Domanda n.1
Insertion-sort nel caso pessimo ha un costo computazionale

medio = $O(n^2)$
ottimo = $O(n)$

O(n)
$O(n^2)$
O(nlog(n))
$o(n^2)$

Domanda n.2
Insertion-sort nel caso medio ha un costo computazionale

O(n)
$O(n^2)$
O(nlog(n))
$o(n^2)$

Domanda n.3
Insertion-sort nel caso ottimo ha un costo computazionale

O(n)
$O(n^2)$
O(nlog(n))
$o(n^2)$

Domanda n.4
ADS
Per definire una struttura dati astratta dobbiamo

definire i dati e le operazioni su di essi
descrivere le strutture dati che contengono i dati
descrivere l'implementazione delle operazioni sui dati

*Domanda n.*5
Quali delle seguenti sono strutture dati astratte ?

un insieme di interi con l'operazione "estrai il massimo"
un vettore di n numeri con l'operazione "estrai il massimo"
un vettore ordinato di n numeri con l'operazione "estrai il massimo"
un heap con l'operazione "estrai il massimo"

p a, coda, ADS
albero, tabella, insieme

Domanda n.6
Nel progettare una struttura dati "buona" si cerca di

minimizzare le risorse usate (tempo, spazio di memoria, ecc.) in funzione delle risorse disponibili mediamente negli elaboratori in commercio al momento
minimizzare le risorse usate (tempo, spazio di memoria, ecc.) in funzione delle risorse che si prevede saranno disponibili negli elaboratori entro qualche anno
massimizzare le risorse usate (tempo, spazio di memoria, ecc.) in funzione delle risorse disponibili mediamente negli elaboratori attuali
minimizzare le risorse usate (tempo, spazio di memoria, ecc.) in funzione della dimensione dell'istanza del problema che vogliamo risolvere

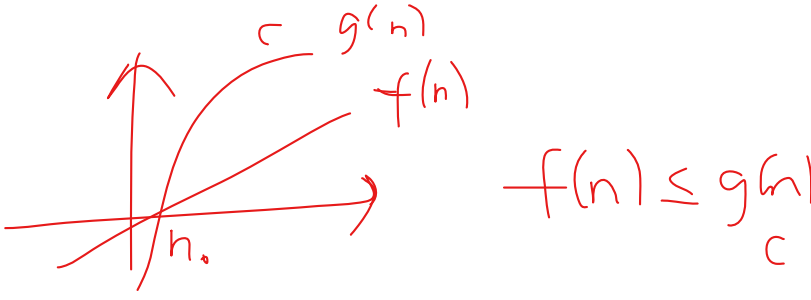
Domanda n.7
Con dimensione n di un problema ci riferiamo

alla lunghezza dell'algoritmo che lo risolve
al numero di istruzioni del programma che lo risolve
alla dimensione dell'input dell'istanza in considerazione
al numero di cicli nell'algoritmo che lo risolve
alla dimensione delle strutture dati utilizzate per risolverlo

*Domanda n.*8
L'analisi asintotica serve per ottenere una stima del tempo di esecuzione

per input molto grandi, considerando i dettagli implementativi
per ogni input, considerando i dettagli implementativi
per input molto grandi, astraendo i dettagli
per input molto piccoli, astraendo i dettagli
per input molto piccoli, considerando i dettagli implementativi
nessuna delle precedenti

*Domanda n.*9
Supponiamo che f(n) sia O(g(n)). Allora

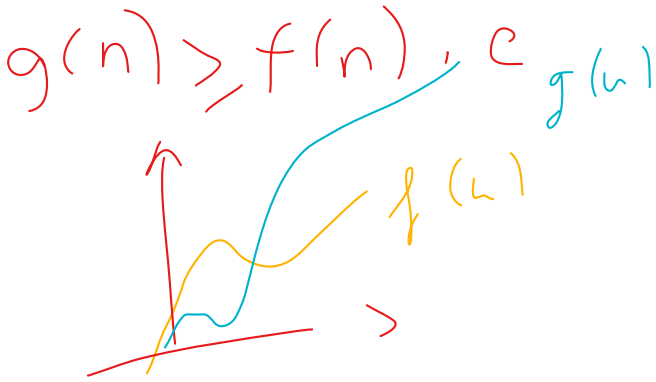


g(n) non è O(f(n))
g(n) è O(f(n))
g(n) può essere O(f(n))
g(n) può essere o(f(n))
g(n) può essere Ω(f(n))

*Domanda n.*10
Supponiamo che f(n) sia o(g(n)). Allora

g(n) non è O(f(n))
g(n) è O(f(n))

g(n) può essere O(f(n))
g(n) può essere o(f(n))
g(n) può essere Ω(f(n))



Domanda n.11
Supponiamo che f(n) sia Θ(g(n)). Allora

g(n) non è O(f(n)
g(n) è O(f(n))
g(n) può essere O(f(n))
g(n) è o(f(n))
g(n) è Ω(f(n))

Domanda n.12
La notazione O-grande si usa

nella valutazione del costo computazionale di un algoritmo nel caso ottimo
nella valutazione del costo computazionale di un algoritmo nel caso medio
nella valutazione del costo computazionale di un algoritmo nel caso pessimo
per determinare limiti inferiori di complessità di problemi computazionali

Domanda n.13
La notazione Ω-grande si usa

nella valutazione del costo computazionale di un algoritmo nel caso ottimo
nella valutazione del costo computazionale di un algoritmo nel caso medio
nella valutazione del costo computazionale di un algoritmo nel caso pessimo
per determinare limiti inferiori di complessità di problemi computazionali

*Domanda n.*14
20n + 36 nlogn è

O(n)
O(n^2)
O(nlog(n))
o(n^2)
o(n)

Domanda n.15
n^2 + n/3 + nlogn è

O(n)
O(n^2)
O(nlog(n))
o(n^2)
o(n)

Domanda n.16
n(n-1)/2 è

O(n)
O(n^2)
O(nlog(n))
o(n^3)
o(n)

Domanda n.17
20logn + 36 n^2logn è

O(n)
O(n^2)
O(nlog(n))
o(n^2)
o(n)

Domanda n.18
20n + 36 nlogn è

Ω(n)
Ω(n^2)
Ω(nlog(n))
Θ(n^2)
Θ(n)

Domanda n.19
n^2 + n/3 + nlogn è

Ω(n)
Ω(n^2)
Ω(nlog(n))
Θ(n^2)
Θ(n)

*Domanda n.*20
n(n-1)/2 è

Ω(n)
Ω(n^2)
Ω(n^2log(n))
Θ(n^2)
Θ(n)

*Domanda n.*21
20logn + 36 n^2logn è

Ω(n)
Ω(n^2)
Ω(n^2log(n))
Θ(n^2)
Θ(n)

*Domanda n.*22
Se il limite per n che va all'infinito di f(n)/g(n) = 0 possiamo dire che

f(n) è O(g(n))
f(n) è o(g(n))
f(n) è Ω (g(n))
g(n) è Ω (g(n))
g(n) è O (g(n))
niente

Domanda n.23
Se f(n) è O(g(n)) allora possiamo dire che

il limite per n che va all'infinito di f(n)/g(n) = 0
il limite per n che va all'infinito di f(n)/g(n) = costante maggiore di 0
niente
il limite per n che va all'infinito di f(n)/g(n) è più infinito
il limite per n che va all'infinito di f(n)/g(n) è finito
il limite per n che va all'infinito di f(n)/g(n) non esiste

*Domanda n.*24
Se f(n) è Θ(g(n)) allora possiamo dire che

il limite per n che va all'infinito di f(n)/g(n) = 0
il limite per n che va all'infinito di f(n)/g(n) = costante maggiore di 0
niente
il limite per n che va all'infinito di f(n)/g(n) è più infinito
il limite per n che va all'infinito di f(n)/g(n) è finito
il limite per n che va all'infinito di f(n)/g(n) non esiste

Domanda n.25
Se f(n) è o(g(n)) allora possiamo dire che

il limite per n che va all'infinito di f(n)/g(n) = 0
il limite per n che va all'infinito di f(n)/g(n) = costante maggiore di 0
niente
il limite per n che va all'infinito di f(n)/g(n) è più infinito
il limite per n che va all'infinito di f(n)/g(n) è finito
il limite per n che va all'infinito di f(n)/g(n) non esiste

Domanda n.26
Se f(n) è ω(g(n)) allora possiamo dire che

il limite per n che va all'infinito di f(n)/g(n) = 0
il limite per n che va all'infinito di f(n)/g(n) = costante maggiore di 0
niente
il limite per n che va all'infinito di f(n)/g(n) è più infinito
il limite per n che va all'infinito di f(n)/g(n) è finito
il limite per n che va all'infinito di f(n)/g(n) non esiste

Domanda n.27
L'approccio divide et impera ha il seguente costo computazionale:

sempre polinomiale
sempre esponenziale
dipende dal problema
non si può dire in anticipo

Domanda n.28
Merge-sort ordina un vettore A[1,..,n]

ordinando ricorsivamente i sottovettori A[1,q], A[q+1,n], dove q è circa n/2, e costruendo un nuovo vettore ordinato dato dalla loro unione
selezionando ad ogni ciclo l'elemento minimo della porzione di vettore non ancora ordinata A[j,…,n] per metterlo nella posizione j

selezionando ad ogni ciclo l'elemento massimo della porzione di vettore non ancora ordinata A[1,…,i] per metterlo nella posizione i
mantenendo un heap nella prima parte A[1,…,i] del vettore, scambiando ad ogni ciclo la radice dello heap con l'elemento in posizione i
ordinando ricorsivamente i sottovettori A[1,q], A[q+1,n] ottenendo il vettore ordinato alla fine del processo
mantenendo un vettore ordinato A[1,…,j-1] nel quale ad ogni ciclo viene aggiunto l'elemento j-esimo del vettore

*Domanda n.*29
Heap-sort ordina un vettore A[1,..,n]

ordinando ricorsivamente i sottovettori A[1,q], A[q+1,n], dove q è circa n/2, e costruendo un nuovo vettore ordinato dato dalla loro unione
selezionando ad ogni ciclo l'elemento minimo della porzione di vettore non ancora ordinata A[j,…,n] per metterlo nella posizione j
selezionando ad ogni ciclo l'elemento massimo della porzione di vettore non ancora ordinata A[1,…,i] per metterlo nella posizione i
mantenendo un heap nella prima parte A[1,…,i] del vettore, scambiando ad ogni ciclo la radice dello heap con l'elemento in posizione i
ordinando ricorsivamente i sottovettori A[1,q], A[q+1,n], dove q dipende dall'implementazione, ottenendo il vettore ordinato alla fine del processo
mantenendo un vettore ordinato A[1,…,j-1] nel quale ad ogni ciclo viene aggiunto l'elemento j-esimo del vettore

Domanda n.30
Quick-sort ordina un vettore A[1,..,n]

ordinando ricorsivamente i sottovettori A[1,q], A[q+1,n], dove q è circa n/2, e costruendo un nuovo vettore ordinato dato dalla loro unione
selezionando ad ogni ciclo l'elemento minimo della porzione di vettore non ancora ordinata A[j,…,n] per metterlo nella posizione j
selezionando ad ogni ciclo l'elemento massimo della porzione di vettore non ancora ordinata A[1,…,i] per metterlo nella posizione i
mantenendo un heap nella prima parte A[1,…,i] del vettore, scambiando ad ogni ciclo la radice dello heap con l'elemento in posizione i
ordinando ricorsivamente i sottovettori A[1,q], A[q+1,n], dove q dipende dall'implementazione, ottenendo il vettore ordinato alla fine del processo
mantenendo un vettore ordinato A[1,…,j-1] nel quale ad ogni ciclo viene aggiunto l'elemento j-esimo del vettore

Domanda n.31
Insertion-sort ordina un vettore A[1,..,n]

ordinando ricorsivamente i sottovettori A[1,q], A[q+1,n], dove q è circa n/2, e costruendo un nuovo vettore ordinato dato dalla loro unione
selezionando ad ogni ciclo l'elemento minimo della porzione di vettore non ancora ordinata A[j,…,n] per metterlo nella posizione j
selezionando ad ogni ciclo l'elemento massimo della porzione di vettore non ancora ordinata A[1,…,i] per metterlo nella posizione i
mantenendo un heap nella prima parte A[1,…,i] del vettore, scambiando ad ogni ciclo la radice dello heap con l'elemento in posizione i
ordinando ricorsivamente i sottovettori A[1,q], A[q+1,n], dove q dipende dall'implementazione, ottenendo il vettore ordinato alla fine del processo
mantenendo un vettore ordinato A[1,…,j-1] nel quale ad ogni ciclo viene aggiunto l'elemento j-esimo del vettore

Domanda n.32
Merge-sort è un algoritmo

stabile
non stabile

Domanda n.33
Heap-sort è un algoritmo

stabile
non stabile

Domanda n.34
Quick-sort è un algoritmo

stabile
non stabile

Domanda n.35
Insertion-sort è un algoritmo

stabile
non stabile

Domanda n.36
Bucket-sort è un algoritmo

stabile

non stabile

Domanda n.37
Counting-sort è un algoritmo

stabile
non stabile

Domanda n.38
Radix-sort è un algoritmo

stabile
non stabile

Domanda n.39
Merge-sort è un algoritmo

in place
non in place
dipende dall'implementazione

Domanda n.40
Heap-sort è un algoritmo

in place
non in place
dipende dall'implementazione

Domanda n.41
Quick sort è un algoritmo

in place
non in place
dipende dall'implementazione

Domanda n.42
Insertion-sort è un algoritmo

in place

non in place
dipende dall'implementazione

Domanda n.43
Bucket-sort è un algoritmo

in place
non in place
dipende dall'implementazione

Domanda n.44
Counting-sort è un algoritmo

in place
non in place
dipende dall'implementazione

Domanda n.45
Radix-sort è un algoritmo

in place
non in place
dipende dall'implementazione

Domanda n.46
Merge-sort nel caso pessimo ha costo computazionale

O(nlog(n))
$O(n^2)$
$o(n^2)$
o(nlog(n))
$o(n^3)$

Domanda n.47
Quick-sort nel caso pessimo ha costo computazionale

O(nlog(n))
$O(n^2)$
$o(n^2)$
o(nlog(n))
$o(n^3)$

Domanda n.48
Heap-sort nel caso pessimo ha costo computazionale

O(nlog(n))
O(n^2)
o(n^2)
o(nlog(n))
o(n^3)

Domanda n.49
Il principio di induzione dice che un'affermazione è vera

per ogni n>=0 se è vera per ogni n<=k con k molto grande
per ogni n>=0 se è vera per 0 e affermazione(n-1) implica affermazione(n)
per ogni n<0 se è vera per ogni n>0
per ogni n>=0 se è vera per n<=k e affermazione(n) implica affermazione(n+1) per ogni n>k

Domanda n.50
Il problema di ordinare n numeri (per input generali) ha una complessità computazionale

Θ(nlog(n))
Θ (n^2)
o(n^2)
O(nlog(n))
Θ(n^3)
Ω(nlog(n))

Domanda n.51
T(n) = T(n/2) + 1 ha soluzione

T(n) = Θ(n^2)
T(n) = Θ(nlog(n))
T(n) = Θ(n)
T(n) = Θ(log(n))

Domanda n.52
T(n) = 2T(n/2) + n ha soluzione

T(n) = Θ($n^{3/2}$)
T(n) = Θ(nlog(n))

T(n) = Θ(n)
T(n) = Θ(log(n))

Domanda n.53
Dato $x=\log_b a$ la soluzione di $T(n) = aT(n/b) + \Theta(n^x)$ è

$T(n) = \Theta(n^x)$
$T(n) = \Theta(n^x \log(n))$
T(n) = Θ(n)
T(n) = Θ(log(n))

Domanda n.54
Ricercare un numero in un vettore ha complessità computazionale

Θ(nlog(n))
$\Theta (n^2)$
$o(n^2)$
O(nlog(n))
Θ(n)
Ω(nlog(n))

Domanda n.55
Ricercare un numero in un vettore ordinato ha complessità computazionale

Θ(log(n))
$\Theta (n^2)$
$o(n^2)$
O(nlog(n))
Θ(n)
Ω(nlog(n))

Domanda n.56
Dato un vettore di n elementi nel quale si devono cercare k<logn elementi

conviene fare le ricerche senza ordinare il vettore
conviene ordinare il vettore prima di fare le ricerche
conviene ordinare il vettore dopo aver fatto le ricerche

Domanda n.57
Dato un vettore di n elementi nel quale si devono cercare k elementi

conviene sempre ordinare il vettore prima di fare la ricerca
non conviene mai ordinare il vettore
se k > n conviene fare le ricerche senza ordinare il vettore
se k > n conviene ordinare il vettore prima di fare le ricerche
se k = n conviene ordinare il vettore dopo aver fatto le ricerche

Domanda n.58
Che costo computazionale ha ricercare il massimo in una priority queue?

Θ(n)
Θ(1)
Θ(log(n))
Dipende dalla implementazione

Domanda n.59
Che costo computazionale ha ricercare il massimo in una priority queue implementata con un vettore non ordinato?

Θ(n)
Θ(1)
Θ(log(n))

Domanda n.60
Che costo computazionale ha ricercare il massimo in una priority queue implementata con un vettore ordinato?

