Les curses de cavalls

Josep Maria Ribó

Juny de 2000

1 Enunciat del problema

Es tracta de dissenyar un estructura de dades que sigui capaç de gestionar curses de cavalls.

Les curses s'identifiquen per un identificador de cursa (que és una cadena). De les curses ens interessen, a més a més, la data de celebració de la cursa, i les posicions d'arribada dels diferents cavalls participants.

Dels cavalls ens interessa el seu nom (atribut que identifica el cavall), la seva data de naixement i el genet que habitualment el munta.

Ens interessa que l'estructura de dades que dissenyem ens permeti accedir ràpidament a:

- Les posicions d'arribada dels cavalls a la cursa que té un determinat identificador i
- Totes les curses a les que ha participat el cavall amb un determinat nom.

(Proposem, per tant, dues *consultes* a les dades).

2 Plantejament de la solució

En l'enunciat d'aquest problema ens apareixen tres conceptes clars:

- **Cavall**, identificat per nom i amb genet i data de naixement com a atributs.
- **Cursa**, identificat per un identificador de cursa i amb l'ordre d'arribada dels cavalls participants i la data de celebració com a atributs.
- **Gestor de curses**, l'estructura de dades que s'encarregarà de mantenir organitzadament tota la informació relativa als cavalls i a les curses de manera que s'hi pugui accedir eficientment.

A més a més serà útil disposar d'una **llista de cadenes** ja que molts cops presentarem la informació resultat d'una consulta com una llista de cadenes (d'identificadors de cursa, en un cas i de noms de cavalls, en un altre).

3 La classe LlistaCadenes

Per la classe `LlistaCadenes` serà útil disposar de les operacions següents:

- `void LlistaCadenes()` Crea una llista buida

- `void afegirCadena(c,err)` Afegeix la cadena `c` al final de la llista. `err` s'activa si no hi ha prou espai a la llista.
- `void afegirCadenaPosicio(c,i,err)` Afegeix la cadena `c` a la posició `i` de la llista. `err` s'activa si no hi ha prou espai a la llista o no existeix la posició `i`.
- `buidar()` Treu de la llista tots els elements que contenia.
- `void posicionarInici()` Situa un cursor a la primera posició de la llista per tal de poder-la recórrer.
- `void obtenirActual(c)` Obté la cadena `c` sobre la qual està posicionat el cursor de la llista.
- `void avançar()` Avança el cursor a la posició següent de la llista.
- `bool final()` Retorna cert si el cursor ha superat ja el darrer element de la llista.

La representació tindria un aspecte com el que segueix:

```
class LlistaCadenes{

  Cadena v[MAXCAD];
  int ncad;
  int cursor;
public:

...

};
```

Exercicis:

1. L'especificació que hem presentat de les operacions de la classe no està completa (hi ha alguns possibles errors que no es contemplen...). Completa-la.
2. Implementa les operacions.
3. Enriqueix-la amb una nova operació: `inserirOrdenat(c,err)` que col.loca la cadena `c` al lloc que li correspon de la llista. Quina precondició hauria de tenir aquesta operació. Especifica-la i implementa-la.

4 Les classes Cavall i Cursa

Les classes `Cavall` i `Cursa` són força directes. Només cal destacar que fem els seus atributs públics per la seva simplicitat. Seria igualment correcte fer-los privats i dotar les dues classes de les operacions corresponents per tal de construir-los i consultar-los.

```
class Cavall{
public:

 Cadena nom;
```

```
 Data dnaix;
 Cadena genet;

};

class Cursa{
public:

Cadena idcursa;
Data datacur;
int numcavalls;
Cadena posicions[MAXCVCUR];
};
```

`MAXCVCUR` és el nombre màxim de cavalls que poden participar en una cursa.

