Tipus recursius de dades (4a sessió)

R. Ferrer i Cancho

Universitat Politècnica de Catalunya

PRO2 (curs 2010-2011)
Versió 0.3

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

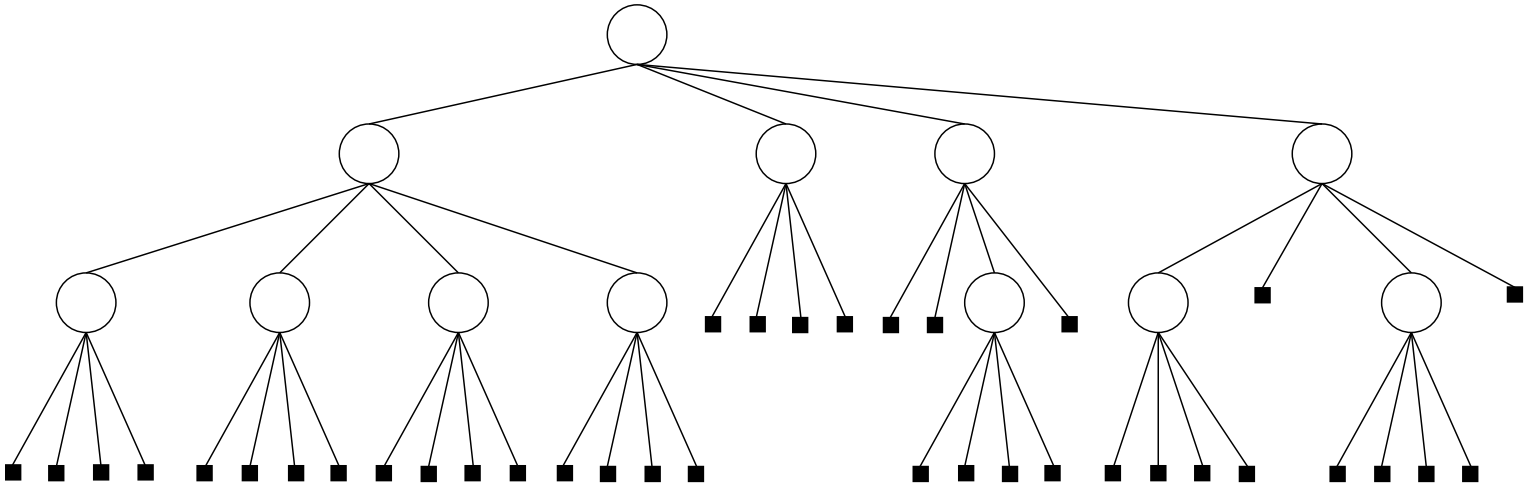
On som?

- Tema 7: Tipus recursius de dades
- 13a sessió

Avui

- Com s'implementen arbres N-aris, generals...
- Combinació d'iteració i recursivitat.

Arbres N-aris



- Generalització dels arbres binaris
- N: nombre de fills (binaris: N=2)
- Implementació:
  - `struct` del node conté N apuntadors a node, un per a cada fill.
  - Operació de consulta del fill i-èssim eficient: vector d'apuntadors a fills.

Definició classe ArbreNari

```
template <class T> class ArbreNari {
  private:
    struct node_arbreNari {
        T info;
        vector<node_arbreNari*> seg;
    };
    int N;
    node_arbreNari* primer_node;
    ...  // especificació i implementació d'operacions privades
  public:
    ...  // especificació i implementació d'operacions públiques
};
```

N: nombre de fills de cada subarbre.

