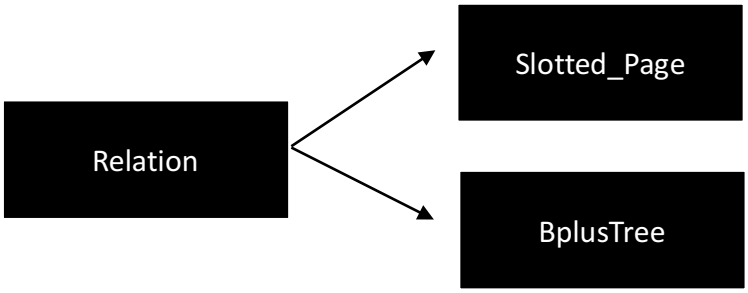
Final Project: B+ tree Index

孫聖授 電信所 R04942031
吳冠融 電信所 R04942070
游肇輝 電信所 R05942046

- 建構 **Relation** 的 Class, 被建構時,同時生成 B-plustree 與 Slotted_Page 兩個 class



Relation 的指令

```
Relation<T>::Relation(const string& relationName, const string& keyType, int
recordLength) : _bpt(), _sp(recordLength)
```

1) InsertRecord **I** 的指令

```
template <class T>
void Relation<T>::insertRecord(T key, string record) {
    int rid = _sp.insertRecord(record);
    _bpt.insertRid(key, rid);
}
```

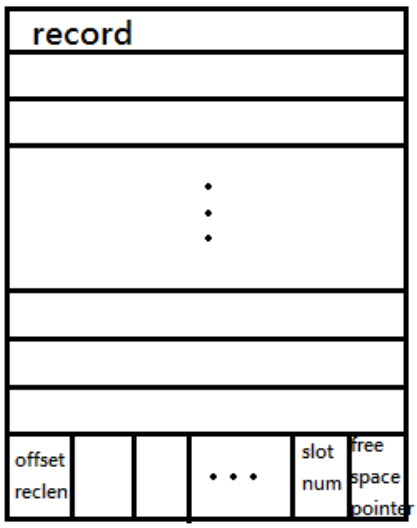
2) deleteRecord **D** 的指令

```
template <class T>
void Relation<T>::deleteRecord(T key) {
    int rid = _bpt.queryRid(key);
    _sp.deleteRecord(rid);
    _bpt.deleteRid(key);
}
```

在 B-plustree 中用 key 來找到此筆 record 並刪除，在 slotted page 中用 rid 來找到此筆 record 並刪除

1. Implementation choices and Slotted Page management :

- SLOTTED PAGE
  - **Slotted Page Format:**
    作業中以 C++物件的形式來 formate slotted page，物件內容包含 pageSize = 512 bytes, free space pointer, slot number, offset, reclen 以及儲存 record 的 slot，而 record 以及對應到的 RID 是以 vector 的形式來實作。
  - **Slot_Number 計算方式**：
    由右圖令 Slot 最大個數為 n
    已知 Free Space Pointer Size = 2 bytes
    Slot Number Size = 2 bytes
    Offset Size = 2 bytes
    Reclen Size = 2 bytes
    Reclen 在建立 Relation 時決定
    因此可列出以下式子：
    (Page Size - Free Space Pointer Size – Slot Number Size) / (Reclen Size + Offset Size + Reclen) ≥ Maximum Slot Num
    ⇨ (512 -2 -2) / (2 +2 + reclen) ≥ n
    又因 int 除法為無條件捨去
    ∴ n = 508 / (4 + reclen)



  - **Insert and Delete Function:**
    實作 Insert 和 Delete 時，以填滿目前最新開的 page 後再回填前面被 Delete 掉的部分為原則來實作。
    Example1:資料已經 Insert 到第 5 頁了，然後接下來對 1, 2, 3, 4 頁中的資料做 Delete 的動作，那之後再繼續 Insert 時，會先將第 5 頁填滿後，再回填前面 1, 2, 3, 4 頁的空缺處。
    Example2: 接續 Example1，如果今天已經回填到 3 頁的空缺處了，此時若再 Delete 掉第 1 頁當中的資料，則先繼續填滿 3, 4, 5 頁，之後再繼續檢查前面是否有需要回填的部分。

