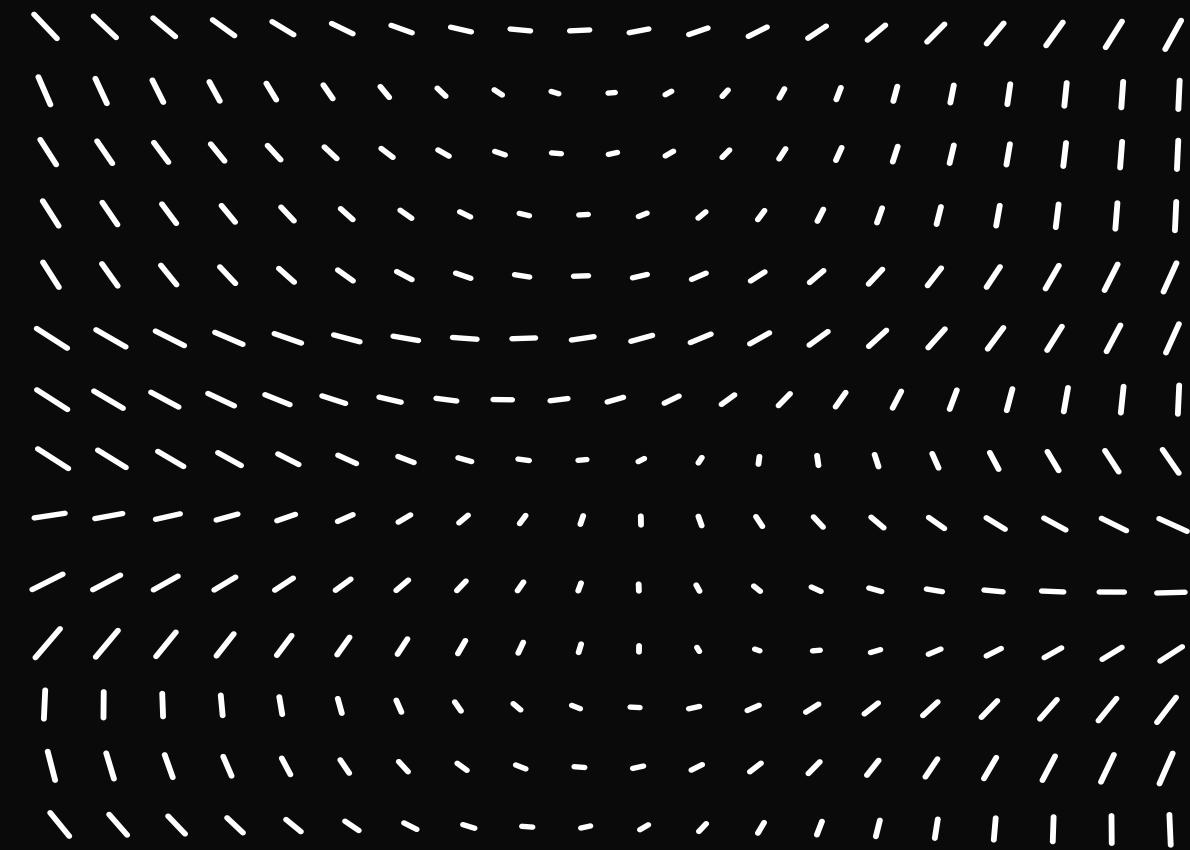


Esami

Algoritmi



APPELLO 1/19

Problema 1 Analisi algoritmo

Si considerino i metodi Java di seguito illustrati.

```
static int ternsearch(int a[], int v, int start, int end) { // array a ordinato in senso non decrescente
    if (end - start < 2) return base(a, v, start, end);
    int onethird = (end - start)/3;
    int i = start + onethird;
    if(a[i] > v) return ternsearch(a, v, start, i-1);
    else if (a[i] == v) return i;
    else {
        int j = i + onethird;
        if(a[j] > v) return ternsearch(a, v, i+1, j-1);
        else if (a[j] == v) return j;
        else return ternsearch(a, v, j+1, end);
    }
}

static int base(int a[], int v, int start, int end) { // array a ordinato in senso non decrescente
    while(start <= end) {
        if(a[start] == v) return start;
        start++;
    }
    return -1;
}

static int ternsearch(int a[], int v) { // array a ordinato in senso non decrescente
    return ternsearch(a, v, 0, a.length-1);
}
```

Sviluppare, *argomentando adeguatamente* (il 50% del punteggio dell'esercizio sarà sulle argomentazioni addotte), quanto segue:

- (a) Determinare il costo temporale asintotico dell'algoritmo descritto da `ternsearch(int[], int)` in funzione della dimensione dell'input. [4/30]
- (b) Confrontare i costi temporali e spaziali di `ternsearch(int[], int)` con quelli di un'ordinaria ricerca binaria su un array ordinato. [2/30]

a) SIA n IL NUMERO DI ELEMENTI NELL'ARRAY

$$C(n) = C + O\left(\frac{n}{3}\right) = C + C + O\left(\frac{n}{9}\right) = \alpha C + O\left(\frac{n}{3^i}\right)$$

SI FERMA QUANDO $\frac{n}{3^i} < 2$ $\frac{n}{2} < 3^i$ $i > \log_3\left(\frac{n}{2}\right)$

$$\log_3\left(\frac{n}{2}\right) \cdot C + \text{cost} \approx O(\log_3 n)$$

b) RIGUARDO AI COSTI TEMPORALI, ASINTOTICAMENTE NON ABBIANO DIFF PER CHÈ $O(\log_3 n)$ DIFFERISCE DA $O(\log_2 n)$ PER UNA SOLA COST.

A LIVELLO SPAZIALE ABBIANO CHE LO SPAZIO AGGIUNTIVO È DATO DALLE CHIARATE RICORSIVE E QUINDI DAI RECORD DI ATTIVAZIONE.

$\log_3 n$ RECORD DI ATT VS $\log_2 n$ RECORD DI ATT

ANCHE A LIVELLO SPAZIALE DIFFERISCONO DI UNA SOLA COST.

Problema 3 Algoritmi

- (a) Con riferimento alle tavole hash con *linear probing*, impiegate per implementare una mappa, descrivere accuratamente la gestione delle operazioni di inserimento e di eliminazione di chiavi. [4/30]
- (b) Progettare un algoritmo (codice o pseudo-codice) che dati un insieme di n interi e un intero k ($1 \leq k \leq n$), determini i k interi più piccoli dell'insieme in tempo $o(n \log n)$, se $k \in o(n)$ (tempo asintoticamente migliore di $n \log n$ se k è asintoticamente più piccolo di n). [4/30]
- (c) Progettare un algoritmo (codice o pseudo-codice) che, dato un grafo semplice G , stabilisce se G è una foresta. [4/30]

a) INSEMENTO:

L'INSEMENTO NELLA MAPPA AVVIENE ASSEGNAVNO UNA PSEUDO CHIAVE ALLA ENTRY IN INPUT IN FUNZIONE DELLA SUA CHIAVE USANDO UNA FUNZIONE DI HASH. TALE PSEUDO CHIAVE RAPPRESENTA LA POSIZIONE NELL'ARRAY IN CUI INSERIRE LA ENTRY

SE IL FATTORE DI CARICO $\lambda = \frac{\# \text{ENTRY} + 1}{\text{DIM. HASH T.}} > \frac{1}{2}$ NEL MOMENTO IN CUI

DEVO INSERIRE LA NUOVA ENTRY ALLORA SI RADOPPIA LA DIM. DELLA HASH TABLE PER POI REINSERIRE TUTTE LE ENTRY GIÀ PRESENTI PIÙ LA NUOVA

NEL CASO DI UNA COLLISIONE, RISOLVENDO CON IL LINEAR PROBING, AVANZO DI UNA CELLA FINCHÉ NON NE TROVO UNA LIBERA, QUI INSERISCO

ESIMINAZIONE:

PER L'ESIMINAZIONE DELLA ENTRY VISITO LA CELLA INDICATA DALLA FUNZIONE DI HASH IN BASE ALLA CHIAVE DELLA ENTRY DA ELIMINARE. AVANZO DI CELLA IN CELLA FINCHÉ NON TROVO LA ENTRY DA ELIMINARE. SE LA TROVO RESTITUISCO LA ENTRY E NELLA TAVOLA LA SOSTITUISCO CON LA SENTINELLA, ALTRIMENTI SE UNA CELLA VUOTA LA ENTRY È

b) FIRST_K (ARRAY a, INT k)

```
MIN_MEAP = ARRAY_TO_MEAP(a);
ARRAY FIRST_K[k];
FOR (INT i=0; i <= n; i++)
    FIRST_K[i] = REMOVE_MIN(MIN_MEAP);
RETURN FIRST_K;
```

}

c) BOOLEAN ISFOREST (GRAPH G) {
 INT FOREST=0;
 FOR NODO IN G.LISTA_NODI DO {
 IF (NODO. STATO == UNEXPLORED) {
 BOOL. CICLO = DFS(NODO);
 IF (CICLO == TRUE) FOREST++;
 RETURN FALSE;
 }
 }
 IF (FOREST > 1) RETURN TRUE;
}

 BOOLEAN DFS (NODO) {
 NODO. STATO = EXPLORING;
 FOR ADT IN NODO. ADIACENTI DO {
 IF (ADT. STATO == UNEXPLORED)
 RETURN TRUE || DFS(ADT);
 ELSE IF (ADT. STATO == EXPLORING)
 RETURN FALSE;
 }
 RETURN TRUE;
}

Problema 4 Sottoalbero completo massimo

Dato un albero binario t , progettare un algoritmo (codice o pseudo-codice) che determini il numero di nodi del più grande sottoalbero completo presente in t .