Copiar jerarquies de nodes

```
static node_arbreNari* copia_node_arbreNari(node_arbreNari* m) {
/* Pre: cert */
/* Post: el resultat és nullptr si m és nullptr; en cas contrari, el resultat
         apunta al node arrel d'una jerarquia de nodes que és una còpia de la
         jerarquia de nodes que té el node apuntat per m com a arrel */
  node_arbreNari* n;
  if (m == nullptr) n = nullptr;
  else {
    n = new node_arbreNari;
    n->info = m->info;
    int N = m->seg.size();
    n->seg = vector<node_arbreNari*>(N);
    for (int i = 0; i < N; ++i)
      n->seg[i] = copia_node_arbreNari(m->seg[i]);
  }
  return n;
}
```

Esborrar jerarquies de nodes

```
static void esborra_node_arbreNari(node_arbreNari* m) {
/* Pre: cert */
/* Post no fa res si m és nullptr; en cas contrari, allibera espai de tots
        els nodes de la jerarquia que té el node apuntat per m com a arrel */
  if (m != nullptr) {
    int N = m->seg.size();
    for (int i = 0; i < N; ++i)
      esborra_node_arbreNari(m->seg[i]);
    delete m;
  }
}
```

Constructures/destructures I

```
ArbreNari(int n) {
/* Pre: n > 0 */
/* Post: el p.i. és un arbre buit d'aritat n */
  N = n;
  primer_node = nullptr;
}

ArbreNari(const T &x, int n) {
/* Pre: n > 0 */
/* Post: el p.i. és un arbre amb arrel x i n fills buits */
  N = n;
  primer_node = new node_arbreNari;
  primer_node->info = x;
  primer_node->seg = vector<node_arbreNari*>(N);
  for (int i=0; i<N; ++i)
    primer_node->seg[i] = nullptr;
}
```

Constructures/destructures II

```
ArbreNari(const ArbreNari& original) {
/* Pre: cert */
/* Post: el p.i. és una còpia d'original */
  N = original.N;
  primer_node = copia_node_arbreNari(original.primer_node);
}

~ArbreNari() {
  esborra_node_arbreNari(primer_node);
}
```

Modificadores I

```
ArbreNari& operator=(const ArbreNari& original) {
  if (this != &original) {
    esborra_node_arbreNari(primer_node);
    N = original.N;
    primer_node = copia_node_arbreNari(original.primer_node);
  }
  return *this;
}
```

`if (this != &original)` és equivalent a
`if (this->primer_node != original->primer_node)` ?

```
void a_buit() {
/* Pre: cert */
/* Post: el p.i. és un arbre buit */
  esborra_node_arbreNari(primer_node);
  primer_node = nullptr;
}
```

Modificadores II

```
void plantar(const T &x, vector<ArbreNari> &v) {
/* Pre: el p.i. és buit; v.size() és igual a l'aritat del p.i.;
        tots els components de v tenen la mateixa aritat que el p.i.
        i cap d'ells és el mateix objecte que el p.i. */

/* Post: el p.i. té x com a arrel i els N elements originals
         de v com a fills, v passa a contenir arbres buits */
  node_arbreNari* aux = new node_arbreNari;
  aux->info = x;
  aux->seg = vector<node_arbreNari*>(N);
  for (int i = 0; i < N; ++i) {
    aux->seg[i] = v[i].primer_node;
    v[i].primer_node = nullptr;
  }
  primer_node = aux;
}
```

Pot estalviar-se aux?

Modificadores III

```
void fills(vector<ArbreNari> &v) {
/* Pre: el p.i. no és buit i li diem A, v és un vector buit */
/* Post: el p.i. és buit, v passa a contenir els N fills de l'arbre A */
   node_arbreNari* aux= primer_node;
   v = vector<ArbreNari> (N, ArbreNari(N));
   for (int i=0; i<N; ++i)
      v[i].primer_node = aux->seg[i];
   primer_node= nullptr;
   delete aux;
}
```

Pot estalviar-se aux?

