資料工程 HW3

404410030 資工四 鄭光宇

系統需求

要執行這支程式，系統必須具備：

- gcc
- make 工具程式

編譯

```
make
```

執行

```
tcount filename [-m memory_limit (in MB) | -s key_size (in byte) | -h
hash_table_size (in MB) | -v virtual_hash_table_size (in MB) ]
```

參數：

-m: 最大記憶體限制（key buffer + hash table） -s: key 的字串長度限制 -h: hash table 的記憶體限制 -v: 虛擬 hash table 大小（記憶體+硬碟容量）

資料

使用 ettoday1~ettoday6 並且以：

，。；：「」等標點符號斷句。（產生較多種類的 key） 中文字開頭，至少6個中文字以上為一句。產生共 `572MB` 的文字擋，共1612萬行句子。

```
peter@peter-ubuntu:~/Data-Engineering/HW3$ time ./parser

real    0m43.413s
user    0m13.356s
sys     0m28.379s
```

```
倒也還蠻有趣的
總要有點創新吧
懷著真誠的心與網友互動
樂在分享　愛上雲端
這是個細雨飛的早晨
總裁王令麟及董事長馬詠睿站在公司門口迎接這群新員工
他一一握手完畢後笑說
這是王令麟重建媒體事業的第一步
他也訓勉大家在一起工作要開心
因為唯有開心的員工才能將歡樂帶給讀者
讀者才會喜歡ETtoady
試圖跳出傳統新聞的窠臼
誰說新聞只有記者可以寫
誰說讀者只能等待新聞被印出來
現在網路這麼發達
每個人都可以發聲
每個人都可能是新聞資訊的來源
在浩瀚的網路世界中尋找並分享好看的新聞
按讚即是參與
看完新聞按讚並留言
如果要發展更進階的參與則是幫忙尋找相關資料
每則推薦底下的長串留言中
常看到更多的相關新聞及線索
```

sentences.txt

實作

hash function

使用開放定址法，處理衝突時使用 double hashing。 hash function 為：

```
unsigned long get_hash_value(const wchar_t *str, unsigned long n) {
    unsigned long ans=0uL;
    while(n--) {
        ans = ((ans*hash_base)%hash_mod + (unsigned long)(*str)) %
hash_mod;
        ++str;
    }
    return ans+1; // shift 1
}
```

其中，hash_base(311), hash_mode 均為質數，方便後面處理衝突情形。 並且將輸出 hash 值加 1，保留 0 的位址為一個空位址，從 1 以後的位址才可以讀寫。

更新與衝突處理

更新 hash table 時，使用 double hashing 處理衝突。為了方便後續查詢，會將 double hashing 探測到的 node 位址，從原來 hash function 對應到的 node 指向到這個探測到的 node。

由於 double hashing 的特性，相對於線性探測，會產生較少的群集現象。

```
void insert_hashtable(wchar_t *str, unsigned long n) {
    unsigned long address = get_hash_value(str, n);
    int match=0;
    unsigned long next=0;
    hash_node *node=NULL, *external=NULL;
    while (!(match=match_from_address(address, str, n))) {
        node=NULL; external=NULL;
        node = read_hash_table(address, &external);
        next = (node==NULL?external:node)->next;
        if (external!=NULL) { free(external); external=NULL; }
        if (next==0) break;
        address = next;
    }
    if (match) { // update
        node=NULL; external=NULL;
        node = read_hash_table(address, &external);
        if (node==NULL) node = external;
        ++(node->count);
        write_hash_table(address, node->key_pos, node->count, node->next);
        if (external!=NULL) { free(external); external=NULL; }
        node = NULL;
        return;
    }
    unsigned long new_address = address;
    unsigned long fails=0;
    for(;;) {
        node=NULL; external=NULL;
        node = read_hash_table(new_address, &external);
        unsigned long count = (node==NULL?external:node)->count;
        if (external!=NULL) { free(external); external=NULL; }
        if (count==0) break;
        ++fails;
        assert(fails<hash_mod); // hash table is not full
        new_address = ((new_address-1)+hash_base2)%hash_mod+1; // note that
the address is shifted by 1
    }
    if (fails>0) {
        node=NULL; external=NULL;
        node = read_hash_table(address, &external);
        if (node==NULL) node = external;
        write_hash_table(address, node->key_pos, node->count, new_address);
        if (external!=NULL) { free(external); external=NULL; }
        node = NULL;
    }
```

```
    write_hash_table(new_address, crr_key_n, 1, 0); // address, key_pos,
count, next
    insert_key_table(str, n);
}
```

