Universitat Politècnica de Catalunya
Facultat d'Informàtica de Barcelona

Titulació: Grau en Enginyeria Informàtica
Assignatura: Programació 2 (PRO2)
Duració: 1h 30m

Curs: Q1 2020–2021 (1r Parcial)
Data: 5 de novembre de 2020

1. **(5 punts)** Considereu la següent acció que modifica un vector d'enters v reordenant els seus elements de manera que els negatius queden en un subvector a l'esquerra, els zeros en un subvector al mig, i els positius en un subvector a la dreta. És irrellevant en quin ordre queden els elements internament tant en el subvector de negatius com en el subvector de positius. L'acció té dos paràmetres enters de sortida a i b que marquen la frontera entre els tres subvectors. Noteu però què, depenent del contingut de v, un o dos d'aquests subvectors podria ser buit.

```
// Pre: v = V
// Post: v és una permutació de V, 0 ≤ a ≤ b ≤ v.size(),
//        tots els elements de v[0...a−1] són negatius,
//        tots els elements de v[a...b−1] són zero,
//        tots els elements de v[b...v.size()−1] són positius

void reordena(vector<int>& v, int& a, int& b) {
    a = [          ];
    b = [          ];
    int c = [          ];
    while ([          ]) {
        if ([          ]) {
            [                              ]
        }
        else if ([          ]) {
            [                              ]
        }
        else {
            [                              ]
        }
    }
}
```

Dins del codi de `reordena` no es pot cridar a l'operació `sort`, però sí es pot cridar a l'operació auxiliar `swap`:

```
// Pre: v = V, 0 ≤ i,j ≤ v.size() − 1, V[i] = X, V[j] = Y
// Post: v[i] = Y, v[j] = X, i per a les altres posicions diferents k
//       entre 0 i v.size() − 1 tenim que v[k] = V[k]
void swap(vector<int>& v, int i, int j);
```

Us proposem el següent invariant **incomplet**, el qual introdueix la variable local entera c:

```
// Inv: v és una permutació de V,
//      tots els elements de v[0...a−1] són negatius,
//      tots els elements de v[a...b−1] són zero,
//      tots els elements de v[c...v.size()−1] són positius
```

Donat aquest invariant, penseu en el significat del subvector $v[b\ldots c-1]$ i en quin hauria de ser el seu estat inicial (abans d'entrar en el bucle) i el seu estat final (després de sortir del bucle). Llavors,

a) (0,5 punts) Completeu l'invariant proposat per al bucle de `reordena` escrivint només les condicions que cal afegir.

b) (0,5 punts) Doneu una funció de fita.

c) (2,5 punts) Ompliu el codi faltant (només els llocs indicats per les capses). Cada capsa s'ha d'omplir amb una expressió o amb una o més instruccions.

d) (1,5 punts) Justifiqueu la correctesa del codi, incloent les inicialitzacions, la condició del bucle, el cos del bucle i per què acabarà sempre (usant l'invariant i la funció de fita donats).

SOLUCIÓ:

a) (0,5 punts) Completeu l'invariant proposat per al bucle de `reordena` escrivint només les condicions que cal afegir.

$0 \leq a \leq b \leq c \leq v.size()$, $v[b\ldots c-1]$ són els elements no reordenats encara

La primera condició és la que cal per a completar l'invariant. La segona és merament descriptiva i no és de fet necessari incloure-la.

b) (0,5 punts) Doneu una funció de fita.

$f = c - b$

que sempre serà ≥ 0, perquè $b \leq c$ forma part de l'invariant.

c) (2,5 punts) Ompliu el codi faltant (només els llocs indicats per les capses). Cada capsa s'ha d'omplir amb una expressió o amb una o més instruccions.

```
void reordena(vector<int>& v, int& a, int& b) {
    a = 0 ;
    b = 0 ;
    int c = v.size() ;
    while (b < c ) {
        if (v[b] < 0 ) {
            swap(v,a,b);  ++a;  ++b;
        }
        else if (v[b] > 0 ) {
            --c;  swap(v,b,c);
        }
        else {
            ++b;
        }
    }
}
```

