Universitat Politècnica de Catalunya
Facultat d'Informàtica de Barcelona

Cognoms, Nom	D.N.I.

Titulació: Grau en Enginyeria Informàtica
Assignatura: Programació 2 (PRO2)
Duració: 2h 30m

Curs: Q2 2018–2019 (2n Parcial)
Data: 7 de juny de 2019

1. **(6 punts)** En aquest problema, treballarem amb una nova classe anomenada Bag. Un objecte d'aquesta classe representa a un conjunt d'elements d'un cert tipus T però a més es manté el conjunt internament ordenat per ordre de consulta (o adició) al conjunt. Això es fa amb la idea de que els elements més freqüentment consultats quedin al començament i per tant sigui més ràpid accedir-hi.

   Considerem la següent definició:

```
template <typename T>
class Bag {
public:
// Constructora, ctora per còpia, destructora, assignació
    ...

// Pre: cert
// Post: s'ha afegit l'element x a la Bag implícita, si x no hi era,
//       altrament no s'ha afegit res; però tant si x ja era present com si
//       s'ha afegit, x passa a ser el primer element del Bag
void afegeix(const T& x);

// Pre: cert
// Post: retorna cert si i només si
//       la Bag implícita conté x; a més, si x estava
//       present x passa a ser el primer element
bool conte(const T& x);

// altres operacions
int cardinalitat() const; // retorna el nombre d'elements de la Bag
T primer() const;  // retorna el primer element de la Bag
  ...
private:
    struct node {
      T info;
      node* seg;
    };
    node* sent; // apuntador al sentinella
    int tam;     // nombre d'elements
}
```

Com veieu, implementem la classe mitjançant una llista **simplement** encadenada amb sentinella (però **no** la tancarem circularment). Tot objecte de la classe, inclús si és una Bag buida, conté un node sentinella (apuntat per `sent`). Quan una Bag té 1 o més elements `sent -> seg` apunta al primer element de la Bag. L'últim element de una Bag no té successor; si `p` és un apuntador a l'últim element llavors `p -> seg == nullptr` (atenció: `p -> seg != sent`, no està tancada circularment). En una Bag buida `sent -> seg == nullptr`.

(a) (2 punts) Implementa un mètode `static` privat que donat un apuntador `p` al primer node d'una cadena de nodes i un valor `x` del tipus `T` retorna un apuntador al predecessor del node, entre el successor de `p` i el final de la llista, que conté `x`, o bé `nullptr` si no existeix cap node així.

La informació del node apuntat per `p` és irrellevant i la vostra solució **no** hauria d'accedir en cap a cas a `p -> info`.

```
// Pre: p != nullptr
// Post: retorna nullptr si cap node entre el succesor de p
//       i el final conté x, altrament retorna un apuntador al
//       predecessor d'un node, entre el successor de p i el final,
//       que conté x
static node* cerca(node* p, const T& x);
```

(b) (2 punts) Implementa un mètode privat que donat un apuntador `p` no nul i tal que `p -> seg != nullptr`, mou el node apuntat per `p -> seg` al començament de la `Bag` implícita. La teva solució no pot crear ni destruir nodes i **no** modifica mai la `info` dels nodes (e.g., no es poden fer `swap`s amb els atributs `info`).

```
// Pre: p != nullptr, p -> seg != nullptr
// Post: mou al node successor de p al començament
// de la Bag implícita
void mou_al_front(node* p);
```

(c) (1 punt) Implementa el mètode "consultor" `conte`. Ha de retornar cert o fals, segons que `x` estigui o no a la `Bag` però a més, si `x` està present, llavors el node que conté `x` ha de colocar-se com a primer element. Justament per aquesta raó el mètode no és `const`, donat que pot modificar l'ordre dels nodes de la llista. Pots fer servir els mètodes privats dels apartats anteriors.

(d) (1 punt) Implementa el mètode `afegeix`. Pots fer servir els mètodes privats dels apartats anteriors.