查詢

```
int match_from_address(unsigned long address, wchar_t *str, unsigned long
n) {
    wchar_t *target=NULL, *external=NULL;
    hash_node *node=NULL, *e_node=NULL;
    node = read_hash_table(address, &e_node);
    if (node==NULL) node = e_node;
    target = read_key_table(node->key_pos, &external);
    if (e_node!=NULL) { free(e_node); e_node=NULL; }
    node = NULL;
    int result = wcsncmp(target==NULL?external:target, str, n);
    if (external!=NULL) { free(external); external=NULL; }
    return result==0?1:0;
}

unsigned long search_hashtable(wchar_t *str, unsigned long n) {
    unsigned long address = get_hash_value(str, n);
    unsigned long next=0;
    hash_node *node=NULL, *external=NULL;
    int match=0;
    while (!(match=match_from_address(address, str, n))) {
        node=NULL; external=NULL;
        node = read_hash_table(address, &external);
        next = (node==NULL?external:node)->next;
        if (external!=NULL) { free(external); external=NULL; }
        if (next==0) break;
        address = next;
    }
    return match?address:0;
}
```

讀寫 hash table 到記憶體/硬碟

```
hash_node* read_hash_table(unsigned long address, hash_node **external) {
    if (address<table_size) {
        return &hash_table[address];
    }
    // in disk
    check_hash_table_boundary(address);
    unsigned long offset = (address-table_size) * sizeof(hash_node);
    hash_node *node = (hash_node*)malloc(sizeof(hash_node));
```

```
    fseek(hash_external, offset, SEEK_SET);
    fread(node, sizeof(hash_node), 1, hash_external);
    *external = node;
    return NULL;
}

void write_hash_table(unsigned long address, unsigned long key_pos,
unsigned long count, unsigned long next) {
    if (address<table_size) {
        hash_table[address].key_pos = key_pos;
        hash_table[address].count = count;
        hash_table[address].next = next;
        return;
    }
    // write to disk
    check_hash_table_boundary(address);
    unsigned long offset = (address-table_size) * sizeof(hash_node);
    hash_node temp;
    temp.key_pos = key_pos;
    temp.count = count;
    temp.next = next;
    fseek(hash_external, offset, SEEK_SET);
    fwrite(&temp, sizeof(hash_node), 1, hash_external);
}
```

讀寫 key buffer 到記憶體/硬碟

```
void insert_key_table(wchar_t *str, unsigned long n) {
    const wchar_t zero=0;
    if (crr_key_n+n+1<=key_table_size) { // space is enough
        wcsncpy(&key_table[crr_key_n], str, n);
        crr_key_n += n;
        key_table[crr_key_n++]=0;
        return;
    }
    if (ext_key_table_base==0xFFFFFFFF) ext_key_table_base = crr_key_n;
    unsigned long offset = (crr_key_n-ext_key_table_base) *
sizeof(wchar_t);
    fseek(key_external, offset, SEEK_SET);
    fwrite(str, sizeof(wchar_t), n, key_external);
    fwrite(&zero, sizeof(wchar_t), 1, key_external);
    crr_key_n += n+1;
}

wchar_t* read_key_table(unsigned long index, wchar_t **ext_str) {
    if (index<ext_key_table_base) return &key_table[index];
    // in disk
    unsigned offset = (index-ext_key_table_base) * sizeof(wchar_t);
```

```
    wchar_t *str = (wchar_t*)malloc(sizeof(wchar_t)*(key_size+1));
    fseek(key_external, offset, SEEK_SET);
    fread(str, sizeof(wchar_t), (key_size+1), key_external);
    *ext_str = str;
    return NULL;
}
```

