

Tipus recursius de dades, IV

Ricard Gavalrà

Programació 2

Facultat d'Informàtica de Barcelona, UPC

Primavera 2018

Aquesta presentació no substitueix els apunts

Contingut

Implementació accedint a la representació

Estructures de dades noves

- Cues ordenades

- Multil·listes

Implementació accedint a la representació

Implementacions amb accés a la representació

- ▶ Avantatge: Eficiència. Assignació d'apuntadors vs. còpia d'estructures
- ▶ Inconvenient: Lligades a una representació. No modularitat
- ▶ Exemple: `sort` com a mètode de la classe `list` a STL

Cerca d'un element en una pila

```
/* Pre: cert */  
/* Post: el resultat indica si x apareix a la pila p.i. */  
bool cerca_pila(const T &x) const;
```

Cerca en una pila: versió iterativa

```
/* Pre: cert */
/* Post: el resultat indica si x apareix a la pila p.i. */
bool cerca_pila(const T &x) const {
    node_pila* act = primer_node;
    /* Inv: cap node anterior a act conté x */
    while (act != NULL) {
        if (act->info == x) return true;
        act = act->seguent;
    }
    return false;
}
```

Exercici: versió amb booleà i sense return dins del bucle

Cerca en una pila: versió recursiva I

```
bool cerca_pila(const T &x) const;
```

Problema: La recursió és ($\text{node} \rightarrow \text{node}$), no ($\text{pila} \rightarrow \text{pila}$)!

Immersió: \rightarrow operació auxiliar, recursiva, amb paràmetre `node*`
la crida inicial fa el pas ($\text{pila} \rightarrow \text{node*}$)

Cerca en una pila: versió recursiva II

```
/* Pre: cert */  
/* Post: el resultat indica si x apareix a la pila p.i. */  
bool cerca_pila(const T &x) const {  
    return cerca_pila_node(primer_node, x);  
}
```

```
/* Pre: cert */  
/* Post: el resultat indica si x apareix a la llista  
         de nodes apuntada per n */  
static bool cerca_pila_node(node_pila* n, const T &x);
```

Atenció a l'static!

Cerca en una pila: versió recursiva III

```
/* Pre: cert */  
/* Post: el resultat indica si x apareix a la llista  
        de nodes apuntada per n */  
static bool cerca_pila_node(node_pila* n, const T &x) {  
    if (n == NULL) return false;  
    else if (n->info == x) return true;  
    else return cerca_pila_node(n->seguent, x);  
}
```

Compte: precondition de l'operador ->

Sumar un valor a tots els elements d'un arbre binari

El plantegem com a nou mètode de la classe arbre binari

```
/* Pre: A és el valor inicial del p.i. */  
/* Post: el p.i. és com A però havent sumat k  
        a tots els seus elements */  
void inc_arbre(const T &k);
```

Sumar un valor a tots els elements d'un arbre binari

```
/* Pre: A és el valor inicial del p.i. */
/* Post: el p.i. és com A però havent sumat k a tots
        els seus elements */
void inc_arbre(const T &k) {
    inc_node(a.primer_node, k);
}

/* Pre: cert */
/* Post: el node apuntat per n i tots els seus següents tenen
        al camp info la suma de k i el seu valor original */
static void inc_node(node_arbre* n, int k) {
    if (n != NULL) {
        n->info += k;
        inc_node(n->segE, k);
        inc_node(n->segD, k);
    }
}
```

Substitució de fulles per un arbre l

Substituir totes les fulles del p.i. que continguin el valor x per l'arbre as

```
/* Pre: A es el valor inicial del p.i. */  
/* Post: el p.i. és com A però havent substituït  
        les fulles que contenen  $x$  per l'arbre  $as$  */  
void subst(int x, const Arbre<int> &as);
```

Substitució de fulles per un arbre ll

```
void subst(int x, const Arbre<int> &as) {  
    subst_node(primer_node, x, as.primer_node);  
}  
  
/* Pre: cert */  
/* Post: els nodes de la jerarquia de nodes que comença  
    al node apuntat per n són tals que si eren una fulla  
    amb x a info han estat substituïts per una còpia de la  
    jeraquia de nodes que comença al node apuntat per ns */  
static void subst_node(node_arbre* &n, int x, node_arbre* ns);
```

