Matemàtica computacional i analítica de dades
Algorítmia i combinatòria en grafs...
Curs 2020–21

4 Ordenar amb `qsort`

La instrucció `qsort` permet ordenar taules a partir d'un criteri de comparació que se li entri sense necessitat de duplicar-les. Els arguments de `qsort` són:

```
void qsort (void* taula, size_t elements_taula, size_t mida_element,
            int (*comparador)(const void*,const void*));
```

i per tant necessita una funció `comparador` que faci la comparació.

El codi següent és un exemple senzill d'ús del `qsort`. Fixeu-vos en la programació de `comparacio` (línies 6–9). A priori, no sap quin tipus de apuntador rebrà (el tipus de `a` i `b`), i, per tant, a la declaració, li arriba un apuntador tipus `void`, i cada cop que s'utilitza un argument ha de dir-li que és un *double* (`double *`).

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  double valors[] = { 5.6, 6.1, 1.2 , 2.3 , 9.8 };
5
6  int comparacio (const void * a, const void * b) {
7      if(*(double*)a<*(double*)b) return 1;
8      else return -1;
9  }
10
11 int main () {
12    int n;
13
14    printf("La llista abans d'ordenar es: \n");
15    for( n = 0 ; n < 5; n++ ) {
16       printf("%g ", valors[n]);
17    }
18    printf("\n");
19
20    qsort(valors, 5, sizeof(double), comparacio);
21
22    printf("Despres d'ordenar, la llista es: \n");
23    for( n = 0 ; n < 5; n++ ) {
24       printf("%g ", valors[n]);
25    }
26    printf("\n");
27
28    return(0);
29 }
```

Volem aplicar aquesta funció per a ordenar les llistes donades en format taula i en format llista enllaçada.

4.1 `qsort` amb taules

Suposem que tenim carregada la taula `alumnes` de la Pràctica 2 i suposem que volem ordenar-la per nota mitjana.

La funció de comparació següent hauria de funcionar:

```
int comparaciomitjana (const void * a, const void * b)
{
if(((Alu *)a)->notes[4]>((Alu *)b)->notes[4]) return 1;
else return -1;
}
```

Per cada exercici guardeu un fitxer diferent amb el nom `Pr4ExY.c`, on `Y` correspon al número de fitxer. Per exemple, el l'exercici següent s'ha de dir `Pr4Ex411.c`.

Exercici 4.1.1: Considereu una de les rutines de la Pràctica 2 que llegeixi la llista d'alumnes i, utilitzant la funció `qsort` i la `comparaciomitjana` que acabem de veure, imprimiu la llista ordenada per nota mitjana.

Exercici 4.1.2: Feu una funció `comparacioniu` i, a partir de la rutina anterior, reordeneu la llista i imprimiu-la ordenada per NIU.

4.2 Mostrar o accedir a una taula en més d'un ordre

Si volem mostrar els elements d'una taula en un ordre diferent, o en diferents ordres, no cal que reordenem la taula cada cop, si no que podem crear vectors d'índex on cada element del vector apunti a la posició de memòria d'un element de la taula. Aquest vector el podem reordenar, i si seguim l'ordre que ens dóna el nou ordre, podrem accedir als elements de la taula en aquest nou ordre i, en particular, obtenir un llistat o fer una cerca en un ordre diferent al que estan guardades les dades.

En aquest cas, la funció `qsort` rebrà aquest vector i el reordenarà. Per altra banda, la funció `comparacio` haurà de llegir cada component d'aquest vector, que serà l'adreça d'un alumne. Llavors, a partir de dues posicions diferents d'aquest vector, haurà de retornar `1` si la primera és més gran que la segona i `-1` altrament.

A aquests exercicis suposem que carreguem la llista de notes de la Pràctica 2 tal i com està a l'arxiu, per tant, sense reordenar-la després ni per NIU, ni per nota mitjana.

Exercici 4.2.1: Definiu la variable `indexmitjanes` com un punter de punters tipus `Alu`, reserveu la memòria necessària i guardeu a cada posició l'adreça on hi ha les dades de cada alumne a la taula.

Feu també una funció `imprimirdesdindex` que, a partir del vector de les adreces (i el nombre d'alumnes) mostri el llistat d'abans en aquest ordre.

Feu una funció `comparacioapartirdindex` que, tenint en compte que li arribaran posicions del vector de l'índex, retorni `1` o `-1` depenent de si la nota mitjana del primer alumne és més gran que la del segon o no.

Ordeneu el vector `indexmitjanes` amb el criteri `comparacioapartirdindex` i mostreu el nou ordre per pantalla.

Exercici 4.2.2: El nombre d'índexs que es poden associar a una llista és arbitrari. Fins i tot si la llista ja està ordenada per l'identificador pot ser interessant mantenir un índex basat en aquest camp de cada bloc. En particular, si teniu un vector `indexniu` tal que `indexniu[i]` tingui la posició de memòria del `i`-èssim NIU considerats de forma ordenada, podreu realitzar les cerques d'un identificador dins la llista de forma binària i no seqüencial.

Creeu un índex nou `indexniu` i ordeneu-lo segons el NIU de cada alumne (caldrà definir la funció comparació per aquest cas).

Feu una funció nova `cercabinarianiu` que a partir d'un NIU (i els arguments que facin

falta), faci una cerca binària i retorni la posició de memòria on es troben les dades de l'estudiant que té aquest NIU.
Per comprovar-ne el funcionament, modifiqueu el programa per a que pregunti un NIU i mostri la informació de les notes corresponents a aquest NIU.

Instruccions finals

Quan acabeu la pràctica, feu el lliurament dels fitxers de codi (que tenen els noms de la forma `Pr4ExY.c` segons el que hem indicat anteriorment) a través del Campus Virtual des de l'apartat de lliuraments de l'assignatura. Recordeu que al principi de cada fitxer hi ha d'haver el vostre nom i NIU.