排序部份

直接使用上次作業完成的 `rsort`

常數/hash_node結構部份

```
const unsigned long hash_base = 311;
const unsigned long hash_base2 = 337;
const unsigned long prime_list[] = {
    2027, 5023, 10061, 20051, 50051,
    100069, 200033, 500083,
    1000691, 2000731, 5000759, 10000139,
    20000327, 50000389, 100000073,
    200000081, 500000057, 1000000123,
    2000000243, 4000003013uL // want more? than go long long -.-
};

typedef struct {
    unsigned long key_pos;
    unsigned long count;
    unsigned long next;
} hash_node;
```

主函式部分

```
int main(const int argc, const char **argv) {
    FILE *fp=NULL;
    setlocale(LC_ALL, ""); // 使用這個， fgetws 才不會出錯
    if (argc==2 && strncmp("--help", argv[1], 6)==0) {
        fprintf(stderr, "Usage:\ntcount filename [-m memory_limit (in MB) |
-s key_size (in byte) | -h hash_table_size (in MB) | -v
virtual_hash_table_size (in MB) ]\n");
        return 0;
    }
    get_args(argc, argv);
    if (argc<2 || !check_exist(argv[1]) ) fp=stdin;
    else fp = fopen(argv[1], "rb");
    /* Process data */

    init_hashtable();
```

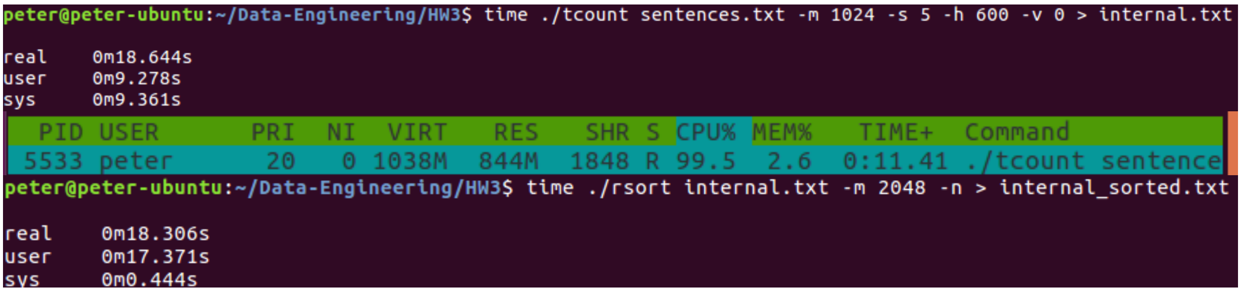
```
    counting(fp);
    dump_table();
    destroy_hashtable();

    /* End process data */
    if (fp!=NULL) {
        fclose(fp); fp=NULL;
    }
    return 0;
}
```

實驗

實驗環境：

- Ubuntu 16.04 64-bit
- 32GB DDR4 (2400MHz)
- AMD Ryzen 5 1600 (6C12T)
- 5400rpm HDD

Case 1: 僅有內部 hash, key buffer



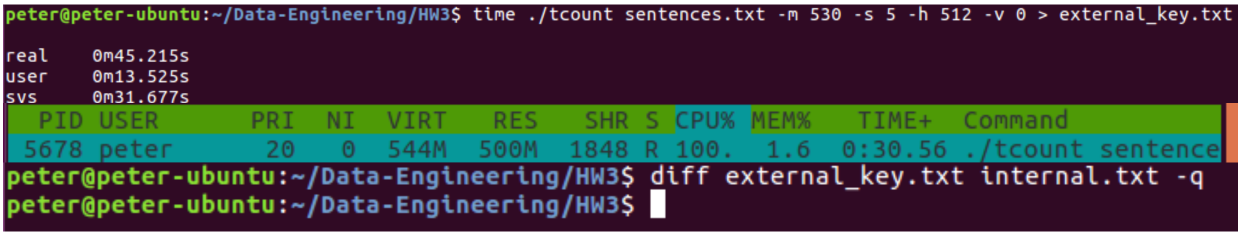
```
peter@peter-ubuntu:~/Data-Engineering/HW3$ time ./tcount sentences.txt -m 1024 -s 5 -h 600 -v 0 > internal.txt

real    0m18.644s
user    0m9.278s
sys     0m9.361s
  PID USER       PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
 5533 peter       20   0 1038M  844M  1848 R 99.5  2.6  0:11.41 ./tcount sentence
peter@peter-ubuntu:~/Data-Engineering/HW3$ time ./rsort internal.txt -m 2048 -n > internal_sorted.txt

real    0m18.306s
user    0m17.371s
sys     0m0.444s
```

