1.
$ g++ HW4_0616075.cpp
$ ./a.out

2.
B Tree 和 B+ Tree 的差異在於:
B Tree 是將資料保存在樹的每一個節點，若是節點存滿了，則往下一層分列，新增資料將往下移。B+ Tree 則是在最底層的 leaf node 儲存資料，存滿了就在最底層分裂，並往上建立索引。B+ Tree 索引是建立在非 leaf node 上，所以搜尋資料時能透過索引，到達 leaf node 去做查詢。最底層的 leaf node 之間會互相連接，所以需要查詢所有資料時，不需要走遍樹中每一個節點。

B+ Tree 的優勢在於當要讀取的資料量大時，因為不可能將所有資料儲存在 cache memory 中，所以需要從磁碟讀取資料。由於建立索引的特性，因此可以減少磁碟的讀取次數，增加讀取資料的效率。

3.
我是以 class 作為 B+Tree 的 node 結構。
int 類別:
sz(儲存多少 key)、szp(儲存多少 node)、level(該 node 在樹中是第幾層)、parPtrID(在 key 達到最大容量，需要分裂時，用來尋找對到哪個 parent node)、key[](int array 用來儲存該 node 中有那些 key value)。
bool 類別:
leaf(用來判別該 node 是否為 leaf)
node 類別:
ptr[](用來連接下一層的 node)、par(連接該 node 對應的 parent node)、sibling(只有在該 node 是 leaf 時，用來連接該 node 與下一個 node，方便之後的 sequential access 能快速得到不同 node 所儲存的 key value)。

value 的存放是用上述的 key array 來儲存、pointer 則是用上述的 ptr array 來儲存。

4.
在 insert 時，會先判斷該 value 要放在哪裡，所以一開始會先從 root node 中儲存的 key 開始比較，若 root node 為空，直接將 value 加入 root node 中。若是 value 小於 root node 中儲存的 key，下一步則會判斷該 node 是不是 leaf，若

是，則直接將該 value 放入剛剛選好的位置。若不是 leaf，則前往該 key 所對應到下一層的 node 去做比較，並用此方式一直比較，直到移動到 leaf 後，再將 value 放入對應位置。

在 insert 完後，會去判斷該 node 中儲存 key array 的 size 有沒有等於程式一開始定義的最多 value 儲存數量，若有，則執行分裂。分裂一開始會去判斷需要分裂的 node 是不是 leaf，若是 leaf，則會重新調整 leaf node 之間的 sibling 連結關係。接著將原本 node 中 key array 裡順序大於 maxn/2 的 value 移到新分裂出 splitnode 中的 key array 裡去做儲存。同時會重新調整原本 node 和 splitnode 裡面儲存 value 數量的 sz 值。接著則是將原本 node 中被移動到 splitnode 那些 value 所對應的下一層 node 位置移動給 splitnode，重新調整 node 和 splitnode 之間的 ptr array 大小 szp 的值。此時會判斷原本 node 有沒有 parent node，若無，創造一個新的 parent node 給 node 和 splitnode 去做連結。並且新增 parent 中會對應到這兩個 node 所需的 key value 和 ptr 位置。

5.

```
[kutk075@linux1 ~/db]$ ./a.out < 0.in
()

(50)
  (10)
  (50 90)

(50)
(10)
QAQ

(50)
(50 90)
Found

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

```
[kutk075@linux1 ~/db]$ ./a.out < 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
```