Solució parcial Programació 2

Enginyeria Informàtica
Maig 2014

Avisos generals:

- **Aquest document mostra una solució possible entre les mereixedores de nota màxima.**
- **Alguns elements de la justificació de l'exercici 1.2 són d'una precisió especialment elevada. Es pot obtenir una nota màxima en aquest exercici sense arribar a aquesta precisió.**

Comentaris addicionals que no formen part de la solució

Problema 1

El problema a resoldre és de cerca perquè el nombre màxim de senars no pot superar k i, si es recorre `seq` des del principi fins al final la primera aparició d'un bloc amb k senars ja permet complir la post.

Una solució naïf (i ineficient) consisteix a recórrer `seq` des del principi i per a cada posició de la llista calcular des de zero el nombre de senars de cada subseqüència anant actualitzant *sub* i una variable local addicional `senars_max` per al valor del màxim nombre de senars en els blocs visitats. Això du a un algorisme amb dos bucles aniuats que realitza de l'ordre de nk operacions on $n =$ `seq.size()`. Una solució eficient del problema (realitzant de l'ordre de n operacions) es basa en afegir tres variables locals:

- `senars`, el nombre de senars del darrer bloc visitat
- `i1`, que assenyala l'inici del darrer bloc visitat.
- `i2`, que assenyala el final del primer bloc no visitat.

Això ens porta a una solució basada en dos bucles consecutius, un de senzill per obtenir un valor inicial per a `sub` i `senars_max` i un altre més complex per anar actualitzant-los amb la resta de blocs.

Problema 2

A part de l'especificació de la funció d'immersió donada a la solució, n'hi ha d'altres de vàlides, per exemple:

Pre: `t` no és buit, `t` $= T$
Post: `suma` és la suma dels elements de T, `mida` és la mida de T,
`m` és la mitjana dels subarbres no buits de T.

Tot i que aquesta funció d'immersió pot donar lloc a un examen de nota màxima, no dona lloc a un codi tant compacte o a una justificació tan senzilla com el de la solució que hem triat.

Problema 1: *Subseqüència amb més senars*

Exercici 1.1

Previs:

- Definim un bloc com una subseqüència de llargada `k` de `seq`.
- El primer element d'una seqüència és l'1-èssim, el 2n element és el 2-èssim,...,etc.

```
1.  void mes_senars(const list<int> &seq, int k, list<int>::const_iterator &sub) {
2.    /* Pre: k >= 1 */
3.    /* Post: sub assenyala el primer element de la subseqüència de seq de llargada k amb
4.             més nombres senars; cas que hi hagi més d'una subseqüència amb un nombre màxim de
5.             senars, sub assenyala el primer element de la que es troba més a prop del principi
6.             de seq; si seq.size() < k (no hi ha cap subseqüència de llargada k) aleshores
7.             sub = seq.end() */
8.    if (seq.size() < k) sub = seq.end();
9.    else {
10.      int j = 0;
11.      int senars = 0;
12.      list<int>::const_iterator i2 = seq.begin();
13.      while (j < k) {
14.         /* Inv 1: (a) 0 <= j <= k;
15.                   (b) i2 assenyala l'element (j+1)-èssim de seq (que pot ser el sentinella);
16.                   (c) senars indica el nombre de senars en els j primers elements de seq */
17.         if (*i2 % 2 == 1) ++senars;
18.         ++i2;
19.         ++j;
20.      }
21.      /* A: i2 assenyala el (k+1)-èssim element de seq;
22.            senars indica el nombre de senars en els k primers elements de seq */
23.      int senars_max = senars;
24.      list<int>::const_iterator i1 = seq.begin();
25.      sub = i1;
26.      while (i2 != seq.end() and senars_max < k) { // la 2a condició indica que és un probl. de cerca
27.         /* Inv 2: (a) i1 i i2 assenyalen elements de seq (o son seq.end());
28.                   (b) si i1 i i2 assenyalen, respectivament, els elements l1-èssim i l2-èssim
29.                       de seq aleshores l2 - l1 = k;
30.                   (c) senars_max és el nombre màxim de senars en els blocs que
31.                       comencen a l'element assenyalat per i1 o anteriors;
32.                   (d) sub assenyala l'inici del primer bloc amb senars_max senars;
33.                   (e) senars és el nombre de senars del bloc que comença on assenyala i1; */
34.         if (*i1 % 2 == 1) --senars;
35.         if (*i2 % 2 == 1) ++senars;
36.         ++i1;
37.         if (senars > senars_max) {
38.            sub = i1;
39.            senars_max = senars;
40.         }
41.         ++i2;
42.      }
43.   }
44. }
```

