Assignment #3

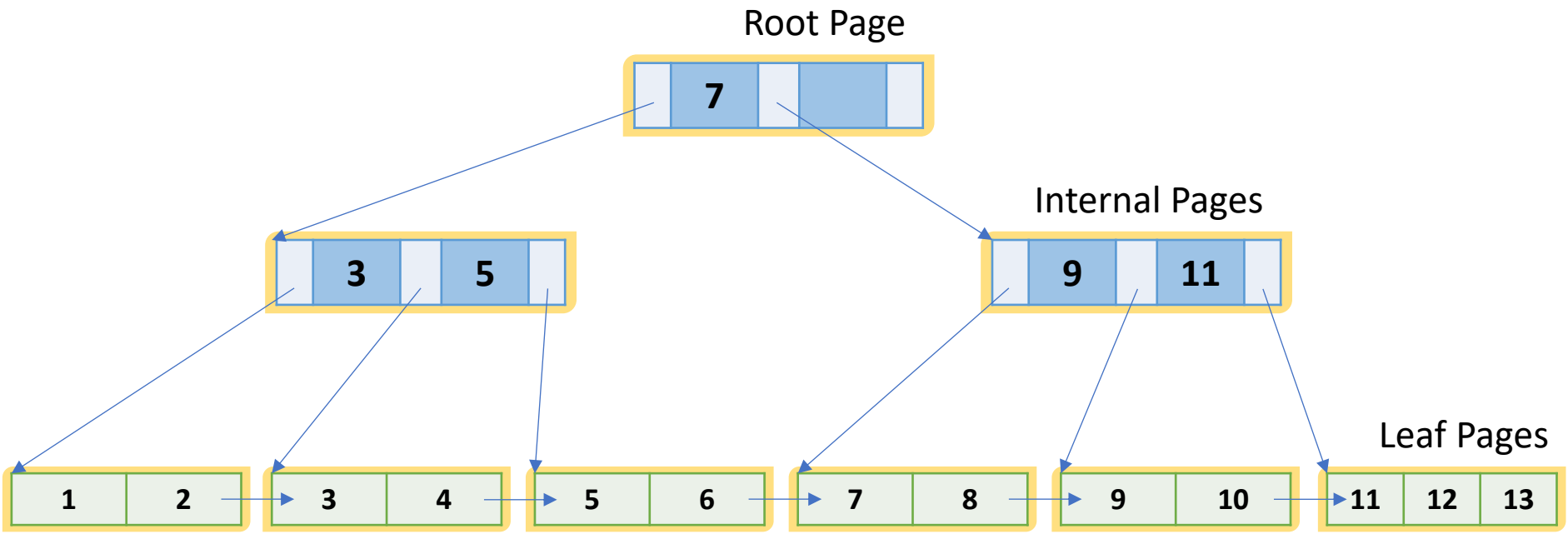
Implementing Augmented B+tree

Implementing Augmented B+tree

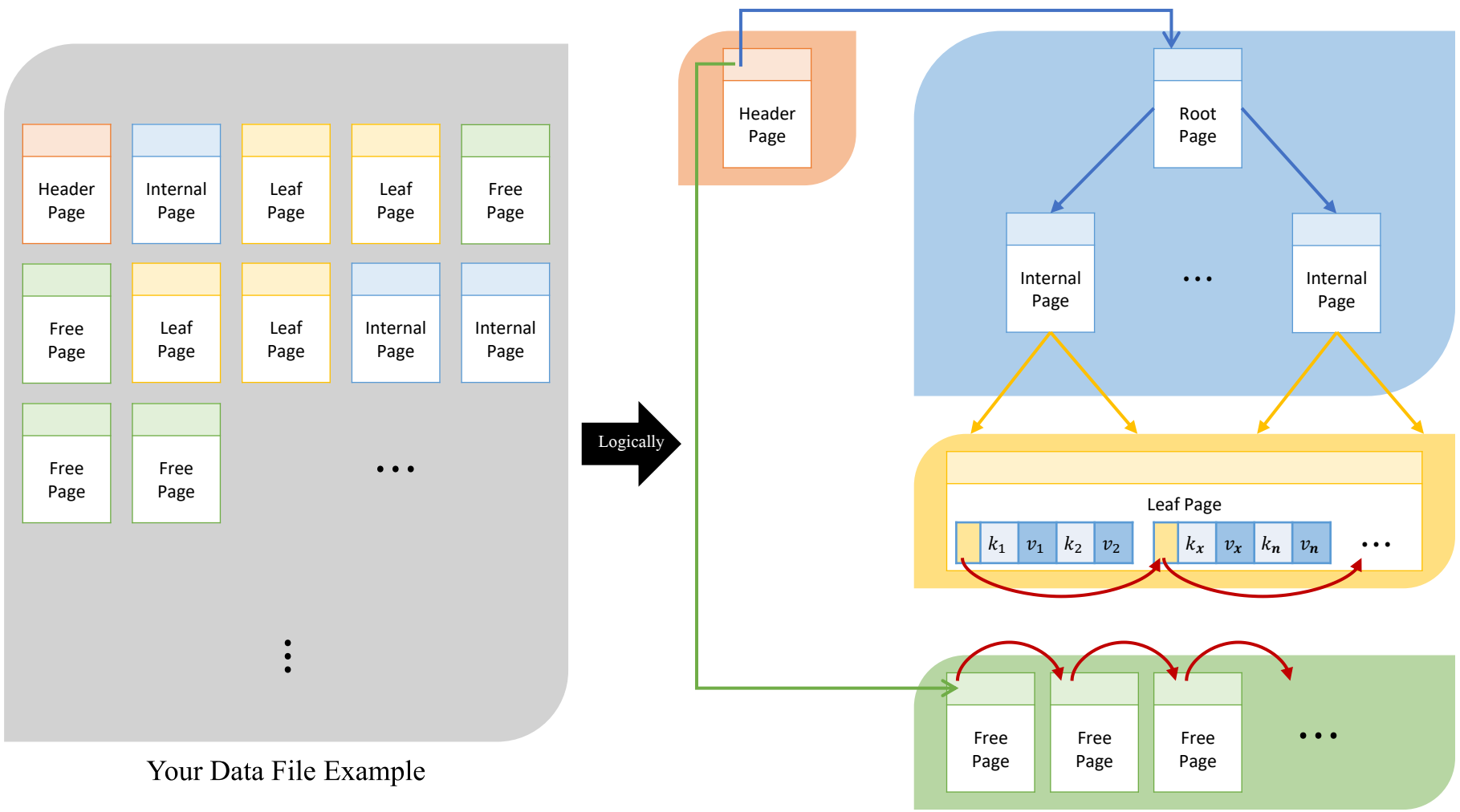
- 이번 과제는 insertion에 key-rotation을 적용한 B+ tree를 구현하고 key-rotation이 성능에 미칠 영향을 추측하는 과제입니다.

