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

Si ottengano limiti superiori e inferiori per la seguente ricorrenza:

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

Grafi – Tutte le strade portano a Roma

Un vertice v in un grafo orientato G si dice di tipo "Roma" se ogni altro vertice w in G può raggiungere v con un cammino orientato che parte da w e arriva a v.

1. Descrivere un algoritmo che dati un grafo G e un vertice v, determina se v è un vertice di tipo "Roma" in G.
2. Descrivere un algoritmo che, dato un grafo G, determina se G contiene un vertice di tipo "Roma".

In entrambi i casi è possibile trovare un algoritmo con complessità $O(m+n)$, ma anche altre complessità verranno considerate.

Griglia quadrata

Si consideri un griglia quadrata $n \times n$ celle.

- Ogni cella è colorata con un colore in $\{1, 2, 3\}$
- Per semplicità, supponete che nella griglia sia presente almeno una cella di colore 1 e almeno una cella di colore 3.
- Supponete di partire da una cella di colore 1
- Ad ogni passo potete muovervi di una cella in alto, in basso, a destra o a sinistra
- L'obiettivo è raggiungere una cella con colore 3

Scrivere un algoritmo che prende in input una griglia rappresentata da una matrice di interi e restituisca il numero minimo di passi *necessari* per raggiungere una qualunque cella di colore 3 a partire da una qualunque cella di colore 1.
Discutere correttezza e complessità dell'algoritmo proposto.

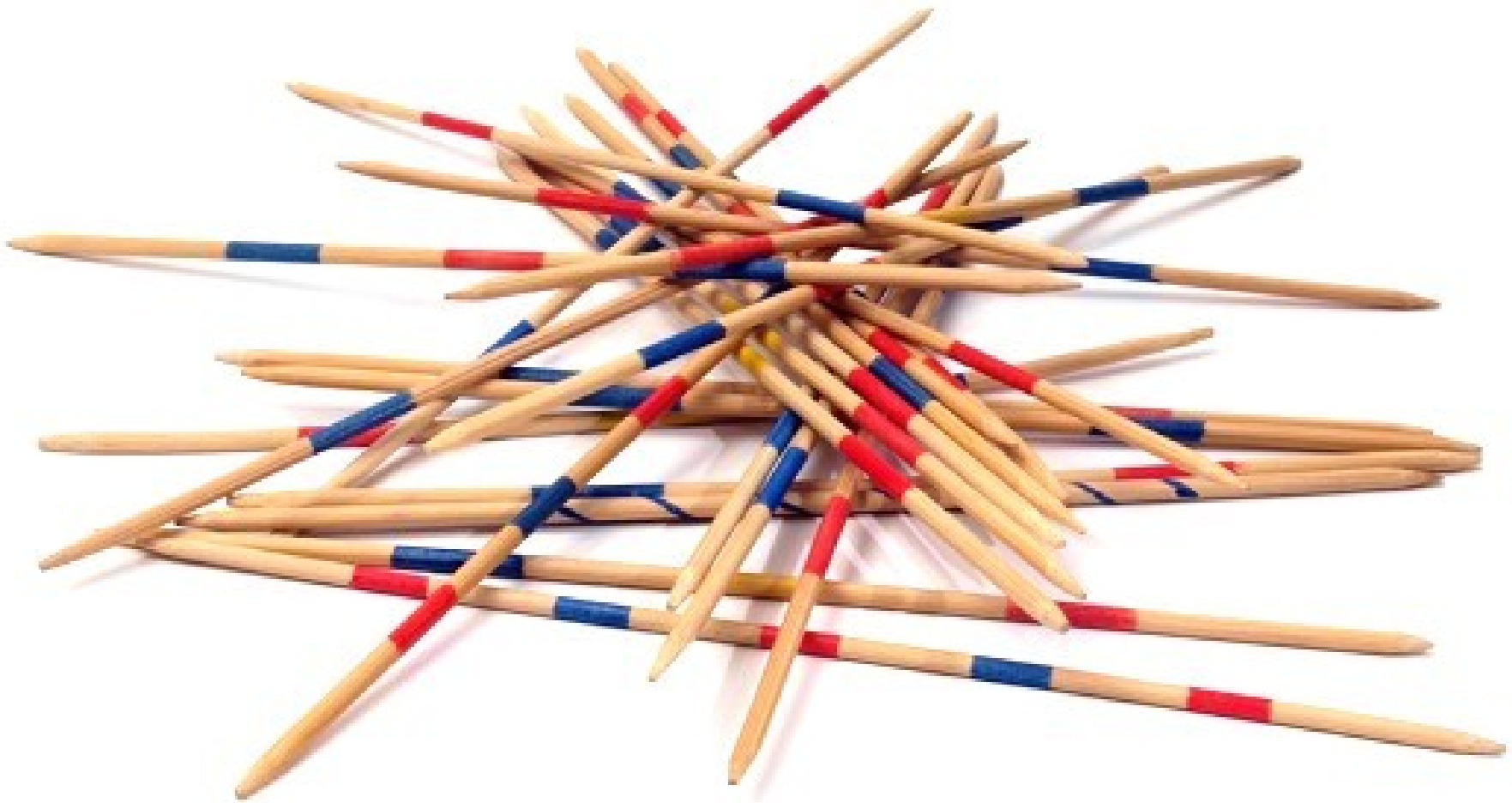
Griglia quadrata

Ad esempio, si consideri la matrice seguente:

1	2	2	3
2	1	2	3
2	2	2	**3**
3	2	**1**	**2**

La risposta da dare è 2, perchè non esistono celle 1 e 3 adiacenti ma esistono percorsi formati da due passi (come quello evidenziato in grassetto, che però non è l'unico).

