gdy jest jedna strona lub 2 (czyli jest max 8 rekordów) to wtedy liczba rekordów w overflow area będzie równa maksymalnej liczbie rekordów w primary area podzielonej na 2.

Po reorganizacji muszę dopisać do rekordów primary area, te które przeniosłem z overflow area, czyli po prostu liczbę rekordów overflow area. I oczywiście po reorganizacji ustawiam liczbę rekordów w overflow area na 0.

Następnie sortuje rekordy z TEMPDATA, ponieważ mam tam rekordy z primary area, odczytane raz wszystkie na raz z overflow area i nie są one posortowane. Dlatego też odczytuje odczytuje tyle rekordów ile jest w TEMDATA o rozmiarze RECORD_SIZE z TEMPDATA i zapisuje je zaczynając od adres &records[0] (wskaznik na pierwszy element wektora records). Następnie wybieram te rekordy, które mają klucz różny od -1, bo przecież tylko rekordy, które mają klucz mogę ze soba posortować dlatego dodaje je do wektora sortableRecords. Klucze z sortableRecords sortuję bąbelkowo. Po posortowaniu wstawiam z powrotem w odpowiednie miejsca do wektora ‘records’. Po wstawieniu zapisuje je spowrotem.

void removeFromBuffer - funkcja ustawia klucz oraz wskaźnik na -1, a side1, side2 i angle na 0. Używana do usuwania (ustawiania wyżej wymienionych wartości) w buforze tymczasowym, gdy przestają już być potrzebne.

void sort() - algorytm sortowania bąbelkowego, który porządkuje tablicę rekordów “Record” według wartości klucza w porządku rosnącym. Działa poprzez iteracyjne porównywanie sąsiednich elementów i zamienianie ich miejscami, przesuwając największe wartości na koniec tablicy w każdej kolejnej iteracji.

Zasady buforowania:

Alokuję dynamicznie bufor bufferPrimary (przy użyciu new), który przechowuje rekordy z obszaru głównego (*primary*).

Rozmiar bufora wynosi PAGESIZE rekordów. Po zakończeniu jego użycia zwalniam go za pomocą “delete[] buffer”.

Podobnie wygląda to z bufferOverflow, gdzie rozmiar bufora wynosi (maksymalną liczbę rekordów w overflow) rekordów.

bufferMove tworzę tak samo jak bufferPrimary, gdzie jego rozmiar wynosi PAGESIZE rekordów.

Następnie otwieram plik “FILEDATA” w trybie odczytu binarnego, z którego będę odczytywał dane.

Tak samo otwieram plik “TEMPDATA” w trybie zapisu binarnego, do którego będę zapisywał dane.

```
// bufor na rekordy z obszaru primary
Record* bufferPrimary = new Record[PAGESIZE];
// bufor na rekordy z obszaru overflow
Record* bufferOverflow = new Record[globalVariable.maxNumberRecordsOverflow];
// bufor na rekordy, ktore beda przenoszone do pliku po reorganizacji
Record* bufferMove = new Record[PAGESIZE];

FILE* fRead = fopen(FILEDATA, "rb");
FILE* fWrite = fopen(TEMPDATA, "wb");
```

Oby odczytywać dane z pliku i zapisywać je używam odpowiednio funkcji fread i fwrite.

W przypadku funkcji “fread” bufor, do którego zostaną zapisane odczytane dane ze strumienia fileRead (FILEDATA) to buffer.

Rozmiar jednego elementu danych (w bajtach) to RECORD_SIZE.

Maksymalna liczba elementów jaka może zostać odczytana to PAGESIZE.

W przypadku funkcji “fwrite” bufor z danymi, które mają zostać zapisane do strumienia fileWrite (TEMPDATA) to buffer.

Rozmiar jednego elementu danych (w bajtach) to RECORD_SIZE.

Maksymalna liczba elementów jaka może zostać zapisana do wskazanego strumienia to PAGESIZE.

```
fread(buffer, RECORD_SIZE, PAGESIZE, fileRead);
read++;
fwrite(buffer, RECORD_SIZE, PAGESIZE, fileWrite);
write++;
```

III. Specyfikacja formatu pliku tekstowego

Plik tekstowy składa się z rekordów, gdzie w przypadku dodawania nowego rekordu należy na początku podać nazwę operacji (add). Kolejną wartością jest bok równoległoboku (side1), następną jest drugi bok równoległoboku (side2) i na końcu kąt (angle). Przykład:

```
add 92 2 60
add 31 2 0
add 22 2 30
add 67 2 60
add 39 2 90
```

Kolejne opcje, które dodałem to możliwość ilustacji wizualnej primary area i overflow area, dzięki komendzie - “show file” oraz ilustracji Indexu - “show index”.