Θ(n)
Θ(1)
Θ(log(n))

Domanda n.61
Che costo computazionale ha ricercare il massimo in una priority queue implementata con un heap?

Θ(n)
Θ(1)
Θ(log(n))

Domanda n.62

Che costo computazionale ha estrarre il massimo in una priority queue implementata con un vettore non ordinato?

Θ(n)
Θ(1)
Θ(log(n))

Domanda n.63
Che costo computazionale ha estrarre il massimo in una priority queue implementata con un vettore ordinato?

Θ(n)
Θ(1)
Θ(log(n))

Domanda n.64
Che costo computazionale ha estrarre il massimo in una priority queue implementata con un heap?

Θ(n)
Θ(1)
Θ(log(n))

Domanda n.65
Che costo computazionale ha inserire un elemento in una priority queue implementata con un vettore non ordinato?

Θ(n)
Θ(1)
Θ(log(n))

Domanda n.66
Che costo computazionale ha inserire un elemento in una priority queue implementata con un vettore ordinato?

Θ(n)
Θ(1)
Θ(log(n))

Domanda n.67

Che costo computazionale ha inserire un elemento in una priority queue implementata con un heap?

Θ(n)
Θ(1)
Θ(log(n))

Domanda n.68
Che costo computazionale ha la procedura Buil-heap?

Θ(n)
Θ(1)
Θ(nlog(n))
Altro
Θ(n^2)

Domanda n.69
Che costo computazionale ha la procedura Heapify?

Θ(n)
Θ(1)
Θ(nlog(n))
Θ(n^2)
Altro

O (log n)

Domanda n.70
Che costo computazionale ha la procedura Partition usata da Quick-sort?

Θ(n)
Θ(1)
Θ(nlog(n))
Θ(n^2)
Altro

Domanda n.71
Che costo computazionale ha Quick-sort nel caso medio?

Θ(n)
Θ(1)
Θ(nlog(n))
Θ(n^2)
Altro

Domanda n.72
È possibile ordinare n numeri in tempo o(nlog(n)) ?

Dipende dalle proprietà dell'input
Per input generali no
si, sempre
si, ma nel caso medio

Domanda n.73
Per algoritmi comparison sort esiste un limite inferiore di complessità (caso pessimo, input generali) pari a

Ω(n)
Ω(1)
Ω(nlog(n))
Ω(n^2)

Domanda n.74
Un albero binario di altezza h con n nodi ha al massimo

n foglie
2^n foglie
2^h foglie
nh foglie
h! foglie

Domanda n.75
Che costo computazionale ha nel caso medio Bucket-sort per input distribuiti uniformemente nell'intervallo [0,...,1)

Θ(log(n))
Θ(n)
o(n)
O(n)
Θ (n^2)

Domanda n.76
Che costo computazionale ha nel caso medio Bucket-sort per input distribuiti uniformemente nell'intervallo [0,...,5)

Θ(log(n))
Θ(n)
o(n)
O(n)
Θ (n^2)

Domanda n.77
Che costo computazionale ha nel caso pessimo Counting-sort per input nel range [1,...,k]

Θ(log(n))
Θ(nk)
Θ(n+k)
O(n+k)
Θ (n^2)

Domanda n.78
Che costo computazionale ha Counting-sort per input nel range [1,...,k]

Θ(log(n))
Θ(nk)
Θ(n+k)
O(n+k)
Θ (n^2)

Domanda n.79
Che costo computazionale ha Radix-sort per input con al massimo d cifre ognuna delle quali nel range [1,...,k] ?

Θ(log(d))
Θ(nd+dk)
Θ(nd)
O(n)
Θ(n^2)

Domanda n.80
Quali dei seguenti metodi permettono di risolvere equazioni ricorsive?

Albero di sostituzione
Albero di ricorsione
Master method
Metodo di intersezione
Metodo di sostituzione

Domanda n.81
Cosa è una priority queue ?

Una struttura dati astratta
Una struttura dati concreta
Un grafo
Un albero

Domanda n.82
Quali operazioni possono essere eseguite in una priority queue ?

Estrazione massimo
Inserimento
Ordinamento

Domanda n.83
Un albero con n nodi è definito come

un grafo aciclico con n-1 archi
un grafo aciclico connesso con n-1 archi
un grafo connesso con n-1 archi
un grafo connesso aciclico

Domanda n.84
La radice di un albero è quel nodo che

non ha figli
non ha padre
ha più figli di tutti gli altri nodi
ha più fratelli di tutti gli altri nodi

Domanda n.85
Una foglia di un albero è un nodo che

non ha figli
non ha padre
ha più figli di tutti gli altri nodi
ha più fratelli di tutti gli altri nodi

Domanda n.86
L’altezza h(a) di un nodo a è data

dal numero di figli del nodo a
dalla distanza tra il nodo a e la radice
dalla distanza massima tra il nodo a ed una foglia
nessuna delle precedenti

Domanda n.87
Un albero si dice binario quando

non ha più di due nodi
ogni nodo non ha più di due figli
l’altezza dell’albero è sempre multipla di due
il numero di nodi dell’albero è potenza di due

Domanda n.88
Un heap può essere definito come

un albero completo
un albero binario completo
un albero binario in cui il numero di nodi non super mai il numero di archi
nessuna delle precedenti

Domanda n.89
In un heap la relazione tra nodi figli dello stesso padre è

il figlio sinistro contiene una chiave minore delle chiave del figlio destro
il figlio destro contiene una chiave minore delle chiave del figlio sinistro
nessuna relazione
il figlio destro contiene una chiave uguale alla chiave del figlio sinistro

Domanda n.90
Il costo computazionale della procedura Heapify è

$\Theta(n)$
$\Theta(1)$
$\Theta(log(n))$
nessuna delle precedenti

Domanda n.91
Il costo computazionale per estrarre il massimo in una priority queue implementata con un heap è

O(n)
O(1)
O(log(n))
Θ(n)

Domanda n.92
Il costo computazionale per inserire un elemento in una priority queue implementata con un heap è

O(n)
O(1)
O(log(n))
Θ(n)

Domanda n.93
Cosa è un algoritmo comparison sort ?

un algoritmo per ordinare in tempo lineare
un algoritmo per ordinare che confronta i numeri tra di loro
un algoritmo per ordinare il cui output dipende solo dall'esito di confronti tra numeri

Domanda n.94
Un algoritmo di ordinamento si dice stabile se

funziona sempre anche quando ci sono malfunzionamenti hardware
produce un output in cui se due elementi uguali dell'input erano in un certo ordine vi rimangono
funziona senza usare memoria aggiuntiva oltre a quella usata per memorizzare l'input
funziona senza usare memoria aggiuntiva oltre a quella usata per memorizzare l'input e l'output

Domanda n.95
Un algoritmo di ordinamento ordina in place se

funziona sempre anche quando ci sono malfunzionamenti hardware
produce un output in cui se due elementi uguali dell'input erano in un certo ordine vi rimangono
funziona senza usare memoria aggiuntiva oltre a quella usata per memorizzare l'input
funziona senza usare memoria aggiuntiva oltre a quella usata per memorizzare l'input e l'output

Domanda n.96
Radix sort è stabile

vero
falso
dipende

Domanda n.97
Quali delle seguenti affermazioni è vera ?

calcolare il massimo in un vettore ha una complessità superiore a calcolare il mediano
calcolare il massimo in un vettore ha una complessità inferiore a calcolare il mediano
calcolare il massimo in un vettore ha la stessa complessità di calcolare il mediano

Domanda n.98
Quali delle seguenti affermazioni è vera ?

calcolare il massimo in un vettore ha una complessità superiore a calcolare l'iesimo elemento nell'ordinamento
calcolare il massimo in un vettore ha una complessità inferiore a calcolare l'iesimo elemento nell'ordinamento
calcolare il massimo in un vettore ha la stessa complessità di calcolare l'iesimo elemento nell'ordinamento

Domanda n.99
È possibile calcolare l'iesimo elemento nell'ordinamento partendo da un vettore non ordinato in tempo lineare

Si
No

Domanda n.100
In una pila gli elementi vengono estratti utilizzando una politica

LIFO
FIFO
TIFO
SCHIFO

Domanda n.101
In una coda gli elementi vengono estratti utilizzando una politica

LIFO
FIFO
TIFO
SCHIFO

Domanda n.102
Qual è il problema nell'usare i vettori per implementare code e pile?

non posso avere strutture dati di dimensioni maggiori della memoria presente nel calcolatore in uso
devo indirizzare gli elementi usando gli indici del vettore
non posso avere strutture dati di dimensioni maggiori di quelle definite in fase di programmazione
le operazioni di ordinamento e selezione sono computazionalmente molto costose

Domanda n.103
In una doubly linked list cancellare un elemento (fornito tramite un puntatore) costa

O(n)
O(nlog(n))
O(1)

Domanda n.104
In una singly linked list cancellare un elemento (fornito tramite un puntatore) costa

O(n)
O(nlog(n))
O(1)

Domanda n.105
Ricercare un elemento in una singly linked list ordinata costa

O(n)
O(log(n))
O(1)

Domanda n.106
Ricercare un elemento in una doubly linked list ordinata costa

O(n)
O(log(n))
O(1)

Domanda n.107
In una doubly linked list inserire un elemento (fornito tramite un puntatore) costa

O(n)
O(nlog(n))
O(1)

Domanda n.108
In una doubly linked list ordinata inserire un elemento (fornito tramite un puntatore) costa

O(n)
O(nlog(n))
O(1)

Domanda n.109
In una coda implementata tramite doubly linked list ENQUEUE costa

O(n)
O(nlog(n))
O(1)

Domanda n.110
In una coda implementata tramite doubly linked list DEQUEUE costa

O(n)
O(nlog(n))
O(1)

Domanda n.111
In una coda implementata tramite doubly linked list con sentinelle DEQUEUE costa

O(n)
O(nlog(n))
O(1)

Domanda n.112
In una pila implementata tramite doubly linked list PUSH o POP costano

O(n)
O(nlog(n))
O(1)
hanno costi diversi

Domanda n.113
Come posso rappresentare alberi generali tramite liste?

ogni nodo ha un puntatore al padre, al figlio sinistro ed alla lista dei fratelli
ogni nodo ha un puntatore per ogni figlio ed uno al padre
la radice ha un puntatore per ogni foglia ed ogni foglia ha un puntatore alla radice
ogni nodo ha un puntatore al padre ed uno al fratello destro

Domanda n.114
Come posso rappresentare alberi binari più semplicemente che con la rappresentazione per gli alberi generali?

ogni nodo ha un puntatore al padre, al figlio sinistro ed alla lista dei fratelli
ogni nodo ha un puntatore per ogni figlio ed uno al padre
la radice ha un puntatore per ogni foglia ed ogni foglia ha un puntatore alla radice
ogni nodo ha un puntatore al padre ed uno al fratello destro

Domanda n.115
Perché si implementano le liste usando le sentinelle ?

perché così facendo si diminuisce il costo computazionale
perché il codice risulta più semplice e pulito
perché è necessario farlo

Domanda n.116
In una tabella hash con indirizzamento diretto le operazioni di inserimento, ricerca e cancellazione hanno un costo computazionale

costante
lineare
logaritmico
quadratico

Domanda n.117
In una tabella hash con funzione hash h, la chiave i viene memorizzata in posizione

i
h(i)
ih(i)
all'inizio della tabella

Domanda n.118
Una buona funzione hash deve

minimizzare le collisioni
minimizzare lo spazio di memoria che serve per essere calcolata
massimizzare le collisioni
essere veloce da calcolare

Domanda n.119
Quando si genera una collisione in una tabella hash ?

quando la memoria della tabella viene esaurita
quando si associa la stessa posizione nella tabella a due chiavi distinte
quando due chiavi distinte hanno molti bit in comune
quando la funzione hash non è surgettiva

Domanda n.120
Una buona funzione hash può essere iniettiva

si
no

Domanda n.121
Se una funzione hash è biettiva

ho troppe collisioni
non ho collisioni, ma la tabella risulta essere di dimensioni troppo grandi
non riesco a inserire elementi nella tabella in tempo costante

Domanda n.122
Quali delle seguenti proprietà sono gradite per una funzione hash ?

iniettività
surgettività
assomigliare a una funzione random
generare posizioni che dipendono da tutti i bit della chiave
nessuna delle precedenti

Domanda n.123
Cosa si intende per simple uniform hashing ?

la tabella si riempie in modo uniforme
le chiavi vengono generate in modo uniforme
la probabilità che si inserisca un elemento nella posizione i della tabella è uguale alla probabilità che si inserisca un elemento nella posizione j, per ogni i,j
la funzione hash è costante

Domanda n.124
Usando il metodo di chaining nella gestione dei conflitti

una tabella con m posizioni può contenere al massimo m elementi
una tabella con m posizioni può contenere più di m elementi

Domanda n.125
Il load factor in una hash table è

il rapporto tra dimensione della tabella e numero di elementi contenuti nella tabella
dimensione della tabella
numero di elementi massimo che può contenere la tabella
il rapporto tra il numero di elementi nella tabella e la dimensione della tabella
nessuno dei precedenti

Domanda n.126
Il costo computazionale medio per la ricerca di un elemento in una tabella hash dove le collisioni vengono gestite con la tecnica di chaining è

Θ(1 + load factor)
Θ(1 + load factor) ma solo nel caso di simple uniform hashing
Θ(load factor + dimensione della tabella)
Θ(load factor + numero di elementi presenti nella tabella)

Domanda n.127

Il costo computazionale per la ricerca di un elemento in una tabella hash dove le collisioni vengono gestite con la tecnica di chaining nel caso pessimo è

Θ(numero di elementi presenti nella tabella)
Θ(1 + load factor)
Θ(load factor + dimensione della tabella)
Θ(load factor + numero di elementi presenti nella tabella)

Domanda n.128
Il costo computazionale per la ricerca di un elemento in una tabella hash dove le collisioni vengono gestite con la tecnica di chaining nel caso medio con simple uniform hashing e n=O(m), cioè il numero di elementi inseriti proporzionale alle dimensioni della tabella, è

O(n)
O(nlog(n))
O(1)

Domanda n.129
Per avere uniform hashing, nota la probabilità Pr(k) di ogni chiave, la somma delle probabilità delle chiavi che collidono (ΣPr(k) per ogni k t.c. h(k)=j) deve essere

uguale alla somma delle probabilità delle altre chiavi che collidono: se m è la dimensione della tabella ΣPr(k) = 1/m
uguale alla media delle probabilità: se n è il numero di chiavi che collidono ΣPr(k)=ΣPr(k)/n
uguale alla probabilità massima delle chiavi che collidono: ΣPr(k)=max(Pr(k))
non è un valore che influisce sull'uniform hashing

Domanda n.130
Nella progettazione di funzioni hash che godano di uniform hashing se non sono note a priori le probabilità delle chiavi Pr(k)

non è possibile usare una funzione hash per indirizzarle
si usano delle euristiche facendo dipendere h da tutti i bit di k cercando di mantenere indipendenza da pattern particolari
si può usare solo una funzione h(k)=k
si usano delle euristiche facendo dipendere h da eventuali pattern presenti nelle chiavi

Domanda n.131
Data una chiave k ed una tabella di dimensione m il Metodo della divisione prevede una funzione hash

h(k) = k
h(k) = k mod m

h(k) = Parte_Intera_Inferiore[m(kA mod m)]
h(k) = Parte_Intera_Inferiore[km]
nessuna delle precedenti

Domanda n.132
In una hash table con gestione delle collisioni tramite open addressing

il load factor non può mai essere maggiore di 1
il load factor non può mai essere minore di 1
il load factor può anche essere maggiore di 1

Domanda n.133
In una hash table (uniform hashing) con gestione delle collisioni tramite open addressing la lunghezza media di una probe

1/(1 - load factor)
1 + load factor
load factor
dimensione della tabella + 1

Domanda n.134
Sia x un nodo di un albero binario di ricerca

la chiave del figlio sinistro di x è minore o uguale della chiave di x
la chiave del figlio destro di x è maggiore della chiave di x
la chiave di x è maggiore o uguale della chiave dei figli di x
la chiave di x è minore o uguale della chiave dei figli di x
nessuna della precedenti

Domanda n.135
la ricerca di un elemento in un albero binario di ricerca con n nodi di altezza h ha un costo computazionale

O(n)
O(log(n))
O(h)
O(log(h))
O(nlog(h)

Domanda n.136

l’inserimento di un elemento in un albero binario di ricerca con n nodi di altezza h ha un costo computazionale

O(n)
O(log(n))
O(h)
O(log(h))
O(nlog(h)

Domanda n.137
la cancellazione di un elemento in un albero binario di ricerca con n nodi di altezza h ha un costo computazionale

O(n)
O(log(n))
O(h)
O(log(h))
O(nlog(h)

Domanda n.138
Sia x un nodo di un albero binario di ricerca. Il successore di x è

il figlio destro di x
il minimo nel sottoalbero sinistro di x
il minimo nel sottoalbero destro di x
il massimo del sottoalbero sinistro di x

Domanda n.139
Il minimo in un albero binario di ricerca

è una foglia
ha al più un figlio
ha almeno un figlio
ha più di due figli

Domanda n.140
Il successore di un nodo in un albero di ricerca viene usato quando

si inserisce un nodo più grande del massimo
si cerca un nodo
si cancella un nodo senza figli
si cancella un nodo con due figli

