Per fare un albero (binario di ricerca) ci vuole...

Dato un vettore V di n interi ordinati e distinti, scrivere una procedura che costruisce un albero binario di ricerca di altezza minima.
Discuterne la correttezza e la complessità.

Good vs bad guys

Fra ogni coppia di wrestler professionisti può esserci una rivalità oppure no. Per ragioni di marketing, è una buona idea dividere i wrestler professionisti in due gruppi, "buoni" e "cattivi", e farli combattere fra di loro.
Supponete di avere in input un insieme di rivalità, rappresentate come un vettore di coppie (x, y), dove x e y sono identificatori di wrestler compresi fra 1 ed n.
Scrivere un algoritmo che restituisca **true** se è possibile suddividere i wrestler professionisti in due sottoinsiemi non vuoti ("buoni" e "cattivi"), non necessariamente della stessa dimensione, in modo tale che non ci siano rivalità all'intero dei due gruppi; **false** altrimenti.

Grafi – Pozzo universale

- Un pozzo universale è un nodo con out-degree uguale a zero e in-degree uguale a $n - 1$.
- Dato un grafo orientato G rappresentato tramite matrice di adiacenza, scrivere un algoritmo che opera in tempo $\Theta(n)$ in grado di determinare se G contiene un pozzo universale.
- È possibile ottenere la stessa complessità con liste di adiacenza?

Ricorrenza $4T(\sqrt{n}) + \log^2 n$

Si ottengano limiti superiori e inferiori per la seguente ricorrenza:

$$T(n) = \begin{cases} 4T(\lfloor \sqrt{n} \rfloor) + \log^2 n & n > 1 \\ 1 & n = 1 \end{cases}$$

Spoiler alert!

Per fare un albero (binario di ricerca) ci vuole...

TREE build-tree-rec(**int**[ ] A, **int** i, **int** j)

```
if i ≤ j then
    int m = ⌊(i + j)/2⌋
    TREE T = new TREE
    T.left = build-tree-rec(A, i, m − 1)
    T.right = build-tree-rec(A, m + 1, j)
    T.key = V[m]
    return T
else
    return nil
```

L'equazione di ricorrenza è pari a: $T(n) = 2T(n/2) + 1$, che dà origine a $T(n) = \Theta(n)$.

Good vs bad guys

Il problema proposto è quello della bi-colorazione di un grafo, che è possibile se e solo se il grafo è bipartito. La bi-colorazione può essere ottenuta facilmente tramite una visita DFS:

---

boolean good-bad-guys(GRAPH G)

```
int[] C = new int[1...G.n]
for u = 1 to n do
    C[u] = -1
foreach u ∈ G.V() do
    if C[u] < 0 then
        if not dfsVisit(G, u, 0, C) then
            return false
return true
```

Good vs bad guys

boolean dfsVisit(GRAPH G, **int** u, **int** c, **int**[ ] C)

```
C[u] = c
foreach v ∈ G.adj(u) do
    if C[v] < 0 then
        if not dfsVisit(G, v, 1 − c, C) then
            return false
    else if C[v] == c then
        return false
return true
```

Pozzo universale

universalSink(**int**[ ][ ] A)

```
int i = 1
boolean candidate = false
while i < n and not candidate do
    int j = i + 1
    while j ≤ n and A[i][j] == 0 do
        j = j + 1
    if j > n then
        candidate = true
    else
        i = j
rowtot = Σ_{j∈{1...n}} A[i][j]
coltot = Σ_{j∈{1...n}} A[j][i]
return rowtot = 0 ∧ coltot = n − 1
```

Ricorrenza $4T(\lfloor\sqrt{n}\rfloor) + \log^2 n$

Poniamo $n = 2^k$. Sostituendo nella ricorrenza otteniamo:

$$\begin{aligned} T(2^k) &= 4T(\sqrt{2^k}) + \log^2 2^k \\ &= 4T(2^{k/2}) + k^2 \end{aligned}$$

Sostituiamo quindi la variable $T(2^k)$ con $S(k)$ e otteniamo (tramite Master Theorem)

$$S(k) = 4S(k/2) + k^2 = \Theta(k^2 \log k)$$

Ri-esprimendo la funzione nei termini di $T(n)$ e $k = \log n$, otteniamo

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