

Universitat de Barcelona

FACULTAT DE MATEMÀTIQUES I INFORMÀTICA

INFORME PRÀCTICA 3: ANÀLISI DEL NOMBRE DE VOLS ENTRE DIFERENTS AEROPORTS

Sistemes Operatius 2

Junjie Li i Manuel Liu Wang

Octubre 2022

1 Introducció

Introducció

2 Treball realitzat

Pregunta 1: Executar el codi i provar el seu funcionament. Observar que l'aplicació imprimeix per pantalla 303 aeroports d'origen. Identifiqueu aeroports coneguts? Nova York? Atlanta? Comenteu els resultats.

```
Origin: STX -- Number of different destinations: 0
Origin: SUN -- Number of different destinations: 0
Origin: SUX -- Number of different destinations: 0
Origin: SWF -- Number of different destinations: 0
Origin: SYR -- Number of different destinations: 0
Origin: TEX -- Number of different destinations: 0
Origin: TLH -- Number of different destinations: 0
Origin: TOL -- Number of different destinations: 0
Origin: TPA -- Number of different destinations: 31
Origin: TRI -- Number of different destinations: 0
Origin: TUL -- Number of different destinations: 6
Origin: TUP -- Number of different destinations: 0
Origin: TUS -- Number of different destinations: 6
Origin: TVC -- Number of different destinations: 0
Origin: TWF -- Number of different destinations: 0
Origin: TXK -- Number of different destinations: 0
Origin: TYR -- Number of different destinations: 0
Origin: TYS -- Number of different destinations: 0
Origin: VLD -- Number of different destinations: 0
Origin: VPS -- Number of different destinations: 0
Origin: WRG -- Number of different destinations: 0
Origin: WYS -- Number of different destinations: 0
Origin: XNA -- Number of different destinations: 0
Origin: YAK -- Number of different destinations: 0
Origin: YKM -- Number of different destinations: 0
Origin: YUM -- Number of different destinations: 0
Tiempo para procesar el fichero: 0.094904 segundos
root@DESKTOP-D024E0C:/mnt/c/ub/3/tardo/so2/practica/practica_3/p3-analisi de codi/codi#
```

Figure 1: resultat del codi

Utilitzant codis IATA, només podem reconèixer alguns dels aeroports més famosos, per exemple: New York - John F.Kennedy (**JFK**), Miami (**MIA**), Los Angeles - International (**LAX**), però a través de https://www.nationsonline.org/oneworld/IATA_Codes/airport_code_list.htm podem conèixer clarament l'aeroport corresponent segons el codi IATA.

Pregunta 2: Quins són els aeroports amb més destinacions? Com us ho feu per saber-ho? Hi ha alguna comanda que us ho permeti saber?

```
root@DESKTOP-D024E0C:/mnt/c/ub/3/tardo/so2/practica/practica_3/p3-analisi de codi/codi# ./analisi aeroportos.csv fitxer_petit.csv | sort -t':' -k 3 -n -r
Origin: LAS -- Number of different destinations: 54
Origin: MDW -- Number of different destinations: 47
Origin: PHX -- Number of different destinations: 42
Origin: BWI -- Number of different destinations: 38
Origin: MCO -- Number of different destinations: 33
Origin: TPA -- Number of different destinations: 31
Origin: HOU -- Number of different destinations: 29
Origin: BNA -- Number of different destinations: 27
Origin: STL -- Number of different destinations: 22
Origin: ABQ -- Number of different destinations: 22
Origin: OAK -- Number of different destinations: 20
```

Figure 2: ./analisi aeroportos.csv fitxer_petit.csv | sort

Utilitzant la següent línia de comanda ens podem indicar directament a la terminal quin aeroport és l'aeroport amb més destinacions: `./analisi aeroportos.csv fitxer_petit.csv | sort -t':' -k 3 -n -r`, Podem

utilitzar **pipe** de bash i **sort** per processar la output generada per codi, L'opció **-t** ens permet triar el delimitador amb **':'**, l'opció **-k 3** selecciona la columna 3 del fitxer de pipe amb **-n** és numeric-sort amb **-r** s'ordena amb més gran a més petit.

.....

