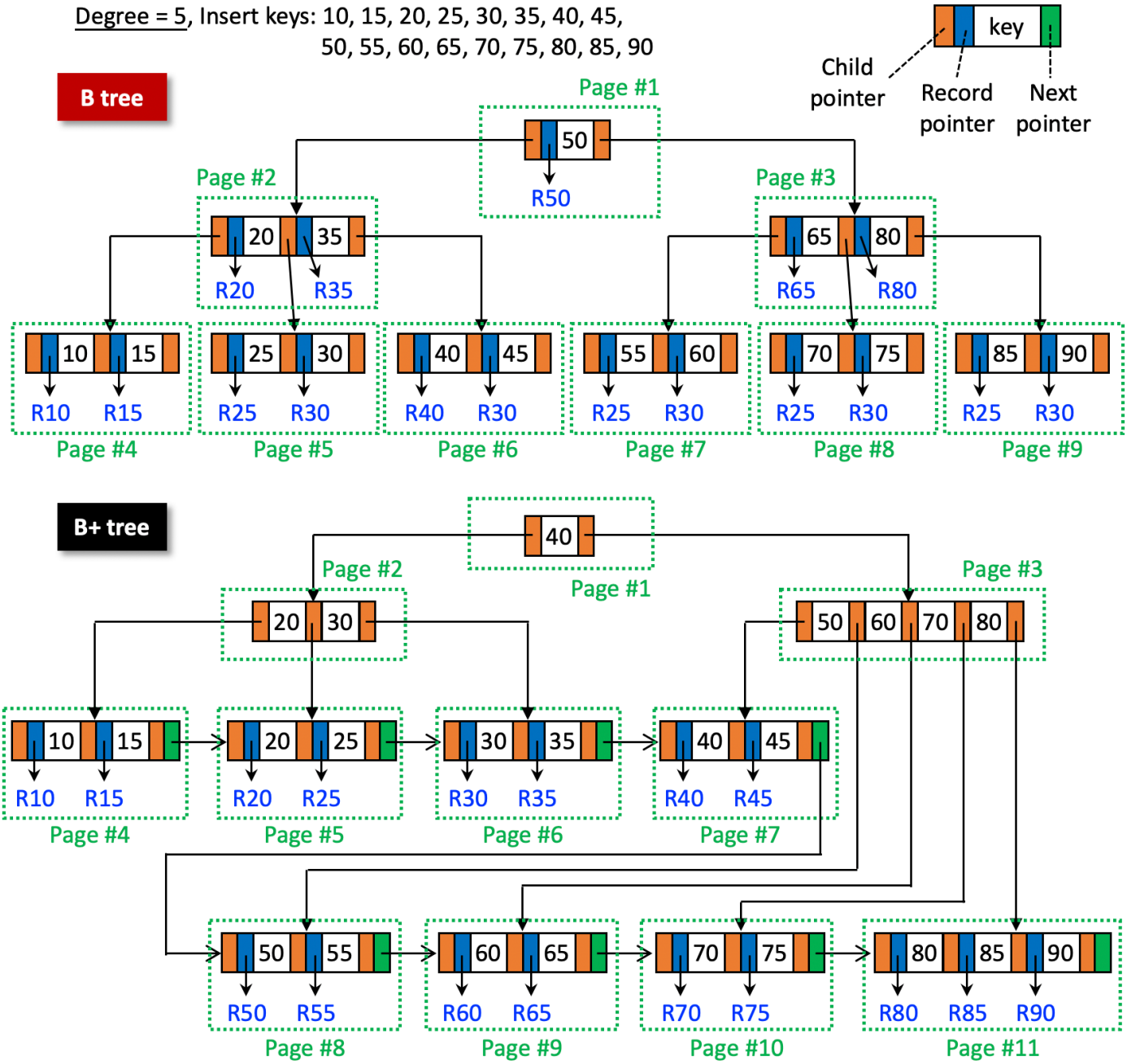
1.

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，假設磁碟讀取一次只能掌握一單位 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



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 struct 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， key 的數量為 nodeSize，如果是 non-leaf node，會有 child node pointer 且數量為 nodeSize + 1；初始狀態會讓 parent, next 和所有 child node pointer 均指向 null pointer。

```
struct Node {
   bool isLeaf;
   int *keys, nodeSize;
   Node *parent, *next;
   Node **children;

   Node(int);
};
```

```
Node::Node(int degree) {
   keys = new int[degree - 1];
   parent = next = nullptr;
   children = new Node*[degree];
   for (int i = 0; i < degree; i++)
      children[i] = nullptr;
   nodeSize = 0;
}
```

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，剩下所有 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。

