Project#2 Milestone2

2018007938 김민관

Design

B+ tree의 경우 탐색은 빠르지만, insert와 delete 연산을 진행할 경우, internal node를 merge하거나 다기 redistribute를 하는 보조연산의 overhead가 크다.

이번 프로젝트의 목적은 보조연산의 overhead를 줄이는것이 목적으로 Delay-merge 방식을 사용하여 overhead 를 줄인다.

기존의 delete의 경우, node에 삭제하고자 하는 key가 있을 경우 key를 삭제하고 삭제가 일어난 node가 root node가 아니고, 삭제가 일어난 node 의 key 개수가 최소 key의 개수보다 작을 때, merge나 redistribution이 발생한다. 따라서 key개수가 최소 key 보다 작으면 다음에 다시 insert를 하는 경우가 있더라도 무조건 merge나 redistribute를 진행한다. 이 보조연산의 횟수를 줄이기 위해 Delay-merge의 경우 node의 key를 삭제하면서 보조연산을 매번 진행하지 않고, node의 key개수가 0이면, 즉 node의 모든 key 가 사라질 경우에만 보조연산을 실행하여 보조연산의 횟수를 줄이고, overhead도 줄인다.

Implement

구현은 의외로 간단하게 진행했다. 해당 key를 삭제한 후, 삭제한 후의 node의 개수가 절반보다 작다면 merge와 redistribute 같은 보조연산을 진행했다면, 이를 절반일때만 진행하는 것이 아니라, 해당 page의 key 가 다 사라졌을 때만 보조연산을 진행하도록 하였다. Internal Page 또한, Page 내의 key가 다 사라진다면 그 때 보조연산을 진행 하도록 구현 하였다.

기존에 있던 delete 함수인 delete_entry에 delete가 진행되는 page가 leaf 인지 internal인지를 구분하는 인자인 int delay를 추가한다.

```
void delete_entry(int64_t key, off_t deloff, int delay);
```

terminal 에서 d 키를 눌러 삭제를 진행할 경우, db_delete함수 내의 delete_entry의 인자 중 delay 값을 0으로 만들어 이때는 키의 개수가 다 사라지기 전까지는 보조연산을 진행하지 않는다.

```
int db_delete(int64_t key) {

    if (rt->num_of_keys == 0) {
        //printf("root is empty\n");
        return -1;
    }

      .
      .
      .

    free(check);
    off_t deloff = find_leaf(key);
    delete_entry(key, deloff, 0);
```

```
    return 0;
}//fin
```

delete_entry 함수에서 우선 삭제하고자 하는 key를 찾아 삭제를 한 후, 삭제한 key가 있는 page를 불러와서 그 page의 key 개수인 num_of_keys의 값이 0인지 아닌지를 판단한다.

num_of_key의 값이 0이 아니라면 보조연산을 진행하지않고, return; 한다. 0 이라면 기존에 있던 보조연산을 진행한다.

```
void delete_entry(int64_t key, off_t deloff, int delay) {
    remove_entry_from_page(key, deloff);

    if (deloff == hp->rpo) {
        adjust_root(deloff);
        return;
    }
    page * not_enough = load_page(deloff);

    //in leaf page
    if(delay == 0)
    {
        if (not_enough->num_of_keys >= 1){
          free(not_enough);
          return;
        }
    }
    //in internal page
    else if(delay == 1)
    {
        //int check = not_enough->is_leaf ? cut(LEAF_MAX) : cut(INTERNAL_MAX);
        if (not_enough->num_of_keys >= 1){
          free(not_enough);
          return;
        }
    }
                .
                .
                .
return;
}
```

delay 값의 기준은 leaf page를 처리할 때와 internal page를 처리할 때를 구분한다. leaf page만을 처리할 경우, delay 값은 0으로 page 내의 key 개수만 확인하면 된다. delay 값이 1일 경우에는 internal page에서 보조연산을 진행해야한다는 의미고 과정은 leaf page와 동일하다.

