

# 資料工程 HW1

404410030 資工四 鄭光宇

## 系統需求

要執行這支程式，系統必須具備：

- 支援 ANSI C 的 gcc
- make 工具程式

## 編譯

```
make
```

## 執行

```
./rsort filename [-d delimiter | -k field | -m memory_limit | -f output_filename (o.
```

延續上次作業要求，另外新增了 `-s` 照資料大小排序。

`-d` 是分隔符號，`-k` 是要作為 key 的 pattern

而 `-c` 是不區分大小寫、`-n` 使用數值排序、`-r` 倒序（降序）排序。

最重要的，`-m` 限制大約使用多少 MB 的記憶體做內部排序。

## 資料

先將助教提供的檔案，合成一個大的 `full_data.rec` 檔

檔案大小 17GB 左右

## 觀察

---

```
@
@url:https://www.youtube.com/watch?v=_X1Aey4MX1c
@published:
@title:проверка микрофона - YouTube
@content:
@author:
@favoriteCount:0
@commentCount:22
@res:144
@duration:60
@category:人物與網誌
@
@url:https://www.youtube.com/watch?v=_VaRJ8D0LZo
@published:2016年1月9日
@title:The Hamsters home - YouTube
@content:
@author:
@favoriteCount:1
@commentCount:6
@res:144
@duration:37
@category:喜劇
@
@url:https://www.youtube.com/watch?v=_W7LT-x0K1g
@published:2017年6月24日
@title:My last hack - YouTube
@content:
@author:
@favoriteCount:2
@commentCount:40
@res:144
@duration:46
@category:人物與網誌
```

資料似乎是以 `_\\n@` 的方式分隔 ( `_` 為空格)

所以之後都使用這個分隔方式處理資料

# 實作

---

下面列出一些主要的程式碼

## 讀參數

跟上一次內部排序用一樣的程式碼

```
33 void get_args(const int argc, const char** args, const char **parameters, int *set_parameters) {
34     /** parse all parameters in following order
35     * -d record_delimiter
36     * -k key_pat
37     * -m memory_limit (in MB)
38     * -f output file path
39     * -c case_insensitive
40     * -r reverse order
41     * -n numerical comparison
42     * -s size_sort
43     */
44     int i = 0, argspos=-1;
45     for (i=0; i<4; ++i) {
46         argspos = parse_parameter(argc, args, parameter_patterns[i]);
47         parameters[i] = argspos<0?default_args[i]:args[argspos+1];
48     }
49     for (i=0; i<4; ++i) {
50         set_parameters[i] = parse_parameter(argc, args, parameter_patterns[i+4])<0?0:1;
51     }
52 }
```

## 資料間的比較函式

```

54  int comp(const void *a, const void *b) {
55      int e=0, f=0;
56      const char *c = *(const char**)a;
57      const char *d = *(const char**)b;
58      int val = 0;
59      if ( set_parameters[3] ) { /* sort by "size" */
60          val = strlen(c) - strlen(d);
61      } else {
62          if (parameters[1]!=NULL) { /* has field */
63              /* not robust enough. need to handle more exceptions */
64              c = strstr(c, parameters[1]); /* jump to that field */
65              if (c==NULL) return -1;
66              d = strstr(d, parameters[1]); /* ,, */
67              if (d==NULL) return 1;
68          }
69          if (set_parameters[2]) { /* numerical comparison? */
70              val = 0;
71              while(c!=NULL&&*c!='\0'&&!isdigit(*c)) ++c, ++val;
72              if (val>0 && *(c-1)=='-') --c;
73              val = 0;
74              while(d!=NULL&&*d!='\0'&&!isdigit(*d)) ++d, ++val;
75              if (val>0 && *(d-1)=='-') --d;
76              e = atoi(c);
77              f = atoi(d);
78              val = e-f;
79          } else { /* lexical order */
80              val = set_parameters[0]?strcasecmp(c,d):strcmp(c,d); /* case insensitive? */
81          }
82      }
83      return set_parameters[1]?-val:val; /* reverse order? */
84  }

```

針對各種參數有不同的比較方式

## 檔案寫出

```

93  void writeout(FILE *fp, char **row, const unsigned long records_cnt, const char write_head) {
94      unsigned long i=0;
95      if (fp==NULL) fp=stdout;
96      for (i=0; i<records_cnt; ++i) {
97          if (i>0 || write_head) {
98              fputs(parameters[0], fp);
99          }
100         fputs(row[i], fp);
101     }
102 }

