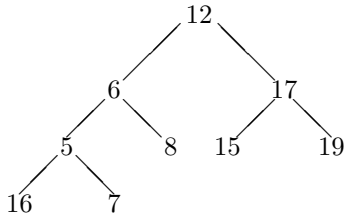
Corso di Algoritmi e Strutture Dati
Anno Accademico 2019–2020
Prof. Giovanni Pighizzini

Esercizi

4. Algoritmi di ordinamento, heap e di nuovo alberi

Esercizio 4.1
Considerate l'albero binario quasi completo nella figura a destra. Applicate `heapSort` all'albero: prima trasformate l'albero in uno heap, partendo dalle foglie, e disegnate la struttura ottenuta; successivamente, per ogni iterazione dell'algoritmo, disegnate lo heap che si ottiene dopo avere rimosso la foglia più a destra nell'ultimo livello, collocato il suo valore nella radice, e risistemato lo heap.[1]



Esercizio 4.2
Considerate il vettore posizionale contenente i seguenti elementi `12 14 20 25 13 10 22 19 24 27 15 16 21`. Disegnatelo come albero binario e applicate `heapSort`, disegnando l'evoluzione dello heap, come per l'esercizio 4.1.

Esercizio 4.3
Considerate il vettore posizionale contenente i seguenti elementi `20 11 18 16 17 3 30 4 19 25`. Applicate `heapSort` *in loco*, scrivendo il contenuto dell'array dopo averlo trasformato in uno heap, e dopo ogni iterazione dell'algoritmo.

Esercizio 4.4
Adattando la strategia bottom-up utilizzata da `heapSort` per trasformare un albero binario quasi completo in un *Max-heap*, trasformate l'albero nella figura dell'esercizio 4.1 in un *Min-heap*.

Esercizio 4.5
Risolvete gli esercizi 4.2 e 4.3, ordinando gli array dati in modo decrescente, anziché crescente.

Esercizio 4.6
Considerate la seguente sequenza di numeri

`2222 4512 27 890 345 1234 9876 1236 2345 231 234 543`

memorizzata in un array.

(a) Supponete di ordinare la sequenza mediante l'algoritmo di ordinamento per selezione. Indicate la sequenza ottenuta al termine di ciascuna iterazione del ciclo principale dell'algoritmo.

(b) Supponete di ordinare la sequenza mediante l'algoritmo di ordinamento per inserimento. Indicate la sequenza ottenuta al termine di ciascuna iterazione del ciclo principale dell'algoritmo.

[1] In assenza di altre indicazioni, con il termine *heap* intendiamo un *Max-heap*.

Data pubblicazione: 22 novembre 2019

(c) Supponete di ordinare la sequenza mediante l'algoritmo `bubbleSort`. Indicate la sequenza ottenuta al termine di ciascuna iterazione del ciclo principale dell'algoritmo.

(d) Supponete di ordinare la sequenza mediante l'algoritmo `radixSort` (base 10). Indicate la sequenza ottenuta al termine di ciascuna iterazione del ciclo principale dell'algoritmo.

(e) Supponete di ordinare la sequenza mediante l'algoritmo `heapSort`. Indicate lo heap che viene ottenuto a partire dalla sequenza, e la sequenza ottenuta al termine di ciascuna iterazione del ciclo principale dell'algoritmo.

(f) Supponete di ordinare la sequenza mediante l'algoritmo `quickSort`, scegliendo come perno il primo elemento. Indicate il contenuto dell'array dopo avere effettuato la partizione, prima delle chiamate ricorsive di `quickSort`.

(g) Supponete di ordinare la sequenza mediante l'algoritmo `mergeSort`. Indicate come viene scomposto e ricomposto l'array ad ogni livello di ricorsione.

(h) Costruite un albero binario di ricerca inserendo, in un albero inizialmente vuoto, gli elementi della sequenza nell'ordine indicato, utilizzando la versione base dell'inserimento (senza nessun bilanciamento). Indicate poi le sequenze che si ottengono visitando l'albero in ordine anticipato, simmetrico, posticipato.

(i) Ripetete il punto precedente, costruendo però un albero AVL. Indicate poi le sequenze che si ottengono visitando l'albero in ordine anticipato, simmetrico, posticipato.

(j) Ripetete il punto precedente, costruendo però un albero 2-3.

Esercizio 4.7
Ripetete l'esercizio 4.6, considerando la sequenza di numeri

```
4567 8776 2324 3434 9876 123 876 1232 234 975 654 444
```

Esercizio 4.8
Considerate un albero binario quasi completo contenente n nodi in cui, nel livello massimo, tutte le foglie si trovano il più a sinistra possibile. Calcolate il valore esatto dell'altezza dell'albero in funzione di n e, per ciascun livello, il numero di nodi.

Supponete di rappresentare l'albero con un vettore posizionale, come quello utilizzato per `heapSort`, memorizzando la radice nella posizione 0. Per ogni profondità p dei nodi presenti nell'albero esprimente in funzione di p l'intervallo di indici dell'array che corrispondono ai nodi di profondità p (si può osservare che profondità 0 corrisponde all'indice 0, profondità 1 agli indici $1, 2$, profondità 2 agli indici $3, \ldots, 6$, cercate di trovare una formula generale e verificatela poi per induzione). Dimostrate che i figli del nodo interno memorizzato in posizione di indice i, sono memorizzati nelle posizioni di indici $2i + 1$ e $2i + 2$. Ricavate infine una formula per ottenere, dall'indice di un nodo diverso dalla radice, l'indice del nodo padre.

Esercizio 4.9
Ripetete la seconda parte dell'esercizio 4.8, supponendo di utilizzare indici di array a partire da 1 e, dunque, di memorizzare la radice in posizione 1.

Esercizio 4.10
Fornite un esempio che mostri che `heapSort` non è un algoritmo di ordinamento *stabile*.

Esercizio 4.11
Scrivete una versione modificata di `radixSort` per trattare anche numeri negativi.

Esercizio 4.12
Scrivete una versione di `radixSort` per ordinare stringhe di lettere sull'alfabeto $\{\texttt{a}, \ldots, \texttt{z}\}$. Fate attenzione al fatto che le stringhe potrebbero avere lunghezza differente.

Esercizio 4.13
Ripetete l'esercizio 4.6, considerando la sequenza contenente le parole

```
gatto topo papera cane formica oca pantera falco lupo zebra ape vespa
```

e l'ordinamento lessicografico.

Esercizio 4.14
Ripetete l'esercizio 4.6, considerando la sequenza contenente le parole

```
giallo rosso verde viola blu nero bianco arancio rosa lilla
```

e l'ordinamento lessicografico.

Esercizio 4.15
Considerate il vettore posizionale contenente i seguenti elementi `16 15 22 17 44 36 8 11 14 24 33 37`. Disegnatelo come albero binario e trasformatelo in uno heap, applicando la strategia utilizzata dalla prima parte di `heapSort`. Applicate poi le seguenti operazioni, nell'ordine indicato, disegnando il contenuto dello heap dopo ciascuna di esse:

- cancellazione di `14`,
- cancellazione di `37`,
- inserimento di `43`,
- sostituzione di `36` con `7`,
- sostituzione di `43` con `4`,
- cancellazione di `15`,
- inserimento di `50`.

Esercizio 4.16
Ripetere l'esercizio precedente costruendo un *Min-heap* anziché un *Max-heap*.