Disk-based B+tree

- B+ tree의 노드 하나를 한 페이지(4KB)에 저장하는 방식으로 B+ tree 전체를 한 파일에 저장할 수 있습니다.
- 그 파일은 Disk에 저장되므로 B+tree management 프로그램이 종료되더라도 B+tree의 구조와 내용물은 그대로 남아있게 됩니다.



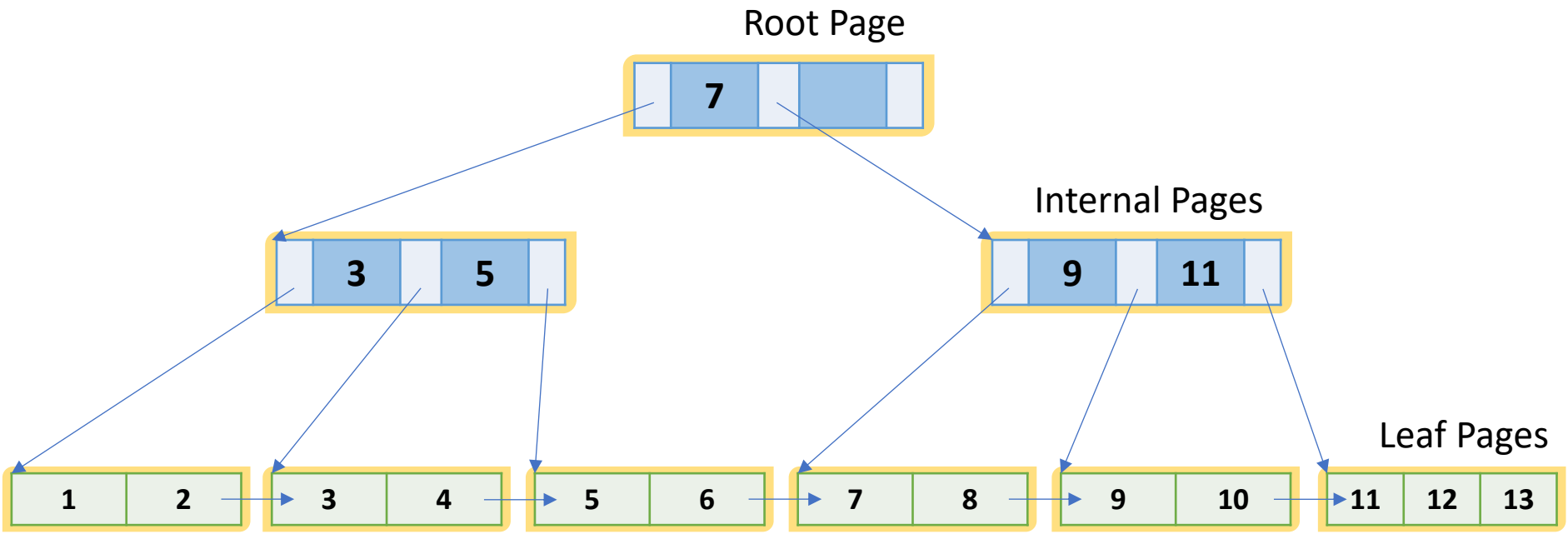
Disk-based B+tree



Your Data File Example

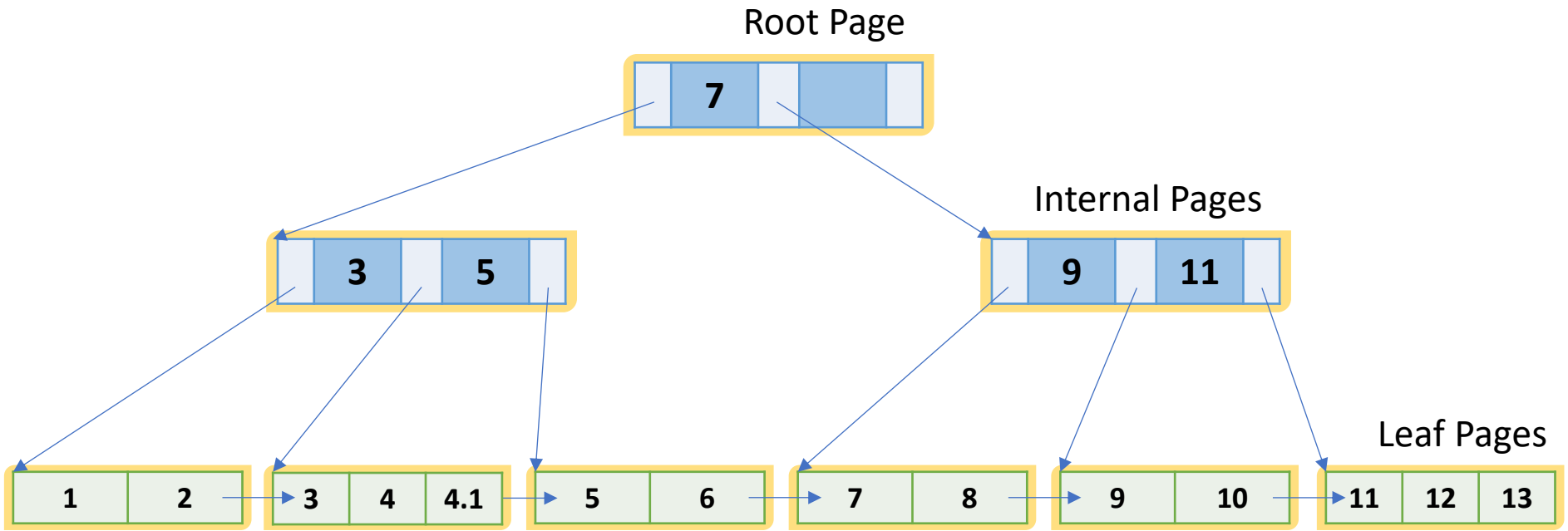
Disk-based B+tree

- B+ tree의 노드 하나를 한 페이지(4KB)에 저장하는 방식으로 B+ tree 전체를 한 파일에 저장할 수 있습니다.
