資料結構 HW_4

110303581 陳俊瑋

- **Mylib.h**

下面的部分從新加入的部分開始。

```
typedef struct MEMBER{
    int score;
    char member[MEMBER_SIZE];
    struct MEMBER *next;
} MEMBER;

typedef struct KeyofSet{
    char key[KEY_SIZE];
    MEMBER *members;    //指向一個 Linked List
} KeyofSet;


typedef struct DatabaseForSet { //有很多 key 的 database
    KeyofSet **sets;    //指標陣列
    int numSets;
} DatabaseForSet;
```

這邊先定義 `MEMBER` 的結構。一個 `MEMBER` 為一個 Linked List 的節點，之後每加入一個 member 都會串進這個鏈結。`KeyofSet` 為定義成一個 key 和一個指向 Linked List 的指標。之後再定義 `DatabaseForSet` 為一指標陣列和一個用來追蹤此指標陣列的 int `numSets`。

- **Mylib.c**

下面的部分從新加入的部分開始。

```
DatabaseForSet* createDatabaseForSet() {
    DatabaseForSet* db =
(DatabaseForSet*)malloc(sizeof(DatabaseForSet));
    db->numSets = 0;
    db->sets = NULL;
    return db;
}

MEMBER *createMember(const int score, const char *member){
    MEMBER *newNode = (MEMBER *)malloc(sizeof(MEMBER));
    if (newNode == NULL){
        perror("Memory allocation failed");
        exit(1);
    }
    newNode->score = score;
```

```
    strncpy(newNode->member, member, MEMBER_SIZE);
    newNode->next = NULL;
    return newNode;
}
```

`createDatabaseForSet` 是用來初始化資料庫，而 `createMember` 是用來創建一個 Linked List 的節點。

```
void addAMember(KeyofSet *set, const char *key, const int score, const
char *member) {
    // 找尋一樣的 member
    MEMBER *head = set->members;
    MEMBER *current = head;
    MEMBER *prev = head;
    while (current != NULL && (strcmp(current->member, member) !=
0)){    //到底部 or 找到一樣的 member 跳出
        prev = current;
        current = current->next;
    }

    if(current == NULL){    // 如果找不到 member
        printf("current=NULL\n");
        current = head;
        prev = NULL;  // 將 prev 設置為 NULL
        while (current != NULL){
            if(current->score > score){
                MEMBER *newMember = createMember(score, member);
                if (prev == NULL) {
                    // 插入在開頭
                    newMember->next = set->members;
                    set->members = newMember;
                }
                else {
                    // 插入在中間
                    prev->next = newMember;
                    newMember->next = current;
                }
                break;
            }
            prev = current;
            current = current->next;
        }

        if(current == NULL){    //加在最後
            MEMBER *newMember = createMember(score, member);
            prev->next = newMember;
```

```
        }
        printf("[%s] add in [%s] Successfully.\n", member, key);
        return;
    }

    //找到對應的 member
    if(current->score == score){    //score 一樣
        printf("[%s] has been add in [%s] before. Scores are the
same.\n", member, key);
        return;
    }
    else{
        //刪除 member 再重新加入
        if(prev == current){    //如果在頭
            set->members = current->next;
        }
        prev->next = current->next;
        free(current);
        addAMember(set, key, score, member);
        printf("Score of [%s] has been update to [%d].\n", member,
score);
    }
}
```

這函式是後面的 `ZADD` 函式提出來的，在 `ZADD` 從資料庫裡找到 key 之後，傳給該 key 的指標給 `addAMember`。`addAMember` 會先用 while 迴圈找尋有沒有一樣的 member，到底部 or 找到一樣的 member 跳出，之後再用 if else 判斷是到底部了還是找到一樣的 member。

如果找不到 member，用來找尋的指標 `current` 會在底部。此時再將 `current` 重製到頭，進行 score 的比較，在進行 Linked List 的插入。

如果找到有一樣的 member，先看 score 有沒有一樣，一樣則不改變，不一樣則刪除該 member，然後用新的 score 再重做一次 `addAMember`。

ZADD:

```
void ZADD(DatabaseForSet* db, const char *key, const int score, const
char *member){
    for (int i = 0; i < db->numSets; i++) {
        if (strcmp(db->sets[i]->key, key) == 0) {   //找到 Key
            addAMember(db->sets[i], key, score, member);
            return;
        }
    }
```

```
    printf("Cannot find key\n");
    // 找不到 key 生成一個 set
    db->sets = (KeyofSet **)realloc(db->sets, sizeof(KeyofSet *) *
(size_t)(db->numSets + 1));
    db->sets[db->numSets] = (KeyofSet *)malloc(sizeof(KeyofSet));
    strcpy(db->sets[db->numSets]->key, key);
    //初始化一個 set
    db->sets[db->numSets]->members = createMember(score, member);
    db->numSets++;
    printf("[%s] add in [%s] Successfully.\n", member, key);
    return;
}
```