Domanda n.141
L’altezza di un albero binario di ricerca con n nodi

è al massimo log(n)
è al massimo n-1
è al massimo radice quadrata di n-1
è costante

Domanda n.142
Un red-black tree è

un albero binario di ricerca con alcune proprietà che garantiscono un certo bilanciamento
un grafo aciclico con più nodi che archi
un albero con nodi rossi e neri colorati in modo da renderlo più leggibile
un albero binario di ricerca nel quale l’ordinamento dei nodi si evidenzia colorandoli

Domanda n.143
In un red-black tree

ogni nodo è rosso o nero e tutte le foglie sono nere
ogni arco è rosso o nero
ogni foglia è rossa
il numero di nodi rossi è lo stesso di quelli neri

Domanda n.144
In un red-black tree

un nodo rosso può non avere figli ma se ne ha sono neri
ogni nodo rosso ha figli neri
ogni nodo nero ha figli rossi
i nodi sono tutti rossi tranne le foglie

Domanda n.145
In un red-black tree la black-height di un nodo x è

il numero di nodi neri (senza contare x) nel cammino da x a una foglia e dipende dal cammino scelto
il numero di nodi neri (senza contare x) nel cammino da x a una foglia ed è la stessa per ogni cammino possibile
il numero di nodi rossi (senza contare x) nel cammino da x a una foglia
il numero di nodi neri (compreso x) nel cammino da x a una foglia

Domanda n.146
Un red-black tree con n nodi ha un'altezza

minore o uguale a log(n)
minore uguale a 2(log(n)+1)
minore o uguale a $2^{log(n)}$

Domanda n.147
Il numero di nodi interni in un red-black tree radicato in x è

non inferiore a bh(x)
non superiore a bh(x)
non inferiore a $2^{bh(x)}$-1
non superiore a $2^{bh(x)}$-1
nessuna delle precedenti

Domanda n.148
Le rotazioni su red-black tree sono

operazioni di ristrutturazione locale sull'albero utilizzate per ripristinarne le proprietà
operazioni di inserimento di nuovi elementi
operazioni di cancellazione di elementi dell'albero
operazioni di ricolarazione di nodi utilizzate per aumentare l'altezza dell'albero

Domanda n.149
La cancellazione di un nodo in un red-black tree si effettua

in maniera analoga a quella negli alberi binari di ricerca
in maniera analoga a quella negli alberi binari di ricerca ma con operazioni di ristrutturazione e ricolorazioni per mantenere le proprietà dell'albero
in maniera analoga a quella negli heap
in maniera analoga a quella negli heap ma con operazioni di ristrutturazione e ricolorazioni per mantenere le proprietà dell'albero

Domanda n.150
Una rotazione su red-black trees ha un costo computazionale

logaritmico nel numero di nodi dell'albero
costante
lineare nel numero di nodi dell'albero

nessuna delle precedenti

Domanda n.151
L'altezza di un red-black tree con n nodi è

O(log(n))
O(n)
Ω(n)
Ω(log(n))

Domanda n.152
Cancellare un nodo in un red-black tree con n nodi ha un costo computazionale

O(n)
Ω(log(n))
O(log(n))
Ω(n)
O(1)

Domanda n.153
Inserire un nodo in un red-black tree con n nodi ha un costo computazionale

O(n)
Ω(log(n))
O(log(n))
Ω(n)
O(1)

Domanda n.154
Quali delle seguenti affermazioni relative alla programmazione dinamica sono vere ?

Lo stesso sottoproblema può essere risolto più volte
Ogni sottoproblema viene risolto una sola volta e il risultato memorizzato in una tabella
La soluzione di un sottoproblema viene memorizzata per un certo lasso di tempo e poi viene rimossa dalla tabella

Domanda n.155
Quali dei seguenti sono passi fondamentali in programmazione dinamica

calcolo delle soluzioni per tutti i possibili sottoproblemi
caratterizzazione della struttura di una soluzione ottima

definizione ricorsiva del valore di una soluzione ottima
enumerazione di tutti i possibili sottoproblemi
costruzione di una soluzione ottima a partire dalle informazioni già calcolate

Domanda n.156
Dire se la seguente affermazione è una condizione necessaria per applicare la programmazione dinamica: una soluzione ottima per il problema contiene al suo interno le soluzioni ottime dei sottoproblemi.

vero
falso
dipende

Domanda n.157
Cosa significa che un problema di ottimizzazione ha sottoproblemi comuni ?

significa che un algoritmo ricorsivo richiede di risolvere più di una volta lo stesso sottoproblema
significa che nella soluzione ottima ci sono contenute le sottosoluzioni dello stesso sottoproblema più volte
significa che per risolvere il problema dobbiamo risolvere più di un sottoproblema
significa che tutti i sottoproblemi sono uguali e quindi è sufficiente risolverne una sola copia

Domanda n.158
Tutti i problemi di ottimizzazione possono essere risolti efficientemente con la programmazione dinamica.

vero
faiso

Domanda n.159
Tutti i problemi di ottimizzazione possono essere risolti efficientemente con l'approccio divide et impera

vero
falso

Domanda n.160
Quali delle seguenti affermazioni meglio descrivono un algoritmo greedy ?

un algoritmo greedy risolve alcuni sottoproblemi e poi calcola la soluzione finale usando le informazioni calcolate

un algoritmo greedy compie una sequenza di scelte basandosi sui dati a disposizione e cercando ad ogni passo di costruire la migliore soluzione possibile
un algoritmo greedy calcola una soluzione e poi cerca di migliorarla fino a trovare quella ottima

Domanda n.161
Gli algoritmi greedy trovano sempre la soluzione ottima.

vero
falso

Domanda n.162
il seguente algoritmo per risolvere il problema del commesso viaggiatore è un algoritmo greedy ? Parto dalla città numero 1 e poi procedo visitando la città più vicina non ancora visitata. Quando tutte le città sono state visitate torno alla città numero 1.

si
no
dipende

Domanda n.163
Esiste un algoritmo greedy per risolvere il commesso viaggiatore ?

si
no
non si sa

Domanda n.164
È vero che, come nel caso della programmazione dinamica, anche per applicare un algoritmo greedy occorre che la soluzione ottima contenga le soluzioni ottime dei sottoproblemi ?

si
no
dipende

Domanda n.165
Quali delle seguenti affermazioni sono vere ?

è sempre possibile risolvere un problema di ottimizzazione con algoritmi greedy o con programmazione dinamica
non è sempre possibile risolvere un problema di ottimizzazione con algoritmi greedy o con programmazione dinamica

ci sono problemi che si possono risolvere con programmazione dinamica ma non si possono risolvere con algoritmi greedy
i problemi che si possono risolvere con programmazione dinamica si possono risolvere anche con algoritmi greedy

Domanda n.166
I problemi di ottimizzazione che soddisfano la proprietà della sottostruttura ottima possono essere risolti con algoritmi greedy

vero
falso
dipende

Domanda n.167
I problemi di ottimizzazione che soddisfano la proprietà della sottostruttura ottima possono essere risolti con la programmazione dinamica

vero
falso
dipende

Domanda n.168
I problemi di ottimizzazione che soddisfano la proprietà della sottostruttura ottima possono essere risolti sia con algoritmi greedy che con la programmazione dinamica

vero
falso
dipende

Domanda n.169
Un algoritmo greedy ha il seguente costo computazionale:

sempre polinomiale
sempre esponenziale
dipende dal problema
nessuno dei precedenti

Domanda n.170
L'approccio della programmazione dinamica ha il seguente costo computazionale:

sempre polinomiale

sempre esponenziale
dipende dal problema
nessuno dei precedenti

Domanda n. 171
La memoria di massa (HD) rispetto a quella centrale (non cache) é circa
1) *Mille volte più grande*
2) *Cento volte più grande*
3) *Della stessa capacità*
4) *Cento volte più piccola*

Domanda n. 172
La memoria centrale (non cache) rispetto a quella di massa (HD) é circa
1) *Mille volte più veloce*
2) *Un milione di volte più veloce*
3) *Della stessa velocità*
4) *mille volte più lenta*

Domanda n. 173
Un blocco di memoria principale, rispetto ad uno di memoria di massa, é circa
1) *mille volte più grande*
2) *duecento volte più grande*
3) *grande uguale*
4) *duecento volte più piccolo*

Domanda n. 174
I B Trees sono una struttura dati che ha l'obiettivo di
1) *massimizzare il numero di accesi alla memoria di massa*
2) *minimizzare il numero di accessi alla memoria principale*
3) *massimizzare il numero di accessi alla memoria principale*
4) *minimizzare il numero di accessi alla memoria di massa*
5) *aiutare la memoria del programmatore*

Domanda n. 175
Su un HD é mediamente più veloce
1) *trovare il settore in cui sono memorizzati i dati da leggere*
2) *leggere il settore (dopo averlo trovato) in cui sono memorizzati i dati di interesse*
3) *formattare il disco*
4) *attendere il settore corretto dopo aver posizionato le testine sulla sua traccia*

Domanda n. 176
Per valutare il tempo di esecuzione di un algoritmo che legge i dati da un HD é necessario considerare
1) *numero di accessi a disco e numero di accessi alla memoria centrale*
2) *numero di accessi a disco e tempo di calcolo della CPU*
3) *numero di accessi alla memoria centrale e numero di operazioni della CPU*
4) *numero di accessi alla memoria centrale e numero di stampe*

Domanda n. 177
Nel calcolo del tempo di esecuzione di un algoritmo che accede a dati su HD, il numero di accessi a disco é misurato come:

1) *numero di pagine lette/scritte*
2) *numero di byte letti*
3) *numero di byte scritti*
4) *numero di spostamenti delle testine di lettura/scrittura*
5) *numero di tracce del HD*

Domanda n. 178
Analizzando un algoritmo per B trees, le notazioni key[x] e DiskRead(x) identificano rispettivamente
1) *la stessa operazione, di lettura dati da disco*
2) *un accesso in memoria centrale e una lettura da HD*
3) *la stessa operazione, di lettura dati da memoria centrale*
4) *una lettura da memoria centrale e una lettura da disco*

Domanda n. 179
Ad ogni nodo di un B tree vengono associati
1) *numero del nodo e puntatori ai nodi figli*
2) *numero delle chiavi del nodo, flag se foglia e puntatori ai figli*
3) *numero delle chiavi, chiavi e flag se nodo interno*
4) *flag se foglia, puntatore al padre e puntatori ai fratelli*
5) *numero delle chiavi, chiavi, puntatore al padre*

Domanda n. 180
Un B tree é un albero
1) *che può essere molto sbilanciato*
2) *in cui tutte le foglie hanno la stessa profondità*
3) *in cui ogni nodo interno ha la stessa profondità*
4) *in cui l'altezza dell'albero é uguale alla profondità di una qualsiasi foglia*
5) *binario (B tree sta per Binary tree)*

Domanda n. 181
In un B tree, il grado minimo t e'
1) *il minimo numero di chiavi memorizzabili in un nodo*
2) *il minimo numero di figli che un nodo interno può avere*
3) *il minimo valore che può assumere la chiave di un nodo*
4) *il massimo numero di chiavi memorizzabili in un nodo*
5) *il minimo valore di temperatura di un nodo*

Domanda n. 182
In un B tree con t=1000 e altezza 2
1) *é necessario accedere 1000 volte a disco prima di ottenere una qualsiasi chiave*
2) *é necessario accedere 2 volte a disco prima di ottenere una qualsiasi chiave*
3) *é possibile memorizzare più di 2.000.000.000 di chiavi*
4) *é possibile memorizzare più di 1.000.000.000 di chiavi*
5) *é possibile memorizzare al più 1.001.000 chiavi*

Domanda n. 183
Un B tree
1) *é un caso particolare di BST*
2) *é un caso particolare di Fibonacci heap*
3) *ha profondità O(log n)*

4) ha profondità O(t), t grado minimo dell'albero
5) implementa il risultato dell'esecuzione dell'algoritmo di Dijkstra da uno a tutti

Domanda n. 184
Un BTree di altezza h con n chiavi e di grado minimo t≥2 ha altezza:
1) $h \geq \log_t((n+1)/2)$
2) $h < \log_t((n+1)/2)$
3) $h \leq \log_t((n+1)/2)$
4) $h > \log_t((n+1)/2)$
5) $h = \log_t((n+1)/2)$

Domanda n. 185
Avendo dati memorizzati con un B tree, il numero di accessi a disco necessari per recuperare un dato e':
1) $O(\log_n t)$
2) $O(t \log_t n)$
3) $O(t \log_n t)$
4) $O(\log_t n)$
5) $O(n \log_t n)$

Domanda n. 186
Avendo dati memorizzati con un B tree, il tempo di CPU necessario per recuperare un dato e':
1) $O(\log_t n)$
2) $O(\log_n t)$
3) $O(n \log_n t)$
4) $O(n \log_t n)$
5) $O(t \log_t n)$

Domanda n. 187
Per operare su un B tree si assume che la radice sia:
1) in memoria centrale
2) su disco
3) il primo nodo
4) il nodo più basso

Domanda n. 188
Fra le operazioni possibili su un B tree rientrano
1) creazione e cancellazione di un albero, aggiornamento e modifica di chiavi
2) cancellazione di un albero, aggiornamento e modifica di chiavi
3) creazione di un nuovo albero, ricerca, inserimento e cancellazione di chiavi.
4) creazione di un albero, ricerca di chiavi e modifica del grado minimo
5) creazione e cancellazione di un albero, modifica del grado minimo e riposizionamento della radice

Domanda n. 189
L'algoritmo B-tree-search

1) *permette di trovare una chiave k in un sottoalbero di radice x (x e k parametri)*
2) *é una procedura che accede ricorsivamente a disco*
3) *permette di trovare in quale albero di radice x é presente la chiave k (k parametro)*
4) *ricerca in modalità breadth-first nel sottoalbero di radice x (x parametro)*

Domanda n. 190
La procedura B-tree-create
1) *ha complessità O(n), n numero di chiavi da inserire nell'albero*
2) *accede al disco n volte, n numero di chiavi da inserire nell'albero*
3) *prevede un numero di accessi a disco O(1)*
4) *ha un tempo di CPU O(1)*
5) *Prende un sottoalbero di un albero T (T parametro) e lo fa diventare un B tree autonomo*

Domanda n. 191
In un B tree un nodo é pieno quando
1) *contiene 2t-1 chiavi (t grado minimo)*
2) *contiene t chiavi (t grado minimo)*
3) *contiene t-1 chiavi (t grado minimo)*
4) *contiene più di t chiavi (t grado minimo)*
5) *contiene n $\log_t n$ chiavi*

Domanda n. 192
In un B tree si divide un nodo
1) *quando vi si vuole inserire una chiave di valore maggiore della più piccola chiave del nodo fratello di destra*
2) *quando é pieno e vi si vuole inserire un'ulteriore chiave*
3) *quando inserendo una nuova chiave l'albero diventerebbe sbilanciato*
4) *mai*
5) *in corrispondenza alla sua chiave mediane*

Domanda n. 193
Il risultato di uno split di un nodo di un B tree e':
1) *nodo padre invariato, due nodi figli*
2) *nodo padre con una chiave in più, due nodi figli invece di uno*
3) *nodo padre con una chiave in meno, due nodi figli invece di uno*
4) *nodo padre invariato, nodo figlio dimezzato con un figlio a sua volta*
5) *nodo padre con una chiave in più, nodo figlio dimezzato con un figlio a sua volta*

Domanda n. 194
I parametri della procedura B-tree-split-child sono:
1) *x: nodo padre, y: primo nodo figlio, z: secondo nodo figlio*
2) *T: b tree su cui lavorare, x: nodo da spazzare, k: chiave mediana*
3) *T: b tree su cui lavorare, x: nodo da spazzare, y: secondo nodo figlio*
4) *x: nodo padre, i: indice in x, y: nodo da spezzare*
5) *T: b tree su cui lavorare, x,y: nodi figli*

Domanda n. 195
La procedura B-tree-split-child
1) *ha complessità $\Theta(t)$*
2) *ha complessità O(t)*
3) *ha complessità O(1)*

4) ha complessità Θ(n)
5) ha complessità O(n)

Domanda n. 196
L'inserimento di un nuovo elemento in un B tree
1) viene effettuato sempre nella foglia più vuota
2) scorre ricorsivamente i nodi, scendendo di un livello se il nodo corrente contiene 2t-1 elementi
3) scorre ricorsivamente i nodi, scendendo di un livello se il nodo corrente contiene t-1 elementi
4) scorre ricorsivamente i nodi, scendendo di un livello se il nodo corrente contiene t elementi
5) viene effettuato sempre nella radice

Domanda n. 197
La procedura B-tree-insert
1) inserisce un elemento in una foglia dell'albero
2) inserisce un sottoalbero in un B tree passato come parametro
3) inserisce una nuova foglia in un B tree passato come parametro
4) inserisce un nuovo elemento nella radice dell'albero
5) modifica la radice dell'albero

Domanda n. 198
Lo split della radice di un B tree
1) fa sì che l'albero abbia due radici
2) fa sì che l'albero cresca (verso l'alto)
3) fa sì che l'albero cresca (verso il basso)
4) non modifica il numero complessivo dei nodi dell'albero