Si richiamano per comodità le definizioni di albero binario completo e di sottoalbero. Un albero binario è detto *completo* se tutti i suoi nodi interni hanno arită (numero figli) due e tutti i rami hanno la stessa lunghezza. Dato un nodo v di un albero t (non necessariamente binario), il *sottoalbero di t radicato in v* è l'albero indotto da tutti i nodi di t che sono discendenti di v (incluso). [6/30]

APPLICO UNA VISITA POSTORDINE

```
INT MAX_COMPLETE(BINARYTREE x) {  

    IF (x == NULL) RETURN 0;  

    INT LEFT = MAX_COMPLETE(x → LEFT);  

    INT RIGHT = MAX_COMPLETE(x → RIGHT);  

    IF (LEFT < 0 && |LEFT| > |RIGHT|) RETURN LEFT;  

    ELSE IF (LEFT < 0 && |LEFT| < |RIGHT|) RETURN RIGHT;  

    ELSE IF (LEFT < 0 && LEFT == RIGHT) RETURN LEFT;
```

$$\frac{n}{2^{2^i}} = 1 \rightarrow n = 2^{2^i} \quad 2^i = \log_2 n \quad i = \frac{\log_2 n}{2}$$

$$C(n) = 3C_i + C\left(\frac{n}{2^i}\right) = 3i \cdot c_i + C\left(\frac{n}{2^{2^i}}\right) = \frac{3 \log_2 n}{2} + 1 \rightarrow O(\log_2 n)$$

Problema 1 Analisi algoritmo

Si considerino i metodi Java di seguito illustrati, assumendo che i parametri x ed y siano sempre interi non negativi.

```
static int funct1(int x) {
    if(x <= 1) return x;
    return funct1(funct1(x/2)) + 1;
}

static long funct2(int y) {
    if(y <= 1) return y;
    return funct2(y-1)*funct1(y);
}
```

Sviluppare, *argomentando adeguatamente* (il 50% del punteggio dell'esercizio sarà sulle argomentazioni addotte), quanto segue:

(a) Determinare il costo temporale asintotico dell'algoritmo descritto da `funct1(int)` in funzione della dimensione dell'input $z_1 = |x|$ (z_1 è la dimensione dell'input e x è l'input). [3/30]

(b) Determinare il costo temporale asintotico dell'algoritmo descritto da `funct2(int)` in funzione della dimensione dell'input $z_2 = |y|$ (z_2 è la dimensione dell'input e y è l'input). [3/30]

a) LA CHIAMATA INTERNA DI FUNCT1 DIMINUISCE L'INTERO IN INPUT FINO A QUANDO $x \leq 1$. A QUEL PUNTO LA FUNZIONE RITORNA x E VIENE FATTA UN'ULTERIORE CHIAMATA CON $x \leq 1$ CHE TERMINA SUBITO.

FACCIAO CHIAMATE INTERNE FINO A QUANDO $\frac{x}{2^i} = 1 \rightarrow i = \log_2 x$

$O(\log_2 x)$ CHIAMATE INTERNE PIÙ UN'ULTERIORE A FUNCT1

$$z = |x| = \log_2 x \rightarrow x = 2^z \quad O(\log_2 x) = O(\log_2 z)$$

b) CHIAMO FUNCT2 FINO A CHE $y \leq 1$, FACCO $y-1$ CHIAMATE SU FUNCT2. A CHIAMATA DI FUNCT2 NE FACCO UNA A FUNCT2 CHE COSTERÀ $\log_2 y$. AVRÒ:

$$\log_2 y + \log_2 (y-1) + \log_2 (y-2) + \dots = \log_2 y != y \log_2 y$$

$$z = |y| = \log_2 y \rightarrow y = 2^z \quad O(2^z \cdot z)$$

Problema 3 Algoritmi

- (a) Il prof. di algoritmi ti chiede all'esame di analizzare il costo computazionale di un algoritmo di cui ti fornisce il codice C. Purtroppo, guardando il codice, non si capisce assolutamente nulla, per cui sembra impossibile eseguire l'analisi. Pur di mostrare un risultato, magari imperfetto, decidi di valutare empiricamente il costo dell'algoritmo eseguendolo molte volte su diversi input. Più in particolare...

Descrivi la metodologia che useresti per stimare il costo asintotico empiricamente.¹

[4/30]

- (b) Progettare un algoritmo (codice o pseudo-codice) che, dato un *grafo diretto* G , stabilisca se G è aciclico o meno.

[4/30]

- (c) Descrivere la rappresentazione di alberi binari mediante array, chiarendone pregi e difetti (70% del punteggio). In quali casi è estremamente conveniente? (30% del punteggio)

[4/30]

a) SI PUÒ ESEGUIRE PIÙ VOLTE CON DIF DI INPUT X E CALCOLARE IL TEMPO MEDIO.

ESEGUE CON INPUT 2x E CALCOLA IL TEMPO MEDIO

ESEGUE CON INPUT 3x E CALCOLA IL TEMPO MEDIO

OSSERVO I RISULTATI E NOTO SE CI SONO VARIAZIONI SIGNIFICATIVE
TIPO SE CON X ABBIAMO TEMPO T E CON 2X ABBIAMO 2T ALLORA
IL COSTO È LINEARE

b) SE HO UN ARCO BACK ALLORA NO UN CYCLO

```
BOOL ISACYCLIC(GRAPH g){  
    FOR(NODE IN g){  
        IF(NODE.STATUS == UNEXPLORED){  
            BOOL FOUND_BACK = D-DFS(NODE);  
            IF(FOUND_BACK == TRUE)  
                RETURN FALSE;  
        }  
    }  
    RETURN TRUE;  
}
```

```
BOOL D-DFS(NODE n){  
    n.STATUS = EXPLORING;  
    FOR(EDGE IN n.OUT-EDGES){  
        IF(EDGE.DESTINATION.STATUS == UNEXPLORED){  
            BOOL FOUND_BACK = D-DFS(EDGE.DESTINATION);  
            IF(FOUND_BACK == TRUE)  
                RETURN TRUE;  
        } ELSE IF(EDGE.DESTINATION.STATUS == EXPLORED)  
            RETURN TRUE;  
    }  
    n.STATUS = EXPLORED;  
    RETURN FALSE;  
}
```

c) NELLA RAPPRESENTAZIONE DI ALBERI BINARI MEDIANTE ARRAY IL VALORE DELL'ENTRY DELL'ALBERO VIENE INSERITO DIRETTAMENTE NELLA CELLA DELL'ARRAY. LA RADICE DELL'ALBERO SI TROVA IN POS 0 ED I FIGLI IN POS $2 \cdot i + 1$ (SX) E $2 \cdot i + 2$ (DX). QUINDI PER ENTRY IN POS i AVREMO FIGLIO SX IN POS $2i + 1$ E FIGLIO DX IN POS $2i + 2$

UN PREGIO DELL'ARRAY È CHE, DATA UNA ENTRY IN POS i , TROVO IL PADRE FACENDO $i/2$ O $i/2 - 1$ SE i PARI, ACCEDENDOVI IN TEMPO COST. INOLTRE SI PUÒ TRASFORMARE UN ALBERO BINARIO IN UN MEAP CON COSTO LINEARE

UN DIFETTO È IL COSTO SPAZIALE CHE, NEL CASO DI UN ALBERO CON ALTEZZA MASSIMA, È ESPONENZIALE, DOVENDO ALLOCARE CELLE ANCHE PER LE ENTRY NON PRESENTI.

QUESTA RAPPRESENTAZIONE È CONVENIENTE PER ALBERI COMPLETI DATO CHE IL DIFETTO SUL COSTO SPAZIALE DECADE. CONVIENE ANCHE PER GLI MEAP

Problema 4 Conteggio nodi k -ari di un albero

Dato un albero generico t (nessuna restrizione sul numero di figli di un nodo) e dato un intero positivo k , progettare un algoritmo (codice o pseudo-codice) che determini il numero di nodi in t che hanno esattamente k figli. Assumere che t sia rappresentato con una struttura bilineare, in cui ogni nodo possiede un riferimento a `firstChild` (primo figlio) e a `nextSibling` (prossimo fratello). [6/30]

```
INT K_CHILDREN(TREE t, INT k){  
    TREE_NODE n = t->RADIX;  
    INT RET=0;  
    LINKED_LIST BFS_VISIT,  
    WHILE (!BFS_VISIT. ISEMPTY()) {  
        TREE_NODE CHILD = n->FIRSTCHILD;  
        IF (CHILD == NULL) CONTINUE;  
        INT SONS=1;  
  
        WHILE (CHILD->NEXT SIBLING != NULL) {  
            CHILD = CHILD->NEXT SIBLING;  
            SONS++;  
        }  
        IF (SONS == k)  
            RET++;  
    }  
    RETURN RET;
```

Problema 1 Analisi algoritmo

Si considerino i metodi Java di seguito illustrati.

```
static void p(int i, int[] a) {
    int v = a[i];
    while(++i < a.length)
        if(v > a[i]) a[i-1]=a[i];
        else break;
    a[i-1] = v;
}

static int[] s(int[] a, int i) {
    if(i == a.length - 1) return a;
    p(i, s(a, i+1)); return a;
}

static int[] s(int[] a) {
    if(a.length > 0) return s(a, 0);
    else return a;
}
```

Sviluppare, argomentando adeguatamente (il 50% del punteggio dell'esercizio sarà sulle argomentazioni addotte), quanto segue:

- (a) Determinare il costo temporale asintotico dell'algoritmo descritto da `s(int[])` in funzione di z , dimensione dell'input. Non usare simboli che non sono definiti (ad esempio: n non è definito). [4/30]
- (b) Il metodo `s(int[])` descrive un celebre algoritmo. Quale? (Spiegare perché). [2/30]

a) SIA n IL # ELEMENTI NELL'ARRAY A
NEL CASO PEGGIORE VENGONO SVOLTE $O(n)$ CHIAMATE DI S ED
OGNUA DI QUESTE FA UNA CHIAMATA A P. RISULTA CHE OGNI CHIAMATA
DI S HA UN COSTO TOTALE DECRESCENTE DOVUTO AL CRESCERE DI i .
ABBIANO:

$$n + n-1 + n-2 + \dots + 1 = \sum_{i=0}^n i = \frac{n(n+1)}{2} = O(n^2) = O(z^2)$$

b) DESCRIVE L'INSERTION SORT.

L'ARRAY VIENE DIVISO IN 2 PARTI: UNA ORDINATA ED UNA NON ORDIN.
IN QUESTO CASO PARTIAMO DALLA FINE DELL'ARRAY. A DESTRA ABBIANO
L'ARRAY ORDINATO, A SX NO. AD OGNI CHIAMATA L'ARRAY ORDINATO VIENE
INCREMENTATO DI UNA CELLA E L'ORDINAMENTO VIENE RIPRISTINATO
FACENDO SCAMBI VERSO DX FINCHÉ IL NUOVO NUMERO NON SI TROVA
IN POS CORRETTA

$$C(n) = c + C(p(n)) + C(n-1) = i c + i C(n-1)$$

$$C(p(n)) = n$$

$$\begin{aligned} C(n) &= c_0 + n + C(n-1) = i c_0 + \sum_{j=0}^{i-1} n - j \\ &= c_0 + n + c_0 + (n-1) + C(n-2) = \\ &= i c_0 + \sum_{j=0}^{i-1} n - j + C(n-i) \end{aligned}$$

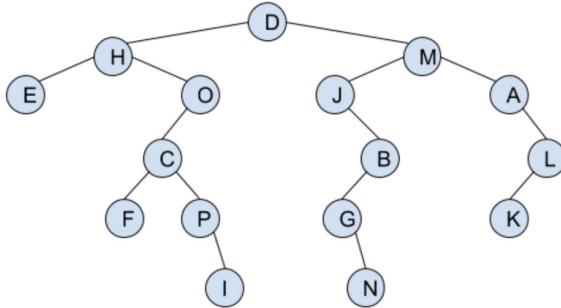
SI FERMA PER: $C(n-i) = C(0) \rightarrow (n-i) = 0 \rightarrow n = i$

$$\rightarrow n c_0 + \sum_{j=0}^{n-1} n - j + C(0) \text{ cost}$$

$$n c_0 + \frac{n(n+1)}{2} + 1 \rightarrow O(n^2) \rightarrow O(3^n)$$

Problema 3 Algoritmi

- (a) Illustrare un algoritmo (codice o pseudo-codice) che, dato un array di interi, ne permuta i valori in modo da trasformarlo nella rappresentazione di un max-heap. [5/30]
- (b) Dato un albero binario t , progettare un algoritmo (codice o pseudo-codice) che determini se t è completo (tutti i suoi livelli hanno il massimo numero di nodi). [5/30]
- (c) Con riferimento all'albero in figura, scrivere in quali sequenza vengono visitati i suoi vertici con: visita pre-order, visita in-order, visita post-order. [3/30]



- (d) Definire gli alberi di Fibonacci (50%) e spiegare a cosa servono (50%).

