

### Esercizio 1

È facile osservare che un limite inferiore per  $T(n)$  è  $\Omega(2^n)$ , in quanto la componente non ricorsiva è pari a  $2^n$ . Proviamo quindi a dimostrare che  $T(n) = O(2^n)$ .

- Caso base:  $T(1) = 1 \leq c2^1$ , il che è vero è per  $c \geq 1/2$ .
- Ipotesi induttiva:  $T(n') \leq c2^{n'}$ , per ogni  $n' < n$ .
- Passo induttivo:

$$\begin{aligned} T(n) &= T(n/2) + 2^n \\ &\leq c2^{n/2} + 2^n \\ &\leq c2^n \end{aligned}$$

L'ultima disequazione è vera per  $c \geq \frac{2^{n/2}}{2^{n/2-1}}$ . Poichè  $\lim_{n \rightarrow +\infty} \frac{2^{n/2}}{2^{n/2-1}} = 1$  e  $\frac{2^{n/2}}{2^{n/2-1}} \leq 2$  per ogni  $n \geq 2$ , la disequazione è soddisfatta da  $c = 2$  e  $m = 2$ .

### Esercizio 2

Per valutare se un albero  $T$  è  $k$ -bilanciato, utilizziamo una procedura `balance(TREE T, int k)` che restituisce  $-1$  se  $T$  non è  $k$ -bilanciato, oppure l'altezza di  $T$  se  $T$  è  $k$ -bilanciato e questa proprietà vale per tutti i suoi sottoalberi.

---

```
int balance(TREE T, int k)
{
  if T == nil then
    return 0
  int L = balance(T.left, k)
  int R = balance(T.right, k)
  if L < 0 or R < 0 or |L - R| > k then
    return -1
  else
    return max(L, R) + 1
}
```

---

Trattandosi di visita in profondità, la complessità della procedura è  $O(n)$ , dove  $n$  è il numero di nodi dell'albero.

### Esercizio 3

Un modo semplice, ma poco efficiente per scrivere tale algoritmo è il seguente:

---

```
boolean 4cycles(GRAPH G)
{
  foreach u ∈ G.V() do
    foreach v ∈ u.adj() do
      foreach w ∈ v.adj() - {u, v} do
        foreach x ∈ w.adj() do
          if u ∈ x.adj() then
            return true
  return false
}
```

---

Questa procedura ha una complessità pari a  $O(n^5)$ .

È possibile costruire una matrice di adiacenza  $M$  di dimensione  $n \times n$  tale per cui  $M[u][v] = \mathbf{true}$  se e solo se esiste  $w$  tale per cui  $(u, w) \in E$  e  $(w, v) \in E$ . In quel caso, esiste un ciclo di dimensione 4 se esistono due nodi  $u, v$  tali per cui  $M[u][v] = M[v][u] = \mathbf{true}$ . Tale algoritmo ha complessità  $O(n^3)$  per la costruzione della matrice.

---

**boolean** 4cycles(GRAPH  $G$ )

---

```

int[][]  $M = \mathbf{new int}[1 \dots G.n][1 \dots G.n]$ 
{ Costruzione della matrice di adiacenza  $M$ , costo  $O(n^3)$  }
foreach  $u \in G.V()$  do
  | foreach  $v \in G.V()$  do
  | |  $M[u][v] = \exists w \in G.V() - \{u, v\} : w \in u.adj(v)$  and  $v \in u.adj(w)$ 
foreach  $u \in G.V()$  do
  | foreach  $v \in G.V()$  do
  | | if  $M[u][v]$  and  $M[v][u]$  then
  | | | return true
return false

```

---

#### Esercizio 4

Il problema si risolve semplicemente utilizzando la programmazione dinamica. Sia  $T(n)$  il numero di modi in cui è possibile scegliere  $n$  scalini; allora  $T(n)$  può essere espresso nel modo seguente:

$$T(n) = \begin{cases} 1 & n = 0 \\ \sum_{k=1}^{\min\{n,4\}} T(n-k) & n > 0 \end{cases}$$

Un algoritmo basato su programmazione dinamica per risolvere il problema è il seguente:

---

**int** countStairs(**int**  $n$ )

---

```

int[]  $T = \mathbf{new int}[0 \dots n]$ 
 $T[0] = 1$ 
for  $i = 1$  to  $n$  do
  |  $T[i] = 0$ 
  | for  $k = 1$  to  $\min(i, 4)$  do
  | |  $T[i] = T[i] + T[i - k]$ 
return  $T[n]$ 

```

---

La complessità della procedura è pari a  $O(n)$ .

Notate che  $T(n)$  è uguale al valore del  $(n + 1)$ -esimo "Tetranacci number"

<http://mathworld.wolfram.com/TetranacciNumber.html>