```

## 讀檔/排序/分割檔/寫出

逐漸讀入檔案，每到了指定的記憶體容量限制，就寫出一個 chunk 到硬碟中、清除記憶體、記錄增加的 chunk 數並記錄他們寫入的路徑。這個函數可以指定要讀多少筆資料，可以協助後面 external sort 的 merge 階段。

```
122 void split_sort(FILE *fp, split_sort_handler *results, record_struct *records, int max_rec) {
123 #define GROW(X, X_new, cap, cnt, type) { cap *= 2; \
124     X_new = NULL; \
125     X_new = (type*)malloc(sizeof(type)*cap); \
126     memcpy(X_new, X, sizeof(type)*cnt); \
127     free(X); \
128     X=X_new; X_new=NULL; }
129 unsigned long i=0;
130 char ch=0;
131 FILE *temp=NULL;
132 if (results==NULL && records==NULL) return;
133
134 char *buffer=NULL, *new_buffer=NULL;
135 char **rows=NULL, **new_rows=NULL;
136 FILE **tmp_fp=NULL, **new_tmp_fp=NULL;
137 char *record_strings = NULL;
138 unsigned long tmp_fp_cap=4;
139 unsigned long tmp_fp_cnt=0;
140 unsigned long buffer_cap=1024;
141 unsigned long buffer_cnt=0;
142 unsigned long rows_cap=1024;
143 unsigned long rows_cnt=0;
144 unsigned long long mem_use=0;
145 unsigned long chunk_n=0;
146 int string_length=0;
147 unsigned long delimiter_length = strlen(parameters[0]);
148 char has_head = 2;
149 const unsigned long long buffer_limit = atoll(parameters[2]) * (1uLL<<20uLL); /* KB=2^10, MB=2^20 */
150 buffer = (char*)malloc(sizeof(char)*buffer_cap);
151 rows = (char**)malloc(sizeof(char*)*rows_cap);
152 tmp_fp = (FILE**)malloc(sizeof(FILE*)*tmp_fp_cap);
```

```

153     if (buffer==NULL||rows==NULL||tmp_fp==NULL) {
154         fprintf(stderr, "Couldn't allocate more memory\n");
155         exit(3);
156     }
157
158     for(;;) {
159         ch = (max_rec>0 && rows_cnt>=max_rec)?EOF:fgetc(fp);
160         if (buffer_cnt+1==buffer_cap) {
161             GROW(buffer, new_buffer, buffer_cap, buffer_cnt, char);
162         }
163         if (ch!=EOF) {
164             buffer[buffer_cnt++] = ch;
165             buffer[buffer_cnt ] = '\0'; /* tail */
166         }
167         string_length = buffer_cnt - delimiter_length;
168         if(string_length>0 && (ch==EOF || strcmp(buffer+string_length, parameters[0], delimiter_length)==0)) { /* new record
169             if (rows_cnt==rows_cap) {
170                 GROW(rows, new_rows, rows_cap, rows_cnt, char* );
171             }
172             record_strings = NULL;
173             if (has_head==2 && (has_head = string_length > delimiter_length && strcmp(buffer, parameters[0], delimiter_length
174                 string_length -= delimiter_length;
175                 record_strings = (char*)malloc(sizeof(char)*(string_length+1));
176                 memcpy(record_strings, buffer+delimiter_length, sizeof(char)*string_length);
177                 record_strings[string_length] = '\0';
178             } else {
179                 record_strings = (char*)malloc(sizeof(char)*(string_length+1));
180                 memcpy(record_strings, buffer, sizeof(char)*string_length);
181                 record_strings[string_length] = '\0';
182             }
183             rows[rows_cnt++] = record_strings;