Przykład wywołania:

```
show file
show index
```

Przykładowy wynik:

```
PRIMARY AREA


PAGE 1

1 1 1 90
3 3 2 30 pointer: 5
8 8 2 30
9 9 2 30 pointer: 15

OVERFLOW AREA

5 5 2 30 pointer: 15
15 15 2 30
reads: 2

INDEX AREA

Key: 1 Page: 0

reads: 1
```

Istnieje również możliwość znalezienia poszukiwanego klucza poprzez wpisanie “find” oraz tuż po tym klucz, który chcemy znaleźć.

Przykład:

```
find 28
find 26
find 27
find 171
```

W przypadku wywołania reorganizacji należy wpisać “reorganize”.

Przykład:

```
reorganize
```

Przykładowe wywołanie i wynik:

Wywołanie:

```
add 1 2 30
add 3 2 30
add 8 2 30
add 9 2 30
add 15 2 30
add 5 2 30
add 7 2 30
find 8
find 21
reorganize
show file
show index
exit
```

Rezultat:

```
(base) lukasz@MacBook-Air-ukasz sbd_2 % ./index_seq
Init values:
reads = 1 writes = 3
ADDED NEW RECORD OF KEY: 1
Choice 'keyboard' or 'file' and input that:
file
Input file name:
t.txt
Already exist
reads = 2 writes = 0
ADDED NEW RECORD OF KEY: 3
reads = 4 writes = 3
ADDED NEW RECORD OF KEY: 8
reads = 4 writes = 3
ADDED NEW RECORD OF KEY: 9
reads = 4 writes = 3
ADDED NEW RECORD OF KEY: 15
reads = 3 writes = 2
ADDED NEW RECORD OF KEY: 5
reads = 3 writes = 2
ADDED NEW RECORD OF KEY: 7
reads = 11 writes = 10
FOUND RECORD ON PAGE: 1 WITH KEY: 8 side1: 8 side2: 2 angle: 30
reads: 2
KEY: 21 DOESN'T EXIST
reads: 4
REORGANIZING
reads: 9 writes: 7
PRIMARY AREA
```

```
PAGE 0

1 1 1 90
3 3 2 30
-1 0 0 0
-1 0 0 0

PAGE 1

5 5 2 30
7 7 2 30
-1 0 0 0
-1 0 0 0

PAGE 2

8 8 2 30
9 9 2 30
-1 0 0 0
-1 0 0 0

PAGE 3

15 15 2 30
-1 0 0 0
-1 0 0 0
-1 0 0 0
```

```
OVERFLOW AREA

-1 0 0 0
-1 0 0 0
-1 0 0 0
-1 0 0 0
reads: 5

INDEX AREA

Key: 1 Page: 0
Key: 5 Page: 1
Key: 8 Page: 2
Key: 15 Page: 3

reads: 1
```

IV. Tryb z klawiatury

Aby wejść do trybu wpisywania z klawiatury należy wpisać “keyboard”

```
Choice 'keyboard' or 'file' and input that:
keyboard
```

Następnie wyświetla się menu wyboru i mogę wybrać opcję, która mnie interesuję:

```
make a choice :
1 - add record:
2 - show index
3 - show file
4 - reorganize
5 - find
6 - end
```

Po wybraniu opcji add należy wpisać bok, bok i kąt

```
Input side1:
2
Input side2:
2
Input angle:
30
ADDED NEW RECORD OF KEY: 2
Added new record
```

Po wybraniu opcji wyświetlenia primary area wygląda to następująco:

```
Your choice is: 3

PRIMARY AREA


PAGE 0

1 1 1 90
2 2 2 30
-1 0 0 0
-1 0 0 0

OVERFLOW AREA

-1 0 0 0
-1 0 0 0
reads: 2
```

Po wybraniu opcji wyświetlenia indexu wygląda to następująco:

```
INDEX AREA

Key: 1 Page: 0
Key: 3 Page: 1
Key: 7 Page: 2
```

Po wybraniu opcji znalezienia klucza wygląda to następująco:

```
Input key to find it and values:
5
FOUND RECORD ON PAGE: 0 WITH KEY: 5 side1: 5 side2: 2 angle: 30
```

Po wybraniu opcji reorganizacji wygląda to w następujący sposób:

```
REORGANIZING:
reads: 6 writes: 6
```

```
PRIMARY AREA


PAGE 0

1 1 1 90
2 2 2 30
-1 0 0 0
-1 0 0 0

PAGE 1

3 3 2 30
5 5 2 30
-1 0 0 0
-1 0 0 0

PAGE 2

7 7 2 30
15 15 2 30
-1 0 0 0
-1 0 0 0

OVERFLOW AREA

-1 0 0 0
-1 0 0 0
-1 0 0 0
-1 0 0 0
```

V. Eksperymenty

Wykonałem 4 eksperymenty bazujące na dodawaniu rekordów. Wygenerowałem losowo odpowiednio 50, 100, 200, 300 operacji dodania rekordu: “add side1 side2 angle”. Następnie do tych samych danych dopisałem w losowych miejscach wywołanie reorganizacji, co skutkowało wykonywaniem reorganizacji w losowych momentach. Chciałem dzięki temu sprawdzić jaki wpływ ma reorganizacja na średnią liczbę odczytów i zapisów i porównać ze sobą wywoływanie reorganizacji tylko wtedy, gdy overflow jest całe zapełnione i chce dodać nowy rekord (bez reorganizacji w losowych miejscach) z wywoływaniem reorganizacji (poprzez dodanie w losowych miejscach wywołania reorganizacji) wraz z jej naturalnym wywołaniem (gdy overflow jest pełne i chce dodać nowy rekord)

