【HW-08】Term Counting by Hash Map

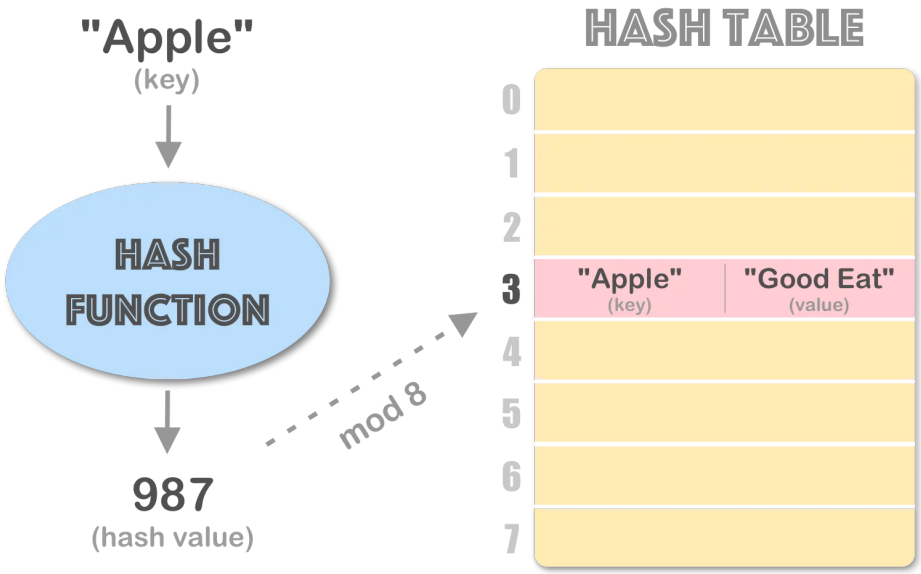
解題紀錄

題目敘述

簡介 (Intro)

雜湊表 (Hash Table) 或稱為 雜湊映射 (Hash Map)，
是一種維護 鍵值對 (Key-Value Pair) 集合的資料結構。

其透過雜湊函式 (Hash Function) ：
將「鍵 (key)」轉換為一個整數 — **雜湊值 (Hash Value)**，
再將雜湊值轉換為陣列的索引 (因表格列數可能小於雜湊值)，
最後，便能將此 **條目 (Entry)** (i.e., 鍵與值) 置於該「桶子 (bucket)」中。

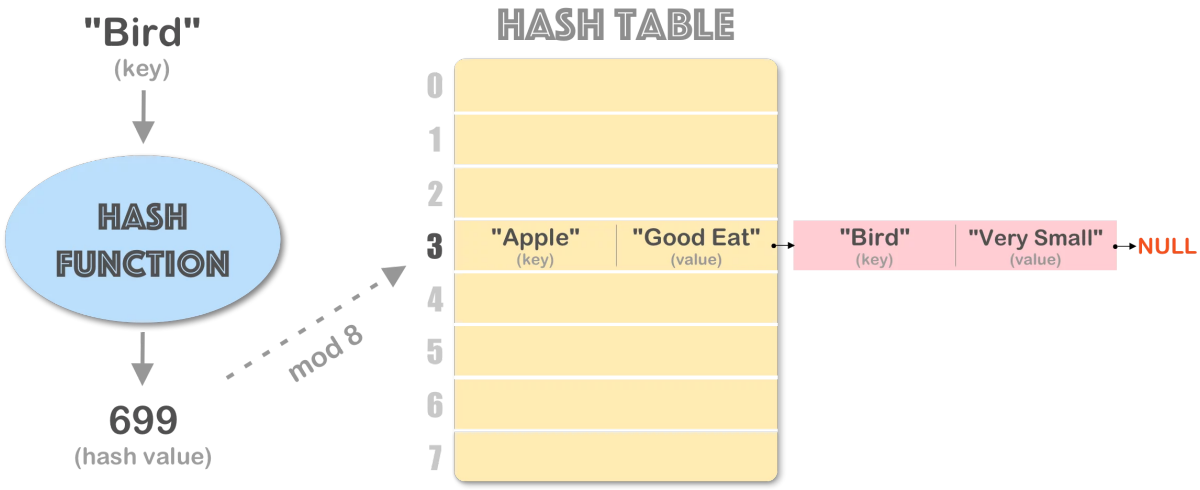


碰撞 (Collision)

然而，當新增多筆資料時，
桶子 (bucket) 可能已經放置過其他條目了，
此種情形稱為 **碰撞 (collision)**。

一種解決碰撞的辦法是：「將條目置於其他空的桶子中」，
這稱作 **開放定址 (Open Addressing)**。

另一種解法則是：「將桶子視為鏈結串列的頭節點」，
即：「將相同索引之條目串在一起」，
這稱作 **獨立鏈結 (Separate Chaining)**。



此次作業請統一練習「**獨立鏈結法**」。

順序 (Order)

