Database HW4 report

請寫出在 linux 環境中編譯及執行你的程式的指令

- 指令:
  g++ hw4.cpp -o hw4_out
  ./hw4_out

請比較 B Tree 及 B+Tree 的不同。
以及這些不同可以讓 B+Tree 獲得什麼好處。

Difference

- B Tree把資料存在每個子節點，一旦超過node容量的就會split分為兩個，往下分裂。
- B+ Tree改良B Tree的結構，將資料存在leaf node，其他節點只存key，且底層的leaf node都會用link list連結起來。
  超過容量時會在最底層split，然後把key值複製一份回傳上去；如果儲存key的node滿了，就照B Tree的規則往上split。

Advantages

- B Tree每次搜尋的速度不穩定，且範圍查詢非常沒效率。
- B+ Tree比B Tree矮胖，每次查詢一定會到leaf node，搜尋速度平均；到了leaf node只要照順序走一遍就找得到了，也較方便做range query。

請詳述本次作業中 B+Tree 的結構，即 B+Tree 中的 node 是用什麼資料結構，
還有是如何存放 value 及 pointer。

- 以class BPtree為主幹，儲存一個root，insert時再從root開始接，用pointer連接起來。
- BP tree裡面放root Node還有其他功能函式，Node裡存value,key還有node其他資訊。

```
class Node {
public:
    bool is_leaf;       //是否為leaf node
    int* key, size;     //存放索引，node size
    Node** ptr;         //pointer用來指向連結的node
    int m;              //存input m
    friend class BPtree;

    Node(int max);
};

class BPtree {
public:
```

```
    Node* root;
    void insert_internal(int, int, Node*, Node*);
    Node* find_parent(Node*, Node*);
    int m;
    BPtree(int max);

    void search(int);
    void insert(int);
    void display(Node*, int);
    void seq_access(int, int);
    Node* getroot();
};
```

請詳述本次作業中是如何實作 B+Tree 的 Insert 功能。如果 Insert 後的 node 需要進行分裂的話，是如何實作的。

- 分成三個函式:insert，insert_internal，find_parent
- 首先call insert function

```
void BPtree::insert(int x) {
    //如果tree是空的，先設成root
    if (root == NULL) {
        root = new Node(m);
        root->key[0] = x;
        root->is_leaf = true;
        root->size = 1;

    }
    else {
        Node* cursor = root;
        Node* parent = cursor;      //設一個pointer cursor用來追蹤目前的位置
        //while迴圈用來找value應該insert的位置在哪個區段
        while (cursor->is_leaf == false) {
            parent = cursor;
            for (int i = 0; i < cursor->size; i++) {
                if (x < cursor->key[i]) {
                    cursor = cursor->ptr[i];
                    break;
                }
                if (i == cursor->size - 1) {
                    cursor = cursor->ptr[i + 1];
                    break;
                }
            }
        }


        if (cursor->size < m) {     //node is not full
            int i = 0;
            //找可以insert new key的地方
            while (x > cursor->key[i] && i < cursor->size) {
                i++;
            }
            //騰出空間給new key，全部右移一格
            for (int j = cursor->size; j > i; j--) {
                cursor->key[j] = cursor->key[j - 1];
            }
```

```
        //insert key:更新value, size, link list
        cursor->key[i] = x;
        cursor->size++;
        cursor->ptr[cursor->size] = cursor->ptr[cursor->size - 1];
        cursor->ptr[cursor->size - 1] = NULL;
    }
    else {      //node is full,need split
        //設一個virtual Node暫存data, newleaf準備存新的data
        Node* newleaf = new Node(m);
        vector<int>virtualNode(m + 1);
        for (int i = 0; i < m; i++) {
            virtualNode[i] = cursor->key[i];        //先copy一份
        }

        int i = 0, j;
        //找可以insert new key的地方
        for (i = 0; i < cursor->size; i++) {
            if (x < virtualNode[i])
                break;
        }
        //騰出空間給new key
        for (int j = m; j > i; j--) {
            virtualNode[j] = virtualNode[j - 1];
        }
        virtualNode[i] = x;

        //cursor為分開後左邊的node, newleaf為右邊
        newleaf->is_leaf = true;
        cursor->size = (m + 1) / 2;
        newleaf->size = (m + 1) - cursor->size;

        cursor->ptr[cursor->size] = newleaf;    //連接新node ()->()
        newleaf->ptr[newleaf->size] = cursor->ptr[m];   //新的右邊的node接到舊的link list
        cursor->ptr[m] = NULL;


        for (i = 0; i < cursor->size; i++) {
            cursor->key[i] = virtualNode[i];
        }
        //全部copy給new node
        int q = cursor->size;
        for (int k = 0; k < newleaf->size; k++) {
            newleaf->key[k] = virtualNode[q];
            q++;
        }

        //modify the parent
        if (cursor == root) {   //如果本來就是root的話
            Node* newroot = new Node(m);
            newroot->key[0] = newleaf->key[0];
            newroot->ptr[0] = cursor;
            newroot->ptr[1] = newleaf;
            newroot->is_leaf = false;
            newroot->size = 1;
            root = newroot;
        }
        else {
            //修改nonleaf node
            insert_internal(newleaf->key[0], m, parent, newleaf);
        }

    }
```

```
    }
}
```