[4/30]

a) `VOID ARRAY2HEAP(INT[] a, INT SIZE){
 FOR(INT i = (SIZE - 1)/2; i >= 0; i--)
 DOWNHEAP(a, i);
}`

`VOID DOWNHEAP(INT[] a, INT FATHER){
 INT LEFT = FATHER * 2 + 1;
 INT RIGHT = FATHER * 2 + 2;
 IF (a[LEFT] > a[RIGHT] && a[FATHER] < a[LEFT]) {
 INT TEMP = a[FATHER];
 a[FATHER] = a[LEFT];
 a[LEFT] = TEMP;
 DOWNHEAP(a, LEFT);
 }
 ELSE IF (a[LEFT] < a[RIGHT] && a[FATHER] < a[RIGHT]) {
 INT TEMP = a[FATHER];
 a[FATHER] = a[RIGHT];
 a[RIGHT] = TEMP;
 DOWNHEAP(a, RIGHT);
 }
}`

b)

```

INT COMPLETO_AUX(BST x){
    IF(x == NULL) RETURN -1;
    IF(x->LEFTS != NULL && x->RIGHTS == NULL ||
       x->LEFTS == NULL && x->RIGHTS != NULL)
        RETURN -2;
    INT LEFT = COMPLETO_AUX(x->LEFTS);
    IF(LEFT == -2) RETURN -2;
    INT RIGHT = COMPLETO_AUX(x->RIGHTS);
    IF(RIGHT == -2) RETURN -2;
    IF(LEFT != RIGHT) RETURN -2;
    ELSE RETURN LEFT + 1;
}

```

```

BOOL COMPLETO(BST x){
    IF(COMPLETO_AUX(x) > 0)
        RETURN TRUE;
    ELSE
        RETURN FALSE;
}

```

c)

PRE-ORDER:

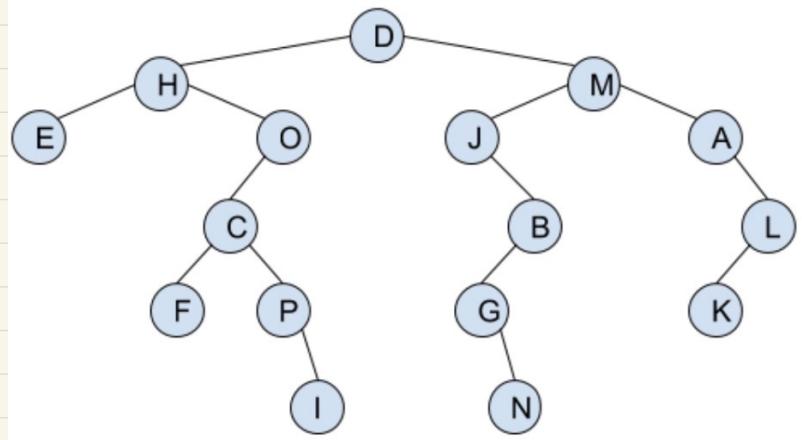
D H E O C F P I M J B G N A L K

IN-ORDER:

E M F C L P I O D J G N B H A K L

POST-ORDER:

E F I P C O W N G B S K L A H D



d)

GLI ALBERI DI FIBONACCI SONO BST BILANCIATI (AVL) CHE HANNO FATTORE DI BILANCIAMENTO 1, OPPURE -1, PER TUTTI I NODI INTERNI. QUESTO LI PORTA AD AVERE L'ALTEZZA MASSIMA PER IL # NODI. TALI ALBERI SERVONO PER DEMONSTRARE CHE L'ALTEZZA DEGLI AVL È AL MASSIMO $O(\log n)$, QUESTO PORTA AD AVERE I COSTI DELLE OPERAZIONI DI BST PARI AL MASSIMO A $O(\log n)$

Analisi algoritmo

Si considerino i metodi Java di seguito illustrati.

```
static void incr(byte[] a, int n, int p) {
    byte b = 0;
    for(int j = 0; j < p; j++)
        b = (byte)(b+a[n-p+j]);
    a[n] = b;
}
```

```
static byte[] raddoppia(byte[] a) {
    byte[] b = new byte[2*a.length];
    for(int i = 0; i < a.length; i++) b[i] = a[i];
    for(int i = a.length; i < b.length; i++)
        incr(b, i, a.length);
    return b;
}
```

Sviluppare, argomentando adeguatamente (il 50% del punteggio dell'esercizio sarà sulle argomentazioni addotte), quanto segue:

- (a) Determinare il costo temporale asintotico dell'algoritmo descritto da `raddoppia(byte[] a)` in funzione di z , la dimensione dell'input. **Non usare simboli che non sono definiti** (ad esempio: n riferita alla dimensione dell'input non è definita). [4/30]
- (b) Spiegare brevemente qual è la relazione tra le componenti di `b` e quelle di `a`. [2/30]

a) IL PRIMO CICLO IN RADDOPPIA COSTA $O(z)$ IN DIR DELL'INPUT INFATTI IL CICLO FA n ITERAZIONI SULL'ARRAY IN INPUT, PERCIÒ LA DIR DELL'INPUT È $z = cn \quad n = O(z)$

IL SECONDO CICLO FA $2n$ ITERAZIONI E A OGNI ITERAZIONE FA UNA CHIARATA A INCR. LA QUALE FA LA PRIMA VOLTA n PASSI, Poi $n-1$ PASSI, ... QUINDI ABBIAMO $n + n-1 + n-2 \dots = O(n^2)$ E QUINDI L'INTERO CICLO CHE CHIARA INCR COSTA.

$$z = cn \rightarrow n = O(z) \rightarrow O(n^2) = O(z^2)$$

INFINE SI HA:

$$O(z) + O(z^2) = O(z^2) \text{ COSTO ALG. IN FUNZIONE DELL'INPUT}$$

b) I PRIMI $a.LENGTH$ ELEMENTI DI `b` SONO UGUALI A QUELLI DI `a`. I RESTANTI $a.LENGTH$ ELEMENTI DI `b` SONO pari ALLA SOMMA DEI NUMERI DELLA PRIMA METÀ DI `b` DA $i - a.LENGTH$ AD $a.LENGTH$ NON COMPRESO

Algoritmi

- (a) Si consideri un grafo $G = (V, E)$ orientato *completo* (per ogni coppia di vertici $u, v \in V$, esistono entrambi gli archi *diretti* (u, v) e (v, u)) di n vertici. Com'è fatto l'albero di visita di una DFS applicata a G ? Si descriva la sua struttura. [5/30]
- (b) Dobbiamo realizzare una mappa (rispetto a chiavi *intero*). Una volta popolata con coppie (chiave, valore), la mappa dovrà gestire le seguenti operazioni (tra parentesi, la frequenza prevista per ciascuna operazione): i) `search(k)` (40% di tutte le operazioni); ii) `range_query(k1, k2)` (60% di tutte le operazioni). Sia n il numero di coppie presenti dopo il popolamento della struttura dati. Assumendo che, tipicamente, $k_2 - k_1 = O(\log n)$ nelle operazioni di range query, quale struttura dati usereste tra una tabella hash e un AVL? Motivare la risposta. [4/30]
- (c) Mostrare come è possibile modificare un albero AVL in modo da implementare una coda di priorità con le stesse prestazioni (asintotiche) di un heap rispetto alle consuete operazioni di ricerca/estrazione del minimo, inserimento/cancellazione di una chiave. Non si richiede la descrizione degli algoritmi che implementano le operazioni menzionate sopra, ma soltanto di descrivere *chiaramente* come queste ultime (e la struttura dati sottostante) vanno modificate per conseguire l'obiettivo. [4/30]

a) L'ALBERO DI VISITA SARÀ DI ALTEZZA $n-1$, INFATTI DA OGNI NODO VISITO UN SOLO ALTRO NODO. QUINDI OGNI NODO AVRÀ 1 SOLO FIGLIO NELL'ALBERO DI VISITA

b) NEL CASO DI UNA SEARCH UNA TABELLA MASH AVRÀ COSTO COST. SE SI MANTIENE IL COEFFICIENTE DI CARICO $\leq \frac{1}{2}$. PER UN AVL IL COSTO SARÀ $O(\log n)$

PER LA RANGE QUERY INVECE, CON LA TABELLA MASH NON POSSO SFRUTTARE ALUN ORDINAMENTO. QUESTO VUOL DIRE CHE DEVO SCANDIRE TUTTA LA MAPPA PER DARE IN USCITA LA RANGE QUERY. L'AVL MI COSTA $O(\log n)$ PER TROVARE k , E POI MO IL COSTO DATO DALLA DIM DELL'OUTPUT PARI A $O(\log n)$ CORRE DA SPECIFICA

QUINDI:

$$\text{MASMT.} \rightarrow m \cdot 0.4 \cdot O(1) + m \cdot 0.6 \cdot O(n) = O(m \cdot n)$$

$$\text{AVL} \rightarrow m \cdot 0.4 \cdot O(\log n) + m \cdot 0.6 \cdot O(2 \log n) = O(m \cdot \log n)$$

CONVIENE L'AVL CON m OPERAZIONI

c) PER AVERE IL MINIMO IN TEMPO COSTANTE MI BASTA MANTENERE UN PUNTATORE AL NODO CON CHIAVE PIÙ PICCOLA DELL'AVL. QUANDO MI VIENE CHIESTO IL MINIMO MI BASTA DARE IN OUTPUT IL NODO PUNTATO DAL PUNTATORE.

NEL CASO DI INSERIMENTO BISOGNA, QUINDI, FARE ATTENZIONE A MANTENERE CORRETTAMENTE TALE PUNTATORE. PER FARE QÒ, AD OGNI INSERIMENTO SI VERIFICA SE IL NODO È PIÙ PICCOLO DEL MINIMO ATTUALE ED IN CASO AFFERMATIVO AGGIORNO.

