R04921094 EECS 葉孟元
R05921099 EECS 振　昊

Project report: B+ Tree Index

1. 項目簡介

請麻煩先讀 README.txt 再看這份報告。為了自己能夠詳細敘述，本報告以中文為主，會夾雜英文專有名詞說明。這份報告可能會比較長，因為自己花了很多心血在這個 project 上，也希望能夠有一個比較詳細的報告。以下內容分為，第 2 節：項目分工。第 3 節：代碼描述，詳細介紹每一個.java 檔的實作方法和自己的假設，包括對 print page 的格式描述。第 4 節：正確性測試，本節會用簡單的 test_data.txt 來構建一個 Integer B+ Tree。第 5 節：重複 Key，對於重複 Key 的一些自己的想法。第 6 節：總結，總結這次報告的心得和實作上遇到的問題。

2. 項目分工

在本項目中我們分別對以下事件進行了施作。

項目	分工
所有代碼	R04921094 葉孟元
測試資料	R05921099 振昊：初始 test_data.txt R04921094 葉孟元: 最後版本 test_data.txt
項目測試	R04921094 葉孟元
項目報告	R04921094 葉孟元

R04921094 EECS 葉孟元
R05921099 EECS 振　昊

3. 代碼描述

本次 project 使用 java，並如 README.txt 所示，我用一個文檔輸出方便助教查看結果。首先，先對這個項目做一個概括。原本非常願意，也非常希望能夠真的架構出一個 Page，然後讓所有的 Index Node 和 Leaf Node 都存放在 page 裡面，但是後來發現這樣就需要針對 char 做不同的操作，而無法使用 java STL。會導致工作量很大，而一個人寫代碼沒有辦法負荷。並且，java 要碰到 disk 的資料也比較麻煩。於是，自己就開不同的 class 來架構出這個 project，在 Tree 上算是多少有模擬的成分，是後來老師也覺得可以使用在 Memory 中實作，所以就順理成章實作下來了。在 Page 上是利用 LinkedList 來維護了一個 Page 的結構，裡面也算是有真實的欄位，只是利用了 LinkedList 畢竟也不能算是完完全全意義上 512 bytes Page。總之，雖然自己測試都做出來了，自己也是知道這並不是一個真的放在 Page 裡面而可以減少 I／O 的結構。

接下來我會針對每個 class 做描述和介紹。自己的假設的演算法的實作會跟這些 class 裡面一同說明。

Type class：enum class，這個 class 是為了限制輸入的 type 只能是 Integer 和 String 兩個型態。

KeyType class：這個 class 是為了處理 Integer 和 String 的兩種不同樹而產生的，在建構的時候會將 Type 和 value 一起傳入。本 class 實作了 Comparable Interface，為了能夠使用 Collections.binarySearch() 來做 query，insert，delete.

Record class：存放在 Page 中的真實資料，由於不在意 memory size 於是為了方便將 key，rid，record 都放在這個 class 中。但是在存取的時候還是通過 pageId 來存取在 Page 中的 record。

Page class：用來存放 Record 的地方，完全按照 project 的要求做，Book-keeping 並不放任何資料。不同的是，我考慮了 garbage collection，因為本次的 project 限制 pageId 為 2 bytes，這樣說明只用 256*256 個 page 可以使用。如果 B+ tree 過大，而不斷插入和刪除的時候，就會出現過多的 garbage，因為 free space 只能不斷往後指。garbage collection 演算法就是基本的將 slotted page 重新架構，把帶有-1 的 slot number 都刪除，重新設定每一餓 slot number，最佳化這個 page 的內容。

Page 的輸出格式：每一個 record 用 “，” 隔開，並且用 char 方式輸出。在沒有 record 的地方用 “*” 代表，最後的 slot directory 分辨一個一個用 Integer 輸出。

R04921094 EECS 葉孟元
R05921099 EECS 振　昊

PageManager class：在每個 BplusTree（等等會介紹）中都有一個 PageManager，主要功能是為了能夠集中管理，這個 Tree 裡面的 rid 分配和將 record 放入 page 中。在此說一下這個類別的演算法，首先先維持一個 page pointer 指向 pageId = k 的 page，如果要加入一個 record，就問 page k 是否允許我放 record 進去。如果可以就結束，如果不可以就說明 free space 不夠了。會直接將 k+1 page，而不考慮 k 的 free space（讓之後的 garbage collection 處理）。如果 page pointer 指到 pageId = 256*256 時，並且還不能插入 record 時。開始做 garbage collection（不夠用了再做）。並設定 page pointer = 0，繼續一個一個 page 問，如果有就插入 record，如果還是沒有足夠的空間，就說明記憶體確實不夠，而出現 error。這個演算法最大的問題是，當不斷插入而沒有刪除而快要滿的時候會不斷做 garbage collection。浪費過多時間。