- 그 파일은 Disk에 저장되므로 B+tree management 프로그램이 종료되더라도 B+tree의 구조와 내용물은 그대로 남아있게 됩니다.



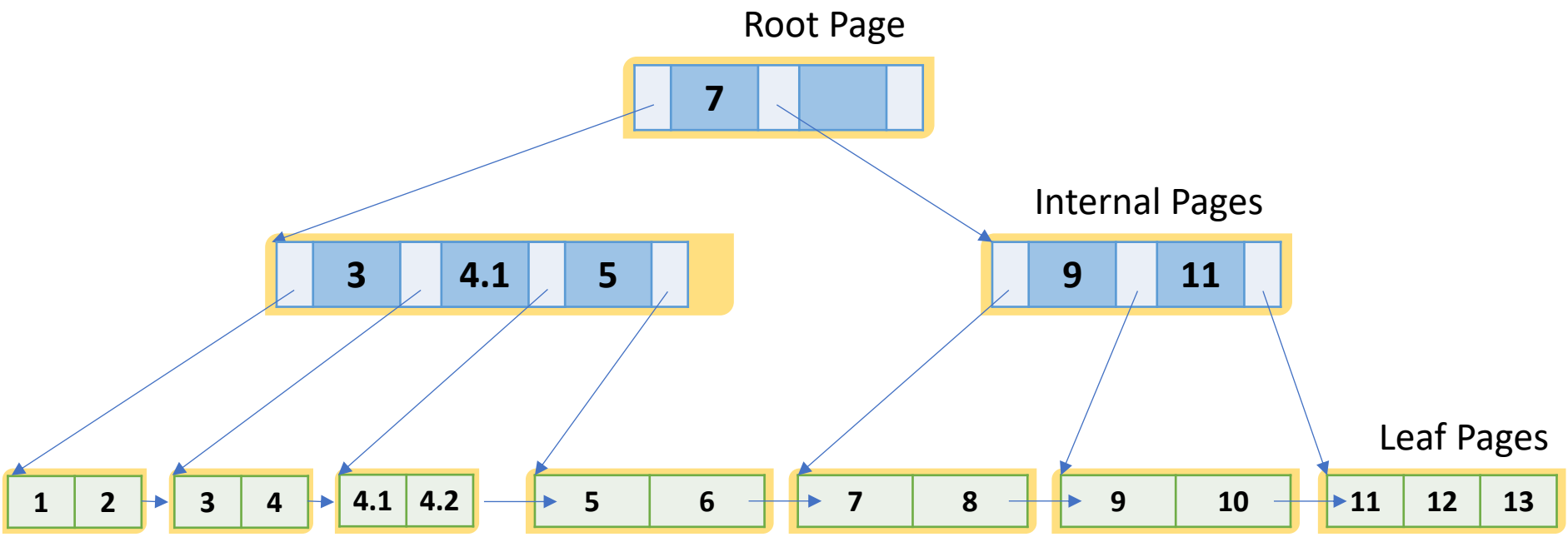
key-rotation insert

- order가 4인 B+tree가 아래와 같은 구조를 가지고 있을 때 4.2를 삽입하면 split이 발생하게 됩니다.
- 최대 key 개수를 초과한 node에서 가장 큰 key를 sibling node로 이동시킬 수 있다면 split 대신 가장 큰 key를 이동 시킵니다.
- sibling node로 이동이 불가능한 경우 node split을 수행합니다.
- 이를 key-rotation insert라고 하겠습니다.



