



Algoritmi e strutture dati 21/22: esercizi per la preparazione all'esame

- 1) Qual è un modo possibile di definire il caso medio di *InsertionSort*?

- A) Ipotizzare che il ciclo interno non si esegua mai. C.M
 B) Ipotizzare che il ciclo interno si esegua sempre. C.P.
 C) Ipotizzare che il ciclo interno si esegua sempre, e per la metà dei passi possibili.
 D) Nessuna delle precedenti.

Ragionamento.

la soluzione A rappresenta il caso migliore
 la soluzione B (supponendo che intenda per tutti i possibili) rappresenta il caso peggiore

- 2) Diciamo che A è un array di interi è **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) $\Theta(k \cdot \log(k))$.

Ragionamento.

Il fatto che ci siano al massimo k posizioni di distanza, non toglie che si entri nel ciclo → non si può eliminare il quadrato

- 3) Qual è la complessità, nel caso peggiore, di *InsertionSort* quando A è quasi ordinato (come definito nell'esercizio 2) e k è $\frac{n}{2}$?

```
proc Parity (A, Key)
{ return RecParity(A, Key, 1, A.length)
```

```
proc RecParity (A, Key, i, j) key = 3
{ if (i = j) fin i = 0 j = 6
  then { if (A[i] = Key) approx 0.1
         then return false
         return true
         k = (i+j)/2 k = 3
         if (RecParity(A, Key, i, k) = RecParity(A, Key, k + 1, j))
         then return true
         return false
  }
```

A [7|3|4|3|8|4|3]

Figure 1. Codice per gli esercizi 5 e 6.

- A) $\Theta(n)$.
 B) $\Theta(k)$.
 C) $\Theta(n^2)$.
 D) $\Theta(k \cdot \log(k))$.

- 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) Nè stabile, nè in place.

- 5) Si consideri un array A 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 sbagliata ed è possibile fornire un controsenso. NO
 B) La procedura è corretta; un'invariante induttiva di *RecParity* è: *dopo ogni chiamata ricorsiva su A, k ha il valore del numero di occorrenze di Key in A*. NO
 C) La procedura è corretta; un'invariante induttiva di *RecParity* è: *al termine di ogni chiamata ricorsiva su A viene restituita la parità del numero di occorrenze di Key in A*.
 D) La procedura è sbagliata, ma si può correggere cambiando l'ultima istruzione.
- 6) Si consideri un array A 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). È vero che, se n è la cardinalità dell'array, la ricorrenza che determina la complessità dell'algoritmo

il cui pseudo codice si trova in Fig. 1 è:

- A) $T(n) = T(\frac{n}{2}) + 1$.
- B)** $T(n) = 2 \cdot T(\frac{n}{2}) + 1$.
- C) $T(n) = T(\frac{n}{2}) + \Theta(n)$.
- D) $T(n) = 2 \cdot T(\frac{n}{2}) + \Theta(n)$.

7) Considerare l'algoritmo *InsertionSort* e costruirne una versione ricorsiva. Quale sarebbe la ricorrenza che ne determina la complessità?

- A)** $T(n) = 2 \cdot T(n - 1) + 1$.
- B) $T(n) = 2 \cdot T(n - 1) + n$
- C) $T(n) = T(n - 1) + 1$.
- D) $T(n) = T(n - 1) + n$.

Pseudo codice.

for e while \rightarrow ricorrenze

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.

Dimostrazione.

Si dimostra: $aRb \rightarrow bRa$

$f(n) = O(g(n))$ No: se $f(n) = O(g(n))$ $g(n) \neq O(f(n))$
 Transitività: aRb $bRc \rightarrow aRc$
 $f(n) = O(g(n))$ $g(n) = O(h(n)) \rightarrow f(n) = O(h(n))$ Si
 Riflessività: aRa
 $f(n) = O(f(n))$ Si

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 $\Omega()$ soddisfa?

A) È simmetrica.

B) È transitiva. **PERCHÉ?** $\Omega() \neq \Omega()$ ↗

C) È transitiva e riflessiva.

D) È transitiva, riflessiva, e simmetrica.

11) 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.

12) Nel mostrare $3 \cdot n^2 + 5 \cdot n + 1 = \Theta(n^2)$ troviamo:

- A) $n > 1$, $c_1 = \frac{5}{4}$, e $c_2 = \frac{21}{4}$. No
- B) $n > 2$, $c_1 = \frac{3}{5}$, e $c_2 = \frac{21}{5}$. No
- C)** $n > 4$, $c_1 = \frac{3}{4}$, e $c_2 = \frac{21}{4}$. Sì
- D) $n > 3$, $c_1 = \frac{5}{5}$, e $c_2 = \frac{21}{5}$.

Dimostrazione.

$$\begin{array}{ll} C_1 n^2 < 3n^2 + 5n + 1 < C_2 n^2 \\ n=1 \quad g < q < c_2 \quad n=5 \quad 25g < 101 < 25c_2 \\ h=2 \quad 4c_1 < 13 < 4c_2 \\ n=3 \quad 9c_1 < 45 < 9c_2 \end{array}$$

13) 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) Si. Infatti, $6 \cdot n^3$ si comporta asintoticamente come n^2 .
- D) Si.

14) 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)** $n_0 = 10, c = 4$. $30 \leq 40$ ✓
- C) $n_0 = 1, c = 1$. $3 \leq 1$ ✗
- D) È vero per tutti gli $n \geq 0$ e tutte le costanti c .

15) È vero che $5 \cdot n + 6 = O(n^2)$?

- A)** No.
- B) Si. Questa affermazione è vera per $c = 1$ e per tutti gli $n > 0$.
- C) Si. Questa affermazione è vera per $c = 1$ e per tutti gli $n > 2$.

PERCHÉ?
non questo?

- D) Si. Questa affermazione è vera per $c = 1$ e per tutti gli $n \geq 6$. NO! Solo $-1 < n < 6$

12) Nella notazione $O()$, $\Omega()$, $\Theta()$: $\log_a b = c$ $a^c = b$

- A) La base dei logaritmi è importante in tutti i casi, ad esempio $\Theta(\log_2(n)) \neq \Theta(\log_3(n))$.
 - B) La base dei logaritmi è importante ma solo nella notazione $O()$. Si ha infatti, ad esempio, che $O(\log_2(n)) \neq O(\log_3(n))$.
 - C)** La base dei logaritmi non è importante. Si ha infatti, ad esempio, che $\Theta(\log_a(n)) = \Theta(\log_b(n))$ per qualsiasi $a \neq b$, con $a, b > 1$, e questo vale anche per le altre due notazioni.
 - D) La base dei logaritmi è importante ma solo nella notazione $\Omega()$. Si ha infatti, ad esempio, che $\Omega(\log_2(n)) \neq \Omega(\log_3(n))$.

17) Si trovi l'andamento asintotico della funzione $\log(n!)$.

Succede che: 15 ottobre

- A) $\log(n!) = O(n)$.
 - B) $\log(n!) = O(\sqrt{n})$.
 - C) $\log(n!) = \Theta(n \cdot \log(n))$.
 - D) Nessuna è corretta.

Dimostrazione.

$$\begin{aligned}
 O(\log(n!)) &= O(\lg(n \cdot n-1) \cdots 1) \\
 &\stackrel{1}{=} O(\lg(n) \cdot \lg(n-1) \cdots \lg(1)) \\
 &\stackrel{1}{=} O(\lg(n) \cdots \lg(n)) \\
 &\stackrel{1}{=} O(n \lg(n))
 \end{aligned}$$

- 18) 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.

Pseudo codice.

fun(A, m, M) {
 for i = 1 to A.length - 1 {
 if m > A[i] {
 m = A[i];
 } else if M < A[i] {
 M = A[i];
 }
 }
 }

 ↴ confronti massimi $2n - 2$

m = 5	1	2	3	4
N = 5	/	3	/	/
	7	/	/	/
	2	1	2	2

A $\frac{1}{5} \frac{2}{5} \frac{3}{5} \frac{4}{5}$

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

Idea.

- 20) Si consideri la ricorrenza:

$$T(n) = 4 \cdot T\left(\frac{n}{2}\right) + 3 \cdot n^2 - 17.$$

Quale delle seguenti è una soluzione?

- A) $\Theta(n^2 \cdot \lg(n))$.
B) $\Theta(n)$.
C) $\Theta(n^3)$.
D) Questa ricorrenza è irrisolvibile.

