Esame di Programmazione 18/9/2020 prova di programmazione tempo 1 ora

Il programma richiede di eseguire un pattern matching dove il pattern è costituito dai campi info dei nodi di un albero binario ordinati secondo il percorso in larghezza, mentre il pattern è semplicemente un array di dimP interi.

Il problema va risolto in 2 passi:

1) Va definita una funzione **iterativa** nodoE* failter(nodo*r) che restituisce la lista L dei nodi estesi nodoE che puntano ai nodi di albero(r) percorsi in larghezza.
   La struttura nodoE è come segue:
   struct nodoE{nodo* info; nodoE* next; nodoE(nodo*a=0, nodoE*b=0){info=a; next=b;}};
   nodo è invece il solito nodo degli alberi binary.

2) Va definita una funzione **ricorsiva** int matchRic(nodoE*L, int*P, int dimP) che riceve la lista L prodotta nel passo 1, il pattern P e la lunghezza del pattern dimP e restituisce la massima lunghezza di match di P che si trova nei nodi puntati da L. Infatti il match da considerare può essere anche incompleto ed è spiegato nell'esempio che segue.

   Esempio 1: . Si consideri il seguente albero binario r:
   10(5(10(1(_,_),2(_,_)),2(3(_,_),1(_,_))),5(_,1(3(_,_),2(_,_)))) esso contiene 12 nodi.
   Allora la lista L che failter deve produrre contiene 12 nodi di tipo nodoE. Il primo di essi punta alla radice r (che ha info=10), il secondo al figlio sinistro (info=5) di r, il terzo al figlio destro (info=5) di r, il quarto al figlio sinistro del figlio sinistro di r (info=10) e così via. Quindi la sequenza dei campi info puntati dai nodoE di L sarà: 10,5,5,10,2,1,1,2,3,1,3,e 2. Nel seguito chiamiamo questa sequenza semplicemente L. Assumiamo ora che P=[2,5,10,2]. Se partiamo dal primo valore di L (10), esso matcha la terza posizione di P, dopo di che il prossimo valore di L (5) non matcha l'ultimo elemento di P che è 2. Per cui il match a partire dalla prima posizione di L ha lunghezza 1. Consideriamo ora il match che parte dalla seconda posizione di L: 5 matcha la seconda posizione di P, poi il successivo 5 non matcha nessun successivo elemento di P. Quindi anche questo match ha lunghezza 1. Partiamo ora dal terzo elemento di L che è 5, esso matcha il secondo elemento di P, il successivo 10 di L matcha il terzo elemento di P, e il successivo elemento di L (2) matcha l'ultimo elemento di P. Quindi abbiamo trovato un match di lunghezza 3. Non è difficile vedere che i successivi tentativi troveranno match più corti di 3 e quindi la risposta che matchRic dovrebbe dare è 3.

   Nota: eventuali funzioni ausiliarie di faiRic devono essere ricorsive e di matchIter devono essere iterative.

   Correttezza:
   1) definire un invariante significativo per il ciclo principale di failter,
   2) definire PRE e POST per matchRic e per eventuali funzioni ausiliarie.