ESTRUCTURES DE DADES I ALGORÍTMICA

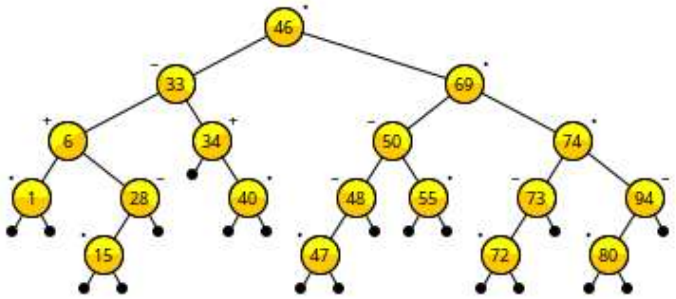
Data: 18/11/2024. Temps total: 2h.

Qüestions ED (50 % ED)**. CAL FER ELS EXERCICIS EN FULLS SEPARATS**

Només pot usar-se el material de teoria que hi ha a la web (sense anotacions).

Cal deixar algun document identificador (**DNI preferentment**) a la taula.

1ed.- (10/60) Es disposa d'un arbre binari d'enters i es vol enriquir amb un nou mètode que determini quants nodes hi ha des de l'arrel fins al node amb contingut **`valor`**, és a dir, quants nodes hi ha amb contingut > arrel i ≤ `valor`. Se sap que l'arbre binari té disposats els seus valors com si fos de cerca (no necessàriament balancejat).



Es demana: implementar un mètode de la classe `ArbreBinariInt` que rebi un valor per paràmetre i digui quants nodes hi ha amb contingut superior a l'arrel (no inclosa) i menor o igual a aquest valor (node inclòs, existeixi o no). Cal **no** recórrer nodes innecessaris. Si el contingut de l'arrel és més gran o igual que el valor, es retornarà 0, i si no hi ha cap node amb el valor, s'indicarà el nombre de nodes que hi ha des de l'arrel fins a la posició on hauria d'anar el node en cas que s'inserís amb aquest valor (incloent aquest últim com si existís). Per exemple, en l'arbre de la figura, si el valor és 73 caldria retornar 7, mentre que si el valor és 28 caldria retornar 0. Si el valor fos 60 (node inexistent), caldria retornar 5.

```
class ArbreBinariInt {
  public: // no cal usar cap altre mètode públic
    ...
    int nNodesDesDArrel(int valor) const; // aquest és el que cal fer
    // pre: cert (error: arbre buit);
    // post: retorna el nombre de nodes des de l'arrel fins a valor (arrel < ... <= valor)

  private:
    struct node {
        int dada;
        shared_ptr<node> fe, fd;
    };
    ... // poden usar-se mètodes privats

    shared_ptr<node> arrel; // representació de l'arbre
};

int ArbreBinariInt::nNodesDesDArrel(int valor) const { // proposta d'immersió;
                                                       // pot canviar-se
    if (arrel == NULL) throw "L'arbre no pot ser buit";
    else if (...) ...
    else return iNNodesDesDArrel(arrel->fd, valor); // iNNodesDesDArrel: mètode privat
}
```

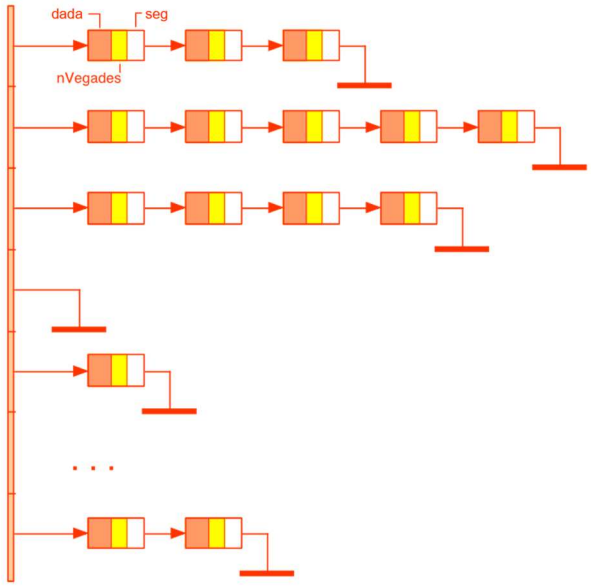
2ed.- (10/60) En un arbre de cerca inicialment buit, inserir balancejadament els valors 31, 16, 25, 13, 8, 50, 28, 30, 64, 44, 68, 47, 62, 63, 61, 69, 60. De l'arbre resultant, fer tots els recorreguts.

(continua →)

3ed.- (10/60) Un multi-conjunt és un conjunt en el qual es permet que els valors estiguin repetits diverses vegades. Es vol implementar un multi-conjunt utilitzant una representació basada en l'encadenament indirecte de manera que, a cada node, es guardi el valor i el nombre de repeticions. Els elements d'un multi-conjunt disposen de la funció de dispersió de què disposen també les claus de les EDFs.

```
template <element>
class MultiConjunt {
  public:
    MultiConjunt(int mida = MIDA_DEFECTE);
    ...
    void Esborrar(const element& e);
    // pre: cert;
    // post: elimina un element e del multi-conjunt
    ...
  private:
    struct node {
        element dada;
        int nVegades;
        node * seg;
    };

    // inv: nDades ≥ 0 ^ 0 ≤ maxDades ^ dades[0..maxDades-1] són punters
    //      a una estructura dinàmica d'elements que són sinònims
    node ** dades;
    int nDades, maxDades; // nDades: número de nodes
};
```



Es demana: implementar el mètode **`Esborrar`**, que esborra una repetició d'un element (no és un error que no existeixi). Si s'esborra l'última repetició, llavors cal eliminar el node. Cal **no usar** cap mètode privat (p. ex. cercar).