NEL CASO DI CANCELLAZIONE DEL MINIMO SI AGGIORNA IL PUNTATORE RICERCANDO IL MINIMO IN TEMPO $O(\log n)$. SE SI ELIMINA UNA CHIAVE NON MINIMA NON MODIFICO NIENTE

BST: max-gap

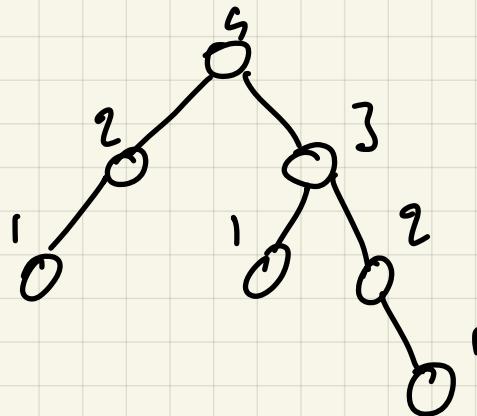
Dato un BST t contenente chiavi intere, progettare un algoritmo (pseudo-codice) che determini e restituisca la massima differenza fra due chiavi adiacenti (rispetto all'ordinamento delle chiavi stesse). L'algoritmo deve avere un costo lineare rispetto al numero di chiavi presenti nel BST. Inoltre, l'algoritmo deve essere *in-place*, ossia non deve far uso di strutture dati di appoggio, a parte eventualmente una quantità di memoria ausiliaria pari a $O(1)$ (tuttavia, saranno particolarmente apprezzate soluzioni che non fanno alcun uso di memoria ausiliaria). Quindi ad esempio, copiare tutte le chiavi in un array ausiliario per risolvere il problema non va bene.
Suggerimento: Per iniziare, si pensi a un semplice algoritmo lineare (che, come scritto sopra, non costituisce una soluzione accettabile) nel caso in cui sia possibile usare un array come struttura dati appoggio. [4/30]

```
INT MAX-GAP(BST  $\pi$ ) {
    INT PREC[2];
    PREC[0] = -1;
    PREC[1] = -1;
    MAX-GAP-AUX( $\pi$ , PREC);
    RETURN PREC[1];
}
```

```
VOID MAX-GAP-AUX(BST  $\pi$ , INT* PREC) {
    IF ( $\pi == \text{NULL}$ ) RETURN;
    IF (PREC[0] == -1)
        PREC[0] =  $\pi \rightarrow \text{VALUE}$ ;
    MAX-GAP-AUX( $\pi \rightarrow \text{LEFT}$ , PREC);
    IF (|PREC[0] -  $\pi \rightarrow \text{VALUE}| > \text{PREC}[1])$ 
```

$$\text{PREC}[1] = |\text{PREC}[0] - \pi \rightarrow \text{VALUE}|;$$

$$\text{PREC}[0] = \pi \rightarrow \text{VALUE};$$

$$\text{MAX-GAP-AUX}(\pi \rightarrow \text{RIGHT}, \text{PREC});$$
}


```

static int f(int q) {
    if(q <= 1) return q;
    if((q % 2) == 0) return g(q>>1);
    else return f(q-1);
}

static int g(int r) {
    if(r <= 1) return f(r);
    if((r % 3) == 0) return f(r/3);
    else return f(r+1);
}

```

Determinare il costo asintotico di caso peggiore di `f(int)` in funzione della dimensione dell'input. Risposte non motivate non saranno prese in considerazione.

$$C(q) = C + O\left(\frac{q}{2}\right) = 2C + O\left(\frac{q}{4}\right) = 3C + O\left(\frac{q}{8}\right) = \dots + O\left(\frac{q}{2^i}\right)$$

SI FERMA QUANDO $\frac{q}{2^i} \leq 1 \rightarrow q \leq 2^i \rightarrow i \geq \log_2 q$

$$O(\log_2 q) \cdot C + \text{cost} \approx O(\log q)$$

IN SIM INPUT $\rightarrow z = |q| = \log_2 q \rightarrow q = 2^z \rightarrow O(z)$

Come cambia l'analisi precedente se l'ultima riga del metodo `f(int)` viene riscritta come segue?

```

else return f(q+1); // invece di return f(q-1)
}

```

PER ESEMPIO, CON $q=3$

1° IF FALLISCE
2° IF FALLISCE
CHIAMO $f(4)$
1° IF FALLISCE
CHIAMO $g(2)$

1° IF FALLISCE
2° IF FALLISCE
CHIAMO $f(3)$

$\left. \begin{matrix} f \\ g \end{matrix} \right\}$

LA COMPUTAZIONE
NON TERMINA

Quesito 3: Algoritmi

- Si supponga di avere una tabella hash di dimensione iniziale n (ossia, la tabella può inizialmente contenere fino a n coppie (chiave, valore)). La tabella è inizialmente vuota. Si supponga che la tabella hash sia implementata particolarmente bene e che per fattori di carico non superiori a 0.5 il suo comportamento in termini di complessità delle operazioni sia ideale (questa è ovviamente una semplificazione, ma abbastanza realistica, almeno in media). Si supponga ora che siano inserite in sequenza $\lfloor \frac{n}{2} \rfloor$ coppie (chiave, valore) e che a seguito di un ulteriore inserimento ($\lfloor \lfloor n/2 \rfloor + 1 \rfloor$ -esimo) la dimensione della tabella venga portata a $2n - 1$ (in modo da mantenere non superiore a 0.5 il fattore di carico). Ciò premesso: i) Si valuti il costo dell'operazione $\lfloor \frac{n}{2} \rfloor + 1$ -esima; ii) si determini il costo medio di ciascun inserimento, pari a $(\text{costo totale di tutte le operazioni}) / (\text{numero di operazioni})$.

Punteggio: [3/30]

- Si consideri un min-heap inizialmente vuoto. Si supponga che, in sequenza, vengano inserite nell'heap n coppie (chiave, valore), in ordine crescente rispetto all'ordinamento delle chiavi. Si valuti il costo complessivo delle n operazioni di inserimento, nel caso peggiore. Giustificare la risposta.

[assumere che le chiavi siano usate come priorità]

Punteggio: [3/30]

- Mostrare un'istanza del problema dei cammini minimi s-t (sorgente-destinazione) su un grafo pesato diretto che non ammette una soluzione a causa della presenza di (almeno) un ciclo di lunghezza negativa. Istanza significa che occorre specificare il grafo, i pesi sugli archi, i nodi sorgente e destinazione.

Punteggio: [3/30]

1) ALL' $\lfloor \frac{n}{2} \rfloor + 1$ -ESIMO INSERIMENTO ABBIAMO, OLTRE AL COSTO COSTANTE

DELL' INSERIMENTO, UN COSTO DOVUTO ALLA RIALLOCAZIONE DELL' ARRAY PER AUMENTARE LA DIM. IL COSTO PER AUMENTARE LA DIM DELLA HASH TABLE È PARI A $O(n + \lfloor \frac{n}{2} \rfloor + 1) = O(n)$. INFATTI DOVREMO FARE n INSERIMENTI NELLA NUOVA TABELLA.

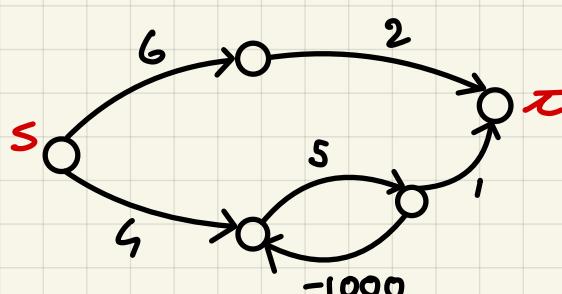
$$\text{IL COSTO MEDIO È } \frac{\frac{n}{2} + n + \frac{n}{2} + 1}{n + 1} = O(1)$$

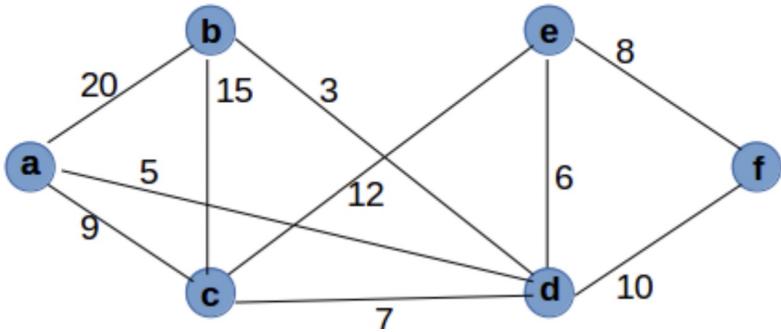
INFATTI ABBIAMO $\lfloor \frac{n}{2} \rfloor$ OPERAZIONI CON COSTO COSTANTE ED UNA CHE HA COSTO $n + \frac{n}{2} + 1$, FACCIO TANTI INSERIMENTI QUANTE SONO LE ENTRY DELLA HASH TABLE

2) INSERENDO NELL' HEAP ENTRY CON PRIORITÀ ORDINATE IN ORDINE CRESCENTE ALLORA AVRÒ n INSERIMENTI CON COSTO COSTANTE. A INSERIMENTO, UPHEAP NON VIENE ESEGUITO DATO CHE LE ENTRY GIÀ PRESENTI NELL' HEAP SONO PIÙ PICCOLE DELLA ENTRY APPENA INSERITA COME FOGLIA.

IL COSTO TOTALE È $O(n)$

3)





- Esiste un unico albero minimo ricoprente per il grafo in figura o può esisterne più di uno?
Non basta rispondere sì o no. Occorre dare una motivazione, anche senza dimostrazione.

Punteggio: [3/30]

- Si mostri l'evoluzione dell'algoritmo di Prim-Jarnik per il grafo in Figura 2 a partire dal nodo **b**. In particolare, per ogni iterazione i dell'algoritmo occorre specificare i) il sotto-insieme T degli archi dell'albero minimo ricoprente già identificati all'inizio dell'iterazione i -esima e ii) l'arco che verrà aggiunto a T nell'iterazione i -esima.