Estalvis

```
void fills(vector<ArbreNari> &v) {
/* Pre: el p.i. no és buit i li diem A, v és un vector buit */
/* Post: el p.i. és buit, v passa a contenir els N fills de l'arbre A */
  v = vector<ArbreNari>(N);
  for (int i = 0; i < N; ++i) v[i].primer_node = primer_node->seg[i];
  delete primer_node;
  primer_node = nullptr;
}
```

Més encara...?

```
void fills(vector<ArbreNari> &v) {
/* Pre: el p.i. no és buit i li diem A, v és un vector buit */
/* Post: el p.i. és buit, v passa a contenir els N fills de l'arbre A */
  v = primer_node->seg;
  delete primer_node;
  primer_node = nullptr;
}
```

Consultores

```
T arrel() const {
/* Pre: el p.i. no és buit */
/* Post: el resultat és l'arrel del p.i. */
    return primer_node->info;
}

bool es_buit() const {
/* Pre: cert */
/* Post: el resultat indica si el p.i. és un arbre buit */
    return primer_node == nullptr;
}

int aritat() const {
/* Pre: cert */
/* Post: el resultat és l'aritat del p.i. */
    return N;
}
```

Eficiència recorreguts arbres N-aris

- Exemple:

```
int suma_elements(ArbreNari<int> &a) {
/* Pre: a=A */
   ...
/* Post: el resultat és la suma dels elements d'A */
}
```

Exemple d'ús d'arbres N-aris: suma de tots elements

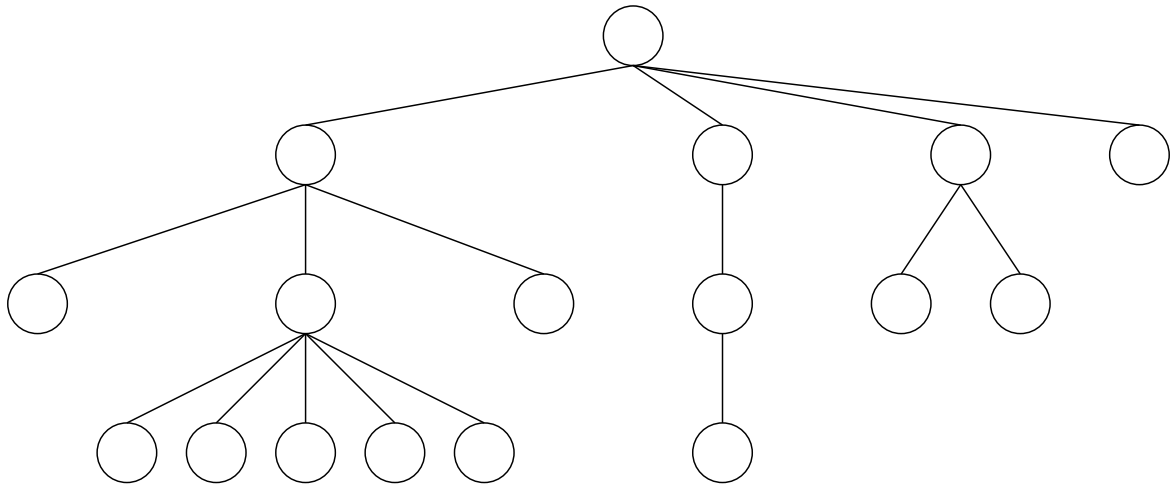
```
int suma_elements(ArbreNari<int> &a) {
/* Pre: a=A */
/* Post: el resultat és la suma dels elements d'A */
  int s;
  if (a.es_buit()) s = 0;
  else {
    s = a.arrel();
    int N = a.aritat();
    vector<ArbreNari<int> > v;
    a.fills(v);
    for (int i = 0; i < N; ++i)
      s += suma_elements(v[i]);
  }
  return s;
}
```

Podem estalviar-nos `int N = a.aritat();`? Pista: Post de `fills`.

Exemple d'ús d'arbres N-aris: sumar un valor k a cada element

```
void inc_arbreNari(ArbreNari<int> &a, int k)
/* Pre: a=A */
/* Post: a és com A però havent sumat k a tots els seus elements */
{
  if (not a.es_buit()) {
    int s = a.arrel() + k;
    int N = a.aritat();
    vector<ArbreNari<int> > v;
    a.fills(v);
    for (int i = 0; i < N; ++i)
      inc_arbreNari(v[i], k);
    a.plantar(s, v);
  }
}
```

Arbres generals I



- Nombre indeterminat de fills.
- Propietat important:
  - **Un arbre general pot ser un arbre buit o un arbre no buit que no conté arbres buits com a subarbres.**
  - Cada subarbre pot tenir un nombre indeterminat, fins i tot zero, de fills no buits, però cap fill buit.