Aquestes dues classes possiblment les situaríem al mateix fitxer que la `GestorCurses`

5 La classe GestorCurses

Aquesta classe és la que s'encarrega de mantenir tota la informació de curses i cavalls organitzada convenientment per tal de poder-la consultar de manera eficient. Preguntem-nos en primer lloc, quines operacions necessitarà per poder fer això.

5.1 Especificació de la classe

La classe `GestorCurses` ha de disposar de les operacions següents:

- `GestorCurses()` Crea un gestor de curses sense cap cursa ni cap cavall.
- `void afegirCavall(c,err)` Afegeix el cavall `c` al gestor. Si no hi ha prou espai o ja hi era, s'activa el booleà `err`.
- `void afegirCursa(c,err)` Afegeix la cursa `c` al gestor. Si no hi ha prou espai o ja hi era, s'activa el booleà `err`.
- `int consulNumCavalls()` Retorna el nombre de cavalls emmagatzemats al gestor.
- `int consulNumCurses()` Retorna el nombre de curses emmagatzemades al gestor.
- `void consulCavall(nomc, c, err)` Retorna les dades del cavall que té per nom `nomc`. `err` s'activa si no hi ha cap cavall amb el nom `nomc`.
- `void consulCursa(idc, c, err)` Retorna les dades de la cursa que té per identificador `idc`. `err` s'activa si no hi ha cap cursa amb l'identificador `idc`.
- `void consulCursesCavall(c,l,err)` La llista `l` conté els identificadors de totes les curses on ha corregut el cavall de nom `c`. `err` s'activa si `l` no té prou capacitat per encabir totes les curses a les que ha participat el cavall `c`.
- `void consulPosicionsCursa(idc,l,err)` La llista `l` conté tots els cavalls (en l'ordre d'arribada) de la cursa amb identificador `idc`. `err` s'activa si `l` no té prou capacitat per encabir tots els cavalls participants en una cursa.

Les tres primeres operacions són necessàries per construir un gestor de curses amb cavalls i curses. Les dues darreres corresponen a les operacions consultores l'eficiència de les quals volem optimitzar i les del mig són operacions útils que és bo que ofereixi la classe.

Exercici:

4. Completar l'especificació de les operacions de la classe.

5.2 Implementació de la classe

Per tal d'emmagatzemar cavalls i curses necessitarem dos vectors amb la definició seg üent:

```
Cavall vcavalls[MAXCV];
int ncavalls;

Cursa vcurses[MAXCR];
int ncurses;
```

`MAXCV` és el nombre màxim de cavalls que podem tenir i `ncavalls` és el nombre de cavalls que tenim emmagatzemats realment al gestor en un moment determinat (`ncavalls` és el primer índex lliure de `vcavalls`).

`MAXCR` és el nombre màxim de curses que podem emmagatzemar i `ncurses` és el nombre de curses que tenim emmagatzemades realment en un moment determinat (`ncurses` és el primer índex lliure de `vcurses`).

El criteri habitual per fer la implementació d'una classe és fer-la en termes d'unes estructures de dades que ens permetin optimitzar el cost temporal de les seves operacions consultores.

En aquest cas, les dues consultores crítiques són `consulPosicionsCursa` i `consulCursesCavall`.

La primera ens demana un accés ràpid per identificador de cursa i la segona per nom de cavall. Això ens suggereix que, a més a més dels dos vectors `vcurses` i `vcavalls`, necessitarem un vector ordenat per identificador de cursa i un altre ordenat per nom de cavall. L'algorisme de cerca dicotòmica ens permetrà accedir a la informació d'aquests dos vectors amb gran eficiència. Veiem amb detall quina serà la informació que contindrà cadascun d'aquests dos vectors addicionals que proposem:

- **Vector ordenat per identificador de cursa**: `videntif`

  Per a tota `i`: $0 \leq$`i`$<$`ncurses`, `videntif[i]` contindrà dues coses:

  - L'i-èsim identificador de cursa (en ordre alfabètic).
  - L'índex del vector `vcurses` on es troba la cursa amb aquest identificador.

