Gyorsrendezés elemzése DDD segítségével

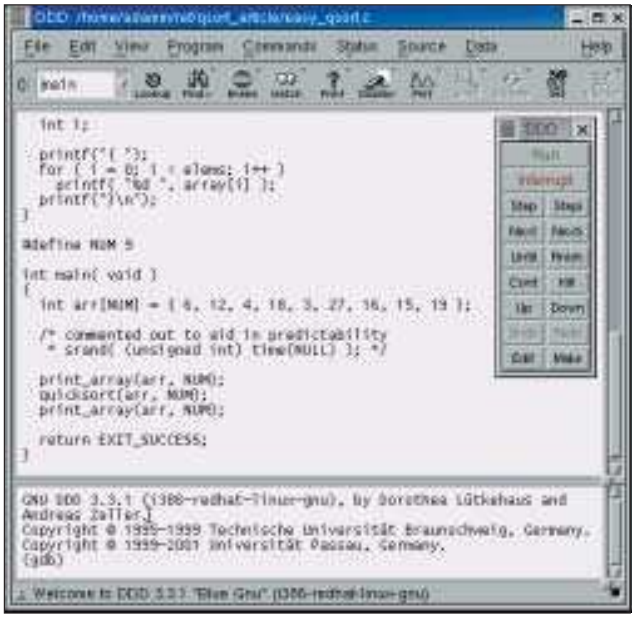
Ismerkedjünk meg egy jó hibakeresővel, és értsük meg a gyorsrendezést!

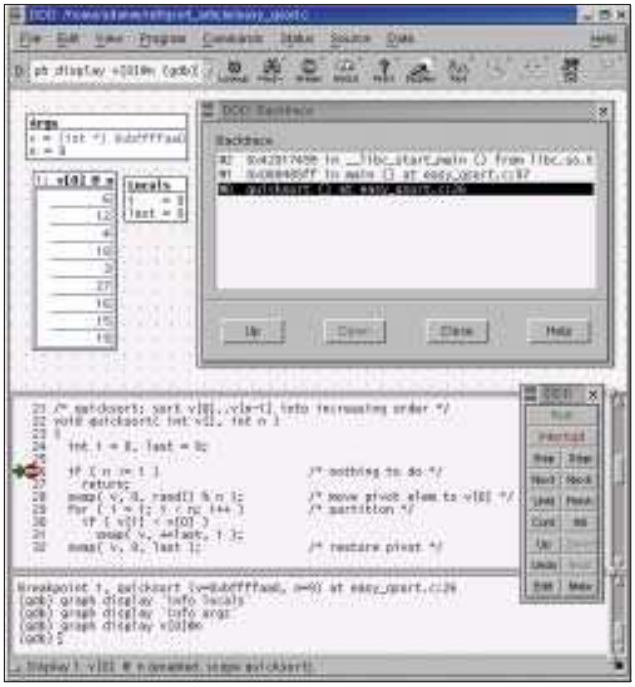
A rendezés a számítógépek által az egyik leggyakrabban végrehajtott művelet, és erre a gyorsrendezés az egyik leghatékonyabb módszer. Írásomban egyrészt bemutatom, mennyire hasznos lehet egy grafikus hibakereső, másrészt ismertetem a gyorsrendezés elvét.
A DDD (Data Display Debugger) egy szabad grafikus felület, amely számos hibakereső felett használható. A továbbiakban a DDD segítségével vizsgáljuk meg a gyorsrendezés egyik egyszerű C-megvalósítását.
Először is be kell szerezni a DDD egy példányát, majd telepíteni kell. Bináris és forráscsomagokat egyaránt találhatunk, ebben RPM alapú terjesztések esetében az rpmfind.net lehet a segítségünkre, Debian-csomagokat pedig a debian.org segítségével keríthetünk. Cikkem írásakor 3.3.1-es DDD-t és Red Hat 7.3-as terjesztést használtam. A továbbiak során három dolgot feltételezünk:

1. GNU/Linux alapú számítógéppel rendelkezünk, ami be van kapcsolva;
2. ismerjük az alapvető C-fogalmakat, mint tömbök, ciklusok és önhívás (recursion);
3. megfelelő C-fordítóval rendelkezünk, például a GNU GCC-vel. Még ha fogalmad sincs a programozásról, próbáld meg áttanulmányozni a kódot – hasznát látod a későbbiek folyamán.