此為 ZADD 函式，先找尋有沒有該 key，存在該 key 則進入 `addAMember`，該 key 不存在的話就生成一個 set，並增加傳入的節點。

輸出結果:

```
KEY:(get/set/update/del)
LIST:(lpush/rpush/lpop/rpop/llen/lrange)
SET:(zadd/zcard/zcount/zinterstore/zunionstore/zrange/zrangebyscore/zra
nk/zrem/zremrangebylex/zremrangebyrank/zremrangebyscore)
EXIT:0
Enter a command: zadd myzset 1 one 2 two
[one] add in [myzset] Successfully.
[two] add in [myzset] Successfully.
Enter a command:
```

ZCARD:

```
void ZCARD(DatabaseForSet *db, const char *key){
    for (int i = 0; i < db->numSets; i++) {
        if (strcmp(db->sets[i]->key, key) == 0) {   //找到 Key
            int count = 0;
            MEMBER *set_pointer = db->sets[i]->members; //head
            while(set_pointer != NULL){
                count++;
                set_pointer = set_pointer->next;
            }
            printf("[%s] has [%d] member.\n", key, count);
            return;
        }
    }
    //找不到 key
    printf("[%s] does not exist.\n", key);
}
```

此為 ZCARD 函式，先找尋該 key，然後把 member 全部走訪一遍，並且用 `count` 來計算 member 的數量。

輸出結果:

```
Enter a command: zadd myzset 1 one 2 two
[one] add in [myzset] Successfully.
[two] add in [myzset] Successfully.
Enter a command: zcard myzset
[myzset] has [2] member.
Enter a command:
```

```
int getZCARD(DatabaseForSet *db, const char *key){……
```

此函數只是為回傳 member 數量的 ZCARD，與上面的 ZCARD 幾乎相同。

ZCOUNT:

```
void ZCOUNT(DatabaseForSet* db, const char *key, const int min, const
int max){
    for (int i = 0; i < db->numSets; i++) {
        if (strcmp(db->sets[i]->key, key) == 0) {   //找到 Key
            int count = 0;
            MEMBER *pointer = db->sets[i]->members; //head
            while(pointer != NULL){
                if((pointer->score >= min) && (pointer->score <=
max)){  //在區間內才做累加
                    count++;
                }
                pointer = pointer->next;
            }
            printf("[%s] has [%d] member between [%d] and [%d].\n", key,
count, min, max);
            return;
        }
    }
    //找不到 key
    printf("[%s] does not exist.\n", key);
}
```

此函數一樣先找尋對應的 key，之後走訪整個 member，但 `count` 只在 `(pointer->score >= min) && (pointer->score <= max)`內做增加，也就是輸入的 min 和 max。

輸出結果:

```
Enter a command: zadd myzset 1 one 2 two 3 three
[one] add in [myzset] Successfully.
[two] add in [myzset] Successfully.
[three] add in [myzset] Successfully.
Enter a command: zcount myzset 1 2
[myzset] has [2] member between [1] and [2].
Enter a command:
```

ZINTERSTORE:

```
void ZINTERSTORE(DatabaseForSet* db, const char *newkey, const char
*key1, const char *key2){
    MEMBER *set1_pointer = NULL;
    MEMBER *set2_pointer = NULL;
    MEMBER *set2_head = NULL;
    for (int i = 0; i < db->numSets; i++) {
        if (strcmp(db->sets[i]->key, key1) == 0) {   //找到 Key1
            set1_pointer = db->sets[i]->members;
        }
        if (strcmp(db->sets[i]->key, key2) == 0) {   //找到 Key1
            set2_pointer = set2_head = db->sets[i]->members;
        }
    }
    while(set1_pointer != NULL){
        char *member1 = set1_pointer->member;   //member1 為 member in
key1
        int score1 = set1_pointer->score;
        int new_score;
        while (set2_pointer != NULL){
            char *member2 = set2_pointer->member;
            int score2 = set2_pointer->score;
            if(strcmp(member1, member2) == 0){  //有一樣的 member
                new_score = score1 + score2;
                ZADD(db, newkey, new_score, member1);
                break;
            }
            set2_pointer = set2_pointer->next;
        }
        set2_pointer = set2_head;
        set1_pointer = set1_pointer->next;
    }
}
```

