Disk Based Buffered BPT

2014004893
이대인

structure.h

BPT 프로그램이 구현되는데에 있어 필요한 모든 구조체를 담고 있는 파일입니다.
Buffering을 추가하게 되면서 모든 함수는 기본적으로 buffer_structure을 사용하여 페이지를 로드/사용 합니다.

```
typedef struct buffer_structure {
    union {
        struct {
            off_t fpo;//free page offset
            off_t rpo;//root page offset
            int64_t num_pages;
            char reserved[4072];
        };
        struct {
            struct page_header;
            union {
                branch_factor entries[248];
                record records[31];
            };
        };
    };
    int tid;
    off_t cpo; //current page offset
    bool is_dirty;
    bool refbit;
    int pin_count;
} buffer_structure;
```

또한 각각의 DB(프로그램)은 실행시에 buffer_manager를 생성하여 버퍼를 관리하게 됩니다.
버퍼관리에는 clock policy를 사용하였습니다.

```
typedef struct buffer_manager {
    int capacity;
    int last_buf;
    //
if binary search is in use, this will tell how many buffers are in
 use for each table
    int table_size[10];
    //buffer location is sorted and will be used for binary search
    struct buf_lookup *buffer_lookup[10];
    struct buffer_structure *buffer_pool;
} buffer_manager;
```

page.h

해당 DB는 모든 정보를 page 단위로 사용합니다.
page의 종류로는 header page, internal page, leaf page가 있으면 모두 4096 bytes 입니다.
page.h 는 페이지를 다루는 모든 함수를 포함합니다.

```
#define toggle_bs false
```

toggle_bs가 true로 설정되었고 num_buf가 100보다 클 경우 버퍼 검색에 있어 binary search algorithm을 사용합니다.
find operation이 많은 경우 훨신 빠른 (O(logN)) 속도를 보장하지만 insertion/deletion이 많은 경우 버퍼를 정렬 시키는 오버헤드가 더 크기 때문에 오히려 DB의 성능을 저하 시킬 수 있습니다. (sorting overhead = O(2 * N) > linear search O(N))

```
int init_db(int num_buf)
```

해당 프로그램 실행시에 필수적으로 실행시켜야 하는 함수입니다.
buffer_manager를 num_buf로 받아온 인자의 크기로 버퍼를 설정하며 초기화 시킵니다.
프로그램 종료시에 필요한 함수입니다.
작업에 실패했을시 -1을 반환하며 성공시 0을 반환합니다.

```
int shutdown_db(void)
```

버퍼 매니저에 남아있는 모든 dirty page를 디스크로 옮기는 작업과 동적할당 되어있던 것들을 free 시킵니다.
성공시 0을 반환합니다.

```
int open_table(char *pathname)
```

테이블(파일)을 열어 DB가 사용가능하게 만들어 줍니다. 해당 파일이 존재할경우 파일을 열며 없을시 새로 생성합니다.
테이블은 최대 10개까지 열 수 있습니다.
파일을 여는데 실패했을시 -1을 반환하며 성공시 0을 반환합니다.

```
int close_table(int table_id)
```

사용하였던 테이블을 닫습니다. 버퍼에 남아있던 해당 테이블의 페이지들을 디스크에 저장하며 해당 table_id는 후에 새롭게 테이블을 열어 사용할 수 있습니다.
작업에 실패했을시 -1을 반환하며 성공시 0을 반환합니다.

```
buffer_structure* open_page(int table_id, off_t po);
```

