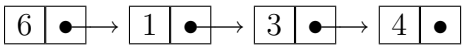
Programmazione II – Canale P-Z
Esame del 25/05/04

Schema delle soluzioni

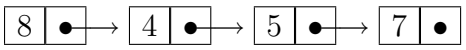
Esercizio 1. Si supponga di voler rappresentare numeri decimali di lunghezza qualsiasi mediante liste nel seguente modo: il primo nodo della lista contiene la cifra delle unità, il secondo quella delle decine, il terzo quella delle centinaia, etc.

Definire le strutture dati necessarie e scrivere una funzione che implementa la somma di due numeri rappresentati mediante liste.

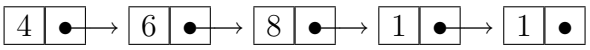
Schema della soluzione La struttura dati da utilizzare è una normale lista di interi. Tenendo conto che il primo elemento della lista contiene le unità, il secondo le decine, il terzo le centinaia, e così via, la seguente lista



rappresenta il numero 4316. La funzione che esegue l'addizione deve ricevere due liste con gli addendi e creare una nuova lista con il risultato. Ad esempio se alla precedente si vuole sommare la lista



il risultato è



Per implementare la somma occorre scandire simultaneamente le due liste sommando gli elementi corrispondenti e copiando il valore ottenuto in una nuova lista. Si osservi però che al momento della somma di due elementi occorre tener conto dell'eventuale riporto ottenuto dalla somma precedente.

Una soluzione molto semplice consiste nello scrivere una funzione ricorsiva che, oltre alle due liste da sommare, prende come terzo argomento il valore del riporto (un intero o addirittura semplicemente un booelano, visto che sommando due numeri con un riporto di al massimo 1, il riporto che si può ottenere è al massimo di 1) e ritorna la lista col risultato. La funzione somma i due elementi in testa alle liste degli addendi, gli aggiunge il riporto e memorizza il valore ottenuto (modulo 10) in un nuovo nodo p, quindi verifica se c'è riporto (se la somma superava 10) e richiama ricorsivamente la funzione sulla coda degli operandi e sul riporto ottenuto. Il risultato della chiamata ricorsiva deve essere attaccato al nodo p, cha a sua volta è il risultato della funzione.

Nella scrittura della funzione ricorsiva occorre fare attenzione ai casi in cui la lista di uno degli operandi è vuota, oppure al caso in cui c'è un riporto ma entrambe le liste degli operandi sono vuote (si veda la somma delle due ultime cifre dell'esempio). Si osservi infatti che la terminazione della somma si ha quando si sono analizzate tutte le cifre degli operandi e non si ha un riporto dall'ultima somma.

Esercizio 2. Si vogliono realizzare mediante puntatori delle liste circolari. Ovvero, delle liste tali che l'ultimo elemento della lista punta sempre al primo elemento della lista.

Definire le strutture dati che si intendono uilizzare per implementare le suddette liste e, supponendo di mantenere come puntatore di accesso alla lista il puntatore al primo elemento, fornire:

- la funzione che aggiunge un elemento in testa alla lista;
- la funzione che elimina un elemento dalla testa della lista (attenzione ai casi limite);
- la funzione cha conta gli elementi nella lista.

Discutere il costo delle suddette operazioni.

Facoltativo Cosa succede se, anziché il puntatore al primo elemento, per accedere alla lista si usa il puntatore all'ultimo elemento della lista? Riscrivere le precedenti funzioni.

Schema della soluzione Il tipo di dato è quello di una normale lista con puntatori, l'unica differenza è che il campo next dell'ultimo elemento della lista, anziché contenere il puntatore `NULL`, contiene il puntatore al primo elemento della lista. Nell'ipotesi in cui l'accesso alla lista avviene attraverso il primo elemento, anche per aggiungere un elemento in testa occorre scandire la lista alla ricerca dell'ultimo elemento. La scansione della lista richiede la ricerca del primo elemento il cui campo next contiene il puntatore al primo elemento della lista ed è quindi un'operazione lineare nella lunghezza della lista. Se p è il puntatore al primo elemento della lista e q all'ultimo,

- per aggiungere un elemento in testa, ci si posiziona su q e si inserisce l'elemento tra p e q come se si trattasse di un inserimento all'interno di una normale lista, l'elemento aggiunto è il nuovo primo elemento della lista;
- per eliminare un elemento in testa, ci si posiziona su q e si elimina p come se fosse un normale successore di p in una lista semplice (si osservi che il campo next di q contiene p), il valore del campo next di p è il nuovo primo elemento della lista;
- per contare gli elementi della lista, basta contare gli elementi scanditi nell'andare da p a q.