Substitució de fulles per un arbre III

```
static void subst_node(node_arbre* &n, int x, node_arbre* ns) {  
    if (n != NULL) {  
        if (n->info == x and n->segE == NULL and n->segD == NULL) {  
            delete n; // equivalent aquí a esborra_node_arbre(n);  
            n = copia_node_arbre(ns);  
        } else {  
            subst_node(n->segE, x, ns);  
            subst_node(n->segD, x, ns);  
        }  
    }  
}
```

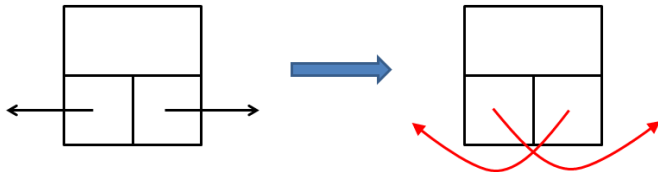
Atenció al pas de n per referència! Alternativa?

Reversar una llista

```
/* Pre: cert */  
/* Post: el p.i. té els mateixos elements que a l'inici però  
        amb l'ordre invertit i el seu punt d'interés apunta  
        al mateix element */  
void revessar();
```

1. Solució amb ops. de la classe: insert, còpies de node ...
2. Solució tocant representació: assignacions d'apuntadors

Reversar una llista



```
void reversar() {  
    node_llista* n = primer_node;  
    while (n != NULL) {  
        /* Inv: per als nodes anteriors al que apunta n,  
           els apuntadors a anterior i seguent han estat  
           intercanviats respecte a l'original */  
        swap(n->seg, n->ant);  
        n = n->ant;  
    }  
    swap(primer_node, ultim_node);  
}
```

Exercici: versió recursiva

Estructures de dades noves

Cues ordenades

- ▶ Modificació de la classe Cua: propietat addicional de poder ser recorregudes en ordre creixent respecte al valor dels seus elements
- ▶ Dos tipus d'ordre: cronològic (com fins ara) + per valor (nou)
- ▶ Cal que hi hagi un operador < definit en el tipus o classe dels elements
- ▶ Cal redefinir la implementació amb més apuntadors

Apuntadors:

- ▶ `primer_node` i `ultim_node` i `seg_cron` per gestionar l'ordre d'arribada a la cua (**ordre cronològic**).
- ▶ `mes_petit` i `seg_ord` per gestionar l'ordre creixent segons el valor dels elements.

Nova definició de la classe

```
template <class T> class CuaOrd {
private:
    struct node_cuaOrd {
        T info;
        node_cuaOrd* seg_ord;
        node_cuaOrd* seg_cron;
    };
    int longitud;
    node_cuaOrd* primer_node;
    node_cuaOrd* ultim_node;
    node_cuaOrd* mes_petit;
    ... // especificació i implementació d'operacions privades
public:
    ... // especificació i implementació d'operacions públiques
};
```


Implementació cues ordenades

- ▶ Veurem només dues operacions públiques: `demanar_torn` (`push`) i `concatenar`
- ▶ Exercici: especificació i implementació d'altres operacions que caldria incloure

Demandar torn (push) |

```
void demanar_torn(const T& x) {  
    /* Pre: cert */  
    /* Post: el p.i. (una CuaOrd) ha quedat modificat afegint x com a darrer  
            element a l'ordre cronològic i on li toqui a l'ordre creixent */  
    ...  
}
```

Demandar torn (push) ||

```
void demandar_torn(const T& x) {  
    /* Pre: cert */  
    /* Post: el p.i. (una CuaOrd) ha quedat modificat afegint x com a darrer  
            element a l'ordre cronològic i on li toqui a l'ordre creixent */  
    node_cuaOrd* n;  
    n = new node_cuaOrd;  
    n->info = x;  
    n->seg_cron = NULL;  
    if (primer_node == NULL) {  
        primer_node = n;  
        ultim_node = n;  
        mes_petit = n;  
        n->seg_ord = NULL;  
    }  
    else {  
        ...  
    }  
    ++longitud;  
}
```

Demandar torn (push) III

```
ultim_node->seg_cron = n;
ultim_node = n;
// Ara s'actualitza l'ordre creixent
if (x < mes_petit->info) {
    n->seg_ord = mes_petit;
    mes_petit = n;
}
else {
    node_cuaOrd* ant = mes_petit;
    bool trobat = false;
    while (ant->seg_ord != NULL and not trobat) {
        if (x < (ant->seg_ord->info) trobat = true;
        else ant = ant->seg_ord;
    }
    n->seg_ord = ant->seg_ord;
    ant->seg_ord = n;
}
```