- insert_internal，改索引值所在的node

```
void BPtree::insert_internal(int x, int m, Node* cursor, Node* child) {
    //insert new key
    if (cursor->size < m) {     //node is not full
        int i = 0;
        for (i = 0; i < cursor->size; i++)
        {
            if (x < cursor->key[i])
            {
                break;
            }
        }
        //key pointer移出空間
        for (int j = cursor->size; j > i; j--) {
            cursor->key[j] = cursor->key[j - 1];
            cursor->ptr[j + 1] = cursor->ptr[j];
        }
        //更新資料，接新的node
        cursor->key[i] = x;
        cursor->size++;
        cursor->ptr[i + 1] = child;
    }
    else {      //node is full
        //設newinternal為split後的node，virtual key和virtual pointer暫存
        Node* newinternal = new Node(m);
        vector<int>virtualkey(m + 1);
        vector<Node*>virtualPtr(m + 2);
        for (int i = 0; i < m; i++) {
            virtualkey[i] = cursor->key[i];
        }
        for (int i = 0; i < m + 1; i++) {
            virtualPtr[i] = cursor->ptr[i];
        }
        int i = 0, j;
        for (i = 0; i < cursor->size; i++)      //find correct position
        {
            if (x < virtualkey[i])
            {
                break;
            }
        }
        //移開value pointer
        for (int j = cursor->size; j > i; j--) {
            virtualkey[j] = virtualkey[j - 1];
            virtualPtr[j + 1] = virtualPtr[j];
        }
        virtualkey[i] = x;
        virtualPtr[i + 1] = child;  //接上new child
        newinternal->is_leaf = false;
        //split
        cursor->size = (m + 1) / 2;
        newinternal->size = m- cursor->size;
        //放入資料到左邊的node
        for (int i = 0; i < cursor->size; i++)
        {
            cursor->key[i] = virtualkey[i];
```