Exercici 1.2

Comencem justificant la correctesa del `if` extern. Quan `seq.size()` $<$ k no hi ha cap bloc a `seq`. Per tant, segons la post cal fer `sub = seq.end()` que és precisament el que el codi de l'`if` fa en aquest cas. Ens queda la part feixuga, que es justificar el codi de l'`else`. Això requereix definir `A`, un estat intermig entre el final del primer bucle i l'inici del segon. Per justificar el 1er bucle cal raonar que la seva execució porta a l'estat `A`. Per justificar el 2on bucle cal raonar que la seva execució a partir de l'estat `A` du a la post.

Correctesa del primer bucle

Inicialitzacions

La instrucció `j = 0;` (línia 10) satisfà la condició $0 \leq$ `j` $\leq$ `k` de l'invariant perquè `k` ≥ 1 segons la pre de `mes_senars`. (si `k` < 0 aleshores no es podria satisfer l'invariant). Amb la instrucció `j = 0;`, l'invariant obliga a què

- `i2` assenyali el 1er element de `seq`, que és el que aconsegueix la instrucció `i2 = seq.begin()`
- `senars` indiqui el nombre de senars en els 0 primers elements de `seq`, és a dir 0, que és el que aconsegueix la instrucció `senars = 0;`

Cos del bucle

Suposem que abans d'executar les instruccions del cos del bucle `j` val `J`. Suposant que Inv 1 (a) se satisfà al principi del cos del bucle, és a dir que $0 \leq$ `J` $\leq$ `k`, la instrucció `++j;` (línia 19) fa que es continuï satisfent Inv 1 (a) perquè segons la condició del bucle `J < k` (i per tant `J+1` $\leq$ `k`).

La instrucció `++j;` obliga a què

- `i2` esdevingui una referència a l'element (`J`+1)-èssim per satisfer Inv 1 (b) al final del bucle. Suposant que Inv 1 (b) és cert abans d'executar el cos del bucle, s'obté que `i2` ja està apuntant al `J`-èssim a l'inici del cos del bucle i per tant només cal avançar `i2` perquè apunti al (`J` +1)-èssim. Això és el que aconsegueix la instrucció `++i2;` (línia 18).
- `senars` indica el nombre de senars en els `J` +1 primers elements de `seq`. Suposant que Inv 1 (c) és cert abans d'executar el cos del bucle, s'obté que `senars` ja indica el nombre de senars en els `J` primers elements i per tant només cal incrementar `senars` si el (`J`+1)-èssim és senar per satisfer Inv 1 (c) al final del cos del bucle. Això és el que aconsegueix la línia 17.

Finalment cal comprovar que les operacions `*i2` i `++i2` son vàlides, és a dir, que respecten la pre de `*` i `++` respectivament. La pre de `*i2` i la de `++i2` és que `i2` assenyala un element de `seq` excloent el sentinella (`i2` $\neq$ `seq.end()`). D'una banda l'invariant afirma que `i2` referència l'element `j`-èssim de `seq`. D'una altra banda, per la condició del bucle sabem que `j < k` i per la branca de `if` extern en la que ens trobem, `k >= seq.size()`. Per tant, `i2` assenyala un element de `seq` amb `seq` $\neq$ `seq.end()`. Pel mateix motiu, `++i2;` respecta la pre de l'operació `++`.

Condició del bucle

Es tracta de raonar que Inv 1 i la negació de la condició del bucle permeten obtenir l'estat `A`.

Quan no se satisfà `j < k`, Inv 1 (a) implica `j = k` (atès que $0 \leq$ `j` $\leq$ `k`). Per tant, el mateix invariant ens indica que `i2` assenyala l'element (`k` + 1)-èssim de `seq` i que `senars` indica el nombre de senars en els `k` primers elements de `seq` tal com s'espera a `A`.

Acabament

Funció fita: $f =$ `k - j`. $f \geq 0$ per Inv 1 (a) i f decreix a cada iteració (a cada iteració `j` s'incrementa en 1 mentre `k` roman constant).