```

```

184         buffer_cnt = 0; /* reset. read next record */
185         mem_use += (unsigned long long)(sizeof(char)*(string_length+1));
186     }
187     if (results!=NULL && (mem_use>=buffer_limit || ch==EOF)) { /* write file */
188         qsort((void*)rows, rows_cnt, sizeof(char*), comp); /* now sort this portion */
189         FILE *temp = tmpfile(); /* w+ mode */
190         writeout(temp, rows, rows_cnt, 0); /* write first delimiter */
191         fputs("\n", temp);
192         fseek(temp, 0, SEEK_SET); /* move to begining of the file */
193         /* remember filepaths */
194         if (tmp_fp_cnt==tmp_fp_cap) {
195             GROW(tmp_fp, new_tmp_fp, tmp_fp_cap, tmp_fp_cnt, FILE* );
196         }
197         tmp_fp[tmp_fp_cnt++] = temp;
198         for (i=0; i<rows_cnt; ++i) {
199             free(rows[i]);
200             rows[i]=NULL;
201         }
202         rows_cnt = 0;
203         mem_use = 0;
204     }
205     if (ch==EOF) break;
206 }
207 free(buffer); buffer=NULL;
208
209 if (results!=NULL) {
210     for (i=0; i<rows_cnt; ++i) {
211         free(rows[i]);
212         rows[i]=NULL;
213     }
214     free(rows); rows=NULL;

```

```

215         results->n_chunk  = tmp_fp_cnt;
216         results->has_head = has_head;
217         results->temp_fp  = tmp_fp;
218     } else {
219         records->n_record = rows_cnt;
220         records->data = rows;
221         records->has_head = has_head;
222     }
223     #undef GROW
224 }

```

## K-way merge

每個 queue 為上個階段已排序的 chunk，我們利用 Winner-Tree 維護所有 queue 中最小的值。並且在每次查詢  $O(1)$ ，每次更新  $O(K)$  的複雜度下，不斷從 Winner-Tree 頂端 pop 出最小值，這些 pop 出來的資料會循序寫入到檔案，這樣就可以得到已經排序的文件。

Winner-Tree 的初始化方式（使用 array 實作）：

1. 初始化所有點為無限大
2. 將所有 queue 的第一個元素放在 Winner-Tree 的 leaf 上（array 尾端）
3. 由於 Winner-Tree 是 (nearly) Complete Binary Tree，從 array 最尾端往開頭走訪，過程中每次檢查如果自己的數值比 parent 小，就將 parent 更新。

Winner-Tree 的更新方式簡單來說：

1. 從頂端 pop 出 key 最小值
2. 從最小值對應回的那個 queue 再讀一筆資料。若資料為空，讀入資料的節點大小為無限大。
3. 走訪到自己的 parent，從 parent 看兩個 child 哪個小，哪個提上去到 parent 的位置。重複步驟直到走到 root，一次更新的操作就完成了。

不斷重複 pop 最小值，直到所有 queue 為空（此時 Winner-Tree 頂端也為空）。完成外部排序。

```
226 void merge_and_out(split_sort_handler *handler) {
227     #define LCH(X) ((X<<1)+1)
228     #define RCH(X) ((X<<1)+2)
229     #define PAR(X) ((X-1)>>1)
230     /* precondition: every file pointer points to the beginning of each file */
231     char complete_bt = (handler->n_chunk&1); /* is complete binary tree */
232     unsigned long node_num = 2 * (handler->n_chunk + (complete_bt==0?1:0)) - 1; /* # of nodes (if not complete. add a dumm
233     unsigned long i=0;
234     unsigned long *nodes = NULL;
235     unsigned long K = handler->n_chunk;
236     unsigned long hidden_nodes=0;
237     FILE *out_fp = stdout;
238     if (parameters[3]!=NULL) {
239         out_fp = fopen(parameters[3], "w");
240         if (out_fp==NULL) exit(5);
241     }
242     record_struct *records = NULL;
243     nodes = (unsigned long*)malloc(sizeof(unsigned long)*node_num);
244     if (nodes==NULL) exit(3);
245     memset(nodes, 0xFF, sizeof(unsigned long)*node_num); /* set infinity (kinda) */
246
247     records = (record_struct*)malloc(sizeof(record_struct)*K);
248     if (records==NULL) exit(4);
249
250     for (i=0; i<K; ++i) split_sort(handler->temp_fp[i], NULL, records+i, 1); /* read first K first elements */
251
252     if (!complete_bt) { /* if not strictly complete */
253         --node_num; /* hide dummy node */
254     }
255 }
```



```

256     hidden_nodes = node_num - K;
257     #define NI2KI(X) (X-hidden_nodes)
258     #define KI2NI(X) (X+hidden_nodes)
259     for (i=hidden_nodes; i<node_num; ++i) nodes[i] = NI2KI(i); /* initialize leaf nodes */
260
261     /* initialize root and hidden node */
262     for (i=node_num-1; i>0; --i) { /* zero based */
263         unsigned long p = nodes[PAR(i)];
264         unsigned long ch = nodes[i];
265         if( p >= K || comp_record( records+p, records+ch )>0 ) { /* if parent is not initialized or is bigger */
266             nodes[PAR(i)] = nodes[i]; /* update */
267         }
268     }
269
270     char first=1;
271     for(;;) { /* while top element is non-empty */
272         if (records[nodes[0]].n_record==0) break;
273         writeout(out_fp, records[nodes[0]].data, 1, first?handler->has_head:1);
274         cleanup_record_struct(records+nodes[0]); /* pop current data */
275         first=0;
276         split_sort(handler->temp_fp[ nodes[0] ], NULL, records+nodes[0], 1); /* read new data from disk */
277         unsigned long ni = KI2NI( nodes[0] );
278         while (ni!=0) { /* until reach root */
279             unsigned long pa = PAR(ni);
280             unsigned long lch = LCH(pa);
281             unsigned long rch = RCH(pa);
282             ni = lch; /* assume that lch is smaller */
283             if ( rch<node_num && comp_record( records+nodes[lch], records+nodes[rch] ) > 0 ) { /* right child exists and sr
284                 ni = rch; /* change to rch */
285             }

