Correzione Primo Esonero, Esercizio 3

Ivano Salvo – Sapienza Università di Roma

Anno Accademico 2011-12

Esercizio 3: *Diciamo che un indice k di un vettore di interi a lungo n ($0 \leq k < n$) è il* baricentro *di a, se la somma degli elementi di a prima di k è uguale alla somma degli elementi a cominciare da k fino ad n. Formalmente: $\sum_{i=0}^{k-1} a[i] = \sum_{i=k}^{n-1} a[i]$.*

Punto 1: *Si scriva una funzione C* `int baricentro(int a[], int n, int *k)` *che restituisce* 1 *se esiste almeno un baricentro nel vettore* `a` *e 0 altrimenti. Quando torna* 1*, essa carica nel parametro* `k` *l'indice del baricentro trovato. Si valuti la complessità della soluzione.*

Indichiamo per semplicità con $sp_k = \sum_{i=0}^{k-1} a[i]$ la somma del prefisso formato dai primi k elementi del vettore, e con con $ss_k = \sum_{i=k}^{n-1} a[i]$ la somma del suffisso formato dagli ultimi $n-k$ elementi del vettore. Se un certo indice k è un baricentro si avrà che $sp_k = ss_k$. Il problema consiste quindi nel trovare un k (se esiste) per cui $sp_k = ss_k$.

In generale il valore di sp_k e ss_k può essere facilmente calcolato, ad esempio con la funzione `sommavet` in Fig. 1. Dovrebbe essere chiaro che su un qualsiasi indice k le chiamate `sommavet(a, 0, k);` e `sommavet(a, k, n);` costano complessivamente n somme. E dovrebbe anche essere chiaro che si fa del lavoro inutile chiamando `sommavet` su indici consecutivi, perchè per $0 \leq k \leq n-1$ abbiamo soddisfatte le relazioni $sp_{k+1} = sp_k + a[k]$ e $ss_{k+1} = ss_k - a[k]$.

Quanto detto sopra, ci suggerisce di fare un ciclo che scandisce il vettore e di considerare due variabili `sp` e `ss` costantemente aggiornate ad ogni iterazione k ai valori di sp_k e ss_k rispettivamente. A ogni iterazione, l'aggiornamento del valore di `sp` e `ss` con le assegnazioni `sp=sp+a[k]` e `ss=ss-a[k]`, garantisce che venga mantenuto l'invariante $sp = sp_k \wedge ss = ss_k$.

```
  void sommavet(int a[], int inf, int sup){
     int s=0;
     for (int i=inf; i<sup; i++) s=s+a[i];
         /* INV: ss=sum[0<=i<n] a[i] */
     return s;
}
```

Figura 1: Funzione per la somma di un segmento di vettore

```
int baricentro(int a[], int n, int* b){
    /* PREC: n = lun a */
    int ss=0; int sp=0;

    for (int k=0; k<n; k++) ss+=a[k];

    for (int k=0; k<n && ss!=sp; k++) {
    /* INV: sp=sum[0<=j<k].a[j] & ss=sum[k<=j<n].a[j] */
        sp+=a[k];
        ss-=a[k];
    }
    if (ss!=sp) return 0;
    *b=i;
    return 1;
}
```

Figura 2: Funzione per calcolare il baricentro di un vettore

In questa situazione, scandendo tutti gli indici k del vettore da 0 a $n-1$ avremmo che trovando un k per cui $ss_k = sp_k$ avremo stabilito l'esistenza di un baricentro. Mentre se arriviamo in fondo al vettore con $k = n$ allora avremmo stabilito che $\forall k.sp_k \neq ss_k$ e quindi la non-esistenza di nessun baricentro. Questo ci suggerisce le guardie con le quali eseguire il ciclo: `(i<n && sp!=ss)`.

Resta da capire come sia possibile entrare nel ciclo soddisfacendo l'invariante. Chiaramente, per $k = 0$, ho $sp_0 = 0$ e di conseguenza sarà sufficiente inizializzare `sp` a 0. Viceversa, ss_0 sarà uguale alla somma di tutti gli elementi del vettore, per cui sarà necessario fare una prima scansione completa del vettore per fare questo calcolo. Globalmente, si eseguono un numero di operazioni proporzionale a $2n$, molto meglio dell'algoritmo ingenuo tratteggiato all'inizio che chiama ripetutamente `sommavet`, che richiede n^2 operazioni. Il codice completo in Figura 2

Domanda 2: *Sotto la precondizione che gli elementi del vettore siano tutti positivi, è possibile migliorare l'efficienza della funzione (anche senza migliorare la complessità asintotica del caso pessimo)? Motivare la risposta (eventualmente con il codice di una funzione* `baricentroPos` *che sfrutta questa precondizione).*

In questo caso possiamo sfruttare il fatto che per ogni $0 \leq k < n-2$ ho che $sp_k \leq sp_{k+1}$ e analogamente $ss_k \geq ss_{k+1}$. Siccome sp_k è una successione monotona crescente, ed ss_k è una successione monotona decrescente, posso uscire dal secondo ciclo while non appena $sp > ss$: in tal caso, anche se non ho trovato un baricentro, non c'è più speranza di trovarlo. Quindi modificando la condizione `sp!=ss` con `sp<ss` si migliora il caso pessimo, evitando in molti casi di andare fino al fondo del vettore quando non necessario. Ovviamente il numero di operazioni rimane sempre $\mathcal{O}(n)$.