Domanda n. 199
L'inserimento di un elemento k in un nodo x di un B tree
1) viene effettuato dalla procedura B-tree-insert
2) viene effettuato dalla procedura B-tree-insert-non-full
3) viene effettuato dalla procedura B-tree-insert-in-node-x
4) viene effettuato manualmente dal programmatore
5) viene effettuato in modalità batch

Domanda n. 200
La procedura B-tree-insert-non-full
1) gestisce l'inserimento di un elemento in una foglia
2) gestisce l'attraversamento del B tree
3) gestisce lo svuotamento di una foglia piena
4) gestisce sia l'inserimento di un elemento in una foglia che l'attraversamento dell'albero

Domanda n. 201
Inserendo successivamente elementi in un B tree può succedere che
1) una foglia diventi padre di un nuovo nodo
2) venga ridefinita la radice dell'albero
3) una foglia venga passata da un nodo interno ad un altro
4) vengano aggiunte nuove foglie
5) vengano aggiunti nuovi nodi interni

Domanda n. 202

Inserendo un elemento in un B tree, il numero di accessi a disco e':

1) *O(n)*
2) *O(t)*
3) *O(h)*
4) *O(th)*
5) *O(t $\log_t$ n)*

Domanda n. 203

Inserendo un elemento in un B tree, il tempo di CPU e':

1) *O(t)*
2) *O(th)*
3) *O(n)*
4) *O(h)*

Domanda n. 204

La cancellazione di un elemento da un B tree

1) *viene effettuata ricorsivamente, dalla radice verso le foglie*
2) *viene effettuata ricorsivamente, da una foglia verso la radice*
3) *avviene solo nel nodo radice*
4) *avviene solo in un nodo foglia*
5) *richiede una espansione completa breadth search dell'albero*

Domanda n. 205

Per la cancellazione di un elemento da un B tree

1) *Si scende ad un nuovo livello dell'albero se il nodo corrente contiene meno di 2t-1 elementi*
2) *Si scende ad un nuovo livello dell'albero se il nodo corrente contiene più di t elementi*
3) *Si scende ad un nuovo livello dell'albero se il nodo corrente contiene più di 2t-1 elementi*
4) *Si scende ad un nuovo livello dell'albero se il nodo corrente contiene meno di t elementi*

Domanda n. 206

La procedura B-tree-delete di cancellazione di un elemento k da un B tree T

1) *gestisce tre casi: k in una foglia, k in un nodo interno, k nella radice*
2) *gestisce due casi: k in una foglia, k in un nodo interno*
3) *gestisce tre casi: k in una foglia, k in un nodo interno, k probabilmente in un nodo di livello inferiore*
4) *gestisce tre casi: k in una foglia, k in un nodo interno, k non presente nell'albero*
5) *gestisce due casi: k presente e k non presente nell'albero*

Domanda n. 207

Dovendo cancellare da un B tree un elemento k che si trova in un sottoalbero di radice c_i, l'operazione di cancellazione-fusione

1) *avviene se c_i ha t-1 elementi*
2) *avviene se c_i e tutti i suoi fratelli hanno t-1 elementi*
3) *avviene se c_i ha meno di t-1 elementi*
4) *avviene se c_i e tutti i suoi fratelli hanno 2t-1 elementi*

Domanda n. 208

La procedura di cancellazione di un elemento da un B tree

1) *ha tempo di CPU O(n)*

2) *ha tempo di CPU O(t $\log_t$ n)*
3) *ha tempo di CPU O(tn)*
4) *ha tempo di CPU O(t $\log_t$ h)*
5) *ha tempo di CPU O(th)*

Domanda n. 209
La procedura di cancellazione di un elemento k da un B tree
1) *richiede esclusivamente di scendere lungo l'albero dalla radice verso una foglia*
2) *richiede di considerare sempre tutti i fratelli di un nodo corrente*
3) *può richiedere di risalire il B-tree verso la radice*
4) *opera sempre a livello foglia del B tree*

Domanda n. 210
Il numero di accessi a disco durante un'operazione di cancellazione di un elemento da un B tree e':
1) *O(t)*
2) *O(h)*
3) *O(th)*
4) *O(t $\log_t$ n)*
5) *O(n)*

Domanda n. 211
Nei B+ tree
1) *gli elementi contenenti dati sono memorizzati solo nelle foglie*
2) *gli elementi contenenti dati sono memorizzati nei nodi interni e nelle foglie*
3) *i nodi dell'albero contengono solo puntatori a record di un database*
4) *i nodi dell'albero sono come quelli di un B tree, a cui però si aggiungono informazioni addizionali*
5) *l'albero é indicizzato*

Domanda n. 212
Una heap binomiale
1) *é un albero binomiale*
2) *é un insieme di alberi binomiali*
3) *é una procedura di ricerca binomiale*
4) *é una particolare struttura dati a base 2*
5) *é una struttura dati algebrica definita su binomi*

Domanda n. 213
I nodi di un albero binomiale B_k sono
1) *al più 2^k*
2) *almeno 2^k*
3) *2^k*
4) *almeno $\log_2 k$*

Domanda n. 214
L'altezza di un albero binomiale con 2^k nodi e'
1) *k*
2) *$\log_2 k$*

3) ***al più k***
4) ***almeno k***
5) ***almeno $\log_2 k$***

Domanda n. 215
In un albero binomiale B_k i nodi a profondità *i* (*i* = 0, ... , *k*) sono

1) ***Al più*** $\binom{k}{i}$
2) ***Almeno*** $\binom{k}{i}$
3) ***Esattamente*** 2^i
4) ***Esattamente*** $\binom{k}{i}$
5) ***Esattamente*** $\log_2 i$

Domanda n. 216
In un albero binomiale B_k, di altezza *k*, la radice ha grado
1) ***k***
2) ***2***
3) ***2^k***
4) ***$\log_2 k$***
5) ***2k***

Domanda n. 217
In una heap binomiale H
1) ***tutti gli alberi binomiali hanno le radici dello stesso grado***
2) ***le radici degli alberi binomiali hanno tutte grado pari***
3) ***non esistono due alberi binomiali le cui radici hanno lo stesso grado***
4) ***in ogni albero binomiale, per ogni nodo si ha che la sua chiave é maggiore o uguale a quella dei genitori***

Domanda n. 218
In ogni nodo di una heap binomiale sono presenti i campi
1) ***p, key, deg, child, left, right***
2) ***p, key, data, deg, child, left, right***
3) ***p, data, deg, child, left, right***
4) ***p, key, deg, child, sibling***
5) ***p, key, data, deg, child, sibling***

Domanda n. 219
La procedura Binomial-Heap-Minimum
1) ***restituisce la chiave minima contenuta in una heap binomiale, che é nel nodo puntato da head***
2) ***restituisce un puntatore al nodo con chiave minima contenuta in una heap binomiale, che é il nodo puntato da head***
3) ***restituisce la chiave minima contenuta in una heap binomiale, che é in una radice di un albero***

4) restituisce un puntatore al nodo con chiave minima contenuta in una heap binomiale, che é in una radice di un albero
5) restituisce la chiave minima contenuta in una heap binomiale, che é in una foglia di un albero

Domanda n. 220
L'unione di due heap binomiali H1 e H2
1) collega le radici di H1 e H2
2) collega tutti gli alberi binomiali di H1 e H2 le cui radici hanno lo stesso grado
3) appende H1 ad una foglia di H2
4) appende H2 ad una foglia di H1
5) appende gli alberi di H2 dopo quelli di H1

Domanda n. 221
La procedura Binomial-Link(y,z) collega
1) le heap binomiali y e z, z diventa padre di y
2) le heap binomiali y e z, y diventa padre di z
3) le heap binomiali y e z, z viene appesa ad una foglia di y
4) gli alberi binomiali y e z, z viene appeso ad una foglia di y
5) gli alberi binomiali y e z le cui radici hanno lo stesso grado, z diventa padre di y

Domanda n. 222
La procedura Binomial-Heap-Union(H1,H2) unisce
1) le due heap binomiali H1 e H2, H2 viene appesa ad una foglia di H1
2) le due heap binomiali H1 e H2, H1 viene appesa ad una foglia di H2
3) i due alberi binomiali H1 e H2, le cui radici hanno lo stesso grado
4) le due heap binomiali H1 e H2, distruggendole
5) i due alberi binomiali H1 e H2, H2 viene appeso ad una foglia di H1

Domanda n. 223
La procedura Binomial-Heap-Insert(H,x) inserisce
1) il nodo x nella heap binomiale H
2) il nodo x nell'albero binomiale H
3) l'elemento nel nodo di chiave x dell'albero binomiale H
4) l'elemento nel nodo di chiave x della heap binomiale H

Domanda n. 224
La procedura Binomial-Heap-Extract-Min(H)
1) restituisce il puntatore al nodo con chiave minima di una heap binomiale H
2) elimina da una heap binomiale H il nodo con chiave minima e restituisce il puntatore a quel nodo
3) restituisce il valore della chiave minima di una heap binomiale H, che é sempre nel nodo puntato da head(H)
4) restituisce il valore della chiave minima di una heap binomiale H, che é sempre in una foglia di H
5) restituisce il valore della chiave minima di una heap binomiale H, che epuò essere in un nodo interno di H

Domanda n. 225
La procedura Binomial-Heap-Decrease-Key(H,x,k)

1) *decrementa del valore della chiave del nodo k la chiave del nodo x di una heap binomiale H. Errore se key[k]>key[x]*
2) *decrementa di k il valore della chiave di un nodo x di una heap binomiale H.*
3) *decrementa del valore del nodo x il valore della chiave key[x] associata ad una heap binomiale H*
4) *assegna un nuovo valore k alla chiave di un nodo x di una heap binomiale H. Errore se k>key[x]*
5) *decrementa di k il valore della chiave di un nodo x di una heap binomiale H. Errore se k>key[x]*

Domanda n. 226

La complessità più alta di una procedura che opera su una heap binomiale e':

1) *Θ(n) nell'unione*
2) *Θ (log n) nell'inserimento*
3) *Θ (n log n) nell'inserimento*
4) *O(log n) nell'inserimento*
5) *Θ(n) nella cancellazione*

Domanda n. 227

Una heap di Fibonacci e'

1) *un insieme di alberi binomiali organizzati a heap*
2) *un albero binomiale organizzato a heap di Fibonacci*
3) *un insieme di alberi, non necessariamente binomiali, con l'ordinamento parziale a heap*
4) *un insieme di heap binomiali in cui ad ogni livello i nodi crescono secondo i numeri di Fibonacci*

Domanda n. 228

Gli alberi componenti una heap di Fibonacci

1) *sono tutti alberi binomiali non ordinati*
2) *sono tutti alberi binomiali ordinati*
3) *sono alberi anche non binomiali ordinati*
4) *sono alberi anche non binomiali non ordinati*
5) *sono alberi di Fibonacci*

Domanda n. 229

Le heap di Fibonacci rispetto a quelle binomiali

1) *sono meno vincolate, permettono di differirne la riorganizzazione*
2) *sono più vincolate, permettono di differirne la riorganizzazione*
3) *sono più vincolate, non é mai necessario riorganizzarle*
4) *sono meno vincolate, non é mai necessario riorganizzarle*
5) *hanno gli stessi vincoli strutturali*

Domanda n. 230

I nodi marcati in una heap di Fibonacci

1) *hanno perso un figlio dall'ultima volta in cui erano diventati loro stessi figli di un altro nodo*
2) *hanno perso un figlio dall'ultima volta in cui erano diventati loro stessi padri di un altro nodo*
3) *hanno perso un figlio dall'ultima volta in cui erano diventati radici della heap*
4) *hanno cambiato padre dall'ultima volta in cui erano diventati loro stessi figli di un altro nodo*

5) hanno cambiato padre dall'ultima volta in cui erano diventati loro stessi padri di un altro nodo

Domanda n. 231
La radice di una heap di fibonacci
1) é il nodo con grado minimo
2) é il nodo con chiave massima
3) é il nodo con chiave minima
4) é anche la radice dell'albero più piccolo
5) é anche la radice dell'albero più grande

Domanda n. 232
Gli elementi di un nodo di una heap di fibonacci sono
1) p, key, data, deg, child, sibling
2) p, key, data, deg, child, left, right
3) p, key, deg, child, sibling
4) p, key, deg, child, left, right
5) p, ancestor, data, deg, child, sibling

Domanda n. 233
La procedura Fib-Heap-Insert(H,x)
1) inserisce il nodo x nella heap di fibonacci H appendendolo come foglia dell'albero più piccolo
2) inserisce il nodo x nella heap di fibonacci H inserendolo fra le radici
3) inserisce il nodo x nella heap di fibonacci H inserendolo come nodo dell'albero più sbilanciato
4) inserisce il di chiave x nella heap di fibonacci H sostituendo la chiave dell'elemento minore

Domanda n. 234
La procedura di estrazione del nodo minimo da una heap di fibonacci
1) é una operazione molto semplice dato che si ha subito il puntatore all'elemento minimo
2) é una operazione semplice ma richiede la ricerca dell'elemento minimo
3) é una operazione complicata che richiede la ristrutturazione dell'albero in cui si trova il nodo
4) é una operazione complicata perché é necessario trovare il nodo minimo

Domanda n. 235
La procedura Fib-Heap-Extract-Min(H)
1) elimina da H il nodo minimo e inizialmente ne inserisce tutti i figli fra le radici di H
2) non elimina da H il nodo minimo ma inizialmente ne inserisce tutti i figli fra le radici di H
3) elimina da H il nodo minimo e identifica il nuovo nodo minimo all'interno della heap
4) individua il nodo minimo di H e ristruttura la heap di conseguenza

Domanda n. 236
La procedura Fib-Heap-Decrease-Key(H,x,k)
1) decrementa di k il valore della chiave del nodo x della heap H
2) decrementa del valore k la chiave del nodo x della heap H, errore se k<key[x]
3) decrementa del valore k la chiave del nodo x della heap H, errore se k>key[x]
4) decrementa del nodo x il nodo k della heap H
5) decrementa al valore k la chiave del nodo x della heap H

Domanda n. 237

La procedura Fib-Heap-Decrease-Key(H,x,k)

1) *richiama la procedura Cut(H,x,y) che toglie x dalla heap H e lo sostituisce con y*
2) *non richiama nessun'altra procedura*
3) *richiama la procedura Cut(H,x,y) che toglie x dalla lista dei figli di y e lo aggiunge fra le radici di H*
4) *fa uso di una chiamata ricorsiva alla procedura Cascading-Cut(H,y)*
5) *richiama solo la procedura Cut(H,x)*

Domanda n. 238

Se D(n), grado massimo per un nodo di una heap di fibonacci con n elementi, é O(lg n) allora il tempo ammortizzato della procedura Extract-Min e'

1) $\Theta(n \lg n)$
2) $O(n \lg n)$
3) $O(\lg n)$
4) $\Omega(n \lg n)$
5) $\Theta(n)$

Domanda n. 239

Il numero di nodi contenuti in un sottoalbero radicato in un nodo x, size(x), cresce

1) *esponenzialmente in deg(x)*
2) *logaritmicamente in deg(x)*
3) *polinomialmente in deg(x)*
4) *linearmente in deg(x)*
5) *non é correlato a deg(x)*

Domanda n. 240

Se x é un nodo di una heap di fibonacci con deg(x)=k e $y_1, \dots, y_k$ ne sono i figli nell'ordine in cui erano stati collegati a x

1) $deg(y_1) = 0$ e $deg(y_1) \geq i$ $i=2, 3, \dots, k$
2) $deg(y_1) \geq 0$ e $deg(y_1) \geq i-2$ $i=2, 3, \dots, k$
3) $deg(y_1) > 0$ e $deg(y_1) \geq i$ $i=2, 3, \dots, k$
4) $deg(y_1) \geq 0$ e $deg(y_1) \geq deg(y_{i-1})$ $i=2, 3, \dots, k$
5) $deg(y_1) > 0$ e $deg(y_1) \leq i-1$ $i=2, 3, \dots, k$

Domanda n. 241

I numeri di fibonacci sono definiti come

1) $F_0=1, F_1=1, F_k = F_{k-1} * F_{k-2}$
2) $F_0=0, F_1=0, F_k = F_{k-1} + F_{k-2}$
3) $F_0=0, F_1=1, F_k = F_{k-1} - F_{k-2}$
4) $F_0=0, F_1=1, F_k = F_{k-1} + F_{k-2}$
5) $F_0=0, F_1=1, F_k = F_{k-1} + F_{k-2}$

Domanda n. 242

Per i numeri di fibonacci vale che

1) $F_{k+2} = F_k + F_{k+1}$ $\forall k \geq 0$
2) $F_k = F_{k-2} + F_{k-1}$ $\forall k \geq 2$