- 19) 21) 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) $T(n) = \Theta(\log(n))$.
 - C) $T(n) = \Theta(n)$.
 - D) Nessuna è corretta.

- 22) Si consideri la ricorrenza $T(n) = 2 \cdot T\left(\frac{n}{2}\right) + n^3$. Usando lo sviluppo, si ottiene:

- $$A) \ T(n) = \Theta(n^{\frac{3}{4}}).$$

Se $T(n) = aT(\frac{n}{b}) + f(n)$, allora:

$$T(n) = \begin{cases} \Theta(n^{g_b(a)}) & \text{se } f(n) = O(n^{g_b(a)} - \varepsilon) \text{ con } \varepsilon > 0 \\ \Theta(n^{g_b(a)} \lg^{k+1}(n)) & \text{se } f(n) = \Theta(n^{g_b(a)} \lg^k(n)) \text{ con } k \geq 0 \\ \Theta(f(n)) & \text{se } f(n) = \Omega(n^{g_b(a)+\varepsilon}) \text{ se } \exists c < 1 \text{ s.t. } af(\frac{n}{b}) \leq c \cdot f(n) \forall n > n_0 \end{cases}$$

20) $T(n) = 4T(\frac{n}{2}) + 3n^2 - 17$

$$a=4 \quad b=2 \quad f(n)=3n^2 - 17$$

$$f(n)=O(n^{g_2(4)-\varepsilon})? \quad \lg_2 4 \rightarrow 2^x=4$$

$$3n^2 - 17 = O(n^2 - \varepsilon)$$

$$\cdot f(n)=O(g(n)) \quad f(n) \leq cg(n)$$

Si

2) $f(n)=\Theta(n \lg_2^4 \lg^k(n))$ ponendo $k=0$

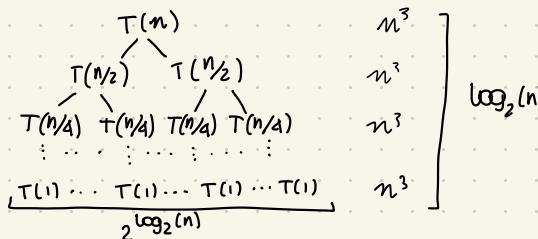
$$3n^2 - 17 = \Theta(n^2) \quad \text{Si}$$

3) $f(n)=\Omega(n^{g_2(4)+\varepsilon}) \quad \exists c < 1 \text{ s.t. } af(\frac{n}{b}) \leq c \cdot f(n)$

$$f(n)=\Omega(n^2+\varepsilon) \quad \text{No}$$

$$\Rightarrow T(n)=\Theta(n^2)$$

22) $T(n) = 2T(\frac{n}{e}) + n^3$

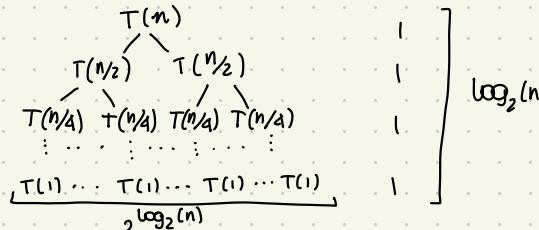


$$T(n) = n + n^3 \lg(n)$$

$$= \Theta(n^3 \lg(n))$$

21) $T(n) = 2T(\frac{n}{2}) + 1$

Svolgimento!



$$T(n) = \underbrace{2^{\log_2(n)}}_1 + 1 \cdot \log_2(n)$$

$$= n + \log_2(n)$$

$$= \Theta(n)$$

$$T(n) = T\left(\frac{3}{4}n\right) + n$$

$$= \frac{4}{3} \ln(n) + n \ln(n)$$

23) $T(n) = T\left(2^{\frac{n}{3}}\right) + 1 \quad \text{H.T.}$

$$a=1 \quad b=\frac{3}{2} \quad f(n)=2$$

$$n^0=1 = O(n^{\log_{3/2} 2 - \varepsilon}) \quad \frac{3}{2}^n = 1 \quad n=0$$

$$1 = O(n^{0-\varepsilon}) = O(n^{-\varepsilon}) \quad \text{No}$$

$$1 = \Theta(n^0 \lg^k(n)) \quad k=0 \Rightarrow 1 = \Theta(1) \quad \text{Si}$$

$$1 = \Omega(n^{0+\varepsilon}) = \Omega(n) \quad \text{No}$$

$$1 \cdot f\left(2^{\frac{n}{3}}\right) \leq c f(n) \quad \text{Eccoci?}$$

$$a=1 \quad b=\frac{4}{3} \quad f(n)=n \quad \log_b a=0$$

$$\textcircled{1} \quad f(n)=O(n^{0-\varepsilon}) \quad \text{No}$$

$$\textcircled{2} \quad f(n)=\Theta(n^0 \lg^k(n)) \quad \text{No}$$

$$\textcircled{3} \quad f(n)=\Omega(n^{0+\varepsilon}) \quad \text{Si}$$

$$T(n) = 4T(\frac{n}{2}) + n^2 \quad a=4 \quad b=2 \quad f(n)=n^2$$

$$\Theta(n^{\log_2 4} \lg^{k+1}(n)) \quad 2^n = 4 \quad n \geq 2$$

$$\Theta(n^2 \lg(n))$$

$$f(n)=\Theta(n^2) \quad \text{Si}$$

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

Dimostrazione.

$$\begin{aligned} T(n) &= n + n^3 \log_2(n) \\ &= \Theta(n^3 \log n) = \Theta(n^3) \end{aligned}$$

- 22) 23) Si consideri la ricorrenza $T(n) = T(\frac{2n}{3}) + 1$. 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.

- 24) Si consideri la ricorrenza $T(n) = 9 \cdot T(\frac{n}{3}) + n$. Usando lo sviluppo, si ottiene:

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

Dimostrazione.

$$\begin{aligned} T(n) &= 3^{\log_3 n} + n \log_3 n \\ &\text{Base uguali esponente diverso!} \\ &3^{\log_3 n} \cdot 3^{\log_3 n} + n \log_3 n \\ &n \cdot n + n \log_3 n \\ &\textcircled{m^2} + n \log_3 n \end{aligned}$$

26)

- Si consideri la seguente dimostrazione per sostituzione della ricorrenza $T(n) = 2 \cdot T(\frac{n}{2}) + 1$. Vogliamo mostrare che $T(n) \leq c \cdot \log(n)$ per qualche c , e lo facciamo con i seguenti passaggi:

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

Possiamo dire che:

- A) La dimostrazione è corretta.
B) La dimostrazione è sbagliata.

- C) La dimostrazione è corretta, e $T(n) = O(\log(n))$.
D) La dimostrazione è sbagliata, ma $T(n) = O(\log(n))$.

28)

- Nel testo usato a lezione (il 'Cormen') appare un errore nell'esercizio 4.3-8. Si chiede di considerare la ricorrenza $T(n) = 4 \cdot T(\frac{n}{2}) + n^2$, 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, $T(n) = \Theta(n^2 \cdot \log(n))$; possiamo mostrare che, mentre assumere $T(n) \leq c \cdot n^2 \cdot \log(n)$ non funziona, la sostituzione funziona con $T(n) \leq c \cdot n^2 \cdot \log(n) - n$.
C) Con il Master Theorem vediamo che, diversamente da quanto indicato, $T(n) \neq \Theta(n \cdot \log(n))$; possiamo mostrare che, mentre assumere $T(n) \leq c \cdot n \cdot \log(n)$ non funziona, la sostituzione funziona con $T(n) \leq c \cdot n \cdot \log(n) - n$.
D) Nessuna delle altre è corretta.

- 27) Si consideri la ricorrenza $T(n) = 5 \cdot T(\frac{n}{3}) + n^2$. È vero che:

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

Dimostrazione. $a=5$ $b=3$ $f(n) = n^2$ $\log_3 5 = 1,46$

$$n^2 = \Theta(n^{1.5-\varepsilon}) \text{ NO}$$

$$n^2 = \Theta(n^{1.5} \log_2^k(n)) \quad 2^k = n$$

$$\begin{aligned} n^2 &= \Omega(n^{1.5+\varepsilon}) \quad \text{se } \varepsilon = 0.5 \\ &\stackrel{!}{=} \Omega(n^2) \quad \checkmark \end{aligned}$$

27)

- 28) Si consideri la seguente dimostrazione per sostituzione della ricorrenza $T(n) = T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + 1$. Vogliamo mostrare che $T(n) \leq c \cdot n - d$ per qualche c, d :

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

Possiamo dire che:

- A) La dimostrazione è errata perché $T(n) = o(n)$.
B) La dimostrazione è corretta e $T(n) = O(n^2)$ NO.
C) La dimostrazione è errata, ma è vero che $T(n) = O(n)$.
D) La dimostrazione è corretta e $T(n) = O(n)$.