È tuttavia possibile fare molto meglio e dirigersi senza indugi verso l'eventuale baricentro, senza il preventivo calcolo della somma di tutti gli elementi, sempre ra-

```
int baricentroPos(int a[], int n, int* b){
  /* PREC: n = lun a & forall 0<=i<n. a[i]>=0 */
   int ss=0; int sp=0; int k=0; int l=n-1;

   while (k<=l) {
    /*INV: sp=sum[0<=i<k].a[i] & ss=sum[l<=i<n].a[i] */
       if (sp<=ss) sp+=a[l++];
              else ss+=a[k--];
   }
   if (ss!=sp) return 0;
   *b=i;
   return 1;
}
```

Figura 3: La funzione baricentro nel caso vettori di interi positivi

gionando sul fatto che sp_k e ss_k assumono valori monotoni rispettivamente crescenti e decrescenti. Infatti, fissati due indici k ed l con $k < l$ ho che $sp_k < ss_l$ implica che il baricentro, se esiste, sia un indice b tale che $k < b \leq l$: infatti posso raggiungere un punto di equilibrio solo incrementando il valore di sp_k. Quindi lo posso cercare incrementando k. Analogamente, se $sp_k > ss_l$, allora $k \leq b < l$ e quindi posso cercarlo decrementando l. Quando infine $sp_k = ss_l$ e $k < l$, è indifferente avanzare a sinistra o a destra. Avanzare da entrambi i lati è possibile, ma costringe a trattare il caso particolare $k + 1 = l$, il che non è una buona idea in termini di chiarezza del programma risultante. La morale della favola è che posso cercare l'eventuale indice b del baricentro calcolando sp_k e ss_l, cominciando con $k = 0$ e $l = n - 1$ e muovendomi veso il centro incrementando k e decrementando l a seconda dei valori di sp_k e ss_l.

Siamo sempre nel reame delle soluzioni lineari, ma abbiamo dimezzato il numero massimo di operazioni. Un eccellente risultato! La funzione `baricentroPos` è riportata in Figura 3.

Domanda 3: FACOLTATIVO: *Scrivere una funzione ricorsiva che risolve lo stesso problema con un'*unica *scansione del vettore* [**Sugg:** *scrivere una funzione ausiliaria con parametri aggiuntivi e ricordare che la scansione ricorsiva di un vettore, di fatto, percorre il vettore 2 volte, all'andata e al ritorno dalle chiamate ricorsive*].

La funzione ricorsiva che traduce la funzione in Fig. 2 con metodi standard porta a scrivere due scansioni ricorsive, ciascuna delle quali mima il comportamento di una delle due scansioni nella funzione in in Fig. 2. L'idea chiave per scrivere una funzione che fa un un'unica scansione è appunto riflettere sul fatto che la scansione ricorsiva di un vettore, di fatto, percorre il vettore 2 volte, all'andata e al ritorno

```
  void printvAR(int a[], int i, int n){
      if (i==n) printf("\n");
         else { printf(" %d",a[i]);
                printvAR(a, i+1, n);
                printf(" %d",a[i]);
              }
}
```

Figura 4: funzione che stampa un vettore "dritto" e "rovescio".

```
int barRecAux(int a[], int i, int n, int sp, int* b){
    int ss;

    if (i==n) return 0;
    sp += a[i];
    ss = barRecAux(a, i+1, n, sp, b);
    if (*b>=0) return ss;
    if (ss==sp) {*b=i+1; return ss; }
    else return ss+a[i];
}

int baricentroRec(int a[], int n, int* k){
    *k=-1;

    barRecAux(a, 0, n, 0, k);
    if (*k<0) return 0;
    return 1;
}
```

Figura 5: funzione baricentro ricorsiva in un'unica passata

dalle chiamate ricorsive. Forse il modo più efficace per convincersene è eseguire la funzione di stampa di un vettore in Fig. 4.

Detto questo, la soluzione dovrebbe discendere da sè: possiamo propagare in "avanti" nelle attivazioni ricorsive il valore di *sp* usando un parametro ausiliario, e propagare all'"indietro" il valore di *ss* usando il valore di ritorno della funzione. In alternativa, si può usare un altro parametro anche per *ss*: comunicare col chiamante è leggermente più difficile che comunicare col chiamato, e richiede l'uso di un parametro passato per indirizzo. Seguiamo la prima strada (vedi Fig. 5). Un parametro passato per indirizzo ci serve comunque per trasmettere all'indietro il valore dell'eventuale baricentro trovato e sospendere ulteriori ricerche. Qui usiamo la convenzione che se la varibile `b` è -1 non ho ancora trovato un baricentro. Ovviamente i possibili valori di un baricentro sono valori positivi (da 0 a $n-1$) e quindi questa codifica è perfettamente lecita.