3) $F_{k+2} = 1 + \sum_{i=0}^{k} F_i \quad \forall k \geq 0$

4) $F_{k+1} = F_k + F_{k-1} \quad \forall k \geq 1$

5) $F_{k+2} = F_k * F_{k+1} \quad \forall k \geq 0$

Domanda n. 243

Se x é un nodo di una heap di fibonacci con deg(x)=k

1) $size(x) \geq F_{k+2} \geq \Phi^k$, con $\Phi^k = (1+\sqrt{5})/2$

2) $size(x) \geq F_{k+2} - \Phi^k$, con $\Phi^k = (1+\sqrt{5})/2$

3) $size(x) \geq F_{k+2} + \Phi^k$, con $\Phi^k = (1+\sqrt{5})/2$

4) $size(x) \geq F_{k+2} * \Phi^k$, con $\Phi^k = (1+\sqrt{5})/2$

5) $\Phi^k \geq size(x) \geq F_{k+2}$, con $\Phi^k = (1+\sqrt{5})/2$

Domanda n. 244

Il grado massimo D(n) di qualsiasi nodo in una heap di fibonacci con n nodi e'

1) *O(n)*
2) *O(n lg n)*
3) *O(lg n)*
4) *O(n lg k)*
5) *O(k lg n)*

Domanda n. 245

Un insieme disgiunto e'

1) *Un insieme in cui sono stati esplicitamente definiti dei sottinsiemi*
2) *Un insieme partizionato in sottinsiemi*
3) *Un insieme i cui sottinsiemi sono degli up-tree*
4) *Un insieme che non gode della proprietà di congiunzione*
5) *Un insieme che é stato staccato da un altro insieme*

Domanda n. 246

Le operazioni principali per gli insiemi disgiunti sono

1) *Extract-Min(x), Make-Set(x), Union(x,y)*
2) *Make-Set(x), Union(x,y), Delete(x,T)*
3) *Make-Root(x,T), Union(x,y), Delete(x,T)*
4) *Make-Set(x), Union(x,y), Find-Set(x)*
5) *Disjunct(x,T), Union(x,y), UpTree(x,T)*

Domanda n. 247

La funzione Make-Set(x) per insiemi disgiunti

1) *inizializza un nuovo insieme contenente il solo elemento x*
2) *crea un nuovo insieme vuoto puntato da x*
3) *prende l'elemento x da un insieme e crea un insieme disgiunto contenente x*
4) *crea l'insieme x*

Domanda n. 248

La funzione Union(x,y) per gli insiemi disgiunti

1) *unisce gli insiemi x e y in un unico insieme x* $\cup$ *y*
2) *crea un nuovo insieme contenente x e y*

3) verifica se x e y sono insieme disgiunti, se non lo sono li unisce
4) unisce gli elementi degli insiemi che contengono x e y, S e T rispettivamente, nell'unico insieme S ∪ T
5) unisce l'elemento x agli elementi dell'insieme y

Domanda n. 249
La funzione Find-Set(x) per insiemi disgiunti
1) trova l'insieme x fra tutti gli insiemi disgiunti
2) trova l'insieme x e verifica che sia disgiunto dagli altri
3) trova l'insieme a cui appartiene l'elemento x
4) trova l'insieme di cui x é la radice
5) trova x

Domanda n. 250
Gli up-tree possono rappresentare insiemi disgiunti
1) collegando i diversi insiemi disgiunti come rami diversi di un up-tree
2) associando un up-tree ad ogni insieme disgiunto
3) collegando i diversi insiemi disgiunti come foglie diverse di un up-tree
4) Facendo risalire verso l'alto nell'up-tree i diversi insiemi disgiunti
5) Associando una diversa radice dello stesso up-tree ad ogni insieme disgiunto

Domanda n. 251
In un up-tree
1) la radice é l'elemento più basso e la ricerca avviene verso l'alto
2) la radice contiene il rappresentante di un insieme, che é padre di se stesso
3) le foglie puntano direttamente alla radice
4) ogni elemento contiene un campo chiave e un puntatore al padre
5) ogni elemento punta solo al padre

Domanda n. 252
L'unione di due up-tree
1) si realizza facendo puntare dalla radice dell'albero che ha più nodi la radice dell'albero che ha meno nodi
2) si realizza inserendo per chiave crescente gli elementi dell'albero con meno nodi fra quelli dell'albero con più nodi
3) si realizza facendo puntare dalla radice dell'albero che ha meno nodi la radice dell'albero che ha più nodi
4) si realizza inserendo per chiave crescente gli elementi dell'albero con più nodi fra quelli dell'albero con meno nodi
5) si realizza collegando l'albero con meno nodi ad una foglia dell'albero con più nodi

Domanda n. 253
La procedura di compressione di cammini per up-tree
1) crea un nuovo up-tree contenente solo il cammino passato come parametro
2) serve nel corso della find-set
3) collega gli elementi del cammino passato come parametro riducendone così la lunghezza
4) invoca l'algoritmo di Dijkstra per identificare l'up-tree di cammini di lunghezza minima
5) fa puntare direttamente alla radice ogni nodo del cammino d'accesso al nodo dato

Domanda n. 254
Il rango (rank) associato ad ogni nodo x di un up-tree rappresenta

1) *il numero di figli di x*
2) *il limite superiore all'altezza di x*
3) *il limite superiore al numero di archi del cammino più lungo fra x e una foglia discendente*
4) *il numero complessivo di discendenti di x*
5) *il numero di antenati di x*

Domanda n. 255

Le procedure che modificano direttamente il rango di un nodo di un up-tree sono

1) *Make-set, link e union*
2) *Find-set e Link*
3) *Make-Set e Union*
4) *Make-Set, Link, Union e Find-set*
5) *Make-set e Link*

Domanda n. 256

La funzione definita come F(0)=1 e F(i)=$2^{F(i-1)}$

1) *é una funzione* $\Theta(2^n)$
2) *é una funzione che cresce molto lentamente*
3) *é tale per cui già F(5) é un valore che eccede tutti i numeri incontrati nella pratica normale*
4) *é una funzione* $O(2^n)$
5) *é una funzione che non serve a niente*

Domanda n. 257

Se i=log*(n)

1) *i é tale per cui F(i) = n*
2) *i é il più piccolo intero tale che F(i) • n*
3) *i é il più piccolo intero tale che log log ... log(n) • 1, log ripetuto i volte*
4) *i é il più piccolo intero tale che log log ... log(n) • 1, log ripetuto i volte*
5) *i é il più piccolo intero tale che log log ... log(i) • 1, log ripetuto n volte*

Domanda n. 258

La funzione log*

1) *é una funzione che cresce molto molto lentamente*
2) *é una funzione polinomiale*
3) *é una funzione tale per cui log*(n) • 5 per ogni numero n incontrato nella pratica normale*
4) *é una funzione tale per cui log*(5) • n per ogni numero n incontrato nella pratica normale*
5) *é una funzione* $\Theta(n/2)$

Domanda n. 259

La funzione di Ackerman A(i,j) é definita come

1) *A(1,j)=1 per j • 1, A(i,1)=A(i-1,2) per i • 1, A(i,j)=A(i-1, A(i,j-1)) per i,j > 1*
2) *A(1,j)=j per j • 1, A(i,1)=i per i > 1, A(i,j)=A(i-1, j) per i,j > 1*
3) *A(1,j)=j per j • 1, A(i,1)=A(i-1,j) per i > 1, A(i,j)=A(i-1,j-1) per i,j > 1*
4) *A(1,j)=2^j per j • 1, A(i,1)=A(i-1,2) per i > 1, A(i,j)=A(i-1, A(i,j-1)) per i,j > 1*
5) *A(1,j)=2^j per j • 1, A(i,1)=*2^i *per i > 1, A(i,j)=A(i-1*A(i,j-1)) per i,j > 1*

Domanda n. 260

L'inversa della funzione di Ackerman $\alpha(m,n)$**, per** $m \bullet n$**, é definita come**

1) α *(m,n)=1/A(m,n))*
2) α *(m,n)=i se i é il più piccolo intero tale per cui A(i,* $\lfloor m/n \rfloor$*) > log(n)*

3) α *(m,n)=i se i é il più piccolo intero tale per cui A(i,* $\lfloor m/n \rfloor$*) > n*

4) α *(m,n)=i se i é il più piccolo intero tale per cui A(m,n) > n*

5) α *(m,n)=i se i é il più piccolo intero tale per cui A(*$\lfloor m/n \rfloor$*) < log(n)*

Domanda n. 261

Per un insieme di up-tree vale che

1) per tutte le radici x di alberi, size[x] • $2^{rank[x]}$

2) per tutte le radici x di alberi, size[x] • rank[x]

3) per tutte le radici x di alberi, size[x] • rank[x]

4) per tutte le radici x di alberi, size[x] = $2^{rank[x]}$

5) per tutte le radici x di alberi, size[x] = rank[x]

Domanda n. 262

Dato una foresta di up-tree, una sequenza di m operazioni make-set, union e find-set, con *n* operazioni make-set, può essere eseguita in tempo

1) O(A(m,n))

2) O(m log n)*

3) O(α(m,n))

4) $O(2^n)$

5) O(A(m,log n))*

Domanda n. 263

Un grafo G e'

1) una rete di connessione di nodi

2) un insieme di archi e vertici

3) un insieme di nodi collegati

4) una rappresentazione cartesiana di una funzione

5) una coppia di insiemi V e E.

Domanda n. 264

In un grafo G=(V,E), un arco *a* $\in$ E e'

1) una coppia {u,v} di vertici, u,v $\in$ *V*

2) un collegamento fra due vertici u e v, u,v $\in$ *V*

3) una connessione fra una coppia {u,v} di vertici, u,v $\in$ *V*

4) una linea che connette due nodi u e v, u,v $\in$ *V*

Domanda n. 265

Un grafo G=(V,E) é diretto (orientato) se

1) gli archi sono rappresentati da delle frecce e non delle linee

2) l'insieme V é formato da vertici e non da nodi

3) gli archi sono coppie ordinate di vertici (u,v)

4) gli archi vanno sempre dal nodo di indice minore a quello di indice maggiore

5) gli archi puntano verso la radice del grafo

Domanda n. 266

Il grado di un vertice v di un grafo e'

1) il numero di archi che é necessario percorrere per andare dalla radice a v

2) il numero di figli di v

3) il numero di vertici connessi da un arco con v

4) il numero di vertici adiacenti a v
5) il numero di nodi di G

Domanda n. 267
Un cammino in un grafo G=(V,E) e'
1) un insieme di archi di E
2) una sequenza $v_1, v_2, \ldots, v_k$ di vertici di G tale che ogni coppia di vertici consecutivi v_i, v_i+1 sia adiacente
3) una sequenza $v_1, v_2, \ldots, v_k$ di vertici di G
4) una sequenza $v_1, v_2, \ldots, v_k$ di vertici di G tale che $v_1 = v_k$
5) un percorso fra vertici di G

Domanda n. 268
La somma dei gradi di tutti i vertici di un grafo G=(V,E) e'
1) uguale al doppio del numero dei vertici in V
2) uguale al numero degli archi in E
3) uguale alla somma degli archi in E
4) compresa fra $|E|$ e $2|E|$
5) uguale al doppio del numero degli archi in E

Domanda n. 269
In un grafo G=(V,E) un cammino elementare e'
1) un cammino in cui non ci sono vertici ripetuti
2) un cammino costituito da un unico arco
3) un cammino costituito da un unico nodo
4) un cammino che parte e ritorna allo stesso nodo
5) un cammino strutturalmente molto semplice

Domanda n. 270
In un grafo G=(V,E) un ciclo e'
1) un qualsiasi cammino che parte e rientra allo stesso nodo
2) un cammino costituito da due percorsi chiusi
3) un cammino che percorre tutti i nodi di G
4) un cammino elementare, tranne che per il primo vertice che coincide con l'ultimo
5) l'unico cammino che partendo dalla radice ne rientra

Domanda n. 271
Un grafo G=(V,E) é connesso se
1) qualsiasi coppia di vertici in V é unita da al più un cammino
2) qualsiasi coppia di vertici adiacenti ha un arco che li connette
3) qualsiasi coppia di vertici in V é unita da almeno un cammino
4) qualsiasi coppia di vertici in V é unita da esattamente un cammino
5) non esistono sottografi disconnessi di G

Domanda n. 272
Un sottografo G* di un grafo G=(V,E) e'
1) un sottinsieme dei vertici e degli archi di G
2) un grafo che ha tutti i vertici in V ma solo un sottinsieme degli archi in E
3) un grafo che ha tutti gli archi in E ma solo un sottinsieme dei vertici in V

4) *un sottinsieme connesso del grafo G*
5) *un grafo contenente solo i vertici più bassi del grafo G*

Domanda n. 273
Una componente connessa G*=(V*,E*) di un grafo G=(V,E) e'
1) *un sottografo connesso di G tale per cui non é possibile aggiungere vertici in G\G* senza perdere la connessione*
2) *un cammino connesso in G*
3) *un sottografo connesso massimale di G*
4) *un qualsiasi sottografo connesso di G*
5) *un sottografo connesso di G tale per cui l'aggiunta di un vertice in G\G* implica l'aggiunta di un arco in E\E**

Domanda n. 274
Un albero e'
1) *un grafo in cui é possibile identificare un nodo radice e dei nodi foglia*
2) *un sottografo connesso massimale di un grafo G=(V,E)*
3) *un sottografo connesso di un grafo G=(V,E) in cui é possibile identificare un nodo radice e dei nodi foglia*
4) *un insieme di cammini di un grafo G=(V,E)*
5) *un grafo connesso senza cicli*

Domanda n. 275
Una foresta e'
1) *una collezione di alberi*
2) *un albero disconnesso*
3) *un insieme di alberi disconnessi a coppie*
4) *un insieme di alberi con la radice comune*
5) *un insieme di alberi orientati*

Domanda n. 276
La lista di adiacenza di un vertice v di un grafo G=(V,E) e'
1) *una lista di tutti gli archi adiacenti a v*
2) *una lista di tutti i vertici di V per cui esiste almeno un cammino che li connette a v*
3) *una lista di tutti i vertici di V per cui esiste al più un cammino che li connette a v*
4) *una lista di tutti i vertici adiacenti a v*
5) *una lista di tutti i vertici di V per cui esiste esattamente un cammino che li connette a v*

Domanda n. 277
Lo spazio necessario per rappresentare tutte le liste di adiacenza di vertici di un grafo G=(V,E) con |V|=n e |E|=m e'
1) *Θ(n)*
2) *Θ (m+n)*
3) *Θ (n log m)*
4) *non polinomiale*
5) *$O(n^m)$*

Domanda n. 278
Una matrice di adiacenza per un grafo G=(V,E) e'
1) *una matrice delle liste di adiacenza di tutti i vertici in V*
2) *una matrice che elenca, per ogni vertice v in V, i vertici adiacenti a v*

3) una matrice che specifica le adiacenze del grafo G
4) una matrice M di variabili intere, in cui M[i,j] é pari al j-esimo vertice adiacente al nodo i
5) una matrice M di variabili booleane con una cella per ogni coppia di vertici, M[i,j]=1 sse l'arco {i,j} é nel grafo

Domanda n. 279
Lo spazio necessario per contenere la matrice di adiacenza di un grafo G con n nodi e'
1) $O(n^2)$
2) $\Theta(n \log n)$
3) $\Theta(n^2)$
4) $O(n \log n)$
5) $O(n)$

Domanda n. 280
Una ricerca per ampiezza (BFS) su un grafo G
1) percorre l'intero grafo G e ne definisce un albero di copertura
2) percorre una componente connessa G e ne definisce un grafo di copertura
3) percorre una componente connessa G e ne definisce un albero di copertura
4) percorre l'intero grafo G e ne definisce un grafo di copertura
5) percorre l'intero grafo G e ne definisce un cammino di copertura

Domanda n. 281
Una ricerca per ampiezza (BFS) su un grafo G
1) dato un vertice sorgente $s \in V$ calcola la distanza da s ad ogni vertice raggiungibile di G
2) data la radice r di G calcola la distanza da r ad ogni vertice raggiungibile di G
3) data la radice r di G calcola la distanza da r ad ogni vertice di G
4) calcola la distanza fra ogni coppia di nodi in V
5) dati due nodi $u,v \in V$ calcola la distanza fra u e v

Domanda n. 282
Durante una ricerca BFS lo stato assumibile da ciascun vertice può essere
1) non espanso, espanso ma non scoperto, scoperto
2) espanso, coperto, scoperto
3) coperto, scoperto, completo
4) interno, esterno, completo
5) non scoperto, scoperto ma non espanso, espanso

Domanda n. 283
Durante una ricerca BFS
1) alla sorgente s viene assegnata distanza 1, ai nodi adiacenti ad s distanza 2, a quelli non scoperti adiacenti a questi ultimi 3, ecc.
2) alla sorgente s viene assegnata distanza 0, ai nodi adiacenti ad s distanza 1, a quelli non scoperti adiacenti a questi ultimi 2, ecc.
3) alla sorgente s viene assegnata distanza 0, ai nodi connessi ad s distanza 1, a quelli non scoperti connessi a questi ultimi 2, ecc.
4) alla sorgente s viene assegnata distanza 0, ai nodi adiacenti ad s distanza 1, a quelli adiacenti a questi ultimi 2, ecc.