Hash Table 不同於 串列 (List) 與 二元搜尋樹 (BST)，
除非使用額外的結構維護或特別設計過的雜湊函式，
其插入**無法**維護條目的**順序性**。

因此，欲取得有序的元素串列，
需分配一個夠大的空間 (e.g., 陣列)，
並搜集所有「已插入的條目」於其中，
最後對其進行排序即可。

[註]：
若碰撞處理使用**開放定址**時，也應採取類似做法，
而非直接對整張表格排序，否則將破壞雜湊性質。

作業 (Homework)

利用 HashMap (HashTable) 統計詞頻，輸入方式與作業六相同。

為使同學聚焦於 **HashMap** 之實作，而無需處理程式流程，
已幫同學們完成 main 函式之撰寫。
(i.e., 輸入/輸出、HashMap 之使用、條目的搜集與排序呼叫)

各位僅需複製以下程式模板，並將**未完成**的函式宣告進行**定義**即可：

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

#define BUF_SIZE 1024
#define MAP_CAP_BITS 5

/* ============ 型別定義 ============ */
/* Map Entry (i.e., Key-Value Pair) */
typedef struct entry {
    char *key;
    int value;

    /*
     * 因碰撞解決方案採：Separate Chaining，
     * 需維護指向下一筆 Entry 之指標
     */
    struct entry *next;
} Entry;


/* Hash Map (i.e., Hash Table) */
typedef struct {
    /*
     * 指向「所有條目」的指標 (i.e., 桶子們)，
     * 以抽象化的角度審視，也可將其命名為 "table"，
     * 意指雜湊表的「表格」實體
     */
```

```
29      Entry *buckets;
30
31      size_t capacity; // 桶子數 (i.e., 表格列數)
32      size_t size; // 已插入的條目數 (初始為 0)
33  } HashMap;
34
35
36  /* ============ 已完成 ============= */
37  /**
38   * map_init - 初始化 HashMap (i.e., 分配雜湊表空間、設置相關屬性
    至初始值)
39   * @map: 欲初始化之 HashMap
40   * @cap_bits: 欲初始化之雜湊表大小 (log2)
41   * @return 0 表示成功, -1 表示失敗
42   */
43  int map_init(HashMap *map, unsigned int cap_bits);
44
45  /**
46   * map_entries - 取得 HashMap 中所有「已插入」的條目
47   * @map: 欲操作之 HashMap
48   * @entries: 「指向 (Entry *) 之指標」的位址
49   * @return 0 表示成功, -1 表示失敗
50   */
51  int map_entries(HashMap *map, Entry ***entries);
52
53
54  /* ============ 未完成 ============= */
55  /**
56   * entry_cmp - 用於 qsort 的比較函式
57   *
58   *             hint:
59   *             就像欲排序 int *arr 時， a, b 需轉型為 (int *)，
60   *             方能解參考 (dereference) 取出其整數值。
61   *
62   *             現在，排序目標之型別： Entry **entries
63   *             請思考 a, b 該如何正確轉型
64   *
65   */
66  int entry_cmp(const void *a, const void *b);
67
68  /**
69   * my_hash_function - 設計一個雜湊函式
70   * @key: 雜湊函式的輸入鍵
71   * @return 雜湊值 (hash value)
72   */
73  size_t my_hash_function(const char *key);
74
75  /**
```

```
 * map_idx - 根據 key 計算雜湊值，並回傳用於雜湊表的索引值 (Index)，
 *           (i.e., 第 i 個桶子 = 表格的第 i 列)
 *
 *           hint:
 *           內部應呼叫 my_hash_function
 *
 * @map: 欲操作之 HashMap
 * @key: 鍵
 * @return 根據 `key` 所計算出的雜湊表索引 (Index)
 */
size_t map_idx(HashMap *map, const char *key);

/**
 * map_put - 新增條目 <key, value> 至 HashMap 中
 *
 *           hint：
 *           使用 `map_idx` 計算索引值，以找到插入的目標桶 (bucket)：
 *
 *               case 1: 桶子為空，將 strdup(key) 及 value 置於桶中。
 *
 *               case 2: 桶子非空，但該桶子或其鏈結串列上存在
 *                       與「欲插入之 `key`」相同的「鍵」，
 *                       則以「欲插入之 `value`」取代舊值。
 *                       (注意：此 case 中並無 (也不應該) 呼叫 strdup)
 *
 *               case 3: 桶子非空，但該桶子或其鏈結串列上皆「不」存在
 *                       與「欲插入之 `key`」相同的「鍵」，
 *                       則分配 (i.e., malloc) 一個 Entry，
 *                       並將 strdup(key) 及 value 置於其中，
 *                       最後將此 Entry 插入至鏈結串列 (頭或尾隨意)。
 *
 * @map: 欲操作之 HashMap
 * @key: 鍵
 * @value: 值
 * @return 0 表示成功, -1 表示失敗
 */
int map_put(HashMap *map, const char *key, int value);