此函數會將兩個 set 取交集，然後匯出一個新的 set。一樣是先找尋兩個 set 的位置，之後進入雙 while 迴圈，先以一個 set1 的 member 作為基準，走訪

set2 的每個 member 找尋一樣的 member，找到一樣則將其 score 相加，用 ZADD 加入或創建新 set，然後用下一個 set1 的 member 當基準。

輸出結果:

```
Enter a command: zadd set1 1 one 2 two 3 three
[one] add in [set1] Successfully.
[two] add in [set1] Successfully.
[three] add in [set1] Successfully.
Enter a command: zadd set2 1 one 2 two
[one] add in [set2] Successfully.
[two] add in [set2] Successfully.
Enter a command: zinterstore newset set1 set2
[one] add in [newset] Successfully.
[two] add in [newset] Successfully.
Enter a command: zrange newset 0 -1
1) one:2
2) two:4
```

ZUNIONSTORE:

```
void ZUNIONSTORE(DatabaseForSet* db, const char *newkey, const char
*key1, const char *key2){
    MEMBER *set1_pointer = NULL;
    MEMBER *set2_pointer = NULL;
    MEMBER *set1_head = NULL;
    MEMBER *set2_head = NULL;
    for (int i = 0; i < db->numSets; i++) {
        if (strcmp(db->sets[i]->key, key1) == 0) {   //找到 Key1
            set1_pointer = set1_head = db->sets[i]->members;
        }
        if (strcmp(db->sets[i]->key, key2) == 0) {   //找到 Key2
            set2_pointer = set2_head = db->sets[i]->members;
        }
    }
    while(set1_pointer != NULL){
        // 重新設置指標
        set2_pointer = set2_head;
        char *member1 = set1_pointer->member;   //member1 為 member in
key1
        int score1 = set1_pointer->score;
        int new_score;
        while (set2_pointer != NULL){
            char *member2 = set2_pointer->member;   //member2 為 member
in key2
            int score2 = set2_pointer->score;
```

```
            if(strcmp(member1, member2) == 0){  //有一樣的 member
                new_score = score1 + score2;
                ZADD(db, newkey, new_score, member1);
                break;
            }
            set2_pointer = set2_pointer->next;
        }
        if(set2_pointer == NULL){
            ZADD(db, newkey, score1, member1);  //member1 有 member2 沒有
的
        }

        set1_pointer = set1_pointer->next;
    }
    // 對第二個 set 進行迭代，唯有和第一個 set 不一樣時再加到聯集中
    set1_pointer = set1_head;
    set2_pointer = set2_head;

    while(set2_pointer != NULL){
        char *member2 = set2_pointer->member;   //member2 為 member in
key2
        while (set1_pointer != NULL){
            char *member1 = set1_pointer->member;   //member1 為 member
in key1
            if(strcmp(member1, member2) == 0){
                break;
            }
            set1_pointer = set1_pointer->next;
        }
        if(set1_pointer == NULL){   //找完第一個 set 發現都沒有
            ZADD(db, newkey, set2_pointer->score, member2);
        }
        // 重新設置指標
        set1_pointer = set1_head;

        set2_pointer = set2_pointer->next;
    }
}
```

此函數會將兩個 set 取聯集，然後匯出一個新的 set。一樣是先找尋兩個 set 的位置，然後一樣先以 set1 的 member 為基準，進行和 set2 的每個 member 比對。有一樣的 member 其 score 要相加，然後用 ZADD 加入新的 set，不一樣的也一樣加入新的 set。

最後在以 set2 的 member 為基準，走訪整個 set1，但有發現不一樣的 member 再加入新的 set 中。

輸出結果:

```
Enter a command: zadd set1 1 one 2 two 3 three
[one] add in [set1] Successfully.
[two] add in [set1] Successfully.
[three] add in [set1] Successfully.
Enter a command: zadd set2 1 one 2 two
[one] add in [set2] Successfully.
[two] add in [set2] Successfully.
Enter a command: zunionstore newset set1 set2
[one] add in [newset] Successfully.
[two] add in [newset] Successfully.
[three] add in [newset] Successfully.
Enter a command: zrange newset 0 -1
1) one:2
2) three:3
3) two:4
```

