

Algoritmi e strutture dati 20/21: esercizi per la preparazione all'esame

- 1) Studiare la complessità di *InsertionSort* nel caso medio.
 Cosa significa questa assunzione, in termini di numero di passi effettuati dal ciclo più interno?
- A) $\Theta(n)$. Ci si aspetta che il ciclo più interno effettui circa 5 passi.
 B) $\Theta(n^2)$. Ci si aspetta che il ciclo più interno effettui circa la metà dei passi possibili.
 C) $\Theta(n^2)$. Ci si aspetta che il ciclo più interno effettui circa n^2 passi.
 D) $\Theta(\lg(n))$. Non si può prevedere il numero di passi del ciclo più interno.
- 2) Diciamo che A è un array di interi **quasi ordinato** se ogni elemento si trova, al massimo, a k posizioni di distanza dalla sua posizione corretta (quella che assumerebbe nell'array ordinato). Qual è la complessità di *InsertionSort* quando A è quasi ordinato e k è una costante?
- A) $\Theta(n)$.
 B) $\Theta(k)$.
 C) $\Theta(n^2)$.
 D) $\Theta(k \cdot \lg(k))$.
- 3) Qual è la complessità di *InsertionSort* quando A è quasi ordinato e k è $\frac{n}{2}$?
- A) $\Theta(n)$.
 B) $\Theta(k)$.
 C) $\Theta(n^2)$.
 D) $\Theta(k \cdot \lg(k))$.
- 4) Diciamo che un algoritmo di ordinamento è **stabile** se e solo se, a fine esecuzione, gli elementi con lo stesso valore non hanno cambiato posizione relativa, ed che è **in place** se e solo se la quantità di memoria utilizzata esternamente all'array di partenza è costante. *InsertionSort* è:
- A) Stabile, ma non in place.
 B) In place, ma non stabile.
 C) Stabile ed in place.
 D) Né stabile, né in place.
- 5) Quali sono, tra quelle elencate, proprietà che la relazione $O()$ soddisfa?
- A) È simmetrica.
 B) È transitiva.
 C) È transitiva e riflessiva.
 D) È transitiva, riflessiva, e simmetrica.
- 6) Quali sono, tra quelle elencate, proprietà che la relazione $\Theta()$ soddisfa?
- A) È simmetrica.
- B) È transitiva.
- C) È transitiva e riflessiva.
- D) È transitiva, riflessiva, e simmetrica.
- 7) Quali sono, tra quelle elencate, proprietà che la relazione $\Theta()$ soddisfa?
- A) È simmetrica.
 B) È transitiva.
 C) È transitiva e riflessiva.
 D) È transitiva, riflessiva, e simmetrica.
- 8) Nel mostrare $3 \cdot n^2 + 5 \cdot n + 1 = \Theta(n^2)$ troviamo:
- A) $n > 1$, $c_1 = \frac{5}{4}$, e $c_2 = \frac{21}{7}$.
 B) $n > 2$, $c_1 = \frac{3}{5}$, e $c_2 = \frac{21}{5}$.
 C) $n > 4$, $c_1 = \frac{3}{4}$, e $c_2 = \frac{21}{4}$.
 D) $n > 3$, $c_1 = \frac{3}{5}$, e $c_2 = \frac{21}{5}$.
- 9) È vero che $6 \cdot n^3 = \Theta(n^2)$?
- A) No. Infatti, per contraddizione, si assume $6 \cdot n^3 = \Theta(n^2)$. Tra le altre cose questo implica che per qualche n_0, c_2 si ha che per ogni $n \geq n_0$ è il caso che $6 \cdot n^3 \leq c_2 \cdot n^2$. Questo significa che $6 \cdot n \leq c_2$, il che è contraddice il fatto che c_2 è una costante.
 B) No. Infatti, per contraddizione, si assume $6 \cdot n^3 = \Theta(n^2)$. Tra le altre cose questo implica che per qualche n_0, c_1 si ha che per ogni $n \geq n_0$ è il caso che $6 \cdot n^3 \geq c_1 \cdot n^2$. Questo significa che $6 \cdot n \geq c_1$, il che è contraddice il fatto che c_1 è una costante.
 C) Sì. Infatti, $6 \cdot n^3$ si comporta asintoticamente come n^2 .
 D) Sì.
- 10) Sappiamo che $3 \cdot n^2 + 2 = O(n^2)$. Per quali costanti n_0, c vale che $3 \cdot n^2 + 2 \leq c \cdot n^2$ per ogni $n > n_0$?
- A) $3 \cdot n^2 + 2 = O(n^2)$ è falso.
 B) $n_0 = 10, c = 4$.
 C) $n_0 = 1, c = 1$.
 D) È vero per tutti gli $n \geq 0$ e tutte le costanti c .
- 11) È vero che $5 \cdot n + 6 = O(n^2)$?
- A) No.
 B) Sì, Per $c = 1$ e per tutti gli $n > 0$.
 C) Sì. Per $c = 1$ e per tutti gli $n > 2$.
 D) Sì. Per $c = 1$ e per tutti gli $n \geq 6$.
- 12) Nella notazione $O(), \Omega(), \Theta()$:
- A) La base dei logaritmi è importante in tutti i casi, ad esempio $\Theta(\lg_2(n)) \neq \Theta(\lg_3(n))$.
 B) La base dei logaritmi è importante ma solo nella notazione $O()$, ad esempio $O(\lg_2(n)) \neq O(\lg_3(n))$.

```

proc Parity (A, Key)
{return RecursiveParity(A, Key, 1, A.length)}

proc RecParity (Key, A, i, j)
if (i = j)
then
{if (A[i] = Key)
then return false
return true
k = ⌊(i+j) / 2⌋
r = 0
if (RecParity(A, Key, i, k) = RecParity(A, Key, k + 1, j))
then return true
return false

```

Figure 1. Codice per l'esercizio 14.

- C) La base dei logaritmi non è importante, infatti $\Theta(\lg_a(n)) = \Theta(\lg_b(n))$ per qualsiasi $a \neq b$, con $a, b > 1$.
- D) Nessuna delle altre affermazioni è corretta.
- 13) Si trovi l'andamento asintotico della funzione $\lg(n!)$. Succede che:
- A) $\lg(n!) = O(n)$.
- B) $\lg(n!) = O(\sqrt{n})$.
- C) $\lg(n!) = \Theta(n \cdot \lg(n))$.
- D) Nessuna è corretta.
- 14) Si consideri un array che contiene chiavi qualsiasi. Si consideri il problema di stabilire se il numero di occorrenze di una certa chiave *Key* è o no pari (0 si considera pari), ed il codice in Fig. 1. È vero che:
- A) La procedura è incorretta.
- B) La procedura è corretta e ha costo logaritmico nella dimensione di *A*.
- C) La procedura è corretta e ha costo lineare nella dimensione di *A*.
- D) La procedura è incorretta, ma si può correggere cambiando l'ultima istruzione ed ottenere una procedura corretta di costo logaritmico nella dimensione dell'array.
- 15) Dato un array *A* di interi non ordinato, vogliamo trovare il massimo ed il minimo di *A*. Il problema ha costo asintotico lineare cioè $O(n)$, perchè l'algoritmo naïve che lo risolve (trova il minimo con una passata su *A*, poi trova il massimo con una seconda passata su *A*) ha costo $O(2 \cdot n) = O(n)$. Tralasciando però in questo esercizio la notazione asintotica, è possibile trovare un algoritmo che risolva questo problema con un numero di confronti (non operazioni qualsiasi) strettamente inferiore a $2 \cdot n$?
- A) Si. È possibile trovare un algoritmo che risolve il problema usando, al più, $\frac{4}{5} \cdot n$ confronti.
- B) Si. È possibile trovare un algoritmo che risolve il problema usando, al più, $\frac{1}{2} \cdot n$ confronti.
- C) Si. È possibile trovare un algoritmo che risolve il problema usando, al più, $\frac{3}{4} \cdot n$ confronti.
- D) Si. È possibile trovare un algoritmo che risolve il problema usando, al più, $\frac{3}{2} \cdot n$ confronti.

- 16) Considerata una matrice quadrata di interi *A*, di lato *n*, tale che ogni riga è ordinata da sinistra a destra, ed ogni colonna è ordinata dall'alto verso il basso. È possibile mostrare che il problema della ricerca di un certo intero *k* in *A* ha complessità, in tempo:

- A) $\Omega(n \cdot \lg(n))$.
- B) $\Omega(n^2)$.
- C) $\Theta(n \cdot \lg(n))$.
- D) $O(n)$.

- 17) Quale delle seguenti è una affermazione vera?

- A) Le notazioni $\Theta()$, $O()$, $\Omega()$ sono transitive.
- B) La notazione $O()$ è simmetrica.
- C) Se $f(n) = O(g(n))$ allora $g(n) = \Omega(f(n))$, ma non è vero il contrario.
- D) Sono tutte vere.

- 18) Si consideri la ricorrenza:

$$T(n) = 8 \cdot T\left(\frac{n}{2}\right) + n^2.$$

Usando lo sviluppo, si ottiene:

- A) $T(n) = \Theta(n^3)$.
- B) $T(n) = \Theta(n^2)$.
- C) $T(n) = \Theta(n \cdot \lg(n))$.
- D) $T(n) = \Theta(n)$.

- 19) Si consideri la ricorrenza:

$$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + 1.$$

Usando lo sviluppo, si ottiene:

- A) $T(n) = \Theta(n^2)$.
- B) $T(n) = \Theta(\lg(n))$.
- C) $T(n) = \Theta(n)$.
- D) Nessuna è corretta.

- 20) Si consideri la ricorrenza:

$$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + n^3.$$

Usando lo sviluppo, si ottiene:

- A) $T(n) = \Theta(n^{\frac{3}{4}})$.
- B) $T(n) = \Theta(n^2)$.
- C) $T(n) = \Theta(n^3)$.
- D) $T(n) = \Theta(n^2 \cdot \lg(n))$.

- 21) Si consideri la ricorrenza

$$T(n) = 9 \cdot T\left(\frac{n}{3}\right) + n.$$

Usando lo sviluppo, si ottiene:

- A) $T(n) = \Theta(n)$.
- B) $T(n) = \Theta(n^2 \cdot \lg(n))$.
- C) $T(n) = \Theta(n)$.
- D) $T(n) = \Theta(n^2)$.

- 22) Si consideri la ricorrenza

$$T(n) = T\left(\frac{2 \cdot n}{3}\right) + 1.$$

È vero che:

- A) Possiamo usare il caso 2 del Master Theorem.
- B) Possiamo usare il caso 1 del Master Theorem.
- C) Possiamo usare il caso 3 del Master Theorem.
- D) Potremmo usare il caso 3 del Master Theorem, ma la seconda condizione non è rispettata.

