Compito di Programmazione 16 luglio 2012

Teoria

(i) Si consideri il seguente programma ricorsivo che riceve in input un albero binario:

```
bool F(nodo* a){
if(!a) return true;
if(F(a->left) !=F(a->right))
  return true;
else
  return false;
}
```

Dire quali sono i valori di verità che F calcola per i seguenti due alberi:

Sulla base dei valori di F per questi 2 alberi, cercate di trovare una POST-condizione valida per tutti i possibili alberi di input.

(ii) Considerate il seguente programma. In caso pensiate sia corretto, spiegate cosa calcola e cosa stampa. Se pensate sia sbagliato, spiegate in dettaglio gli errori. In ogni caso, per spiegare le vostre conclusioni, disegnate uno schema dettagliato che mostri la relazione tra le variabili e i puntatori in gioco. In nessun caso verranno assegnati punti per risposte senza spiegazioni.

```
int* f(int *p){int b=0,*x=&b; x=p+2; p++; return x-3; }

main() {int b[]={1,2,3,4,5},*q=b+2; *f(q)=*q; cout<<b[0]<<b[1]<<b[2]<<b[3]<<b[4];}
```

Programmazione.

Introduzione: il problema riguarda il pattern matching di un pattern char P[dimP] in un testo char T[dimT]. I match che cerchiamo sono completi (cioè tutto P va trovato) e non necessariamente contigui. Un tale match è specificato da dimP indici, i_0,...,i_(dimP-1), con 0 <= i_0 < i_1 < i_2 <...< i_(dimP-1) <= dimT-1 e tali che T[i_k] =P[k] per ogni k ∈ [0..dimP-1]. Il bilancio di un tale match è (i_(dimP-1) - i_0) +1 - dimP. Questo valore è il numero degli elementi di T compresi in [i_0..i_(dimP-1)] che non partecipano al match, intuitivamente potremmo dire che queste posizioni sono i buchi del match.

Esempio: T=[0,2,1,0,2,0,1] e P=[0,2,0]. Ci sono 2 match: [0,1,3] e [3,4,5], il bilancio del primo è (3-0)+1-3=1 mentre quello del secondo è (5-3)+1-3=0. Infatti il primo match ha un buco mentre il secondo è contiguo e quindi non ha buchi. Si osservi che i match devono essere completi.

Parte iterativa: si tratta di definire una funzione iterativa int match(char*T, int dimT, char*P, int dimP) che deve soddisfare la seguente coppia di PRE e POST:

PRE=(T e P sono array di caratteri, rispettivamente, di dimT e dimP elementi tutti definiti)

POST=(la funzione restituisce un intero x tale che, se x>=0, allora esiste in T almeno un match di P e x è il bilancio minimo tra i match esistenti, se invece x=-1, allora non esiste alcun match di P in T)

match deve usare una funzione ausiliaria bool M(char*T, int dimT, char*P, int dimP, int inizio, int & fine) che deve soddisfare la seguente coppia di PRE e POST:

PRE=(T e P sono array di caratteri, rispettivamente, di dimT e dimP elementi tutti definiti, inizio è in [0..dimT-1])

POST=(la funzione restituisce true sse esiste un match completo e tale che T[inizio]=P[0] ; inoltre, in questo caso, T[fine]= P[dimP-1] e quello trovato è il match a bilancio minimo che inizia nella posizione inizio)

Si osservi che la richiesta di POST che il match sia quello a bilancio minimo equivale a richiedere che sia il match che, una volta fissato l'inizio, finisce il più presto possibile, cioè con valore di fine minimo.

Oltre a match, si chiede di realizzare anche la funzione M, di specificare l'invariante del suo ciclo principale e di dimostrare, tramite questo invariante, la condizione d'uscita del ciclo stesso.

Parte ricorsiva: realizzare versioni ricorsive matchR e MR delle funzioni iterative match e M.

Mentre MR deve avere lo stesso prototipo di M, quello di matchR dovrà essere il seguente: int matchR(char*T, int dimT, char*P, int dimP, int inizio); in cui inizio sarà l'indice a partire da cui cercare il match a bilancio minimo.

Specificare PRE e POST di queste 2 funzioni ricorsive eventualmente modificando i PRE e POST dati in precedenza per match e M. Delineare la dimostrazione induttiva della funzione ricorsiva MR.

Attenzione:

1) la POST di MR (come quella di M) deve garantire che se il match è trovato esso inizia da T[inizio]. Questo richiede una PRE particolare per MR (diversa dalla PRE di M) e matchR deve invocare MR in modo che questa PRE sia verificata.

2) In matchR e MR è importante scorrere gli array T e P nel modo giusto. Suggerisco che T sia scorso incrementando l'indice inizio (e lasciando T e dimT fissi), mentre P sia scorso incrementando P e diminuendo dimP.