Scritto di Programmazione 28/6/2013

Note importanti: nella vostra home trovate 2 file, esercizio1.cpp ed esercizio2.cpp relativi all'esercizio iterativo e a quello ricorsivo, rispettivamente. Ciascuno di questi file contiene un main in cui si aprono i file di i/o (input1, output1 e input2, output2) e si eseguono le letture dei dati che i file di input devono contenere. Le letture spiegano i dati che i file di input devono contenere. I main eseguono anche le scritture finali sui file in output, quindi l'i/o è già fatto. Il file esercizio2.cpp contiene anche la funzione ricorsiva FR che è la base per risolvere il problema ricorsivo. Dovete consegnare i 2 file, esercizio1.cpp ed esercizio2.cpp, ampliati con le funzioni richieste nel seguito. Il comando di consegna (eseguito in una directory che contiene solo i file da consegnare) è: consegna esercitazione

Introduzione: sia per la parte iterativa che per quella ricorsiva, consideriamo un problema di pattern matching. Il pattern è un array P di dimP interi, il testo è un array T ad una dimensione di R*C elementi che però nell'esercizio "va visto" come un array di 2 dimensioni con R righe e C colonne . Tutti gli elementi di T sono definiti.

Innanzitutto sia ben chiaro che nell'esercizio verranno considerati solo match **contigui e non necessariamente completi**. La lunghezza di un match è il numero di elementi del pattern (partendo dal primo elemento) che vengono trovati in posizioni contigue di T. Il massimo match di P in T è il match di lunghezza massima di P in T.

Esempio: se T=[0 1 1 3 4 5] e P=[1 3 5] allora il massimo match di P in T è di lunghezza 2 e si trova in T[2..3]=[1 3]. E' un match parziale e resta da matchare l'ultimo elemento del pattern [5]. Se P=[1 3 4] allora troveremmo un match completo in T[2..4], mentre, se P=[2 0 1], allora il match massimo sarebbe di lunghezza 0 (T non contiene 2) e resterebbe da matchare l'intero pattern P.[1]

Parte iterativa: si tratta di "vedere" T come un array a 2 dimensioni R*C. Si considera la prima colonna di T e si cerca il match massimo di P su questa colonna. Se è completo, allora si continua a cercare di nuovo l'intero P nella seconda colonna. Se invece il match è parziale (anche di lunghezza 0) si cerca sulla seconda colonna il resto del pattern. E così di seguito fino a che il pattern finisce nel qual caso si cerca l'intero P nella successiva colonna (se c'è). L'output da produrre è un array di interi Q che ha C elementi e deve venire riempito in modo che per ogni i in [0..C-1], Q[i] sia la lunghezza del massimo match trovato nella colonna i.

Esempio: nel seguito T viene "visto" come fosse a 2 dimensioni con 3 righe e 5 colonne. Sia chiaro che T è invece ad 1 dimensione (int*).

T= 1 0 1 2 0 e P=[0 1 2]. Alla fine di F il risultato è Q=[2 1 0 0 3]

0 1 2 1 1

1 2 2 1 2

La soluzione da trovare consiste di 2 funzioni, F e match, che hanno i seguenti prototipi e pre- e post-condizioni:

[1] Nell'esempio non succede, ma in generale ci possono essere più mach con la stessa lunghezza massima tra tutti i match esistenti. Non ha alcuna importanza distinguerli tra loro.

PRE_F=(T ha R*C elementi definiti, P ha dimP elementi definiti, R, C e dimP sono maggiori di 0, Q ha C elementi di valore qualsiasi)

void F(int* T, int R, int C, int* P, int dimP, int*Q)

POST_F=(Q [0] è il massimo match di P nella colonna 0 di T, Q[1] il massimo match del resto di P nella colonna 1 e così via. Se il resto di P risulta vuoto allora si ricomincia con l'intero P)

PRE_match= PRE_F && (r in [0..R-1], c in [0..C-1], inizio in [0..dimP-1)]

int match(int*T, int r, int c, int R, int C, int*P, int dimP, int inizio)

POST_mach= (la funzione restituisce la lunghezza del match massimo del pattern P[inizio..dimP-1]tale che il match **inizia** nella posizione T[r][c] della colonna c e che **si estende il più possibile** in questa colonna, cioè in T[r..R-1][c])

Scrivere le due funzioni F e match. Associare ai cicli di F degli invarianti e delineare una dimostrazione di correttezza di F rispetto a PRE_F e POST_F.

Parte ricorsiva: si tratta di risolvere lo stesso esercizio della parte iterativa con la sola differenza che i match vanno cercati sulle righe di T (e non più sulle colonne). Per questo motivo Q ha R elementi (e non C). Per facilitare la soluzione viene data (nel file esercizio2.cpp) la funzione ricorsiva FR assieme alle sue pre- e post-condizioni. FR è la funzione principale della soluzione. Essa invoca un'altra funzione ricorsiva R1 che deve obbedire alle seguenti pre- e post-condizioni:

PRE_R1=(T ha C elementi definiti, P ha dimP elementi definiti)

int R1(int*T, int C, int*P, int dimP)

POST_R1=(la funzione restituisce la lunghezza del match massimo di P[0..dimP-1] in T[0..C-1])

La funzione R1 va realizzata. Il mio consiglio è di fare in modo che R1 invochi un'altra funzione ricorsiva. Altrimenti R1 rischia di complicarsi troppo. In caso si usi una seconda funzione, le sue pre- e post-condizioni vanno specificate.

In ogni caso si richiede di dimostrare induttivamente la correttezza della vostra funzione R1 rispetto a PRE_R1 e POST_R1.