ESTRUCTURES DE DADES I ALGORÍTMICA

Data: 18/11/2024. Temps total: 2h.

Problema ED (50 % ED)**. CAL FER ELS EXERCICIS EN FULLS SEPARATS**

Només pot usar-se el material de teoria que hi ha a la web (sense anotacions).

Cal deixar algun document identificador (**DNI preferentment**) a la taula.

4ed.- (30/60) Després d'haver-se assabentat que els coneixements previs dels estudiants en algunes assignatures no és l'adequat, la Maria A. Politec, directora del centre Estem Preparats Sobradament, vol saber quins estudiants estan matriculats en assignatures d'un nivell més avançat del que els pertoca.

En el seu centre es fan diferents carreres, i cadascuna té una durada d'entre 2 i 5 cursos. A cada curs s'imparteixen un conjunt d'assignatures. Cada assignatura consta d'un cert nombre de temes (entre 5 i 15 per assignatura). En el centre hi ha matriculats una gran quantitat d'alumnes, cadascun en una de les carreres. De cada alumne es coneix la seva matrícula (assignatures que està cursant actualment) i el seu expedient (assignatures ja finalitzades que ha cursat prèviament).

```
class ex2024 {
    map <codiCarr, vector<list<codiAssig>>> carreres;     // plans d'estudis per cursos
    map <codiAssig, dadesAssig> assignatures;             // dades de les assignatures
    map <codiAssig, list<codiAssig>> prerequisits;        // prerequisits de les assignatures
    map <codiAssig, vector<codiTema>> temaris;            // temes de les assignatures
    map <dni, dadesAlumne> alumnes;                       // dades dels alumnes
    map <dni, map<codiAssig, nota>> matricula;            // matrícula del semestre actual
    map <dni, map<codiAssig, map<any, nota>>> expedient; // assignatures ja cursades
public:
    void prerequisits_incomplerts() const;
    // pre: cert; post: per cada alumne que no compleix els prerequisits, llista les assignatures afectades
    //     La consideració de superar o no els prerequisits s'ha de fer seguint les indicacions de l'apartat a.
    // Exemple de sortida, assumint que EDA i MTP 2 tenen com a prerequisit haver aprovat MTP 1, i que en Josep
    //     (matriculat a EDA i MTP 2) i la Maria (matriculada només a EDA) no han aprovat MTP 1:
    // Josep Moragues:
    //     Estructura de dades i algorítmica
    //     Metodologia i tecnologia de la programació II
    // Maria Mercè Marçal:
    //     Estructura de Dades i Algorítmica
};
```

En aquesta estructura, **`carreres`** conté la llista d'assignatures de cada curs (1, 2, 3…) del pla d'estudis, **`assignatures`** conté les dades de cada assignatura, **`prerequisits`** conté la llista d'assignatures que són prerequisit d'alguna assignatura (si una assignatura no té prerequisits, no hi figurarà), **`temaris`** conté els temes de cada assignatura, **`alumnes`** conté les dades dels alumnes, **`matricula`** conté la matrícula de cada alumne en el semestre actual (i espai per posar la nota quan toqui) i **`expedient`** conté tot l'històric d'assignatures ja cursades dels alumnes per curs acadèmic (any), juntament amb la nota que han obtingut. Al final de cada semestre, quan ja s'han entrat totes les notes, les dades de **`matricula`** es posen a **`expedient`**, però durant el semestre les notes de l'expedient no contenen les dades de matrícula actual.

Cada element s'identifica amb una clau (**`codiCarr`**, **`codiAssig`**, **`codiTema`**...) que **no** és un valor numèric, però té les operacions de les claus. Els valors **`any`** i **`nota`** corresponen a enters amb valors escaients (2020, 2021, 2022… i 0..100). La classe **`dadesAssig`** guarda la informació referent a una assignatura (atributs: nom, crèdits, n. grups...), i la classe **`dadesAlumne`** guarda la informació personal de l'alumne (atributs: nom, cognoms, adreça, data de naixement, codi de carrera...).

Es demana:

a. Implementar en C++ el mètode **`prerequisits_incomplerts`** que mostri tots els alumnes que estan matriculats actualment (en aquest semestre) en assignatures que tenen prerequisits sense haver-los aprovat prèviament. Cal mostrar el nom de cada alumne afectat (els que tinguin els prerequisits aprovats no s'han de mostrar) i, per cadascun d'ells, la llista d'assignatures matriculades amb almenys un prerequisit no aprovat. Es considera que una assignatura no està aprovada si no s'hi ha matriculat mai o si

l'última matrícula està suspesa (nota inferior a 50). Cal considerar que el nombre d'assignatures matriculades que cursa un alumne en un semestre és molt inferior al nombre total d'assignatures que tenen prerequisits.

Detalls: Les classes clau **no** són enters i disposen de les operacions de les claus. Les classes **`dadesXxx`** s'han implementat a part i disposen de constructor per defecte, constructor amb paràmetres i mètodes per consultar el valor dels seus atributs (de l'estil **`x.obtAtribut()`**) que retornen enters o cadenes segons l'atribut que es consulti. Aquestes classes no tenen mètodes d'entrada/sortida (**`x.mostrar()`** o **`<<`**) ni per retornar diversos atributs.

A tenir en compte:

- No es considera una bona pràctica fer còpia de les estructures STL (`vector`, `list`, `map`...); és preferible obtenir els iteradors per fer les consultes.
- No s'han d'usar estructures auxiliars si no és necessari.
- Millor no fer servir declaracions **`auto`**; com a molt, dos cops en tot el problema.

Detall final: no tota la informació de l'estructura de dades és necessària per resoldre l'exercici.