Esercizi

3. Liste lineari

Esercizio 3.1
Nella strategia di inserimento in una lista ordinata presentata a lezione, se vi sono già nodi con la stessa chiave di quello che deve essere inserito, il nuovo nodo viene inserito prima di essi. Studiate come modificare la strategia in modo che l'inserimento avvenga dopo di essi.

Esercizio 3.2
Nella strategia di cancellazione in una lista ordinata presentata nello pseudocodice allegato alla lezione 13, viene cancellato il primo nodo con chiave uguale a quella specificata. Scrivete due varianti che effettuino, rispettivamente, la cancellazione di tutti i nodi con chiave uguale a quella specificata e la cancellazione dell'ultimo nodo con chiave uguale a quella specificata.

Esercizio 3.3
In questo esercizio prendiamo in esame il problema di ordinare in maniera non decrescente una lista. Considerate i tre algoritmi di ordinamento elementari che abbiamo studiato per ordinare array (*selectionSort*, *insertionSort*, *bubbleSort*). Per ognuno di essi studiate se vi sia un modo naturale di adattarlo all'ordinamento *in loco* di una lista (cioè senza ricorrere a strutture ausiliare), in modo che il numero di confronti e il tempo utilizzato siano gli stessi degli algoritmi visti sugli array.

- Se ritenete che l'algoritmo non possa essere adattato, spiegatene le ragioni evidenziando le difficoltà che si incontrano.
- Se ritenete che l'algoritmo possa essere adattato, spiegate quali cambiamenti sia utile effettuare. Scrivete poi lo pseudocodice dell'algoritmo modificato.

Esercizio 3.4
Ripetete l'esercizio 3.3 per l'algoritmo *quickSort*. In questo caso potete utilizzare delle liste ausiliare in cui spostare temporaneamente i nodi della lista originaria (ma senza creare nodi aggiuntivi – è opportuno far ricorso ad alcuni puntatori ausiliari).

Esercizio 3.5
Scrivete un algoritmo che date due liste ordinate in maniera non descrescente ne effettui il merge, costruendo una nuova lista ordinata in maniera non decrescente, costituita dai nodi presenti nelle due liste originarie.
Nota. Non si richiede di copiare gli elementi delle due liste, ma di *spostarli* nella nuova lista. Pertanto alla fine le due liste date dovranno essere vuote.

Esercizio 3.6
Ripetete l'esercizio 3.4 per l'algoritmo *mergeSort*. Ricordatevi che l'algoritmo *mergeSort* utilizza un array ausiliario per il merge, che utilizza spazio proporzionale al numero di elementi. Riuscite ad evitare questo ulteriore utilizzo di spazio?

Data pubblicazione: 27 ottobre 2021