**1.**

編譯： $ g++ -g -Wall -std=c++11 main.cpp -o demo

執行： $ ./demo

**2.**

由於磁碟I/O速度比內存記憶體I/O速度慢許多，B tree和B+ tree的使用能盡可能減少磁碟讀取次數，假設磁碟讀取一次能掌握一單位page的記憶體大小，這裡B tree和B+ tree讓每個node剛好是一單位page的記憶體大小，下圖示範了在degree = 5，17個不同key值依序插入到B tree和B+ tree的結果；在B tree中每個node均存有key、child pointer和record pointer，訪問child pointer可以得到child node在磁碟中位置，record pointer則紀錄了某筆資料在磁碟中的位置，且在B tree中插入的每個key均不會重複；而在B+ tree中只有leaf node可以透過record pointer得到某筆資料在磁碟中的位置，因此所有leaf node涵蓋了插入的每個key，且leaf node還存有next pointer以用來訪問下一個相鄰的leaf node，在B+ tree的non-leaf node中只存有key和

一張含有 文字, 螢幕擷取畫面, 櫃子 的圖片

自動產生的描述

child pointer，且key會與在leaf node中的key重複。

在進行range query時，使用B+ tree比起B tree在磁碟讀取上擁有更少的次數，用圖中的例子，假設內存記憶體一次只能處理一個page，若要query key range = (15, 45)，使用B tree時page被讀取處理順序為#1 → #2 → #4 → #1 → #2 → #5 → #1 → #2 → #6，一共9次磁碟讀取，而使用B+ tree為#1 → #2 → #4 → #5 → #6 → #7，一共6次磁碟讀取；從圖中可以看出B+ tree得利於將所有key都存在leaf node，且leaf node的next pointer可以使訪問連續key在不同node時更加容易。

**3.**

本作業是使用C++實作，下圖為node的source code，node中所有變數用struct Node打包；isLeaf用來判斷此node是否為leaf node；keys指向存放key的array；nodeSize記錄目前node含有多少key；parent指向non-leaf parent node，在進行node split產生non-leaf時操作更方便；next只在leaf node使用，指向下一個leaf node；children指向Node pointer array，用於non-leaf node，array中的Node pointer則指向child node；下圖中右側為Node的constructor，傳入參數為degree，在Node生成時會指定最多只能放degree – 1個key和degree個child node；初始狀態還沒放入key，因此nodeSize = 0，在放入key後，如果是non-leaf node，key的數量為nodeSize，child node pointer的數量為nodeSize + 1；初始狀態會讓parent, next和所有child node pointer均指向null pointer。

一張含有 文字 的圖片

自動產生的描述

**4.**

本實作整個B+ tree用class bPlusTree打包，其中以root指向整棵樹的根Node；void bPlusTree::insert(int)用於insert功能，而void bPlusTree::insertInternal(Node\*, Node\*, int)用於處理有Node進行split並向上傳key時。

void bPlusTree::insert(int)執行流程如下：

1. 判斷tree是否是空的，若為空，使用動態記憶體配置產生一個struct Node，用root指向它，並設定為leaf node，將key存入，結束void bPlusTree::insert(int)。

2. 若tree不是空的，由root往下尋找key應該insert的leaf node。

3. 找到key應該insert的leaf node後，這裡稱targetLeafNode，檢查targetLeafNode是否還有空位能insert key，若還有空位，將key插入並保持Node中所有key是排序的。

4. targetLeafNode若沒有空位，需進行leaf node split，首先會將targetLeafNode中所有key還有要插入的key，有排序的放在一個暫時array，這裡稱temp，可以得知temp中key總數量為degree；接著，動態記憶體配置一個新的struct Node，這裡稱newLeafNode；最後將temp裡前degree / 2個key分配至targetLeafNode，後(degree – degree / 2)個key分配至newLeafNode；圖像上可以想像是targetLeafNode向右分裂出了newLeafNode，因此要讓newLeafNode的next指向原本targetLeafNode的next指向的leafNode．再讓targetLeafNode的next指向newLeafNode。

5. 若targetLeafNode為root，split會向上產生一個non-leaf node，此non-leaf node設定為root，並將newLeafNode的第一個key複製存入；最後設定newLeafNode的children依序為targetLeafNode、newLeafNode，targetLeafNode和newLeafNode的parent為root，結束void bPlusTree::insert(int)。

6. 若targetLeafNode不是root，要讓targetLeafNode和newLeafNode連接到同一個parent node，並將newLeafNode的第一個key插入parent node，這裡會呼叫void bPlusTree::insertInternal(Node\*, Node\*, int)。

void bPlusTree::insertInternal(Node\*, Node\*, int)的第一個傳入參數為parent node pointer，第二個傳入參數為child node pointer，第三個傳入參數為insert到parent node的key，這裡暫稱parent node為oldNonLeafNode，執行流程如下：

1. 若oldNonLeafNode尚有空位，將key與child node pointer插入對應位置，使插入完成後key是排序的，也要讓所有child node pointer排序正確，之後就可結束void bPlusTree::insertInternal(Node\*, Node\*, int)。

2. 若oldNonLeafNode已滿，需要進行split，首先會開兩個暫時array來放置key和node pointer，這裡稱tempKeys和tempChildren，array size分別為degree和degree + 1；oldNonLeafNode中所有key和要插入的key會放入tempKeys保持排序，oldNonLeafNode中所有child node pointer和傳入的child node pointer也會以對應排序放入tempChildren；接著，動態記憶體配置一個新的struct Node，這裡稱newNonLeafNode； 再來，把tempKeys中前degree/2個key放入oldNonLeafNode，tempKeys中後degree/2個key放入newNonLeafNode、tempChildren中前(degree + 1)/2個node pointer放入oldNonLeafNode，tempChildren中後(degree + 1)/2個node pointer放入newNonLeafNode；這裡要注意有些child node對應的parent會變成newNonLeafNode，因此所有child node pointer在重新分配時都會更新一次對應的parent。

3. 上個步驟很類似void bPlusTree::insert(int)中的步驟4，但tempKeys最中間的key並沒有被分配至oldNonLeafNode或newNonLeafNode，而是要向上插入non-leaf node；首先判斷上面是否有non-leaf node，即判斷oldNonLeafNode是否為root，若為root，進行與void bPlusTree::insert(int)中的步驟5一樣動作；若不是root，則進行與void bPlusTree::insert(int)中的步驟6一樣動作。

**5.**

範例1: 範例2:

一張含有 文字 的圖片

自動產生的描述 一張含有 文字 的圖片

自動產生的描述