- 29) Consideriamo le ricorrenze $T(n) = 7 \cdot T(\frac{n}{2}) + n^2$ e $T'(n) = a \cdot T'(\frac{n}{4}) + n^2$. Ci domandiamo: qual è il più

alto valore intero di a per il quale l'algoritmo che ha complessità $T'(n)$ è asintoticamente più veloce di quello con complessità $T(n)$?

- A) 46.
- B) 47.
- C) 48.**
- D) 149.

Dimostrazione.

$$\begin{aligned}
 T(n) &= 7 \left(\frac{n}{2}\right) + n^2 \quad a=7 \quad b=2 \quad f(n) = n^2 \quad \log_2 7 = 2.8 \\
 n^2 &= O(n^{2.8-\varepsilon}) \quad \text{se} \\
 \text{caso: } T(n) &= \Theta(n^{2.8}) \\
 T'(n) &= a T'\left(\frac{n}{4}\right) + n^2 \quad a=b=4 \quad n^2 \quad \log_4 a \\
 \log_2(2) &= 2.00 \\
 \log_4(4) &= 2.00 \\
 \log_4(3) &= 2.76 \\
 \log_4(2) &= 2.77 \\
 \log_4(1.5) &= 2.79 \\
 \rightarrow \log_4(a) &> \log_2(7) \\
 \text{non capisco perché } <, \text{ due} \\
 \text{cresce più velocemente} \\
 \text{dovrebbe essere giusto} >
 \end{aligned}$$

- 30) 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) $\Theta(n \cdot \log(n))$ in tutti i casi.
- D) $\Theta(n^2)$ nel caso peggiore, $\Theta(n)$ nel caso medio.

- 31) 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.

- 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 spazzamenti, 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 spazzamenti). Il problema di trovare il minimo elemento in un array di questo tipo ha complessità, nel caso peggiore:

- A) $O(\log(n))$.
- B) $\Theta(n)$.**

C) $\Theta(n \cdot \log(n))$.

D) $\Theta(n^2)$.

Pseudo codice.

```

for (i = 0 ; i < A.length, i++)
  { if (A[i+1] < A[i])
    { max = A[i+1];
      break
    }
  }
  
```

- 34) Chi, tra *MergeSort* e *InsertionSort*, si comporterebbe meglio sul problema di ordinare un array di n posizioni quasi ordinato (come definito nell'esercizio 2), con k costante?

- A) *MergeSort*: chiaramente $\Theta(n \cdot \log(n))$ è sempre meglio di $\Theta(n^2)$.
- B) *MergeSort*: chiaramente $\Theta(n \cdot \log(n))$ 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? $3T\left(\frac{n}{3}\right) + \Theta(n) \quad a=3 \quad b=3 \quad f(n) = \Theta(n) \quad \log_3 3 = 1$

- A) In maniera identica.**
- B) La nuova versione, nel caso peggiore, ha complessità $T(n) = \Theta(n \cdot \log^3(n))$, pertanto si comporta in modo peggiore.
- C) La nuova versione, nel caso peggiore, ha complessità $T(n) = \Theta(n \cdot \log_3(n))$, 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) Sì.**
- 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 $O(n \cdot \log(n))$.

Idea.

- 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*? $i = p - 1 = \underline{\hspace{2cm} 2234}$

- A) x .
B) $r + 1$.
C) r .
D) p .

$R=5 \quad P=1$
 $J=1$
 $J=2$
 $J=3$
 $J=4$

RETURN S

- 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) $O(\log(n))$.
 - B) $\Theta(n^2)$.
 - C) $\Theta(n \cdot \log(n))$.
 - D) $O(n)$.

Pseudo codice.

Merge ugualate mo
 if ($j \leq n_2$)
 {; f ([i] $\leftarrow R[j]$)
 {man = $R[j]$

Dimostrazione.

Idea: trovo il 1° elemento con
e lo uso come PILOT in partita
che ha complessità $\Theta(n)$

Ma essendo solo O e 1
partition O(n)

solo $O(1)$
 $\rightarrow O(k)$ $k \leq n!$
 $C_{\text{Medio}} \xrightarrow{\text{constante}}$
 $O(n) \in C.p$

- 39) Applicare $\text{Partition}(A, 1, 5)$ all'array $A = [7, 1, 4, 5, 3]$. Qual è il valore restituito (cioè l'indice del pivot)?

- (A) 2.
B) 1.
C) 3.

- D) Dipende dall'esecuzione.

```

A[4,1,3,5,2] A[1,3,4,5,7]
Partition(A,1,5) j=3
i=3 j=1 A[3] ≤ 3? NO
for(O,4) j=4 A[4] ≤ 3? NO
j=1 A[1] ≤ 3? NO
A[1] ≤ 3? SI SWAP i+1 R
j=2 A[2] ≤ 3? SI 2 5
i=1 SWAP i,j RETURN 3
SWAP i,j

```

- 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.

- 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 perchè si tratta di una chiamata casuale.

- D) $O(\log(n))$.

- 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) $\Theta(n \cdot \log(n))$.
- C) $O(n \cdot \log(n))$ nel caso peggiore.
- D) $O(n)$** nel caso medio.

Pseudo codice.

```

PARTITION
ma return i
  
```

- 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 (come definito nell'esercizio 2): 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) No. Sotto queste ipotesi, l'unico modo di eseguire l'ordinamento è usando un metodo elementare, e pertanto, la complessità del problema è $\Omega(n^2)$.
- B) No. Dobbiamo ordinarlo con un metodo basato sui confronti e pertanto il limite minimo è $O(n \cdot \log(n))$.
- C) Sì. Possiamo ordinarlo con un metodo basato sui confronti e pertanto il limite minimo è $O(3 \cdot \log(3))$ (ci sono solo tre elementi diversi); $O(3 \cdot \log(3))$ è $O(1)$ e quindi è lineare in n .
- D) Sì.** 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.

Dimostrazione.

- 48) Sia A un array di n interi positivi tale che ogni elemento di A è minore o uguale a $3 \cdot n$. Possiamo mostrare che il problema di contare quanti elementi di A si trovano nell'intervallo $[k_1, k_2]$, con $k_1 \leq k_2$, ha complessità:

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

- 49) Sia A un array di n interi positivi tale che ogni elemento di A è minore o uguale a $5 \cdot n$, e k un intero positivo dispari. 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) $\Theta(n \cdot \log(n))$.** a spazio
- C) $O(n)$.**
- D) $O(\log(n))$.

