whw3

B911012 금예인

1. HASH_PRIME이 각각 7000, 6997, 120000, 11117, 22000, 22307일 때, collisions의 수는 각각 2305, 1821, 2733, 1235, 1359, 655였다.

```
clang-7 -pthread -lm -o main main.c
./main
CASE: HASH_PRIME = 7000
The number of collisions were 2305
The number of comparisons were 9356
```

```
clang-7 -pthread -lm -o main main.c
./main
CASE: HASH_PRIME = 6997
The number of collisions were 1821
The number of comparisons were 8115
```

```
clang-7 -pthread -lm -o main main.c
./main
CASE: HASH_PRIME = 12000
The number of collisions were 2733
The number of comparisons were 11633
```

```
clang-7 -pthread -lm -o main main.c
./main
CASE: HASH_PRIME = 11117
The number of collisions were 1235
The number of comparisons were 7216
```

```
clang-7 -pthread -lm -o main main.c
./main
CASE: HASH_PRIME = 22000
The number of collisions were 1359
The number of comparisons were 7550
```

```
clang-7 -pthread -lm -o main main.c
./main
CASE: HASH_PRIME = 22307
The number of collisions were 655
The number of comparisons were 6469
```

=> HASH_PRIME이 비슷한 값일 때, composite number일 때보다 prime number일 때 collisions의 수가 작았다. 또한, prime number인 경우인 M2, M4, M6를 비교했을 때는, HASH_PRIME이 클수록 collisions의 수가 작았다.

2. HASH_PRIME이 11117인 경우 comparisons의 수는 직접 컴퓨터 프로그램을 통해 계산해본 결과 7216이었다.

사용한 전체 코드는 다음과 같다.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define HASH_PRIME M (7000/6997/12000/11117/22000/22307)
#define POOL_SIZE 6000

int hash(char key[5])
{
    int i;
    long long x;
    x = 0;
    for (i = 0; i < 4; i++) {
        x = x + key[i];
        x = x << 8;
    }
    x = x + key[4];
    return x % HASH_PRIME;
}

struct record {
    char name[5];
    struct record* next;
};

struct record pool[POOL_SIZE];
struct record* top = pool;

int compare(char key[5], struct record*);

struct record* hash_table[HASH_PRIME];

void init_hash_table()
{
    int i;
```

```
    for (i = 0; i < HASH_PRIME; i++)
        hash_table[i] = NULL;
}

void init_pool()
{
    int i;
    struct record* r = pool;
    struct record* s;

    pool[POOL_SIZE - 1].next = NULL;

    for (i = 1; i < POOL_SIZE; i++) {
        s = r++;
        s->next = r;
    }
}

void init()
{
    init_pool();
    init_hash_table();
}

struct record* new_node()
{
    struct record* r;

    if (top == NULL)
        return NULL;

    r = top;
    top = r->next;
    return r;
}

int add(char* name)
{
    int hh = hash(name);
    struct record* r = hash_table[hh];
    struct record* n = new_node();
    int i = 0;
```

```
    if (n == NULL) {
        return 0;
    }

    if (r == NULL) {
        hash_table[hh] = n;
        n->next = NULL;
        for (i = 0; i < 5; i++) {
            n->name[i] = name[i];
        }
        return 0;
    }
    else {
        hash_table[hh] = n;
        n->next = r;
        for (i = 0; i < 5; i++) {
            n->name[i] = name[i];
        }
        return 1;
    }

}

int search(char name[3])
{
    struct record* r = hash_table[hash(name)];
    int i = 1;

    while (r != NULL) {
        if (compare(name, r) == 0) {
            return i;
        }
        else {
            r = r->next;
            i++;
        }
    }

    return 0;
}

int compare(char key[5], struct record* r)
```

```
{
    return strncmp(key, r->name, 5);
}

int main() {
    FILE* data = fopen("words.dat", "r");

    int i = 0;
    int j = 0;
    int k = 0;
    int sum = 0;
    int sum_c = 0;
    char key[300];
    char key_n[5757][5];

    init();

    for (i = 0; i < 4; i++) {
        fgets(key, 300, data);
        for (k = 0; k < 300; k++)
            key[k] = '\0';
    }

    for (i = 0; i < 5757;) {
        fgets(key, 300, data);
        for (j = 0; j < 5; j++)
            key_n[i][j] = key[j];
        for (k = 0; k < 300; k++)
            key[k] = '\0';
        sum += add(key_n[i]);
        i++;
    }

    fclose(data);

    for (i = 0; i < 5757;) {
        sum_c += search(key_n[i]);
        i++;
    }

    printf("CASE: HASH_PRIME = %d\n", HASH_PRIME);
    printf("The number of collisions were %d\n", sum);
    printf("The number of comparisons were %d\n", sum_c);
```

```
    return 0;
}
```