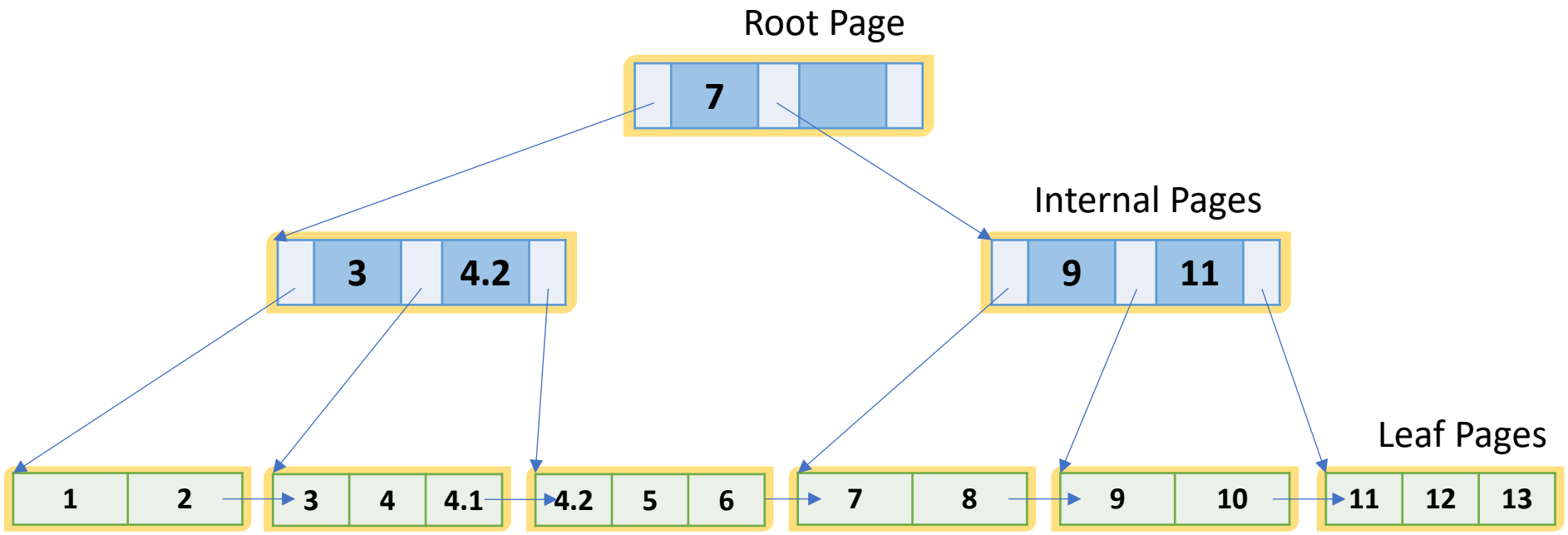
key-rotation insert

- key-rotation insert를 적용하지 않은 경우



key-rotation insert

- key-rotation insert를 적용한 경우



key-rotation insert

- key-rotation insert가 B+ tree 성능 향상에 어떤 영향을 미칠지 고찰 및 분석하여 위키에 작성해주시길 바랍니다.
- 실험적인 분석, 수학적인 증명 등 어떤 방식을 택해도 상관없습니다.
- 주장과 근거의 논리성을 바탕으로 점수가 부여될 예정입니다.

Specification

❖ 아래의 명세를 모두 만족해야 합니다

- main.c는 아래의 기능을 수행하도록 구현되었습니다.

1. **open <pathname>**
   - Open existing data file using ‘pathname’ or create one if not existed.
   - All other 3 commands below should be handled after open data file.
2. **insert <key> <value>**
   - Insert input ‘key/value’ (record) to data file at the right place.
   - Same key should not be inserted (no duplicate).
3. **find <key>**
   - Find the record containing input ‘key’ and return matching ‘value’.
4. **delete <key>**
   - Find the matching record and delete it if found.

Specification

❖ 아래의 명세를 모두 만족해야 합니다

- Your library (libbpt.a) should provide those API servies.

1. **int open_table (char *pathname); → Already implemented**
   - Open existing data file using ‘pathname’ or create one if not existed.
   - If success, return the unique table_id, which represents the own table in this database. Otherwise, return negative value.

2. **int db_insert (int64_t key, char * value);**
   - Insert input ‘key/value’ (record) to data file at the right place.
   - If success, return 0. Otherwise, return non-zero value.
   - Apply key-rotation inserting method

3. **char* db_find (int64_t key);**
   - Find the record contatining input ‘key’.
   - If found matching ‘key’, store matched ‘value’ string n ret_val and return 0. Otherwise, return non-zero value.
   - Memory allocation for record structure(ret_val) should occur in caller function.

4. **int db_delete (int64_t key);**
   - Find the matching record and delete it if found.
   - If success, return 0. Otherwise, return non-zero value.

