esercizi Mori

Esercizio. Dato un array che contiene solo numeri negativi e positivi (nessun valore pari a zero), riorganizzarlo in modo che tutti i numeri negativi stiano a sinistra di quelli positivi.

```
DEF ES1(A):   A e' l'array
    INT POS_TO_REPLACE = 0;
    FOR i IN RANGE(LEN(A)); itera per la lunghezza di A
        IF(A[i] < 0):
            TEMP= A[i];
            A[i] = A[POS_TO_REPLACE];
            A[POS_TO_REPLACE] =TEMP;
            POS_TO_REPLACE += 1;
```

COSTO COMPUTAZIONALE $\Theta(n)$

Esercizio. Progettare un algoritmo che, dato in input un array che rappresenta uno heap, restituisca il valore minimo. Fare le opportune considerazioni sul costo computazionale.

```
DEF MINHEAP(A):
    INT RANGE_TO_SEARCH = ABS(LEN(A) · 0,5);
    INT MIN= A[0];
    WHILE(RANGE_TO_SEARCH <= LEN(A)):
        IF(A[RANGE_TO_SEARCH] < MIN):
            MIN = A[RANGE_TO_SEARCH];
        RANGE_TO_SEARCH += 1;
```

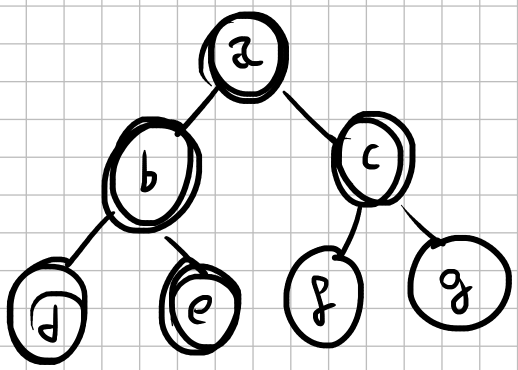
ha costo $\Theta(n)$

Esercizio. Scrivere le funzioni Enqueue e Dequeue per una coda con priorità implementata su Max Heap (in cui la chiave massima ha priorità più alta).

Essendo la radice l'elemento con priorità maggiore, si estrarrà quello, chiamando poi heapify sulla radice

* DA CONTINUARE

Esercizio. Implementazione della visita in pre-order su un albero binario memorizzato con la notazione posizionale.



[a, b, c, d, e, f, g]

1 2 5 3 4 6 7

→ CONTR. NODO E VISITO

ha figli? VISITO DESTRO

VISITA:

VISITO N

ESISTE N→R? SE SI VISITO N→R $\quad 2i+1 \quad 2i+2$

ESISTE N→L? SE SI VISITO N→L

```
DEF VISITA_IN_ORDINE(A, i)
    PRINT(A[i]);   nodo VISITATO
    IF( LEN(A) > 2·i + 1 ):
        VISITA_IN_ORDINE(A, 2·i + 1);   VISITA SINISTRO
    IF( LEN(A) > 2·i + 2):
        VISITA_IN_ORDINE(A, 2·i + 2);   VISITA DESTRO
```

Esercizio.

In un albero binario, si definisca *sbilanciamento* di un nodo il valore assoluto tra il numero di nodi nel suo sottoalbero sinistro ed il numero di nodi nel suo sottoalbero destro.

Dato un albero binario di cui si conosce il puntatore alla radice, trovare il massimo tra gli sbilanciamenti dei suoi nodi.

La funzione deve essere ricorsiva.

definisco prima 2 funzioni, una per calcolare il numero di figli di un nodo, e l'altra per lo sbilanciamento di un nodo.

```
DEF NUM_FIGLI(R):    R è la radice
    INT A = 0;
    IF(R->LEFT != NONE):
        A += 1 + NUM_FIGLI(R->LEFT);
    IF(R->RIGHT != NONE):
        A += 1 + NUM_FIGLI(R->RIGHT);
    RETURN A;
```

```
DEF SBILANCIAMENTO(R):
    RETURN ABS(NUM_FIGLI(R->LEFT)-NUM_FIGLI(R->RIGHT));
```

adesso ho bisogno di visitare ogni nodo, e calcolarne lo SBILANCIAMENTO.

```
DEF RIEMPIMENTO(A, R):
    A.ENQUEUE(SBILANCIAMENTO(R));
    IF(R->RIGHT != NONE):
        RIEMPIMENTO(A, R->RIGHT);
    IF(R->LEFT != NONE):
        RIEMPIMENTO(A, R->LEFT);
```

useremo una coda nella quale inserire tutti gli SBILANCIAMENTI. A e' il puntatore al primo elemento della coda.

```
DEF MAX_SB(R):
    INT* A; // inizializzo il puntatore della CODA
    RIEMPIMENTO(A, R);
    i = A;
    MAX = 0;
    WHILE(i->VALUE != NONE):
        IF (i->VALUE > MAX):
            MAX = i->VALUE;
        i = i->NEXT;
    RETURN MAX;
```