ZRANGE:

```
void ZRANGE(DatabaseForSet* db, const char *key, int start, int stop){
    MEMBER *set_pointer;
    MEMBER *set_head;
    int num = 0;
    for (int i = 0; i < db->numSets; i++) {
        if (strcmp(db->sets[i]->key, key) == 0) {   //找到 Key1
            set_pointer = set_head = db->sets[i]->members;
            while (set_pointer != NULL){
                // 算長度
                num++;
                set_pointer = set_pointer->next;
            }
        }
    }
    // 重新設置指標
    set_pointer = set_head;
    if(num == 0) {
        printf("[%s] does not exist.\n", key);
        return;
    }
    else{
        if(stop < 0){
            stop = num + stop; //把 stop 改成到哪裡開始
        }

        int counter = 0;
        while (set_pointer != NULL){
            if(counter >= start && counter <= stop){
```

```
                printf("%d) %s:%d\n", counter+1, set_pointer->member,
set_pointer->score);
            }
            counter++;
            set_pointer = set_pointer->next;
        }
    }
}
```

此函數也是一樣先找到對應的 key，由於要考慮 max 是負數的情況，要換算成正的要先算總 member 數。換算完之後再用 `counter` 去做累加，到了指定的區間後再做印出。

輸出結果:

```
Enter a command: zadd myzset 1 one 2 two 3 three 4 four 5 five
[one] add in [myzset] Successfully.
[two] add in [myzset] Successfully.
[three] add in [myzset] Successfully.
[four] add in [myzset] Successfully.
[five] add in [myzset] Successfully.
Enter a command: zrange myzset 0 -1
1) one:1
2) two:2
3) three:3
4) four:4
5) five:5
Enter a command: zrange myzset 2 4
3) three:3
4) four:4
5) five:5
```

ZRANGEBYSCORE:

```
void ZRANGEBYSCORE(DatabaseForSet* db, const char *key, const int min,
const int max){
    MEMBER *set_pointer;
    int count = 0;
    for (int i = 0; i < db->numSets; i++) {
        if (strcmp(db->sets[i]->key, key) == 0) {   //找到 Key
            set_pointer = db->sets[i]->members;
            while (set_pointer != NULL){
                if((set_pointer->score >= min) && (set_pointer->score <=
max)){
                    printf("%d) %s:%d\n", count+1, set_pointer->member,
set_pointer->score);
                    count++;
```

```
                }
                set_pointer = set_pointer->next;
            }
        }
    }
}
```

此函數與上面的 ZRANGE 相似，一樣是先找到 key 之後，開始走訪 member。當 member 的 score 在區間時，印出該 member。

輸出結果:

```
Enter a command: zadd myzset 1 one 2 two 3 three 4 four 5 five
[one] add in [myzset] Successfully.
[two] add in [myzset] Successfully.
[three] add in [myzset] Successfully.
[four] add in [myzset] Successfully.
[five] add in [myzset] Successfully.
Enter a command: zrangebyscore myzset 2 3
1) two:2
2) three:3
Enter a command:
```

ZRANK:

```
void ZRANK(DatabaseForSet* db, const char *key, const char *member){
    MEMBER *set_pointer;
    int count = 0;
    for (int i = 0; i < db->numSets; i++) {
        if (strcmp(db->sets[i]->key, key) == 0) {   //找到 Key
            set_pointer = db->sets[i]->members;
            while (set_pointer != NULL){
                if(strcmp(set_pointer->member, member) == 0){
                    printf("The rank of [%s] in [%s] is [%d].\n",
member, key, count);
                    return;
                }
                count++;
                set_pointer = set_pointer->next;
            }
        }
    }
}
```

此函式一樣是先找到 key，之後進入 member 走訪，用 `count` 從 0 開始累加計算出 rank。到了指定的 rank 之後就印出。

```
輸出結果:
Enter a command: zadd myzset 1 one 2 two 3 three 4 four 5 five
[one] add in [myzset] Successfully.
[two] add in [myzset] Successfully.
[three] add in [myzset] Successfully.
[four] add in [myzset] Successfully.
[five] add in [myzset] Successfully.
Enter a command: zrank myzset three
The rank of [three] in [myzset] is [2].
```

ZREM:

```
void ZREM(DatabaseForSet* db, const char *key, const char *member) {
    for (int i = 0; i < db->numSets; i++) {
        if (strcmp(db->sets[i]->key, key) == 0) {   //找到 Key
            MEMBER *set_pointer = db->sets[i]->members;
            MEMBER *prev = NULL;

            while (set_pointer != NULL) {
                if (strcmp(set_pointer->member, member) == 0) {
                    if (prev == NULL) {     // 要刪除的節點在頭
                        db->sets[i]->members = set_pointer->next;
                    }
                    else {
                        prev->next = set_pointer->next;
                    }
                    printf("[%s] has been removed from [%s].\n", member,
key);
                    free(set_pointer);
                    // 如果整個 key 列表已經空了將整個 set 刪除
                    if (db->sets[i]->members == NULL) {
                        free(db->sets[i]);
                        // 將 key 從數組中刪除
                        for (int j = i; j < db->numSets - 1; j++) {
                            db->sets[j] = db->sets[j + 1];
                        }
                        db->numSets--;
                    }
                    return;
                }

                prev = set_pointer;
                set_pointer = set_pointer->next;
            }

            printf("[%s] not found in [%s].\n", member, key);
            return;
```

```
        }
    }
    printf("Cannot find the key\n");
}
```