Punteggio: [3/30]

1) IL GRAFO AMMETTE UN UNICO MST PERCHÉ NON HA ARCHI CON PESI UGUALI. AVERE ARCHI CON PESI UGUALI È INFATTI UNA CONDIZIONE NECESSARIA PER AVERE PIÙ DI UN MST

UTILIZZANDO PER ESEMPIO L'ALGORITMO DI PRIM-JARNIK, QUANDO DEVO INSERIRE UN NODO NELLA NUOVA POTREI AVERE CHE TALE NODO È COLLEGATO ALLA NUOVA ATTRAVERSO 2 ARCHI DI PESO UGUALE, DIMOSTRANDO CHE ESISTE PIÙ DI UN MST

2) L'ALGORITMO INIZIALIZZA UNA CODA PRIORITARIA CON TUTTI I NODI POSTI A DISTANZA $+\infty$, MENTRE LA SORENTE A DISTANZA 0. INIZIALMENTE $T = \emptyset$

1° IT

L'ALGORITMO ESTRAE b DALLA CODA E IMPOSTA LA DISTANZA

$$a \rightarrow 20 \quad c \rightarrow 15 \quad d \rightarrow 3 \quad T = \emptyset$$

2° IT

L'ALGORITMO ESTRAE d DALLA CODA E IMPOSTA

$$e \rightarrow 6 \quad f \rightarrow 10 \quad a \rightarrow 5 \quad c \rightarrow 7 \quad T = \{b, d\}$$

3° IT

L'ALGORITMO ESTRAE a DALLA CODA E NON LA MODIFICA

$$e \rightarrow 6 \quad f \rightarrow 10 \quad c \rightarrow 7 \quad T = \{b, d\}, \{d, a\}$$

4° IT

L'ALGORITMO ESTRAE e DALLA CODA E IMPOSTA

r->8 c->7

T= <b,d>, <d,a>, <d,e>

5° IT

L'ALGORITMO ESTRAE c DALLA CODA E NON LA MODIFICA

r->8 T= <b,d>, <d,a>, <d,e>, <d,c>

6° IT

L'ALGORITMO ESTRAE r, CODA VUOTA

T= <b,d>, <d,a>, <d,e>, <d,c>, <c,r>

Quesito 1: Analisi algoritmo

Il seguente codice Java implementa un algoritmo che determina la lunghezza della più lunga sottosequenza crescente di una sequenza data, ed è basato sul paradigma della programmazione dinamica.

```
1 static int CeilIndex(int A[], int l, int r, int key) {  
2     while (r - l > 1) {  
3         int m = l + (r - l) / 2;  
4         if (A[m] >= key) r = m;  
5         else l = m;  
6     }  
7     return r;  
8 }  
9  
10 static int LongestIncreasingSubsequenceLength(int A[], int size) {  
11     int[] tailTable = new int[size];  
12     int len;  
13     tailTable[0] = A[0];  
14     len = 1;  
15     for (int i = 1; i < size; i++)  
16         if (A[i] < tailTable[0])  
17             tailTable[0] = A[i];  
18         else if (A[i] > tailTable[len - 1])  
19             tailTable[len++] = A[i];  
20         else tailTable[CeilIndex(tailTable, -1, len - 1, A[i])] = A[i];  
21     return len;  
22 }
```

Si risponda ai seguenti quesiti:

1. Determinare il costo asintotico dell'algoritmo `LongestIncreasingSubsequenceLength`.

Punteggio: [4/30]

2. Determinare il contenuto finale dell'array `tailTable` sulla seguente sequenza di input :

```
1 | {0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15}
```

Punteggio: [3/30]

Quesito 3: Algoritmi

1. Si consideri una tabella hash a *indirizzamento aperto* di dimensione $N = 11$ e a chiavi intere. Si supponga che la funzione di compressione sia $f(x) = x \bmod N$ e che le collisioni siano risolte mediante *esplorazione lineare*. Dare un esempio di aggregazione (clustering) *primaria*, generata dall'inserimento in successione di 4 coppie (chiave, valore). Occorre spiegare cos'è l'aggregazione e perché si verifica nell'esempio proposto.

Punteggio: [3/30]

2. Si mostri come si può usare un heap minimale per individuare, con un costo $O(n + k \log n)$, i k valori più piccoli tra quelli contenuti in un *array non ordinato* contenente n chiavi intere. Occorre giustificare la risposta.

Punteggio: [3/30]

3. Con riferimento alla Fig. 1, descrivere la sequenza con cui vengono visitati i nodi dell'albero binario, per ciascuna delle visite in pre-ordine, post-ordine e simmetrico.

Punteggio: [3/30]

1) INSERISCO ENTRY CON LE SEGUENTI CHIAV: 11, 27, 33, 44

$f(x) = 0 \wedge$ ENTRY. LA TAVOLA HASH SARÀ QUINDI:

0	11
1	27
2	33
3	44
.	.
.	.
11	.

IL CLUSTERING PRIMARIO AVVIENE QUANDO HO IN INPUT CHIAVI CHE MANNO LO STESSO RISULTATO DELLA FUNZIONE DI HASH, GENERANDO UNA COLLISIONE, LA QUALE, SE RISOLTA CON ESPLORAZIONE LINEARE, MI PORTA AD INSERIRE LA ENTRY NELLA CELLA SUBITO SUCCESSIVA. GIÀ PORTA AD AUMENTARE LE PROBABILITÀ DI COLLUSIONE ED I COSTI DELLA SUA RISOLUZIONE
IN QUESTO ESEMPIO SI VERIFICA PERCHÉ INSERISCO TUTTE ENTRY CON PARI $f(x)$

2) DATO L'ARRAY IN INPUT QUESTO PUÒ ESSERE TRASFORMATO IN HEAP ATTRAVERSO UNA COSTRUZIONE BOTTOM UP (CON COSTO $O(n)$) CON L'ALGORITMO ARRAY TO HEAP. FATTO GIÀ RIMUOVO k VOLTE IL MINIMO CON COSTO $\log n$, OTTENENDO $O(n + k \log n)$

3)

PRE-ORDER:

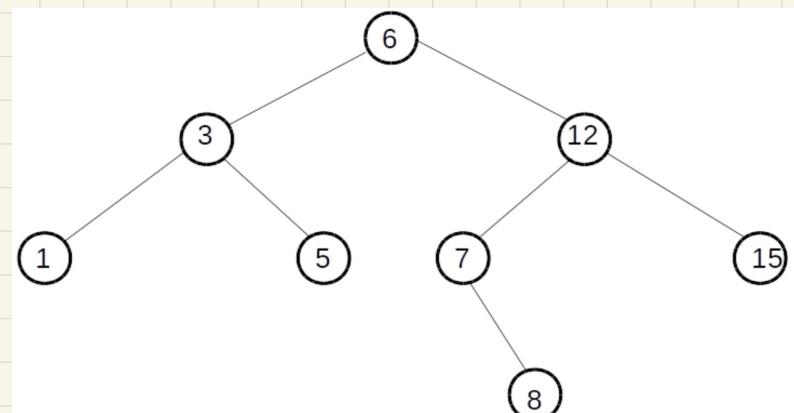
6 3 1 5 12 7 8 15

IN-ORDER:

1 3 5 6 7 8 12 15

POST-ORDER:

1 5 3 8 7 15 12 6



Quesito 4:

Una società di web analytics ha collezionato dati sui "like" attribuiti da utenti (che chiameremo "soggetti") a prodotti/servizi (che chiameremo "oggetti"). Tali dati vengono descritti attraverso un opportuno grafo i cui nodi descrivono utenti e/o prodotti/servizi, mentre gli archi descrivono l'attribuzione dei "like". L'azienda tuttavia nutre dei dubbi sulla qualità dei dati poiché ritiene che alcuni utenti abbiano usato il pulsante "like" a casaccio, senza dunque caratterizzare davvero le loro preferenze. Poiché non esiste un algoritmo che consenta di individuare tale categoria di utenti, l'azienda decide di fare alcune analisi, approntando alcuni semplici strumenti.

Uno di tali strumenti, detto `contaAnaloghi`, è teso a valutare l'unicità di un utente: in particolare, dato un utente u , si vuole determinare il numero di utenti (diversi da u , e detti *analoghi* ad u) i cui like definiscono insiemi di oggetti che sono superinsiemi di quello definito dai like di u . Se ad esempio u ha attribuito il like agli oggetti $\{a, d\}$ e

- v ha messo like su $\{a, b, e\}$ (intersezione non nulla, non superinsieme)
- w su $\{a, c, d\}$ (superinsieme)
- x su $\{a, d\}$ (superinsieme, uguale)
- y su $\{d\}$, (intersezione nulla, sottoinsieme, non superinsieme)

allora gli analoghi di u saranno 2 (w ed x).

Si risponda alle domande seguenti:

- Definire con precisione il grafo usato per rappresentare i "like", descrivendone caratteristiche e proprietà (semplice, connesso, orientato, bipartito, ecc.). Definire in termini di problemi su grafi l'input e l'output dell'algoritmo implementato da `contaAnaloghi`.

Punteggio: [3/30]

- Si descriva (è sufficiente lo pseudo-codice o comunque una descrizione dettagliata) l'algoritmo usato per implementare `contaAnaloghi`. La descrizione può essere anche ad alto livello concettuale ed usare primitive non elementari, purché quest'ultime appartengano al corredo standard del tipo/classe `Graph`. Descrizioni basate su codice C/Java sono comunque considerate accettabili.

Punteggio: [3/30]

1) È UN GRAFO ORIENTATO BI PARTITO

AVREMO INFATTI IL CLUSTER DI UTENTI ED IL CLUSTER DEI SERVIZI
DUE UTENTI NON POSSONO METTERSI LIKE TRA DI LORO E QUINDI
NON CI SARANNO NEI ARCHI FRA 2 UTENTI, LO STESSO VALE PER I
SERVIZI!

È ORIENTATO PERCHÉ È UN UTENTE CHE METTE LIKE AD UN
SERVIZIO, E NON ANCHE VICEVERSA. IL GRAFO NON AMMETTE CAPPI
POICHE' UN UTENTE NON PUÒ METTERSI LIKE DA SOLO.

L'INPUT SONO UN GRAFO E L'UTENTE DI CI CONTARE GLI ANALOGHI
L'OUTPUT È L'INTERO RAPPRESENTANTE IL # ANALOGHI TROVATI

2) CONSIDERIAMO UN GRAFO RAPPRESENTATO DA LISTE DI ADIACENZA
E CON LISTA PER UTENTI E LISTA PER SERVIZI

CONTAALOGHI (GRAPH g, UTENTE u){

```
INT ESPLORATI = 0;
INT ANALOGHI = 0;
FOR NODO IN U. ADIACENTI DO {
    NODO. STATO = ESPLORATO;
    ESPLORATI++;
}
FOR USER IN g. LISTA. UTENTI ≠ u DO {
    INT TEMP = 0;
    FOR SERVICE IN USER. ADIACENTI DO {
        IF SERVICE. STATO == ESPLORATO
            TEMP++;
    }
    IF TEMP == ESPLORATI
        ANALOGHI++;
}
RETURN ANALOGHI;
```

}

Quesito 1: Analisi algoritmo

Il seguente codice Java implementa il celebre Crivello di Eratostene per il calcolo (stampa) di tutti i primi non superiori a un limite dato n .

```

1  static void eratostene(int n) {
2      int[] list = new int[n+1]; // creato già già azzerato
3
4      // marca i pari > 2
5      for(int j = 4; j <= n; j += 2) list[j]++;
6
7      // marca gli altri numeri composti
8      for(int i = 3; i <= n; i += 2)
9          if(list[i] == 0)
10             for(int j = 2*i; j <= n; j += i)
11                 list[j]++; // marca j
12
13     // stampa i primi
14     System.out.print("{");
15     if(2 <= n) {
16         System.out.print("+"2);
17         for(int i = 3; i <= n; i += 2)
18             if(list[i] == 0) System.out.print(", "+i); // stampa i non marcati
19     }
20     System.out.print("}\n");
21 }
```

Si risponda ai seguenti quesiti:

1. Stimare il costo asintotico *temporale* dell'algoritmo `eratostene`, in funzione della dimensione dell'input, chiarendo se la stima sia esatta (*tight*) o per eccesso; nel secondo caso, spiegare dove interviene la sovrastima.