Domanda n. 284
Durante una ricerca BFS l'etichetta di ogni vertice v corrisponde a

1) *il minimo numero di archi che é necessario percorrere per andare da s a v*
2) *l'identificativo della componente connessa di v*
3) *il numero di archi nel cammino da s a v*
4) *la lunghezza del cammino più breve da s a v*
5) *il rango di v*

Domanda n. 285
L'algoritmo BFS
1) *inizializza la sorgente s e poi espande ricorsivamente l'albero*
2) *inizializza la sorgente s e poi espande ricorsivamente il grafo*
3) *inizializza tutti i vertici e poi procede ricorsivamente*
4) *inizializza l'albero di copertura e poi lo costruisce in modo connesso*
5) *inizializza lo stato dei vertici, inizializza s e poi espande iterativamente l'albero di copertura*

Domanda n. 286
Il tempo di CPU dell'algoritmo BFS e'
1) *O(V+E)*
2) *O(V*E)*
3) *Θ(V log E)*
4) *Θ(V*E)*
5) *O(V log E)*

Domanda n. 287
Il tempo di CPU dell'algoritmo BFS e'
1) *quadratico (rappresentando il grafo con liste di adiacenza)*
2) *lineare (rappresentando il grafo con liste di adiacenza)*
3) *quadratico (rappresentando il grafo con matrici di adiacenza)*
4) *lineare (rappresentando il grafo con matrici di adiacenza)*
5) *non polinomiale*

Domanda n. 288
Il sottografo G'=(V',E') dei predecessori costruito dalla procedura BFS applicata ad un grafo G a partire da una sorgente s
1) *é un grafo contenente tutti i vertici raggiungibili da s tale per cui per ogni v ∈ V' il cammino da s a v é il cammino minimo da s a v in G*
2) *é un grafo contenente tutti i nodi esplorati da BFS durante la ricerca e tale per cui {u,v}∈ É sse u,v ∈* **E**
3) *é un grafo contenente tutti i nodi esplorati da BFS durante la ricerca e tale per cui {u,v} ∈É sse esiste un cammino da s a v e da s a u in G*
4) *é un albero contenente tutti i vertici raggiungibili da s tale per cui per ogni v ∈ V' il cammino da s a v é il cammino minimo da s a v in G*
5) *é un albero contenente tutti gli antenati di ogni vertice v, così come identificati dalla procedura BFS quando applicata a G*

Domanda n. 289
La ricerca in profondità (DFS) applicata ad un grafo G
1) *percorre una componente connessa di G senza mai riconsiderare più volte uno stesso nodo*
2) *percorre una componente connessa di G effettuando backtrack su nodi già esplorati*
3) *percorre una componente connessa di G effettuando backtrack sul nodo sorgente s*
4) *percorre l'intero grafo G effettuando backtrack di reinizializzazione della ricerca*
5) *percorre l'intero grafo G in ordine di valori ascendenti delle chiavi associate ai nodi*

Domanda n. 290

La procedura DFS

1) ***Ha una fase di inizializzazione di tutti i vertici e una fase di esplorazione ricorsiva***
2) ***Ha una fase di inizializzazione della sorgente e una di esplorazione iterativa dell'intero grafo G***
3) ***ha una fase di inizializzazione di tutti i vertici, una di inizializzazione della sorgente e una di backtrack***
4) ***Ha una fase di inizializzazione di tutti i vertici e una di backtrack***
5) ***Ha una fase di esplorazione ricorsiva e una di backtrack***

Domanda n. 291

La procedura DFS(G,s) si appoggia alla procedura DFS-Visit(u) che

1) ***ad ogni invocazione inizializza un nuovo albero con radice in u***
2) ***esplora iterativamente il sottoalbero radicato in u***
3) ***effettua il backtracking nella ricerca DFS su G***
4) ***identifica il cammino minimo dalla sorgente s al nodo u***
5) ***visita con procedura DFS il nodo u***

Domanda n. 292

Al termine della procedura DFS applicata ad un grafo G=(V,E) ogni nodo v $\in$ V ha associato

1) ***niente***
2) ***una etichetta: la lunghezza del cammino minimo da s a v***
3) ***una etichetta: il tempo di visita di v***
4) ***una etichetta: il tempo di fine***
5) ***due etichette: tempo di visita e tempo di fine***

Domanda n. 293

Il tempo di CPU della procedura DFS e'

1) ***$\Theta(V \log E)$***
2) ***$\Theta(V+E)$***
3) ***$O(V \log E)$***
4) ***$\Theta(V*E)$***
5) ***$\Theta(E \log V)$***

Domanda n. 294

Il sottografo dei predecessori G'=(V',E') costruito dalla procedura DFS applicata a un grafo G

1) ***costituisce l'albero dei cammini minimi dalla sorgente s ad ogni nodo v di V***
2) ***costituisce l'albero dei cammini minimi dalla sorgente s ad ogni nodo della componente connessa di G in cui si trova s***
3) ***costituisce un cammino minimo dalla sorgente s ad ogni nodo v di V***
4) ***forma una foresta di sottoalberi DF***
5) ***forma un albero DF***

Domanda n. 295

Durante la procedura DFS ogni vertice v e'

1) ***coperto prima di f[v], scoperto fra f[v] e d[v], esploso dopo d[v]***
2) ***inesplorato prima di f[v] e esplorato dopo***
3) ***bianco prima di f[v] e grigio dopo***
4) ***bianco prima di f[v], grigio dopo e nero alla fine***

5) *bianco prima di d[v], grigio fra d[v] e f[v], nero dopo f[v]*

Domanda n. 296

Durante la procedura DFS i vertici grigi

1) *formano un albero binario, implementabile come una heap binomiale*
2) *formano una catena lineare, implementabile come uno stack*
3) *formano un ciclo, implementabile come uno stack*
4) *formano un sottoalbero dei predecessori, implementabile come una heap di Fibonacci*
5) *formano un sottografo generico di G, implementabile con una matrice di adiacenza*

Domanda n. 297

Il teorema delle parentesi suggerisce che

1) *la storia di inizio e fine visita dei veri nodi può essere rappresentata da una espressione ben formata*
2) *gli intervalli di inizio e fine visita dei diversi nodi sono disgiunti*
3) *l'intervallo di visita di un nodo u é contenuto propriamente in quello di un altro nodo v*
4) *l'intervallo di un nodo u discendente di v non interseca l'intervallo di v*
5) *l'intervallo della sorgente é il più stretto fra tutti gli intervalli associati ai noti*

Domanda n. 298

Il teorema del cammino bianco asserisce che in una foresta DFS di un grafo G=(V,E) un vertice v é discendente di un vertice u sse

1) *al tempo f[u] il vertice v é raggiungibile da u con un cammino di soli archi bianchi*
2) *al tempo d[u] il vertice v é raggiungibile da u con un cammino di soli archi bianchi*
3) *al tempo f[v] il vertice v é raggiungibile da u con un cammino di soli archi bianchi*
4) *al tempo d[v] il vertice v é raggiungibile da u con un cammino di soli archi bianchi*
5) *al tempo f[v] esiste un cammino di archi bianchi che connette u a v*

Domanda n. 299

A seguito dell'applicazione di una DFS ad un grafo G gli archi di G possono essere classificati come archi:

1) *bianchi, neri e grigi*
2) *red e black*
3) *dell'albero, all'indietro, in avanti, di attraversamento*
4) *all'indietro, in avanti e diagonali*
5) *dell'albero, in avanti, di backtrack*

Domanda n. 300

In una DFS di un grafo non orientato G, ogni arco di G

1) *é un arco dell'albero oppure un arco di backtrack*
2) *può essere un arco dell'albero, all'indietro, in a vanti o di attraversamento*
3) *é un arco dell'albero oppure un arco all'indietro*
4) *può essere un arco bianco, grigio o nero*
5) *é un arco non orientato*

Domanda n. 301

Un DAG é un grafo

1) *diretto che non contiene cicli diretti*
2) *diretto e connesso*
3) *diretto e orientato*
4) *diretto che non contiene cicli nella versione non orientata del grafo*

5) diretto che on contiene cammini minimi

Domanda n. 302
Un DAG può essere utilizzato per:
1) rappresentare cammini minimi su reti stradali
2) rappresentare precedenze fra eventi
3) memorizzare alberi DFS
4) identificare componenti connesse
5) memorizzare pagine su disco rigido

Domanda n. 303
Un ordinamento topologico di un DAG permette di
1) indurre un ordinamento totale dall'ordinamento parziale rappresentato nel DAG
2) specificare le relazioni topologiche esistenti fra i nodi del DAG
3) identificare le sottostrutture topologicamente ben definite presenti nell'ordinamento rappresentato dal DAG
4) sequenziare le attività rappresentate dal DAG
5) identificare un ordinamento parziale fra i nodi del DAG

Domanda n. 304
Un ordinamento topologico di un DAG é un ordinamento
1) polinomiale dei vertici, tale che per ogni coppia di vertici u e v del DAG viene identificato un verso per l'arco (u,v)
2) ricorsivo, che ordina lessicograficamente i vertici del DAG
3) non polinomiale, che costruisce un DAG a partire da un grafo non diretto e connesso
4) lineare dei vertici, tale che per ogni arco (u,v) del DAG, u appare prima di v nell'ordinamento
5) logaritmico, che identifica il vertice u topologicamente più vicino ad un vertice v dato

Domanda n. 305
La procedura Topological-Sort(G)
1) é una procedura ricorsiva di ordinamento topologico del grafo G
2) identifica un ordinamento topologico di un DAG G calcolando i tempi di fine visita f[v] per ogni vertice v del grafo
3) calcola i tempi di inizio e fine visita di ogni nodo del DAG G
4) introduce gli archi transitivi in G che permettono di rappresentare un ordinamento totale
5) costruisce una lista concatenata di vertici corrispondente all'ordinamento topologico

Domanda n. 306
Un grafo diretto G é aciclico
1) se può essere rappresentato con un DAG
2) sse una ricerca BFS su G non produce archi di attraversamento
3) sse una ricerca DFS su G non produce archi all'indietro
4) sse una ricerca DFS su G termina in tempo polinomiale
5) sse una ricerca BFS su G termina in tempo polinomiale

Domanda n. 307
In un DAG che rappresenta le relazioni di precedenza fra attività associate ai nodi, l'esistenza di un arco (u,v) significa che
1) l'attività u può iniziare solo quando v é completata

2) *esiste una relazione di ordinamento topologico fra u e v*
3) *l'attività v può iniziare solo quando u é completata*
4) *l'attività v non può iniziare prima dell'attività u*
5) *le attività u e v devono iniziare insieme*

Domanda n. 308
Il tempo di CPU dell'ordinamento topologico di un DAG G=(V,E) e'
1) *uguale a quello di DFS(G)*
2) *O(V)*
3) *O(V log E)*
4) *O(E log V)*
5) *O(V+E)*

Domanda n. 309
Per la dimostrazione di correttezza dell'ordinamento topologico di un DAG G=(V,E) si dimostra che
1) *il tempo di CPU di Topological-Order(G) é O(V+E)*
2) *se (u,v) ∈ E allora f[u] < f[v]*
3) *l'ordinamento topologico é transitivo rispetto all'ordinamento parziale indotto dal DAG*
4) *se (u,v) ∈ E allora f[u] > f[v]*
5) *se u ∈ V e v ∈ V allora (u,v) ∈ E*

Domanda n. 310
Una componente fortemente connessa di un grafo orientato G=(V,E) e'
1) *un sottinsieme di vertici di V tale che esiste un cammino elementare che li congiunge tutti*
2) *un insieme massimale di vertici U ⊆ V tale che esiste un cammino elementare che li congiunge tutti*
3) *un insieme massimale di vertici U ⊆ V tale che U é un insieme connesso*
4) *un sottinsieme U di vertici di V tale che se u ∈ U e v ∈ U allora ciascuno dei due vertici é raggiungibile dall'altro*
5) *un insieme massimale di vertici U ⊆ V tale che se u ∈ U e v ∈ U allora ciascuno dei due vertici é raggiungibile dall'altro*

Domanda n. 311
Un grafo GT=(V,ET) é il trasposto di un grafo G=(V,E) se
1) *V=V e ET = E*
2) *ET={(u,v):(v,u) ∈ E}*
3) *gli archi in ET sono i trasposti degli archi in E*
4) *gli archi in ET sono gli stessi di quelli in E ma con il senso di percorrenza rovesciato*
5) *i vertici in V sono estremi sia degli archi in E che di quelli in ET*

Domanda n. 312
Un grafo G e il suo trasposto GT
1) *hanno le stesse componenti fortemente connesse*
2) *hanno cli stessi archi*
3) *hanno gli stessi cammini elementari*
4) *hanno gli stessi alberi di copertura*
5) *hanno gli stessi DAG*

Domanda n. 313
L'algoritmo Strongly-Connected-Components(G), con G=(V,E)

1) trova in tempo O(V log E) le componenti fort.connesse di G
2) trova in tempo O(V+E) le componenti fort.connesse di G
3) trova in tempo O(E log V) le componenti fort.connesse di G
4) trova in tempo O(E log V) le componenti connesse di G
5) trova in tempo O(V log E) le componenti fort.connesse del trasposto di G

Domanda n. 314
Se due vertici u e v sono in una stessa componente fortemente connessa allora
1) nessun cammino esce da questa CFC
2) nessun cammino fra loro esce da questa CFC
3) nessuna altra componente connessa contiene u e v
4) esiste una CFC che contiene sia u che v
5) u appartiene al grafo G e v al trasposto di G

Domanda n. 315
In una DFS, i vertici di una stessa componente fortemente connessa
1) sono tutti foglie dell'albero DFS
2) sono tutti collegati fra loro da archi bianchi
3) sono posti tutti nello stesso albero DFS
4) sono posti in ordine inverso nell'albero DFS
5) sono tutti i vertici di G

Domanda n. 316
Un avo ϕ(u) di un vertice u é il vertice w
1) più lontano da u
2) visitato per primo nella DFS che espande w
3) che massimizza f[w]
4) che ha il maggior numero di discendenti, fra cui w
5) che ha come figlio il padre di u

Domanda n. 317
In un grafo orientato G=(V,E) l'avo ϕ(u) di un qualunque u in V in una qualunque visita in profondità di G e'
1) un antenato di u
2) il padre di u
3) il padre del padre di u
4) la radice dell'albero DFS (o BFS nel caso di ricerca in ampiezza)
5) nessuno dei casi precedenti

Domanda n. 318
In ogni visita in profondità di un grafo orientato G=(V,E), per ogni vertice u in V i vertici u e ϕ(u):
1) sono visitati in tempi successivi
2) sono vertici espansi
3) sono vertici esplosi
4) appartengono alla stessa DFS
5) appartengono alla stessa CFC

Domanda n. 319
In un grafo orientato G=(V,E) due vertici u e v in V appartengono alla stessa CFC sse
1) sono entrambi vertici bianchi

2) ***sono espansi a turno***
3) ***hanno lo stesso padre***
4) ***hanno lo stesso avo in una visita in profondità di G***
5) ***nessuno dei precedenti***

Domanda n. 320
La correttezza della procedura strongly-connected-components(G) viene dimostrata
1) ***per ricorsione***
2) ***per assurdo***
3) ***per induzione***
4) ***per deduzione***
5) ***per infrazione***

Domanda n. 321
Un albero di copertura minimo di un grafo G=(V,E)
1) ***un albero T che collega tutti i vertici del grafo G***
2) ***un albero T che connette tutti i vertici in V tale che la somma dei pesi associati agli archi di T sia minima***
3) ***un albero T che minimizza la somma dei pesi associati agli archi di T***
4) ***un albero T che copre il minor numero possibile di vertici di G***
5) ***un albero T=(V,E'), É $\subseteq$ E, tale che la somma dei pesi associati agli archi di T sia minima***

Domanda n. 322
L'algoritmo MST-Kruskal(G,w) utilizza le procedure
1) ***Make-Set, Find-Set, Union***
2) ***Make-Set, Delete-Set***
3) ***Find-Arc, Insert-Arc***
4) ***Make-Set, Find-Arc, Compute-Cost***
5) ***Make-Set, Union, Compute-Cost***

Domanda n. 323
L'algoritmo MST-Kruskal(G,w)
1) ***trova l'albero dei cammini minimi del grafo G***
2) ***trova l'albero di copertura minimo del grafo G***
3) ***trova un cammino minimo del grafo G***
4) ***trova tutti gli alberi di copertura del grafo G***
5) ***costruisce un albero che copre tutti i vertici in V***

Domanda n. 324
L'algoritmo MST-Kruskal(G,w)
1) ***costruisce un MST scegliendo sempre l'arco di costo minimo fra quelli non inseriti***
2) ***costruisce un MST scegliendo sempre l'arco di costo minimo fra quelli non inseriti che non chiuda un anello***
3) ***costruisce un MST scegliendo sempre l'arco di costo minimo fra i non inseriti che non chiuda un anello e che sia connesso al sottoalbero corrente***
4) ***costruisce un MST scegliendo sempre l'arco di costo minimo che non chiuda un anello***
5) ***costruisce un MST scegliendo sempre l'arco di costo massimo fra quelli non inseriti che non chiuda un anello***

Domanda n. 325
L'algoritmo MST-Kruskal(G,w)

1) *mantiene gli archi di G ordinati per costo decrescente*
2) *mantiene gli archi di G ordinati per costo non crescente*
3) *mantiene gli archi di G ordinati per costo non decrescente*
4) *mantiene gli archi di G ordinati per costo crescente*
5) *non necessita di particolari ordinamenti degli archi di G*