23) Si consideri la ricorrenza:

$$T(n) = 5 \cdot T\left(\frac{n}{3}\right) + n^2.$$

È vero che:

- A) $T(n) = \Theta(n \cdot \lg^2(n))$.
- B) $T(n) = \Theta(n \cdot \lg(n))$.
- C) $T(n) = O(n^2)$.
- D) $T(n) = O(n)$.

$$\begin{aligned} p_n &\xrightarrow{5 \rightarrow 3^k=5 \rightarrow 1.465} \\ f(n) &= n^2 \\ \Omega(f(n)) &= \Theta(n^2) \quad \epsilon \approx 0.5 \end{aligned}$$

24) Si consideri la ricorrenza:

$$T(n) = 10 \cdot T\left(\frac{n}{3}\right) + 3 \cdot n^2 + n.$$

La sua soluzione è:

- A) $\Theta(n^2)$.
- B) $\Theta(n^{\lg_3(10)})$.
- C) $\Theta(n \cdot \lg(n))$.
- D) Nessuna delle soluzioni proposte è corretta.

25) Per quale tra le seguenti ricorrenze il Master Theorem non è applicabile?

- A) $T(n) = 9 \cdot T\left(\frac{n}{3}\right) + 3 \cdot n$.
- B) $T(n) = T\left(\frac{2 \cdot n}{3}\right) + 43$.
- C) $T(n) = 10 \cdot T\left(\frac{n}{3}\right) + 3 \cdot n^2 + 3 \cdot n$.
- D) Tutte queste ricorrenze possono essere risolte con il Master Theorem.

26) Si consideri la seguente dimostrazione per sostituzione della ricorrenza

$$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + 1.$$

Vogliamo mostrare che $T(n) \leq c \cdot \lg(n)$ per qualche c :

$$\begin{aligned} T(n) &\leq 2 \cdot c \cdot \lg\left(\frac{n}{2}\right) + 1 \\ &\leq 2 \cdot c \cdot \lg(n) - 2 \cdot c + 1 \\ &\leq c \cdot \lg(n) \end{aligned}$$

- A) La dimostrazione è corretta.
- B) La dimostrazione è sbagliata; l'errore si trova all'ultimo passaggio.
- C) La dimostrazione è corretta, e $T(n) = O(\lg(n))$.
- D) La dimostrazione è sbagliata, ma $T(n) = O(\lg(n))$.

27) Si consideri la seguente dimostrazione per sostituzione della ricorrenza

$$T(n) = T\left(\lfloor \frac{n}{2} \rfloor\right) + T\left(\lceil \frac{n}{2} \rceil\right) + 1.$$

Vogliamo mostrare che $T(n) \leq c \cdot n - d$ per qualche c, d :

$$\begin{aligned} T(n) &\leq (c \cdot \lfloor \frac{n}{2} \rfloor - d) + (c \cdot \lceil \frac{n}{2} \rceil - d) + 1 \\ &\leq c \cdot n - 2 \cdot d + 1 \\ &\leq c \cdot n - d \end{aligned} \quad \text{se } d \geq 1$$

- A) La dimostrazione è incorretta perché $T(n) = o(n)$.
- B) La dimostrazione è corretta e se ne deduce che $T(n) = O(n^2)$.
- C) La dimostrazione è incorretta, ma è vero che $T(n) = O(n)$.
- D) La dimostrazione è corretta e se ne deduce che $T(n) = O(n)$.

28) Nel testo usato a lezione (il 'Cormen') appare un errore nell'esercizio 4.3-8. Si chiede di considerare la ricorrenza

$$T(n) = 4 \cdot T\left(\frac{n}{2}\right) + n^2$$

e di mostrare che il metodo della sostituzione *fallisce* cercando di dimostrare che $T(n) \leq c \cdot n^2$. Qual è l'errore nell'esercizio?

- A) Non c'è errore: la sostituzione non funziona con $T(n) \leq c \cdot n^2$ ma funziona con $T(n) \leq c \cdot n^2 - d$.
- B) Con il Master Theorem vediamo che, diversamente da quanto indicato, $T(n) = \Theta(n^2 \cdot \lg(n))$; possiamo mostrare che, mentre assumere $T(n) \leq c \cdot n^2 \cdot \lg(n)$ non funziona, la sostituzione funziona con $T(n) \leq c \cdot n^2 \cdot \lg(n) - n$.
- C) Con il Master Theorem vediamo che, diversamente da quanto indicato, $T(n) = \Theta(n \cdot \lg(n))$; possiamo mostrare che, mentre assumere $T(n) \leq c \cdot n \cdot \lg(n)$ non funziona, la sostituzione funziona con $T(n) \leq c \cdot n \cdot \lg(n) - n$.
- D) Nessuna delle altre è corretta.

29) Qual è la complessità di *SelectionSort*?

- A) $\Theta(n^2)$ in tutti i casi.
- B) $\Theta(n^2)$ nel caso peggiore, $\Theta(n)$ nel caso migliore.
- C) $\Theta(n \cdot \lg(n))$ in tutti i casi.
- D) $\Theta(n^2)$ nel caso peggiore, $\Theta(n)$ nel caso medio.

30) Qual è una invariante del ciclo più esterno di *SelectionSort*?

- A) Dopo la j -esima esecuzione del ciclo più esterno, l'array $A[j+1, \dots, n]$ contiene gli $n-j$ elementi più piccoli di A ed è ordinato in maniera non decrescente.
- B) Dopo la j -esima esecuzione del ciclo più esterno, l'array $A[1, \dots, j]$ contiene gli j elementi più piccoli di A ed è ordinato in maniera non crescente.
- C) Dopo la j -esima esecuzione del ciclo più esterno, l'array $A[1, \dots, j]$ contiene gli j elementi più piccoli di A ed è ordinato in maniera non decrescente.
- D) Dopo la j -esima esecuzione del ciclo più esterno, l'array $A[j, \dots, n-1]$ contiene gli $n-j$ elementi più piccoli di A ed è ordinato in maniera non decrescente.

31) Perchè l'invariante del ciclo più esterno di *SelectionSort* vista nell'esercizio precedente, effettivamente dimostra la correttezza dell'algoritmo?

- A) Perchè l'algoritmo è elementare.
- B) In realtà, l'algoritmo non è corretto.
- C) L'invariante è vera, tra le altre cose, dopo l'ultima esecuzione. In questo caso, dice che l'array $A[j, \dots, n-1]$ contiene gli $n-j$ elementi più piccoli di A ed è

- ordinato in maniera crescente; ma questo implica che, in realtà, tutto A è ordinato, cioè che l'algoritmo è corretto.
- D) L'invariante è vera, tra le altre cose, dopo l'ultima esecuzione. In questo caso, dice che l'array $A[1, \dots, n-1]$ contiene gli $n - 1$ elementi più piccoli di A ed è ordinato in maniera non decrescente; ma questo implica che, in realtà, tutto A è ordinato, cioè che l'algoritmo è corretto.
- 32) *SelectionSort* è:
- A) Stabile.
 - B) In place, ma non necessariamente stabile.
 - C) Stabile, ma non necessariamente in place.
 - D) Stabile e in place.
- 33) Si consideri un array ordinato circolarmente con k spiazzamenti, cioè un array tale che, se tutti gli elementi fossero spostati di esattamente k posizioni, sarebbe ordinato (per esempio, $A = [5, 7, 10, 12, 30, 3, 4]$ è ordinato circolarmente con 2 spiazzamenti). Il problema di trovare il minimo elemento in un array di questo tipo ha complessità, nel caso peggiore:
- A) $O(\lg(n))$.
 - B) $\Theta(n)$.
 - C) $\Theta(n \cdot \lg(n))$.
 - D) $\Theta(n^2)$.
- 34) Chi, tra *MergeSort* e *InsertionSort*, si comporterebbe meglio sul problema di ordinare un array di n posizioni quasi ordinato, con k costante?
- A) *MergeSort*: chiaramente $\Theta(n \cdot \lg(n))$ è sempre meglio di $\Theta(n^2)$.
 - B) *MergeSort*: chiaramente $\Theta(n \cdot \lg(n))$ è sempre meglio di $\Theta(k \cdot n)$.
 - C) *InsertionSort*: il suo tempo di esecuzione nel caso peggiore è $\Theta(k \cdot n)$, e quando k è costante questo diventa $\Theta(n)$.
 - D) È impossibile stabilirlo, se non sperimentalmente.
- 35) In una versione alternativa di *MergeSort* facciamo tre chiamate ricorsive a tre sotto-array e poi utilizziamo una versione di *Merge* che è capace di riordinare in un solo array tre, invece di due, sotto-array già ordinati. Come si comporta, asintoticamente, questa nuova versione rispetto all'originale?
- A) In maniera identica.
 - B) La nuova versione, nel caso peggiore, ha complessità $T(n) = \Theta(n \cdot \lg^3(n))$, pertanto si comporta in modo peggiore.
 - C) La nuova versione, nel caso peggiore, ha complessità $T(n) = \Theta(n \cdot \lg_3(n))$, pertanto si comporta in modo migliore.
 - D) È impossibile stabilirlo, se non sperimentalmente.
- 36) Nella nostra implementazione, *MergeSort* è un algoritmo di ordinamento stabile?
- A) No, in quanto non è in place.
 - B) Non è possibile stabilirlo.
- C) Sí.
D) No.
- 37) Sia A un array di interi, e chiamiamo **inversione** una coppia di indici $i < j$ tale che $A[i] > A[j]$. Il problema di calcolare il numero di inversioni di A :
- A) È irrisolvibile.
 - B) Ha complessità $\Omega(n^2)$.
 - C) Ha complessità $\Theta(n^2)$
 - D) Ha complessità $O(n \cdot \lg(n))$.
- 38) È possibile utilizzare una strategia simile a quella di *MergeSort* per trovare il massimo elemento di un array A non ordinato?
- A) No, il problema è risolvibile unicamente utilizzando un approccio iterativo.
 - B) Si. L'algoritmo risultante ha complessità $\Theta(\lg(n))$, risultando migliore di quella dell'approccio standard.
 - C) Si. L'algoritmo risultante ha complessità $\Theta(n)$, risultando uguale a quella dell'approccio standard.
 - D) Si. L'algoritmo risultante ha complessità $\Theta(n^2)$, risultando peggiore di quella dell'approccio standard.
- 39) Applicare *Partition*($A, 1, 5$) all'array $A = [7, 1, 4, 5, 3]$. Qual è il valore restituito (cioè *indice del pivot*)?
- A) 2.
 - B) 1.
 - C) 3.
 - D) Dipende dall'esecuzione.
- 40) La scelta di $A[r]$ come pivot in *Partition* è completamente arbitraria. Cambiarla con qualsiasi altro elemento di A genera un'altra versione di *QuickSort* che ha la stessa complessità e lo stesso difetto: esiste una istanza di input che certamente provoca il peggior comportamento computazionale.
- A) Vero.
 - B) Falso.
 - C) Dipende dal fatto che A non è già ordinato in partenza.
 - D) Non è possibile stabilirlo.
- 41) Sotto quali condizioni vale l'ultimo passaggio nella dimostrazione relativa al calcolo della complessità del caso medio di *RandomizedQuickSort*?
- A) Sempre.
 - B) Quando a è abbastanza grande, cioè quando $\frac{a \cdot n}{4}$ è asintoticamente più grande di $b + \Theta(n)$.
 - C) Quando a è abbastanza piccolo, cioè quando $\frac{a \cdot n}{4}$ è asintoticamente più piccolo di $b + \Theta(n)$.
 - D) Esiste una condizione, ma non è nessuna di quelle dette precedentemente.
- 42) Si consideri la nostra implementazione di *Partition*, ricordando che i valori p ed r sono, rispettivamente, gli indici del primo e dell'ultimo elemento della parte dell'array A in esame. Nel caso in cui A contenga tutti elementi uguali, che valore restituisce *Partition*?
- A) x .
 - B) $r + 1$.

