Esercizio a tempo del 24/5/2018

Si tratta di costruire un albero binario bilanciato nella maniera che abbiamo già visto in vari esercizi. Poi si vuole percorrere l'albero in un ordine qualsiasi e si vuole costruire una lista concatenata che contiene un numero di nodi uguale ai nodi dell'albero e con gli stessi campi info ei nodi dell'albero, ma in ordine crescente. Vediamo il seguente esempio.

Esempio. Se l'albero è 'r'('a'(_,'m'(_,_)),'f'(_,'b'(_,_))) allora la lista concatenata da costruire dovrà avere 5 nodi (come l'albero) e i campi info dei nodi devono essere 'a'->'b'->'f'->'m'->'r'.

Visto che l'esercizio coinvolge sia alberi che liste, è necessario avere una struttura per i nodi degli alberi ed una per quelli delle liste. La struttura per gli alberi è nodoA, mentre quella per le liste è nodo, come al solito. In entrambi i casi il campo info ha tipo char.

Per risolvere l'esercizio conviene usare la libreria delle code vista nell'esercizio 1 della settimana del 30/4. Conviene però aumentare la libreria con una funzione push_list che inserisce un'intera lista (e non un solo nodo, come la push) alla fine della lista gestita da una coda. Più precisamente:

PRE=(lista(L) è ben formata (potenzialmente vuota) e Q è una coda ben formata) (una coda Q è detta ben formata se inizio e fine sono entrambi 0 oppure puntano al primo e all'ultimo nodo di una lista ben formata)

void push_list(nodo*L, coda & Q)

POST=(restituisce la coda Q, ben formata, che gestisce la lista iniziale a cui è stata concatenata alla fine lista(L))

Oltre a push_list, l'esercizio richiede di scrivere (almeno) 2 funzioni, una ricorsiva ed una iterativa.

La funzione ricorsiva è build_list che ubbidisce alla seguente specifica:

PRE=(albero(r) ben formato)

nodo* build_list(nodoA*r)

POST=(restituisce la lista ordinata che consiste di un numero di nodi pari a quelli di albero(r) e i cui campi info sono gli stessi di albero(r))

La funzione iterativa invece deve costruire una lista ordinata partendo da 2 liste ordinate. Quindi la sua specifica è la seguente:

PRE=(lista(L1) e lista(L2) ben formate)

nodo* fuse(nodo*L1,nodo*L2)

POST=(restituisce la lista ordinata costruita disponendo in ordine i nodi di L1 ed L2)

Consiglio: conviene che fuse usi la libreria delle code, in particolare, sia push che push_list.

ATTENZIONE: la funzione build_list deve costruire la lista finale e quindi deve creare nuovi nodi. Invece fuse, non deve creare nuovi nodi, ma solo concatenare opportunamente quelli di L1 ed L2.

Correttezza:

1) Scrivere la prova della correttezza di push_list
2) Scrivere la prova di correttezza di build_list
3) Scrivere un invariante per il ciclo di fuse.

E’ richiesta la consegna dei file code.h e code.cpp (opportunamente modificati) e quella del file es-24-5-2018.cpp che contiene tutto il resto.