2. B+ tree Implementation

- 定義 BPlustree 的 class，其中我們以 **nested class** 的結構來實現，在 BPlustree 中的 class 中包含著另一 class 為 BPlusTreeNode，也就是說 BPlustree 的 private member 為 BPlusTreeNode class，而主要**以 BPlusTreeNode 來操作 tree 當中的 operations.**
- BPlusTreeNode 當中的資料有以下:

```
private:
    class BPlusTreeNode {
        public:
            BPlusTreeNode(bool isLeaf, bool isRoot, int maxKeyCount); //constructor
            void printNode();
            int _maxKeyCount;                                   //每個node最多key的數量
            int _keyCount;                                      //node當中key的數目
            bool _isLeaf;                                       //此node是否為leaf
            bool _isRoot;                                       //此node是否為root
            T* _keys;                                           //每個key的值
            void** _pointers;                                   //node當中的pointer,指向下一個node
            void* _father;                                      //此node的father
            void* _prev;                                        //指向此node的pointer (sibling)
            void* _next;                                        //此node指向下一node的pointer (sibling)
    };
```

每個 node (page)的大小為 512 Bytes，由 **key 的大小(可能為 int: 4 Bytes, string: 10 Bytes)**，可以去計算出_maxKeyCount 的大小,也就是每個 node 當中最多可以有幾個 key。

Key 的大小若為 x, _maxKeyCount 為 n,

n*x+ (n+1)*4+4(_prev)+4(_next) <= 512

(n+1): pointer 會比 key 的數量再多一個

n=(500/(x+4)), x=4 o 10 => n= 62 or 35

- BPlusTreeNode 中的 insert

```
template <class T>
bool BPT<T>::insertValue(T key, int value) {
    // find the leaf page that the key may be in.
    BPT_NODE* leaf = findLeaf(key);

    // if key in leaf->_keys, don't insert
    FOR(i, 0, leaf->_keyCount)
        if (key == leaf->_keys[i])
            return false;

    insertInNode(leaf, key, (void *)value);
    return true;
}
```

必須先找到要插入的 key 所在的 leaf node,並排除重複的 key (此 project 下假設並無重複的 key)

並把此 leaf node,key value 傳入 insertInNode

```
template <class T>
void BPT<T>::insertInNode(BPT_NODE* node, T key, void* value) {
    // cout << "node: " << node << ", key: " << key << ", value: " << value << endl;
    // find position to insert
    int insertIdx = 0;
    while (insertIdx < node->_keyCount && node->_keys[insertIdx] < key) insertIdx++;

    // move the key and values after the position right
    for (int i = node->_keyCount; i > insertIdx; i--)
        node->_keys[i] = node->_keys[i-1];
    for (int i = node->_keyCount + 1; i > insertIdx + 1; i--)
        node->_pointers[i] = node->_pointers[i-1];

    // insert key and value
    node->_keys[insertIdx] = key;
    node->_pointers[insertIdx + 1] = value;
    node->_keyCount++;

    // check if this node need to split
    if (node->_keyCount == node->_maxKeyCount) {
        splitNode(node);
    }
}
```

在 insertInNode 當中,
必須先找到此 key 在 leaf node 正確的位置,並且要將此位置右邊的 key 都右移
移完後就可以開始插入 key 和 value,必須注意(底線處)的是在 **leaf node 當中的 pointer 與 key 數量是相等的**
而當 node 中 key 數量滿的時候必須進行 splitNode.

- **BPlusTreeNode 中的 splitNode**

必須是創造一個新的 node 去放右邊的 key,移動在最右邊的 (_maxKeyCount-_maxKeyCount/2)個 key 到新的 node 上, 並調整新的 node 上 key 的 pointer 所指向的位置

如果此 node 不是 root 的話,必須去更新 node 的 father pointer
若 node 為 root, 必須再去再創新的 root

- **BPlusTreeNode 中的 delete**

```
template <class T>
bool BPT<T>::deleteValue(T key) {
    // find the leaf page that the key may be in.
    BPT_NODE* leaf = findLeaf(key);

    // if key in leaf->_keys, delete it
    FOR(i, 0, leaf->_keyCount) {
        if (key == leaf->_keys[i]) {
            deleteInNode(leaf, key);
            return true;
        }
    }

    // if key doesn't exist
    cout << "error: key " << key << " doesn't exist.\n";
    return false;
}
```