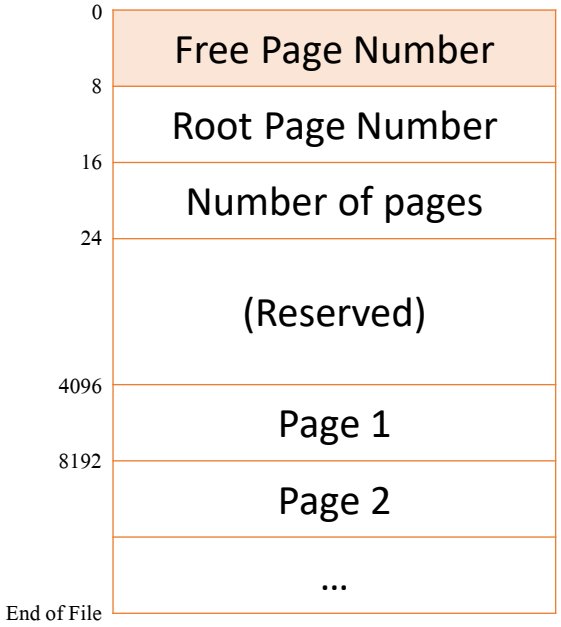
Specification

❖ 아래의 명세를 모두 만족해야 합니다

- All update operation (insert / delete) should be applied to your data file as an operation unit. That means one update operation should change the data file layout correctly.
- Note that your code must be worked as other students' data file. That means, your code should handle open(), find(), insert(), and delete() API with other students' data file as well.
- So follow the data file layout described from next slides.
  - We fixed the on-disk page size with 4096 Bytes.
  - We fixed the record (key + value) size with 128 (8 + 120) Bytes.
    - Type : (key ➔ integer) & (value ➔ string)
  - There are 4 types of page
    1. Header page (special, containing metadata)
    2. Free page (maintained by free page list)
    3. Leaf page (containing records)
    4. Internal page (indexing internal/leaf page)

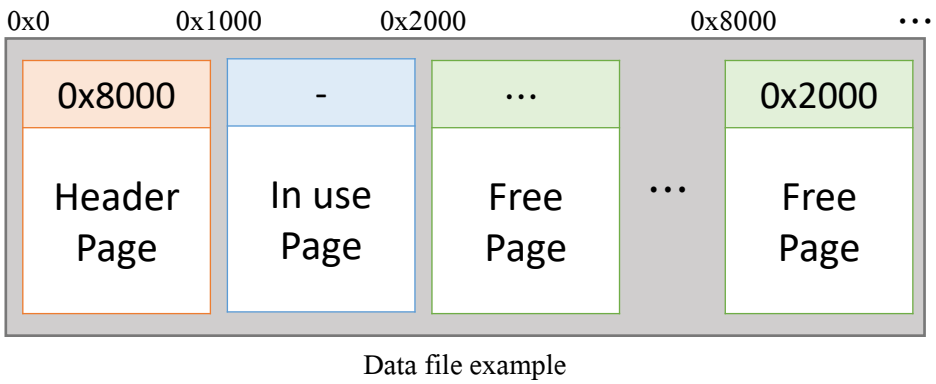
Specification

- **Header Page (Special)**
  - Header page is the first page (offset 0-4095) of a data file, and contains metadata.
  - When we open the data file at first, initializing disk-based B+tree should be done using this header page.
    - Free page number : [0-7]
      - points the first free page (head of free page list)
      - 0, if there is no free page left.
    - Root page number : [8-15]
      - pointing the root page within the data file.
    - Number of pages : [16-23]
      - how many pages exist in this data file now.

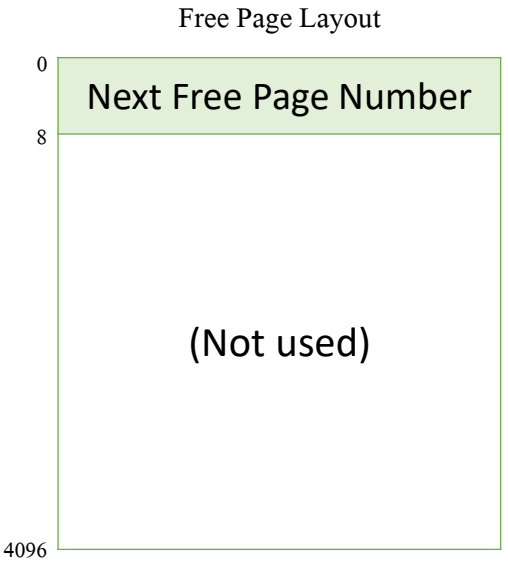


Specification

- **Free Page**
  - In previous slide, header page contains the position of the first free page.
  - Free pages are linked and allocation is managed by the free page list.
    - Free page number : [0-7]
      - points the first free page (head of free page list)
      - 0, if there is no free page left.



Data file example



Specification

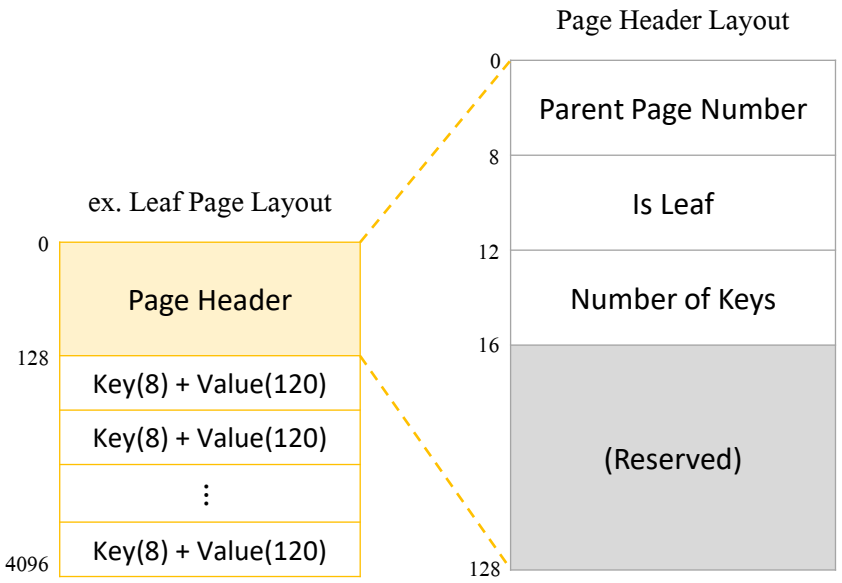
- **Page Header**
  - Leaf / Internal page have first 128 bytes as a page header.
  - Leaf / Internal page should contain those data