Node class：這個 class 是 TreeNode（就是 Index Node，抱歉，老師的 project 說明還沒有的時候就已經用了這個名字，所以後來就很麻煩 refactor 了。）和 LeafNode 的 base class，裡面主要特點是有 keys，也就是剛剛的 KeyType 所構成的 linkedlist，這是所有 Node（TreeNode 和 IndexNode）都有的資料。而且，利用 TreeNode 和 LeafNode 都是 Node derived class 這一特性，可以將每一個 node children 都轉型成 Node，這樣就解決了 TreeNode（Index Node）下面的 children 可能是 TreeNode 也可能是 LeafNode。裡面還有一些實用的 function，這裡就不一一敘述。(下同，只介紹最重要部分)

LeafNode class: 這個類別的 Node class 的衍生類別。裡面主要有 LinkedList<Record>，這其實是一開始的誤解，原本以為不用 slotted page 來存放 record，而直接放在 LeafNode 上。後來知道要有 slotted page 於是就採用從 Record 中拿出 rid 再在 Page 中找 Record 的做法，雖然比較笨（Record 中就已經有 content），但這才老師要求的對的做法。在 LeafNode 中實現了 add function，當出現 size>=2d 時會 return 一個新的 splite LeafNode。這個 LeafNode 也是一個訊號，告訴 parent Index Node 需要新增 key，而不斷做到 recursively add。我基本將 add（Insert）的做法包在了 LeafNode 和 TreeNode 之中。而同時也做了在 delete 時的 rebuid（向同一個 parent 的 neighbour 借 key）。但是，merge 做的比較髒放在 BplusTree 之中，會在 BplusTree 中實作。

TreeNode class: 也就是 Index Node，這個類別 Node class 的衍生類別，裡面有 LinkedList<Node> children，利用 LeafNode 和 TreeNode 都是 Node 的衍生類別，而完美解決不同 children 問題，盡管這個在 B+ tree 真正實作上不會遇到，因為只是 pageId 而已，但是既然用的是 java 就無法用 pageId 來作為 reference。與 LeafNode 同樣的，也做了 add function 和 rebuild，但是由於 Index

Node 是將 key 往上推擠，所以不能和 LeafNode 用同樣的代碼，也就無法一起寫在 Node 裡。

BplusTree class：這是最主要的 class，所有的 B+ tree 運作都要通過這個 class。之前在 TreeNode 和 LeafNode 中包含 split 和 rebuild。但我卻沒有設計好，而將 merge 放在 BplusTree 之中（應該設計成 static function 放在 LeafNode 和 Tree 中會比較好）。

Main class：最一開始的 project 的入口，基本是讀檔寫檔與 parser 的功能。還有 Handling invalid input。

特別說明：因為 B+ tree 只是一個基本架構，裡面的實作細節還有很多的不同，這樣會導致 Tree 的結構有差異，在此詳細說明這次的項目採用的做法。

Split：和一般的 B+ tree 有一樣的做法，基本沒有變動。將右邊 leaf node 第一個 key 一份，往上傳遞，先 split 再加新的 record。將 index node 中間 key，往上推擠，由於是推擠，先加 key 再決定中間 key 是哪個。

Rebuild：同樣秉持著，leaf node 是複製，而 index node 是推擠的概念。會先問左邊是否可以 rebuild，再問右邊是否可以 rebuild（當然，前提是左右有鄰居）。如果不可以才會進行 merge。

Merge：leaf node 為刪除 key，index node 為把 key 拉下來再 merge。同樣地，先問左邊，再問右邊。
開始的狀態

Scan & c：這兩個指令分別採用了不同的算法，雖然 c 可以用 Scan 來做，但是當 Leaf Node 很多時，如果在 c 的時候不看 Leaf Node 執行時間會有很大差異。會相差 $O(\{d^k \mid k: tree\ height\})$

Initial State & Final State：之所以特別提是因為在課本上並沒有提到怎麼把一棵樹從一開始建立起來，我才用的做法是：一開始是一個 LeafNode，當長出 Index Node 時，將 root 往上移。而在 Final State 則是反過來，不斷減少，直到得到 LeafNode。