- C) r .
D) p .
- 43) Dato un array A di n posizioni, ognuna delle quali contiene solamente 0 oppure 1, qual è la complessità del problema di ottenere l'array modificato in modo che tutti gli 0 appaiano prima di tutti gli 1 senza utilizzare nessuna struttura dati aggiuntiva?
- A) $O(\lg(n))$.
B) $\Theta(n^2)$.
C) $\Theta(n \cdot \lg(n))$.
D) $O(n)$.
- 44) Si consideri *RandomizedQuickSort*, e si consideri una ricorrenza che esprima il numero di chiamate alla funzione *Random* che vengono effettuate nel peggior caso. Quale è, tra queste?
- A) $T(n) = T(n - 1) + 1$, cioè $\Theta(n^2)$.
B) $T(n) = T(n - 1) + 1$, cioè $O(n)$.
C) È impossibile stabilirlo perché si tratta di una chiamata casuale.
D) $O(\lg(n))$.
- 45) Si consideri il problema di trovare il k -esimo elemento più piccolo di un array A di n posizioni, senza ordinarlo. Usando *Partition*, possiamo trovare un algoritmo di complessità:
- A) $\Theta(n^2)$ nel caso migliore.
B) $\Theta(n \cdot \lg(n))$.
C) $O(n \cdot \lg(n))$ nel caso peggiore.
D) $O(n)$ nel caso medio.
- 46) Si consideri il problema di ordinare istanze di numeri interi positivi diversi da zero. Si danno le seguenti ipotesi: ogni istanza è lunga al massimo 100, le chiavi sono totalmente casuali, ed ogni elemento di ogni istanza è un intero inferiore o uguale a 10^6 . Tra le seguenti scelte, qual è la più efficiente, in media?
- A) Le ipotesi permettono di utilizzare efficientemente *CountingSort*: in media ogni ordinamento prenderà tempo dell'ordine di 100 unità.
B) È conveniente usare *QuickSort*: l'ipotesi di istanze casuali ci permette di dire che in media ogni ordinamento prenderà tempo dell'ordine di 700 unità.
C) Le ipotesi permettono di dedurre che ogni istanza sarà quasi ordinata: utilizzando *InsertionSort*, impiegheremo, per ogni istanza, un tempo dell'ordine di 100 più il numero di coppie nell'ordine inverso.
D) La B) e la A) sono entrambe corrette, ma la scelta A) è più efficiente.
- 47) Si consideri un array A di n posizioni, in ognuna delle quali è presente uno tra i valori 1, 0 e -1 . È possibile ordinare A in tempo lineare rispetto a n ?
- A) Nessuna delle altre è corretta.
B) No. Dobbiamo ordinarlo con un metodo basato sui confronti e pertanto il limite minimo è $O(n \cdot \lg(n))$.
C) Sì. Possiamo ordinarlo con un metodo basato sui confronti e pertanto il limite minimo è $O(3 \cdot \lg(3))$ (ci sono solo tre elementi diversi); $O(3 \cdot \lg(3))$ è $O(1)$ e quindi è lineare in n .
- D) Si. Possiamo usare *CountingSort* come visto in classe, pre-processando A : sommiamo 1 a tutti gli elementi, eseguiamo *CountingSort*, e poi sottraiamo 1 a tutti gli elementi.
- 48) Se nel codice di *CountingSort* invertiamo l'ordine dell'ultimo ciclo:
- A) L'algoritmo diventa scorretto.
B) L'algoritmo diventa non stabile.
C) L'algoritmo rimane invariato.
D) Nessuna delle precedenti.
- 49) Sia A un array di n interi positivi, e k un intero positivo dispari. Utilizzando il principio di *CountingSort*, ed un array di appoggio fatto di soli valori Booleani, di lunghezza k , possiamo mostrare che il problema di stabilire se esistono $i, j \geq 1$, $i \neq j$, tali che $A[i] + A[j] = k$ ha complessità:
- A) $\Theta(n^2)$.
B) $\Theta(n \cdot \lg(n))$.
C) $O(n)$.
D) $O(\lg(n))$.
- 50) Si consideri la sequenza 4-2-3, 6-2-1, 7-0-3, 1-4-4, 9-4-2, dove le cifre a sinistra sono più significative. Dopo la seconda esecuzione del ciclo **for** più esterno di *RadixSort*, qual è il terzo elemento nell'ordinamento degli elementi?
- A) 7-0-3.
B) 9-4-2.
C) 6-2-1.
D) 4-2-3.
- 51) Si supponga che $S = [3, 12, 5, 1, ?, ?, ?, ?, ?]$ sia uno stack con $S.\max = 8$. Allora è vero che:
- A) $S.\top = 3$; dopo $\text{Push}(S, 6)$ il risultato sarà $S = [3, 12, 5, 1, 6, ?, ?, ?]$ con $S.\top = 4$, e dopo $\text{Pop}(S)=6$, e $\text{Pop}(S)=1$ avremo $S = [3, 12, 5, ?, ?, ?, ?, ?]$ con $S.\top = 2$.
B) $S.\top = 4$; dopo $\text{Push}(S, 6)$ il risultato sarà $S = [3, 12, 5, 1, 6, ?, ?, ?]$ con $S.\top = 5$, e dopo $\text{Pop}(S)=6$, e $\text{Pop}(S)=1$ avremo $S = [3, 12, 5, ?, ?, ?, ?, ?]$ con $S.\top = 3$.
C) $S.\top = 4$; dopo $\text{Push}(S, 6)$ il risultato sarà $S = [3, 12, 5, 1, 6, ?, ?, ?]$ con $S.\top = 5$, e dopo $\text{Pop}(S)=1$, e $\text{Pop}(S)=6$ avremo $S = [3, 12, 5, ?, ?, ?, ?, ?]$ con $S.\top = 3$.
D) $S.\top = 4$; dopo $\text{Push}(S, 6)$ il risultato sarà $S = [3, 12, 5, 1, 6, ?, ?, ?]$ con $S.\top = 5$, e dopo $\text{Pop}(S)=3$, e $\text{Pop}(S)=12$ avremo $S = [?, ?, 5, 1, 6, ?, ?, ?]$ con $S.\top = 3$.
- 52) L'idea di implementare una coda su un array in maniera circolare è utile perché?
- A) In realtà implementando una coda su un array in maniera lineare si otterrebbero le stesse prestazioni.
B) Perchè quest'idea consente di risparmiare spazio in memoria, considerato che occupano solo le posizioni che hanno correntemente un valore.

- C) Perchè quest'idea consente di risparmiare tempo, considerato che l'implementazione lineare richiederebbe $\Omega(n)$ per ogni operazione di *Dequeue*.
 D) Perchè quest'idea consente di assicurare la correttezza di *Enqueue* e *Dequeue*: l'implementazione lineare potrebbe essere scorretta.
- 53) È l'array $A = [23, 17, 14, 6, 13, 10, 1, 5, 7, 12]$ una max-heap?
 A) Sí.
 B) No.
 C) No, perchè la proprietà delle max-heap non è rispettata per nessuna terna padre-destro-sinistro.
 D) No, perchè la proprietà delle max-heap non è rispettata per almeno una terna padre-destro-sinistro.
- 54) Considera le chiavi 1,2,3,4. Quante max-heap diverse si possono costruire con queste chiavi?
 A) 3.
 B) 4.
 C) 2.
 D) 1.
- 55) Si consideri l'array $A = [16, 4, 10, 14, 7, 9, 3, 2, 8, 1]$. Quale delle seguenti affermazioni è vera?
 A) A può essere una max-heap. Inoltre, applicando *MaxHeapify*($A, 2$), l'algoritmo termina alla terza chiamata ricorsiva, e restituisce una permutazione degli elementi di A che la rende una min-heap.
 B) A non può essere una max-heap. Inoltre, applicando *MaxHeapify*($A, 2$), l'algoritmo termina alla prima chiamata ricorsiva, e restituisce una permutazione degli elementi di A che la rende una max-heap.
 C) A non è una max-heap. Inoltre, applicando *MaxHeapify*($A, 2$), l'algoritmo termina alla terza chiamata ricorsiva, e restituisce una permutazione degli elementi di A che la rende una max-heap.
 D) A non è una max-heap. Inoltre, applicando *MaxHeapify*($A, 2$), l'algoritmo fallisce perchè viene chiamato su una struttura che non rispetta le precondizioni richieste.
- 56) È possibile che la min-heap H sia di altezza 3 e contenga, nell'ordine, esattamente i seguenti elementi: 1, 5, 7, 9, 12, 10, 8?
 A) No, perchè se H è di altezza 3 deve contenere almeno 8 elementi.
 B) No, perchè l'ordine non rispetta la proprietà.
 C) No, perchè l'albero corrispondente non sarebbe completo.
 D) Sí.
- 57) Si consideri l'array $A = [7, 1, 14, 2, 10, 20]$, ed si applichi *BuildMaxHeap* su A . Quanti elementi di A sono rimasti nella stessa posizione prima e dopo l'esecuzione?
 A) 1.
 B) 2.
 C) Nessuno.
 D) 3.
- 58) Si considerino k array di numeri interi A_1, \dots, A_k , tutti ordinati, tali che il numero totale di elementi degli array è n . Qual è la complessità del problema di produrre un array B che contiene tutti gli n numeri ordinati?
 A) $O(k \cdot \lg(k))$.
 B) $O(n \cdot \lg(k))$.
 C) $O(k \cdot \lg(n))$.
 D) $O(n \cdot \lg(n))$.
- 59) Si immagini che A contenga n chiavi, con la seguente ipotesi: A è divisibile in k parti disgiunte, tale che ogni parte è ordinata in maniera crescente oppure decrescente. Qual è la complessità del problema di produrre un array B che contiene tutti gli n numeri ordinati?
 A) $O(k \cdot \lg(k))$.
 B) $O(n \cdot \lg(k))$.
 C) $O(k \cdot \lg(n))$.
 D) $O(n \cdot \lg(n))$.
- 60) Si consideri una min-heap. È noto che l'estrazione del minimo costa $O(\lg(n))$, dove n è il numero di elementi della heap. Quanto costa l'estrazione del massimo da una min-heap?
 A) $\Theta(n^2)$.
 B) $O(\lg(n))$, applicando lo stesso algoritmo per l'estrazione del minimo.
 C) $O(\lg(n))$.
 D) $O(n)$.
- 61) Si consideri il problema di trovare il k -esimo elemento più piccolo di un array A di n posizioni, senza ordinarlo. Usando una max-heap possiamo trovare un algoritmo di complessità:
 A) $O(\lg(n)^2)$.
 B) $O(k + (n - k) \cdot \lg(k))$ nel caso peggiore.
 C) $O(\lg(n))$ nel caso peggiore.
 D) Non si può usare una max-heap per risolvere questo problema.
- 62) Si consideri il problema di trovare il k -esimo elemento più piccolo di un array A di n posizioni, senza ordinarlo. Usando una min-heap, possiamo trovare un algoritmo di complessità:
 A) $\Theta(n)$.
 B) $O(n + k \cdot \lg(n))$ nel caso peggiore.
 C) $O(k \cdot \lg(n))$ nel caso peggiore.
 D) Non si può usare una min-heap per risolvere questo problema.
- 63) Supponiamo che $H = [9, 7, 2, 1, 4, 1, 1]$ sia una max-heap, e supponiamo di chiamare *IncreaseKey*($H, 6, 6$). Qual è il risultato?
 A) $H = [9, 7, 1, 6, 2, 4, 1]$.
 B) $H = [9, 1, 6, 1, 4, 7, 2]$.
 C) $H = [9, 6, 7, 1, 4, 2, 1]$.
 D) $H = [9, 7, 6, 1, 4, 2, 1]$.
- 64) Quale, tra queste, è una invariante di *DecreaseKey*?