Pseudo codice.

```

① ordino
② uso due puntatori che si trovano in mezzo
    → O(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 la terza tripla nell'ordinamento?

- | | | | | |
|------------------|-------|-------|-------|-------|
| A) 7-0-3. | 4 2 3 | 6 2 1 | 7 0 3 | 1 4 4 |
| B) 9-4-2. | 6 2 1 | 9 4 2 | 6 2 1 | 4 2 3 |
| C) 6-2-1. | 7 0 3 | 4 2 3 | 4 2 3 | 6 2 1 |
| D) 4-2-3. | 1 4 4 | 7 0 3 | 9 4 2 | 7 0 3 |
| | 9 4 2 | 1 4 4 | 1 4 4 | 9 4 2 |

9 - 5, 7, 12, 10, 8

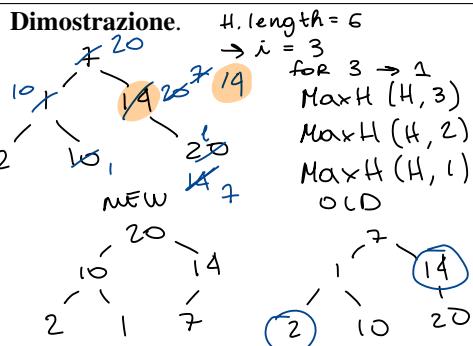
l'altezza è il numero max di archi tra radice e foglia

- 58) 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, perché se H è di altezza 3 deve contenere almeno 8 elementi.
 B) No, perché l'ordine non rispetta la proprietà.
 C) No, perché l'albero corrispondente non sarebbe completo.
 D) Sì.

- 59) 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.



- 60) Si considerino k array di numeri interi A_1, \dots, A_k , tutti ordinati, tali che il numero di 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) $O(k \cdot \log(k))$.
 B) $O(n \cdot \log(k))$.
 C) $O(k \cdot \log(n))$.
 D) $O(n \cdot \log(n))$.

Dimostrazione.

Qual è la complessità del problema di produrre un array B che contiene tutti gli n numeri ordinati?

- A) $O(k \cdot \log(k))$.
 B) $O(n \cdot \log(k))$.
 C) $O(k \cdot \log(n))$.
 D) $O(n \cdot \log(n))$.

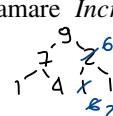
Dimostrazione.

- 62) Si consideri una min-heap. È noto che l'estrazione del minimo costa $O(\log(n))$, dove n è il numero di elementi della heap. Quanto costa l'estrazione del massimo da una min-heap?

- A) $\Theta(n^2)$.
 B) $O(\log(n))$, applicando lo stesso algoritmo per l'estrazione del minimo.
 C) $O(\log(n))$.
 D) $O(n)$.

- 63) Supponiamo che $H = [9, 7, 2, 1, 4, 6, 1]$ 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) 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) $O(\log(n)^2)$.
 B) $O(k + (n - k) \cdot \log(k))$ nel caso peggiore.

- 61) 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.

- C) $O(\log(n))$ nel caso peggiore.
 D) Non si può usare una max-heap per risolvere questo problema.

Dimostrazione.

- 66) 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) $O(n + k \cdot \log(n))$ nel caso peggiore.
 C) $O(k \cdot \log(n))$ nel caso peggiore.
 D) Non si può usare una min-heap per risolvere questo problema.

Dimostrazione.

- 67) 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? $M=9$ $h(k)=k \bmod 9 + 1$
 A) 28, 10, 19. $k=5 \quad 5+1=6$
 B) 10, 28. $k=28 \quad 1+1=2$
 C) 28, 10, 15
 D) 10, 19, 28.
- 68) 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. 128 numero primo
 B) 253. 256 lontano da 2^p
 C) 257.
 D) 500. 512
 Il minimo numero primo pari è 2

- 69) 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?

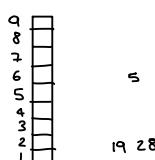
$$h(k) = \lfloor m \cdot (k \cdot A - \lfloor k \cdot A \rfloor) \rfloor + 1$$

 A) La 1. 0.37 $\rightarrow 0.17 \rightarrow 2$ 2.56 . 2
 B) La 4. 12 6 1 41 4
 C) La 7. 124 62 2
 D) La 1 e la 2. 3.47 1.35 6

- 70) 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.

- 71) Sia A un array di n interi positivi tale che ogni elemento di A è minore o uguale a n^3 , e k un intero positivo dispari. Possiamo trovare un algoritmo che permette di stabilire se esistono $i, j \geq 1$, $i \neq j$, tali che $A[i] + A[j] = k$ che ha complessità, nel caso medio:
 A) $\Theta(n^2)$.
 B) $\Theta(n \cdot \log(n))$.
 C) $O(n)$.
 D) $O(\log(n))$.

Dimostrazione.



- 72) 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) $\Theta(\log(n))$.
- (C) $\Theta(\log^2(n))$.
- (D) $\Theta(n)$.



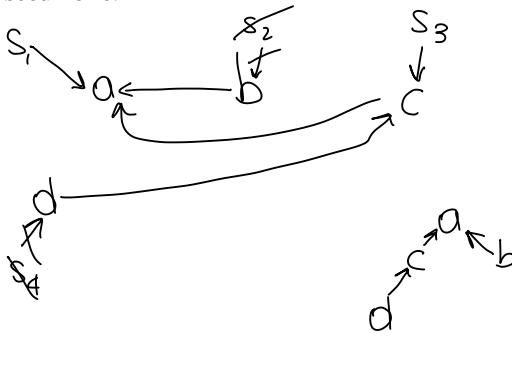
73) In una foresta \mathcal{S} di alberi k -ari che rappresentano insiemi disgiunti, supponiamo che $S_1 = \{2, 3, 7\}$, dove $2.p = 2$, $3.p = 2$, e $7.p = 3$, e che $S_2 = \{4, 8\}$, dove $4.p = 4$ e $8.p = 4$. Quali sono i minimi ranghi possibili di 2 e di 4, e cosa accade, in questo caso, eseguendo *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.

Esecuzione.



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.

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) 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.

77) 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.

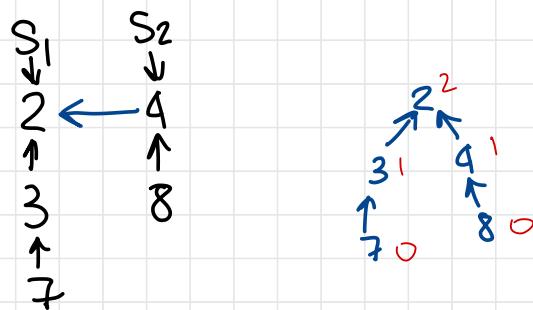
78) Nella rappresentazione degli insiemi disgiunti con foreste di alberi k -ari, con unione per rango e compressione del percorso, quale delle seguenti è vera?

- 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.

79) 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.

- 76) 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 $\text{Union}(3, 8)$, 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.

Ragionamento.

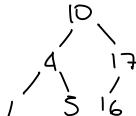
- 80) In base al ragionamento utilizzato nell'esercizio precedente, qual è l'equazione di ricorrenza che descrive la complessità del caso peggiore di una procedura ricorsiva che ricostruisce un albero a partire dalla sua visita in-order e pre-order?

- 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) $T(n) = T(\frac{n}{2}) + O(n^2)$.

- 81) In questo esercizio diciamo che un albero binario con n chiavi (non necessariamente di ricerca) è **bilanciato** se per ogni nodo, la differenza massima tra le lunghezze di due percorsi semplici qualsiasi da esso ad una foglia è al massimo uno. Una semplice visita permette di stabilire se un albero binario è bilanciato secondo questa definizione. Qual è, però, lo spazio minimo necessario per risolvere questo problema (escludendo lo spazio necessario alla ricorsione)?

- A) $O(1)$.
- B) $O(n)$.
- C) $O(n^2)$.
- D) $O(\log(n))$.

Dimostrazione.

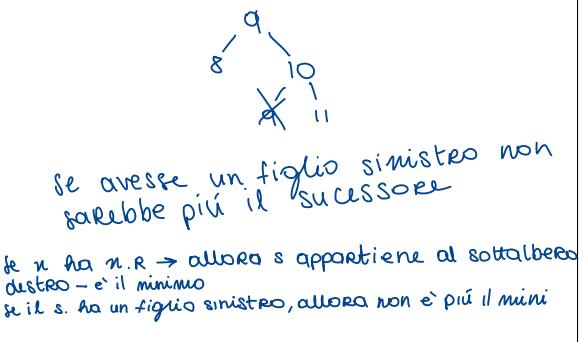


- A) Gli alberi utilizzati come base per insiemi disgiunti sono sempre k -ari, mentre quelli utilizzati come struttura dati classica sono sempre binari.
B) Non c'è alcuna differenza.
C) Gli alberi utilizzati come base per insiemi disgiunti non hanno puntatori ai figli; quindi, non possono essere visitati, 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.

- 83) 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.

Dimostrazione.



- 84) 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) Si può avere di altezza 2, ma non di altezza 6.
- B) Si 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.

- 85) 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.

- 86) Considerando la nostra implementazione di *BST-TreeDelete*, 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

- 82) 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?

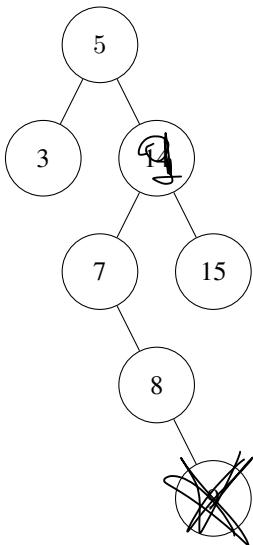
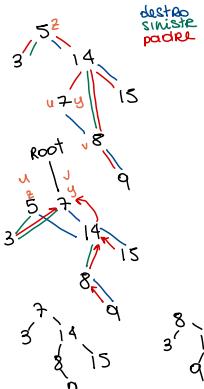


Figure 2. Codice per l'esercizio 87.

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) Sì.

Dimostrazione. \checkmark



- 87) Considerando l'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:

- 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.

- 88) Si supponga che T sia un albero binario di ricerca con chiavi intere, e che A sia un array di interi ordinato, e

```
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 88.

```
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 89.

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 ($i < A.length$).
- C) ($x \neq \text{nil}$) or ($i \leq A.length$).
- D) ($x \neq \text{nil}$) and ($i \leq A.length$).

- 89) 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 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.

- 90) 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) $x.key \neq y.key$.

 \checkmark

- 91) Si supponga che T sia un albero qualsiasi con chiavi intere, e che Cl sia una variabile globale condivisa da

```

proc StillBST( $T, x$ )
   $y = \text{TreeSuccessor}(T, x)$ 
  if ( $y = \text{nil}$ )
    then return true
  if (*)
    then return true
  return false

```

Figure 5. Codice per gli esercizio 99.