Pregunta 3: Quan triga el codi executar-se sobre els fitxers 2007.csv o 2008.csv? Quina comanda es pot fer servir per mesurar el temps (real, usuari i sistema) d'un procés? Aquests fitxers es poden baixar des del campus virtual de l'assignatura.

Podem utilitzar la comanda del **time** per obtenir el temps d'execució del programa en diferents fitxers (fitxer_petit.csv, 2007.csv, 2008.csv), A partir de la captura de pantalla següent, podem comparar que el codi triga més temps a executar 2008.csv entre els tres fitxers.

```
Origin: YAK -- Number of different destinations: 0
Origin: YKM -- Number of different destinations: 0
Origin: YUM -- Number of different destinations: 0
Tiempo para procesar el fichero: 0.055181 segundos

real    0m0.082s
user    0m0.028s
sys      0m0.011s
root@DESKTOP-D024E0C:/mnt/c/ub/3/tardo/so2/practica/practica_3/p3-analisi de codi/codi# time ./analisi aerop
orts.csv fitxer_petit.csv
```

Figure 3: time de fitxer_petit.csv

```
Origin: YAK -- Number of different destinations: 2
Origin: YKM -- Number of different destinations: 1
Origin: YUM -- Number of different destinations: 6
Tiempo para procesar el fichero: 37.553307 segundos

real    0m37.560s
user    0m12.840s
sys      0m1.086s
root@DESKTOP-D024E0C:/mnt/c/ub/3/tardo/so2/practica/practica_3/p3-analisi de codi/codi# time ./analisi aerop
orts.csv 2007.csv
```

Figure 4: time de 2007.csv

```
Origin: YAK -- Number of different destinations: 2
Origin: YKM -- Number of different destinations: 1
Origin: YUM -- Number of different destinations: 6
Tiempo para procesar el fichero: 44.796193 segundos

real    0m44.822s
user    0m12.018s
sys      0m0.967s
root@DESKTOP-D024E0C:/mnt/c/ub/3/tardo/so2/practica/practica_3/p3-analisi de codi/codi# time ./analisi aerop
orts.csv 2008.csv
```

Figure 5: time de 2008.csv

.....

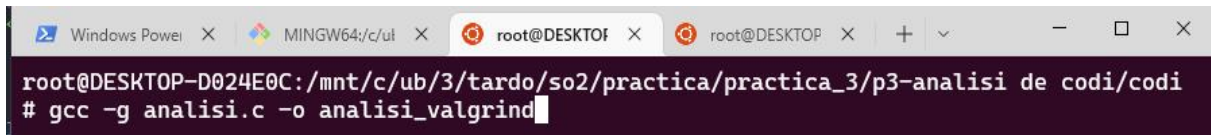
Pregunta 4: Quants vols emmagatzemen aquests dos fitxers (2007.csv i 2008.csv)? Amb quina comanda podeu esbrinar aquesta informació?

```
root@DESKTOP-D024E0C:/mnt/c/ub/3/tardo/so2/practica/practica_3/p3-analisi de codi/codi# cat 2007.csv | wc -l
7453216
root@DESKTOP-D024E0C:/mnt/c/ub/3/tardo/so2/practica/practica_3/p3-analisi de codi/codi# cat 2008.csv | wc -l
7009729
```

Figure 6: quantitat de vols

Quan utilitzem **wc -l** podem conèixer el nombre de vols per cada fitxers, **wc** és una línia d'ordres per a estadístiques i **-l** pot calcular el nombre de línies del fitxer, perquè al fitxer, cada línia representa la informació de cada vol.