Domanda n. 326
Per provare la correttezza dell'algoritmo di kruskal applicato a un grafo G=(V,E) si considera una partizione del grafo G'=(V,E') corrente in due componenti non connesse G1 e G2 e si prova che
1) *l'unione di G1 e G2 fornisce l'albero di copertura cercato*
2) *(V,É ∪ e), e arco di costo minimo che unisce G1 e G2, é un sottografo di qualche MST*
3) *il grafo G'\G1 ∪ G2 é un sottografo della MST cercata*
4) *(V,É ∪ e), e arco di costo minimo che unisce G1 e G2, é un MST*
5) *(V,É ∪ e), e arco di costo minimo che unisce G1 e G2, é un sottografo di G*

Domanda n. 327
La complessità in tempo dell'algoritmo di kruskal e'
1) *O(V) + O(E log E)*
2) *O(E log* E)*
3) *Θ(V log E)*
4) *Θ (E log V)*
5) *O(E log E)*

Domanda n. 328
L'algoritmo MST-Prim(G,w,r) utilizza le procedure
1) *Make-Set, Find-Set, Union*
2) *Extract-Min*
3) *Find-Arc, Extract-Min*
4) *Make-Set, Find-Arc, Compute-Cost*
5) *Make-Set, Union, Compute-Cost*

Domanda n. 329
L'algoritmo MST-Prim(G,w,r)
1) *trova l'albero dei cammini minimi del grafo G*
2) *trova l'albero di copertura minimo del grafo G*
3) *trova un cammino minimo del grafo G*
4) *trova tutti gli alberi di copertura del grafo G*
5) *costruisce un albero che copre tutti i vertici in V*

Domanda n. 330
L'algoritmo MST-Prim(G,w,r)
1) *costruisce un MST scegliendo sempre l'arco di costo minimo fra quelli non inseriti*
2) *costruisce un MST scegliendo sempre l'arco di costo minimo fra quelli non inseriti che non chiuda un anello*
3) *costruisce un MST scegliendo sempre l'arco di costo minimo fra i non inseriti che non chiuda un anello e che sia connesso al sottoalbero corrente*
4) *costruisce un MST scegliendo sempre l'arco di costo minimo che non chiuda un anello*
5) *costruisce un MST scegliendo sempre l'arco di costo massimo fra quelli non inseriti che non chiuda un anello*

Domanda n. 331

L'algoritmo MST-Prim(G,w,r)

1) *mantiene gli archi di G ordinati per costo decrescente*
2) *mantiene gli archi di G ordinati per costo non crescente*
3) *mantiene gli archi di G ordinati per costo non decrescente*
4) *mantiene gli archi di G ordinati per costo crescente*
5) *non necessita di particolari ordinamenti delegando alla Extract-Min la gestione dei costi degli archi di G*

Domanda n. 332

La complessità in tempo dell'algoritmo di Prim e'

1) $O(E + V \log V)$
2) $O(E \log^* E)$
3) $\Theta(V \log E)$
4) $\Theta(E \log V)$
5) $O(E \log E)$

Domanda n. 333

Il problema dei cammini minimi con sorgente singola, dato un grafo G=(V,E) con pesi w:E->R e un vertice s ∈ V, richiede di trovare

1) *per ogni arco e ∈ E il cammino di peso minimo originato in s e passante per e*
2) *per ogni vertice v ∈ V il cammino di peso minimo da s a v*
3) *per ogni peso w il cammino più breve di peso w originato in s*
4) *per ogni vertice v ∈ V tutti i cammini originati in s che terminano in v*
5) *per ogni CFC di G, il cammino di peso minimo che connette s alla CFC*

Domanda n. 334

Il problema dell'individuazione dei cammini minimi con sorgente singola su un grafo G=(V,E) può essere esteso a comprendere

1) *l'individuazione del cammino minimo fra una coppia di vertici u e v*
2) *l'individuazione del cammino minimo fra due CFC*
3) *l'individuazione del cammino minimo fra ogni coppia di archi in E*
4) *l'individuazione del cammino minimo fra una sorgente s ∈ V ed ogni altro vertice in V*
5) *l'individuazione dei cammini minimi fra tutte le coppie di vertici in V*

Domanda n. 335

L'individuazione dei cammini minimi con sorgente singola sul grafo G=(V,E) può essere effettuata sotto l'ipotesi che

1) *non esistano in G componenti fortemente connesse di peso negativo*
2) *non esistano in G archi di peso negativo*
3) *non esistano in G vertici di peso negativo*
4) *non esistano in G cicli di peso negativo*
5) *non esistano in G cammini di peso negativo*

Domanda n. 336

La rappresentazione interna dei cammini nel problema dell'individuazione dei cammini minimi con sorgente singola é analoga a quella

1) *delle CFC*
2) *delle heap di Fibonacci*
3) *delle heap binomiali*
4) *degli MST*

5) *degli alberi BFS*

Domanda n. 337

Nella rappresentazione interna dei cammini minimi, per ogni vertice v ∈ V viene mantenuto

1) *un predecessore π(v)*
2) *un figlio ϕ(v)*
3) *un puntatore alla sorgente σ(v)*
4) *un grafo G(v)*
5) *non so*

Domanda n. 338

L'individuazione dei cammini minimi con sorgente singola s sul grafo G=(V,E) comporta l'individuazione di un albero dei cammini minimi $G_\pi=(V_\pi, E_\pi)$ in cui

1) *V_π é l'insieme dei vertici raggiungibili da s in G*
2) *per ogni v ∈ V_π l'unico cammino semplice da s a v in G_π é il cammino minimo da s a v in G*
3) *per ogni v ∈ V_π l'unico cammino semplice da s a v in G_π é un cammino minimo da s a v in G*
4) *G_π forma un albero con radice in v*
5) *per ogni e ∈ E_π l'unico cammino semplice da s a v in G_π é il cammino minimo passante per e*

Domanda n. 339

L'algoritmo IntializeSingleSource, in Dijkstra:

1) *inizializza l'array contenente i dati del rafforzamento del costo del cammino minimo da s a ogni v ∈ V*
2) *inizializza l'array contenente i dati del rilassamento del costo del cammino minimo dalla sorgente s ad ogni vertice v*
3) *inizializza variabili d[v] contenenti, per ogni v, una stima del costo del cammino minimo espressa come un suo limite superiore*
4) *inizializza a 0 l'array d[v], per ogni v ∈ V, e le variabili poi usate in Dijkstra*
5) *inizializza le variabili relative alla sorgente s*

Domanda n. 340

L'algoritmo Relax, in Dijkstra

1) *per un arco (u,v) verifica se é possibile migliorare il cammino minimo per v passante per u*
2) *per un vertice v verifica se é possibile rilassare la stima del cammino di costo minimo che arriva a v*
3) *per la sorgente s, verifica se é possibile rilassare il cammino di costo minimo da s ad ogni v ∈ V*
4) *per il grafo G=(V,E), verifica se é possibile identificare un rilassamento dei costi dei cammini minimi da s*
5) *per il grafo G=(V,E), verifica se é possibile identificare un rilassamento del costo del cammino minimo da s a v ∈ V*

Domanda n. 341

L'algoritmo di Dijkstra, sorgente singola s

1) *percorre in maniera BFS il grafo G a partire dal vertice s*
2) *percorre in maniera DFS il grafo G a partire dal vertice s*

3) *associa ad ogni vertice v ∈ V una stima del rilassamento dell'arco (s,v) che porta in v*
4) *aggiorna ricorsivamente i puntatori del cammino minimo da s ad ogni v ∈ V*
5) *mantiene un insieme S di vertici v per cui il costo del cammino minimo da s a v é già stato determinato*

Domanda n. 342
La correttezza dell'algoritmo di Dijkstra, applicato ad un grafo orientato e pesato G=(V,E) e sorgente s, si prova dimostrando che
1) *l'algoritmo termina in un tempo di CPU che é O(V^2)*
2) *al termine dell'esecuzione é possibile raggiungere ogni vertice v ∈ V a partire da S*
3) *al termine dell'esecuzione l'algoritmo ha individuato ciò che doveva individuare*
4) *al termine dell'esecuzione si ha d[u]=δ(s,u) per ogni u ∈ V*
5) *l'asserto della correttezza di Dijkstra é rispettato per ogni possibile s ∈ V*

Domanda n. 343
La complessità computazionale dell'algoritmo di Dijkstra e'
1) *O(E^2)*
2) *O(V^2 + E)*
3) *O(V log E)*
4) *O(V^2)*
5) *O(V^2 log E)*

Domanda n. 344
L'algoritmo di Bellman - Ford risolve il problema
1) *dell'albero di copertura minimo di un grafo*
2) *dei cammini minimi fra tutte le coppie di vertici di un grafo*
3) *dei cammini minimi da una sorgente ad ogni altro nodo di un grafo*
4) *dei cammini minimi da una sorgente ad ogni altro nodo di un grafo, accettando archi di costo negativo*
5) *dell'individuazione delle componenti connesse di un grafo*

Domanda n. 345
L'algoritmo di Bellman - Ford
1) *é applicabile quando tutti gli archi del grafo hanno costo negativo*
2) *permette di individuare cammini minimi anche in un grafo con archi di costo negativo*
3) *restituisce false se esistono cicli di costo negativo e true altrimenti*
4) *"restituisce un identificativo di ""nessuna soluzione"" nel caso esistano cicli di costo negativo"*
5) *é una versione meno efficiente dell'algoritmo di Dijkstra*

Domanda n. 346
L'algoritmo di Bellman - Ford
1) *si basa sulla stessa funzione Relax usata anche da Dijkstra*
2) *permette di individuare l'albero di copertura minimo anche in grafi con archi di costo negativo*
3) *permette di individuare in modo deterministico i camminimi minimi in un grafo, senza rilassamenti o approssimazioni*
4) *individua in modo randomizzato i cammini minimi in un grafo*
5) *é una procedura ricorsiva per l'individuazione dei cammini minimi*

Domanda n. 347

L'algortimo di Bellmann-Ford applicato a un grafo G=(V,E) restituisce

1) *true se G non contiene archi di costo negativo, false altrimenti*
2) *l'albero dei cammini minimi da s a ogni v in V*
3) *true se G non contiene archi di costo negativo, nel qual caso si ha che d[v] = δ(s,v) per ogni v $\in$ V, false altrimenti*
4) *false se ci sono archi negativi nel grafo, true se G non contiene archi di costo negativo, nel qual caso si ha che d[v] = δ(s,v) per ogni v $\in$ V*
5) *l'albero di copertura minimo del grafo G*

Domanda n. 348

La complessità dell'algoritmo di Bellmann - Ford e'

1) *O(V E)*
2) *O(V+E)*
3) *O(V log E)*
4) *O(E log V)*
5) *O(V^2)*

Domanda n. 349

Nel caso di un DAG é possibile calcolare i cammini minimi sfruttando

1) *un ordinamento lineare dei vertici di G*
2) *un ordinamento quadratico dei vertici di G*
3) *un ordinamento crescente dei vertici di G*
4) *un ordinamento topologico dei vertici di G*
5) *un ordinamento binario dei vertici di G*

Domanda n. 350

La complessità di DAG-shortest-path e'

1) *O(V E)*
2) *O(V log E)*
3) *O(E log V)*
4) *O(V^2)*
5) *O(V+E)*

Domanda n. 351

Il problema dei cammini minimi fra tutte le coppie, dato un grafo G=(V,E) con pesi w:E->R, richiede di

1) *trovare per ogni coppia u,v $\in$ V il minimo costo di un cammino da u a v*
2) *trovare per ogni coppia u,v $\in$ E il minimo costo di un cammino da u a v*
3) *trovare per ogni coppia u,v $\in$ V il minimo costo di un cammino da s a v passante per u*
4) *trovare per ogni coppia u,v $\in$ V il minimo costo di un cammino da s a u passante per v*
5) *trovare per ogni arco (u,v) $\in$ E il minimo costo di un cammino da s passante per (u,v)*

Domanda n. 352

La matrice dei predecessori $\Pi=\{\pi_{uv}\}$ calcolata assieme ai cammini minimi fra tutte le coppie di un grafo G=(V,E) é tale per cui

1) *π_{uv} é NIL se non c'é un cammino da u a v, altrimenti é un puntatore a un predecessore di v su di un cammino minimo da u*
2) *π_{uv} é NIL se u=v, altrimenti é un puntatore a un predecessore di v su di un cammino minimo da u*

3) π_{uv} *é NIL se u=v o se non c'é un cammino da u a v, altrimenti é un predecessore di v su di un cammino minimo da u*

4) π_{uv} *é NIL se u=v o se non c'é un cammino da u a v, altrimenti é un predecessore di v su di un cammino minimo in G*

Domanda n. 353

La riga i-esima della matrice dei predecessori identifica

1) un albero dei cammini minimi con radice in i
2) tutti i cammini minimi che conducono a i
3) tutti i cammini minimi passanti per i
4) tutti gli alberi di copertura contenenti i
5) un albero di copertura minimo radicato in i

Domanda n. 354

L'algoritmo di Floyd-Warshall risolve il problema

1) dell'albero di copertura minimo di un grafo
2) dei cammini minimi fra tutte le coppie di vertici di un grafo
3) dei cammini minimi da una sorgente ad ogni altro nodo di un grafo
4) dei cammini minimi da una sorgente ad ogni altro nodo di un grafo, accettando archi di costo negativo
5) dell'individuazione delle componenti connesse di un grafo

Domanda n. 355

L'algoritmo di Floyd-Warshall

1) é un algoritmo di programmazione dinamica
2) é un algoritmo greedy
3) é un algoritmo approssimato
4) é un algoritmo euristico
5) é un algoritmo ottimo

Domanda n. 356

L'algoritmo di Floyd-Warshall

1) accetta cicli di costo negativo
2) assume che tutti gli archi abbiano costo non negativo
3) assume che tutti gli archi abbiano costo non positivo
4) non fa assunzioni sui costi degli archi
5) accetta archi di peso negativo ma non cicli negativi

Domanda n. 357

Il passo ricorsivo implementato da Floyd-Warshall per estendere un cammino minimo da s a t passando solo per i nodi $v_1, \ldots, v_i$ e':

1) $d_{st}(k) = min(d_{st}(k-1), d_{sk}(k-1))$ *(se k > 0)*
2) $d_{st}(k) = d_{st}(k-1) + d_{kt}(k-1)$ *(se k > 0)*
3) $d_{st}(k) = min(d_{st}(k-1), d_{sk}(k-1) + d_{kt}(k-1))$ *(se k > 0)*
4) $d_{st}(k) = min(d_{st}(k-1), d_{sk}(k))$ *(se k > 0)*
5) non c'é nessun passo ricorsivo

Domanda n. 358

L'algoritmo di Floyd-Warshall viene realizzato

1) con una chiamata ricorsiva a Floyd-Warshall(W)

2) con tre cicli for innestati
3) Basandosi sull'algoritmo Relax
4) con due cicli while innestati
5) in pseudocodice

Domanda n. 359
La complessità computazionale dell'algoritmo di Floyd-Warshall applicato a un grafo G=(V,E) e'
1) O(V^3)
2) Θ(V^3)
3) O(V^2)
4) O(V E)
5) O(V log E)

Domanda n. 360
I problemi decisionali sono la classe di problemi dove per ogni possibile ingresso un algoritmo deve
1) trovare la corrispondente soluzione di costo minimo
2) risolvere il problema
3) computare un'uscita corrispondente alla stringa di input ricevuta in ingresso
4) "scegliere una di due risposte possibili: ""si"" o ""no"""
5) individuare la decisione che risolve il problema

Domanda n. 361
La classe delle funzioni corrispondenti a problemi decisionali é quella delle funzioni
1) computabili del tipo f: N → {0,1}
2) f: N → {0,1}
3) computabili del tipo f: N → R
4) "computabili del tipo f: stringhe → {""si"",""no""}"
5) le funzioni non c'entrano niente

Domanda n. 362
Il problema del sottografo completo richiede:
1) Dato un grafo G, stabilire se G contiene un sottografo completo
2) Dati un grafo G di n vertici, stabilire se G contiene un sottografo completo
3) Dati un grafo G e un intero n, stabilire se il grafo G contiene un sottografo completo con n vertici
4) Dati un grafo G di n vertici, stabilire se G contiene un sottografo completo con n archi

Domanda n. 363
Il problema del cammino hamiltoniano per un grafo G richiede:
1) stabilire se esiste un ciclo che tocchi tutti i vertici di G una e una sola volta
2) stabilire se esiste un cammino che tocchi tutti i vertici di G una e una sola volta tornando alla fine al vertice di partenza
3) stabilire se esiste un cammino che tocchi tutti i vertici di G
4) stabilire se esiste un cammino elementare che tocchi tutti i vertici di G
5) stabilire se esiste un cammino che tocchi tutti i vertici di G una e una sola volta

Domanda n. 364
Il problema del cammino euleriano, dato un grafo G, richiede:
1) stabilire se esiste un cammino che tocchi tutti i vertici di G una e una sola volta

2) stabilire se esiste un cammino elementare che percorra tutti gli archi di G una e una sola volta
3) stabilire se esiste un cammino che percorra tutti gli archi di G una e una sola volta
4) stabilire se esiste un cammino che percorra tutti gli archi di G