- A) All'inizio di ogni iterazione del ciclo **while**, $H[Parent(i)]$ è necessariamente minore di $H[i]$.
- B) All'inizio di ogni iterazione del ciclo **while**, $H[Parent(i)]$ è necessariamente radice di una min-heap.
- C) All'inizio di ogni iterazione del ciclo **while**, H è una min-heap ovunque, eccetto, al peggio, l'indice i , perché $H[i]$ potrebbe essere minore di $H[Parent(i)]$.
- D) All'inizio di ogni iterazione del ciclo **while**, H è una min-heap ovunque, eccetto, al peggio, l'indice i , perché $H[i]$ potrebbe essere maggiore di $H[Parent(i)]$.
- 65) Supponiamo che T sia una tabella hash con conflitti risolti via chaining attraverso liste, e che la dimensione di T sia 9 (dalla posizione 1 alla posizione 9 entrambe comprese). Supponiamo che $h(k) = k \bmod 9 + 1$. In T inizialmente vuota si inseriscono le chiavi 5, 28, 19, 15, 20, 33, 12, 17, 10. Qual è, alla fine di tutti gli inserimenti, lo stato della lista puntata dalla posizione 2, letta da sinistra a destra?
- A) 28,10,19.
 B) 10,28.
 C) 28,10,15.
 D) 10,19,28.
- 66) Supponiamo di dover memorizzare 10000 chiavi intere diverse, e che abbiamo a disposizione fino a 500 slot ($1 \leq m \leq 500$) in una tabella hash con conflitti risolti via chaining. Volendo usare il metodo del modulo, quale, tra le seguenti, è la migliore scelta per m ?
- A) 317.
 B) 255.
 C) 257.
 D) 500.
- 67) Supponiamo di avere una tabella hash con conflitti risolti via chaining, $m = 7$ (posizioni dalla 1 alla 7 comprese), e di usare il metodo della moltiplicazione con $A = 0.5$ per inserire nella tabella, inizialmente vuota, i seguenti valori nell'ordine: 0.37, 12, 124, 3.47, 2.56, 41. Quale posizione della tabella avrà, alla fine degli inserimenti, la lista più lunga?
- A) La 1.
 B) La 4.
 C) La 7.
 D) La 1 e la 2.
- 68) Supponiamo di avere una tabella hash con indirizzamento aperto, $m = 7$ (posizioni dalla 1 alla 7 comprese), e di usare il metodo del probing lineare per inserire nella tabella, inizialmente vuota, i seguenti valori nell'ordine: 12, 5, 8, 19. Quale posizione della tabella occuperà l'elemento a chiave 19, se $h'(k) = (k \bmod m) + 1$?
- A) La 1.
 B) La 4.
 C) La 7.
 D) Nessuna delle precedenti.
- 69) Si consideri l'implementazione basata su liste senza unione pesata per insiemi disgiunti, e si consideri il risultato dell'analisi ammortizzata per m operazioni di cui n *MakeSet*. Qual è, nel caso peggiore, il costo medio di ogni operazione?
- A) $\Theta(1)$.
 B) $\Theta(\lg(n))$.
 C) $\Theta(\lg^2(n))$.
 D) $\Theta(n)$.
- 70) In una foresta \mathcal{S} di alberi k -ari che rappresentano insiemi disgiunti, supponiamo che $S_1 = \{2, 3, 7\}$, dove $2.p = 2$, $3.p = 2$, e $7.p = 3$, e che $S_2 = \{4, 8\}$, dove $4.p = 4$ e $8.p = 4$. Quali sono i minimi ranghi possibili di 2 e di 4, e cosa accade, in questo caso, eseguendo *Union*(3, 8), assumendo di utilizzare l'unione per rango e la compressione del percorso?
- A) $2.rank = 2$, $4.rank = 3$, e il risultato è un solo insieme, radicato in 4, tale che i padri di S_1 sono invariati, $4.p = 8$, e $2.rank = 1$.
 B) $2.rank = 3$, $4.rank = 1$, e il risultato è un solo insieme, radicato in 3, tale che i padri di S_1 sono invariati, $4.p = 3$, e $2.rank = 2$.
 C) $2.rank = 2$, $4.rank = 1$, e il risultato è un solo insieme, radicato in 2, tale che i padri di S_1 sono invariati, $4.p = 2$, e $2.rank = 2$.
 D) Nessuna delle altre risposte è corretta.
- 71) Si considerino le seguenti operazioni su insiemi disgiunti implementati con foreste di alberi (con unione per rango e compressione del percorso) sulle chiavi a, b, c, d, e , nell'ordine: *MakeSet*(a), *MakeSet*(b), *Union*(a, b), *MakeSet*(c), *MakeSet*(d), *Union*(c, d), *Union*(a, c). Qual è il rango del rappresentante di c alla fine delle operazioni?
- A) 1.
 B) 2.
 C) 3.
 D) 4.
- 72) Nella rappresentazione degli insiemi disgiunti con foreste di alberi k -ari, con unione per rango e compressione del percorso, quale delle seguenti è vera?
- A) La complessità di *MakeSet* è $\Theta(n+m)$, nel caso in cui abbiamo fatto m operazioni di cui n *MakeSet* prima di quella analizzata.
 B) La complessità di *Union* nel caso peggiore è $\Theta(n)$, dove n è il numero di operazioni di *MakeSet* che sono state fatte precedentemente.
 C) La complessità di *FindSet* nel caso peggiore è $O(1)$.
 D) La complessità di *FindSet* nel caso peggiore è $O(h)$, dove h è l'altezza dell'albero su cui l'operazione è chiamata.
- 73) In una foresta \mathcal{S} di alberi k -ari che rappresentano insiemi disgiunti, supponiamo che $S_1 = \{2, 3, 7\}$, dove $2.p = 2$, $3.p = 2$, e $7.p = 3$, e che $S_2 = \{4, 8\}$, dove $4.p = 4$ e $8.p = 4$. Quali sono i minimi ranghi possibili di 2 e di 4, e cosa accade, in questo caso, eseguendo