4. 正確性測試

使用 test_data.txt 來作為正確性測試的資料，採用 d=2 可以直接偵錯誤的值來測試，並輸出 test_data_result.txt。因為有將 BplusTree Overwrite toString()方法，基本上就是用 bfs 將 Node 屬性和 key 全部印出來，可能比較難看出每一層的關係，但是由於每個 TreeNode 決定了下一層的數量，所以在少數的 debug 中還是可以的。如果助教有空閒也可以直接在 Main 中合適的地方打 System.out.println(BplusTree)驗證樹的結構。在這邊也為了助教方便自己在特殊地方加入幾個已經被 comment 的指令，助教只要 uncomment 就可以看到 Tree 的結構。分別是：
Main.java line number: 81, 82, 95, 96 看每次 insert 後的結果。
Main.java line number: 125, 127 看每次 delete 後的結果

以下是正確性驗證的說明：
開始的 root 為 LeafNode，不斷新增值。

```
Insert: 0
LeafNode: 0,


Insert: 416
LeafNode: 0,416,


Insert: 40
LeafNode: 0,40,416,


Insert: 427
LeafNode: 0,40,416,427,
```

第一次 LeafNode split，產生 TreeNode，將 root 指向新的 TreeNode。並留下的是 0，40 與 416，427 再將 416 作為 key 複製，並且將 4 插入 0，40.
驗證初始建立條件可以成功，並且 LeafNode 可以正確 split。

```
Insert: 4
TreeNode: 416,
LeafNode: 0,4,40,
LeafNode: 416,427,
```

為了節省篇幅，省略一些步驟，不斷新增資料，直到 TreeNode split 前。

```
Insert: 335
TreeNode: 40,416,
LeafNode: 0,4,
LeafNode: 40,301,335,
LeafNode: 416,426,427,451,


Insert: 51
TreeNode: 40,416,
LeafNode: 0,4,
LeafNode: 40,51,301,335,
LeafNode: 416,426,427,451,


Insert: 214
TreeNode: 40,301,416,
LeafNode: 0,4,
LeafNode: 40,51,214,
LeafNode: 301,335,
LeafNode: 416,426,427,451,


Insert: 193
TreeNode: 40,301,416,
LeafNode: 0,4,
LeafNode: 40,51,193,214,
LeafNode: 301,335,
LeafNode: 416,426,427,451,


Insert: 477
TreeNode: 40,301,416,427,
LeafNode: 0,4,
LeafNode: 40,51,193,214,
LeafNode: 301,335,
LeafNode: 416,426,
LeafNode: 427,451,477,
```

TreeNode split，產生新的 root，並分別留下 size=2 的兩個 TreeNode。

驗證 TreeNode 可以正確 split

```
Insert: 158
TreeNode: 301,
TreeNode: 40,193,
TreeNode: 416,427,
LeafNode: 0,4,
LeafNode: 40,51,158,
LeafNode: 193,214,
LeafNode: 301,335,
LeafNode: 416,426,
LeafNode: 427,451,477,
```

經過一系列新增資料之後，得到如下 B+ tree，利用這個結果我們來驗證 Scan, q, p, c 等指令操作。這些截圖實在產生的 test_data_result.txt 中，助教可以自己跑跑看。

```
Insert: 159
TreeNode: 301,
TreeNode: 40,158,193,
TreeNode: 416,427,
LeafNode: 0,4,
LeafNode: 40,51,58,
LeafNode: 158,159,179,
LeafNode: 193,214,277,
LeafNode: 301,334,335,
LeafNode: 416,426,
LeafNode: 427,435,451,477,
```

驗證 Scan

```
52  Scan Chat
53  7,3
```

驗證 q

```
32  q Chat 179
33  179, 27, Rid:0,14
```

可以數處 179 的 length 為 27

```
24  I, Chat, 179, "vwfeoqdwacszjfbuzxknqaxvbdc"; 334, "zcwnasnha
    iykoosjpjwdulzpyjfrwhmgt hvkuohcrhe ernijbsffqxstcasri"; 277
    speykiazckcsxvzii vawdnskflcvyyd"; 159, "pct oseigzqmmxmtabw
25  0,14
26  1,0
27  1,1
28  1,2
29  1,3
30  1,4
```

由於 range query 用 rid 並不好驗證，於是我將輸出 key 都 print 在 console 上，此代碼在 Main.java line 204，可以 uncomment 執行，即可看到 range query key，由於這個不是老師規定輸出格式，所以只用 println 來產生。

```
35  q Chat 179 400
36  0,14
37  0,11
38  0,10
39  1,2
40  0,7
41  1,0
42  0,8
```

```
179,193,214,277,301,334,335,
```

