HW05 README

姓名、學號和e-mail

1. 姓名 : 王謙靜
2. 學號 : 409410050
3. e-mail : chen910606@gmail.com

使用的語言與版本

本次作業使用 c++ 撰寫，版本為11.2.0

```
forward@LAPTOP-HNTDAHGN:/mnt/d/database/hw5$ g++ --version
g++ (Ubuntu 11.2.0-19ubuntu1) 11.2.0
Copyright (C) 2021 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

如何操作程式

我把 B+ Tree 寫成一個 class，他的名字是 `BPTree`，把 `BPTree.h` 引入 code中即可使用這個 class。
B+ Tree key 的資料型態為 `int`，存的值為 `string`。

編譯

在 terminal 中，走到檔案資料夾後，輸入 `make` 即可編譯，並產生執行檔 `a.out`
輸入 `make example.out`，可以編譯 `example.cpp`，裡面的 code 是此 readme 中的 example，並產生執行檔 `example.out`。

執行

在 terminal 輸入 `./a.out < testcase2`, `./a.out < testcase3`, `./a.out < testcase4`，可以執行作業投影片中的範例。
直接執行 `a.out` 的話，一開始會先輸入一個大於 0 的整數代表 order，之後會建立一棵B+ Tree，可以藉由下方指令操作B+ Tree。

```
"h" 印出可以用的指令
"+ key, value" 插入一筆(key, value)的新資料，其中key為整數，value 為字串(insert)
"- key" 將 key 所連結到的那筆資料刪除。
"$" 輸出現在 B+ tree 的樣子
"f key" 尋找 key 所指向的資料
"r l, r" 找範圍為 l key 到 r key 的資料
"n l, r" 找範圍不在 l key 到 r key 的所有資料
"q" 離開
```

在離開一般模式之後，會詢問是否要輸出 bulk loading 的 example，輸入 `Y` 會輸出範例，`N` 則會離開。
可將 `main.cpp` 中的第一行的註解拿掉，這樣會將提示字輸出到 `stderr` 這個檔案中。

BPTree member function

以下列出 public 的 member function

constructor(initialize)

提供了兩個 constructor

```
// initialize tree without bulk loading
BPTree(int order);

// initialize tree without bulk loading
BPTree(int order, vector<pair<int, string>> bulk_loading_input);
```

- order : B+ Tree 中最大 key 數等於 order 乘以2
- bulk loading : 最一開始建造 B+ Tree 時，我們可以透過 bulk loading 的方式來限定每一個 page 的 fill factor，以及取得較好的執行效率。在這裡實作時，限定每一個 page 的最大 key 數為 order。
- bulk_loading_input : 型態為裝有 pair 的 vector， pair 的第一項放 key，第二項放 value。若有相同key的資料，只會留下value較小的那一個。

Complexity

- 沒有 bulk loading 時 O(1)
- 有 bulk loading 為排序的複雜度 O(nlogn)，其中 n 為 vector 的大小

Example

```
//without bulk loading
BPTree bpt(4);


//bulking loading example
vector<pair<int, string>> input;
input.emplace_back(9, "nine");
input.emplace_back(10, "ten");
input.emplace_back(8, "eight");
input.emplace_back(1, "one");
input.emplace_back(4, "four");
input.emplace_back(0, "zero");
input.emplace_back(23, "twenty-three");
input.emplace_back(29, "twenty-nine");
BPTree bpt2 = BPTree(2, input);
```

insert

```
int insert(int key, string value);
```

在 B+ Tree 中插入一筆資料，若樹中已存在一筆 key 相同的資料，則會回傳 -1 ，且不會將這筆資料加入樹中，反之則回傳 0，代表插入成功。

Example

```
bpt.insert(1, "one");
bpt.display();
```

輸出

```
1:[]
        1.1:[1]
                1: one
```

erase

因為 delete 為 c++ 的保留字，故將 function 的名稱改為 erase。

```
int delete(int key);
```

刪除樹中對應 key 值的那一筆資料，若整個 tree node 為空，會刪除那個 node ，並將上方的路標刪掉。

Example

```
bpt.erase(1);
bpt.display();
```

輸出

```
The tree is empty now!
```

display

```
void display(void);
```

按照投影片的方法輸出樹的內容，若樹為空則輸出 `The tree is empty now!`

Example

```
bpt2.display();
```

輸出

```
1:[23]
        1.1:[4, 9]
                1.1.1:[0, 1]
                        0: zero
                        1: one
                1.1.2:[4, 8]
                        4: four
                        8: eight
                1.1.3:[9, 10]
```

```
                        9: nine
                        10: ten
        1.2:[]
                1.2.1:[23, 29]
                        23: twenty-three
                        29: twenty-nine
```

find, find_smallest

```
string find(int);
string find_smallest();
```

- find : 找 key 所對應的資料
- fins_smallest : 找最小 key 所對應的資料

Example

```
cout<<bpt2.find(9)<<endl;
cout<<bpt2.find_smallest()<<endl;
```

輸出

```
nine
zero
```

get all data

```
vector<pair<int, string>> get_all_data();
```

回傳樹上所有的資料，回傳型態為裝有 pair 的 vector，pair 的 first是 key，second 是 value。

Example

```
auto res = bpt2.get_all_data();
for(auto e:res){
        cout<<e.first<<":"<<e.second<<endl;
}
```

輸出

```
0:zero
1:one
4:four
8:eight
9:nine
10:ten
23:twenty-three
29:twenty-nine
```

range search

```
vector<pair<int, string>> range_search(int lkey, int rkey);
vector<pair<int, string>> range_search_not(int lkey, int rkey);
```

- range_search 找範圍從 lkey 到 rkey 的資料。
- range_search_not 找範圍不在 lkey 到 rkey 內的資料。
- 回傳型態皆為裝有 pair 的 vector，pair 的 first是 key，second 是 value

Example

```
cout<<"range search"<<endl;
res = bpt2.range_search(4, 8);
for(auto e:res){
        cout<<e.first<<":"<<e.second<<endl;
}
cout<<"range search not"<<endl;
res = bpt2.range_search_not(4, 8);
for(auto e:res){
        cout<<e.first<<":"<<e.second<<endl;
}
```

輸出

```
range search
4:four
8:eight
range search not
9:nine
10:ten
23:twenty-three
29:twenty-nine
0:zero
1:one
```

destructor

很平常的的讓物件消失

reference

- https://www.youtube.com/watch?v=lTJy2MXI4lY
- https://github.com/solangii/b-plus-tree
- https://www.796t.com/content/1545533123.html
- https://www.geeksforgeeks.org/introduction-of-b-tree/
- https://hackmd.io/@w8qbx0fdRK2ETRnEzcLK2A/SyjKs-mxg?type=view

bonus

- bulk loading : 在離開後，會輸出另外一棵由bulk loading 初始化的 B+ Tree
- range search : 可以找範圍內的所有值
- range search not: 可以找不在範圍內的所有值

- 遇到相同資料的處理：會先判斷key有沒有在B+ Tree中，若已存在則不會放入B+ Tree中