key의 개수가 0이면 기존의 보조연산 진행한다. code는 기존과 동일하다.

```
void delete_entry(int64_t key, off_t deloff, int delay) {
                  .
                  .
                  .
    //key 값의 개수가 0 이라 보조연산 진행
      int neighbor_index, k_prime_index;
      off_t neighbor_offset, parent_offset;
      int64_t k_prime;
      parent_offset = not_enough->parent_page_offset;
      page * parent = load_page(parent_offset);

      if (parent->next_offset == deloff) {
```

```
            neighbor_index = -2;
            neighbor_offset = parent->b_f[0].p_offset;
            k_prime = parent->b_f[0].key;
            k_prime_index = 0;
        }
        else if(parent->b_f[0].p_offset == deloff) {
            neighbor_index = -1;
            neighbor_offset = parent->next_offset;
            k_prime_index = 0;
            k_prime = parent->b_f[0].key;
        }
        else {
            int i;

            for (i = 0; i <= parent->num_of_keys; i++)
                if (parent->b_f[i].p_offset == deloff) break;
            neighbor_index = i - 1;
            neighbor_offset = parent->b_f[i - 1].p_offset;
            k_prime_index = i;
            k_prime = parent->b_f[i].key;
        }

        page * neighbor = load_page(neighbor_offset);
        int max = not_enough->is_leaf ? LEAF_MAX : INTERNAL_MAX - 1;
        int why = neighbor->num_of_keys + not_enough->num_of_keys;
        if (why <= max) {
            free(not_enough);
            free(parent);
            free(neighbor);
            coalesce_pages(deloff, neighbor_index, neighbor_offset, parent_offset, k_prime);
        }
        else {
            free(not_enough);
            free(parent);
            free(neighbor);
            redistribute_pages(deloff, neighbor_index, neighbor_offset, parent_offset, k_prime, k_prime_index);

        }
    return;
}
```

coalesce_pages 함수에서 마지막 delete_entry를 선언하여 진행하는 부분에서의 delay 값은 internal page 내부에 관련된 것이므로, 1로 설정하여 delete_entry 함수를 실행한다.

```
//merge 함수
void coalesce_pages(off_t will_be_coal, int nbor_index, off_t nbor_off, off_t par_off, int64_t k_prime) {
    page *wbc, *nbor, *parent;
    off_t newp, wbf;

                .
                .
                .

    delete_entry(k_prime, par_off, 1);
    free(wbc);
    usetofree(wbf);
    free(nbor);
    free(parent);
    return;
}//fin
```

위와 같이 leaf page에서 num_of_key 즉 page 내의 key의 개수가 0이 될때만 보조연산을 진행하므로, 보조연산에 대한 overhead가 기존의 b+ tree보다 줄어든다.

Result

bpt.h 에 보조연산이 일어나는 횟수를 저장하는 변수인 int overhead를 선언하고 main.c 에서 이를 0 으로 초기화한다.

```
//bpt.h
int overhead;

// main.c
overhead = 0;
```

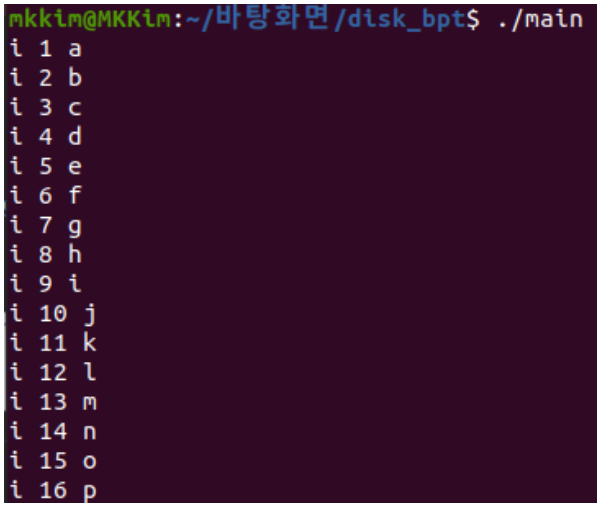
그리고 보조연산이 일어나는 부분에 overhead를 하나씩 더 하고, 그 횟수를 출력하도록 했다.

```
    // bpt.c
    if (why <= max) {
        free(not_enough);
        free(parent);
        free(neighbor);
        overhead += 1;
        printf("Overhead Count : %d\n", overhead);
        coalesce_pages(deloff, neighbor_index, neighbor_offset, parent_offset, k_prime);
    }
    else {
        free(not_enough);
        free(parent);
        free(neighbor);
        overhead += 1;
        printf("Overhead Count : %d\n", overhead);
        redistribute_pages(deloff, neighbor_index, neighbor_offset, parent_offset, k_prime, k_prime_index);
    }
```

