2024 B+ tree implementation assignment

Course name: Database Systems (ITE2038)

- B+ tree index 구현

Professor: Sang-Wook Kim (email: wook@hanyang.ac.kr)

TAs: Dong-Hyuk Seo (email: hyuk125@agape.hanyang.ac.kr)
Seongeun Ryu (email: ryuseong@agape.hanyang.ac.kr)

1. Assignment Title

- Implementation of a B+ tree index

2. Environment

- OS: Windows or mac os
- Language: Java or Python (any version is ok)
  - C++ language is also allowed, but not recommended

3. Constraints - Overall

- The B+ tree index should be stored in a single file (index file)
- The file contains all the meta information for the index and the index nodes
- **The internal organization of the file is not considered in grading**
- The program should provide following functions:
  - **Search**
    - ✔ A single key search **AND** a range search
  - **Insertion of a key**
  - **Deletion of a key**
    - ✔ The deleted entry should be completely removed from the index and the file
- Assumption
  - Keys and values are all in the integer type
  - **Duplicated keys are not allowed for insertions**
  - The keys in a node are stored in an ASCENDING order
- **POLICY on COPY • DO NOT COPY someone else's program**
  - **DO NOT USE functions/methods/routines from existing code/library/programs in pre-implemented B+ tree indexes or any other similar tree-based indexes**
  - **All these actions are regarded as COPY and so will be handled accordingly**

4. Constraints - Internal Structure

- Each node of a B+ tree index should contain the following data inside:
  - Non-leaf node
    - ✔ *m*: # of keys
    - ✔ *p*: an array of<*key, left_child_node*> pairs
    - ✔ *r*: a pointer to the rightmost child node
  - Leaf node
    - ✔ *m*: # of keys
    - ✔ *p_*: an array of<*key, value(or pointer to the value)*> pairs
    - ✔ *r*: a pointer to the right sibling node

5. Constraints - Interface

- The program *should support* ***command-line interface***
- The following commands should be implemented:
  - **Data File Creation**
    - ✔ Command: *program -c index_file b*
      - □ *program*: name of the program (bptree)
      - □ *index_file*: name of a new index file
      - □ *b*: size of each node (max. # of child nodes)
    - ✔ This command creates a new index file containing an empty index with node size *b*
      - □ If the file already exists, it is overwritten
    - ✔ Example
      - □ **java bptree -c index.dat 8**
  - **Insertion**
    - ✔ Command: program -i *index_file data_file*
      - □ *data_file*: name of the input data file that has a number of key-value pairs to be inserted
    - ✔ This command inserts all the key-value pairs inside the data_file into the index in the index_file
      - □ The insertion causes the modification of the index file
      - □ Insertions are performed in the same order of key-value pairs in the data file
    - ✔ The data file is provided as a .csv file (Comma Separated Values)

      - □ Each line of the data file contains a key-value pair
        - <key>,<value>\n
      - □ Data file example (input.csv)

```
53,3358290
16,6334568
63,128174
91,2455794
38,999283
81,3386744
99,28612
55,9139826
1,7697582
61,3415875
```

쉼표로 구분
(하나의 행이 레코드)

실제값
(값을 가리키는 포인터라고 해도됨)
이렇게 생각하는게 편할듯.

index, (key)

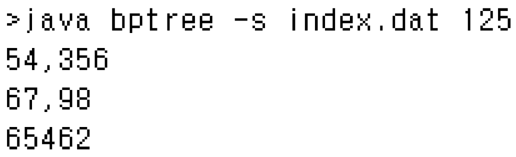
    - ✔ Example
      - □ **java bptree -i index.dat input.csv**
  - **Deletion**
    - ✔ Command: *program -d index_file data_file*
      - □ *data_file*: name of the input data file that has a number of keys to be deleted
    - ✔ This command deletes all the key-value pairs inside the input data file from the index
      - □ The deletion causes the modification of the index file
      - □ Deletions are performed in the same order of keys in the data file
    - ✔ The input data file is provided as a .csv file (Comma Separated Values)
      - □ Each line of the data file contains only a key value

            - <key>\n
        - Deletion file example (delete.csv)

```
63
99
1
53
91
```

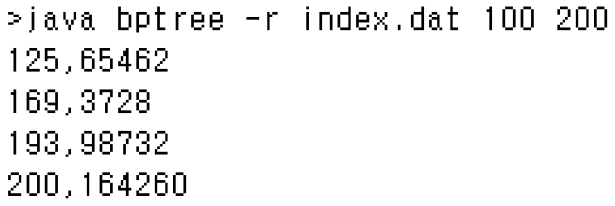
    - Example
        - **java bptree -d index.dat delete.csv**

- **Single Key Search**
    - Command: *program -s index_file key*
        - *key* : key value to be searched
    - This command returns a value of a pointer to a record with the key
    - Output format
        - Print output to the *stdout*
        - While searching, the program prints each non-leaf node in the path that the search passes through
            - Print all the keys in the node in a single line
            - <key1>,<key2>,…,<keym>\n
        - When the search reaches the leaf node having the search key, print the value matched with the search key
            - <value>\n
            - If not found, print 'NOT FOUND'
        - Example (This is not the same dataset as above example.)
            - **java bptree -s index.dat 125**



```
>java bptree -s index.dat 125
54,356
67,98
65462
```

- **Ranged Search**
    - Command: *program -r index_file start_key end_key*
        - *start_key*: lower bound of the range search
        - *end_key*: upper bound of the ranged search
    - This command returns the values of pointers to records having the keys within the range provided
    - Output format
        - Print output to the *stdout*
        - Print all the key-value pairs with the key between *start_key* and *end_key* (**including start_key and end_key**)
            - <key1>,<value1>\n<key2>,<value2>\n…
        - Note that *start_key* and *end_key* may not be in the index
            - The program prints only the key-value pairs between them
    - Example
        - **java bptree -r index.dat 100 200**



```
>java bptree -r index.dat 100 200
125,65462
169,3728
193,98732
200,164260
```

6. How to turn in

(1) Write your program

(2) Write a document (pdf file) that contains:

- Summary of your algorithm
- Detailed description of your codes (for each function)
- Instructions for compiling your source codes at TA's computer (e.g. screenshot) (Important!!)
  - You MUST SUBMIT instructions for compiling your source codes. If TAs read your instructions but cannot compile your program, you will get a penalty. Please, write the instructions carefully.
- Any other specification of your implementation and testing

(3) Put what you wrote in the submission folder named 'B-tree_Assignment_OOO'

- Submissions should contain an executable file, all source files, and the document
  - If you use python, there is no need to contain an executable file.
- The file structure is as follows
  - B-tree_Assignment\
    Source\
    {Your source files}.java/.py\
    {Your executable file}.exe/.jar
    {Your document}.pdf

(4) Push it to the **LMS**

- **Due date**
  - **Completed before 25 September: 100%**
  - **Completed before 2 October: 70%**
  - **After 2 October: 0%**

You can ask questions about the assignment via Piazza(announced later) community.
YOU WILL GET SERIOUS PENALTIES IF YOU DO COPY OR CHEAT

Good luck!