1. Esercizi di programmazione

Esercizio 1.1 – Cerca duplicati

Dato un vettore di n interi, scrivere un algoritmo che restituisca vero se un elemento compare due volte. Determinare una limitazione inferiore e una superiore alla complessità di questo problema.

Esercizio 1.2 – Cerca la coppia

Dato un array $A[1 \dots n]$ di interi e un intero v, scrivere un algoritmo che determini se esistono due elementi in A la cui somma sia esattamente v.

Esercizio 1.3 – Cerca la coppia porta-sfiga

Dato un array $A[1 \dots n]$ di interi e un intero v, scrivere un algoritmo che determini se esistono due elementi in A la cui somma sia esattamente 17.

2. Limiti superiori e inferiori

Esercizio 2.1

Trovare un limite asintotico superiore e un limite asintotico inferiore alla seguente ricorrenza, facendo uso del metodo di sostituzione:

$$T(n) = 2T(n/8) + 2T(n/4) + n$$

Esercizio 2.2 – Cicli FOR!

Si dimostri, per induzione, che $\sum_{i=1}^{n} i^h$ è $O(n^{h+1})$.

3. Alberi

Dalla visita all'albero

Gli ordini di visita di un albero binario di 9 nodi sono i seguenti:

- A, E, B, F, G, C, D, I, H (anticipato)
- B, G, C, F, E, H, I, D, A (posticipato)
- B, E, G, F, C, A, D, H, I (simmetrico).

Si ricostruisca l'albero binario e si illustri *brevemente* il ragionamento.

Visite!

- Dato un albero radicato T, calcolare la sua altezza
- Dato un albero radicato T, calcolare il numero totale di nodi
- Dato un albero radicato T, stampare tutti i nodi a profondità h

Larghezza albero

La larghezza di un albero ordinato è il numero massimo di nodi che stanno tutti al medesimo livello. Si fornisca una funzione che calcoli in tempo ottimo la larghezza di un albero ordinato T di n nodi.

Pozzo universale

Data una rappresentazione con matrice di adiacenza, progettare un algoritmo che opera in tempo $\Theta(n)$ in grado di determinare se un grafo orientato contiene un pozzo universale: ovvero un nodo con out-degree uguale a zero e in-degree uguale a $n-1$. È possibile ottenere la stessa complessità con liste di adiacenza?

Spoiler alert!

Sommatorie

- Caso base ($h = 0$):

$$\sum_{i=1}^{n} i^0 = \sum_{i=1}^{n} 1 = n = n^{h+1}$$

- Passo induttivo. Supponiamo che la proprietà sia vera per ogni $k < h$; vogliamo dimostrare che la proprietà è vera per h:

$$\sum_{i=1}^{n} i^h = \sum_{i=1}^{n} i^{h-1} i \leq \sum_{i=1}^{n} i^{h-1} n = n \sum_{i=1}^{n} i^{h-1} = nO(n^h) = O(n^{h+1})$$

Larghezza (1)

```
integer larghezza(TREE t)
integer larghezza ← 1
integer level ← 1
integer count ← 1
QUEUE Q ← Queue()
Q.enqueue(t)
t.level ← 0
while not Q.isEmpty() do
    TREE u ← Q.dequeue()
    if u.level ≠ level then
        level ← u.level
        count ← 0
    count ← count + 1
    larghezza ← max(larghezza, count)
    TREE v ← u.leftmostChild()
    while v ≠ nil do
        v.level ← u.level + 1
        Q.enqueue(v)
        v ← v.rightSibling()
return larghezza
```

Larghezza (2)

```
integer larghezza(TREE t)
---
VECTOR count ← new VECTOR
larghezza(t, count, 0)
return max(count)
```

```
larghezza(TREE t, VECTOR count, integer level)
---
if t ≠ nil then
    count[level] ← count[level] + 1
    larghezza(t.left, count, level + 1)
    larghezza(t.right, count, level + 1)
```

Larghezza (3)

```
integer larghezza(TREE t)
integer count ← 1    % Numero di nodi da visitare del livello corrente; inizialmente la radice
integer larghezza ← 1    % Massimo larghezza trovata finora; inizialmente la radice
QUEUE Q ← Queue()
Q.enqueue(t)
while not Q.isEmpty() do
    TREE u ← Q.dequeue()
    TREE v ← u.leftmostChild()
    while v ≠ nil do
        Q.enqueue(v)
        v ← v.rightSibling()
    count ← count − 1
    if count = 0 then    % Nuovo livello
        count = Q.size()
        larghezza ← max(larghezza, count)
return larghezza
```

Pozzo universale

```
universalSink(integer[][] A)
i ← 1
candidate ← false
while i < n ∧ candidate = false do
    j ← i + 1
    while j ≤ n ∧ A[i, j] = 0 do
        j ← j + 1
    if j > n then
        candidate ← true
    else
        i ← j
rowtot = Σ_{j∈{1...n}−{i}} A[i, j]
coltot = Σ_{j∈{1...n}−{i}} A[j, i]
return rowtot = 0 ∧ coltot = n − 1
```