Esercizio 1 proposto e discusso il 5/6/2015 (ispirato dall'esame del 31/3/2014)

Va consegnato entro il 30/6/2015

Si tratta di un esercizio di pattern matching (non contiguo e non completo) tra un array P con dimP elementi (il pattern) ed una lista L (il testo nel quale si cerca P). La lista è costituita di nodi con 2 campi info, come segue:

struct nodo{int info1,info2; nodo* next; nodo(int a=0, int b=0, nodo* c=0){info1=a; info2=b; next=c;}};

L'idea è che info1 sia il normale campo informativo, mentre info2 contiene l'indice del nodo. Quindi il primo nodo ha info2=0, il secondo ha campo info2=1 e così via.

Vediamo un esempio. I nodi della lista L verranno rappresentati da una coppia di interi tra parentesi, mentre il campo next è rappresentato da una freccia ->.

Esempio 1: sia L=(4,0)->(0,1)->(2,2)->(1,3)->(0,4)->(2,5)->(0,6) e P=[0,2,2]. Esiste un match completo di P che inizia nel nodo di indice 1 di L , i primi 2 eleementi di P vengono trovati nel campo info1 dei nodi di indice 1 e 2 di L, poi il match si interrompe e per matchare anche l'ultimo elemento di P (P[2]=2) è necessario saltare 2 nodi (i nodi di indice 3 e 4) . Questo match è descritto dalla seguente lista:

(1,2)->(2,1) che dice che per matchare P[0] è necessario saltare 1 nodo di L (la prima componente di (1,2)), poi, la seconda componente della coppia (1,2) dice che il match continua per 2 nodi contigui di L, poi la seconda coppia (2,1) dice che è necessario saltare 2 nodi su L per matchare P[2] su nodo seguente.

Vogliamo una funzione ricorsiva che data una lista L e un pattern P produca la lista che rappresenta il match di lunghezza massima esistente. Visto che possono esistere vari match di lunghezza massima, vogliamo sempre quello in cui ogni elemento di P viene matchato su L non appena è possibile farlo.

Si noti che la lista di coppie da produrre è dello stesso tipo nodo mostrato prima, ma, in questo caso, info1 è il numero di nodi da saltare per trovare il prossimo match, mentre info2 indica per quanti nodi il match si prolunga senza interruzzioni.

Esempio 2: supponiamo ora che L sia come nell'Esempio 1, ma che P=[0,2,2,4]. In questo caso la lista che rappresenta il match è ancora (1,2)->(2,1). P[3]=4 non trova match su L dopo il nodo di indice 5 che matcha P[2]. Nessun ulteriore nodo viene prodotto in questo caso di insuccesso. Quindi le coppie da produrre devono avere sempre info1>=0 e info2>0. In altri termini, una coppia viene prodotta solo quando c'è da rappresentare un match di qualche elemento di P in L.

Consideriamo ora che P=[4,0,0,0,3], allora la lista che rappresenta questo match è:

(0,2)->(2,1)->(1,1)

Si osservi che vengono matchati solo i primi 4 elementi di P. 4 è infatti la somma dei campi info2 dei 3 nodi. Inoltre la somma di entrambe le componenti dei 3 nodi ha valore 7 che indica che il match usa tutta la lista L che ha infatti 7 nodi.

Si tratta di scrivere una funzione ricorsiva f0 che soddisfi la seguente specifica:

PRE=(lista(L) è corretta, P[0..dimP-1] è definito)

nodo* f0(nodo*L, int*P; int dimP);

POST=(restituisce la lista del match di P[0..dimP-1] in L, come descritto prima)

Consiglio: conviene scrivere delle funzioni ausiliarie (anch'esse ricorsive) che calcoli i valori da inserire nella lista del match che va costruita. Cercate di scrivere pre- e post-condizioni di tutte le funzioni.