HW5

EE3B 110305504 黃聖倫

1. 程式架構

```
struct sset
{
    int index;
    char *key;
    struct sset *prev;
    struct sset *next;
};
typedef struct sset SSET;
typedef SSET *SSNODE;
```

圖一

定義在指定 key 下的 member，用雙向鏈結連接。

```
//define sorted set name
struct sorted
{
    char *sorted_name;
    SSNODE head;
    struct sorted *next;
    struct sorted *prev;
};
typedef struct sorted SORTED;
typedef SORTED *SNODE;
```

圖二

儲存不同 key 的 linked list。裡面包含 key 下的 member，儲存其 head。

```
SSNODE findsort(const SSNODE  head,char *key)
{
    SSNODE ptr=head;
    while(ptr!=NULL)
    {
        if(strcmp(ptr->key, key) == 0)
        {
            return ptr;
        }
        else
        {
            ptr=ptr->next;
        }
    }
    return NULL;
}
```

圖三

針對已知 key，找到已存入 member’s address。

```
SSNODE deletesort(const SSNODE head, char *key)
{
    SSNODE ptr2 = findsort(head, key);
    if (ptr2 == head)
    {
        SSNODE ptr = ptr2;
        if (ptr2->next != NULL)
        {
            ptr2->next->prev = NULL;
        }
        ptr2 = ptr2->next;
        free(ptr);
        return ptr2;
    }
    else if (ptr2 != NULL)
    {
        ptr2->prev->next = ptr2->next;
        if (ptr2->next != NULL)
        {
            ptr2->next->prev = ptr2->prev;
        }
        free(ptr2);
    }
    else
    {
        printf("Nothing can delete from linked list\n");
    }
    return head;
}
```

圖四

特過 findsort 找到對應位址，並將其刪除，再將 doubly linked list 相接。

```
SSNODE createsort(SSNODE head, int index, char *key)
{
    SSNODE ptr = head;
    // Check if the key already exists in the list, and if so, delete it
    SSNODE existingNode = findsort(head, key);
    if (existingNode != NULL)
    {
        existingNode = deletesort(head, key);
        ptr=existingNode;
        //return existingNode;
    }


    // Create a new node
    SSNODE newNode = (SSNODE)malloc(sizeof(SSET));
    newNode->index = index;
    newNode->key = strdup(key);
    newNode->prev = NULL;
    newNode->next = NULL;

    // If the list is empty or the new node should be the head
    if (ptr == NULL || index < ptr->index)
    {
        newNode->next = ptr;
        if (ptr != NULL)
        {
            ptr->prev = newNode;
```

圖五

將新輸入 member 和其 index，存入。因為如果重複附值，刪除舊的資訊。所以使用 findsort，如果找到再用 deletesort 刪除。之後再進行存儲。

```
SNODE findname(const SNODE head, const char *name)
{
    SNODE ptr = head;
    while (ptr != NULL)
    {
        if (strcmp(ptr->sorted_name, name) == 0)
        {
            return ptr;
        }
        else
        {
            ptr = ptr->next;
        }
    }
    return NULL;
}
```

圖六

通過指定的 key，找到存儲此 key 的位址，方便之後獲取其 head 位址。

```
SNODE deletename(const SNODE head, char *name)
{
    SNODE ptr2 = findname(head, name);

    if (ptr2 == head)
    {
        SNODE newHead = ptr2->next;
        if (newHead != NULL)
        {
            newHead->prev = NULL;
        }
        free(ptr2);
        return newHead;
    }
    else if (ptr2 != NULL)
    {
        if (ptr2->prev != NULL)
        {
            ptr2->prev->next = ptr2->next;
        }
        if (ptr2->next != NULL)
        {
            ptr2->next->prev = ptr2->prev;
        }
        free(ptr2);
    }
    else
    {
        printf("NOTHING TO DELETE\n");
    }
    return head;
}
```

圖七

通過 findname 找到指定 key 位址，再將其刪除，並連接剩餘 doubly linked list。

```
SNODE createname(SNODE head, char *name)
{
    SNODE ptr = head;

    if (ptr == NULL)
    {
        ptr = (SNODE)malloc(sizeof(SORTED));
        ptr->head = NULL;
        ptr->sorted_name = strdup(name);
        ptr->next = NULL;
        ptr->prev = NULL;
        return ptr;
    }
    else
    {
        while (ptr->next != NULL)
        {
            ptr = ptr->next;
        }

        SNODE ptr2;
        ptr2 = (SNODE)malloc(sizeof(SORTED));
        ptr2->head = NULL;
        ptr2->sorted_name = strdup(name);
        ptr2->next = ptr->next;
        if (ptr->next != NULL)
        {
            ptr->next->prev = ptr2;
        }
        ptr2->prev = ptr;
        ptr->next = ptr2;

        return head;
    }
}
```