驗證 p

```
49  p Chat 0
50  Hello world!,pccrltd jvwwi gcllukzi,uz kratlywwjolphof
    pdwxdpycqugteuh,brbdnotwiiixtnidvch
    dbtdqorjbgm,hnhsvpzfimhwvcho edflprfccsbaq,xifdugoibzocdggix
    tikszfesvewgkwfiydeamcgw,lnhvjdtmdtj,bsfkl
    hmewpkverzeohnvcp lgkbkitmjirylamqt lzonhjwjcppuyhk,sywao wq
    kyusmxpdhvnjvxxsxomq,gyrdxbreasymuvoefmjarizhzybsbjiwdvyzx
    xplmiwsum,bqohsvxrmhvkdfjuhbcinif vlomqciqxmnytashit,sd rmj
    hucfqlhvjeemnacyyzwrjy,xfykfcyglfdezqeymb,cgfkrehytcomwqs tl
    bq,vwfeoqdwacszjfbuzxknqaxvbdc,**********
    ,27,422,20,402,18,384,29,355,42,313,47,266,28,238,57,181,11,
    170,41,129,30,99,31,68,34,34,22,12,12,0,15,449
```

驗證 c

```
55  c Chat
56  3,2
```

最後是 delete，麻煩助教結合上下圖看差別。

```
delete key: 51
TreeNode: 301,
TreeNode: 40,158,193,
TreeNode: 416,427,
LeafNode: 0,4,
LeafNode: 40,58,
LeafNode: 158,159,179,
LeafNode: 193,214,277,
LeafNode: 301,334,335,
LeafNode: 416,426,
LeafNode: 427,435,451,477,
```

R04921094 EECS 葉孟元
R05921099 EECS 振　昊

驗證 rebuild

delete 58 向右邊鄰居借一個 key=158 做 rebuild，並更改 parent key 為 159

```
58
delete key: 58
TreeNode: 301,
TreeNode: 40,159,193,
TreeNode: 416,427,
LeafNode: 0,4,
LeafNode: 40,158,
LeafNode: 159,179,
LeafNode: 193,214,277,
LeafNode: 301,334,335,
LeafNode: 416,426,
LeafNode: 427,435,451,477,
```

驗證 merge

和右鄰居 merge 行程 0，40，158 並刪除 parent key 40

```
delete key: 4
TreeNode: 301,
TreeNode: 159,193,
TreeNode: 416,427,
LeafNode: 0,40,158,
LeafNode: 159,179,
LeafNode: 193,214,277,
LeafNode: 301,334,335,
LeafNode: 416,426,
LeafNode: 427,435,451,477,
```

最特殊的降低樹高。

由於刪除了很多，中間省略一些步驟，將 334 刪除，導致 LeafNode merge 成 301，335，426。並刪除 parent key 335，由於 335 的刪除又導致新的 merge，最後得到如下圖。

```
Delete:416
TreeNode: 301,
TreeNode: 159,193,
TreeNode: 335,427,
LeafNode: 0,158,
LeafNode: 159,179,
LeafNode: 193,214,277,
LeafNode: 301,334,
LeafNode: 335,426,
LeafNode: 427,435,451,477,


Delete:334
TreeNode: 159,193,301,427,
LeafNode: 0,158,
LeafNode: 159,179,
LeafNode: 193,214,277,
LeafNode: 301,335,426,
LeafNode: 427,435,451,477,
```

5. 重複 Key

如果要在本 project 上再新增處理重複 Key 的問題，為了方便改寫會採用 overflow page，而不是採用更改 insert 和 query 而將每一個 query 都視為 rangequery 的做法。原因是：由於我們是類似模擬的做法，在每一個 class 上可以新增不同的結構。而如果有重複的值，我只要在 LeafNode 中的 Record 下增加一個 LinkedList 就可以將所有的 rid 都記錄下來，並不會花費太多時間改寫，所以這就是我的考量。盡管會犧牲一些效率，但是會比較方便重構。

6. 心得感想

R04921094 葉孟元：這次的項目花了很多時間和精力，雖然很辛苦也熬夜了兩天，但是真的很值得，完整學到了 B+ tree 的做法，和之前只是模糊的知道很不一樣。熬夜的原因並不是沒有提前規劃，而且根本沒想到這個 project 會需要這麼多時間。只是想當然爾的覺得 B+ tree 而已，和 Red-Black tree 很像，會很容易。但是開始寫的時候才發現越來越多的問題，Leaf Node merge 和 Index Node merge 的差別導致需要每個都分開實作，由於 B+ tree 只是一個架構，並沒有統一的教科書式的演算法，我需要處理很多很多問題。非常感謝助教的協助，一直發問題給助教，沒有助教的幫忙我也不會順利完成。最後如果有什麼問題麻煩發信問我: r04921094.ntu.edu.tw