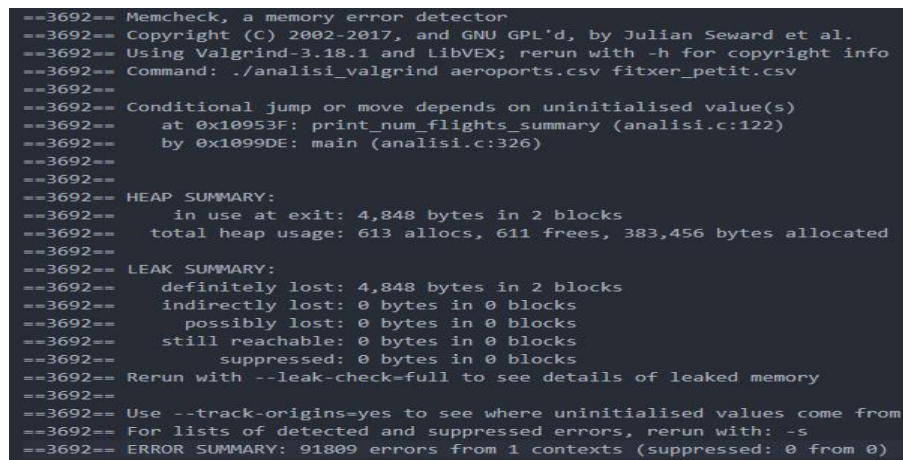
Pregunta 5: Exercicis de valgrind.



```
root@DESKTOP-D024E0C:/mnt/c/ub/3/tardo/so2/practica/practica_3/p3-analisi de codi/codi
# gcc -g analisi.c -o analisi_valgrind
```

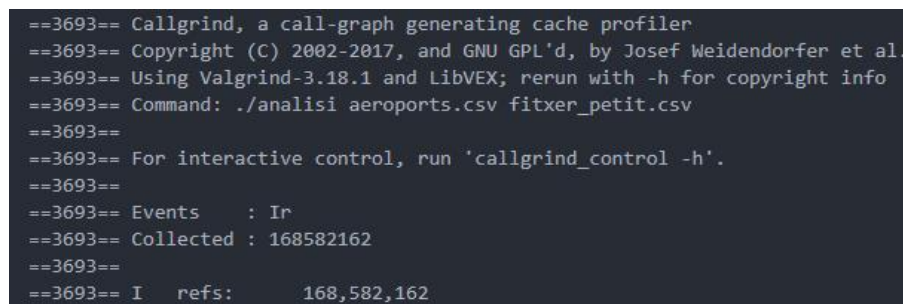
Figure 7: compilar amb valgrind

Per utilitzar l'eina de valgrind per debuggar el nostre codi, hem d'utilitzar una altra manera de compilar el nostre codi, aquí fem servir l'opció **gcc -g** per compilar el nostre codi de manera que es pugui utilitzar valgrind



```
==3692== Memcheck, a memory error detector
==3692== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==3692== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==3692== Command: ./analisi_valgrind aeroportos.csv fitxer_petit.csv
==3692== Conditional jump or move depends on uninitialised value(s)
==3692==   at 0x10953F: print_num_flights_summary (analisi.c:122)
==3692==   by 0x1099DE: main (analisi.c:326)
==3692==
==3692== HEAP SUMMARY:
==3692==   in use at exit: 4,848 bytes in 2 blocks
==3692==   total heap usage: 613 allocs, 611 frees, 383,456 bytes allocated
==3692==
==3692== LEAK SUMMARY:
==3692==   definitely lost: 4,848 bytes in 2 blocks
==3692==   indirectly lost: 0 bytes in 0 blocks
==3692==   possibly lost: 0 bytes in 0 blocks
==3692==   still reachable: 0 bytes in 0 blocks
==3692==   suppressed: 0 bytes in 0 blocks
==3692== Rerun with --leak-check=full to see details of leaked memory
==3692==
==3692== Use --track-origins=yes to see where uninitialised values come from
==3692== For lists of detected and suppressed errors, rerun with: -s
==3692== ERROR SUMMARY: 91809 errors from 1 contexts (suppressed: 0 from 0)
```

Figure 8: executar amb valgrind



```
==3693== Callgrind, a call-graph generating cache profiler
==3693== Copyright (C) 2002-2017, and GNU GPL'd, by Josef Weidendorfer et al.
==3693== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==3693== Command: ./analisi aeroportos.csv fitxer_petit.csv
==3693==
==3693== For interactive control, run 'callgrind_control -h'.
==3693==
==3693== Events      : Ir
==3693== Collected : 168582162
==3693==
==3693== I    refs:    168,582,162
```