```

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 91.

tutte le chiamate di *PLS* e inizializzata a zero, il cui codice è mostrato in Fig. 6, e si supponga di chiamare $PLS(x.root, 1)$. 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.

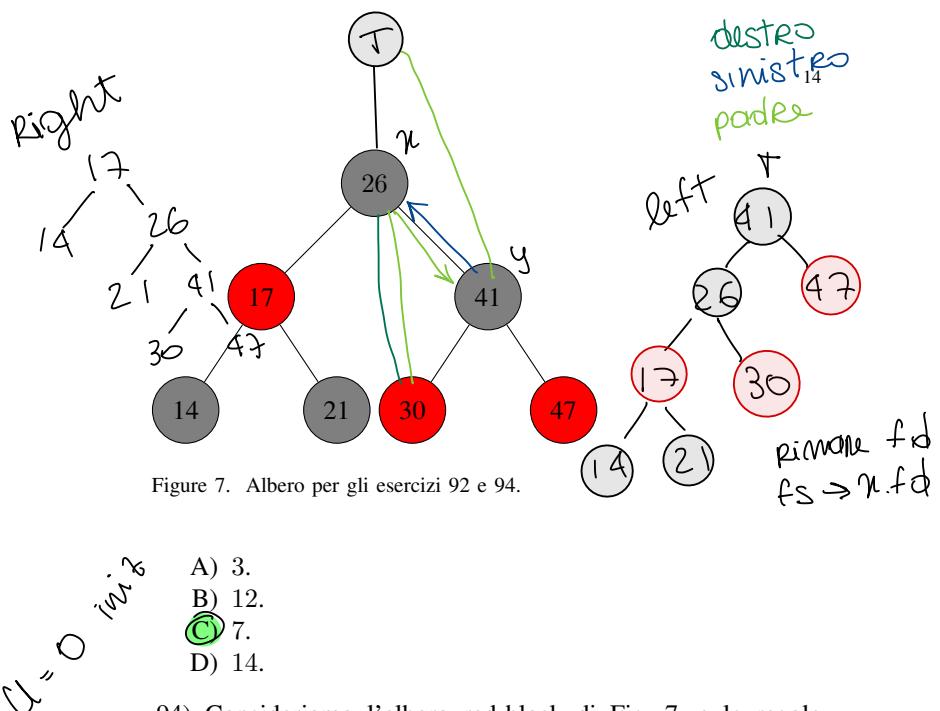
C) Quello di stampare tutte e sole le chiavi che sono presenti sul “costato sinistro” di T , cioè tutte e sole quelle che si trovano su nodi non nascosti da nodi allo stesso livello e più a sinistra.

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, sulla foglia del ramo stesso.

92) 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.

93) 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.

94) 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?

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

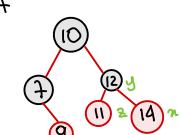
95) Si consideri il codice visto in classe per l'inserimento di una nuova chiave in un albero red-black T . Supponiamo adesso che T sia un albero red-black inizialmente vuoto, e di inserire, nell'ordine: 10,7,14,9,11,12. Dopo tutti gli inserimenti, che chiave ha il figlio destro della radice e di che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

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

Albero risultante

12 7 14 9 11 12

Tenot



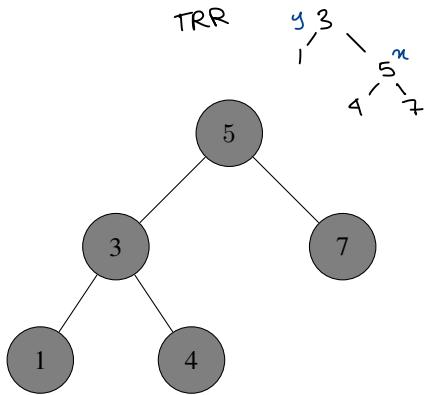


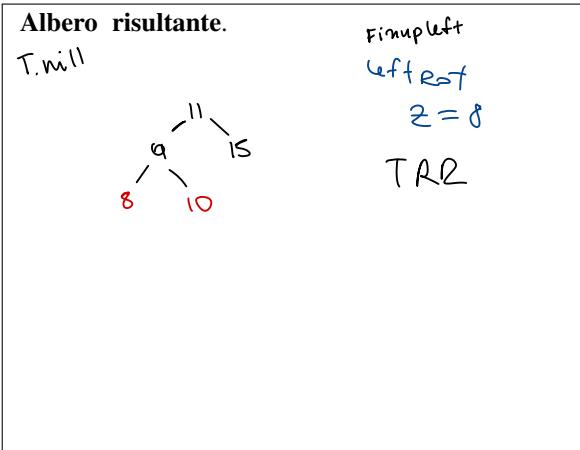
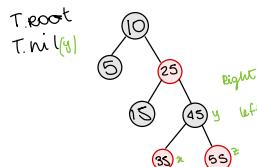
Figure 8. Albero binario di ricerca per l'esercizio 99.

- 96) 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.

- 97) 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.



- 98) 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) $\Omega(n^2)$, sia la ricerca del successore che quella del predecessore è $\Theta(\log(n))$
- B) $O(\log(n))$. quella del predecessore è $\Theta(\log(n))$
- C) $O(n)$. TreeInorder è $\Theta(n) \Rightarrow \Theta(n \log(n))$
- (D) $\Theta(n \cdot \log(n))$.

- 99) 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.

- 100) Si consideri un albero B T con tutti i nodi pieni, $t = 50$, e $h = 2$. Quante chiavi contiene? Si è in radice a 2($t-1$) nodi altri min 2 nodi in $h=2$ min 2 t nodi in $h=2$ ogni nodo con $t > 0$ ha min $t-1$ chiavi ($2^{t-1} - 1$) + $2t(2^{t-1}) + 4t^2(2^{t-1}) = n_k$ $2^{t-1} + 4t^2 - 2t + 8t^2 - 4t^2 = n_k$ $n_k = 8t^3 - 1 = 10000000 - 1 = 999999$

- A) 100000. $R \leq \log_{50} (n+1)$
- B) 500000. $50 \leq \frac{n+1}{2} \rightarrow 50^2 \leq \frac{n+1}{2}$
- C) 990000. $5000 = n+1 \quad n \geq 4999$
- (D) 999999. $n_k = 8t^3 - 1$



- 101) 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? ogni nodo ha min 3 figli

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

Dimostrazione.

Ogni n massimo $2t-1 = 5$ chiavi

1 2 3 4 5

Ogni n (tranne R) minimo $3-1 = 2$ chiavi

$2 \leq k \leq 5$

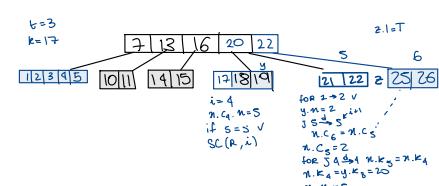


- 102) Qual è la complessità in termini di CPU dell'operazione di ricerca di una chiave in un albero B se, invece della ricerca lineare sul nodo, usiamo la ricerca binaria? nel senso BST?