Punteggio: [4/30]

2. Stimare il costo asintotico *spaziale* dell'algoritmo `eratostene`, in funzione della dimensione dell'input, chiarendo se la stima sia esatta (*tight*) o per eccesso; nel secondo caso, spiegare dove interviene la sovrastima.

Punteggio: [3/30]

Quesito 3: Algoritmi

- Definire gli alberi di Fibonacci e spiegare perché sono interessanti. Quanti nodi hanno gli alberi di Fibonacci di altezza 6? Perché gli alberi di Fibonacci di una data altezza hanno tutti lo stesso numero di nodi?

Punteggio: [3/30]

- Si consideri un albero binario completo di altezza h , con chiavi associate ai nodi e tale che la chiave associata a ogni nodo interno sia il max fra le chiavi dei due figli (tale tipologia di albero binario è detta *torneo*). Naturalmente la radice contiene la massima chiave dell'albero. Quanti confronti sono necessari e sufficienti per determinare la seconda chiave in ordine di grandezza (assumere per semplicità che tutte le chiavi siano distinte). E quanti per determinare la terza? È possibile derivare un algoritmo di ordinamento? Se sì, quale sarà il suo costo?

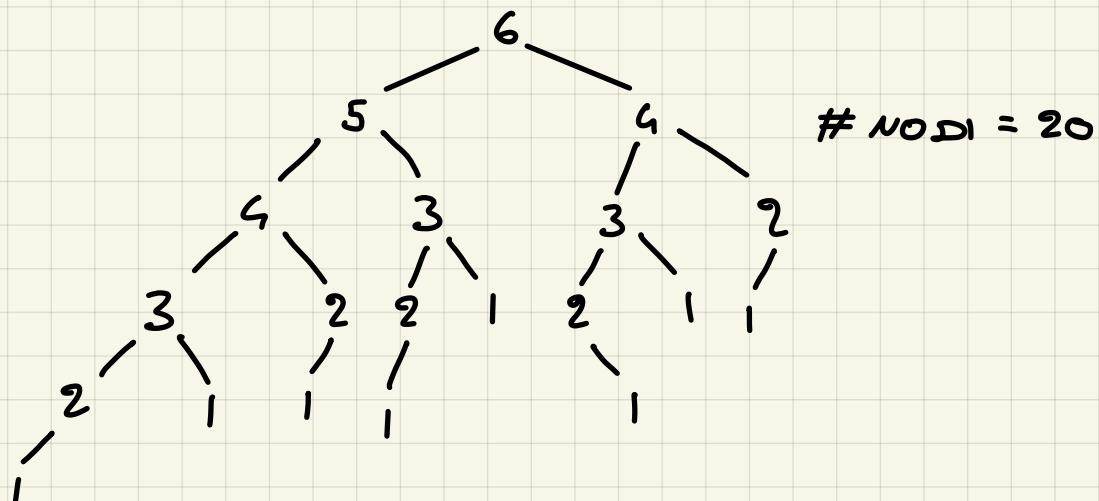
Punteggio: [4/30]

- Dato un grafo semplice e connesso, con n nodi ed m archi, è possibile determinare in tempo costante se esso sia aciclico o meno? Spiegare. N.B. Il grafo è connesso per ipotesi.

Punteggio: [2/30]

GLI ALBERI DI FIB SONO TUTTI GLI ALBERI BINARI DI RICERCA BILANCIATI CARATTERIZZATI DA FATTORE DI BILANCIAMENTO 1 O -1 PER TUTTI I NODI INTERNI.
SONO IMPORTANTI PERCHÉ MAMNO L'ALTEZZA MAX PER UN ALBERO BINARIO BILANCIATO CON IL MINOR # NODI PER QUELL'ALTEZZA, QUESTO PERMETTE DI DEMONSTRARE CHE USANDO UN AVL POSSO SVOLGERE OPERAZIONI DI BST CON COSTO AL MASSIMO $\log n$

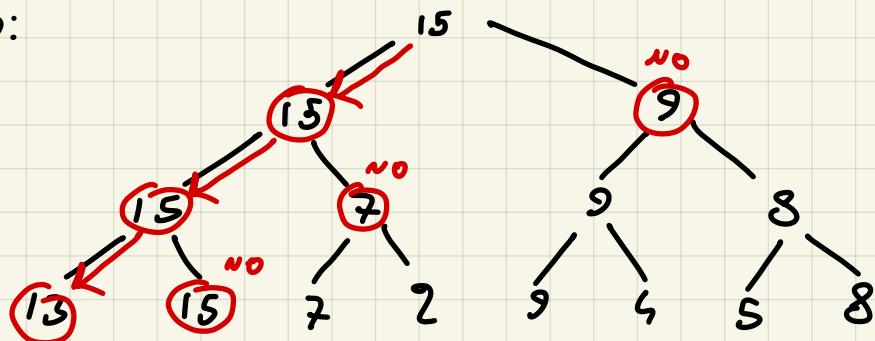
$h=6$



GLI ALBERI DI FIB DI UNA CERTA h MAMNO TUTTI LO STESSO # NODI PERCHÉ VA MANTENUTO IL FDB 1 O -1 PER TUTTI I NODI INTERNI E, QUINDI, A PARITÀ DI h HO SEMPRE BISOGNO DELLO STESSO # NODI

2) ESEMPIO TORNEO:

2° PIÙ GRANDE:



VEDO TUTTI QUELLI CHE HANNO PERSO CON IL PRIMO: 13, 7, 9

PER TROVARE IL 3° PIÙ GRANDE PONGO 15 A $-\infty$ E FARLO SALIRE 13 ALLA RADICE. IL PIÙ GRANDE TRA I PERDENTI È 9. SERVONO 3 CONFRONTI

$h = \log n$ ESSENDO COMPLETO

IL COSTO TOTALE È $n \log n$ DATO CHE FARLO n VOLTE IL PROX DI SOPRA

3) UN GRAFO ACICLICO E CONNESSO È CHIAMATO ALBERO ED HA $n-1$ ARCHI

$m < n-1$ NON PUÒ ESSERE POICHÉ È IL MINIMO # ARCHI PER UN GRAFO CONNESSO

$m = n-1$ È UN ALBERO, QUINDI ACICLICO

$m > n-1$ SIGNIFICA CHE AGGIUNGO ARCHI ALL'ALBERO, CHIUDO UN CYCLO

PASSO QUINDI VERIFICARLO IN TEMPO COSTANTE

Quesito 4:

Si consideri il grafo non diretto e pesato della Figura 2:

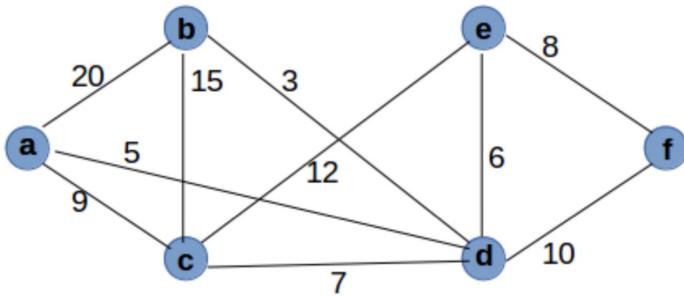


Figura 2. Esempio di grafo non diretto pesato.

Si risponda alle domande seguenti:

- Esiste un unico albero minimo ricoprente per il grafo in figura o può esisterne più di uno?
Non basta rispondere sì o no. Occorre dare una motivazione, anche senza dimostrazione.
Punteggio: [3/30]
- Si mostri l'evoluzione dell'algoritmo di Prim-Jarnik per il grafo in Figura 2 a partire dal nodo **b**.
In particolare, per ogni iterazione i dell'algoritmo occorre specificare i) il sottoinsieme T degli archi dell'albero minimo ricoprente già identificati all'inizio dell'iterazione i -esima e ii) l'arco che verrà aggiunto a T nell'iterazione i -esima.

1) IL GRAFO AMMETTE UN UNICO MST PERCHÉ NON HA ARCHI CON PESI UGUALI. AVERE ARCHI CON PESI UGUALI È INFATI UNA CONDIZIONE NECESSARIA PER AVERE PIÙ DI UN MST

UTILIZZANDO PER ESEMPIO L'ALGORITMO DI PRIM-JARNIK, QUANDO DEVO INSERIRE UN NODO NELLA NUOVA POTREI AVERE CHE TALE NODO È COLLEGATO ALLA NUOVA ATTRAVERSO 2 ARCHI DI PESO UGUALE, DIMOSTRANDO CHE ESISTE PIÙ DI UN MST

2) L'ALGORITMO INIZIALIZZA UNA CODA PRIORITARIA CON TUTTI I NODI POSTI A DISTANZA $+\infty$, MENTRE LA SORENTE A DISTANZA 0. INIZIALMENTE $T = \emptyset$

1° IT

L'ALGORITMO ESTRAE **b** DALLA CODA E IMPOSTA LA DISTANZA

$$a \rightarrow 20 \quad c \rightarrow 15 \quad d \rightarrow 3 \quad T = \emptyset$$

2° IT

L'ALGORITMO ESTRAE **d** DALLA CODA E IMPOSTA

$$c \rightarrow 6 \quad f \rightarrow 10 \quad a \rightarrow 5 \quad c \rightarrow 7 \quad T = \{b, d\}$$

3° IT

L'ALGORITMO ESTRAE **a** DALLA CODA E NON LA MODIFICA

$$e \rightarrow 6 \quad f \rightarrow 10 \quad c \rightarrow 7 \quad T = \{b, d, a\}$$

4° IT

L'ALGORITMO ESTRAE e DALLA CODA E IMPOSTA

f-> 8 c-> 7

T = <b,d>, <d,a>, <d,e>

5° IT

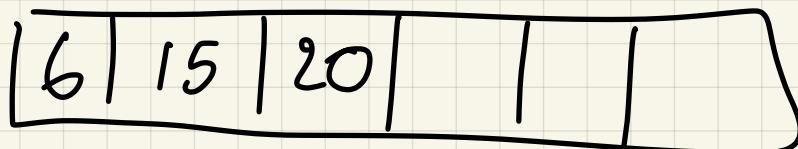
L'ALGORITMO ESTRAE c DALLA CODA E NON LA MODIFICA

f-> 8 T = <b,d>, <d,a>, <d,e>, <d,c>

6° IT

L'ALGORITMO ESTRAE f, CODA VUOTA

T = <b,d>, <d,a>, <d,e>, <d,c>, <c,f>



6 15 20 2 0 25

Quesito 2: Algoritmi

1. Si consideri una tabella hash di dimensione $N = 13$ con gestione delle collisioni mediante scansione lineare (*linear probing*) e funzione hash (di compressione) $h(k) = k \bmod N$, dove k è la chiave. Descrivere la *successione degli stati* della tabella (ossia l'evoluzione del suo contenuto) a seguito dell'inserimento delle chiavi seguenti nella successione indicata: 10, 26, 52, 13, 3, 33, 60, 42.