先找到此 key 所在的 leaf node, 並找到此 key,將此 key 從此 leaf node 當中刪除

```
template <class T>
void BPT<T>::deleteInNode(BPT_NODE* node, T key) {
    int deleteIdx = 0 ;
    while (key != node->_keys[deleteIdx]) deleteIdx++ ;
    for (int i = deleteIdx; i < node->_keyCount - 1; i++)
        node->_keys[i] = node->_keys[i+1];
    for (int i = deleteIdx + 1; i < node->_keyCount; i++)
        node->_pointers[i] = node->_pointers[i+1];
    node->_keyCount--;

    // if the number of elements in a node < _maxKeyCount/2
    // redistribute the keys and values in siblings
    if (node->_keyCount < (_maxKeyCount+1)/2) {
        redistributeNode(node);
    }
}
```

在 leaf node 當中找到 key 的位置,刪除此 key 並調整 pointer 指的位置

若 node 當中 key 的數目小於 _maxKeyCount/2 的話,
此 node 須要去做 merge, 也就是當中的 redistrubuteNode()

- BPlusTreeNode 中的 redistrubuteNode()

If the page is a leaf page, merge its siblings, and remove the siblings from the paren.

If the page is a non leaf page, merge its siblings and pull down the key from the parent. Then remove the key and the pointer from the parent

3. Test Design:

- Unit Testing:以程式最小的邏輯單元為對象,撰寫測試程式,來驗證邏輯正確與否

  Test_bpt.cpp test_slotted_page.cpp test_relation.cpp test_bptSim.cpp

- 在 B+ tree 當中

  insert (<T> key, int RID) delete (key) 需要檢查的幾種狀況

  i) Duplicate key

  insert (1, 12), insert (1,34)

  ii) 刪除不存在的 key

  insert (1,12), insert (2, 34), delete (3,13)

- Test Result

  In **test_relation.cpp**

```cpp
#include <string>
#include "Relation.h"
using namespace std;

#define FOR(i,a,b) for(int i=(a);i<(b);i++)

int main() {
    auto rel = Relation<int>("student", "integer", 100);
    FOR(i, 0, 100) {
        rel.insertRecord(i, to_string(i));
    }
    FOR(i, 10, 20) {
        rel.deleteRecord(i);
    }
    // rel.insertRecord(3, "ashuf");
    // rel.insertRecord(5, "fwejoilskd");
    rel.scanIndex();
    rel.queryRid(5);
    rel.rangeQueryRid(5, 10);
    rel.printPage(0);
    rel.printStatistics();
    return 0;
}
```

```
Scan index pages of relation student:
    # of leaf pages: 2
    # of total index pages: 3
Query relation student with key 5:
    RID: 65537
Range query relation student between 5 and 10:
    RID list:
    65537
    65538
    65539
    131072
    131073
Records in page 0:
    slot 0: 0
    slot 1: 1
    slot 2: 2
    slot 3: 3
Statistics of relation student:
    # of index pages: 3
    # of slotted data pages: 25
```

4. Case Study: Duplicate Key

- **FUNCTION DISCUSSION:**
  實作 B+ Tree Index 系統時，如果以這次作業的 Function 來看的話，在做一些基本的 Scan, Display Data Page of a Relation, Range Query Using Index 以及 File, Index Statistics 的功能時，是否考量 Duplicate Keys 的狀況，對於上述功能似乎沒有太大的影響，不過 Range Query Using Index 若加入 Duplicate Keys 的話，同一個 Key 值可能同時對應到多個 RID，在這種狀況底下如果要進行 one-to-one Search 的話，可能會失去其功能性。
- **FUNCTION INFLUENCE:**
  考慮 Duplicate Keys 的狀況之後，主要影響較大的功能為：Single Value Index Search 以及 Delete Record。
  - **Single Value Index Search**
    起因於同一個 Key 值會對應到許多個 Record，所以若單純對 Key 值作 Search 的話會產生 Error，系統不曉得該回傳哪一筆結果。

    解決辦法：考慮到 Duplicates Key 的狀況時，當針對同一個 Key 值有多筆 insert 且要對 key 值作 search 的動作時，則在實作上需要增加搜尋條件，如：插入的時間序、RID 或者 Record 的關鍵字。
  - **Delete Record**
    同上，起因於同一個 Key 值會對應到許多個 Record，因此對 Key 值作 delete 的動作時，系統無法判斷該刪除哪一筆資料。
    解決辦法：考慮到 Duplicates Key 的狀況時，當針對同一個 Key 值有多筆 Insert 且要對 Key 值作 Delete 的動作時，則在實作上需要增加其他刪除條件，如：插入的時間序、RID 或者 Record 的關鍵字。