Correctesa del segon bucle

Inicialitzacions

Inv 2 (a) i Inv 2 (b) se satisfan gràcies a `A` (`i2` assenyala el (`k`+1)-èssim element de `seq`) i gràcies a la linia 24 que fa que `i2` assenyali l'1-èssim (pel que fa a Inv 2 (b) fixeu-vos que la diferència de posicions és `k`). Tal com afirma `A`, `senars` indica el nombre de senars en els `k` primers elements de `seq`, aleshores `senars` també satisfà Inv 2 (e) gràcies a la línia 24. La instrucció `senars_max = senars` fa que se satisfaci Inv 2 (c) perquè `i1` assenyala l'inici del 1er bloc possible. Satisfent-se Inv 2 (c), la instrucció `sub = i1;` fa que es compleixi Inv 2 (d) perquè `i1` assenyala el primer bloc possible gràcies a la línia 24.

Cos del bucle

A l'inici del bucle, sabem per Inv 2 (a) que `i1` i `i2` assenyalen elements de `seq`. La instrucció `++i2;` fa que es continuï satisfent la part de Inv 2 (a) relativa a `i2` al final del cos del bucle perquè executar `++i2` sobre un element de `seq` (Inv 2 (a)) quan sabem que `i2!=seq.end()` gràcies a la condició del bucle, sempre deixarà `i2` sobre un element de `seq` (que pot ser el sentinella). La part relativa a `i1` de Inv 2 (a) se satisfà segur perquè, si Inv 2 (b) se satisfà al principi del bucle, llavors i1 assenyala elements que precedeixen els assenyalats per `i2` i ja hem argumentat que `i2` $\neq$ `seq.end()` al principi del bucle.

Si Inv (2) (b) se satisfà a l'inici del cos del bucle, es torna a satisfer al final perquè `i1` i `i2` avancen una posició a cada iteració.

Definim x com el bloc que comença a `i1` a l'inici del bucle i y com el bloc següent (a l'inici del cos del bucle `i2` assenyala el final d'y). La instrucció `++i2;` implica que `senars` ha de passar a ser el nombre senars de y per tal de satisfer Inv 2 (e). Segons Inv 2 (e), `senars` és, a l'inici del bucle, el nombre de senars de x. El bloc y comparteix els seus `k - 1` primers elements amb el bloc x. Les línies 34-35 s'encarreguen d'actualitzar el valor inicials de senars sobre x en el valor de senars sobre y mitjançat els elements no compartits de x i y per tal de complir Inv 2 (e). Si senars de y supera `senars_max`, cal actualitzar `sub` i `senars_max` per tal que continuïn satisfent Inv 2 (c) i (d). Això és el que aconsegueixen les línies 37-39.

Cal comprovar si `++i1;`, `++i2`, `*i1` i `*i2` respecten la pre de `++` i `*`, és a dir cal assegurar `i1`,`i2` assenyalen elements de `seq` amb `i1`,`i2` $\neq$ `seq.end()`. Això ja ho hem fet més amunt en raonar sobre Inv (a).

Condició del bucle

El bucle pot acabar perquè `i2` = `seq.end()` o `senars_max` $\geq$ `k` (o tots dos a la vegada). En el primer cas, Inv 2 (b) i (c) impliquen que `senars_max` és el nombre màxim de senars en tots el blocs de `seq` possibles, tal com espera la post. Tenint `senars_max` el valor esperat per la post, Inv (d) fa

que `sub` compleixi la post de forma immediata. En el segon cas, es té que `senars_max` = `k` pel fet que `senars_max` $\leq$ `k` per definició. A més, `senars_max` = `k` implica segons Inv 2 (d) que `sub` assenyala l'inici del primer bloc comprès per `senars_max` i per tant assenyala el bloc que cal segons la post perquè cap bloc posterior pot tenir més de `k` senars.

Acabament

Funció fita f = nombre d'elements de `seq` entre `i2` i `seq.end()`. $f \geq 0$ per Inv 2 (a). f decreix a cada iteració perquè a cada iteració s'avança `i2` mentre `seq` roman constant.

Problema 2: *Mitjana màxima d'un subarbre*

Exercici 2.1

Definim la immersió

```
void mm_i(Arbre<double> &t, double &suma, int &mida, double &m)
```

amb la següent especificació:

Pre: `t` $= T$
Post: `suma` és la suma dels elements de T, `mida` és la mida de T, i
si `mida > 0`, llavors `m` és la mitjana dels subarbres no buits de T.