Figure 9: executar amb valgrind tool=callgrind

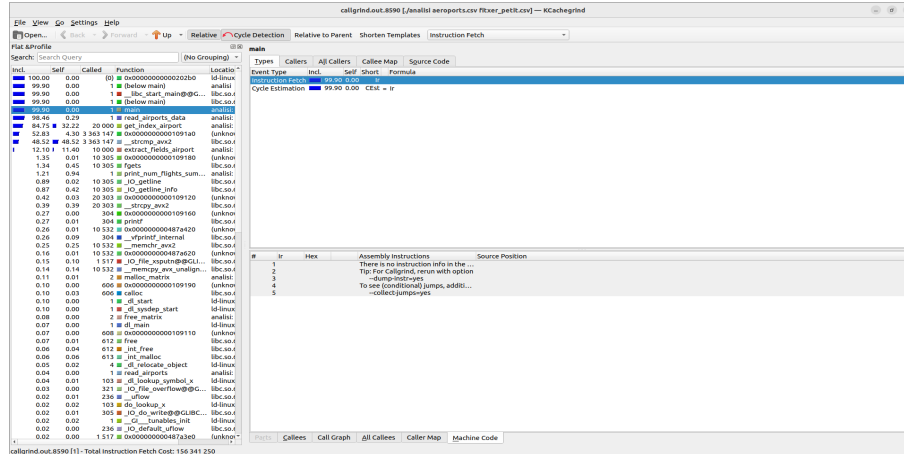


Figure 10: executar amb kcachegrind

La imatge de anterior son els cas d'executar valgrind de manera diferent

Pregunta 6: El codi entregat no és net, sinó que té errors de programació. Els podeu identificar? Heu de trobar aquests errors, comentar los a l'informe, i tornar el codi net amb el fitxer ZIP que hey de lliurar al campus virtual.

Segons el diagnòstic de valgrind anterior, podem saber que hi ha dos problemes al codi:

```
6  ==144== Conditional jump or move depends on uninitialised value(s)
7  ==144==   at 0x10953F: print_num_flights_summary (analisi.c:123)
8  ==144==   by 0x1099DE: main (analisi.c:327)
9  ==144==   Uninitialised value was created by a heap allocation
10 ==144==   at 0x4848899: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
11 ==144==   by 0x109305: malloc_matrix (analisi.c:28)
12 ==144==   by 0x10998F: main (analisi.c:318)
```

Figure 11: error uninitialized value

```
1 void **malloc_matrix(int nrow, int ncol, size_t size){
2
3     int i;
4     void **ptr = NULL;
5
6     ptr = (void **) malloc(sizeof(void *) * nrow);
7     for(i = 0; i < nrow; i++){
8         ptr[i] = NULL;
9         ptr[i] = (void *) calloc(1, size * ncol);
10    }
11
12    return ptr;
13 }
```

Figure 12: Solució amb error 2

El primer problema és que quan s'utilitza **malloc**, **malloc** només assigna espai però no inicialitza les variables, així que aquí, buidem el valor de `ptr[i]` de cadascun a **NULL** per evitar el problema heretat de les variables d'adreça, i després utilitzem **calloc** per a la inicialització de `ptr[i]`, perquè l'espai assignat per **calloc** s'inicialitzarà automàticament a 0.

```
15 ==144== HEAP SUMMARY:
16 ==144==    in use at exit: 4,848 bytes in 2 blocks
17 ==144== total heap usage: 613 allocs, 611 frees, 383,456 bytes allocated
18 ==144==
19 ==144== 2,424 bytes in 1 blocks are definitely lost in loss record 1 of 2
20 ==144==    at 0x4848899: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
21 ==144==    by 0x1092D0: malloc_matrix (analisi.c:26)
22 ==144==    by 0x109977: main (analisi.c:317)
23 ==144==
24 ==144== 2,424 bytes in 1 blocks are definitely lost in loss record 2 of 2
25 ==144==    at 0x4848899: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
26 ==144==    by 0x1092D0: malloc_matrix (analisi.c:26)
27 ==144==    by 0x10998F: main (analisi.c:318)
```

Figure 13: error memory leak

```
void free_matrix(void **matrix, int nrow)
{
    int i;
    for(i = 0; i < nrow; i++){
        free(matrix[i]);
    }
    free(matrix);
}
```

Figure 14: solució amb error 2

El segon problema és el problema de la pèrdua de memòria del heap. El codi original només va alliberar la memòria de cada fila de la matrix, però s'ha oblidat d'alliberar la memòria de la pròpia de la matrix, de manera que va provocar la pèrdua de memòria del heap. Al nostre codi, hem posat un **free** per matrix pròpia, hem arreglat el problema.

3 Conclusió

conclusio