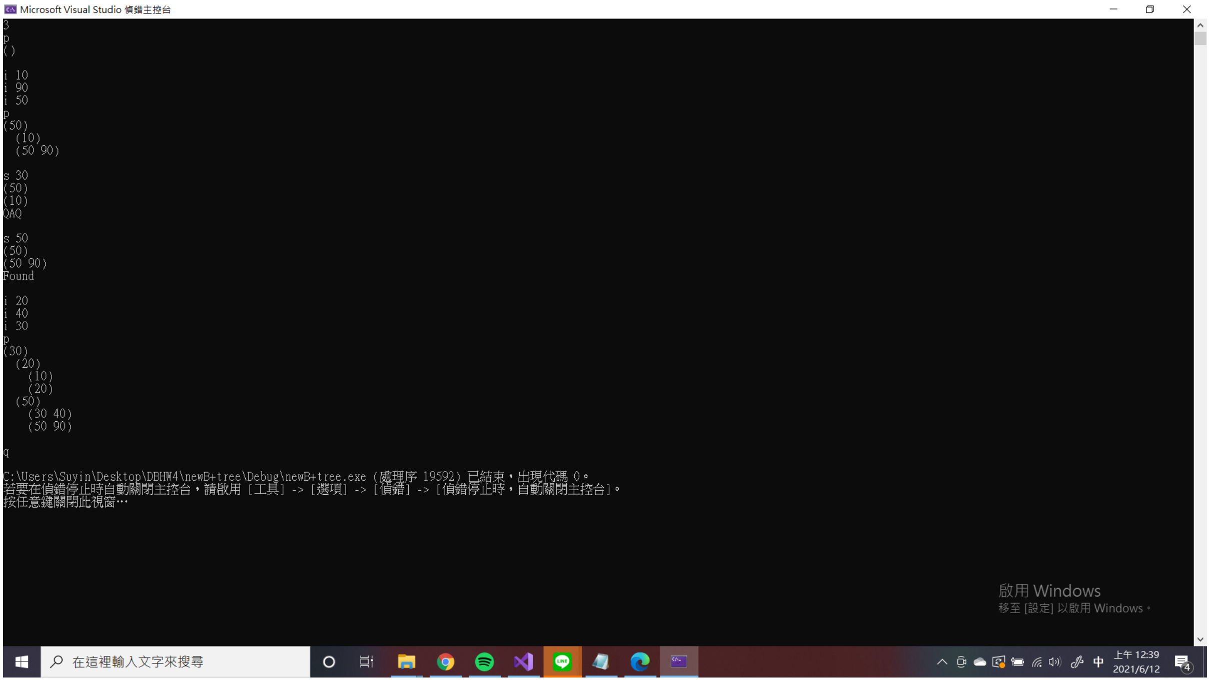
```
            cursor->ptr[i] = virtualPtr[i];
        }
        cursor->ptr[cursor->size] = virtualPtr[cursor->size];

        //除了中間要push up 的元素，其他都copy到node裡
        j = cursor->size + 1;
        for (i = 0; i < newinternal->size; i++) {
            newinternal->key[i] = virtualkey[j];
            j++;
        }
        j = cursor->size + 1;
        for (i = 0; i < newinternal->size + 1; i++) {
            newinternal->ptr[i] = virtualPtr[j];
            j++;
        }

        //push middle one up
        if (cursor == root) {
            Node* newroot = new Node(m);
            newroot->key[0] = virtualkey[cursor->size];
            newroot->ptr[0] = cursor;
            newroot->ptr[1] = newinternal;
            newroot->is_leaf = false;
            newroot->size = 1;
            root = newroot;
        }
        else {  //如果還沒到root node就繼續往上丟key
            insert_internal(virtualkey[cursor->size], m, find_parent(root, cursor), newinternal);
        }

    }
}
```

- find_parent，找到該node的parent

```
Node* BPtree::find_parent(Node* cursor, Node* child) {
    Node* parent = NULL;
    //leaf can not be a parent
    if (cursor->is_leaf || (cursor->ptr[0])->is_leaf) {
        return NULL;
    }
    for (int i = 0; i < cursor->size + 1; i++) {
        if (cursor->ptr[i] == child) {
            parent = cursor;
            return parent;
        }
        else {
            parent = find_parent(cursor->ptr[i], child);
            if (parent != NULL)
                return parent;
        }
    }
    return parent;
}
```