    (see the *node* structure in include/bpt.h)
    - Parent page number : [0-7]

      - If Leaf / internal page, this field points the position of parent page.
    - Is Leaf : [8-11]

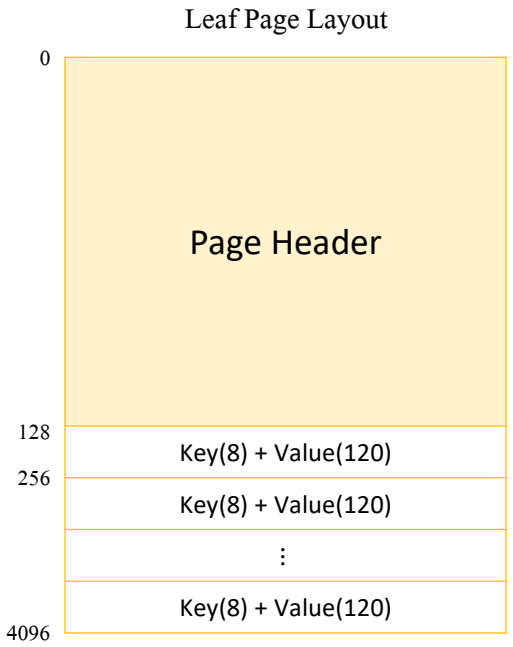
      - 0 is internal page, 1 is leaf page.
    - Number of keys : [12-15]

      - the number of keys within this page.



Specification

- **Leaf Page**
  - Leaf page contains the key/value records. (Keys are sorted in the page)
  - One record size is 128 bytes and we contain maximum 31 records per one data page.
  - First 128 bytes will be used as a page header for other types of pages.
  - Branching factor (order) = 32

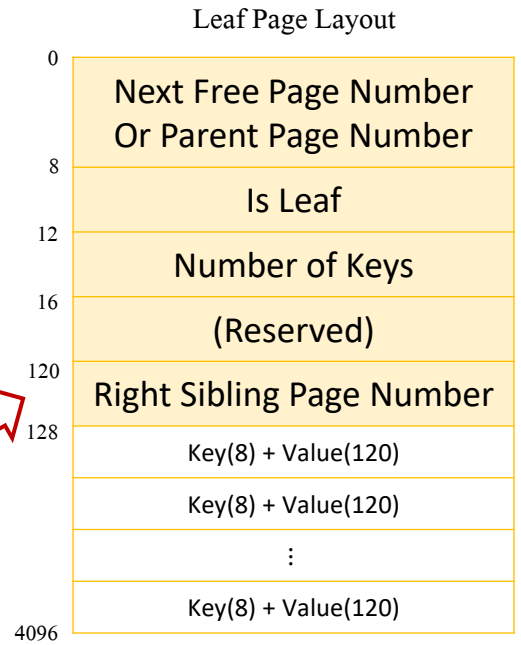


Specification

- **Leaf Page**
  - Leaf page contains the key/value records. (Keys are sorted in the page)
  - One record size is 128 bytes and we contain maximum 31 records per one data page.
  - First 128 bytes will be used as a page header for other types of pages.
  - Branching factor (order) = 32

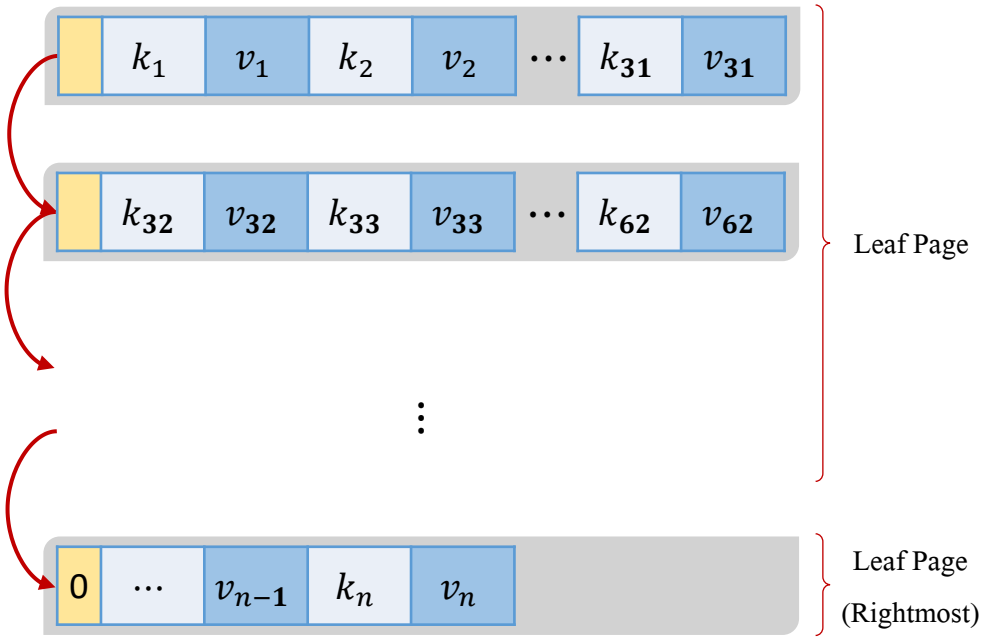
- We can say that the order of leaf page in disk-based B+tree is 32, but there is a minor problem.
- There should be one more page number added to store right sibling page number for leaf page.