Domanda n. 365
Il problema SAT richiede:
1) Data una k-CNF F stabilire se F é soddisfacibile, cioé se esiste un assegnamento di valori 0 e 1 alle variabili in F per cui il valore di F diventi
2) Data una CNF F stabilire se F é soddisfacibile, cioé se esiste un assegnamento di valori 0 e 1 alle variabili in F per cui il valore di F diventi 1
3) Data una CNF F stabilire se F é soddisfacibile, cioé se esiste un assegnamento ammissibile di valori 0 e 1 alle variabili in F
4) Data una CNF F stabilire se F é soddisfacibile, cioé se esiste un assegnamento di valori 0 e 1 alle variabili in F per cui F é soddisfatta

Domanda n. 366
Una k-CNF e'
1) una CNF definita su k congiunti
2) un insieme di k CNF ognuna singolarmente soddisfacibile
3) la disgiunzione di k CNF
4) una CNF in cui ogni congiunto ha k termini
5) non ricordo

Domanda n. 367
Un problema di ottimizzazione richiede
1) "di individuare quale fra ""si"" o ""no"" sia la soluzione del problema"
2) di trovare la soluzione ottima per il problema dato
3) di trovare il massimo o il minimo di una funzione
4) di utilizzare in modo ottimo le risorse disponibili
5) di ottimizzare i dati del problema

Domanda n. 368
Dato un problema di ottimizzazione PO e il corrispondente problema decisionale PD
1) PO e PD hanno la stessa complessità
2) PD é computazionalmente più complesso di PO
3) PO é computazionalmente più complesso di PD
4) é possibile ricondurre PO a PD tramite una ricerca binomiale su una soglia sul costo ottimo
5) non esiste sempre un problema decisionale corrispondente ad un problema di ottimizzazione

Domanda n. 369
Una istanza di un problema
1) é una specifica della struttura di come appare
2) é la specifica della modalità con cui passare il problema ad un algoritmo
3) é una implementazione del problema
4) è un suo caso particolare in cui vengono specificati tutti i suoi elementi costitutivi
5) é una stringa binaria

Domanda n. 370

Per codifica di un problema si intende:

1) *la corrispondenza fra l'insieme delle istanze del problema e un insieme di stringhe binarie: e: I -> {0,1}**
2) *corrispondenza fra il problema e l'insieme delle sue istanze*
3) *la modalità con cui si passa il problema all'algoritmo risolutivo*
4) *l'individuazione di un opportuno schema biettivo di corrispondenza fra gli elementi del problema e le strutture sintattiche rappresentative*
5) *l'implementazione del problema*

Domanda n. 371

Un problema decisionale PD é nella classe P

1) *se esiste un algoritmo che risolve un'istanza del problema PD in tempo polinomiale*
2) *se esiste un algoritmo che risolve qualsiasi istanza del problema PD in tempo polinomiale*
3) *se é possibile risolvere qualsiasi istanza del problema PD in tempo polinomiale*
4) *se esiste un algoritmo che risolve qualsiasi istanza del problema PD in tempo non deterministico polinomiale*
5) *se é facile da risolvere*

Domanda n. 372

Una funzione f:{0,1}* -> {0,1}* é calcolabile in tempo polinomiale

1) *se esiste un algoritmo polinomiale che, dato in input un qualsiasi x={0,1}*, produce come output {0,1}**
2) *"se esiste un algoritmo che, dato in input un qualsiasi x={0,1}*, produce come output ""si"" in tempo polinomiale"*
3) *"se esiste un algoritmo che, dato in input un qualsiasi x={0,1}*, produce come output ""no"" in tempo polinomiale"*
4) *se esiste un algoritmo polinomiale che, dato in input un qualsiasi x={0,1}*, produce come output f(x).*
5) *se il tempo di calcolo é del tipo n^x*

Domanda n. 373

Un problema decisionale PD é nella classe NP

1) *se non esiste un algoritmo che risolve qualsiasi istanza di PD in tempo polinomiale (rispetto alla dimensione della istanza).*
2) *se esiste un algoritmo che risolve qualsiasi istanza di PD in tempo non polinomiale (rispetto alla dimensione della istanza).*
3) *se esiste un algoritmo approssimato che risolve qualsiasi istanza di PD in tempo polinomiale (rispetto alla dimensione della istanza).*
4) *se esiste un algoritmo deterministico che risolve qualsiasi istanza di PD in tempo non polinomiale (rispetto alla dimensione della istanza).*
5) *se esiste un algoritmo non-deterministico che risolve qualsiasi istanza di PD in tempo polinomiale (rispetto alla dimensione della istanza).*

Domanda n. 374

Un algoritmo non deterministico é un programma

1) *che può anche contenere istruzioni del tipo goto {L1,...,Ln}*
2) *che può dare risultati diversi ad ogni esecuzione*
3) *che risolve problemi NP*
4) *che contiene chiamate alla funzione random()*
5) *che non é definito completamente*

Domanda n. 375
Per realizzazione di un algoritmo non deterministico A si intende
1) l'implementazione dell'algoritmo A
2) l'individuazione dello pseudocodice di A
3) ciascuna delle possibili diverse esecuzioni di A
4) l'applicazione di A ad un'istanza del problema che risolve

Domanda n. 376
Un algoritmo non deterministico A calcola una funzione f : N -> {0,1} se:
1) per ogni a ∈ N tale che f(a)=1 tutte le realizzazioni di A terminano
2) per ogni a ∈ N tale che f(a)=1 tutte le realizzazioni di A terminano restituendo 1
3) per ogni a ∈ N tale che f(a)=1 esiste una realizzazione di A che ritorna 1
4) per ogni a ∈ N tale che f(a)=0 esiste una realizzazione di A che ritorna 0
5) per ogni a ∈ N tale che f(a)=0 tutte le realizzazioni di A terminano restituendo 0

Domanda n. 377
Per il problema del ciclo euleriano esiste un algoritmo risolutore:
1) polinomiale
2) lineare
3) non deterministico polinomiale
4) logaritmico
5) non deterministico lineare

Domanda n. 378
Si sa che
1) $NP \subseteq P$
2) $P \subset NP$
3) $P = NP$
4) $P \neq NP$
5) $P \subset NP$

Domanda n. 379
***f* : N -> {0, 1} é riducibile polinomialmente a g : N -> {0, 1} se**
1) esiste una funzione h tale che per ogni x: f(x) = g(h(x))
2) esiste una funzione h, calcolabile in tempo polinomiale, tale che per ogni x: g(x) = f(h(x))
3) esiste una funzione h, calcolabile in tempo polinomiale, tale che per ogni x: f(x) = h(g(x))
4) esiste una funzione h, calcolabile in tempo polinomiale, tale che per ogni x: f(x) = g(h(x))
5) esiste una funzione h, calcolabile in tempo polinomiale, tale che per ogni x: h(x) = g(f(x))

Domanda n. 380
***f* : N -> {0,1} é NP-completo se e solo se:**
1) non é risolvibile in tempo polinomiale
2) f ∈ NP
3) per ogni g ∈ NP, g é riducibile polinomialmente a f
4) non esistono algoritmi, neanche non deterministici, che risolvono f in tempo polinomiale
5) f non é in P

Domanda n. 381
Per dimostrare la NP completezza di una funzione f dalla definizione di NP completezza si richiede di:
1) dimostrare che f é non deterministica polinomiale

2) dimostrare che f non é in P
3) dimostrare che f é riducibile polinomialmente qualunque altra funzione in NP
4) che la funzione é in NP
5) dimostrare che qualunque altra funzione in NP é riducibile polinomialmente alla funzione data

Domanda n. 382
Per dimostrare la NP completezza di una funzione f é necessario
1) mostrare che la funzione f é in NP
2) ridurre polinomialmente f a g, per ogni g ∈ NP
3) mostrare che g $<_p$ f per qualche problema g che é già noto essere NP completo
4) mostrare che f non é in P

Domanda n. 383
La prova di NP completezza del probelma del sottografo completo (CSP) può essere fatta
1) riducendo SAT a CSP
2) riducendo CSP a SAT
3) riducendo CSP a TSP
4) aumentando SAT a CSP
5) aumentando CSP a SAT

Domanda n. 384
Un algoritmo approssimato A ha scostamento garantito r(n) se per ogni input di dimensione n
1) il costo C della soluzione prodotta da A si discosta di r(n) dal costo C di una soluzione ottima: max(C-C*, C*-C) • r(n)*
2) il costo C della soluzione prodotta da A si discosta per un fattore r(n) dal costo C di una soluzione ottima: C*/C • r(n)*
3) il costo C della soluzione prodotta da A é diverso dal costo C di una soluzione al massimo r(n) volte su n prove*
4) il costo C della soluzione prodotta da A si discosta per un fattore r(n) dal costo C di una soluzione ottima: max(C/C*, C*/C) • r(n)*
*5) l'algoritmo A produce una soluzione C distante almeno r(n) dalla soluzione ottima C**

Domanda n. 385
L'errore relativo ε di un algoritmo approssimato A è dato da
1) |C-C|/C**
2) |C-C|/C*
3) |C-C|/C**
4) |C-C|/C*

Domanda n. 386
L'algoritmo A ha un errore relativo limitato da ε (n) se
1) |C-C|/C* ? ε(n)*
2) |C-C|/C* = ε (n)*
3) |C-C|/C* > ε (n)*
4) |C-C|/C* ? ε (n)*
5) |C-C|/C* > ε (n)*

Domanda n. 387
Uno schema di approssimazione (SdA) per un problema di ottimizzazione è

1) *una modifica di un'istanza del problema che rende un algoritmo approssimato A in grado di risolvere l'istanza con errore relativo limitato da ε.*
2) *un problema decisionale associato a quello da risolvere che trasforma un'istanza del problema in una risolvibile con errore relativo limitato da ε.*
3) *un algoritmo approssimato A che prende in input un'istanza del problema e un valore $\varepsilon>0$ e risolve l'istanza con errore relativo limitato da ε.*
4) *una trasformazione dell'istanza di input risolvibile con errore relativo limitato da ε.*

Domanda n. 388

Uno schema di approssimazione polinomiale è uno SdA che

1) *risolve il problema con $\varepsilon > 0$ che cresce in modo solo polinomiale rispetto a n*
2) *usa un algoritmo di approssimazione A polinomiale che ha errore garantito massimo pari a ε*
3) *é in grado di risolvere in tempo polinomiale qualsiasi problema in P*
4) *é in grado di risolvere in tempo polinoimiale qualsiasi istanza di un problema in P*
5) *per qualsiasi $\varepsilon>0$ dato richiede un tempo polinomiale rispetto a n per risolvere il problema*

Domanda n. 389

Uno SdA pienamente polinomiale (fully polynomial approximation scheme) è uno SdA polinomiale

1) *il cui tempo di esecuzione è polinomiale sia per $1/\varepsilon$ che per la dimensione n dell'input dell'istanza.*
2) *il cui tempo di esecuzione é polinomiale per qualunque problema e per qualunque dimensione*
3) *il cui tempo di esecuzione é polinomiale anche per problemi in NP*
4) *il cui tempo di esecuzione è polinomiale sia per ε che per la dimensione n dell'input dell'istanza.*
5) *che é polinomiale sia nel tempo di esecuzione che nell'occupazione di memoria*

Domanda n. 390

Il problema vertex cover, dato un grafo G=(V,E), richiede di:

1) *trovare il massimo numero di archi che abbiano un estremo in uno dei vertici in V*
2) *trovare il minimo numero di archi tale per cui ogni vertice in V abbia almeno un arco uscente*
3) *trovare un sottinsieme S di dimensione minima dei vertici in V, tale per cui ogni arco abbia un vertice in S*
4) *trovare il massimo numero di vertici in V tale per cui gli archi restino coperti*
5) *coprire tutti i vertici di G con un cammino hamiltoniano*

Domanda n. 391

Un algoritmo approssimato per vertex cover richiede di

1) *scegliere l'arco di costo massimo, togliere gli altri archi incidenti sui suoi estremi, poi iterare*
2) *scegliere l'arco di costo minimo, togliere gli altri archi incidenti sui suoi estremi, poi iterare*
3) *scegliere un vertice a caso, coprirlo con un arco uscente e iterare*
4) *scegliere un arco a caso, togliere gli altri archi incidenti sui suoi estremi, poi iterare*
5) *scegliere il vertice di grado massimo, coprirlo con un arco uscente e iterare*

Domanda n. 392

L'algoritmo approssimato presentato a lezione per il vertex cover ha rapporto limite pari a:

1) *2*

2) *1*
3) *n*
4) *log n*
5) *1/ε*

Domanda n. 393
Il problema del commesso viaggiatore (TSP), dato un grafo completo non orientato pesato G=(V,E,w), con costi interi non negativi, si chiede di:
1) *trovare un albero di copertura di costo minimo*
2) *trovare un cammino hamiltoniano di costo minimo*
3) *trovare un sottografo completo di costo minimo*
4) *trovare un viaggio di costo minimo*
5) *trovare un ciclo hamiltoniano su G di costo minimo*

Domanda n. 394
L'ipotesi della disuguaglianza triangolare per i costi associati agli archi di un grafo pesato assume che
1) *è sempre più economico andare direttamente da un nodo u a un nodo v direttamente piuttosto che passando per nodi intermedi w*
2) *ogni cammino da u a v ne ha sempre un altro u-w, w-v di costo uguale*
3) *per ogni cammino diretto da u a v ne esiste sempre uno indiretto alternativo*
4) *una triangolarizzazione dei costi associati a un cammino hamiltoniano non può portare a soluzioni di costo inferiore*

Domanda n. 395
Un algoritmo di approssimazione per il TSP per un grafo G
1) *é basato su una componente connessa massima per G*
2) *é basato su un albero di copertura minimo per G*
3) *é basato su un cammino minimo per G*
4) *é basato su un cammino hamiltoniano minimo per G*
5) *é basato su un grado minimo per G*

Domanda n. 396
L'algoritmo approx-TSP-tour ha grado limite pari a
1) *2*
2) *1*
3) *n*
4) *log n*
5) *ε*

Domanda n. 397
Un generico algoritmo euristico
1) *garantisce di trovare la soluzione ottima di un problema di ottimizzazione*
2) *garantisce di trovare una soluzione con errore limitato per un problema di ottimizzazione*
3) *garantisce di trovare una buona soluzione di un problema di ottimizzazione*
4) *non garantisce niente sulla qualità delle soluzioni trovate*

Domanda n. 398
Un algoritmo euristico costruttivo per un problema di ottimizzazione P
1) *costruisce una buone soluzione migliorando successivamente soluzioni peggiori*
2) *costruisce una buona soluzione analizzando l'intorno di altre soluzioni*

3) costruisce una soluzione completando via soluzioni parziali
4) risolve costruttivamente il problema P

Domanda n. 399
L'insieme di vicinanza di una soluzione data
1) specifica un sottinsieme di soluzioni ammissibili
2) specifica quali soluzioni hanno un costo simile a quello della soluzione data
3) é definito sulla base di una funzione di vicinanza
4) specifica quali soluzioni sono ottenibili facilmente a partire da quella data

Domanda n. 400
Un algoritmo euristico di ricerca locale
1) parte da una soluzione vuota e la completa sulla base dell'insieme di vicinanza
2) parte da una soluzione inammissibile e la rende ammissibile
3) parte da una soluzione del problema decisionale e la converte in una soluzione del problema di ottimizzazione
4) parte da una soluzione ammissibile e passa successivamente attraverso altre funzioni ammissibili

Domanda n. 401
Semplici ricerche locali per il problema del TSP sono:
1) 2-eur, 3-eur
2) 2-opt, 3-opt
3) 2-opt, 2-eur
4) Lin Pernigan e 2-opt
5) Simplesso e Lin Pernigan

DOMANDE APERTE

Chiarire la differenza tra problema computazionale e di istanza di un problema computazionale. Esemplificare.

Chiarire la differenza tra complessità computazionale di un problema e per costo computazionale di un algoritmo. Esemplificare.

Definire il concetto di strutture dati astratte. Esemplificare.

Chiarire la differenza tra strutture dati astratte e concrete. Esemplificare.

Perché la complessità di un problema e il costo computazionale di un algoritmo vengono misurate usando la notazione asintotica?

Quali vantaggi introduce l'approccio divide et impera ? È sempre applicabile ? Quando lo è ? Come si analizza il costo computazionale di un algoritmo divide et impera ?

Quali sono le principali tecniche per la risoluzione di equazioni ricorsive ? Descriverle brevemente.

In quali casi è possibile ordinare una sequenza di n numeri con un costo computazionale inferiore a nlog(n) (inferiore significa o-piccolo)?

Definire il concetto di priority queue e descriverne le principali implementazioni con particolare attenzione al costo computazionale delle singole operazioni.

Quali sono le tecniche principali per memorizzare grafi in memoria centrale ? Quando è più conveniente una tecnica rispetto ad un altra ?

Quali sono le tecniche principali per memorizzare alberi in memoria centrale ? Quando è più conveniente una tecnica rispetto ad un altra ?

Come definiresti un algoritmo di ordinamento comparison sort ? Descrivi un algoritmo di ordinamento comparison sort e uno non comparison sort.

È possibile ordinare n numeri in tempo o(n)? perché ?

Nell'algoritmo select visto a lezione si suddividono i numeri in input in gruppi di cardinalità 5. Come cambia il costo computazionale dell'algoritmo se suddividiamo gli elementi in gruppi di cardinalità 3 ? In gruppi di cardinalità 7 ?

Tabelle hash