  Aquest vector tindrà un element per cada cursa (en total: `ncurses` elements).

  La seva definició és la següent:

  ```
  Parell videntif[MAXCR];
  ```

  On, la classe `Parell` ve definida com segueix:

  ```
  class Parell{
  public:
   Cadena element;
   int index;
  };
  ```

Amb aquest vector, l'accés als cavalls que han participat en la cursa `idc` és molt ràpid:

Es fa una cerca dicotòmica a `videntif` de la cursa amb identificador `idc`, la qual retorna un índex `k`. Si `videntif[k].elem=idc`, els cavalls d'aquesta cursa seran a `vcurses[videntif[k].index].posicions`.

- **Vector ordenat per nom de cavall**: `vnomc`

  Aquest vector tindrà `nvnomc` elements.

  Per a tota `i`: $0 \leq$`i`$<$`nvnomc`, `vnomc[i]` contindrà dues coses:

  - L'i-èsim nom de cavall (en ordre alfabètic). Cada cavall apareixerà un cop per cada cursa en la que participi.
  - L'índex del vector `vcurses` on es troba una de les curses on participa el cavall amb aquell nom.

  La seva definició és la següent:

```
Parell vnomc[MAXCR*MAXCVCURSA];
int nvnomc;
```

  Un cavall amb nom `nomc` haurà participat a totes les curses indicades als elements del vector `vnomc` que tinguin `nomc` com a nom de cavall.

Notes:

- MAXCVCURSA és el nombre màxim de cavalls que poden participar en una cursa.
- No és adient que el vector `vcurses` estigui ordenat per identificador (per tal d'estalviar `videntif`) ja que les successives insercions de curses modificarien la ubicació dels elements d'aquest vector i això malmetria el contingut de `vnomc`).

5.3 Exemple

La figura 1 conté un exemple amb els cavalls:

```
pollos 12-05-1980 pepet
valent 11-06-1983 anneta
estressat 01-02-1980 lluiset
espitat 12-10-1987 carles
```

i les curses (en aquest ordre d'inserció):

```
20000c 23-09-2000 {valent estressat pollos}
40000c 21-12-2000 {estressat valent}
30000c 12-01-2000 {espitat valent}
10000c 12-08-2000 {pollos valent}
```

La inserció d'una nova cursa:

```
35000c 01-01-2000 {espitat pollos}
```

ens obligarà a fer les insercions que s'assenyalen amb fletxes a la figura 1: una al vector `vcurses`, una altra a `videntif` i, al vector `vnomc` cal fer una inserció per cada cavall que participa en aquesta cursa.