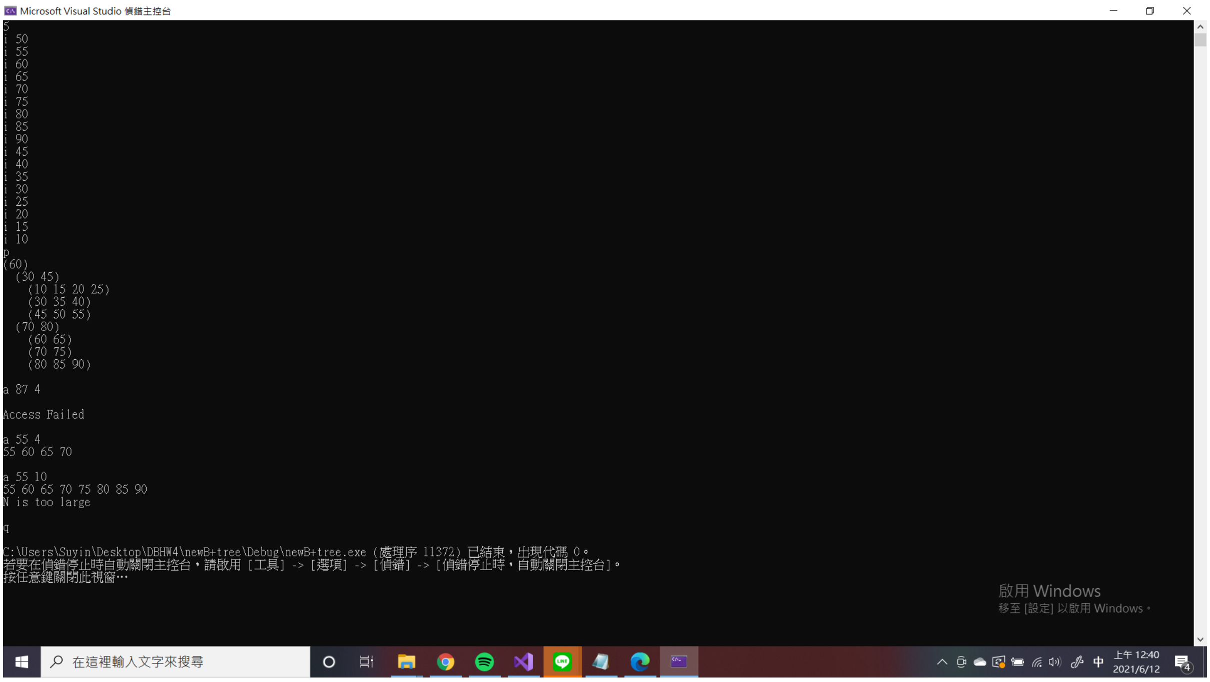
請截圖執行完【範例 1】及【範例 2】後的結果。

範例1



Microsoft Visual Studio 偵錯主控台
3
p
( )
i 10
i 90
i 50
p
(50)
(10)
(50 90)
s 30
(50)
(10)
QAQ
s 50
(50)
(50 90)
Found
i 20
i 40
i 30
p
(30)
(20)
(10)
(20)
(50)
(30 40)
(50 90)
q
C:\Users\Suyin\Desktop\DBHW4\newB+tree\Debug\newB+tree.exe (處理序 19592) 已結束，出現代碼 0。
若要在偵錯停止時自動關閉主控台，請啟用 [工具] -> [選項] -> [偵錯] -> [偵錯停止時，自動關閉主控台]。
按任意鍵關閉此視窗…
啟用 Windows
移至 [設定] 以啟用 Windows。
在這裡輸入文字來搜尋
上午 12:39
2021/6/12

範例2



Microsoft Visual Studio 偵錯主控台
5
i 50
i 55
i 60
i 65
i 70
i 75
i 80
i 85
i 90
i 45
i 40
i 35
i 30
i 25
i 20
i 15
i 10
p
(60)
(30 45)
(10 15 20 25)
(30 35 40)
(45 50 55)
(70 80)
(60 65)
(70 75)
(80 85 90)
a 87 4
Access Failed
a 55 4
55 60 65 70
a 55 10
55 60 65 70 75 80 85 90
N is too large
q
C:\Users\Suyin\Desktop\DBHW4\newB+tree\Debug\newB+tree.exe (處理序 11372) 已結束，出現代碼 0。
若要在偵錯停止時自動關閉主控台，請啟用 [工具] -> [選項] -> [偵錯] -> [偵錯停止時，自動關閉主控台]。
按任意鍵關閉此視窗…
啟用 Windows
移至 [設定] 以啟用 Windows。
在這裡輸入文字來搜尋
上午 12:40
2021/6/12