- Union(3,8)*, assumendo di utilizzare l'unione per rango e la compressione del percorso?
- A) $2.rank = 2$, $4.rank = 3$, e il risultato è un solo insieme, radicato in 4, tale che i padri di S_1 sono invariati, $4.p = 8$, e $2.rank = 1$.
 B) $2.rank = 3$, $4.rank = 1$, e il risultato è un solo insieme, radicato in 3, tale che i padri di S_1 sono invariati, $4.p = 3$, e $2.rank = 2$.
 C) $2.rank = 2$, $4.rank = 1$, e il risultato è un solo insieme, radicato in 2, tale che i padri di S_1 sono invariati, $4.p = 2$, e $2.rank = 2$.
 D) Nessuna delle altre risposte è corretta.
- 74) Si considerino le seguenti operazioni su insiemi disgiunti implementati con foreste di alberi, con unione per rango e compressione del percorso, sulle chiavi a, b, c, d, e , nell'ordine: $MakeSet(a)$, $MakeSet(b)$, $Union(a, b)$, $MakeSet(c)$, $MakeSet(d)$, $Union(c, d)$, $Union(a, c)$. Qual è il rango del rappresentante di c alla fine delle operazioni?
- A) 1.
 B) 2.
 C) 3.
 D) 4.
- 75) Nella rappresentazione degli insiemi disgiunti con foreste di alberi k -ari, con unione per rango e compressione del percorso, quale delle seguenti è vera?
- A) La complessità di $MakeSet$ è $\Theta(n+m)$, nel caso in cui abbiamo fatto m operazioni di cui n $MakeSet$ prima di quella analizzata.
 B) La complessità di $Union$ nel caso peggiore è $\Theta(n)$, dove n è il numero di operazioni di $MakeSet$ che sono state fatte precedentemente.
 C) La complessità di $FindSet$ nel caso peggiore è $O(1)$.
 D) La complessità di $FindSet$ nel caso peggiore è $O(h)$, dove h è l'altezza dell'albero su cui l'operazione è chiamata.
- 76) Si consideri un albero binario qualsiasi tale che la sequenza *BAFILGSCHD* è il risultato della sua visita pre-order, e la sequenza *IFLAGSBHCD* quello della sua visita in-order. Quanto è alto l'albero?
- A) 3.
 B) 4.
 C) 2.
 D) 5.
- 77) In base al ragionamento utilizzato nell'esercizio precedente, qual è l'equazione di ricorrenza che descrive la complessità del caso peggiore di una procedura ricorsiva che ricostruisce un albero a partire dalla sua visita in-order e pre-order?
- A) $T(n) = T(n - 1) + O(n)$.
 B) $T(n) = T(\frac{n}{2}) + O(n)$.
 C) $T(n) = 2 \cdot T(\frac{n}{2}) + O(n)$.
 D) $T(n) = T(\frac{n}{2}) + O(n^2)$.
- 78) Quali sono le differenze più rilevanti tra gli alberi utilizzati nelle strutture per insiemi disgiunti (implementazione con foreste di alberi k -ari) e quelli utilizzati come struttura dati per inserimento, cancellazione, visite, e operazioni simili?
- A) Gli alberi utilizzati come base per insiemi disgiunti sono sempre k -ari, mentre quelli utilizzati come struttura dati classica sono sempre binari.
 B) Non c'è alcuna differenza.
 C) Gli alberi utilizzati come base per insiemi disgiunti non hanno puntatori ai figli; quindi, non possono essere visitati, né la loro struttura può essere mantenuta da operazioni di inserimento o cancellazione, che non hanno senso.
 D) Gli alberi utilizzati come base per insiemi disgiunti in realtà sono array visiti come alberi.
- 79) In ogni albero binario di ricerca, ogni nodo con due figli è tale che:
- A) Il suo successore ha sempre un figlio sinistro.
 B) Il suo predecessore ha sempre un figlio destro.
 C) Il suo successore non ha mai un figlio sinistro.
 D) Tutte le altre risposte sono incorrette.
- 80) È possibile avere un albero binario di ricerca con le chiavi $\{1, 4, 5, 10, 16, 17, 21\}$ e di altezza 6? E di altezza 2? In questo ultimo caso, quale sarebbe la chiave della radice?
- A) Sí può avere di altezza 2, ma non di altezza 6.
 B) Sí può avere di altezza 6, ma non di altezza 2.
 C) Sí in entrambi i casi; per avere un albero binario di ricerca di altezza 2 esiste una sola soluzione e la chiave della radice è 5.
 D) Sí in entrambi i casi; per avere un albero binario di ricerca di altezza 2 esiste una sola soluzione e la chiave della radice è 10.
- 81) In un albero binario di ricerca T sono presenti tutti e soli i numeri interi da 1 a 100. Cercando la chiave 55, quale delle seguenti *non* può essere una sequenza di chiavi visitate?
- A) 10, 75, 64, 43, 60, 57, 55.
 B) 90, 12, 68, 34, 62, 45, 55.
 C) 9, 85, 47, 68, 43, 57, 55.
 D) 79, 14, 72, 56, 16, 53, 55.
- 82) Considerando la nostra implementazione di *BSTTreeDelete*, è vero che l'operazione di eliminazione di una chiave da un albero binario di ricerca è *commutativa*? È vero, cioè, che se si eliminano dallo stesso oggetto 2 nodi diversi in un particolare ordine, si ottiene esattamente lo stesso risultato che eliminandoli nell'ordine inverso?
- A) No, in generale non è vero.
 B) No. Un controesempio è dato dall'eliminazione di due nodi foglia.
 C) È impossibile stabilirlo.
 D) Sí.
- 83) Considerando il albero binario di ricerca in Fig. 2, e immaginando una versione di *BSTTreeDelete* nella quale si utilizzi il predecessore immediato, invece del successore immediato, dopo aver eliminato la chiave 14, l'oggetto risultante:

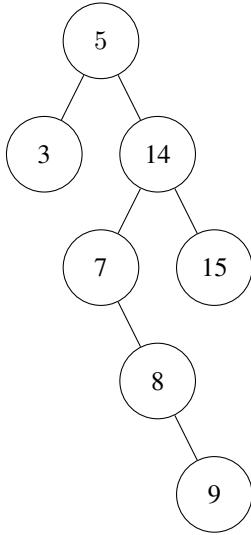


Figure 2. Codice per l'esercizio 83.

```

proc MergeTreeArray (T, A)
  x = TreeMinimum(T.root)
  i = 1
  while (*)
    if (x.key ≤ A[i])
      then
        Print(x.key)
        x = TreeSuccessor(x)
      else
        Print(A[i])
        i = i + 1
    while (x ≠ Nil)
      Print(x.key)
      x = TreeSuccessor(x)
    while (i ≤ A.length)
      Print(A[i])
      i = i + 1
  
```

Figure 3. Codice per l'esercizio 84.

- A) Ha altezza 2, la chiave della radice è 9, e il figlio sinistro del nodo che contiene 7 contiene la chiave 3.
 B) Ha altezza 4, la chiave della radice è 7, e il figlio sinistro del nodo che contiene 9 contiene la chiave 3.
 C) Ha altezza 3, la chiave della radice è 5, e il figlio sinistro del nodo che contiene 7 contiene la chiave 9.
 D) Ha altezza 3, la chiave della radice è 5, e il figlio sinistro del nodo che contiene 9 contiene la chiave 7.
- 84) Si supponga che T sia un albero binario di ricerca con chiavi intere, e che A sia un array di interi ordinato, e di voler stampare la successione di elementi di $T \cup A$ in ordine non decrescente. Si consideri il codice in Fig. 3. Qual è una istruzione corretta da inserire al posto di $*$?
- A) $(x \neq \text{Nil})$ and $(A[i] \leq A[A.length])$.
 B) $(x = \text{Nil})$ and $(i \leq A.length)$.
 C) $(x \neq \text{Nil})$ or $(i \leq A.length)$.
 D) $(x \neq \text{Nil})$ and $(i \leq A.length)$.
- 85) Si consideri due alberi binari di ricerca T, T' , tali che ogni chiave di T è minore di ogni chiave di T' . Si consideri il problema di costruire un albero binario di ricerca T'' che contiene tutte e sole le chiavi di $T \cup T'$, ed il codice

```

proc BSTUnion (T, T')
  T'' = ∅
  x = TreeMinimum(T')
  TreeDelete(T', x)
  T''.root = x
  x.left = T.root
  x.right = T'.root
  x.p = Nil
  return T''
  
```

Figure 4. Codice per l'esercizio 85.

```

proc StillBST (T, x)
  y = TreeSuccessor(T, x)
  if (y = Nil)
    then return true
  if (*)
    then return true
  return false
  
```

Figure 5. Codice per l'esercizio 95.

in Fig. 4. È vero che:

- A) La procedura è corretta.
 B) La procedura è incorretta.
 C) La procedura è corretta, ma sotto l'ipotesi che T e T' siano bilanciati.
 D) La procedura è corretta, ma sotto l'ipotesi che T e T' siano completi.
- 86) Si consideri un albero binario di ricerca T , ed un nodo $x \in T$. Supponiamo di cambiare la chiave $x.key$ con una chiave più grande, ed il problema di stabilire se T' risultante è ancora un albero binario di ricerca. Si consideri il codice in Fig. 8. Al posto di $*$ possiamo inserire:
- A) $x.key \geq y.key$.
 B) $x.key = y.key$.
 C) $x.key \leq y.key$.
 D) $x.key \neq y.key$.
- 87) Si supponga che T sia un albero qualsiasi con chiavi intere, che Cl sia una variabile globale condivisa da tutte le chiamate di PLS e inizializzata a zero, il cui codice è mostrato in Fig. 6, e si supponga di chiamare $PLS(x.root, Cl)$. Qual è l'effetto?
- A) Quello di stampare tutte e sole le chiavi che sono presenti sulla "frontiera" di T , cioè tutte e sole quelle che si trovano su nodi foglia.
 B) Quello di stampare tutte e sole le chiavi che sono presenti sul "ramo di mezzo" di T , cioè tutte e sole quelle che si trovano su nodi che hanno altri nodi sia a destra che a sinistra sullo stesso livello.
 C) Quello di stampare tutte e sole le chiavi che sono presenti sul "costato sinistro" di T , cioè tutte e sole quelle che si trovano su nodi non nascosti da nodi allo stesso livello e più a sinistra.
 D) Quello di stampare tutte e sole le chiavi che sono presenti sui "rami dispari" di T , cioè tutte e sole quelle che si trovano su nodi che appartengono a rami con un indice dispari contando da uno, da sinistra verso destra,

```

proc PLS (x, Ml)
  if (Cl < Ml)
    then
      { Print(x.key)
        Cl = Ml
        if (x.left ≠ Nil)
          then PLS(x.left, Ml + 1)
        if (x.right ≠ Nil)
          then PLS(x.right, Ml + 1)
      }
  
```

Figure 6. Codice per l'esercizio 87.

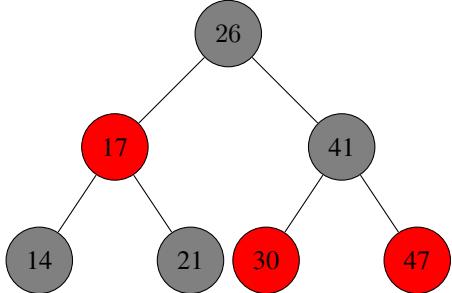


Figure 7. Oggetto per gli esercizi 88 e 90.

sulla foglia del ramo stesso.

- 88) Si consideri il codice di *BSTTreeLeftRotate* visto in classe ed applicala sul nodo x che contiene la chiave 26 nel albero red-black in Fig. 7. Qual è l'altezza dell'oggetto risultante? Qual è la chiave del figlio sinistro della radice?

- A) L'altezza è 2, e il figlio sinistro della radice è **Nil**.
- B) L'altezza è 3, e il figlio sinistro della radice ha chiave 47.
- C) L'altezza è 2, e il figlio sinistro della radice ha chiave 41.
- D) L'altezza è 3, e il figlio sinistro della radice ha chiave 26.**

- 89) Si consideri un albero binario di ricerca con radice 10, figlio sinistro 3 e figlio destro 14; il nodo 3 ha solo figlio destro, con valore 7, ed il nodo 14 ha solo figlio sinistro, con valore 12. Si consideri la procedura *BSTTreeLeftRotate*, costruire la sua corrispondente per rotazione destra, e applicarla all'oggetto descritto sul nodo radice. Qual è il valore del figlio sinistro del nodo 10?