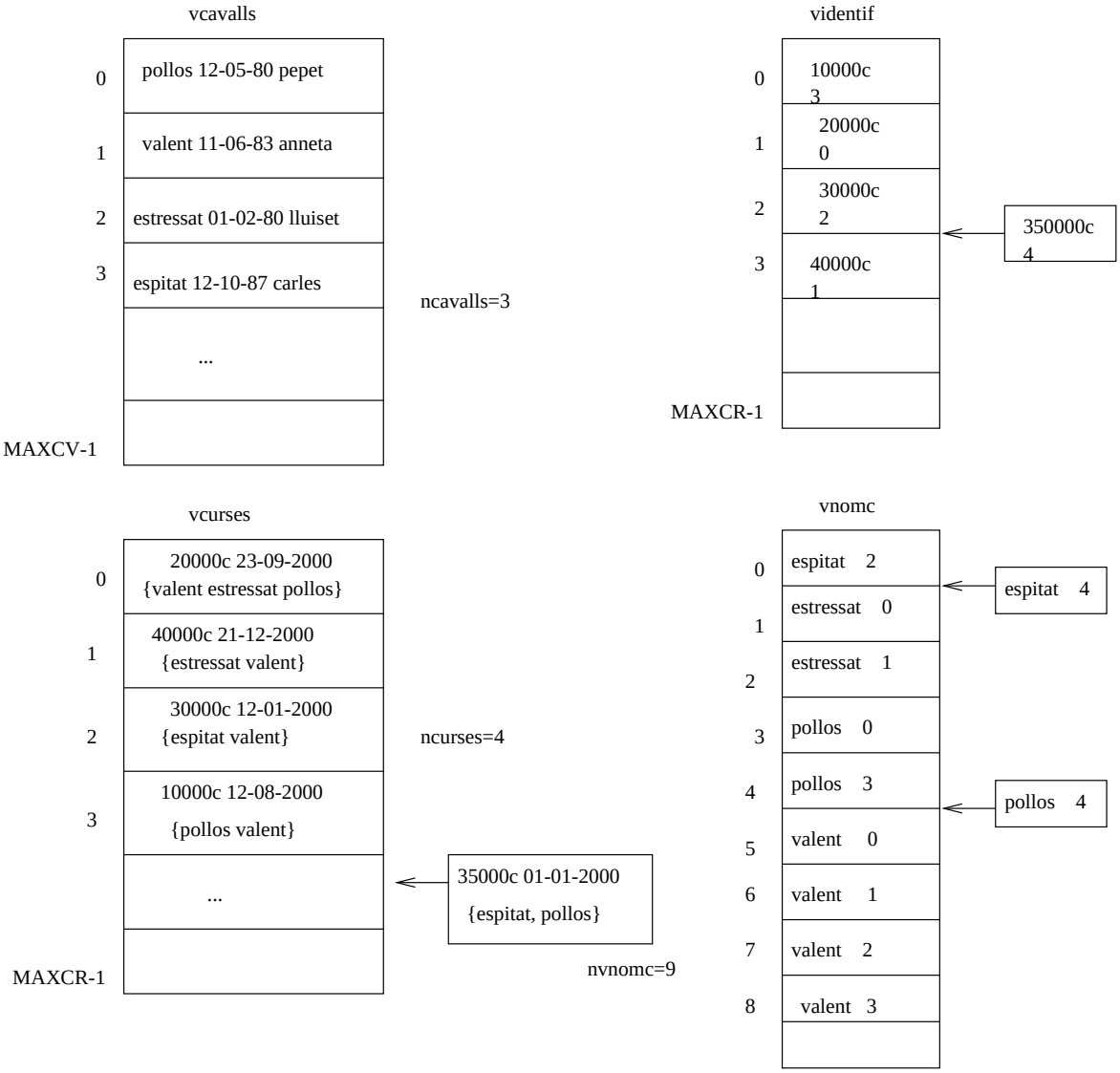

Figura 1: Visualització de la representació de la classe `GestorCurses`

5.4 Codi de les operacions

```
//Fitxer gestor.h

#include "cadena.h"
#include "llistacadenes.h"


class Parell{
public:
 Cadena element;
 int index;
};


class Cavall{
public:

 Cadena nom;
 Data dnaix;
 Cadena genet;

};

class Cursa{
public:

Cadena idcursa;
Data datacur;
int numcavalls;
Cadena posicions[MAXCVCUR];
};

class GestorCurses{

  Cavall vcavalls[MAXCV];
  int ncavalls;

  Cursa vcurses[MAXCR];
  int ncurses;

  Parell videntif[MAXCR];

  Parell vnomc[MAXCR*MAXCVCURSA];
  int nvnomc;

public:

  GestorCurses();
  void afegirCavall(const Cavall& c, bool& err);
  void afegirCursa(const Cursa& c,bool& err);
  int consulNumCavalls();
```

```
  int consulNumCurses();
  void consulCavall(const Cadena& nomc, Cavall& c, bool& err);
  void consulCursa(const Cadena& idc, Cursa& c,bool& err);
  void consulCursesCavall(const Cadena& c,LlistaCadenes& l, bool& err);
  void consulPosicionsCursa(const Cadena& idc,LlistaCadenes& l, bool& err);
};


//Fitxer gestor.cpp

#include "gestor.h"
#include "cadena.h"
#include "llistacadenes.h"


GestorCurses::GestorCurses()
{
 nvnomc=0;
 ncurses=0;
 ncavalls=0;
}


void GestorCurses::afegirCavall(const Cavall& c, bool& err)
{
  int i;
  bool trobat;

  if (ncavalls==MAXCV) err=true;
  else{

     i=0;
     while (!trobat && i<ncavalls){
       trobat=(vcavalls[i].nom==c.nom);
       i=i+1;
     }
     if (trobat) err=true;
     else{
       vcavalls[ncavalls]=c;
       err=false;
       ncavalls=ncavalls+1;
     }
 }

}


void GestorCurses::afegirCursa(const Cursa& c,bool& err)
{
  bool trobat;
  int i;

  if (ncurses==MAXCR) err=true;
```

```
  else{
       i=0;
       while(!trobat && i<ncurses){
         trobat=(vcurses[i].idcursa==c.idcursa);
         i=i+1;
       }
       if (trobat) err=true;
       else{

      //insercio a vcurses:

         vcurses[ncurses]=c; //convenient sobrecarregar = a Cursa

      //insercio a videntif:

         afegirOrdenat(videntif,ncurses,c.idcursa,ncurses);

      //insercio a vnomc (cal fer una insercio per cada cavall de la
      //cursa c:

         for(i=0;i<c.numcavalls;i++){

           afegirOrdenat(vnomc,nvnomc,c.posicions[i],ncurses);
           nvnomc=nvnomc+1;
         }

      ncurses=ncurses+1;