Segnalare gli inserimenti che danno luogo a collisione, specificando le posizioni coinvolte. Il punteggio dipenderà anche dalla completezza e dalla chiarezza espositiva. Non occorre scrivere tanto ma scrivere bene.

Punteggio: [8/30]

0	1	2	3	4	5	6	7	8	9	10	11	12
26	52	13	3	42		33	8	61	10	76		

10 MOD 13 = 10
 26 MOD 13 = 0
 52 MOD 13 = 0 → 53 MOD 13 = 1 ← $k+1 \bmod N$
 76 MOD 13 = 11
 13 MOD 13 = 0 → 15 MOD 13 = 2
 3 MOD 13 = 3
 33 MOD 13 = 7
 60 MOD 13 = 8 → 61 MOD 13 = 9
 42 MOD 13 = 3 → 43 MOD 13 = 4

5 COLLISIONI

2. Si consideri una lista non ordinata rappresentata con un array. L'array è inizializzato a una dimensione costante k . Si consideri ora una successione di n inserimenti. Il generico inserimento ha costo costante *a meno che l'array non sia già pieno*. Se l'array è pieno l'inserimento determina i) l'allocazione di un *nuovo* array di dimensione $x + k$ se x era la dimensione precedente dell'array, ii) la copia del contenuto del vecchio array nel nuovo (la copia del *singolo* valore dal vecchio array al nuovo ha costo costante), iii) l'inserimento del nuovo elemento (nel nuovo array). Si calcoli il *costo temporale complessivo asintotico* della successione degli n inserimenti (si supponga che non vi siano mai rimozioni di elementi).

Occorre dare un'argomentazione quantitativa, rigorosa (prova) e chiara, giustificando adeguatamente la risposta. Il punteggio dipenderà soprattutto da questo.

Punteggio: [7/30]

$$k + k_i \geq n \quad i = \frac{n-k}{k} = \frac{n}{k} - 1 \quad \# \text{ ALLOCAZIONI RISPETTO A } n$$

$$C = nC_0 + \left(\frac{n}{k} - 1\right) Ca + \sum_{j=0}^{\frac{n}{k}-1} k_j = nC_0 + \left(\frac{n}{k} - 1\right) Ca + k \frac{\left(\frac{n}{k} - 1\right)\left(\frac{n}{k}\right)}{2}$$

/ \ COSTO TUTTE
 COSTO n COPIE DOPO
 INSERIMENTI DELLE
 $\frac{n}{k} - 1$ ALLOC.
\ L' ALLOCA.

$$= nC_0 + \left(\frac{n}{k} - 1\right) Ca + \frac{(n-k)n}{2} \Rightarrow \frac{n^2 - kn}{2} \approx O(n^2)$$

Quesito 3:

Una società di telecomunicazioni deve realizzare una dorsale in fibra ottica destinata a garantire la connettività tra n località principali. Il costo per stabilire un collegamento tra due località u e v è direttamente proporzionale alla distanza chilometrica w_{uv} tra di esse.

L'obiettivo è progettare un algoritmo `backbone` che restituisca le coppie di località da connettere mediante un collegamento in fibra ottica in modo tale che: 1) ogni località sia raggiungibile da ogni altra località mediante una successione di tratte in fibra ottica; 2) il costo complessivo delle tratte in fibra ottica da realizzare sia minimo. Per essere precisi, il costo è pari a $\sum_{(u,v) \in C} w_{uv}$, dove C è l'insieme delle coppie di località tra le quali verrà realizzato un collegamento in fibra ottica. Ciò premesso si risponda alle domande seguenti:

- Definire con precisione il grafo usato per rappresentare il problema, specificando cosa rappresentano i nodi, qual è l'insieme degli archi e cosa rappresentano gli eventuali pesi sugli archi. Descriverne ulteriori proprietà di interesse (grafo semplice, connesso, orientato o meno, ecc.). Infine, definire in termini di problemi su grafi l'input e l'output dell'algoritmo `backbone`.

Punteggio: [4/30]

- Si descriva (è sufficiente lo pseudo-codice o comunque una descrizione dettagliata) l'algoritmo `backbone`. La descrizione può essere anche ad alto livello concettuale ed usare primitive non elementari, ad esempio corrispondenti ad algoritmi noti studiati nel corso. Descrizioni basate su (pseudo) codice C/Java sono comunque considerate accettabili.

Punteggio: [3/30]

1)

È UN GRAFO NON ORIENTATO POICHÉ DA U A V C'È LA FIBRA E VICEVERSA, È CONNESSO POICHÉ OGNI LOCALITÀ È RAGGIUNGIBILE DA OGNI ALTRO

I NODI SONO LA LOCALITÀ
GLI ARCHI SONO IL COLLEGAMENTO TRA LE LOC
I PESI SONO LA DISTANZA CHILOMETRICA

L'INPUT SARÀ UN GRAFO E UN INSIEME C , MENTRE L'OUTPUT È UN INSIEME DI LOCALITÀ DA CONNETTERE (COPPIE DI ARCHI)

2)

È SUFFICIENTE ESEGUIRE UN CYCLE FOR SU TUTTI I NODI DEL GRAFO. A NODO EFFETTUO UNA BFS CHE RITORNERÀ L'INSIEME DEI NODI RAGGIUNGIBILI DALLA SORGENTE. SE LA SIZE DI QUESTO INSIEME È UGUALE ALLA SIZE DEL #NODI PASSO AL PROSSIMO, ALTRIMENTI INDIVIDUAO L'ARCO MANCANTE DA AGGIUNGERE ALLA LISTA IN OUTPUT.

Quesito 2: Algoritmi

1. Si consideri il problema di realizzare una funzione che associa hashcode a chiavi di tipo **Stringa**. Si supponga che la soluzione proposta preveda che l'hashcode di una stringa s sia ottenuto sommando gli interi che corrispondono ai codici ASCII dei primi 4 caratteri di s . Mostrare e discutere attraverso esempi i potenziali svantaggi di questa scelta se l'obiettivo è minimizzare il numero di collisioni tra chiavi diverse.

Occorre dare argomentazioni convincenti. Il punteggio dipenderà anche dalla completezza e dalla chiarezza espositiva. Non occorre scrivere tanto ma scrivere bene.

Punteggio: [8/30]

LE DUE STRINGHE "CASA" E "ASAC" SONO DIVERSE MA PRESENTANO LO STESSO HASHCODE, OTTENENDO COSÌ UNA COLLISIONE

PUÒ SUCCEDERE PERCHÉ SOMMIAMO IL # HASH CODE DELLA STRINGA

$$\begin{array}{ll} A = 1 & ABLD = 10 \\ B = ? & AA\Delta D = 10 \\ C = 3 & ACCL = 10 \\ D = 5 & \end{array} \quad \left. \begin{array}{l} \text{COLLISIONI} \\ \text{ } \end{array} \right\}$$

2. Si consideri una lista non ordinata rappresentata con un array. L'array è inizializzato a una dimensione pari a 1. Si consideri ora una successione di n inserimenti. Il generico inserimento ha costo costante *a meno che l'array non sia già pieno*. Se l'array è pieno l'inserimento determina i) l'allocazione di un *nuovo* array di dimensione $3x$ se x era la dimensione precedente dell'array, ii) la copia del contenuto del vecchio array nel nuovo (la copia del *singolo* valore dal vecchio array al nuovo ha costo costante), iii) l'inserimento del nuovo elemento (nel nuovo array).

- o a) Si calcoli il costo temporale complessivo asintotico della successione degli n inserimenti (si supponga che non vi siano mai rimozioni di elementi).
- o b) Si calcoli il costo ammortizzato per elemento.

Occorre dare un'argomentazione quantitativa, rigorosa (prova) e chiara, giustificando adeguatamente la risposta. Il punteggio dipenderà soprattutto da questo.

Punteggio: [7/30]

$$\begin{array}{ll} \text{DIM } IN = 1 & \text{SE } ! \text{PIENO } \quad C(I) = c_0 = \text{COST} \\ & \text{SE } \text{PIENO } \quad C(I) = c_a + n + c_0 \end{array}$$

$$\begin{aligned} C(n \text{ INS}) &= c_0 (\text{PRIMO INS}) + c_a + 1 + c_0 (\text{TERZO}) + c_a + 3 + c_0 \\ &\quad + C(n - 4) = \end{aligned}$$

$$3^i = \text{DIM ARRAY DOPO GLI INSER} \quad 3^i \geq n \quad i = \log_3 n$$

$$c_0 = \text{INSERIRI}$$

$$c_a = \text{ALLOCAZ}$$

$$\sum_{j=0}^{\log_3 n} 3^j < 3^{\log_3(n+1)}$$

COPIE

$$C(n) = n c_0 + \log_3 n c_a + \sum_{j=0}^{\log_3 n} 3^j < n c_0 + \log_3 n c_a + 3^{\log_3(n+1)}$$

$$\hookrightarrow \cancel{n c_0} + \cancel{\log_3 n c_a} + (n+1) = O(n)$$

Quesito 3:

Si supponga di avere una lista non ordinata contenente n coppie (key, e) , dove e è il riferimento a un oggetto (ad esempio una pagina Web) e key è una chiave avente valore numerico positivo.

- Si proponga un algoritmo che sia il più efficiente possibile che, presa in input una lista come sopra, restituisca la sottolista ordinata contenente le k coppie il cui valore della chiave è più alto, in ordine decrescente rispetto al valore delle chiavi

Punteggio: [4/30]

- Si calcoli il costo temporale asintotico dell'algoritmo proposto.

Punteggio: [3/30]

Occorre descrivere chiaramente l'algoritmo proposto e dare un'argomentazione quantitativa e per il costo asintotico. Il punteggio dipenderà soprattutto da questo

```
VOID FUNCT (LINKEDLIST l) {
    LINKEDLIST LIST = NEW LINKEDLIST ();
    MERGE (l);
    FOR (INT i = LENGTH(l)-1 ; i > LENGTH-1 - k ; i--)
        LIST. ADDLAST (l[i]);
    RETURN LIST;
}
```

$$\text{COSTO MERGE} + \text{COSTO FOR} = O(n \log n) + O(k) = O(n \log n)$$

↓

NEL CASO PEGGIORÉ ($k=n$)
AVREMO $O(n)$

2. Sia T un albero binario con chiavi intere associate ai nodi. Si indichi con $T.\text{root}$ la radice dell'albero. Dato un nodo v , siano $v.\text{key}$, $v.\text{left}$ e $v.\text{right}$ rispettivamente la chiave associata a v e i suoi figli sinistro e destro. Scrivere lo pseudo-codice di un algoritmo che, dato T , restituisca true se T è un albero binario di ricerca (Binary Search Tree o BST), false altrimenti.

Occorre descrivere l'algoritmo con uno pseudo-codice chiaro. Il punteggio dipenderà dalla correttezza e dall'efficienza dell'algoritmo proposto, nonché dalla chiarezza della sua descrizione.