Le precedenti operazioni sono valide nel caso generale. I casi limite da prendere in considerazione sono: per quanto riguarda l'inserimento, il caso di lista vuota (si osservi che il caso di un solo elemento non è problematico); per quanto riguarda l'eliminazione, il caso di lista con un solo elemento (il tentativo di eliminazione da una lista vuota va trattato come un errore).

Siccome tutte le operazioni richiedono la scansione completa della lista (per posizionarsi sull'ultimo elemento q), tutte le operazioni sono lineari nella lunghezza della lista.

Se per accedere alla lista si usa il puntatore all'ultimo elemento, nel caso di inserimento e cancellazione non è più necessario scandire la lista, le due operazioni divengono quindi di costo costante. Contare il numero degli elementi rimane invece lineare (ovviamente, non si può evitare di scandire la lista per contare gli elementi).

Esercizio 3. Definire le strutture dati per l'implementazione mediante puntatori del tipo `BTREE` degli alberi binari etichettati con interi (ovvero, una variabile di tipo `BTREE` è un puntatore ad un nodo dell'albero) e il codice (incluse le eventuali funzioni ausiliarie) necessario all'implementazione della funzione `void appfoglie(BTREE t)` che, se `t` non è vuoto, apppende ad ogni foglia dell'albero i seguenti nodi:

- come figlio sinistro, un nodo contenente il minimo valore che appare nel cammino dalla radice alla foglia;
- come figlio destro, un nodo contenente il massimo valore che appare nel cammino dalla radice alla foglia.

Schema della soluzione Lo schema dell'algoritmo ricorsivo che risolve il problema è quello di una visita in preordine. Infatti, una possibile soluzione consiste nello scrivere una funzione ricorsiva `minmaxric` che riceve tre argomenti: un nodo v dell'albero, il minimo m e il massimo M del cammino dell'albero che conduce dalla radice a v escluso. Ad ogni chiamata della funzione su di un nodo diverso dalla radice

1. la funzione deve verificare se il contenuto di v è minore di m o maggiore di M e calcolare i nuovi valori m' ed M' del minimo e massimo del cammino che va dalla radice a v incluso;
2. quindi, se v è una foglia, aggiungere i nuovi nodi contenenti m' ed M', altrimenti, richiamare ricorsivamente `minmaxric` sui figli di v, passando negli altri due parametri m' ed M'.

Nel caso della radice, ovvero alla prima chiamata della funzione, il cammino che va dalla radice r ad r escluso è il cammino vuoto e pertanto i valori di m ed M non sono definiti. Ovviamente, non si può scegliere 0 perché si avrebbero problemi nel caso di un cammino con minimo positivo o massimo negativo. Una prima soluzione, consiste nello scrivere una funzione `minmax` che riceve come solo parametro la radice dell'albero (segnalando un errore in caso di albero vuoto) e, dopo aver inizializzato il minimo e il massimo al valore i contenuto nella radice r, prosegue come nel punto 2 sopra descritto, aggiungendo i nodi se r è una foglia, o chiamando `minmaxric` sui figli di r, altrimenti. È facile vedere che lo stesso risultato si può ottenere anche semplicemente richiamando in `minmax` la `minmaxric` su r passando i negli altri due parametri (ovvero, assumendo che per il cammino che arriva alla radice $m = M = i$).

Esercizio 4. Scrivere una funzione che, preso come input un albero binario etichettato con interi, rimpiazza il contenuto di ciascun nodo v con il numero di nodi che si trovano nel sottoalbero di radice v. (Fornire anche la definizione delle strutture dati utilizzate per la memorizzazione degli alberi.)
Attenzione La migliore soluzione è lineare nel numero dei nodi dell'albero.

Schema della soluzione Lo schema della soluzione ricorsiva (più efficiente) di questo esercizio è quello di una visita in postordine. Occorre scrivere una funzione `rimpnumnodi` che riceve come parametro un nodo v e, dopo aver sostituito i valori dei nodi nell'albero di radice v in base a quanto specificato nell'esercizio, ritorna il numero dei nodi nell'albero di radice v.

Per scrivere `rimpnumnodi` si osservi che

- se v è vuoto, è sufficiente ritornare il valore 0;
- se v non è vuoto, il valore n dei nodi nell'albero è pari alla somma del risultato di `rimpnumnodi` sul figlio sinistro e sul figlio destro di v incrementata di 1; quindi, `rimpnumnodi` deve semplicemente scrivere n in v e ritornarne il valore.

Questa soluzione è chiaramente lineare, essendo una semplice visita in postordine. Si osservi invece che, una soluzione basata su di una funzione `numnodi` che calcola il numero dei nodi in un albero senza eseguire rimpiazzamenti e su di una visita che ad ogni nodo richiama `numnodi`, è quadratica.