圖八

將指定 key 存入 doubly linked list，並會傳 head，以利後續尋找。

```
SNODE ZADD(SNODE SHEAD_1)
{
    SNODE SHEAD=NULL,ptr;
    char name[20];
    scanf("%s", name);
    SHEAD=createname(SHEAD_1,name);
    ptr=findname(SHEAD,name);
    char input[100];
    fgets(input, sizeof(input), stdin); // 讀整行輸入
    size_t len=strlen(input);
    if(len>0 && input[len-1]=='\n'){
        input[len-1]='\0';
    }
    char* token = strtok(input, " "); // 以空格分隔
    int index;
    while (token != NULL) {
        //*token = '\0';
        //if(strcmp(cmd, "ZADD") == 0) //LPUSH(&head, &tail, key, token);
        index = atoi(token); // Convert token to integer
        token = strtok(NULL, " ");
        ptr->head=createsort(ptr->head, index, token);
        token = strtok(NULL, " ");
    }
    return SHEAD;
}
```

圖九

ZADD 可一次輸入多筆 MEMBER 和其 INDEX。通過對空格切割再呼叫 createsort，將資料存入指定 key。並回傳 key 所在的 head。

```
void ZCARD(const SNODE const head)
{
    SNODE ptr;
    SSNODE ptr2;
    char name[20];
    scanf("%s", name);
    ptr=findname(head,name);
    ptr2=ptr->head;
    int total=0;
    while (ptr2!=NULL)
    {
        total+=1;
        ptr2=ptr2->next;
    }
    printf("The number of elements in %s has %d elements.\n",name,total);
    return;
}
```

圖十

呼叫 findname 找指定 key 所在位址。再用其 head 去拜訪，得到儲存在此 head 的所有 member 數量。

```
void ZCOUNT(const SNODE const head)
{
    SNODE ptr;
    SSNODE ptr2;
    char name[20];
    int min=-1,max=-1;
    scanf("%s", name);
    ptr=createname(head,name);
    ptr=findname(head,name);
    scanf("%d",&min);
    scanf("%d",&max);
    if(min>max)
    {
        printf("Minimum is larger than Maximum\n");
        return ;
    }
    ptr=findname(head,name);
    ptr2=ptr->head;
    int total=0;
    while (ptr2!=NULL)
    {
        if(ptr2->index>=min && ptr2->index<=max)
        {
            total+=1;
            ptr2=ptr2->next;
        }
        else if(ptr2->index<min)
        {
            ptr2=ptr2->next;
        }
        else if(ptr2->index>max) break;
    }
    printf("The number of elements between %d and %d in %s has %d elements.\n",min,max,name,total);
    return;
}
```

圖十一

呼叫 findname 找指定 key 所在位址。再用其 head 去拜訪，得到儲存在 min 和 max 之間 head 的所有 member 數量。

```c
void intersection(SNODE head, SSNODE head1, SSNODE head2, int power1, int power2) {
    // Clear the original linked list
    if (head->head != NULL) {
        free(head->head);
        head->head = NULL;
    }

    // head1 is the set to be stored
    SSNODE returnhead = NULL;

    if (head1 == NULL || head2 == NULL) {
        return;
    } else {
        SSNODE ptr = head1;
        while (ptr != NULL) {
            SSNODE ptr2 = findsort(head2, ptr->key);
            if (ptr2 == NULL) {
                ptr = ptr->next;
                continue;
            } else {
                int power = (ptr->index) * power1 + (ptr2->index) * power2;
                returnhead = createsort(returnhead, power, ptr->key);
            }
            ptr = ptr->next;
        }

        // Update the head with the result
        head->head = returnhead;
        return;
    }
}
```

圖十二

將 head1 和 head2 所儲存一樣 key 的資料分別乘上 power1,power2，再將其存入 head 裡面。因為存入一訂有兩筆已存入的 key，一定不是 head，所以不需回傳。

```c
void ZINTERSTORE(SNODE head)
{
    int power1, power2;
    SSNODE ptr1, ptr2;
    SNODE headfake, ptrname, ptrname2;
    char name[20];
    char result_name[20];

    scanf("%s", result_name);
    scanf("%s", name);
    ptrname = findname(head, name);
    if (ptrname == NULL) {
        printf("Set '%s' not found\n", name);
        return;
    }
    scanf("%s", name);
    ptrname2 = findname(head, name);
    if (ptrname2 == NULL) {
        printf("Set '%s' not found\n", name);
        return;
    }
    ptr1 = ptrname->head;
    ptr2 = ptrname2->head;
    scanf("%d", &power1);
    scanf("%d", &power2);

    if ((headfake = findname(head, result_name)) != NULL) {
        intersection(headfake, ptr1, ptr2, power1, power2);
    } else {
        headfake = createname(head, result_name);
        headfake = findname(head, result_name);
        intersection(headfake, ptr1, ptr2, power1, power2);
    }
```