- A) 3.
- B) 12.
- C) 7.**
- D) 14.

- 90) Consideriamo l'albero red-black di Fig. 7, e le regole viste nella teoria:

1. Ogni nodo è rosso o nero;
2. La radice è nera;
3. Ogni foglia (esterna, **Nil**) è nera;
4. Se un nodo è rosso, entrambi i suoi figli sono neri;
5. Per ogni nodo, tutti i percorsi semplici da lui alle sue foglie, contengono lo stesso numero di nodi neri.

Inserendo quale delle seguenti chiavi, in un nodo rosso, violiamo sicuramente la proprietà 4?

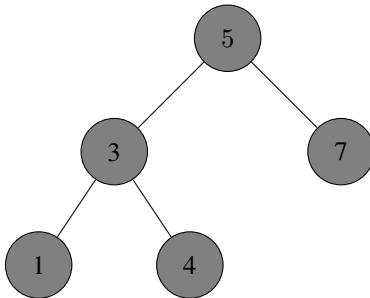


Figure 8. Albero binario di ricerca per l'esercizio 95.

- A) 7.
- B) 31.**
- C) 18.
- D) 23.

- 91) Si consideri il codice visto in classe per l'inserimento di una nuova chiave in un albero red-black T . Supponiamo adesso che T sia un albero red-black inizialmente vuoto, e di inserire, nell'ordine: 10,7,14,9,11,12. Dopo tutti gli inserimenti, che chiave ha il figlio destro della radice e di che colore è?

- A) 12, rosso.
- B) 11, rosso.
- C) 10, nero.
- D) Nessuna è corretta.**

- 92) Sia T un albero red-black con tre nodi interni: 10, la radice, nera, e due figli, 5 e 15, entrambi rossi. Inseriamo, nell'ordine, le chiavi: 25,35,45,55,65. All'inserire quale tra queste chiavi cambia la chiave della radice (inizialmente 10)?

- A) 65.**
- B) 55.
- C) 45.
- D) 35.

- 93) Si consideri il codice visto in classe per l'inserimento di una nuova chiave in un albero red-black T . Supponiamo adesso che T sia un albero red-black inizialmente vuoto, e di inserire, nell'ordine: 15,10,11,8,9. Dopo tutti gli inserimenti, qual è la posizione e qual è il colore del nodo che contiene la chiave 9?

- A) Figlio sinistro della radice, nero.**
- B) Figlio destro della radice, rosso.
- C) Figlio sinistro del nodo che contiene la chiave 15, nero.
- D) Figlio destro del nodo che contiene la chiave 8, rosso.

- 94) Sia T un albero red-black le cui chiavi sono interi. Qual è il costo computazione dell'operazione di *Between* che, dati due valori interi $k < k'$, stampa tutte le chiavi di T di valore compreso tra k e k' ?

- A) $\Omega(n^2)$,
- B) $O(\lg(n))$.
- C) $O(n)$.
- D) $\Theta(n \cdot \lg(n))$.**

95) Supponiamo di applicare *BSTTreeRightRotate* sull'albero binario di ricerca in figura 8, centrato nel nodo che contiene la chiave 5. Quali sono, dopo la rotazione, i figli del nodo che contiene la chiave 5?

- A) Figlio sinistro: 1; figlio destro: 4.
- B) Figlio sinistro: 3; figlio destro: 7.
- C) Figlio sinistro: 1; figlio destro: 7.
- D) Figlio sinistro: 4; figlio destro: 7.**

96) Si consideri un albero B T con tutti i nodi pieni, $t = 50$, e $h = 2$. Quante chiavi contiene?

- A) 100000.
- B) 500000.
- C) 990000.
- D) 999999.**

97) Quanti sono i possibili alberi B che si possono costruire con tutte e sole le chiavi $\{1, 2, 3, 4, 5\}$, con grado minimo 3?

- A) 1.
- B) 2.**
- C) 3.
- D) 4.

98) Qual è la complessità in termini di CPU dell'operazione di ricerca di una chiave in un albero B se, invece della ricerca lineare sul nodo, usiamo la ricerca binaria?

- A) la complessità non cambia.
- B) $O(t \cdot \lg(\lg(n)))$.
- C) $O(\lg_t(n))$.
- D) $O(\lg(n))$.**

99) Sia T un albero B con $t = 3$, composto dalla radice con gli elementi 7, 13, 16, 22, e da 5 figli diretti, nell'ordine, con gli elementi: 1, 3, 4, 5 - 10, 11 - 14, 15 - 18, 19, 20, 21, 22 - 25, 26. Dopo l'esecuzione di *BTTreeInsert*($T, 2$) e *BTTreeInsert*($T, 17$), quante chiavi ha la radice? Quanto è alto T ? Quali sono le chiavi di $T.root.c_4$??

- A) 5. 1. 17, 18, 19, 21, 22.
- B) 5. 2. 18, 19.
- D) 4. 1. 17, 18, 19, 21.**
- D) 5. 1. 17, 18, 19.

100) Si consideri un albero B, con $t = 3$, con un nodo interno x che contiene le chiavi B, L . Tra i suoi figli, il nodo $x.c_2$ contiene le chiavi D, E, F, G, H . Dopo aver eseguito *BTTreeSplitChild*($x, 2$), e assumendo di utilizzare l'ordinamento alfabetico, succede che:

- A) x contiene B, E, L , $x.c_2$ contiene D, F , e $x.c_3$ contiene G, H .
- B) x contiene B, F, L , $x.c_2$ contiene D, E , e $x.c_3$ contiene G, H .**
- C) x contiene F, L , $x.c_2$ contiene B, D, E , e $x.c_3$ contiene G, H .
- D) x contiene B, F, H , $x.c_2$ contiene D, E , e $x.c_3$ contiene G, L .

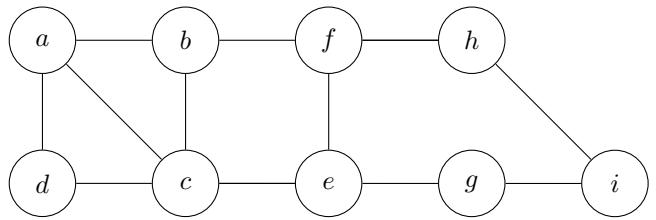


Figure 9. Grafo per gli esercizi 101 e 102.

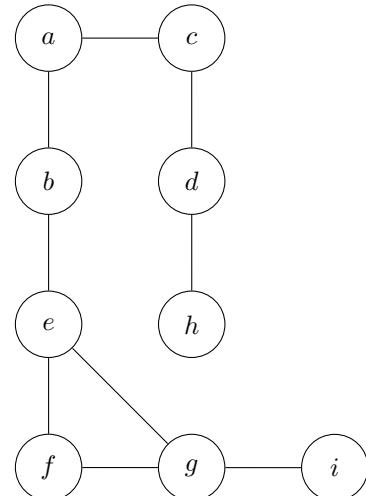


Figure 10. Grafo per gli esercizi 103 e 108.

101) Si esegua *BreadthFirstSearch* sul grafo di Fig. 9 assumendo che la sorgente sia a e che i nodi vengano sempre estratti in ordine alfabetico e si dica, dopo la visita, a cosa punta $e.\pi$:

- A) c .
- B) f .
- C) e .**
- D) g .

102) Si esegua *BreadthFirstSearch* sul grafo di Fig. 9, assumendo che la sorgente sia c e che i nodi vengano sempre estratti in ordine alfabetico e si dica quale delle seguenti è una configurazione della coda che si verifica durante la visita:

- A) $Q = [a, b, f]$.
- B) $Q = [c, b, d]$.
- C) $Q = [d, e, f]$.**
- D) $Q = [a, b, e]$.

103) Si esegua *BreadthFirstSearch* sul grafo di Fig. 10, assumendo che la sorgente sia a e che i nodi vengano sempre estratti in ordine alfabetico e si dica quanti nodi vengono visitati, compresa la sorgente, prima di visitare il nodo h , escluso.

- A) 7.**
- B) 8.
- C) 5.
- D) 4.

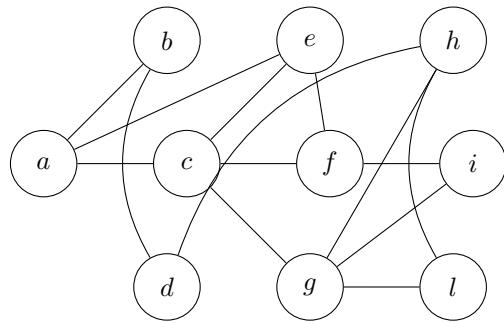


Figure 11. Grafo per l'esercizio 104.

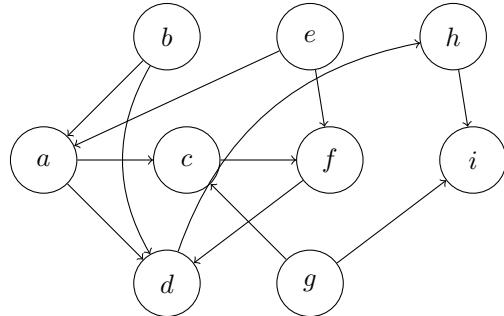


Figure 12. Grafo per gli esercizi 106 e 107.

- 104) Sia G il grafo in Fig. 11. Assumendo che $s = a$, e che i vertici in una lista di adiacenza siano visitati in ordine alfabetico, qual è lo stato della coda Q durante l'esecuzione di $BreadthFirstSearch(G, s)$ nel momento in cui i vertici nell'insieme f diventa nero?

- A) $Q = [g, h, i]$.
- B) $Q = \emptyset$.
- C) $Q = [b, c, d]$.
- D) $Q = [h, i, f, e]$.

- 105) Si consideri un grafo indiretto non pesato, ed il problema di stabilire quali sono le sue componenti connesse. Una applicazione di $BreadthFirstSearch$ potrebbe risolvere il problema, ma quale potrebbe essere una soluzione alternativa?

- A) Un algoritmo di ordinamento.
- B) Una soluzione ispirata a *Partition*.
- C) Una soluzione basata su strutture dati per insiemi disgiunti.
- D) Un algoritmo di ricerca binaria.

- 106) Si esegua $DepthFirstSearch$ sul grafo di Fig. 12, assumendo che la prima sorgente sia a e che i nodi vengano sempre estratti in ordine alfabetico e si dica, dopo la visita, qual è l'intervallo inizio scoperta - fine scoperta del vertice d .

- A) $[17, 18]$.
- B) $[3, 4]$.
- C) $[4, 12]$
- D) $[4, 9]$.