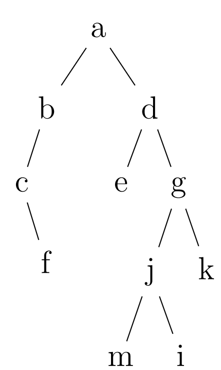
2. **(4 punts)** Considerem la següent definició d'una classe ArbreBinari en C++

```
template <typename T>
class ArbreBinari{
private:
   struct node {
      T info;
      node* esq;
      node* dre;
   };
   node* arrel; // apuntador a l'arrel de l'arbre
   ...
public:
   ...
// Pre: l'arbre implícit no és buit
// Post: retorna la longitud mitja dels camins de l'arbre
double longitud_mitja_camins() const;
   ...

};
```

L'objectiu "final" d'aquest problema (apartat c) és implementar el mètode consultor `longitud_mitja_camins()` que, com el seu nom indica, retorna la longitud mitja dels camins en l'arbre. Per tal d'obtenir la longitud mitja s'ha de calcular el que s'anomena *IPL* de l'arbre (de l'anglés *internal path length*), que és la suma de la longitud de tots els camins entre l'arrel i tots els nodes de l'arbre. Un cop calculat l'IPL, la longitud mitja s'obté dividint l'IPL per la talla de l'arbre (que per la precondició, és no nula).

Per exemple, si l'arbre és



llavors tenim que el seu IPL és

$$\text{IPL} = 0 + 1 + 1 + 2 + 2 + 2 + 3 + 3 + 3 + 4 + 4 = 25$$

i per tant que la longitud mitja de camins és $25/11 = 2.27\ldots$

És a dir, definim formalment

$$\text{IPL}(t) = \sum_{x \in t} \text{longitud}(x, t), \qquad (*)$$

ón $\text{longitud}(x, t)$ és la longitud del camí des de l'arrel de l'arbre t fins al node x; si t és buit ($t = \square$) llavors definim $\text{IPL}(\square) = 0$, doncs el sumatori ($*$) té zero termes.

La longitud mitja de camins buscada serà doncs $\text{IPL}(t)/\text{talla}(t)$ si t no es un arbre buit. La funció $\text{talla}(t)$ denota la talla o mida de l'arbre t, és a dir, el nombre de nodes de t.

(a) (1 punt) Demostra que la següent definició **recursiva** de la funció IPL

$$\text{IPL}(t) = \begin{cases} 0 & \text{si } t = \square, \\ \text{IPL}(\ell) + \text{IPL}(r) + \text{talla}(\ell) + \text{talla}(r) & \text{si } t = \texttt{ArbreBinari}(x, \ell, r), \end{cases}$$

és equivalent a la definició d'IPL donada a l'equació ($*$).

La funció talla, que ens dona la talla d'un arbre binari t, també es pot definit recursivament com segueix:

$$\text{talla}(t) = \begin{cases} 0 & \text{si } t = \square, \\ 1 + \text{talla}(\ell) + \text{talla}(r) & \text{si } t = \texttt{ArbreBinari}(x, \ell, r). \end{cases}$$

D'altra banda, la funció $\text{longitud}(x, t)$ admet la següent definició recursiva:

$$\text{longitud}(x, t) = \begin{cases} 0 & \text{si } t = \square, \\ 0 & \text{si } t = \texttt{ArbreBinari}(x, \ell, r), \\ 1 + \text{longitud}(x, \ell) & \text{si } t = \texttt{ArbreBinari}(y, \ell, r) \text{ i } x \in \ell, \\ 1 + \text{longitud}(x, r) & \text{si } t = \texttt{ArbreBinari}(y, \ell, r) \text{ i } x \in r. \end{cases}$$

Podeu fer servir aquestes dues definicions recursives (la de talla i la de longitud) per a demostrar la equivalència entre les definicions recursiva i no recursiva ($*$) del'IPL.

(b) (2 punts) Especifica i implementa en C++ un o més mètodes privats de la classe ArbreBinari que ens permetin calcular eficientment l'IPL de un arbre. Una solució eficient hauria de visitar cada node de l'arbre un nombre constant de cops com a molt. En particular, la teva solució **no** ha de calcular $\text{longitud}(x, t)$ explícitament i separada per a cada node x de l'arbre t. Explica si necessites o és convenient fer alguna immersió d'especificació i/o d'eficiència.

(c) (1 punt) Implementa el mètode `longitud_mitja_camins()` usant la o les funcions privades de l'apartat anterior.