Una solució alternativa, que també compleix l'invariant i té les mateixes inicialitzacions i condició del bucle que l'anterior, tindria el següent cos del bucle:

```
        if (v[c-1] < 0 ) {
            swap(v,b,c-1);  swap(v,a,b);  ++a;  ++b;
        }
        else if (v[c-1] > 0 ) {
            --c;
        }
        else {
            swap(v,b,c-1);  ++b;
        }
```

Qualsevol altre possible codi que pugui satisfer la Pre/Post, però que no reordeni a cada iteració del bucle un element del subvector $v[b \dots c-1]$, no satisfà l'invariant donat. Això inclou codis on el rol dels índexs b i c s'ha intercanviat en relació a l'invariant, però que al sortir del bucle, quan $b = c$, satisfan la Post. També inclou codis que deixen els tres subvectors ordenats a l'esquerre i on el subvector $v[c \dots v.size()-1]$ a la dreta conté els elements que queden per reordenar.

d) (1,5 punts) Justifiqueu la correctesa del codi, incloent les inicialitzacions, la condició del bucle, el cos del bucle i per què acabarà sempre (usant l'invariant i la funció de fita donats).

- (Inicialitzacions) Amb les inicializacions donades es compleix $0 \leq a \leq b \leq c \leq v.size()$ i, a més, $v[0 \dots a-1] = v[a \dots b-1] = v[0 \dots -1]$ és un subvector buit que compleix qualsevol propietat sobre els seus elements i $v[c \dots v.size()-1] = v[v.size() \dots v.size()-1]$ és un altre subvector buit que també compleix qualsevol propietat sobre els seus elements. També podem dir que el subvector $v[b \dots c-1] = v[0 \dots v.size()-1]$ és igual a V i, per tant, tot el vector resta per reordenar.

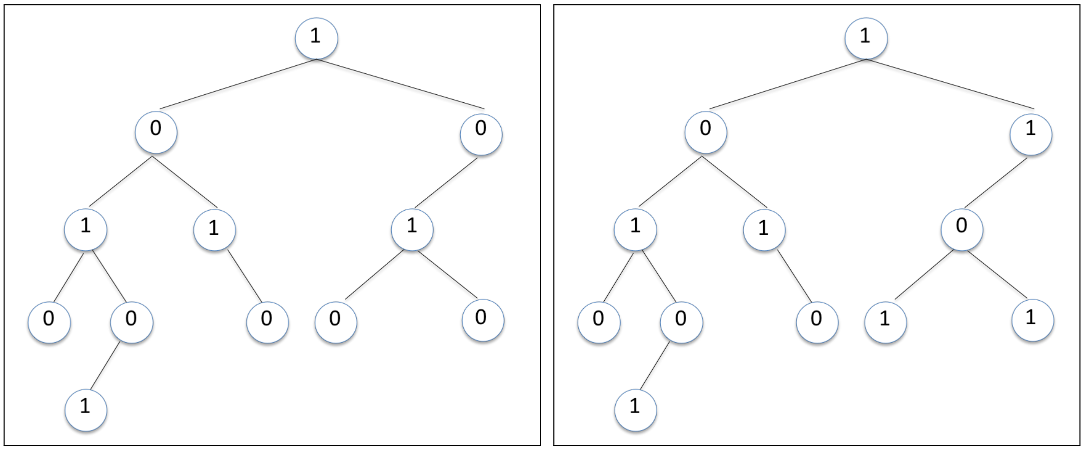
- (Condició del bucle) La negació de la condició del bucle és $b \geq c$. Això conjuntament amb la part de l'invariant que diu $b \leq c$ implica que al sortir del bucle tenim $b = c$. Aquesta condició $b = c$ i l'invariant implica que es compleix la Post al sortir del bucle (i també podem dir que el subvector $v[b \dots c-1] = v[c \dots c-1]$ és un subvector buit i no queda cap element de V per reordenar).
- (Cos del bucle) Justificació del primer cos del bucle presentat:
  - (Completesa) Amb les tres branques del condicional cobrim tots els casos possibles del valor de $v[b]$.
  - (Cas $v[b] < 0$) Hem d'intercanviar els valors de $v[a]$ i $v[b]$ per a poder estendre el subvector de negatius $v[0 \dots a-1]$ a $v[0 \dots a]$ i desplaçar el subvector de zeros de $v[a \dots b-1]$ a $v[a+1 \dots b]$. Un cop fet això, incrementant a i b es reestableix l'invariant.
  - (Cas $v[b] > 0$) Hem d'intercanviar els valors de $v[b]$ i $v[c-1]$ per a poder estendre el subvector de positius $v[c \dots v.size()-1]$ a $v[c-1 \dots v.size()-1]$ i (abans o després) decrementar c per a reestablir l'invariant.
  - (Cas $v[b] == 0$) Només cal incrementar b per a estendre el subvector de zeros $v[a \dots b-1]$ una posició més a la dreta i que es torni a complir l'invariant.
- (Acabament) La funció de fita $f = c-b$ és estrictament positiva quan s'entra en el cos del bucle per la condició d'entrada $b < c$. I $f = c - b$ decreix estrictament a cada iteració perquè a les tres branques del condicional o bé s'incrementa b o bé es decrementa c.