```

```

287         ni = pa; /* bottom up */
288     }
289 }
290
291 free(nodes); nodes=NULL;
292 for (i=0; i<K; ++i) cleanup_record_struct(records+i);
293 free(records);
294 records=NULL;
295 fputs("\n", out_fp);
296 if (out_fp!=NULL && out_fp!=stdout) fclose(out_fp);
297 out_fp=NULL;
298 #undef KI2NI
299 #undef NI2KI
300 #undef LCH
301 #undef RCH
302 #undef PAR
303 }

```

## 主函式部分

```
305  int main(const int argc, const char **argv) {
306      int i=0;
307      FILE *fp=NULL;
308      int records_cnt = 0;
309      char has_head = 1;
310      split_sort_handler handle;
311      record_struct records;
312      if (argc<2) {
313          fprintf(stderr, "Usage:\nrsort filename [-d delimiter | -k field | -m memory_limit | -f output_filename (o.w. stdout)
314          exit(2);
315      }
316      get_args(argc, argv, parameters, set_parameters);
317      fp = fopen(argv[1], "r");
318      split_sort(fp, &handle, &records, -1);
319      fclose(fp); fp=NULL;
320      merge_and_out(&handle);
321      cleanup_split_sort_handle(&handle);
322      return 0;
323  }
```

## 實驗

### 照資料大小排序（`-s` 模式）

---

```
time ./rsort full_data.rec -d '$' \n@' -m 4096 -s -f sorted_by_size.rec
```

@favoriteCount:3  
@viewCount:310  
@res:144  
@duration:247  
@category:音楽  
@  
@url:https://www.youtube.com/watch?v=P4ElpYYUB9Q  
@published:2014年5月24日  
@title:【CoD:G】 ハセシンの実況プラベ ～クラメンと1対1シリーズ#7vsレオジン～part9  
@content:どーもハセシンです(・ω´・)♪ 今回はレオジンさんと1対1をしていきましたが、  
メチャ感を楽しんでもらえたら幸いです(\*´ω`\*)♪ ってことで楽しんで見ていってね☆彡 チ  
@author:遊 戯  
@favoriteCount:543  
@viewCount:43520  
@res:144  
@duration:837  
@category:遊 戯  
@  
@url:https://www.youtube.com/watch?v=P4-njgdDBXA  
@published:2012年5月16日  
@title:Lucy - may 2012 - Jones Ranch Arena - wednesday night - YouTube  
@content:I&#39;m really proud of her!!  
@author:smokumifugotum  
@favoriteCount:0  
@viewCount:85  
@res:144  
@duration:40  
@category:寵 物 與 動 物  
@  
@url:https://www.youtube.com/watch?v=P3ugroaSQFk  
@published:2011年3月16日  
@title:新幹線那須塩原まで 乗客さまざまな思い - YouTube  
@content:3/17 とちぎテレビニュース  
@author:Tetsuro Tanaka  
@favoriteCount:1  
@viewCount:3121  
@res:144  
@duration:99  
@category:新 聞 與 政 治  
@  
@url:https://www.youtube.com/watch?v=P4AH6CFRzik  
@published:2016年10月4日  
@title:Barryman 626-6866 - YouTube

real	7m15.260s
user	6m36.695s
sys	0m18.974s

可以看出最大的一筆資料因為沒有恰當的 `_\\n@` 分隔

排序大約花費 7 分鐘左右

## 字典序（預設模式）

---

字典序排出來看起來很亂，這裡指附上執行時間供參考

```
real    6m52.696s
user    6m12.542s
sys     0m20.103s
```

