Diseño Modular III

Josefina Sierra Santibáñez

5 de marzo de 2017

Mejoras en las implementaciones de Cjt_estudiants y Estudiant

Operación pública y estática de la clase Estudiant para determinar si el DNI de un objeto de la clase Estudiant es menor que el DNI de otro objeto de la clase Estudiant.

```
static bool dni_menor(const Estudiant& e1,
                      const Estudiant& e2);
/* Pre: cert  */
/* Post: el resultat indica si el DNI d'e1
         es mes petit que el d'e2 */
```

Uso de la operación **sort** de la biblioteca <algorithm> para ordenar una parte de un vector, en lugar del vector entero.

```
void Cjt_estudiants::ordenar_cjt_estudiants() {
  sort(vest.begin(), vest.begin() + nest,
       Estudiant::dni_menor);
}
```

Nueva versión de la búsqueda dicotómica

```
static int cerca_dicot(const vector<int>& v,
                       int esq, int dre, int x) {
/*Pre: v.size()=n > 0, v[esq...dre] esta ordenat
creixentment, 0<=esq<=n, -1<=dre<n, esq<=dre+1 */
/* Post: si x esta a v[esq ... dre], retorna la
posicio que hi ocupa; si no, retorna la posicio
que hauria d'ocupar */
  int i;
  bool trobat = false;
  while (not trobat and esq <= dre) {
    i = (esq + dre)/2;
    if (x < v[i]) dre = i - 1;
    else if (x > v[i]) esq = i + 1;
    else trobat = true;
  }
  if (trobat) return i;
  else return esq; // posicio hauria d'ocupar
}
```

Nueva versión de consultar

```
void consultar_estudiant(int x, bool& trobat,
                         Estudiant& e) const;
/* Pre: cert */
/* Post: si existeix un estudiant al p.i. amb
DNI=x, trobat es true i e es aquest estudiant;
altrament trobat es false */
```

Utilizamos la versión anterior de la búsqueda dicotómica para

- determinar si `vest[0...nest-1]` contiene un estudiante con DNI igual a `x`
- averiguar la posición donde se encuentra el estudiante con DNI igual a `x` en `vest[0...nest-1]`, si hay alguno

Nueva versión de `modificar`

```
void modificar_estudiant(const Estudiant& e,
                         bool& trobat);
/* Pre: cert */
/* Post: si el p.i. original conte un estudiant
amb el DNI d'e, trobat es true i aquest estudiant
ha estat substituit per e al p.i.; si no, trobat
es false i el p.i. es igual al p.i. original */
```

Utilizamos la versión anterior de la búsqueda dicotómica para

- determinar si en `vest[0...nest-1]` hay un estudiante con el mismo DNI que e
- averiguar la posición donde se encuentra en `vest[0...nest-1]` el estudiante con el mismo DNI que e, si hay alguno

Nueva versión de afegir

```
void afegir_estudiant(const Estudiant &est,
                      bool& trobat);
/* Pre: el p.i. no esta ple */
/* Post: si el p.i. original conte un estudiant
amb el dni d'est trobat es true; si no, trobat
es false i s'ha afegit l'estudiant est al p.i.*/
```

Utilizamos la versión anterior de la búsqueda dicotómica para

- determinar si `est` está en `vest[0...nest-1]`
- averiguar la posición donde debemos insertar `est` en `vest[0...nest]` si `est` no está en el p.i.

Nueva versión de esborrar

```
void esborrar_estudiant(int x, bool& trobat);
/* Pre: cert */
/* Post: si el p.i. original tenia un estudiant
amb DNI=x, trobat es true i aquest estudiant ha
quedat eliminat del p.i.; si no, trobat es false
i el p.i. es igual al p.i. original */
```

Utilizamos la versión anterior de la búsqueda dicotómica para

- determinar si `vest[0...nest-1]` contiene un estudiante con DNI igual a `x`
- averiguar la posición donde se encuentra el estudiante con DNI igual a `x` en `vest[0...nest-1]`, si hay alguno