      err=false;
}

void GestorCurses::consulPosicionsCursa(const Cadena& idc,
                                        LlistaCadenes& l,
                                        bool& err)
{
  int i,k,ind;

  l.buidar();   //ens assegurem que l esta buida.
  cercaDicotomica(videntif,ncurses,idc,k);
  err=false;

  if (k<0) ; //no s'ha trobat l'identificador. no fer res.
  else if (!(videntif[k].element==idc)) ; //idem.
  else{
    ind=videntif[k].index;
    i=0;
    while (i<vcurses[ind].numelements && !err){

        l.inserirCadena(vcurses[ind].posicions[i],err);
        i=i+1;
    }
  }
}
```

```
void GestorCurses::consulCursesCavall(const Cadena& c,
                                      LlistaCadenes& l,
                                      bool& err)
{
   bool mateixCavall;
   int k;

   l.buidar();
   cercaDicotomica(vnomc,nvnomc,c,k);
   err=false;

   if (k<0) ;
   else if (!(vnomc[k].element==c)) ;
   else{

     mateixCavall=true;
     while(mateixCavall && k>=0 && !err){

      if(c==vnomc[k].element){
        l.inserirCadena(vcurses[vnomc[k].index].idcursa,err);
        k=k-1;
      }
      else{ mateixCavall=false;}
    }
  }

}

void GestorCurses::inserirOrdenat(Parell v[], int n,
                                  const Cadena& x,
                                  int ind)
{

  //EXERCICI

  //P: v es un vector inicialitzat amb n elements.
  //   n es menor que la mida del vector.

  //Q: S'ha inserit el parell <x,ind> al vector de parells v que te
  //   n elements. L'ordenacio es fa ordenada segons el camp element.

  //Nomes cal retocar lleugerment l'accio inserirOrdenat d'INIPROG
  //per tal que pugui treballar amb vectors de parells.

  //Notar que es una operacio privada de GestorCurses
}

void GestorCurses::cercaDicotomica(Parell v[], int n,
                                   const Cadena& x,
                                   int& k)
{
```

```
  //EXERCICI

  //P: v es un vector inicilaitzat amb n elements i ordenat pel camp
  //   'element'.
  //Q: k es l'index de v que conte el mes gran entre els valors del
  //   camp 'element' que siguin mes petits o iguals que x. (k pot ser -1).

  //Nomes cal retocar lleugerment l'accio cercaDicotomica d'INIPROG
  //per tal que pugui treballar amb vectors de parells.

  //Notar que es una operacio privada de GestorCurses
}
```