이렇게 설정을 한 후, 보조연산을 더 자주 발생하게 하기 위해, LEAF_MAX 와 INTERNAL_MAX 의 값을 5 로 바꾸어 overhead 의 총합을 비교해본다.

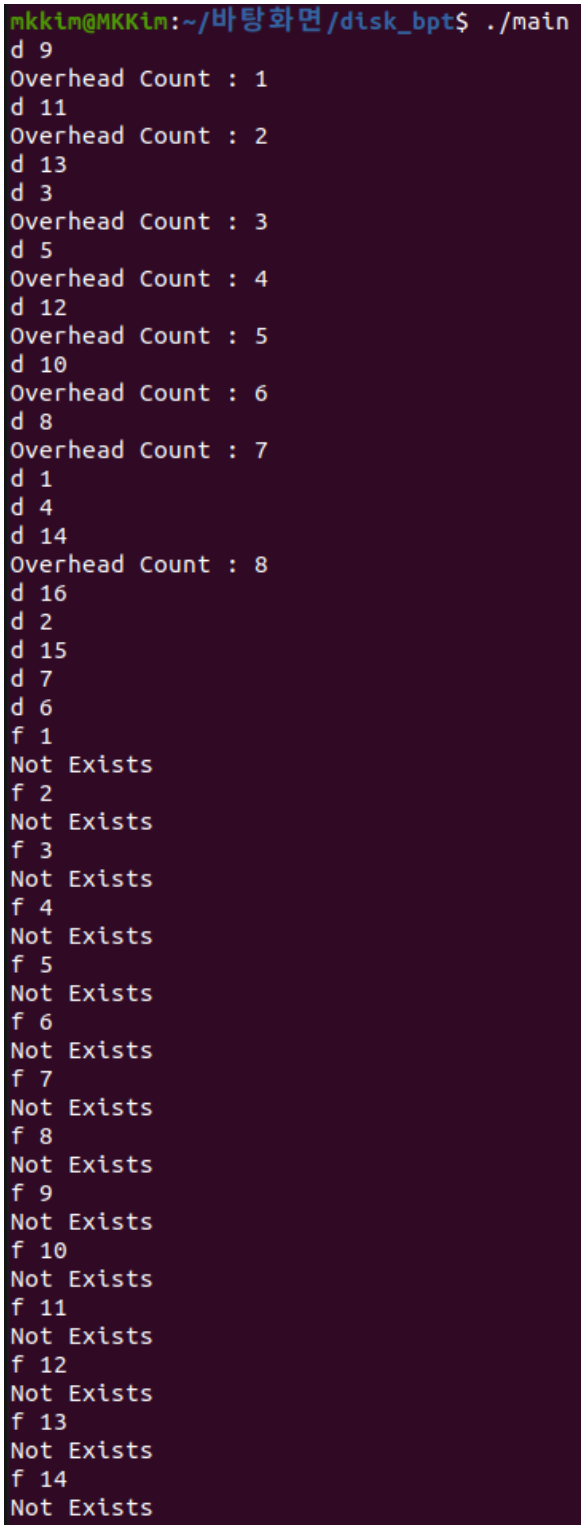
```
#define LEAF_MAX 5
#define INTERNAL_MAX 5
```

- test.db 에는 key가 1 ~ 16까지 내용은 알파벳의 순서대로 a ~ p 까지의 data가 들어 있다.