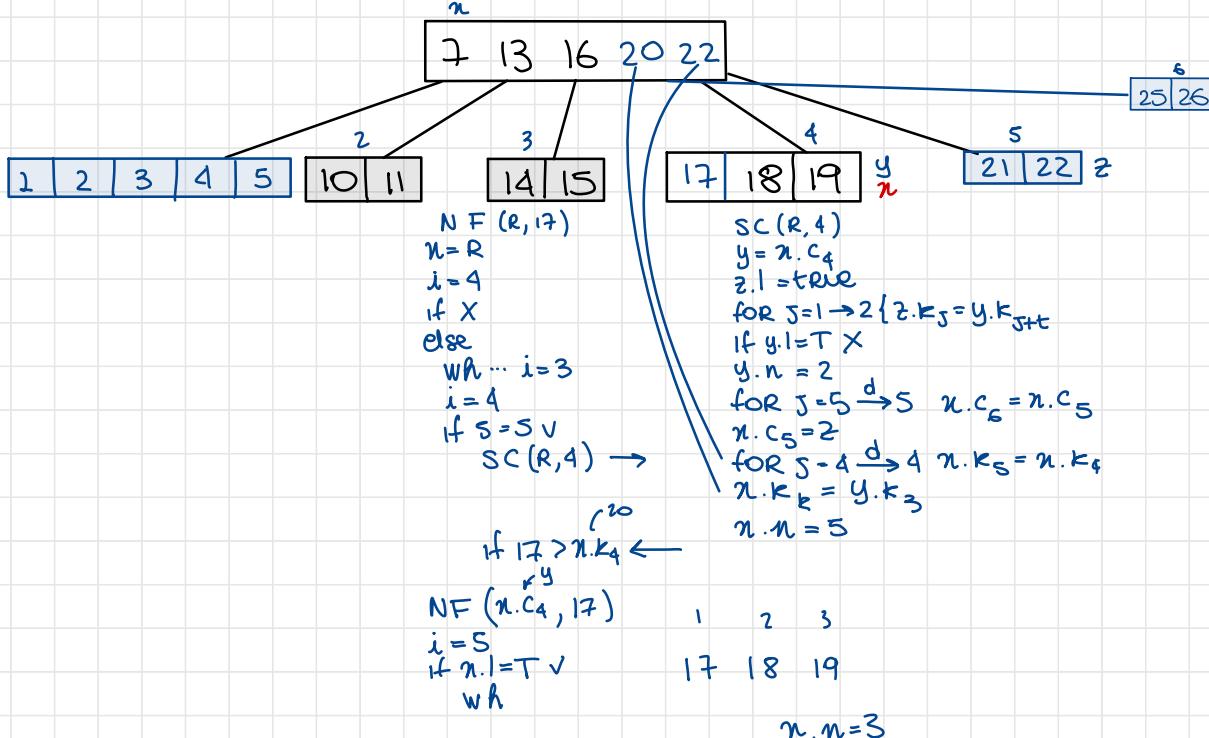
- A) la complessità non cambia.
- B) $O(t \cdot \log(\log(n)))$.
- C) $O(\log_t(n))$.
- (D) $O(\log(n))$.

- 103) 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 *BTreeInsert(T, 2)* e *BTreeInsert(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.
- C) 4. 1, 17, 18, 19, 21.
- (D) 5. 1, 17, 18, 19.



- 103) 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 $BTreeInsert(T, 2)$ e $BTreeInsert(T, 17)$, quante chiavi ha la radice? Quanto è alto T ? Quali sono le chiavi di $T.root.c_4$??



~~105~~ ~~106~~ ~~107~~ ~~108~~

porta con la sorgente nulla coda $\leftarrow Q = \emptyset$; Enqueue($a, 1$)
poi si estraia $u = \text{Dequeue}(Q)$
e inserisca tutti i vertici a lui collegati \leftarrow for $v \in A[\text{adj}[u]] \dots \text{Enqueue}$

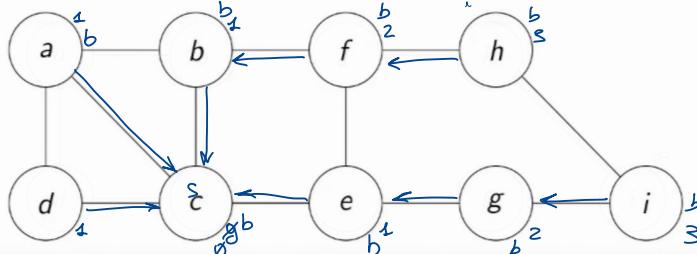


Figure 9. Grafo per gli esercizi 105 e 106.

Dimostrazione.

- 104) 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 $BTreeSplitChild(x, 2)$, 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 .

- 105) 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 .

- 106) 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) $Q = [a, b, e]$.

cabdefghi

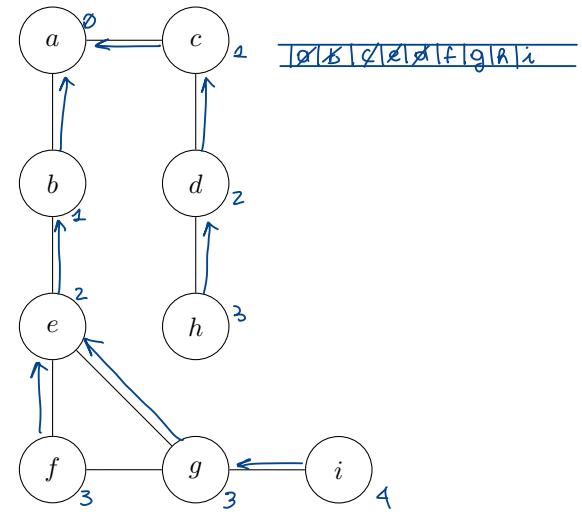


Figure 10. Grafo per gli esercizi 107 e 112.

$$s = a \quad f = b$$

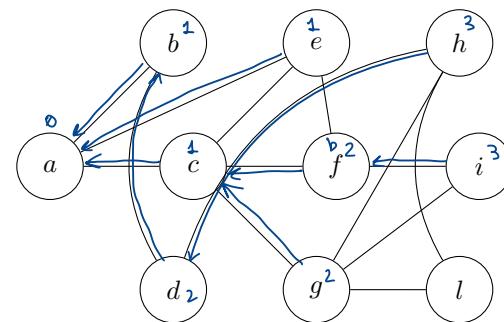


Figure 11. Grafo per l'esercizio 108.

~~105~~ ~~106~~ ~~107~~ ~~108~~

- 107) 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.

- 108) 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 $\text{BreadthFirstSearch}(G, s)$ nel momento in cui f diventa nero?

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

~~105~~ ~~106~~ ~~107~~ ~~108~~

- 109) 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.

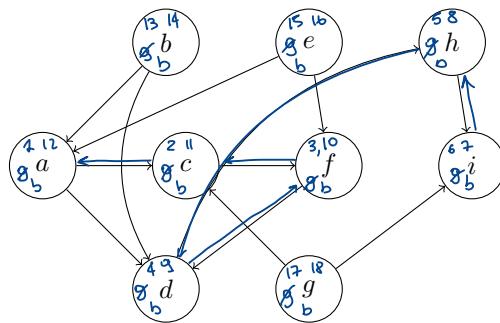


Figure 12. Grafo per gli esercizi 110 e 111.

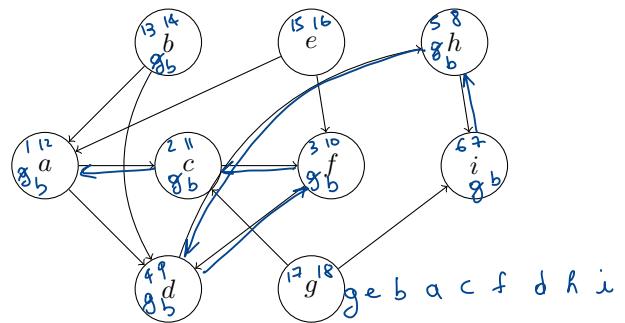


Figure 13. Grafo per l'esercizio 114.

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

Idea.

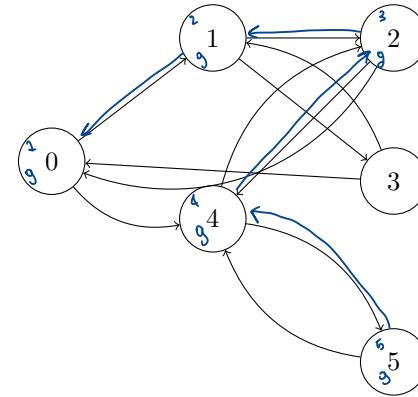
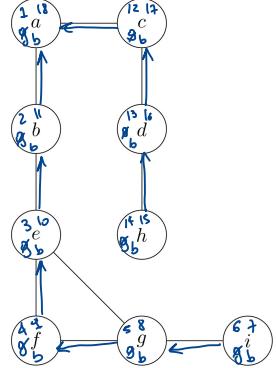


Figure 14. Grafo per l'esercizio 115.

