Tipus recursius de dades (3a sessió)

R. Ferrer i Cancho

Universitat Politècnica de Catalunya

PRO2 (curs 2010-2011)
Versió 0.5 (agraïments: Ricard Gavaldà, Borja Vallés,...)

Avís: aquesta presentació no pretén ser un substitut dels apunts oficials de l'assignatura.

On som?

- Tema 7: Tipus recursius de dades
- 12a sessió

Avui

- Algorismes iteratius i recursius que accedeixen directament a la representacio basada en nodes.

Implimentacions directes d'operacions

- Implementar operacions directament sobre una estructura recursiva, sense fer servir les operacions primitives.
- Cerca d'un element en una pila:

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

- Avantatge: eficiència.
- Exemple: `sort` com a mètode de la classe `list` a STL.

Cerca en una pila: versió recursiva

Operació auxiliar recursiva.

```
static bool cerca_pila_node(node_pila* n, const T &x) {
/* Pre: cert */
/* Post: el resultat diu si x és l'info de *n o d'un dels seus següents */
  bool b;
  if (n == nullptr) b = false;
  else if (n->info == x) b = true;
  else b = cerca_pila_node(n->seguent, x);
  return b;
}
```

Compte: precondició de l'operador ->.

Cerca en una pila: versió iterativa

```
bool cerca_pila(const T &x) const
/* Pre: cert */
/* Post: el resultat indica si existeix un element x a la pila p.i. */
{
   node_pila* act;
   act = primer_node;
   b = false;
   /* Inv: b = x hi és a la part visitada del p.i. <=>
           b = x hi és a algun node anterior a act */
   while (act != nullptr and not b) {
     b = (act->info == x);
     act = act->seguent;
   }
   return b;
}
```

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

Nou mètode de la classe arbre binari

```
void inc_arbre(const T &k);
/* 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 */
```

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

```
void inc_arbre(const T &k) {
/* 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 */
  inc_node(a.primer_node, k);
}

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

Substitució de fulles per un arbre I

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

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

Interès: cal fer servir l'operació de còpia de nodes

Substitució de fulles per un arbre II

Operació auxiliar

```
static void subst_node(node_arbre* &n, int x, node_arbre* ns) {
/* Pre: cert */
/* Post: els nodes de la jerarquia de nodes que comença al node apuntat
   per n tals que el seu camp info valia x i no tenien següents han estat
   substituïts per una còpia de la jeraquia de nodes que comença al node
   apuntat per ns */
  if (n != nullptr) {
     if (n->info == x and n->segE == nullptr and n->segD == nullptr) {
        delete n; // equivalent aqui a esborra_node_arbre(n);
        n = copia_node_arbre(ns);
     }
     else {
        subst_node(n->segE, x, ns);
        subst_node(n->segD, x, ns);
     }
  }
}
```

Atenció: cal passar n per referència. Per què?

Inversió de l'ordre dels elements d'una llista I

- Solució iterativa.
- Nova operació de la classe `Llista` i per tant té un paràmetre implícit llista.

```
void invertir() {
/* 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 no s'ha mogut */
}
```

Inversió de l'ordre dels elements d'una llista II

Idea: intercanviar els punters als nodes anterior i següent de cada node

```
void invertir() {
/* 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 no s'ha mogut */
  if (longitud > 1)
     node_llista* n = primer_node;
     while (n != nullptr) {
       /* Inv: els nodes anteriors al que apunta n en la cadena que comença a primer_node
               han intercanviat els seus punters a l'anterior i al següent node */
       swap(n->seg, n->ant);
       n = n->ant;
     }
     swap(primer_node, ultim_node);
  }
}
```

Podem estalviar-nos l'if?

Inversió de l'ordre dels elements d'una llista III

Idea: compactar, p. e. estalviar intercanvis (llista d'un sol element i darrer node)