2. **(5 punts)** Un arbre 0-1 és un arbre binari amb valors que són només zeros i uns, que compleix que, per a tot node n, si el valor a n és 0 aleshores el valor a les arrels del seu fill dret i del seu fill esquerre (si existeixen) és 1 i, recíprocament, si el valor a n és 1 aleshores el valor a les arrels del seu fill dret i del seu fill esquerre (si existeixen) és 0.

   Es demana escriure i justificar la correcció d'una funció recursiva que donat un arbre binari d'enters ens digui si és un arbre 0-1. L'especificació Pre/Post de la funció és:

```
// Pre: a conte nomes zeros i uns
bool arbre01(const BinTree<int> &a);
// Post: El resultat ens diu si a es un arbre 0-1
```

   Per exemple, l'arbre a l'esquerra és un arbre 0-1, però l'arbre a la dreta no ho és pas:



a) (2,5 punts) Implementa la funció `arbre01`.

b) (2,5 punts) Justifica la correcció i acabament de la teva implementació de la funció `arbre01`.

SOLUCIÓ:

a) (2,5 punts) Implementa la funció `arbre01`.

```
// Pre: a conte nomes zeros i uns
bool arbre01(const BinTree<int> &a){
     if (a.empty()) return true;
     else {
        BinTree<int> b = a.left();
        BinTree<int> c = a.right();
        bool ab = b.empty() or (b.value() != a.value());
        bool ac = c.empty() or (c.value() != a.value());
        return ab and ac and arbre01(b) and arbre01(c);
     }
}
// Post: El resultat ens diu si a es un arbre 0-1
```

b) (2,5 punts) Justifica la correcció i acabament de la teva implementació de la funció `arbre01`.

- *Terminació:* Prenem com funció de mida $|a| = a.size()$, es a dir el nombre de nodes d'a. Clarament, si $|a| = 0$ ens trobem al cas base. Per altra banda, a cada crida recursiva la funció de mida decreix.
- *Cas base:* Si a és buit la resposta ha de ser *true*, com fa la funció.
- *Els arguments a les crides recursives compleixen la Pre:* Si l'arbre a només conté zeros i uns, evidentment els seus fills també contindran només zeros i uns.

- *Cas general:*
    - Les variables ab i ac ens diuen si el valor a l'arrel de a es diferent al valor dels arrels dels seus fills esquerre i dret, si existeixen.
    - Per altra banda, com que a les crides $arbre01(b)$ i $arbre01(c)$, b i c compleixen la Pre, podem aplicar inducció i suposar que les seves respostes són correctes, es a dir, que retornen *true* si i només si tots els seus nodes compleixen que el seu valor és diferent del valor a l'arrel dels seus fills.
    - Com a conseqüència, el que retorna la funció, $ab\ \ and\ ac\ and\ arbre01(b)\ and\ arbre01(c)$ ens diu si a és un arbre 0-1.