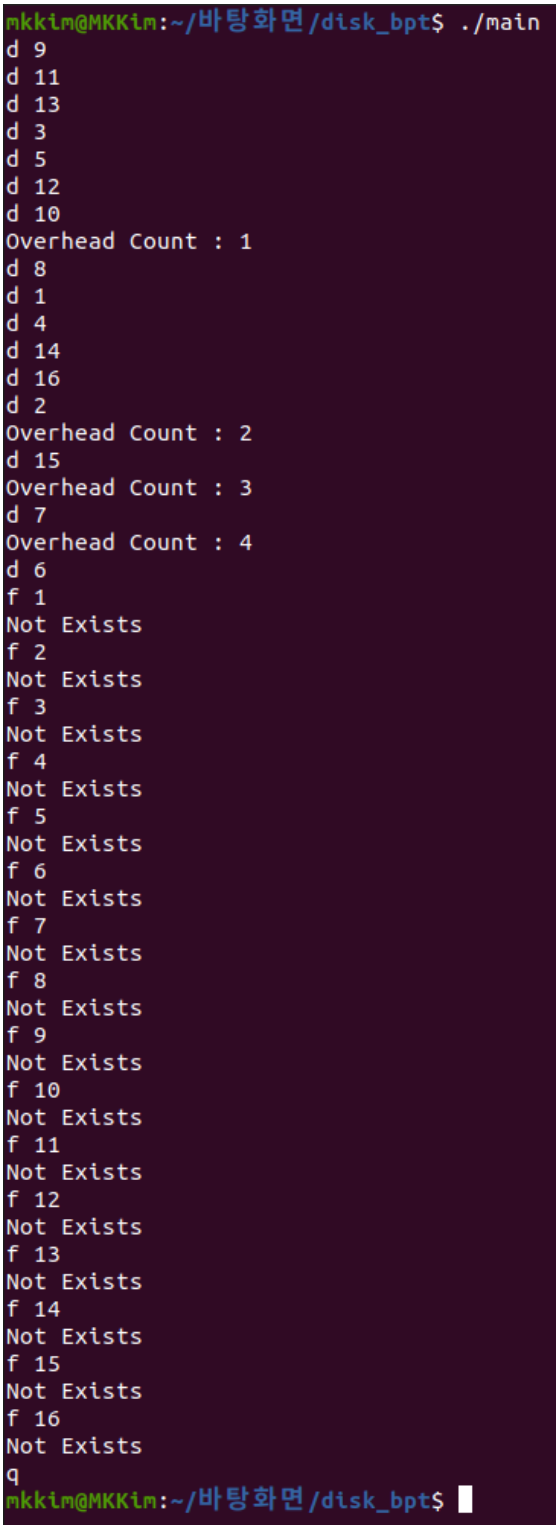
- 기존의 on-disk와 delay_merge를 추가한 on-disk에서 각각 key 를 다 삭제하면서 발생하는 overhead의 총합을 비교한다.

1. 기존의 on-disk



총 overhead의 숫자가 8이다.

2. delay-merge를 추가한 overhead의 총합



overhead의 총합이 4이다.

- Overhead의 총합이 기존의 on-disk는 **8**, delay-merge를 추가한 on-disk는 **4**로 2배 정도 적게 delay-merge 를 추가했을 때가 보조연산의 횟수가 적게 발생했다. 따라서 overhead의 양이 적어졌다.

Trouble Shooting

1. 기존의 code를 분석하여 처음에는 leaf page의 key가 다 사라졌을 때, 보조 연산을 진행했다. 하지만, 이렇게 된다면, internal page의 key 수가 다 사라지지 않으면 보조 연산을 진행하지 않았다.

   이를 해결하기 위해서, 공부해본 결과, leaf page의 key가 존재하지 않더라도, inernal page에는 삭제한 key가 남아 있어도 된다는 것을 알았다. 이를 활용하여, page 가 leaf 든 internal이든 간에 모두 그 안의 key의 개수가 1 보다 작아지면 보조연산을 진행하게 했다.

2. delete가 진행되는 부분이 leaf 인지 internal 인지를 구분하는것이 하나의 문제점이었다. 이를 구분하기 위해, int delay 인자를 parameter로 만들어서 0 이면 lead, 1이면 internal에서 delete가 진행되게 구현했다.