C. A. R. Hoare 1962-ben vázolta fel a gyorsrendezés algoritmusát, amit negyven év elteltével még mindig széles körben használnak. A gyorsrendezés talán „oszd meg és uralkodj” felfogása miatt lett „gyors”, mivel a nagyobb elemeket kisebbekre osztva szükségtelenné teszi nagy számú összehasonlítás elvégzését. Ellentéte például a legkisebb kiválasztásának elvére épülő rendezés, amely minden elemet összehasonlít az összes többivel. Természetesen ez nem feltétlenül jelenti azt, hogy a gyorsrendezés mindig gyorsabb vagy ez a legjobb rendezési mód, egyszerűen csak jó ismerni. Az írásomban szereplő megvalósítás nem javított és nem bővíthető, kizárólag egész számokból álló tömbökkel működik.
A kód jelentős része ***Brian W. Kernighan*** és ***Rob Pike*** „The Practice of Programming” című művéből származik.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

/* Brian W. Kernighan Øs Rob Pike The
Practice of Programming
* c mß k nyvØnek 32-34. oldalÆr l
*/

/* swap: v[i] Øs v[j] felcserØlØse */
void swap(int v[], int i, int j)
{
       int tmp;
tmp = v[i];
v[i] = v[j];
v[j] = tmp;
}

/* quicksort: a v[0]..v[n-1] elemek n vekv1
   sorrendbe rendezØse */

void quicksort(int v[], int n)
{
       int i = 0, last = 0;
       if (n<=1)
              return; /* semmit nem csinÆl */
       swap(v, 0, rand() % n);
/* az elvÆlaszt  elem mozgatÆsa v[0]-ba */

for(i = 1; i < n, i++)    /* kettØosztÆs */
              if ( v[i] < v[0])
                     swap(v, ++last, i);

swap(v, 0, last);
/* az elvÆlaszt  elem visszaÆll tÆsa */

quicksort(v, last);
/* a kisebb Ørtøkek rendezØse */

quicksort(v+last+1, n-last-1);
/* a nagyobb Ørtøkek rendezØse */

}


void print_array(const int array[], int elems)
{
       int i;
       printf("{");

for(i = 0; i < elems; i++)
              printf("%d ", array[i]);

printf("}\n");

}


#define NUM 9

int main(void)
{
       int arr[NUM] =
           ➥{6, 12, 4, 18, 3, 27, 16, 15, 19};

/* megjegyzØsbe raktuk, hogy elıre lÆtni
   lehessen a rendezØs menetØt
   srand( (unsigned int) time(NULL) ); */

print_array(arr, NUM);
```





```
quicksort(arr, NUM);
print_array(arr, NUM);
return EXIT_SUCCESS;
}
```

Lépésről lépésre

Mentsd a kódot egy ***easy_qsort.c*** nevű állományba. Ezután fordítsd le:

```
$ gcc -Wall -pedantic -ansi -o qsortof -g
  ➥easy_qsort.c
```

A GCC-nek megadott kapcsolók közül a `-g` a legfontosabb, amely hibakeresési hivatkozással (symbol) látja el a kódot. Próbáld lefuttatni a programot, így kiderül, minden rendben van-e:

```
$ ./qsortof

{ 6 12 4 18 3 27 16 15 19 }

{3 4 6 12 15 16 18 19 27 }
```

A kimenet első sora a rendezetlen tömb, a másik pedig a gyorsrendezés futtatása utáni helyzetet tükrözi. Vajon hogyan működik? Hívjuk segítségül új barátunkat, a DDD-t:

```
$ ddd qsortof
```

Ezzel elindul a DDD. Zárd be a felbukkanó ***Tipp*** vagy ***Névjegy*** ablakot, és az első ábrán láthatóhoz hasonlók tárulnak eléd. Nem árt, ha bekapcsolod a sorok számozását. Jelöld be a ***Display Source Line Numbers*** (Forráskódsorok sorszámának megjelenítése) négyzetet, ezt az ***Edit, Preferences, Source*** (Szerkesztés, Beállítások, Forrás) menüpont alatt találod.
Most már csak egy töréspontot kell hozzáadnunk, és indulhat a hibakeresés.
Jelöld ki a semmittevő sort úgy, hogy az oldalt megjelenő sorszámára kattintasz. Ezután a () gombnál kattints a ***Set/Delete Breakpoint*** (Töréspont hozzáadása/törlése) parancsra, és a parancsablakban kattints a ***Run*** (Futtatás) gombra.
Ekkor egy piros stopjel jelenik meg a töréspontot tartalmazó sorban, illetve egy zöld nyíl is felbukkan ugyanitt (ez jelzi az éppen végrehajtandó kódrészletet). Jelenítsünk meg néhány belső adatot a DDD segítségével. Először válaszd a ***Data, Display Local Variables*** (Adat, Helyi változók megjelenítése) menüpontot. A második legyen a ***Data, Display Arguments*** (Adat, Átadott értékek megjelenítése), majd következzen a ***Status, Backtrace*** (Állapot, Visszakövetés). Végül gépeld be a `graph display v[0]@n` parancsot a konzolablakba, majd nyomd le az ENTER billentyűt. Ekkor felbukkannak a `v[]` tömb elemei, 0-tól n-ig (lásd a második ábrát).

Vajon mi történik odabent?

A töréspontot úgy adtuk meg, hogy ha az átadott tömb csak egyetlen elemet tartalmaz vagy üres, akkor önhívással folyik, a vizsgálat feltétlenül szükséges, hiszen ha egy vagy nulla elemet adunk át, akkor le kell állítani az önhívást (erről később még lesz szó).
A vizsgálat után fogunk egy elválasztó elemet, és a tömb elejére mozgatjuk. Kattints a ***Next*** (Tovább) gombra, míg a zöld nyíl a `swap` függvény hívásáig nem ugrik. A ***Next*** gombra kattintva sorról sorra haladhatsz, az alfüggvények meghívása nélkül; a ***Step*** (Lépés) gombra kattintva viszont az alfüggvényekbe is belépsz. Kattints még egyszer a ***Next*** (Tovább) gombra, és figyeld, hogy mi történik.
Az én gépem felcserélte `v[]` nulladik és első elemét, vagyis a `rand() % n` hívás 1-et adott vissza. Ha többször is megvizsgálod a programot, észre fogod venni, hogy a `rand() % n` mindig 1-et ad. Véletlen létére elég egyhangú, nemde? Ebben a példában az `srand()` hívás megjegyzésbe tételével elmarad az álvéletlenszám-generátor megbolygatása, ezért a `rand()` mindig megjósolható eredményt ad.
Az így meghatározott elválasztó elemet használjuk fel arra, hogy a számokat nála kisebbekre és nagyobbakra osszuk. Az elválasztó elem azért került a `v[0]` helyre, mert amíg a teljes tömböt át nem vizsgáltuk, addig nem ismerjük a pontos helyét. A kettéosztást végző ciklus végiglépked a tömb összes elemén, és összehasonlítja őket az elválasztó elemmel; ez esetemben a 12 volt. A `last` elem növelése előzetesen történik (preincre-

ment), így az egyes cellában lévő elemet önmagával cseréljük fel. Igen, én is tudom, feleslegesen. Az algoritmus hatékonyabbá tétele kellemes feladat az önmaguk sanyargatását élvező olvasók számára. Még meg akartam említeni, hogy az ***Interrupt*** (Megszakítás) gombra kattintva a program futását bármikor megszakíthatjuk, majd a ***Run*** (Futtatás) gombbal újraindíthatjuk. Kattintgass a ***Next*** (Tovább) gombra, amíg az `i` értéke 3, a `last` változóé pedig 2 lesz; ehhez figyeld a ***Locals*** (Helyi változók) ablak tartalmát. A kettéosztó hurok `if` elágazása most a 18-at és a 12-t hasonlítja össze. A vizsgálat sikertelenül zárul (kattints), ezért az `i`-t növeljük (kattints), a `last` változó értéke 2 marad.
Maradjunk szoros kapcsolatban a ***Next*** gombbal, amíg az `i` értéke 9 nem lesz. A tömb most a következőképpen néz ki:

```
{ 12 6 4 3 18 27 16 15 19 }
```

Újra kattintás, és a 12, amely eddig a kisebb számok között tévelygett, helyet cserél a 3-mal.
Miután az elválasztó elem az eredeti helyére kerül, önhívással újra meghívjuk a `quicksort` függvényt, miközben átadunk neki egy mutatót `v[0]`-ra, és közöljük vele, hogy egy háromelemű – a kisebb számokat tartalmazó - tömböt fog kapni. Ezután újra meghívjuk, de ezúttal `v[4]`-re mutatunk és ötelemű tömböt adunk neki, most a nagyobb számokkal. Újra és újra meghívjuk a `quicksort` függvényt, egészen addig, ameddig az átadott tömbökben csupán egyetlen elem marad. Ekkor megindul a hívások visszatérése – elsőként az utolsó –, és mire a `main` függvénybe érkezünk, rendezett tömböt kapunk. Kifújhatjuk magunkat!

Ami jól jöhet

Ha elég mélyen elmerülsz az önhívású `quicksort` hívások mocsarában, egy ponton a `v[0]@n` ablak el fog tűnni. Egy megfelelő gombot létrehozva egy pillanat alatt újra előcsalogathatod. Gombot a ***Commands, Edit Buttons..., Data Buttons*** (Parancsok, Gombok szerkesztése, Adatgombok) menüpont segítségével hozhatsz létre. A szövegbeviteli mezőbe gépeld be az alábbiakat:

```
graph display v[0]@n // Varray
```

Egy ***Varray*** nevű gombnak kell megjelennie a ***Data*** (Adat) ablak tetején. Ha a `v[0]@n` ablak olvas (letiltódik), kattints rá az egér jobb gombjával, és válaszd az ***Undisplay*** (Elrejtés) parancsot. Ezután kattints a ***Varray*** gombra. Ha az ablak továbbra sem jelenik meg, próbálj néhányszor rákattintani a ***Next*** (Tovább) gombra, és próbálkozz újra.

A múlt bűvöletében

Korábban bekapcsoltattam veled a ***Backtrace*** (Visszakövetés) ablakot. Miközben `quicksort()` hívások tornyosulnak feletted, a ***Backtrace*** ablak soraira kattintva érdekes lehet belekukkantani a verem tartalmába. Láthatod, hogyan jutottál el a pillanatnyi függvényhívási környezethez, és a futás más időpontjaiban milyen adatok voltak a veremben.

A gépi kód ablak

Ha eléggé elkábultál, tégy egy kísérletet a ***View, Machine Code*** (Nézet, Gépi kód) paranccsal, amellyel megjelenítheted az éppen folyó műveletek assembly utasításait.

További érdekes algoritmusok és adatszerkezetek

A DDD csodálatos látvány, amikor láncolt listákat vagy más adatszerkezeteket jelenít meg. Próbáld ki!

Linux Journal 2003. január, 105. szám

Adam Monsen
Érdeklődő természetű orvostanhallgató, jelenleg a washingtoni classmates.com kezdeményezés programozója. Szeret zongorázni, szörfözni, amatőr artista. GNU/Linux Red Hat 7.3 alapú gépén Perl, Java és néha C nyelven kódolgat.

KAPCSOLÓDÓ CÍMEK

A Limit Theorem for „Quicksort", *Uwe Rösler*, 1999. május 2.:
➲ http://citeseer.nj.nec.com/rosler99limit.html
DDD-honlap ➲ http://www.gnu.org/software/ddd
Quicksort, *C.A.R. Hoare*, 1962. április
➲ http://www3.oup.co.uk/computer_journal/hdb/Volume_05/Issue_01/050010.sgm.abs.html
Brian W. Kernighan és *Rob Pike* "The Practice of Programming" 32-34. oldal (ISBN: 0-201-61586-X)
GDB honlap ➲ http://www.gnu.org/software/gdb