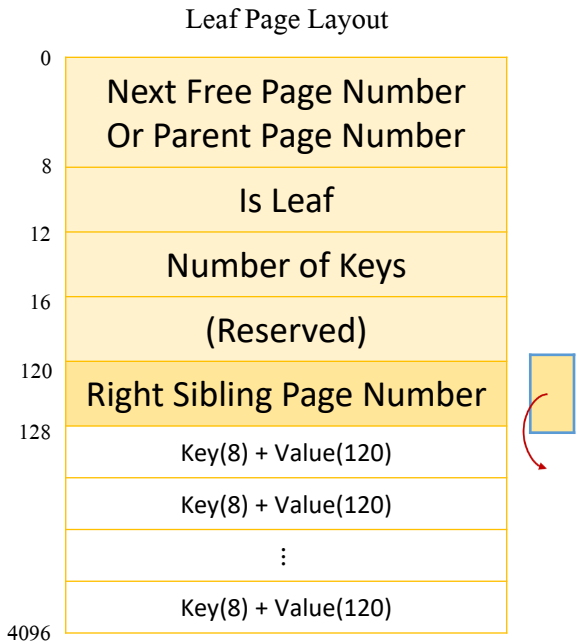
  (see the comments of $node$ structure in include/bpt.h)
- So we define one special page number at the end of page header.
- If rightmost leaf page, right sibling page number field is 0.



Specification

- **Leaf Page**





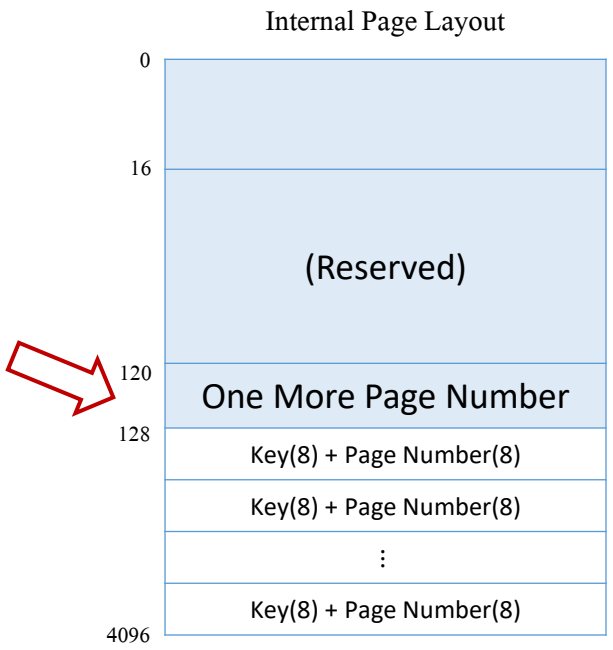
Specification

- **Internal Page**
  - Internal page is similar to leaf page, but instead of containing 120 bytes of values, it contains 8 bytes of another page (internal or leaf) number.
  - Internal page also needs one more page number to interpret key ranges and we use the field which is specially defined in the leaf page for indicating right sibling.
  - Branching factor (order) = 249

➢ Internal page can have maximum 248 entries

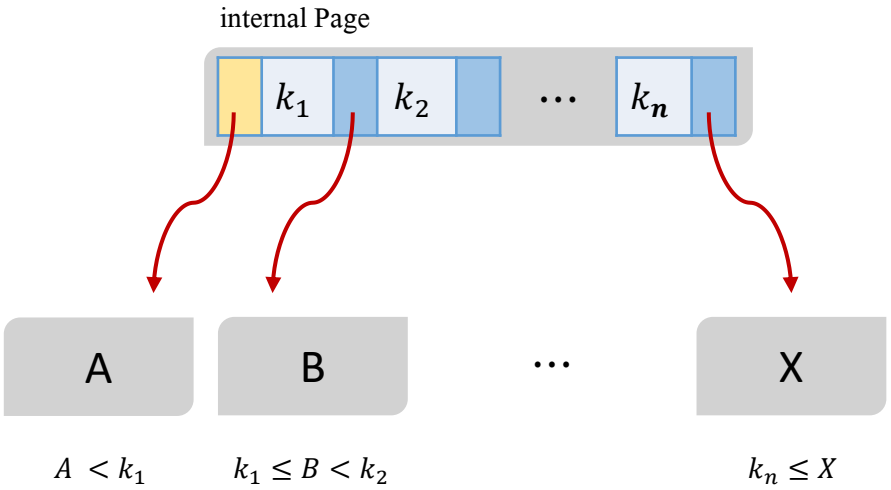
(We learned in LAB class)