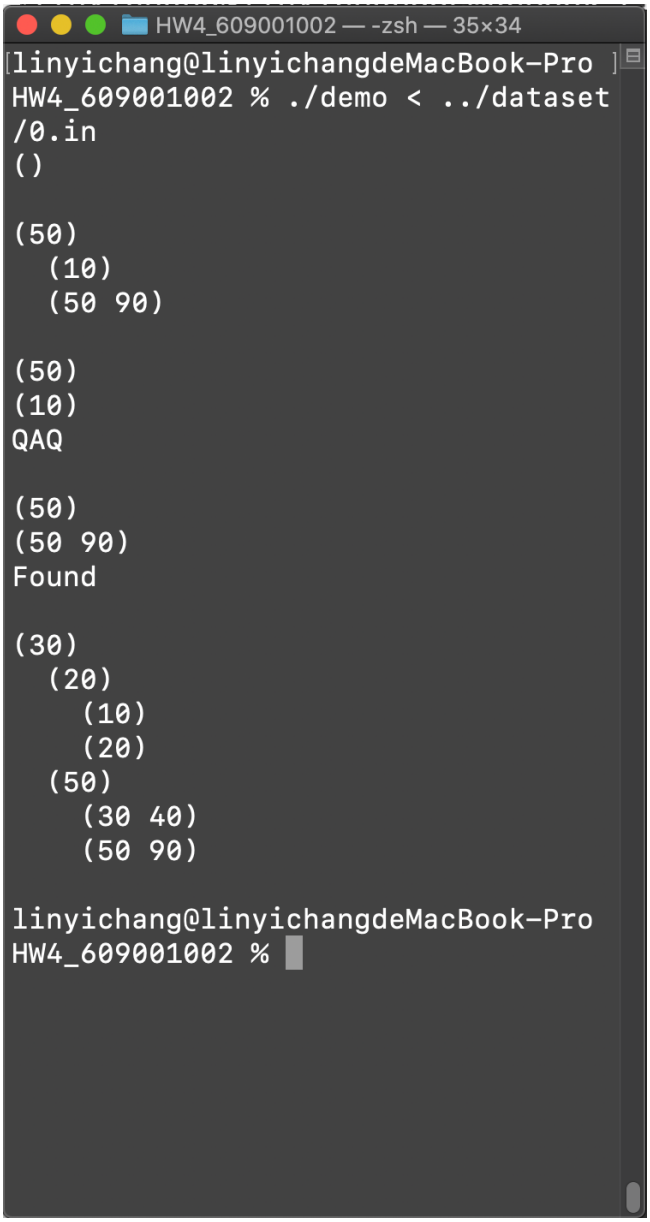
5. 若 targetLeafNode 為 root，split 會向上產生一個 non-leaf node，讓此新 non-leaf node 成為 root，並將 newLeafNode 的第一個 key 複製存入；最後設定 root 的 children 依序為 targetLeafNode、newLeafNode，targetLeafNode 和 newLeafNode 的 parent 為 root，結束 void bPlusTree::insert(int)。
6. 若 targetLeafNode 不是 root，要讓 targetLeafNode 和 newLeafNode 連接到同一個 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，首先會用兩個暫時 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:


HW4_609001002 — -zsh — 35×34
[linyichang@linyichangdeMacBook-Pro]
HW4_609001002 % ./demo < ../dataset
/0.in
()

(50)
  (10)
  (50 90)

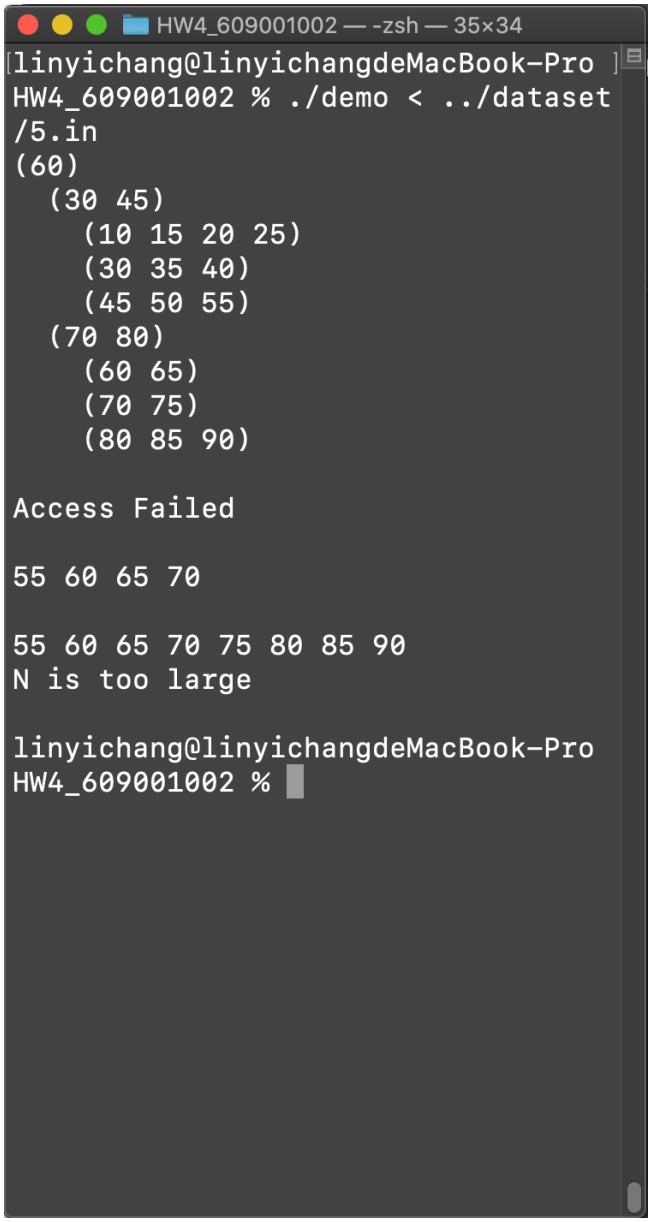
(50)
(10)
QAQ

(50)
(50 90)
Found

(30)
  (20)
    (10)
    (20)
  (50)
    (30 40)
    (50 90)

linyichang@linyichangdeMacBook-Pro
HW4_609001002 %

範例 2:


HW4_609001002 — -zsh — 35×34
[linyichang@linyichangdeMacBook-Pro]
HW4_609001002 % ./demo < ../dataset
/5.in
(60)
  (30 45)
    (10 15 20 25)
    (30 35 40)
    (45 50 55)
  (70 80)
    (60 65)
    (70 75)
    (80 85 90)

Access Failed

55 60 65 70

55 60 65 70 75 80 85 90
N is too large

linyichang@linyichangdeMacBook-Pro
HW4_609001002 %