此函數會刪除一個 member，一開始一樣先找到 key，走訪該 member，刪除的方法與一般 Linked List 相同。刪除完成之後如果 `db->sets[i]->members == NULL`，也就是指標指向空，該 set 已經空了，就將該 set 指標 free 掉。

輸出結果:

```
Enter a command: zadd myzset 1 one 2 two 3 three 4 four 5 five
[one] add in [myzset] Successfully.
[two] add in [myzset] Successfully.
[three] add in [myzset] Successfully.
[four] add in [myzset] Successfully.
[five] add in [myzset] Successfully.
Enter a command: zrem myzset four
[four] has been removed from [myzset].
Enter a command: zrange myzset 0 -1
1) one:1
2) two:2
3) three:3
4) five:5
```

ZREMRANGEBYSCORE:

```
void ZREMRANGEBYSCORE(DatabaseForSet* db, const char *key, const int
min, const int max){
    MEMBER *set_pointer;
    for (int i = 0; i < db->numSets; i++) {
        if (strcmp(db->sets[i]->key, key) == 0) {   //找到 Key
            set_pointer = db->sets[i]->members;
            while (set_pointer != NULL){
                char *member = set_pointer->member;
                if((set_pointer->score >= min) && (set_pointer->score <=
max)){
                    set_pointer = set_pointer->next;
                    ZREM(db, key, member);
                }
                else{
                    set_pointer = set_pointer->next;
                }
            }
        }
    }
}
```

此函數會刪除區間的 member，一開始一樣先是找到 key，走訪 member，當該 member 的 score 在區間時，呼叫 ZREM 函式把它移除。

輸出結果:

```
Enter a command: zadd myzset 1 one 2 two 3 three 4 four 5 five
[one] add in [myzset] Successfully.
[two] add in [myzset] Successfully.
[three] add in [myzset] Successfully.
[four] add in [myzset] Successfully.
[five] add in [myzset] Successfully.
Enter a command: zremrangebyscore myzset 2 4
[two] has been removed from [myzset].
[three] has been removed from [myzset].
[four] has been removed from [myzset].
Enter a command: zrange myzset 0 -1
1) one:1
2) five:5
```

我也有做 ZREMRANGEBYLEX 和 ZREMRANGEBYRANK，但由於教授後來說不用做，我就只放輸出結果，但大體上與 ZREMRANGEBYSCORE 類似，只是 ZREMRANGEBYLEX 多了”[”與”(”的判斷。

ZREMRANGEBYLEX:

輸出結果:

```
Enter a command: zadd myzset 1 one 2 two 3 three 4 four 5 five
[one] add in [myzset] Successfully.
[two] add in [myzset] Successfully.
[three] add in [myzset] Successfully.
[four] add in [myzset] Successfully.
[five] add in [myzset] Successfully.
Enter a command: zremrangebylex myzset (one (three
[two] has been removed from [myzset].
Enter a command: zrange myzset 0 -1
1) one:1
2) three:3
3) four:4
4) five:5
Enter a command: zadd myzset 1 one 2 two 3 three 4 four 5 five
[one] has been add in [myzset] before. Scores are the same.
[two] add in [myzset] Successfully.
[three] has been add in [myzset] before. Scores are the same.
[four] has been add in [myzset] before. Scores are the same.
```

```
[five] has been add in [myzset] before. Scores are the same.
Enter a command: zremrangebylex myzset [one (four
[one] has been removed from [myzset].
[two] has been removed from [myzset].
[three] has been removed from [myzset].
```

ZREMRANGEBYRANK

輸出結果:

```
Enter a command: zadd myzset 1 one 2 two 3 three 4 four 5 five
[one] add in [myzset] Successfully.
[two] add in [myzset] Successfully.
[three] add in [myzset] Successfully.
[four] add in [myzset] Successfully.
[five] add in [myzset] Successfully.
Enter a command: zremrangebyrank myzset 2 4
[three] has been removed from [myzset].
[four] has been removed from [myzset].
[five] has been removed from [myzset].
```