## 時間序（照發佈時間的字串排序）

---

```
time ./rsort full_data.rec -d '$' \\n@' -m 4096 -k "@published:" -f sorted_by_publish_time.rec
```

```
@url:https://www.youtube.com/watch?v=pVAHiKfhxqU
@published:2017年8月9日
@title:😄😄 - YouTube
@content:ทำคริปนี้พูดค าว่า &quot;ต่อมา&quot;;มากเกินไปต้องขอโทษด้วยนะค่ะ ชื่อ👉เหมย. อายุ👉11.
@author:

@favoriteCount:0
@viewCount:6
@res:144
@duration:158
@category:遊戲
@
@url:https://www.youtube.com/watch?v=ELiqq12KG3M
@published:2017年8月9日
@title:😭😭😭 - YouTube
@content:
@favoriteCount:0
@viewCount:25
@res:144
@duration:21
@category:遊戲
@
@url:https://www.youtube.com/watch?v=axU3R-Eesek
@published:2017年8月9日
@title:😱Vlogging at the supermarket 😱 - YouTube
@content:Guys do me a favor : ➡️Tap my bell ➡️Suscribe ➡️turn on my post notificat
@author:

@favoriteCount:2
@viewCount:32
@res:144
@duration:92
@category:遊戲
@
@url:https://www.youtube.com/watch?v=xVRRYsy_fX4
@published:2017年8月9日
@title:😱Yeni mod paketim (MC:SG 16)720p60😱 - YouTube
@content:Herkeze tekrardan merhaba bugün sizlerle Minecraft oynadik like atmayi,ab
-----...
@author:

@author:BF4-Team-Delta
```

```
real      7m30.829s
user      6m50.910s
sys       0m19.383s
```

可以看出成功照日期排序， 2017年8月9日 排在一起了

排序大約 7min

## 驗證程式碼正確性

生成 1000000 個隨機整數（約 5.7MB）

```
for ((i=0; i<1000000; ++i)); do echo $RANDOM >> random_numbers; done
```

使用 `sort` 排序，作為標準答案

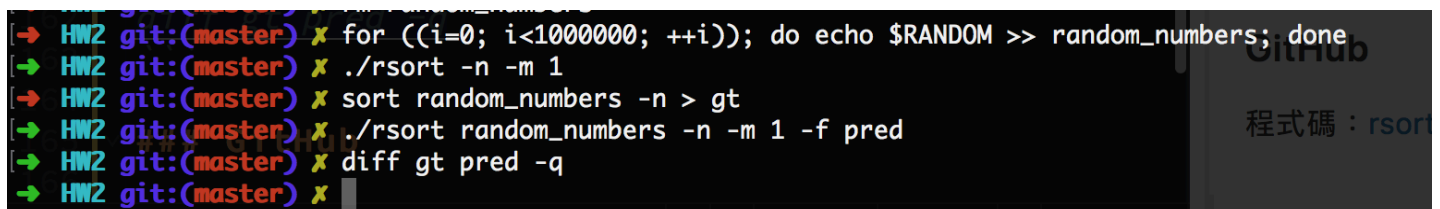
```
sort random_numbers -n > gt
```

使用 `rsort` 進行排序（用很小的 buffer size，為了驗證 k-way merge 能正常工作）

```
./rsort random_numbers -n -m 1 -f pred
```

比較兩者結果，沒報錯答案就是一樣

```
diff gt pred -q
```



```
→ HW2 git:(master) x for ((i=0; i<1000000; ++i)); do echo $RANDOM >> random_numbers; done
→ HW2 git:(master) x ./rsort -n -m 1
→ HW2 git:(master) x sort random_numbers -n > gt
→ HW2 git:(master) x ./rsort random_numbers -n -m 1 -f pred
→ HW2 git:(master) x diff gt pred -q
→ HW2 git:(master) x
```

## GitHub

程式碼：<https://github.com/peter0749/Data-Engineering/blob/master/HW2/rsort.c>