# *Commento all'es. 1*

Si propongono due soluzioni, che differiscono per la parte di calcolo della distanza tra due nodi.

Dei due problemi affrontati, il primo, il conteggio dei nodi completi tra due livelli, viene affrontato mediante la funzione `contaCompleti`, la quale funge unicamente da wrapper per la funzione ricorsiva `contaCompletiR` (che riceve il parametro aggiuntivo `depth`). La funzione ha l'obiettivo di visitare l'albero sino al livello L2, conteggiando i nodi completi di livello maggiore o uguale a L1. Il conteggio viene gestito mediante il valore di ritorno (in alternativa si sarebbe potuto utilizzare un parametro per riferimento.

Per la distanza tra due nodi, data la loro chiave, si tratta di due sotto-problemi:

- ricerca dei nodi
- misura della distanza

Si propongono due soluzioni, che partono entrambe da una funzione wrapper `distanza`:

a) due chiamate distinte della funzione `contaR`, la quale cerca una delle due chiavi e, se trovata, ne ritorna la profondità (distanza dalla radice). La funzione, inoltre, sfrutta un campo aggiuntivo nel nodo dell'albero, per ricordare il precedente passaggio, e calcolare la lunghezza del tratto comune ai due cammini. In base a questo, la distanza tra i due nodi viene calcolata come somma dei due cammini (radice-nodo) a cui si sottrae il tratto comune (conteggiato due volte).

b) una chiamata alla funzione ricorsiva `distanzaR`, che ha come obiettivo trovare l'antenato comune, di profondità massima, dei due nodi cercati. Tale nodo può eventualmente essere uno dei due cercati. Una volta trovato l'antenato comune, da questo di parte per misurare il cammino destro e quello sinistro verso i due nodi, tramite la funzione `contaD`.

# *Commento all'es. 2*

La funzione `splitStringa` scandisce la stringa di ingresso sostituendo ad ogni occorrenza del separatore il carattere di finestringa `\0`. Mediante scansione della stringa si identificano una alla volte le sottostringhe, inserendole una alla volta in coda ad una lista di cui si mantengono il puntatore alla testa e alla coda (funzione `listInsTailFast` Puntatori e strutture dati dinamiche pag. 94). Nota la lunghezza della sottostringa corrente, tramite aritmetica dei puntatori si avanza nella stringa all'inizio della sottostringa successiva.

# *Commento all'es. 3*

Secondo la usuale filosofia, si definiscono un nuovo tipo per l'oggetto (`Item` che contiene le coordinate del punto e la sua distanza dall'origine) e le funzioni che vi operano in lettura, visualizzazione, estrazione della chiave (la distanza) e confronto tra chiavi. Si usa una funzione di calcolo della distanza dall'origine `distFromOrig`. La distanza è calcolata una sola volta quando viene letto il nuovo punto. Per la lista si definisce una struttura per il nodo e una struttura wrapper che contiene il puntatore alla testa (`l->head`). Vista la semplicità dei dati contenuti

nella struttura wrapper, se ne sarebbe potuto fare a meno. La si è introdotta per esemplificare quanto proposto nel testo "Puntatori e strutture dati dinamiche" cap. 5.4. Le funzioni di generazione di un nuovo nodo, di visualizzazione della lista e di inserzione in lista ordinata sono quelle standard, tenendo conto della struttura wrapper definita per la lista. Si introduce una funzione di `listInit` per allocare la struttura wrapper della lista e ritornarne il puntatore.