- 107) Si esegua $DepthFirstSearch$ sul grafo di Fig. 12, assumendo che la prima sorgente sia a e che i nodi vengano

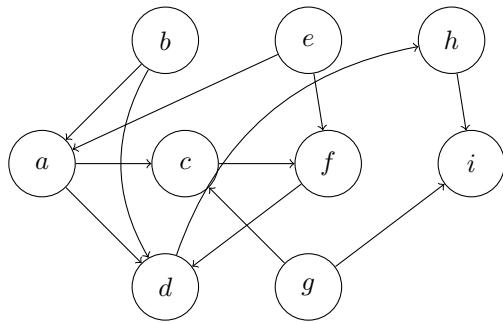


Figure 13. Grafo per l'esercizio 110.

sempre estratti in ordine alfabetico e si dica, dopo la visita, qual è il vertice scoperto per ultimo.

- A) g .
- B) h .
- C) e
- D) b .

- 108) Si esegua $DepthFirstSearch$ sul grafo di Fig. 10, assumendo che la sorgente sia a e che i nodi vengano sempre estratti in ordine alfabetico e si dica quanti nodi vengono visitati, compresa la sorgente, prima di visitare il nodo h , escluso.

- A) 7.
- B) 8.
- C) 5.
- D) 4.

- 109) Dato un grafo diretto G e due vertici distinti u, v , dopo un'esecuzione di $DepthFirstSearch$ possiamo dire che u è un discendente di v in uno degli alberi della foresta se solo se:

- A) Non è possibile stabilirlo semplicemente osservando i tempi di inizio e fine visita di u e v .
- B) Gli intervalli $[u.d, u.f]$ e $[v.d, v.f]$ sono completamente disgiunti.
- C) $[v.d, v.f]$ è contenuto in $[u.d, u.f]$,
- D) $[u.d, u.f]$ è contenuto in $[v.d, v.f]$.

- 110) Qual è un ordinamento topologico del grafo in Fig. 13?

- A) $c, f, a, b, d, h, i, g, e$.
- B) $c, e, b, a, d, h, i, g, f$.
- C) $g, e, b, a, c, f, d, h, i$.
- D) $g, e, b, f, d, h, i, c, a$.

- 111) Qual è un ordinamento topologico del grafo di Fig. 14?

- A) $0, 1, 4, 5, 2, 3$.
- B) $5, 2, 4, 3, 1, 0$.
- C) $0, 1, 2, 5, 4, 3$.
- D) Il grafo nella figura non ha un ordinamento topologico.

- 112) Si consideri la seguente affermazione: in ogni grafo diretto aciclico, l'ordinamento topologico è unico. Questa affermazione:

- A) È vera. Possiamo dimostrarlo per induzione sulla cardinalità dell'insieme E degli archi.

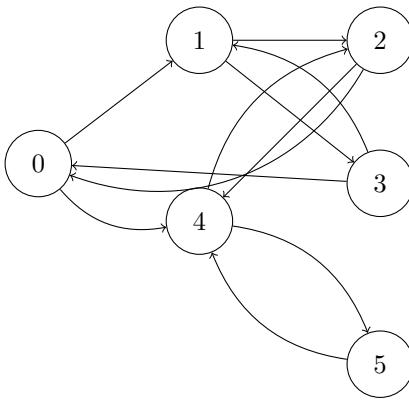


Figure 14. Grafo per l'esercizio 111.

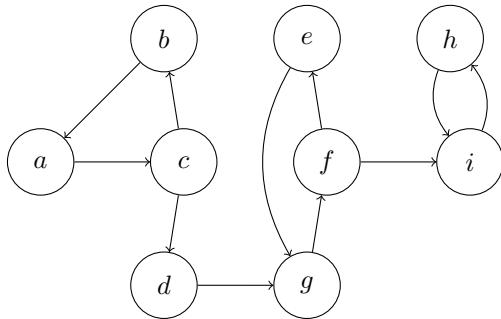


Figure 15. Grafo diretto per l'esercizio 114.

- B) È vera. Possiamo dimostrarlo per induzione sulla cardinalità dell'insieme V dei vertici.
- C) È falsa. Un controsenso potrebbe essere un grafo con due soli vertici a, b , e nessun arco, che ammette due ordinamenti topologici. Uno o l'altro vengono restituiti da *TopologicalSort* dipendendo dall'ordine in cui i vertici sono esaminati da *DepthFirstSearch*.
- D) È falsa. Un controsenso potrebbe essere un grafo con due soli vertici a, b , ed un solo arco da a verso b , che ammette due ordinamenti topologici. Uno o l'altro vengono restituiti da *TopologicalSort* dipendendo dall'ordine in cui i vertici sono esaminati da *DepthFirstSearch*.
- 113) Si consideri il seguente problema. Sia dato un grafo non pesato con archi diretti e non diretti (cioè alcuni archi sono diretti, ed altri no). Per ipotesi, il grafo, limitato ai soli archi diretti, è aciclico, e si vuole dare una direzione agli archi che non sono diretti in maniera di assicurare che il grafo risultante, considerando tutti gli archi, è ancora aciclico. Quale, tra i seguenti algoritmi noti, può essere utile a questo scopo?
- A) Ordinamento topologico.
- B) Albero di copertura minima.
- C) Componenti fortemente connesse.
- D) Il problema non è risolvibile in maniera efficiente; è necessario provare tutte le possibilità e controllare, per ognuna di esse, se si è formato un ciclo.

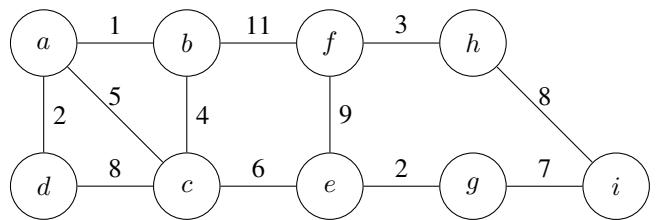


Figure 16. Grafo per gli esercizi 117 e 124.

- 114) Quante sono le componenti fortemente connesse del grafo in Fig. 15?
- A) 1.
- B) 2.
- C) 3.
- D) 4.
- 115) Quale delle seguenti affermazioni è falsa, rispetto a grafi diretti?
- A) Eliminando archi, il numero delle componenti fortemente connesse aumenta o rimane uguale.
- B) Aggiungendo archi, il numero delle componenti fortemente connesse diminuisce sempre.
- C) Il numero delle componenti fortemente connesse è compreso tra 1 e il numero dei vertici.
- D) Il grafo delle componenti fortemente connesse può essere topologicamente ordinato.
- 116) Sia G un grafo diretto in cui ogni vertice rappresenta un punto di interesse in una città, e ogni arco rappresenta una strada, a senso unico, che connette due punti di interesse. Per poter ottimizzare i percorsi turistici della città, un sistema basato su intelligenza artificiale ha bisogno di sapere se esistono, e quante sono, zone della città dalle quali, una volta usciti, non è possibile rientrare senza violare un senso unico. Quale algoritmo su grafi, tra quelli che abbiamo studiato, può essere utile a questo scopo?
- A) *MST-Kruskal*.
- B) *MST-Prim*.
- C) *StronglyConnectedComponents*.
- D) *Dijkstra*.
- 117) Utilizzare *MST-Prim* per calcolare il peso di un albero di copertura minima del grafo in Fig. 16?
- A) 17.
- B) 24.
- C) 31.
- D) 33.
- 118) In *MST-Prim*, l'istruzione $v.key = w(u, v)$ è, in realtà, una istruzione di?
- A) *IncreaseKey*.
- B) *Enqueue*.
- C) *Dequeue*.
- D) *DecreaseKey*.
- 119) Sia G il grafo in Fig. 17. Quale arco non appare nel MST prodotto da *MST-Prim* se $r = a$ e se, in caso di chiavi uguali, la coda di priorità utilizza l'ordine alfabetico?

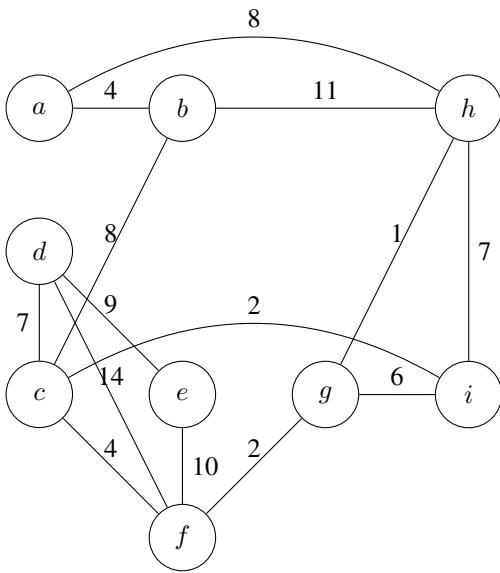


Figure 17. Grafo per l'esercizio 119.