Concatenar I

```
void concatenar(CuaOrd &c2) {  
    /* Pre: cert */  
    /* Post: el p.i. ha estat modificat posant després del seu últim element (cronològic)  
            tots els elements de c2 en el mateix ordre cronològic en el qual hi eren  
            a c2, i amb tots els elements reorganitzats per satisfer l'ordre creixent;  
            c2 queda buida */  
    ...  
}
```

Concatenar II

```
void concatenar(CuaOrd &c2) {
    /* Pre: cert */
    /* Post: el p.i. ha estat modificat posant després del seu últim element (cronològic)
            tots els elements de c2 en el mateix ordre cronològic en el qual hi eren
            a c2, i amb tots els elements reorganitzats per satisfer l'ordre creixent;
            c2 queda buida */
    if (c2.primer_node != NULL) { // només cal fer alguna cosa si c2 no és buida
        if (primer_node == NULL) { // si el p.i. és buit passa a tenir els camps de c2
            primer_node = c2.primer_node;
            ultim_node = c2.ultim_node;
            mes_petit = c2.mes_petit;
        }
        else {
            ...
        }
    }
    // Actualitzem longitud del p.i. i els camps de c2
    longitud += c2.longitud;
    c2.primer_node = NULL;
    c2.ultim_node = NULL;
    c2.mes_petit = NULL;
    c2.longitud = 0;
}
}
```

Concatenar III

```

                                                    // sinó, connectem l'últim del p.i
ultim_node->seg_cron = c2.primer_node; // amb el primer de c2
ultim_node = c2.ultim_node;           // i actualitzem aquell
// Ara fem el merge dels nodes de les dues cues segon l'ordre creixent;
// els nodes tractats i connectats a la nova cua arriben fins ant
node_cuaOrd *ant, *act1, *act2;
act1 = mes_petit; act2 = c2.mes_petit;
if (act2->info < act1->info) {
    mes_petit = act2;
    ant = act2;
    act2 = act2->seg_ord;
}
else {
    ant = act1;
    act1 = act1->seg_ord;
}
while ...
```

Concatenar IV

(continuació)

```
while (act1 != NULL and act2 != NULL) {  
    if (act2->info < act1->info) {  
        ant->seg_ord = act2;  
        ant = act2;  
        act2 = act2->seg_ord;  
    }  
    else {  
        ant->seg_ord = act1;  
        ant = act1;  
        act1 = act1->seg_ord;  
    }  
}  
if (act1 != NULL) ant->seg_ord = act1;  
else ant->seg_ord = act2;
```

Multillistes: Motivació

Volem guardar una taula molt gran però molt *esparsa*: molts elements nuls

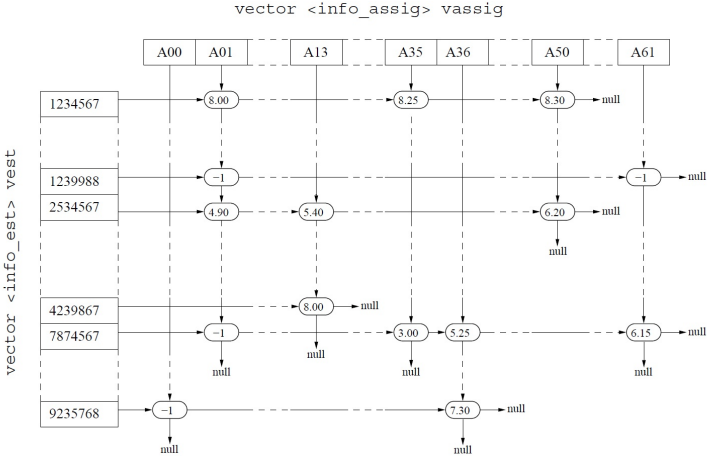
Necessitem:

- ▶ Donat un índex de fila, recuperar tots els elements no nuls de la fila
- ▶ Donat un índex de columna, recuperar tots els elements no nuls de la columna

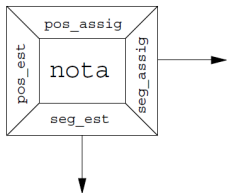
Exemple: Taula per guardar *cursos de la FIB*:

“l'estudiant X estava matriculat a Y i ha tret nota Z”

Multillistes: Esquema



Multil·listes: Node



Implementació i detalls: → apunts