Amb aquesta especificació ja podem implementar `mitjana_maxima`:

```
1.  double mitjana_maxima(Arbre<double> &t) {
2.      double m, suma;
3.      int mida;
4.      mm_i(t, suma, mida, m);
5.      return m;
6.  }
```

No cal comprovar la pre de `mm_i` a la crida de la línia 4 perquè és equivalent a cert. Atès que la pre de `mitjana_maxima` garanteix que `t` no és buit, després d'executar la línia 4 tenim que `mida` > 0 i per tant que `m` és la mitjana màxima dels subarbres no buits de T. És a dir, el valor retornat satisfà la post de `mitjana_maxima`.

Exercici 2.2

```
 1.  void mm_i(Arbre<double> &t, double &suma, int &mida, double &m) {
 2.      if (t.es_buit()) suma = mida = 0;
 3.      else {
 4.          Arbre<double> t1, t2;
 5.          double x = t.arrel();
 6.          t.fills(t1, t2);
 7.          double suma1, suma2, m1, m2;
 8.          int mida1, mida2;
 9.          mm_i(t1, suma1, mida1, m1);
10.          mm_i(t2, suma2, mida2, m2);
11.          suma = suma1 + suma2 + x;
12.          mida = mida1 + mida2 + 1;
13.          m = suma/mida;
14.          if (mida1 > 0 and m1 > m) m = m1;
15.          if (mida2 > 0 and m2 > m) m = m2;
16.      }
17.  }
```

Exercici 2.3

Cas base: T és buit. En aquest cas la suma dels elements de T és 0 (perquè no hi ha elements) i la seva mida és 0. Per tant la línia 2 fa el que s'ha de fer per satisfer la post.

Cas inductiu: T no és buit. Sigui X el valor de l'arrel de T i siguin $T1$ i $T2$ els arbres esquerre i dret de T. Després d'executar les línies 5 i 6 tenim la condició

$\texttt{x} = X$, $\texttt{t1} = T1$, i $\texttt{t2} = T2$.

Després d'executar la crida recursiva de la línia 9 la hipòtesi d'inducció ens dóna:

H1: `suma1` és la suma dels elements de $T1$, `mida1` és la mida de $T1$, i
si $\texttt{mida1} > 0$, llavors `m1` és la mitjana màxima dels subarbres no buits de $T1$

Fixem-nos que la línia 9 no afecta les variables `x` i `t2`, o sigui que després d'executar-la seguim tenint $\texttt{x} = X$ i $\texttt{t2} = T2$. Per tant, després d'executar la crida recursiva de la línia 10 la hipòtesi d'inducció ens dóna:

H2: `suma2` és la suma dels elements de $T2$, `mida2` és la mida de $T2$, i
si $\texttt{mida2} > 0$, llavors `m2` és la mitjana màxima dels subarbres no buits de $T2$

La línia 10 no afecta les variables `x`, `suma1`, `mida1`, `m2` o sigui que $H1$ segueix sent vàlida després d'executar-se la línia 10, i a més $\texttt{x} = X$. Combinant això últim amb $H1$ i $H2$, després d'executar la línia 10 tenim que la suma dels elements de T és $\texttt{suma1} + \texttt{suma2} + \texttt{x}$ i la mida de T és $\texttt{mida1} + \texttt{mida2} + 1$. Per tant les línies 11 i 12 fan que se satisfaci la part de la post relativa a les variables `suma` i `mida`. Finalment, les línies 13, 14 i 15 s'encarreguen d'assignar el màxim de les mitjanes dels subarbres no buits de T a `m` per tal que se satisfaci la part de la post relativa a `m`. Primer s'assigna a `m` la mitjana de T, i després s'actualiza si s'escau amb les mitjanes màximes dels subarbres no buits de $T1$ i $T2$, que per $H1$ i $H2$ són `m1` i `m2` sempre que $\texttt{mida1} > 0$ i $\texttt{mida2} > 0$.

Finalment cal comprovar que les pres de les operacions i les crides recursives estan garantides quan s'executen. Per les crides recursives no cal comprovar res perquè la pre de `mm_i` és equivalent a cert. La crida a `arrel()` de la línia 5 necessita que `t` no estigui buit, la qual cosa està garantida per la condició de l'`if` que porta al cas inductiu. La crida a `fills()` a la línia 6 també requereix que `t` no sigui buit, i a més requereix que `t1` i `t2` siguin buits, les quals coses estan garantides perquè les línies 5 i 6 no afecten a `t`, `t1` o `t2`.