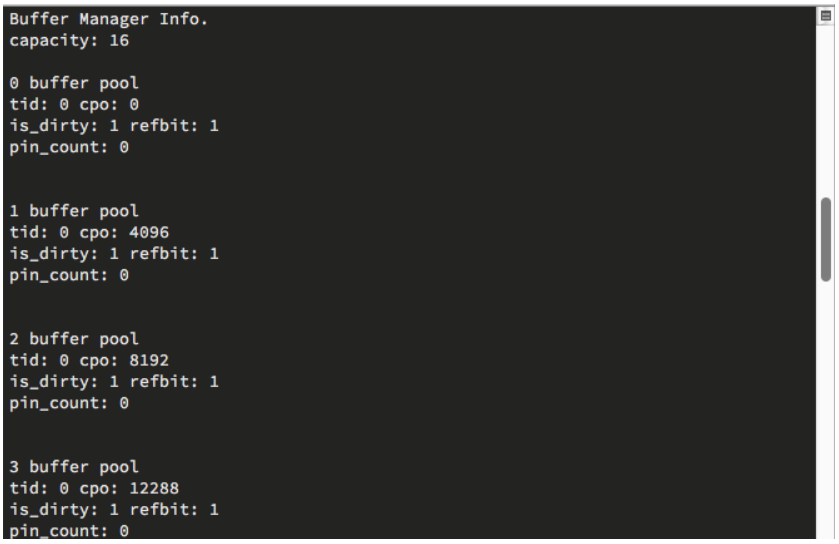
해당 offset에 위치한 페이지를 불러옵니다. 만약 버퍼 풀에 존재할 경우 그대로 리턴 시키며, 버퍼풀에 없을경우 디스크에서 페이지를 읽어옵니다.
버퍼 풀은 CLOCK POLICY에 의거하여 불필요한 페이지를 빼고 새로운 페이지를 집어넣을 수 있습니다.

```
void write_buffer(buffer_structure *cur_buf);
```
open_page()에서 빼는 페이지가 dirty page라면 디스크에 변경된 페이지를 쓰는 작업을 실시합니다.

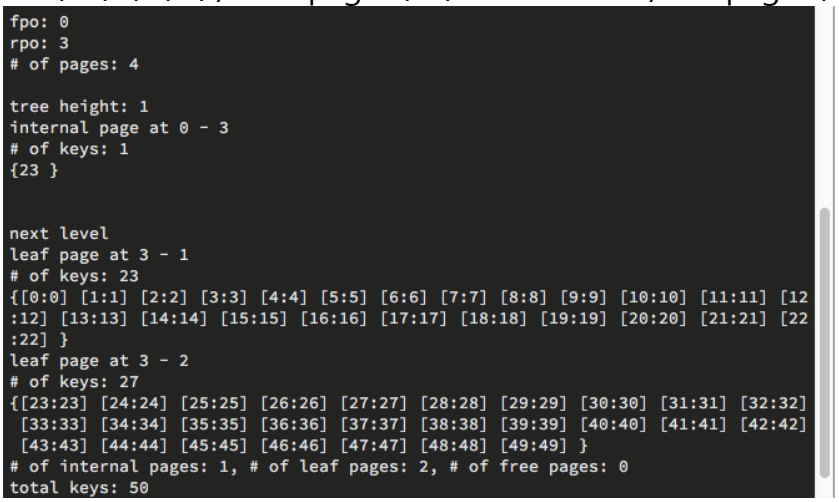
```
buffer_structure* get_free_page(int table_id, off_t ppo, off_t *page_loc, int is_leaf);
```
free page가 있을 경우 해당 free page를 반환하며, 없을 경우 DB file의 크기를 증가시키며 새로운 페이지를 생성 합니다. 해당 함수는 DB에 새로운 페이지를 생성할 수 있는 유일한 함수입니다.

```
void add_free_page(int table_id, off_t page_loc);
```
entry/record의 수가 0인 페이지를 free page로 전환합니다. 전환된 free page는 header page의 fpo에 의해 관리됩니다.

```
void show_buffer_manager(void);
```
현재 버퍼의 상태를 출력합니다.




```
void print_tree(int table_id);
```
현재 테이블의 상태를 출력합니다.
전체 페이지 수/ free page 수와 모든 internal/leaf page의 자료들을 출력시킵니다.



bpt.h

Amittai Aviram의 코드를 사용, Disk Base & Buffering을 위하여 함수를 변경하였습니다.

```
void find_and_print(int table_id, int64_t key);
```
해당 table에서 key를 찾아 존재할시 "key: , value: "의 형태로 출력합니다.
key가 없을 경우 "Record not found under key: "를 출력합니다.

```
char* find(int table_id, int64_t key);
```
해당 table에 key값이 존재할경우 value 문자열을 반환합니다. key가 없을 경우 NULL을 반환합니다.

```
int insert(int table_id, int64_t key, char *value);
```
해당 table에 <key:value>를 저장합니다.
value 문자열이 너무 길거나, 키값이 이미 존재할 경우 -1을 반환합니다.
성공시 0을 반환합니다.

```
int delete(int table_id, int64_t key);
```