Arbres generals II

- Diferents formes d'implementar-los:
  - Estratègia de representació anomenada "primer fill, germà dret".
  - A cada node, seqüència de punters a node per accedir als fills.
- Perill: consulta ineficient del fill i-èssim (si per arribar al fill i-essim cal passar primer pels anteriors)
- Solució: accés eficient amb vector de punters a cada node.
- Repte: nombre variable i indeterminat de fills a cada subarbre.
  - Nombre màxim de fills. Problema: malbaratament de memòria i no sempre possible
  - Redimensionament dinàmic (`push_back` amb vectors de STL). Problemes d'ineficiència temporal i espacial.

Definició de la classe ArbreGen

```
template <class T> class ArbreGen
  private:
    struct node_arbreGen {
        T info;
        vector<node_arbreGen*> seg;
    };
    node_arbreGen* primer_node;
    ...  // especificació i implementació d'operacions privades
  public:
    ...  // especificació i implementació d'operacions públiques
};
```

Important: Ja no tenim un atribut amb el nombre de fills (per a tota la classe) ni tans sols per a cada node (es pot obtenir amb `seg.size()`

Copiar i esborrar jerarquies de nodes

Idèntiques a les dels arbres N-aris (només canviar tipus dels nodes)

Constructores/destructores I

```
ArbreGen() {
/* Pre: cert */
/* Post: el p.i. és un arbre general buit */
  primer_node = nullptr;
}

ArbreGen(const T &x) {
/* Pre: cert */
/* Post: el p.i. és un arbre general amb arrel x i sense fills */
  primer_node = new node_arbreGen;
  primer_node->info = x;
  // No cal fer primer_node->seg = vector<node_arbreGen*>(0);
}
```

Constructores/destructores II

```
ArbreGen(const ArbreGen& original) {
/* Pre: cert */
/* Post: el p.i. és una còpia d'original */
  primer_node = copia_node_arbreGen(original.primer_node);
}

~ArbreGen() {
  esborra_node_arbreGen(primer_node);
}
```

Modificadores I

```
ArbreGen& operator=(const ArbreGen& original) {
  if (this != &original) {
    esborra_node_arbreGen(primer_node);
    primer_node = copia_node_arbreGen(original.primer_node);
  }
  return *this;
}

void a_buit() {
/* Pre: cert */
/* Post: el p.i. és un arbre general buit */
  esborra_node_arbreGen(primer_node);
  primer_node = nullptr;
}
```

Modificadores II

```
void plantar(const T &x) {
/* Pre: el p.i. és buit */
/* Post: el p.i. té x com a arrel i zero fills */
  primer_node = new node_arbreGen;
  primer_node->info = x;
  // No cal fer primer_node->seg = vector<node_arbreGen*>(0);
}

void plantar(const T &x, vector<ArbreGen> &v) {
/* Pre:  el p.i. és buit, v.size()>0 i cap arbre de v és buit */
/* Post: el p.i. té x com a arrel i els elements originals
         de v com a fills, v passa a contenir arbres buits */
  node_arbreGen* aux= new node_arbreGen;
  aux->info = x;
  int n = v.size();
  aux->seg = vector<node_arbreGen*>(n);
  for (int i = 0; i < n; ++i) {
    aux->seg[i] = v[i].primer_node;
    v[i].primer_node = nullptr;
  }
  primer_node = aux;
}
```

Pot estalviar-se aux?

