Examen final, Programació 2

Enginyeria Informàtica
Gener 2015
Temps estimat: 2h 30m

- Feu bona lletra.
- Lliureu els dos problemes (en full separats, encara que siguin en blanc).

Problema 1 (5 punts)

Considerem la implementació amb nodes encadenats del tipus `stack<T>`, amb un apuntador al node cim. La recordem a continuació:

```
template <class T> class stack {
   private:
       struct node {
         T info;
         node* seg;
       };

       int altura;
       node* primer;

       ...  // especificació d'operacions privades

   public:
       ...  // especificació d'operacions públiques
};
```

Suposem que `T` disposa d'una operació d'ordre $<$ (i la seva variant $\leq$). Es demana implementar una operació pròpia de la classe que ordeni el seu paràmetre implícit, de manera que l'element més petit quedi al cim, i tot altre element sigui $\geq$ que el que té damunt.

La implementació no pot usar cap operació pública de la classe, ni pot crear noves piles, ni altres estructures de dades (llistes, cues, vectors, ...).

Us proposem fer servir una variant de l'algorisme de selecció: Repetidament, triar el node amb l'element més *gran* de la pila i traslladar-lo a la primera posició d'una nova cadena de nodes, que al final conté doncs els elements ordenats de petit a gran.

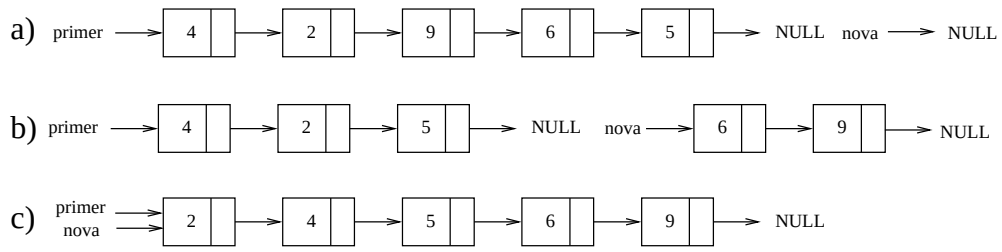
Digueu el codi que cal posar a `CONDICIO_1`, `CONDICIO_2`, `INSTRUCCIONS_1` i `INSTRUCCIONS_2` per implementar aquesta idea en aquest codi:

```
// Pre: el p.i. és una pila P
// Post: el p.i. conté els mateixos elements de P, però ordenats
// del més petit (al cim) al més gran (al fons)
void ordena() {
    node* nova = NULL;
    // a)
    while (CONDICIO_1) {
        node* p = primer;
        node* max = primer;
        node* ant_max = NULL;
        while (CONDICIO_2) {
            INSTRUCCIONS_1
        }
        INSTRUCCIONS_2
        // b)
    }
    primer = nova;
    // c)
}
```

sabent que es tracta de mantenir aquests dos invariants:

- *Invariant del bucle extern:* `nova` apunta a una cadena de nodes que conté els elements més grans de `P`, ordenats de més petit a més gran; `primer` apunta a la cadena de nodes que resulta d'eliminar en `P` els elements que són a la cadena apuntada per `nova`.
- *Invariant del bucle intern:* `primer` no és NULL, `p` apunta a un node de la cadena que comença a `primer`, `max` apunta a un node que té el valor màxim dels que són entre l'apuntat per `primer` i l'apuntat per `p`, ambdós inclosos, i `ant_max` apunta al node anterior a l'apuntat per `max`, excepte si aquest no té anterior, cas en què `ant_max` és NULL.

En l'esquema següent es mostra l'estat dels apuntadors `primer` i `nova` en els punts a) i c) de la plantilla de codi, i en el punt b) després d'executar **dues** iteracions del bucle extern.



No cal justificar el codi. Els invariants que us donem són per guiar la vostra implementació. Es valorarà molt l'eficiència de la solució proposada.

Problema 2 (5 punts)

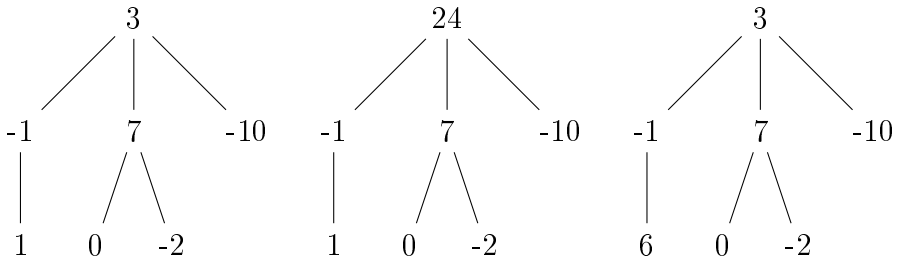
Considerem la representació habitual amb nodes de la classe *ArbreGen* per manegar arbres generals genèrics d'elements de tipus `T`:

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

Suposem arbres generals on els elements són nombres enters. Volem afegir-hi una nova operació pública amb la següent especificació

```
int suma_max_subarbre() const;
/* Pre: el p.i. no és buit */
/* Post: el resultat és la suma del subarbre del p.i. amb suma màxima */
```

Els tres arbres generals següents



són pràcticament idèntics. La crida a `suma_max_subarbre` amb l'arbre de l'esquerra com a p.i. ha d'obtenir 5 (el subarbre de suma màxima és el que té arrel 7). Amb l'arbre del centre ha d'obtenir 19 (el subarbre de suma màxima és el que té arrel 24). Amb l'arbre de la dreta ha d'obtenir 6 (el subarbre de suma màxima és el que té arrel 6). Fixeu-vos que el p.i. ha de quedar intacte després d'executar aquesta operació.

Heu de dissenyar una implementació d'aquesta operació que **no usi cap operació pública dels arbres generals** (per això no les recordem en aquest enunciat) sinó que accedeixi directament als atributs de la classe *ArbreGen*.

Es valoraran negativament solucions que realitzen visites, còpies o esborraments de nodes innecessàriament.

Rumieu els Problemes 2.1 i 2.2 conjuntament.

Problema 2.1 (1 punt)

Doneu l'especificació (capçalera, pre i post) d'una acció o funció d'immersió privada de nom `i_suma_max_subarbre` que permeti solucionar el problema. Recomanem que aquesta operació no tingui paràmetres que siguin arbres generals.

Problema 2.2 (1 punt)

Doneu una implementació eficient de l'operació `suma_max_subarbre`, fent ús de `i_suma_max_subarbre`.

Problema 2.3 (3 punts)

Doneu una implementació eficient de l'operació `i_suma_max_subarbre`.