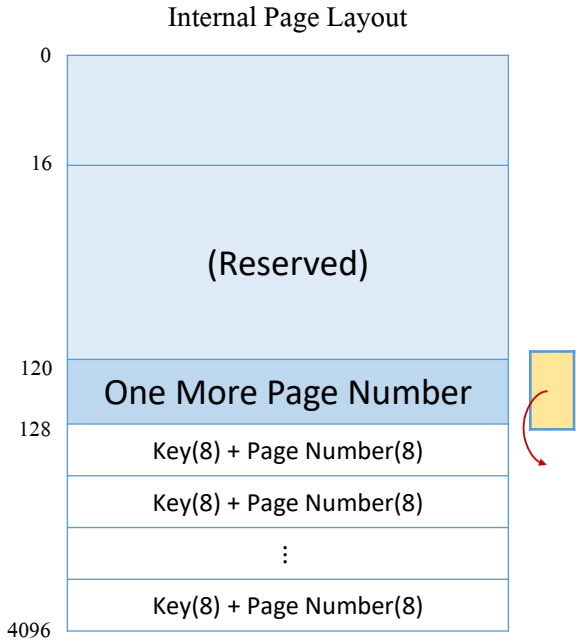
➢ (4096 - 128) / (8+8) = 248.



Specification

- **Internal Page**



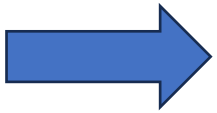


Specification

- 과제 수행에 필요한 파일이 제공됩니다.
- 제공된 파일은 아래와 같은 구조를 가지고 있습니다.
  - ass_3_files
    - include
    - lib
    - src
      - bpt.c
      - main.c
    - Makefile
- Makefile은 절대 수정하지 마세요.
- main.c에 open_table 안의 인자를 “test.db”에서 “DB[학번].db”로 수정하여 제출하세요.

```c
#include "bpt.h"

int main(){
    int64_t input;
    char instruction;
    char buf[120];
    char *result;
    open_table("test.db");
```



```c
#include "bpt.h"

int main(){
    int64_t input;
    char instruction;
    char buf[120];
    char *result;
    open_table("DB2022123123.db");
```

Specification

- main.c의 다른 부분은 절대 수정하지 마세요.
- bpt.c에 적절한 내용을 추가하거나 수정하여 과제를 수행하세요.
  - Disk read/write와 관련된 API가 일부 구현되어 있으며 이를 적절히 활용해도 됩니다.
  - 그 외에 과제 수행에 필요한 다른 함수는 직접 정의하여 사용하세요.
- 제공된 in-memory B+tree 코드를 참고하여 과제를 수행하시길 바랍니다.

Judging System

- 이번 프로젝트의 채점 환경(**Default Version**)은 아래와 같습니다

```
gcc (Ubuntu 11.4.0-1ubuntu1~22.04.1) 11.4.0
GNU Make 4.3
```

❖ Version차이로 인하여 생기는 문제는 조교가 책임지지 않으며 **빌드와 실행이 안 될 경우 점수는 0점 입니다**

Submit

❖ Code

- **Completeness** : 명세의 요구 조건을 모두 올바르게 구현해야 합니다
- **Defensiveness** : 발생할 수 있는 예외 상황에 대처할 수 있어야 합니다
- **Comment** : 코드에는 반드시 주석이 있어야 합니다

❖ Wiki

- **Design** : 명세에서 요구하는 조건에 대해서 어떻게 구현할 계획인지, 어떤 자료구조와 알고리즘이 필요한지, 수업시간에 배운 이론이 어떻게 적용되어야 하는지 등 자신만의 디자인을 서술합니다
- **Implement** : 코드를 그대로 복사하여 문서에 붙여놓지 마시고 본인이 새롭게 구현하거나 수정한 부분에 대해서 무엇이 기존과 다른지, 해당 코드가 무엇을 목적으로 하는지에 대한 설명을 구체적으로 서술합니다
- **Result** : 해당 명세에서 요구한 부분이 정상적으로 동작하는 실행 결과를 첨부하고, 이에 대한 동작 과정에 대해 설명합니다
- **Trouble shooting** : 과제를 수행하면서 마주하였던 문제와 이에 대한 해결 과정을 서술합니다. 혹여 문제를 해결하지 못하였다면 어떤 문제였고 어떻게 해결하려 하였는지에 대해서 서술합니다.
- **Key-rotation insert에 대한 고찰** : key-rotation insert가 B+tree 성능에 어떤 영향을 미칠지(전체적으로 좋아진다/별 차이없다/ 전체적으로 나빠진다/특정 경우에서만 좋아진다/ 특정 경우에서만 나빠진다 등등)고찰하고 이에 대한 근거를 작성합니다.

Submit

❖ 아래와 같이 디렉토리를 구성한 후 압축하여 제출하시길 바랍니다.

- ass_3_[이름]_[학번] (예시: ass_3_주정현_2022123123)
- └ include
- └ lib
- └ src
  - └ bpt.c
  - └ main.c
- └ Makefile
- └ Wiki(ass_3_wiki_[이름]_[학번].pdf)

- **제출 양식을 엄수하고 의미 없는 파일 및 디렉토리는 반드시 제거하여 제출하시길 바랍니다.**
- **제출 기한: 2024년 11월 28일 23시 59분**
- **추가 제출 기한: 2024년 12월 1일 23시 59분 (이후 제출시 0점처리)**
- **추가 제출시 총점에서 (14 + ((늦은 시간(분) / 12) * 0.1)%가 감점됩니다.**
  - 예시) 2시간 지각 제출 후 80점을 득점했다면 결과적으로 68점(80 * (1 – 0.15))을 얻게 됩니다.