Punteggio: [7/30]

```

BOOLEAN IS-BST (TREE t){
    BOOLEAN RES;
    RES = IS-BST-AUX(t.ROOT);
    RETURN RES;
}

BOOLEAN IS-BST-AUX (NODE n){
    IF (n.IS-FOGLIA()) RETURN TRUE;
    IF (n.LEFT != NULL && n.RIGHT != NULL){
        IF (n.LEFT.KEY < n.KEY && n.RIGHT.KEY > n.KEY)
            RETURN TRUE && IS-BST-AUX(n.LEFT)
            && IS-BST-AUX(n.RIGHT);
        RETURN FALSE;
    }
    ELSE IF (n.LEFT != NULL){
        IF (n.LEFT.KEY < n.KEY)
            RETURN TRUE && IS-BST-AUX(n.LEFT);
        RETURN FALSE;
    }
    ELSE IF (n.RIGHT != NULL){
        IF (n.RIGHT.KEY > n.KEY)
            RETURN TRUE && IS-BST-AUX(n.RIGHT);
        RETURN FALSE;
    }
}

```

Quesito 3:

Scrivere in pseudo-codice un algoritmo *efficiente* che, dati due alberi binari di ricerca (Binary Search Tree o BST) T_1 e T_2 a chiavi intere, restituisca una *lista ordinata* contenente le chiavi contenute in *entrambi* i BST (intersezione). Per uniformità nella scrittura dello pseudo-codice, dato un BST T , si indichi con $T.root$ la radice di T . Inoltre, dato un nodo v , si usino $v.key$, $v.left$ e $v.right$ per indicare rispettivamente la chiave associata a v e i suoi figli sinistro e destro.

Note. Non è richiesto di verificare che gli alberi binari T_1 e T_2 in input all'algoritmo siano effettivamente BST, si supponga che lo siano sempre. Si noti anche che esiste un algoritmo avente costo asintotico lineare per risolvere il problema, ossia $O(n_1 + n_2)$, se n_1 e n_2 indicano rispettivamente il numero di chiavi contenute in T_1 e T_2 .

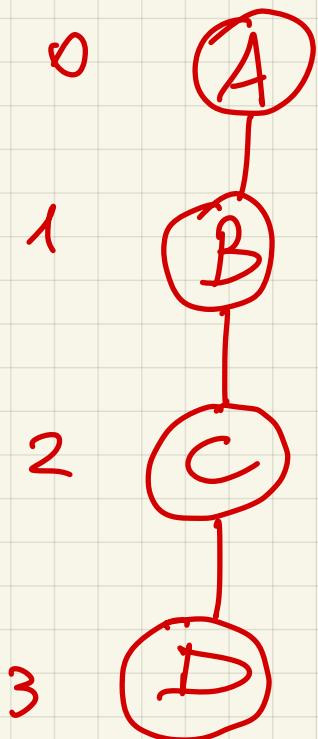
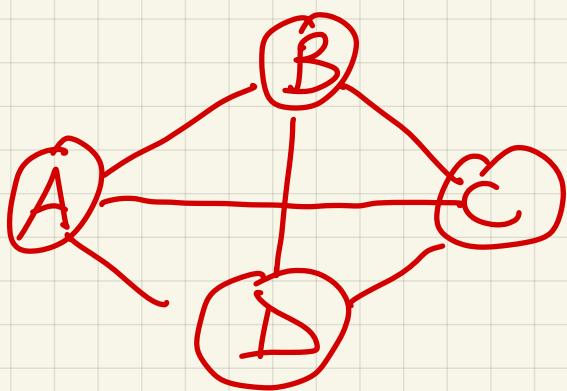
Occorre descrivere l'algoritmo con uno pseudo-codice chiaro. Il punteggio dipenderà dalla correttezza e dall'efficienza dell'algoritmo proposto, nonché dalla chiarezza della sua descrizione.

Punteggio: [7/30]

LISTA CRESCENTE T_1 LISTA CRESCENTE T_2 > MERGE

```
VOID LIST-MIN (NODE ROOT, LINKEDLIST L){  
    IF (n.LEFT != NULL)  
        LIST-MIN (n.LEFT, L);  
    L.ADD (n.KEY);  
    IF (n.RIGHT != NULL)  
        LIST-MIN (n.RIGHT, L);  
}
```

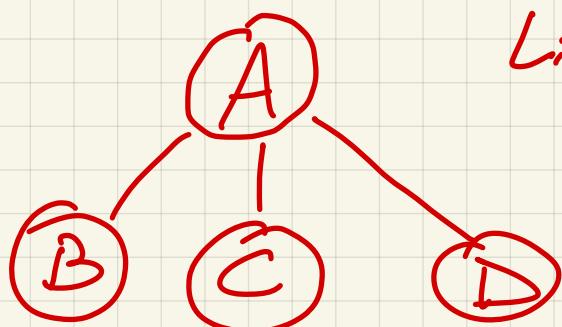
```
LINKEDLIST FUNCT (T1, T2) {  
    LINKEDLIST L1 = NEW LINKEDLIST();  
    LINKEDLIST L2 = NEW LINKEDLIST();  
    LINKEDLIST L = NEW LINKEDLIST();  
  
    LIST-MIN (T1.ROOT, L1);  
    LIST-MIN (T2.ROOT, L2);  
  
    MERGE (L, L1, L2);  
  
    RETURN L;  
}
```



algoritmo DFS

altezza albero = $n - 1$

foglie = 1



Livello max BFS:

1

Foglie: $n - 1$

GRADO MIN = 1
GRADO MAX = $n - 1$

Quesito 2: Algoritmi

1. Si consideri un grafo $G = (V, E)$ avente n vertici, *non diretto e completo* (ossia, avente un arco non diretto per ogni possibile coppia (u, v) di vertici). Si caratterizzino le proprietà dell'albero di visita DFS (Depth First Search) per questo grafo a partire dal generico vertice $s \in V$. In particolare, caratterizzare, *motivando* brevemente ogni risposta:

- o L'altezza dell'albero di visita
- o Il numero di foglie dell'albero di visita
- o Il grado (numero di archi incidenti in un nodo) minimo e il grado massimo dell'albero di visita

Punteggio: [4/30]

Rispondere alle stesse domande per la visita BFS (Breadth First Search).

Punteggio: [4/30]

- 1) - L'ALTEZZA SARÀ $n-1$, PERCHÉ LA DFS ESPORA OGNI VERTICE IN PROFONDITÀ FINCHE NON RAGGIUNGE UNA FOGLIA, QUINDI RISALE AL VERTICE PREL E CONTINUA L'ESPLORAZIONE FINO A QUANDO NON HA VISITATO TUTTI I VERTICI
- # FOGLIE = # VERTICI CHE MANNO GRADO 1 NEL GRAFO.
IN UN GRAFO COMPLETO OGNI VERTICE HA GRADO $n-1$, TRAMMESTO PER SE STESSO, QUINDI # FOGLIE = $n-1$
-

2) -

2. Sia T un albero binario con chiavi intere associate ai nodi. Si indichi con $T.\text{root}$ la radice dell'albero. Dato un nodo v , siano $v.\text{key}$, $v.\text{left}$ e $v.\text{right}$ rispettivamente la chiave associata a v e i suoi figli sinistro e destro. Scrivere lo pseudo-codice di un algoritmo che, dato T , restituisca true se T le chiavi associate ai nodi di T soddisfano la proprietà di ordinamento di un heap minimale, false altrimenti. Si noti che non è richiesto di verificare che T sia un albero binario completo.

Occorre descrivere l'algoritmo con uno pseudo-codice chiaro. Il punteggio dipenderà dalla correttezza e dall'efficienza dell'algoritmo proposto, nonché dalla chiarezza della sua descrizione.

Punteggio: [7/30]

```
BOOLEAN IS-MINH (TREE t){  
    BOOLEAN RES;  
    RES = IS-MINH-AUX (t.ROOT);  
    RETURN RES;  
}
```

```
BOOLEAN IS-MINH-AUX (NODE n){  
    IF (n.IS-FOGNA()) RETURN TRUE;  
    IF (n.LEFT != NULL && n.RIGHT != NULL){  
        IF (n.LEFT.KEY > n.KEY && n.RIGHT.KEY > n.KEY)  
            RETURN TRUE && IS-MINH-AUX (n.LEFT)  
            && IS-MINH-AUX (n.RIGHT);  
        RETURN FALSE;  
    }  
    ELSE IF (n.LEFT != NULL){  
        IF (n.LEFT.KEY > n.KEY)  
            RETURN TRUE && IS-MINH-AUX (n.LEFT);  
        RETURN FALSE;  
    }  
    ELSE IF (n.RIGHT != NULL){  
        IF (n.RIGHT.KEY > n.KEY)  
            RETURN TRUE && IS-MINH-AUX (n.RIGHT);  
        RETURN FALSE;  
    }  
}
```

Quesito 3:

Si supponga di avere due liste ℓ_1 ed ℓ_2 , *ordinate* in modo crescente, contenenti rispettivamente n_1 e n_2 stringhe (l'ordinamento è ovviamente quello lessicografico). Si supponga che una stringa possa apparire *al più una volta* in ciascuna lista.

- Si proponga e *si scriva lo pseudo-codice* del più efficiente algoritmo possibile che, prese in ingresso le due liste, restituisca una lista ℓ *ordinata*, contenente le stringhe che compaiono in ℓ_1 ma non in ℓ_2 . Ad esempio, se $\ell_1 = \{a, ac, b, bc, d, eh\}$ e $\ell_2 = \{a, ad, bc, eh\}$, allora $\ell = \{ac, b, d\}$.

Nota: a parte variabili scalari di appoggio, non si possono usare ulteriori strutture dati oltre alle liste.

Punteggio: [4/30]

- Si calcoli il costo asintotico dell'algoritmo proposto.

Punteggio: [3/30]

Occorre dare un'argomentazione quantitativa per il costo asintotico. Il punteggio dipenderà molto dalla chiarezza espositiva e dalla solidità delle argomentazioni proposte, nonché dalla semplicità dell'algoritmo proposto.

LINKEDLIST FUNCT (ℓ_1, ℓ_2, n_1, n_2) {
 L RES = NEW LIST;
 FOR (INT i=0; i < n_1; i++) { O(n_1)
 STRING s = $\ell_1[i]$
 IF (!BINARY RESEARCH(s)) O($\log_2 n_2$)
 L . ADD LAST(s);
 }
 RETURN RES;
}

$$\downarrow O(n, \log_2 n_2)$$

```
if( n.isfolia ) {  
    return 1;  
}  
else if ( n.left != null and n.right != null ) {  
    return 1 + max ( altezza (n.left), altezza (n.right) )  
}  
else if ( n.left == null ) {  
    return 1 + altezza (n.left)  
}  
else if ( n.right == null ) {  
    return 1 + altezza (n.right)  
}  
}
```