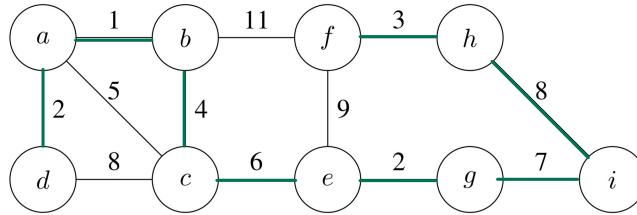
- 110) Si esegua *DepthFirstSearch* sul grafo di Fig. 12, assumendo che la prima sorgente sia *a* e che i nodi vengano sempre estratti in ordine alfabetico e si dica, dopo la visita, qual è l'intervallo inizio scoperta - fine scoperta del vertice *d*.
- A) [17, 18].
 B) [3, 4].
 C) [4, 12]
 D) [4, 9].
- 111) 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 il vertice scoperto per ultimo.
- A) *g*.
 B) *h*.
 C) *e*
 D) *b*.
- 112) 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. *non si guarda il time!*
- A) 7.
 B) 8.
- 113) 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 $[v.d, v.f]$.
- 114) 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*.
- 115) 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. *perché è aclico*
- 116) Si consideri la seguente affermazione: in ogni grafo diretto aciclico, l'ordinamento topologico è unico. Questa affermazione:

NON

112

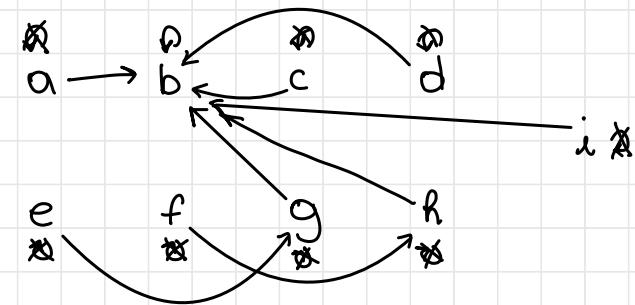


128



PRUSKAL :

a	b	1
a	d	2
e	g	1
f	i	3
b	c	4
a	c	5
c	e	6
g	i	7
c	d	8
f	i	8
e	f	9



COSTO MST: 33

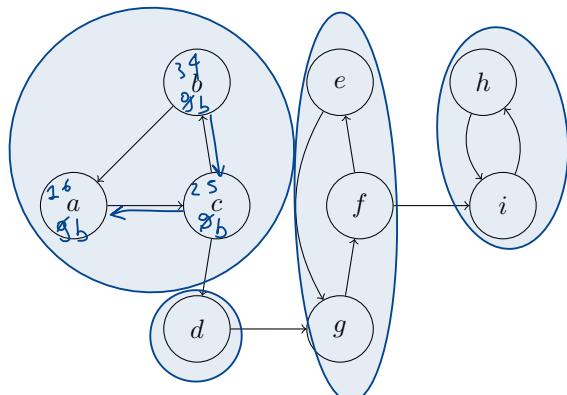


Figure 15. Grafo diretto per l'esercizio 118.

- (A) É vera. Possiamo dimostrarlo per induzione sulla cardinalità dell'insieme E degli archi.
 (B) É vera. Possiamo dimostrarlo per induzione sulla cardinalità dell'insieme V dei vertici. **a e b unici**, nessun altro
 (C) É falsa. Per dimostrarlo, è sufficiente un contrempio.
 (D) É falsa. Per dimostrarlo, ragioniamo per induzione sul numero di passi durante una visita in profondità.
- 117) 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, sia 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.

Idea.

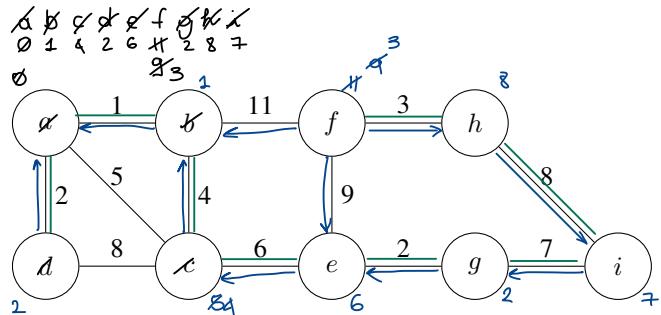


Figure 16. Grafo per gli esercizi 121 e 128.

- 119) 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.
- 120) 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. ✓

Dimostrazione.

- 118) Quante sono le componenti fortemente connesse del grafo in Fig. 15?

- A) 1. **a, c, b**
 B) 2. **d, f, e**
 (C) 3. **i, h**
 (D) 4. **g**

connesso da solo!
 - devono essere
 tutti gli elementi di G
 nella lista di SCC

- 121) 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.

- 122) In MST-Prim, l'istruzione $v.key = W(u, v)$ è, in realtà, una istruzione di?

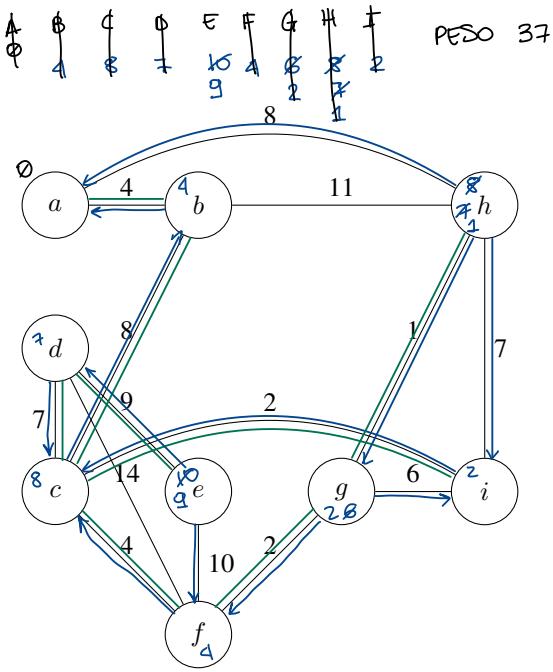


Figure 17. Grafo per l'esercizio 123.

a b f g
b c h g
c d i u
d e t
e f

- A) IncreaseKey.
B) Enqueue.
C) Dequeue.
D) DecreaseKey.

123) 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?

- A) (a, h).
B) (c, i). ✓
C) (a, b). ✓
D) (c, d). ✓

124) Supponiamo che G sia un grafo indiretto pesato in cui gli archi possono avere peso negativo. E' possibile utilizzare *MST-Prim* per calcolarne un albero di copertura minimo?

- A) No. La dimostrazione di correttezza fallisce in questo caso, e il risultato potrebbe non essere corretto.
B) No. La definizione di albero di copertura minimo perde di senso in caso di archi di peso negativo.
C) Si.
D) Si, ma la complessità risulterebbe asintoticamente peggiore.

125) 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: $\Theta(|V|)$ (per l'inizializzazione), $\Theta(|V|)$ (per la 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: $\Theta(|V|)$ (per l'inizializzazione), $\Theta(|V|)$ (per la costruzione della coda), $\Theta(|E|)$ (per le estrazioni del minimo, sotto l'ipotesi di grafi sparsi), e $\Theta(|E|)$ (per i decrementi,

analisi ammortizzata). Il costo totale è $\Theta(|E|)$.

- C) I vari elementi dell'algoritmo sommano così: $\Theta(|V|)$ (per l'inizializzazione), $\Theta(|V|^2)$ (per la 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: $\Theta(|V|)$ (per l'inizializzazione), $\Theta(|V|)$ (per la 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 è ancora $\Theta(|V|^2)$, ma con costanti nascoste inferiori al caso di grafi densi.