tcount: 18.6s, 844MB

排序後結果：

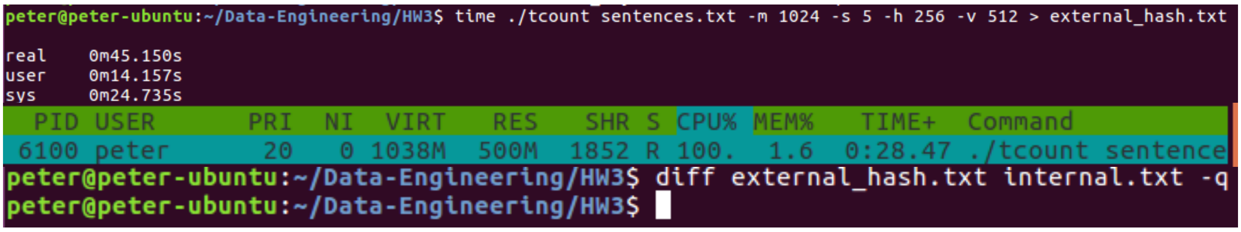
```
5626, 報導內容:
5629, 前開大陸被
5766, 尚應公告關
5958, 粉絲團 ★
6053, 交易標的最
6058, 價格及交易
6069, 本次交易之
6086, 並應公告選
6094, 交付或付款
6107, 含與公司及
6115, 含付款期間
6235, 契約限制條
6396, 每單位價格
6507, 傳播媒體名
6512, 其他敘明事
6536, 圖／記者黃
6543, 價格決定之
6585, 接受資金貸
6734, 加入『ET
7421, 許可從事競
7536, 所擔任該大
8127, 如遭刪除請
8563, 圖／記者陳
8572, 取得或處分
9175, 粉絲團就對
9391, 新任生效日
9407, 新任者姓名
9424, 舊任者姓名
9522, 發生變動日
10176, 董事會決議
10496, 一手掌握林
10592, 迄事實發生
10797, 由達志影像
11954, 異動原因:
13296, 圖／達志影
15600, 因應措施:
15871, 接收更多精
16777, 電視台未經
16806, 不得部分或
17967, 交易相對人
24396, 公司名稱:
24853, 相互持股比
25094, 發生緣由:
25643, 與公司關係
42291, 圖／翻攝自
66785, 其他應敘明
76325, 資料如有虛
76325, 均由該公司
76325, 由本系統對
peter@peter-ubuntu:~/Data-Engineering/HW3$
```

Case 2: 大部份 key buffer 在硬碟，所有 hash table 在記憶體



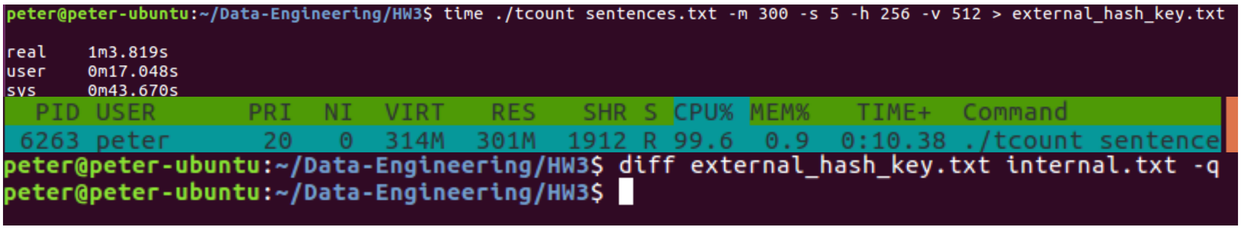
tcount: 45.2s, 500MB

Case 3: 1/2 hash table 在硬碟，所有 key buffer 在記憶體



tcount: 45.2s, 500MB

Case 4: 1/2 hash table 在硬碟，幾乎所有 key buffer 在硬碟



tcount: 63.8s, 301MB

結論

當記憶體不足，寫到 HDD 時，hash table 的性能下降得非常明顯。

GitHub

程式碼： [tcount.c]