/**
 * map_get - 根據 `key`，取得已插入至 HashMap 的對應「值」之「位址」
 *
```

```
 *           hint:
 *           參照 `map_put` 方法進行搜尋
 *
 * @map: 欲操作之 HashMap
 * @key: 鍵
 * @return 對應「值」之「位址」，若不存在該條目則回傳 NULL
 */
int *map_get(HashMap *map, const char *key);

/**
 * map_destroy - 釋放 HashMap 所有相關資源，包含：
 *               @ 接在桶子後方的鏈結串列之節點
 *               @ 每個條目的 key
 *               @ 整張雜湊表
 *
 * @map: 欲操作之 HashMap
 */
void map_destroy(HashMap *map);

int main() {
    /* ================== INITIALIZE ==================
    */
    HashMap map;
    if (map_init(&map, MAP_CAP_BITS) < 0)
        fprintf(stderr, "map_init error\n");

    /* ================== INPUT TERMS ==================
    */
    char buf[BUF_SIZE];
    while (fgets(buf, BUF_SIZE, stdin)) {
        /* 去除輸入後方「可能」的換行字元 */
        buf[strcspn(buf, "\r\n")] = '\0';

        /* 目前是否欲對 term 作遞減 */
        const _Bool decrease = (*buf == '-');

        char *term;
        int increment; // 增量值

        if (decrease) {
            term = buf + 1;
            increment = -1;
        } else {
            term = buf;
            increment = 1;
        }

        /* 根據 term 取得對應的值「位址」 */
```

```
163         int *value = map_get(&map, term);
164 
165         if (value)
166             (*value) += increment; // 根據增量值，修改 term
    count
167         else if (!decrease) // 若未插入過該 term，且不為遞減模式
168             map_put(&map, term, 1);
169     }
170 
171     /* =============== OUTPUT TERM COUNT ================
    */
172     {
173         const size_t size = map.size;
174         Entry **entries;
175         if (map_entries(&map, &entries) < 0) // 取得所有條目
176             fprintf(stderr, "map_entries error\n");
177 
178         /* 根據 entry_cmp 排序*/
179         qsort(entries, size, sizeof(Entry *), entry_cmp);
180 
181         /* 按照排序結果依序輸出*/
182         for (size_t i = 0; i < size; i++) {
183             Entry *e = entries[i];
184             const char *term = e->key;
185             const int count = e->value;
186             printf("%d %s\n", count, term);
187         }
188 
189         /* 釋放條目指標空間 */
190         free(entries);
191     }
192 
193     /* ================== TERMINATE ====================
    */
194     map_destroy(&map);
195 }
196 
197 int map_init(HashMap *map, unsigned int cap_bits) {
198     const size_t capacity = 1u << cap_bits; // 計算 2 的
    cap_bts 次方
199     map->buckets = calloc(capacity, sizeof(Entry)); // 分配
    「全為 0」的空間
200     map->capacity = capacity;
201     map->size = 0;
202 
203     return -(map->buckets == NULL);
204 }
205 
```

```c
int map_entries(HashMap *map, Entry ***entries) {
    Entry *table = map->buckets;
    const size_t capacity = map->capacity;
    size_t size = map->size;

    /* 若目前雜湊表無任何元素，則回傳 0 */
    if (!size) {
        *entries = NULL;
        return 0;
    }

    /* 否則，根據目前元素數量分配空間 */
    Entry **ret = malloc(size * sizeof(Entry *));
    if (!ret)
        return -1; // 分配失敗則傳回 -1

    size_t j = 0;
    for (size_t i = 0; size && i < capacity; i++) {
        Entry *e = table + i;
        if (!e->key)
            continue; // 若目前 bucket 為空，則查找下一個 bucket

        while (e) {
            ret[j++] = e; // 添加目前的 entry
            size--;
            e = e->next; // 查找鏈結串列節點之 entry
        }
    }

    *entries = ret;
    return 0;
}
```

輸入說明

每行一個單詞，長度不超過 1024，
開頭若有「減號」則該詞次數減一

輸出說明

輸出說明

根據單詞出現次數由大到小輸出，印出其出現次數及單詞 (參照作業六)

輸入/輸出範例 1

執行參數

無

輸入

```
apple
banana
apple
apple
banana
cake
cake
-apple
cake

```

輸出

```
3·cake
2·apple
2·banana

```

作答限制

基礎限制

執行時間上限	記憶體上限	開檔上限
1 秒	32 MB	0 個
指標	陣列	全域變數
✓	✓	✓

作業已過繳交期限，查看解題紀錄