Dla każdego z nich obliczyłem średnią liczbę odczytów i zapisów, która obliczyłem dzięki sumowaniu liczby odczytów i zapisów (zmienne zdefiniowane w GlobalVariables jako totalReads i totalWrites), po wykonaniu się funkcji addRecord i następnie dzieląc wynik poprzez liczbę rzeczywiście dodanych rekordów (kontroluje to zmienna “added” zdefiniowana w GlobalVariables). Na potrzebę eksperymentu przyjąłem współczynnik ALPHA równy 0.25, 0.5, 0.75 oraz 1, maksymalną ilość rekordów w primary area = 4, maksymalną ilość rekordów w overflow area jako połowa maksymalnej ilości z primary area oraz rozmiar strony ustawiony na 4.

Wyniki eksperymentów:

Współczynnik ALPHA = 0.25

Liczba wygenerowanych operacji dodawania rekordu	Średnia liczba odczytów	Średnia liczba zapisów
50	13	11
100	27	23
200	52	45
300	94	82

Wraz z losową ilością reorganizacji w pliku:

Liczba wygenerowanych operacji dodawania rekordu	Średnia liczba odczytów	Średnia liczba zapisów
50	19	16
100	34	29
200	61	53
300	118	104

Współczynnik ALPHA = 0.5

Liczba wygenerowanych operacji dodawania rekordu	Średnia liczba odczytów	Średnia liczba zapisów
50	9	8
100	17	14
200	31	27
300	59	50

Wraz z losową ilością reorganizacji w pliku:

Liczba wygenerowanych operacji dodawania rekordu	Średnia liczba odczytów	Średnia liczba zapisów
50	10	8
100	18	15
200	32	28
300	61	53

Współczynnik ALPHA = 0.75

Liczba wygenerowanych operacji dodawania rekordu	Średnia liczba odczytów	Średnia liczba zapisów
50	8	7
100	14	11
200	24	20
300	44	37

Wraz z losową ilością reorganizacji w pliku:

Liczba wygenerowanych operacji dodawania rekordu	Średnia liczba odczytów	Średnia liczba zapisów

50	8	6
100	14	12
200	22	19
300	43	36

Współczynnik ALPHA = 1

Liczba wygenerowanych operacji dodawania rekordu	Średnia liczba odczytów	Średnia liczba zapisów
50	7	6
100	13	11
200	22	18
300	40	33

Wraz z losową ilością reorganizacji w pliku:

Liczba wygenerowanych operacji dodawania rekordu	Średnia liczba odczytów	Średnia liczba zapisów
50	7	6
100	13	10
200	21	18
300	39	33

Wnioski z eksperymentów:

Średnia liczba odczytów i zapisów wzrasta ze względu na zwiększenie stron w primary area. Liczba odczytów jest większa od liczby zapisów ze względu na wyszukiwanie klucza w indeksie.

Po wykonaniu widać, że liczba średnich odczytów i zapisów zaczyna zbiegać się do podobnego wyniku w miarę zwiększania wartości ALPHA. Dlaczego?

Domyślam się, że przez to, że im większa wartość ALPHA tym mniej stron muszę przetworzyć, bo zmniejsza mi się ich ilość tworzenia się podczas reorganizacji.

Widać, że współczynnik 0.25 jest dość słaby, ponieważ liczba odczytów i zapisów jest porównywalnie do innych współczynników wysoka szczególnie dla coraz to większej liczby rekordów. Współczynnik ALPHA 0.5 w porównaniu do współczynnika 0.75 gorzej radzi sobie z coraz to większą liczbą rekordów. Natomiast współczynnik 0.75 i 1 nie różnią się od siebie naprawdę w małym stopniu. Uważam, że rozsądnym jest wybór pomiędzy 0.5 a 0.75. Możliwe, że współczynnik 0.7 byłby najlepszym wyborem. Dzięki temu pozostawię miejsce na nowe rekordy i zwiększę efektywność operacji wyszukiwania klucza, ze względu na potrzebną mniejszą liczbę stron po reorganizacji.

Z kolei wraz z dodaniem wywołania reorganizacji w losowych momentach widać, że dla współczynnika 0.25 i 0.5 liczba średnich odczytów i zapisów rośnie. Natomiast dla współczynnika 0.75 i 1 zaczyna w bardzo małym stopniu maleć.

Dzieje się tak, ponieważ przy współczynniku 0.25 i 0.5 musi powstać, więcej stron po reorganizacji w porównaniu ze współczynnikiem 0.75 i 1. Skutkuje to większą ilością stron, a co za tym idzie potrzebą większej ilości odczytów i zapisów.