```
void invertir() {
/* 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 no s'ha mogut */
  node_llista* n = primer_node;
  while (n != ultim_node) {
    /* Inv: els nodes anteriors al que apunta n en la cadena que comença a primer_node
            han intercanviat els seus punters a l'anterior i al següent node */
    node_llista* aux = n->seg;
    n->seg = n->ant;
    n->ant = aux;
    n = aux;
  }
  if (n != nullptr and n != primer_node) {
    n->seg = n->ant;
    n->ant = nullptr;
    ultim_node = primer_node;
    primer_node = n;
  }
}
```

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:

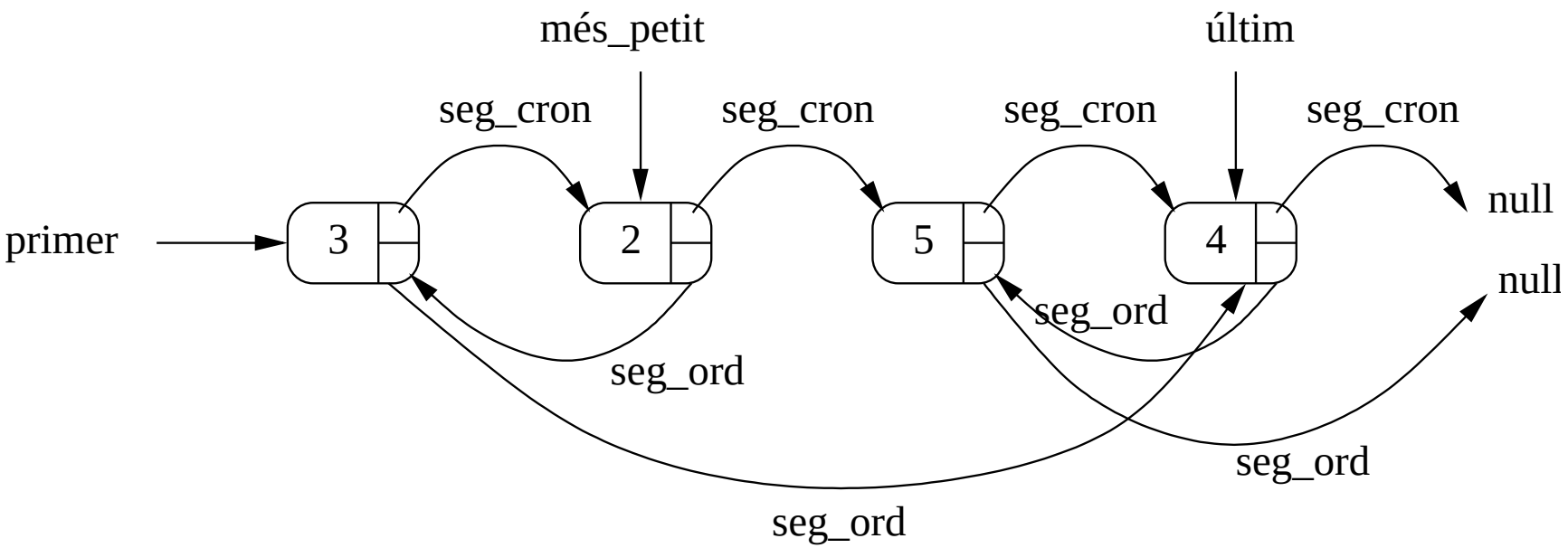
- `primer_node` i `ultim_node` i `seg_cron` per gestionar l'ordre d'arribada a la cua (**ordre cronològic**).
- `mes_petit` i `seg_ord` per gestionar l'ordre creixent segons el valor dels elements.

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
};
```

Esquema de la implementacio

Exemple:



Implementació cues ordenades

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

Demanar torn (push) I

```
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 */
...
}
```

Demanar torn (push) II

```
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 */
  node_cuaOrd*  n;
  n = new node_cuaOrd;
  n->info = x;
  n->seg_cron = nullptr;
  if (primer_node == nullptr) {
    primer_node = n;
    ultim_node = n;
    mes_petit = n;
    n->seg_ord = nullptr;
  }
  else {
    ...
  }
  ++longitud;
}
```

Demanar 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 != nullptr 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 != nullptr) { // només cal fer alguna cosa si c2 no és buida
    if (primer_node == nullptr) {  // 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 = nullptr;
    c2.ultim_node = nullptr;
    c2.mes_petit = nullptr;
    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 != nullptr and act2 != nullptr) {
  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 != nullptr) ant->seg_ord = act1;
else ant->seg_ord = act2;
```

Multillistes

Veure apunts oficials assignatura.