圖十三

使用 createname，創新的 key，再用 findname 找到其位置，之後呼叫 intersection，去進行 ZINTERSTORE 步驟。

```
void ZUNIONSTORE(const SNODE head) {
    int power1, power2;
    SSNODE ptr1, ptr2;
    SNODE headfake, ptrname, ptrname2;
    char name[20];
    char result_name[20];

    scanf("%s", result_name);
    scanf("%s", name);
    ptrname = findname(head, name);
    if (ptrname == NULL) {
        printf("Set '%s' not found\n", name);
        return;
    }
    scanf("%s", name);
    ptrname2 = findname(head, name);
    if (ptrname2 == NULL) {
        printf("Set '%s' not found\n", name);
        return;
    }
    ptr1 = ptrname->head;
    ptr2 = ptrname2->head;
    scanf("%d", &power1);
    scanf("%d", &power2);
```

圖十四

ZUNIONSTORE 複製和 ZINTERSTORE 一樣的程式碼。之後再分別對 PTR1,PTR2，去搜尋有無一樣資料已存入 ZINTERSTORE 所創的 KEY 裡面，如果沒有，則將其乘上 POWER1,POWER2，加入在此 doubly linked list 裡面。

```
void ZRANGE(SNODE head) {
    char name[20];
    scanf("%s",name);
    SNODE ptr=findname(head,name);
    SSNODE ptr1=ptr->head;
    if(ptr1==NULL)
    {
        printf("Can't find this key\n");
        return ;
    }
    int length = 0;
    int start,stop;
    int index=0;
    scanf("%d",&start);
    scanf("%d",&stop);
    // Calculate the length of the list
    while (ptr1 != NULL) {
        length++;
        ptr1 = ptr1->next;
    }
    if (start < 0) {
        start = length + start;
```

圖十五

通過相訪問指定 key 之 linked list，去判斷其 length。再重新尋找，當 member 的 index 大於 start 和小於 stop，則 index+1。如果 member 的 index 大於 stop，則 return ;結束程式，並印出結果。

```
void ZRANGEBYSCORE(SNODE head) {
    char name[20];
    scanf("%s",name);
    SNODE ptr=findname(head,name);
    SSNODE ptr1=ptr->head;
    if(ptr1==NULL)
    {
        printf("Can't find this key\n");
        return ;
    }
    int length = 0;
    int start,stop;
    scanf("%d",&start);
    scanf("%d",&stop);
    // Calculate the length of the list
    while (ptr1 != NULL) {// Print elements within the specified range
        if (ptr1->index >= start && ptr1->index <= stop) {
            printf("%s:%d ", ptr1->key, ptr1->index);
        }
        ptr1 = ptr1->next;
    }
    printf("\n");
}
```

圖十六

和 zrange 相似。但這次是用分數去比較而不是其位址(index)。

```
void ZRANK(SNODE head) {
    char name[20];
    scanf("%s", name);
    SNODE ptr = findname(head, name);
    if (ptr == NULL) {
        printf("Set '%s' not found\n", name);
        return;
    }
    scanf("%s", name);
    SSNODE ptr1 = ptr->head;
    int index = 0;
    while (ptr1 != NULL) {
        if (strcmp(ptr1->key, name) == 0) {
            printf("%d\n", index);
            return;
        } else {
            index++;
            ptr1 = ptr1->next;
        }
    }
    // If the key is not found in the set
    printf("Key '%s' not found in set '%s'\n", name, ptr->sorted_name);
}
```

圖十七

先用 findname 找到指定 key address。再通過訪問 key->head，找到其 member 的序號並印出 index 結束程式，如果不是 member，則 index+1。

```
void ZREM(SNODE head)
{
    char name[20];
    scanf("%s", name);
    SNODE ptr = findname(head, name);
    if (ptr == NULL) {
        printf("Set '%s' not found\n", name);
        return;
    }
    SSNODE ptr1=ptr->head;
    char input[100];
    fgets(input, sizeof(input), stdin); // 讀整行輸入
    size_t len=strlen(input);
    if(len>0 && input[len-1]=='\n'){
        input[len-1]='\0';
    }
    char* token = strtok(input, " "); // 以空格分隔
    while (token != NULL) {
        ptr1=deletesort(ptr1, token);
        token = strtok(NULL, " ");
    }
    ptr->head=ptr1;
}
```