Esercizio. Siano dati due array A e B, composti da $n \geq 1$ ed $m \geq 1$ numeri reali, rispettivamente. Gli array sono entrambi ordinati in senso crescente. A e B non contengono valori duplicati; tuttavia, uno stesso valore potrebbe essere presente una volta in A e una volta in B.
Progettare un algoritmo iterativo efficiente che stampi i valori che appartengono all'unione (intesa in senso insiemistico) di A e di B.
Ad esempio, se $A = [1,3,4,6]$ e $B = [2,3,4,7]$, l'algoritmo deve stampare $1, 2, 3, 4, 6, 7$.
Di tale algoritmo:

- Si dia una descrizione a parole e si scriva lo pseudocodice;
- Si determini il costo computazionale, in funzione di n ed m;

L'idea è quella di confrontare i due array già ordinati ed unirli, similmente per come avviene nella funzione FONDI del MERGE SORT, non considerando duplicati, tramite una apposita funzione di CONTROLLO.

```
DEF CHECK_DUP(A, K) :        A è l'array, K l'elemento
    INT i = 0;
    WHILE (i < LEN(A)) :
        IF (K == A[i]) :
            RETURN TRUE;
        i += 1;
    RETURN FALSE;
```

```
DEF FONDI(A, B):
    INT C[LEN(A)+LEN(B)]; // UN ARRAY LUNGO AL PIU' LA
                             SOMMA TRA A e B.
    INT i = 0;
    INT J = 0;
    INT Z = 0;
    WHILE((i < LEN(A)) AND (J < LEN(B)):    Θ(n)
        IF (A[i] < B[J]):
            IF (NOT CHECK_DUP(C, A[i])):    O(n)
                C[Z] = A[i];
            i += 1;
            Z += 1;
        ELSE IF(B[J] < A[i])
            IF (NOT CHECK_DUP(C, B[J])):
                C[Z] = B[J];
            J += 1;
            Z += 1;
        ELSE IF(B[J] == A[i]):
            IF (NOT CHECK_DUP(C, B[J])):
                C[Z] = B[J];
            J += 1;
            Z += 1;
            i += 1;

    IF (i == LEN(A)):
        WHILE(J < LEN(B)):
```

```
            IF (NOT CHECK_DUP(C, B[J]):
                C[Z] = B[J];
            J += 1;
            Z += 1;
    IF (J == LEN(B)):
        WHILE(i < LEN(A)):
            IF (NOT CHECK_DUP(C, A[i]):
                C[Z] = A[i];
            i += 1;
            Z += 1;
// HO C COMPLETO, LO STAMPO:

    Z = 0;
    WHILE (Z < LEN(C)):
        IF(C[Z] == NULL):
            BREAK;
        PRINT(C[Z]);

        Z += 1;
```

$\Theta(n^2)$

Esercizio. Si risolva la seguente equazione di ricorrenza con due metodi diversi, per ciascuno dettagliando il procedimento usato:

$T(n) = T(n/2) + \Theta(n^2)$ se $n > 1$

$T(1) = \Theta(1)$

tenendo conto che n è una potenza di 2.

metodo iterativo

$$T(n) = \left[T\left(\frac{n}{2^2}\right) + \Theta\left(\left[\frac{n}{2}\right]^2\right)\right] + \Theta(n^2)$$

$$T(n) = T\left(\frac{n}{2^k}\right) + \sum_{i=0}^{k-1} \Theta\left(\left[\frac{n}{2^i}\right]^2\right)$$

$$T(n) = T\left(\frac{n}{2^k}\right) + \Theta(n^2) \sum_{i=0}^{k-1} \left(\frac{1}{4}\right)^i$$ fino a $k = \log(n)$

$$T(n) = \Theta(\log(n)) + \Theta(n^2) \left[\frac{\left(\frac{1}{4}\right)^{\log(n)} - 1}{-\frac{3}{4}} \right]$$

$$T(n) = \Theta(\log(n)) + \Theta(n^2) \left[\left[\frac{1}{n^2} - 1\right] \cdot \left[-\frac{4}{3}\right] \right]$$

$$T(n) = \Theta(\log(n)) + \Theta(n^2) \left[\frac{4}{3n^2} + \frac{4}{3} \right]$$

$$T(n) = \Theta(\log(n)) + \Theta(1) \left[\frac{4}{3} + \frac{n^2 \cdot 4}{3} \right]$$

$$T(n) = \Theta(\log(n)) + \Theta\left(\frac{4 + 4n^2}{3}\right) = \Theta(n^2)$$

metodo principale

$f(n) = \Theta(n^2)$ $\quad f(n) = \Omega(1)$ $\quad$ ESISTE $c < 1$ esempio $c = \frac{1}{1.5}$ $\quad$ QUINDI $T(n) = \Theta(f(n))$

$n^{\log_b a} = n^0 = 1$ $\quad$ E $\Theta\left(\frac{n^2}{2}\right) \le c \cdot \Theta(n^2)$

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