126) 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: $\Theta(|V|)$ (per l'inizializzazione), $\Theta(|V|)$ (per la costruzione della coda), $\Theta(|V| \cdot \log(|V|))$ (per le estrazioni del minimo), e $\Theta(|E| \cdot \log(|V|)) = \Theta(|V|^2 \cdot \log(|V|))$ (per i decrementi, analisi ammortizzata), per un totale di $\Theta(|V|^2 \cdot \log(|V|))$.
B) I vari elementi dell'algoritmo sommano così: $\Theta(|V|)$ (per l'inizializzazione), $\Theta(|V|)$ (per la costruzione della coda), $\Theta(|V| \cdot \log(|V|))$ (per le estrazioni del minimo), e $\Theta(|E| \cdot \log(|E|)) = \Theta(|V| \cdot \log(|E|))$ (per i decrementi, analisi ammortizzata), per un totale di $\Theta(|V| \cdot \log(|E|))$.
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.

127) 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. E' sufficiente invertire il senso di tutte le disugualanze.
D) Si. *MST-Prim* calcola l'albero di copertura massimo senza che sia necessario alcun cambio; *MST-Kruskal*, invece, no.

128) 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,f). suo lto a pg. 22
B) (e,g).
C) (c,d).
D) (h,i).

a b ①
a d ②
e g ③
f h ④
b c ⑤
c e ⑥
g i ⑦
h j ⑧
e f ⑨

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

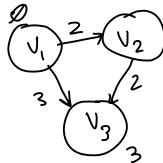
130) Si supponga che G sia un grafo indiretto, pesato, e che tutti i pesi siano interi tra 1 e un numero $O(|E|)$. Possiamo implementare MST-Kruskal in modo che, sotto queste ipotesi, la sua complessità per grafi sparsi sia asintoticamente migliore di quella calcolata senza ipotesi sul grafo?

- A) Sì. Utilizzando foreste di alberi per implementare insiemi disgiunti, la complessità può abbassarsi a $O(\log(|E|))$.
- 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 sbagliate.

Idea.

131) Sia G un grafo diretto pesato, e siano $v_1, v_2, v_3 \in G.V$, con $W_{12} = 2, W_{23} = 2, W_{13} = 3$. 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.



132) Si consideri il grafo diretto in Fig. 18. Qual è il valore di $c.\pi$ dopo l'esecuzione di Bellman-Ford? E qual è il

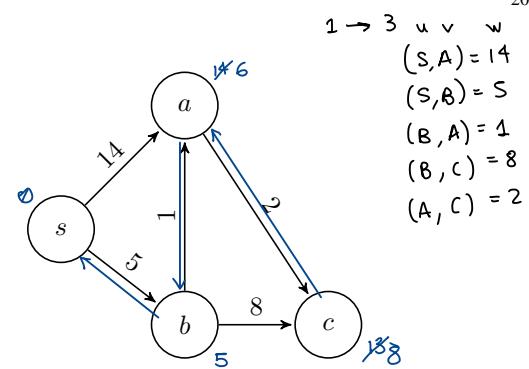


Figure 18. Grafo per gli esercizi 132 e 133.

S B C A

S ~~0~~

A ~~∞~~ \triangleright 9

B ~~∞~~ 1

C ~~∞~~ 6

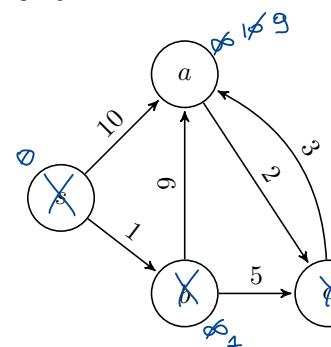


Figure 19. Grafo per l'esercizio 134.

valore del suo (del padre di c) attributo $.d$, assumendo s come sorgente?

- A) $b, 6$.
- B)** $a, 6$.
- C) $c, 0$.
- D) s, ∞ .

133) Si consideri il grafo diretto in Fig. 18. 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.

Dimostrazione.

134) Si consideri il grafo diretto in Fig. 19. Qual è il valore di $a.d$ dopo l'esecuzione di *Dijkstra*?

- A) 10.

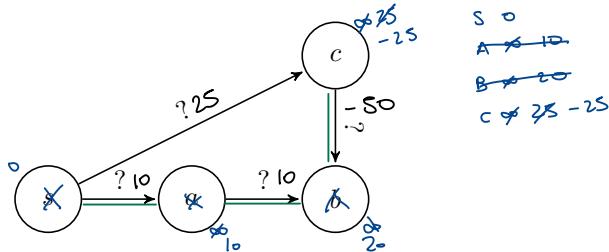
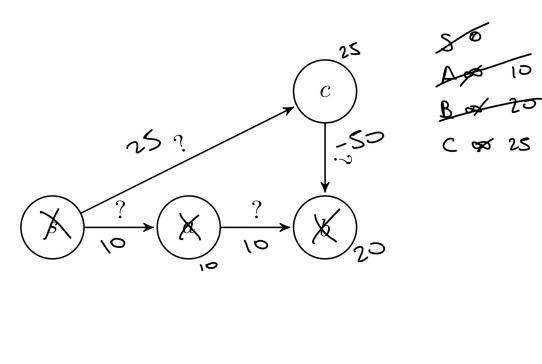


Figure 20. Grafo per l'esercizio 135.

- (B) 9.
C) 7.
D) ∞ .

- 135) Si consideri il grafo diretto in Fig. 20. Come si possono distribuire i pesi 10, 10, 25, -50 sugli archi in maniera da garantire che Dijkstra, eseguito con sorgente s , fallisca?
- A) $W_{sc} = -50, W_{cb} = 25, W_{sa} = 10, W_{ab} = 10$.
(B) $W_{sc} = 25, W_{cb} = -50, W_{sa} = 10, W_{ab} = 10$.
C) $W_{sc} = 25, W_{cb} = 10, W_{sa} = -50, W_{ab} = 10$.
D) ~~Dijkstra fallisce sempre in presenza di un arco negativo~~.

Dimostrazione.



- 136) 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?
- (A) Sì.
B) No.
C) No, ma esiste un algoritmo per il calcolo dell'albero di copertura minimi che, invece, funzionerebbe.
D) Sì, ma dovrebbe essere opportunamente modificato.

- 137) Nell'ambito del problema dei cammini minimi tra tutte le coppie, e dell'algoritmo della moltiplicazione di matrici per la sua soluzione, che cosa rappresenta il valore L_{ij}^m , per una certa coppia di vertici i, j ?
- A) Il peso del cammino minimo tra i e j .
B) Il peso del cammino minimo tra i e j usando almeno m archi.
C) Il peso del cammino minimo tra i e j usando esattamente m archi.
D) Il peso del cammino minimo tra i e j usando al più m archi.

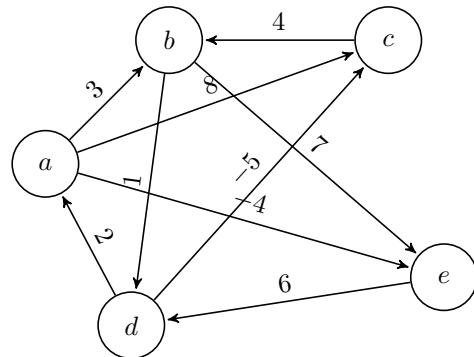


Figure 21. Grafo per l'esercizio 139.

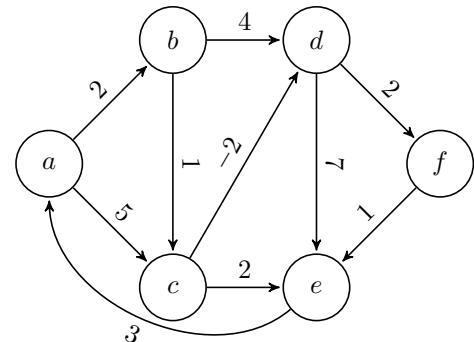


Figure 22. Grafo per l'esercizio 140.

- 138) Si consideri il grafo diretto in Fig. 21. Durante l'esecuzione di *SlowAllPairsMatrix* oppure di *FastAllPairsMatrix*, qual è il valore di L_{ad}^2 ?
- A) 2.
B) ∞ .
C) 13.
D) Non è definito.
- 139) Si consideri il grafo diretto in Fig. 21. Durante l'esecuzione di *Floyd-Warshall*, qual è il valore di D_{ec}^3 ?
- A) 3.
B) ∞ .
C) 10.
D) -2.
- 140) Si consideri il grafo diretto in Fig. 22. Durante l'esecuzione di *Floyd-Warshall*, qual è il valore di D_{da}^5 ?
- A) 10.
B) 5.
C) 6.
D) ∞ .