Ricorrenza, algoritmo

Ricorrenza con minimo

Determinare un limite superiore e inferiore alla seguente ricorrenza tramite il metodo di sostituzione:

$$T(n) = \begin{cases} \min\limits_{1 \le k \le n-1} \{T[k] + T(n-k)\} + 1 & n > 1 \\ 1 & n \le 1 \end{cases}$$

Sali e scendi

Sia A un vettore contenente n valori interi *distinti*, tale per cui esiste un indice k tale per cui:

- Gli elementi $A[1 \ldots k]$ sono ordinati in senso crescente;
- Gli elementi $A[k \ldots n]$ sono ordinati in senso decrescente.

Scrivere un algoritmo efficiente per individuare l'indice k e discuterne la complessità.

Connetti il grafo

- Progettare un algoritmo efficiente che dato un grafo non orientato, restituisca il numero minimo di archi da aggiungere per renderlo connesso.
- Progettare un algoritmo efficiente che dato un grafo non orientato, aggiunga il numero minimo di archi necessari a renderlo connesso.
- Progettare un algoritmo efficiente che dato un grafo non orientato, restituisca vero se esso è un albero, false altrimenti.
- Progettare un algoritmo efficiente che dato un grafo non orientato che non è un albero, restituisca il numero minimo di archi che vanno rimossi e il numero minimo di archi che vanno aggiunti per renderlo un albero.

Spoiler Alert!

Connetti il grafo - 1

```
integer numberToConnect(Graph G)
integer[] id ← new integer[1...G.n]
foreach u ∈ G.V() do id[u] ← 0
integer conta ← 0
foreach u ∈ G.V() do
    if id[u] = 0 then
        conta ← conta + 1
        ccdfs(G, conta, u, id)
return conta − 1

ccdfs(Graph G, integer conta, Node u, integer[] id)
    id[u] ← conta
    foreach v ∈ G.adj(u) do
        if id[v] = 0 then
            ccdfs(G, conta, v, id)
```

Connetti il grafo - 2

numberToConnect(GRAPH G)

```
integer[] id ← new integer[1...G.n]
integer[] rappresentante ← new integer[1...G.n]
foreach u ∈ G.V() do id[u] ← 0
integer conta ← 0
foreach u ∈ G.V() do
    if id[u] = 0 then
        conta ← conta + 1
        rappresentante[conta] = u
        ccdfs(G, conta, u, id)
for i ← 2 to conta do
    G.insertEdge(rappresentante[i − 1], rappresentante[i])

ccdfs(GRAPH G, integer conta, NODE u, integer[] id)
    id[u] ← conta
    foreach v ∈ G.adj(u) do
        if id[v] = 0 then
            ccdfs(G, conta, v, id)
```