- 121) Qual è la complessità di *MST-Prim* eseguito su grafi sparsi (quindi con $|E| << |V|^2$) e implementando la coda di priorità con un array senza struttura?
- A) I vari elementi dell'algoritmo sommano così: $\Theta(|V|)$ (inizializzazione), $\Theta(|V|)$ (costruzione della coda), $\Theta(|V|^2)$ (per le estrazioni del minimo), e $\Theta(|E|)$ (per i decrementi, analisi ammortizzata, non si può sostituire con $\Theta(|E|)$ sotto queste ipotesi). Il costo totale diventa dunque $\Theta(|V| + |E|)$.
- B) I vari elementi dell'algoritmo sommano così: $\Theta(|V|)$ (inizializzazione), $\Theta(|V|)$ (costruzione della coda), $\Theta(|E|)$ (per le estrazioni del minimo, sotto l'ipotesi di grafo denso), e $\Theta(|E|)$ (per i decrementi, analisi ammortizzata). Il costo totale è $\Theta(|E|)$.
- C) I vari elementi dell'algoritmo sommano così: $\Theta(|V|)$ (inizializzazione), $\Theta(|V|^2)$ (costruzione della coda, sotto l'ipotesi di grafo denso), $\Theta(|V|^2)$ (per le estrazioni del minimo), e $\Theta(|E|)$ (per i decrementi, analisi ammortizzata). Il costo totale è $\Theta(|V|^2 + |E|)$.
- D) I vari elementi dell'algoritmo sommano così: $\Theta(|V|)$ (inizializzazione), $\Theta(|V|)$ (costruzione della coda,
- A)** (a, h) .
B) (c, i) .
C) (a, b) .
D) (c, d) .
- 122) Qual è la complessità di *MST-Prim* eseguito su grafi densi (quindi con $|E| = O(|V|^2)$) e implementando la coda di priorità con una heap binaria?
- A) I vari elementi dell'algoritmo sommano così: $\Theta(|V|)$ (inizializzazione), $\Theta(|V|)$ (costruzione della coda), $\Theta(|V| \cdot \lg(|V|))$ (per le estrazioni del minimo), e $\Theta(|E| \cdot \lg(|V|)) = \Theta(|V|^2 \cdot \lg(|V|))$ (per i decrementi, analisi ammortizzata), per un totale di $\Theta(|V|^2 \cdot \lg(|V|))$.
- B) I vari elementi dell'algoritmo sommano così: $\Theta(|V|)$ (inizializzazione), $\Theta(|V|)$ (costruzione della coda), $\Theta(|V| \cdot \lg(|V|))$ (per le estrazioni del minimo), e $\Theta(|E| \cdot \lg(|E|)) = \Theta(|V| \cdot \lg(|E|))$ (per i decrementi, analisi ammortizzata), per un totale di $\Theta(|V| \cdot \lg(|E|))$.
- C) In questo caso, l'elemento dell'algoritmo che prevale sugli altri è l'inizializzazione della coda, dunque la complessità totale è $\Theta(|V|^2)$.
- D) Tutte le altre risposte sono sbagliate.
- 123) Supponiamo che G sia un grafo indiretto pesato. Possiamo utilizzare *MST-Prim* oppure *MST-Kruskal* per calcolare il peso di un albero di copertura massimo?
- A) No. Il problema è ben definito, ma è necessario utilizzare un altro tipo di algoritmo.
- B) No. La definizione di albero di copertura massimo è priva di senso.
- C) Si. È sufficiente invertire il senso di tutte le disuguaglianze.
- D) Si. *MST-Prim* calcola l'albero di copertura massimo senza che sia necessario alcun cambio; *MST-Kruskal*, invece, no.
- 124) Si consideri il grafo in Fig.16, ed applicare *MST-Kruskal* per calcolare l'albero di copertura minimo. Quale, tra questi archi, non è certamente parte del risultato, se gli archi di peso uguale sono ordinati lessicograficamente?
- A) (e, c) .
B) (e, g) .
C) (c, d) .
D) (h, i) .
- 125) Qual è la complessità di *MST-Kruskal* eseguito su grafi densi (quindi con $|E| = O(|V|^2)$), quando gli insiemi disgiunti sono gestiti con liste e utilizzando l'unione pesata?
- A) *MST-Kruskal* non può essere implementato utilizzando insiemi disgiunti su liste.
- B) Abbiamo: inizializzazione ($O(1)$), ordinamento: $(\Theta(|E| \cdot \lg(|E|)))$, e $O((|V| + |E|)$ diverse operazioni su insiemi, di cui $O(|V|)$ sono *MakeSet* (per un totale di $\Theta(|V|^2)$). Totale, $\Theta(|E|^2)$. Quindi, su grafi densi, l'implementazione con foreste di alberi ha chiaramente un impatto sulla complessità asintotica.

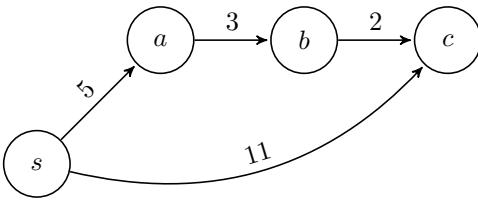


Figure 18. Grafo per l'esercizio 128.

- C) Abbiamo: inizializzazione ($O(1)$), ordinamento: $(\Theta(|E| \cdot \lg(|E|))) = \Theta(|V|^2 \cdot \lg(|E|)) = \Theta(|V|^2 \cdot \lg(|V|))$, e $O(|V| + |E|)$ diverse operazioni su insiemi, di cui $O(|V|)$ sono *MakeSet* (per un totale di $\Theta(|V| \cdot \lg(|V|))$). Totale, $\Theta(|V|^2 \cdot \lg(|V|))$. Quindi, su grafi densi, l'implementazione con foreste di alberi non ha un impatto sulla complessità asintotica.
D) Tutte le altre risposte sono sbagliate.

126) Si supponga che G sia un grafo indiretto, pesato, e che tutti i pesi siano interi tra 1 e un numero $O(|E|)$. Possiamo implementare *MST-Kruskal* in modo che, sotto queste ipotesi, la sua complessità per grafi sparsi sia asintoticamente migliore di quella calcolata senza ipotesi sul grafo?

- A) Sì. Utilizzando foreste di alberi per implementare insiemi disgiunti, la complessità può abbassarsi a $O(\lg(|E|))$.
B) Sì. Utilizzando foreste di alberi per implementare insiemi disgiunti, la complessità può abbassarsi a $O((|V| + |E|) \cdot \alpha(|V|))$.
C) No. La complessità non risente di queste ipotesi.
D) Tutte le altre risposte sono incorrette.

127) Sia G un grafo diretto pesato, e siano $v_1, v_2, v_3 \in G.V$, con $w(v_1, v_2) = 2, w(v_2, v_3) = 2, w(v_1, v_3) = 3$. Se $v_1.d = 0, v_2.d = 2, v_3.d = 3$, rilassando l'arco (v_2, v_3) :

- A) $v_3.d$ non cambia.
B) $v_3.d$ cambia.
C) $v_3.\pi$ cambia.
D) $v_2.d$ cambia.

128) Assumendo che gli archi nel grafo nella Fig. 18 vengano esplorati da *Bellman-Ford* nell'ordine $(b, c), (a, b), (s, a), (s, c)$ in ogni esecuzione del ciclo **for** più interno, dopo la seconda esecuzione del ciclo **for** più esterno, qual è il valore di $c.d$, assumendo s come sorgente?

- A) 11.
B) 10.
C) 5.
D) ∞ .

129) Si consideri il grafo diretto in Fig. 19. Qual è il valore di $c.\pi$ dopo l'esecuzione di *Bellman-Ford*? E qual è il valore del suo (del padre di c) attributo $.d$, assumendo s come sorgente?

- A) $b, 6$.
B) $a, 6$.

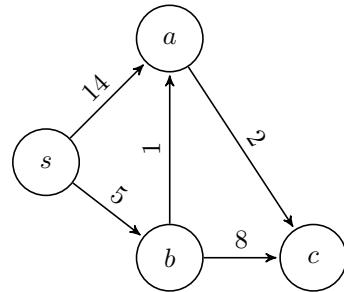


Figure 19. Grafo per gli esercizi 129 e 130.

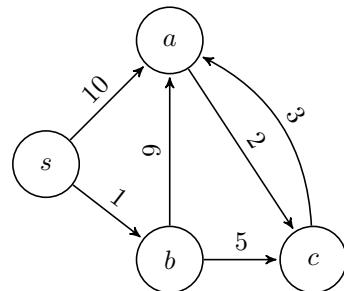


Figure 20. Grafo per l'esercizio 131.

- C) $c, 0$.
D) s, ∞ .

130) Si consideri il grafo diretto in Fig. 19. Nel caso peggiore, quante chiamate a *Relax* devono essere effettuate da *Bellman-Ford* prima che $c.d$ assuma il valore corretto, assumendo s come sorgente? E da *DAGShortestPath*?

- A) 8, 4.
B) 10, 6.
C) 12, 5.
D) 21, 7.

131) Si consideri il grafo diretto in Fig. 20. Qual è il valore di $a.d$ dopo l'esecuzione di *Dijkstra*?

- A) 10.
B) 9.
C) 7.
D) ∞ .

132) Si consideri il grafo diretto in Fig. 21. Come si possono distribuire i pesi 10, 10, 25, -50 sugli archi in maniera da garantire che *Dijkstra*, eseguito con sorgente s , fallisca?

- A) $w(s, c) = -50, w(c, b) = 25, w(s, a) = 10, w(a, b) = 10$.
B) $w(s, c) = 25, w(c, b) = -50, w(s, a) = 10, w(a, b) = 10$.
C) $w(s, c) = 25, w(c, b) = 10, w(s, a) = -50, w(a, b) = 10$.
D) *Dijkstra* fallisce sempre in presenza di un arco negativo.

133) Si consideri la soluzione all'esercizio precedente. Se mantenendo gli stessi pesi gli archi vengono resi indiretti, è vero che *MST-Prim* calcolerebbe un albero di copertura minima corretto sul grafo risultante?

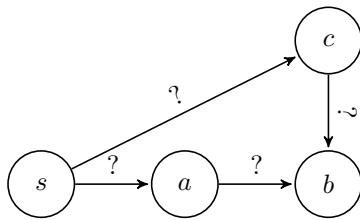


Figure 21. Grafo per l'esercizio 132.

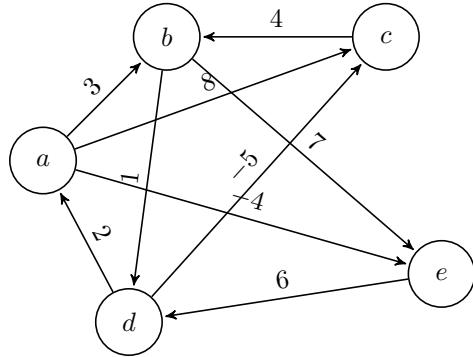


Figure 22. Grafo per l'esercizio 135.

- A) Si.
 B) No.
 C) No, ma esiste un algoritmo per il calcolo dell'albero di copertura minimi che, invece, funzionerebbe.
 D) Si, ma dovrebbe essere opportunamente modificato.
- 134) Si progetti un algoritmo per trovare calcolare il peso del percorso minimo, in un grafo diretto pesato senza archi negativi, da un certo vertice u ad un certo vertice v passando per un certo vertice t dato.
- 135) Si consideri il grafo diretto in Fig. 22. Durante l'esecuzione di *Floyd-Warshall*, qual è il valore di $D^3[e, c]$?
- A) 3.
 B) ∞ .
 C) 10.
 D) -2.
- 136) Si consideri il grafo diretto in Fig. 23. Durante l'esecuzione di *Floyd-Warshall*, qual è il valore di $D^5[d, a]$?
- A) 10.
 B) 5.
 C) 6.
 D) ∞ .

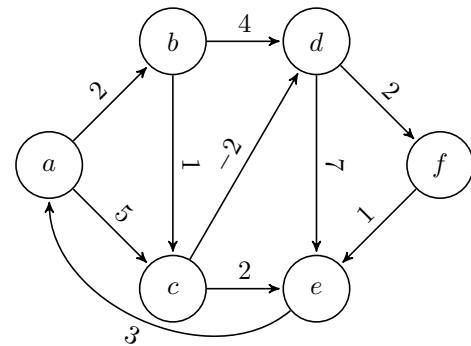


Figure 23. Grafo per l'esercizio 136.