Exercicis:

5. Dissenyar les accions `cercaDicotomica` i `inserirOrdenat` a partir de l'adaptació lleugeríssima de les que hem vist a l'assignatura d'INIPROG. L'adaptació s'ha de fer per tal que les accions funcionin correctament sobre un vector de parells cadena-index , on l'ordenació es fa pel camp cadena (a INIPROG operaven sobre vectors de naturals).

6 Utilització de la classe

Entre les diverses accions que podem dissenyar per utilitzar la classe `GestorCurses`, fem-ne una que rebi com a paràmetre un text que conté un grapat de curses amb el seguent format:

- Cada paràgraf, una cursa.
- La primera frase de cada paràgraf: identificador de cursa (un sol mot).
- La segona frase de cada paràgraf: la data (tres digits).
- La tercera frase: els cavalls participants en l'ordre d'arribada.

I retorni un objecte de la classe `GestorCurses` que podrà ser consultat/manipulat per alguna altra acció.

```
//FITXER usuari.cpp

#include "gestor.h"
#include "cadena.h"
#include "text.h"
#include "paragraf.h"



void adquirirCurses(Text& curses, GestorCurses& gc)
{

  curses.posicionarInici();
```

```
  while(!curses.finalText()){

    carregarCursa(curses,gc);
    curses.posicionarSegParagraf();
 }

}

void carregarCursa(Text& curses,GestorCurses& gc)
{
  Cursa c;
  int i;

  curses.obtenirParaula(c.idcursa,err);
  curses.posicionarSegFrase();

  llegirDataCursa(curses,c.datacur);
  curses.posicionarSegFrase();

  i=0;
  while(!curses.finalFrase()){

    curses.obtenirParaula(c.posicions[i],err);
    i=i+1;
  }
  c.numcavalls=i;
  gc.afegirCursa(c,err);
}


void llegirDataCursa(Text& t, Data& d)
{

  t.obtenirParaula(diaCadena);
  dia=(diaCadena.posicio(0)-'0')*10+(diaCadena.posicio(1)-'10');
  d.assignDia(dia);

  t.obtenirParaula(mesCadena);
  mes=(mesCadena.posicio(0)-'0')*10+(mesCadena.posicio(1)-'10');
  d.assignMes(mes);

  t.obtenirParaula(anyCadena);
  dia=(anyCadena.posicio(0)-'0')*1000+(anyCadena.posicio(1)-'10')100+
       (anyCadena.posicio(2)-'0')*10+(anyCadena.posicio(3)-'0');
  d.assignAny(any);
}
```

Al codi anterior, no hi ha un tractament acurat dels errors.

Notem un detall fonamental:

La informació de l'objecte `curses` és la mateixa que la de l'objecte `gc` però aquesta informació a `gc` està **estructurada** de manera que, a diferència de a `curses`, es pot consultar eficientment.

7 Problemes addicionals

6. Enriquir la classe GestorCurses amb una operació que obtingui tota la informació sobre una cursa que es va celebrar en una determinada data (en general, si acceptem que en una data es pot celebrar més d'una cursa, voldrem la informació de totes elles). Aquesta operació, si es vol que sigui eficient, obligarà a ampliar la representació de la classe.

7. Dissenyar una acció que llegeixi de l'entrada estàndar un seguit de curses i de cavalls amb un determinat format i les vagi afegint a un objecte de la classe GestorCurses.

8. En afegir una cursa no es té en compte que tots els cavalls que participen a la cursa existeixin realment al gestor. Modifica convenientment l'acció `afegirCursa` de manera que aquest aspecte sí es controli.