圖十八

ZREM 通過 FINDNAME 找到要刪除 MEMBER 所在的 KEY 位址。在通過字串切割使用 deletesort 刪除需要刪除的 member。

```c
void ZREMRANGEBYSCORE(SNODE head)
{
    char name[20];
    scanf("%s", name);
    SNODE ptr = findname(head, name);
    if (ptr == NULL) {
        printf("Set '%s' not found\n", name);
        return;
    }
    SSNODE ptr1=ptr->head;
    SSNODE ptr2=ptr->head;
    int min,max;
    scanf("%d",&min);
    scanf("%d",&max);
    if(min>max)
    {
        printf("MIN LARGER THAN MAX\n");
        return ;
    }
    while(ptr1!=NULL)
    {
        if(ptr1->index>=min && ptr1->index<=max)
        {
            ptr2=deletesort(ptr2,ptr1->key);
            ptr1=ptr1->next;
        }
        else if(ptr1->index>max)
        {
            ptr->head=ptr2;
            return ;
        }
        else if(ptr1->index<min)
        {
            ptr1=ptr1->next;
        }
```

圖十九

將分數在 min 和 max 之間的 member 透過遍歷 linked list 去找尋，再將其刪除。如果當現在位址大於 max，則停止程式。

程式碼範例輸出

```
Three motion to choose
(LPUSH, RPUSH, LPOP, RPOP, LLEN, LRANGE)
(SET, GET, DEL, UPD)
(ZADD,ZCARD,ZCOUNT,ZINTERSTORE,ZUNIONSTORE,ZRANGE,ZRANGEBYSCORE,ZRANK,ZREM,ZREMRANGEBYSCORE)
Enter the operation:ZADD LMS 23 KV 30 SC 11 KT 77 LD 2 ASD 1 ADS
Enter the operation:ZADD WSL 1 ASD 2 ASDF 3 ADS 1 WA
Enter the operation:ZADD HAHA 2 AS 1 ASD 3 WA 4 AWD
Enter the operation:ZCARD LMS
The number of elements in LMS has 6 elements.
Enter the operation:ZCOUNT LMS 10 50
The number of elements between 10 and 50 in LMS has 3 elements.
Enter the operation:ZINTERSTORE WS WSL HAHA 1 2
Enter the operation:ZUNIONSTORE WP WSL HAHA 2 3
Enter the operation:ZRANGE WS 0 -1
ASD:3 WA:7
Enter the operation:ZRANGE WP 0 -1
ASDF:4 ASD:5 ADS:6 AS:6 WA:11 AWD:12
Enter the operation:ZRANGEBYSCORE LMS 5 10

Enter the operation:ZRANGEBYSCORE LMS 10 70
KT:11 KV:23 SC:30
Enter the operation:ZRANK WP 2
Key '2' not found in set 'WP'
Enter the operation:ZRANK WP AS
3
Enter the operation:ZREM WP AS ASD
Enter the operation:ZRANGE WP 0 -1
ASDF:4 ADS:6 WA:11 AWD:12
Enter the operation:ZREMRANGEBYSCORE LMS 10 35
Enter the operation:ZRANGE LMS 0 -1
ADS:1 ASD:2 LD:77
```

首先先輸入三個 ZADD key 給 LMS,WSL,HAHA。
使用 ZCARD LMS，可得到其中有 6 個 MEMBER，與輸入相同。
接著測試 ZCOUNT LMS，給定最小值和最大值去判斷，得到結果為 3(KV,SC,KT)。與預期相同。
之後先對 WSL HAHA 取交集(ZINTERSTORE)，再分別乘上倍數，KEY 為 WS。
再對 WSL HAHA 取聯集(ZUNIONSTORE)，再分別乘上倍數，KEY 為 WP。
接著分別使用 ZRANGE 取第一個到最後一個 MEMBER，並印出。
可發現 WS 為 WSL 1(ASD)*1,1(WA)*1，HAHA 1(ASD)*2,3(WA)*2，得到結果為 ASD(3),WA(7)與預期一樣。
可發現 WP 為 WSL 1(ASD)*2,2(ASDF)*2,3(ADS)*2,1(WA)*2
HAHA 2(AS)*3,1(ASD)*3,3(WA)*3,4(AWD)*3，
得到結果為 ASDF(4),ASD(5),ADS(6),AS(6),WA(11),AWD(12)與預期一樣。
再來測試 ZRANGEBYSCORE 因為沒有資料在這個區間，所以沒有印東西。
再測試一個可以的區間，有印出正確結果(KT:11,KV:23,SC:30)。
接著測試 ZRANK，輸入不存在的 MEMBER，沒找到。
再輸入 WP 的 AS。由前面 ZZRANGE，可知 WP INDEX=0 為 ASDF，所以 AS 為 3。與預期一樣。
接著再測試 ZREM 刪除其中 WP 的 AS 和 ASD。

在印出後可發現 LINKED LIST 裡已無此兩個 MEMBER。
最後測試 ZREMRANGEBYSCORE，給定 KEY=LMS，和分數區間。
之後用 ZRANGE 印出所有可發現符合其中得(KT:11,KV:23,SC:30)已被刪除，因此與預期符合。
以上為我的程式輸出範例。