Modificadores III

```
void afegir_fill(const ArbreGen &a) {
/* Pre: el p.i. i a no són buits */
/* Post: el p.i. té un fill més que a l'inici, i aquest nou darrer fill
         és una còpia de l'arbre a */
   (primer_node->seg).push_back(copia_node_arbreGen(a.primer_node));
}
```

Modificadores IV

```
void fill(const ArbreGen &a, int i) {
/* Pre: el p.i. és buit, a no és buit, i està entre 1 i el nombre de
        fills d'a */
/* Post: el p.i és una còpia del fill i-èssim d'a */
   primer_node = copia_node_arbreGen((a.primer_node)->seg[i-1]);
}

void fills(vector<ArbreGen> &v) {
/* Pre: el p.i. no és buit i li diem A, v és un vector buit */
/* Post: el p.i. és buit, v passa a contenir els fills de l'arbre A */
   node_arbreGen* aux= primer_node;
   int n = aux->seg.size();
   v = vector<ArbreGen> (n);
   for (int i=0; i<n; ++i)
      v[i].primer_node = aux->seg[i];
   primer_node= nullptr;
   delete aux;
}
```

Eficiencia de `fills(...)` enfront `fill(...)` succesivament.

Estalvis

Estalvi de aux?

```
void fills(vector<ArbreGen> &v) {
/* Pre: el p.i. no és buit i li diem A, v és un vector buit */
/* Post: el p.i. és buit, v passa a contenir els fills de l'arbre A */
  int n = primer_node->seg.size();
  v = vector<ArbreGen>(n);
  for (int i = 0; i < n; ++i) v[i].primer_node = primer_node->seg[i];
  delete primer_node;
  primer_node = nullptr;
}
```

Encara més...?

```
void fills(vector<ArbreGen> &v) {
/* Pre: el p.i. no és buit i li diem A, v és un vector buit */
/* Post: el p.i. és buit, v passa a contenir els fills de l'arbre A */
  v = primer_node->seg;
  delete primer_node;
  primer_node = nullptr;
}
```

Consultores

```
T arrel() const {
/* Pre: el p.i. no és buit */
/* Post: el resultat és l'arrel del p.i. */
    return primer_node->info;
}

bool es_buit() const {
/* Pre: cert */
/* Post: el resultat indica si el p.i. és un arbre buit */
    return primer_node == nullptr;
}

int nombre_fills() const {
/* Pre: el p.i. no és buit */
/* Post: el resultat és el nombre de fills del p.i. */
   return (primer_node->seg).size();
}
```

Exemple d'ús d'arbres generals: suma de tots els elements

```
int suma_elements(ArbreGen<int> &a) {
/* Pre: a=A */
/* Post: el resultat és la suma dels elements d'A */
  int s;
  if (a.es_buit()) s = 0;
  else {
    s = a.arrel();
    vector<ArbreGen<int> > v;
    a.fills(v);
    int n = v.size();
    for (int i = 0; i < n; ++i)
      s += suma_elements(v[i]);
    }
  }
  return s;
}
```

Exemple d'ús d'arbres generals: sumar un valor k a cada element

```
void inc_arbreGen(ArbreGen<int> &a, int k)
/* Pre: a=A */
/* Post: a és com A però havent sumat k a tots els seus elements */
{
  if (not a.es_buit()) {
    int s = a.arrel() + k;
    vector<ArbreGen<int> > v;
    a.fills(v);
    int n = v.size();
    if (n == 0) a.plantar(s); // ús del plantar sense paràmetres!
    else {
      for (int i = 0; i < n; ++i) inc_arbreGen(v[i], k);
      a.plantar(s, v);
    }
  }
}
```

Possible optimització?

Canviar `a.plantar(s)` per `a.plantar(s,vector<int>(0))` o `a.plantar(s,v)`?

```
void inc_arbreGen(ArbreGen<int> &a, int k)
/* Pre: a=A */
/* Post: a és com A però havent sumat k a tots els seus elements */
{
  if (not a.es_buit()) {
    int s = a.arrel() + k;
    vector<ArbreGen<int> > v;
    a.fills(v);
    int n = v.size();
    for (int i = 0; i < n; ++i)
      inc_arbreGen(v[i], k);
    a.plantar(s, v);
  }
}
```

Compte amb la Pre de `a.plantar(s, v);`!