Shangai



Shangai

- Nel gioco dello Shangai, un bastoncino può essere rimosso se nessun altro bastoncino lo sovrasta.
- Una volta rimosso, è possibile che i bastoncini che erano sovrastati da esso possano essere rimossi.
- È anche possibile tuttavia che ad un certo punto nessun bastoncino possa essere rimosso, in quanto sovrastato da altri bastoncini.
- L'input è dato dai vettori X e Y di dimensione m, contenenti numeri da 1 a n. I vettori vanno interpretati in questo modo: per ogni indice i, il bastoncino $X[i]$ sovrasta il bastoncino $Y[i]$.
- Scrivere un algoritmo che prenda in input i vettori X, Y oltre alle dimensioni n ed m, e restituisca **true** se e solo se è possibile rimuovere tutti i bastoncini presenti, **false** altrimenti.

Anagrammi

Un'anagramma è una parola o frase ottenuta riarrangiando le lettere di un'altra parola o frase. Per esempio, "notremors" è un anagramma di "montresor".
Si supponga di avere in input un vettore di n stringhe di lunghezza massima k; si scriva un algoritmo che stampi in output tutti i gruppi di anagrammi contenuti in queste n stringhe. Se ne discuta correttezza e complessità.
Esempio di input: `rosa, pippo, poppi, raso, orsa, giappone`
Esempio di output:

```
rosa, raso, orsa
pippo, poppi
giappone
```

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?

Spoiler alert!

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

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

Tutte le strade portano a Roma – 2012/05/03

Operando sul grafo trasposto – un nodo è Roma se da esso è possibile raggiungere tutti i nodi.

---

boolean isRoma(GRAPH G^T, NODE v)

---

boolean[ ] id = **new int**[$1 \ldots G^T.n$] = $\{0\}$
ccdfs($G^T, 1, v, id$)
foreach $u \in G^T$.V() **do**
    if $id[u] == 0$ **then**
        return false
return true

Tutte le strade portano a Roma – 2012/05/03

E' possibile ripetere Roma a partire da tutti i nodi, con un costo pari a $O(n(m+n)) = O(mn)$. Altrimenti, si consideri un ordinamento topologico del grafo trasposto: se il primo non è di tipo Roma, allora nessuno lo è; se è di tipo Roma, allora potrebbero essercene altri ma basta il primo. Chiamiamo quindi isRoma() a partire da esso.

---

boolean Roma(GRAPH G^T)

STACK S = topsort(G^T)
NODE $v = S$.pop()
return isRoma(G^T, v)

Il costo è $O(m+n)$.

Griglia quadrata

```
int grid(int[][] M, int n)
int[] dr = [−1, 0, +1, 0]                    % Mosse possibili sulle righe
int[] dc = [0, −1, 0, +1]                    % Mosse possibili sulle colonne
int[] distance = new int[1...n][1...n]
Queue Q = Queue()
for r = 1 to n do
    for c = 1 to n do
        distance[r][c] = iif(M[r][c] == 1, 0, −1)
        if M[r][c] == 1 then
            Q.enqueue(⟨r, c⟩)

[...]
```

Griglia quadrata

int grid(**int**[][] M, **int** n)

```
[...]
while not Q.isEmpty() do
    int, int r, c = Q.dequeue()    % Riga, colonna della cella visitata correntemente
    for i = 1 to 4 do
        nr = r + dr[i]                                  % Nuova riga
        nc = c + dc[i]                                  % Nuova colonna
        if 1 ≤ nr ≤ n and 1 ≤ nc ≤ n and distance[nr][nc] < 0 then
            distance[nr][nc] = distance[r][c] + 1
            if M[nr][nc] == 3 then
                return distance[nr][nc]
            else
                Q.enqueue(⟨nr, nc⟩)
```

Shangai

boolean shangai(**int**[ ] X, **int**[ ] Y, **int** n, **int** m)

```
GRAPH G = Graph()
for i = 1 to n do
    G.addNode(i)
for i = 1 to m do
    G.addEdge(X[i], Y[i])
return not hasCycle(G)
```

Complessità: $O(m+n)$

Shangai

boolean hasCycle(GRAPH G)

int $clock = 0$
int[ ] dt = **new int**$[1 \ldots G.n] = \{0\}$
int[ ] ft = **new int**$[1 \ldots G.n] = \{0\}$
for $u = 1$ **to** $G.n$ **do**
    if $dt[u] == 0$ **and** hasCycle$(G, u, \&clock, dt, ft)$ **then**
        return true
return false

Anagrammi – 2011/09/07

```
anagrams(Item [][] S, int n)
Hash H = Hash()
for i = 1 to n do
    sorted = sort(S[i])
    Set S = H.lookup(sorted)
    if S == nil then
        S = Set()
    S.insert(S[i])
    H.insert(sorted, S)
foreach k ∈ H do
    Set S = H.lookup(k)
    print S
```

Il costo di questo algoritmo è $O(nk \log k + n)$.

Pozzo universale

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

int $i = 1$
boolean $candidate =$ **false**
while $i < n$ **and not** $candidate$ **do**
    int $j = i + 1$
    while $j \leq n$ **and** $A[i][j] == 0$ **do**
        $j = j + 1$
    if $j > n$ **then**
        $candidate =$ **true**
    else
        $i = j$
$rowtot = \sum_{j \in \{1 \ldots n\}} A[i][j]$
$coltot = \sum_{j \in \{1 \ldots n\}} A[j][i]$
return $rowtot == 0 \wedge coltot == n - 1$