

JUTGE PRO2 FIB

L13. Classes Arbre (Binari, N-ari i General)

GitHub: <https://github.com/MUX-enjoyer/PRO2-FIB-2025>

Índex de Fitxers

- X09609 Cerca de subarbres binaris.cc** (pàgina 2)
- X15014 Arbre de sumes d'un arbre donat.cc** (pàgina 3)
- X18899 Arbre de sumes d'un arbre n-ari.cc** (pàgina 4)
- X60365 Cerca en un arbre general.cc** (pàgina 5)
- X67695 Suma màxima dels camins d'un arbre binari.cc** (pàgina 6)
- X75329 Nombre d'aparicions d'un valor en un arbre binari.cc** (pàgina 7)

Exercici: X09609 Cerca de subarbres binaris.cc

```
1 /* Pre: p.i. = A, asub es buit */
2 /* Post: si A conte x, asub es el subarbre d'A resultat de la cerca; si A no
   conte x, asub es buit */
3 void sub_arrel(Arbre& asub, const T& x) {
4     int dist_min = -1;
5     trobar_sub_arrel(primer_node, asub, x, 0, dist_min);
6 }
7
8 void trobar_sub_arrel(node_arbre* node, Arbre& asub, const T& x, int nivell,
   int& dist_min) {
9     if (node != NULL) {
10         ++nivell;
11         if (node->info == x && (dist_min == -1 || nivell < dist_min)) {
12             asub.primer_node = copia_node_arbre(node);
13             dist_min = nivell;
14         }
15         else {
16             trobar_sub_arrel(node->segE, asub, x, nivell, dist_min);
17             trobar_sub_arrel(node->segD, asub, x, nivell, dist_min);
18         }
19     }
20 }
```

Exercici: X15014 Arbre de sumes d'un arbre donat.cc

```
1 /* Pre: cert */
2 /* Post: l'arbre asum és l'arbre suma del p.i. */
3 void arb_sumes(Arbre<int> &asum) const {
4     asum.primer_node = aux_arb_sumes(primer_node);
5 }
6
7 node_arbre* aux_arb_sumes(node_arbre* node) const {
8     if (node == NULL) return NULL;
9
10    node_arbre* n = new node_arbre;
11    n->info = node->info;
12
13    if (node->segE != NULL) {
14        n->segE = aux_arb_sumes(node->segE);
15        n->info += n->segE->info;
16    }
17    if (node->segD != NULL) {
18        n->segD = aux_arb_sumes(node->segD);
19        n->info += n->segD->info;
20    }
21
22    return n;
23 }
```

Exercici: X18899 Arbre de sumes d'un arbre n-ari.cc

```
1 /* Pre: cert */
2 /* Post: asum és un arbre amb la mateixa estructura que el p.i. i cada node és
   la suma del node corresponent al p.i i tots els seus descendents al p.i. */
3 void arbsuma(ArbreNari& asum) const {
4     asum.primer_node = aux_arb_sumes(primer_node);
5 }
6
7 node_arbreNari* aux_arb_sumes(node_arbreNari* node) const {
8     if (node == NULL) return NULL;
9
10    node_arbreNari* n = new node_arbreNari;
11    n->info = node->info;
12
13    n->seg = vector<node_arbreNari*>(node->seg.size());
14
15    for (int i = 0; i < node->seg.size(); ++i) {
16        if (node->seg[i] != NULL) {
17            n->seg[i] = aux_arb_sumes(node->seg[i]);
18            n->info += n->seg[i]->info;
19        }
20    }
21    return n;
22 }
```

Exercici: X60365 Cerca en un arbre general.cc

```
1  /* Pre: cert */
2  /* Post: el resultat indica si x es troba al p.i. o no */
3  bool buscar(const T& x) const {
4  return buscarRecursiu(primer_node, x);
5  }
6
7  bool buscarRecursiu(node_arbreGen* node, const T& valor) const {
8  if (node == NULL) return false;
9  if (node->info == valor) return true;
10
11  for (int i = 0; i < node->seg.size(); ++i) {
12  if (buscarRecursiu(node->seg[i], valor)) return true;
13  }
14  return false;
15 }
```

Exercici: X67695 Suma màxima dels camins d'un arbre binari.cc

```
1 /* Pre: el parametre implícit no es buit */
2 /* Post: el resultat es la suma del camí de suma màxima del parametre implícit
*/
3 T max_suma_cami() const {
4     return suma_max(primer_node);
5 }
6
7 int suma_max(const node_arbre* node) const {
8     if (node == NULL) return 0;
9
10    int suma = node->info;
11    int suma_esq = suma_max(node->segE);
12    int suma_dreta = suma_max(node->segD);
13
14    if (suma_esq > suma_dreta) return suma + suma_esq;
15    else return suma + suma_dreta;
16 }
```

Exercici: X75329 Nombre d'aparicions d'un valor en un arbre binari.cc

```
1 /* Pre: cert */
2 /* Post: el resultat indica el nombre d'aparicions de x en el p.i. */
3 int freq(const T& x) const {
4     return freq_aux(primer_node, x);
5 }
6
7
8 int freq_aux(const node_arbre* node, const T& x) const {
9     if (node == NULL) return 0;
10
11     int comptador = 0;
12     if (node->info == x) ++comptador;
13
14     comptador += freq_aux(node->segE, x);
15     comptador += freq_aux(node->segD, x);
16
17     return comptador;
18 }
```