Famiglia

Trovare i limiti superiori e inferiori più stretti possibili per la seguente famiglia di equazioni di ricorrenza, per valori di a interi positivi.

$$T(n) = \begin{cases} aT(\lfloor n/2 \rfloor) + n^{a-1} & n \geq 2 \\ 1 & n < 2 \end{cases}$$

Analisi – Problema – $T(m) + T(n-m)$

Si consideri la seguente equazione di ricorrenza:

$$T(n) = \begin{cases} T(m) + T(n-m) + 1 & n > m \\ 1 & n \leq m \end{cases}$$

dove m è una costante intera positiva.

Si trovino, tramite il metodo della sostituzione, un limite superiore ed un limite inferiore per $T(n)$. Fare particolare attenzione ai casi base.

Maggioranza

Scrivere una funzione

boolean hasMajority(**int**[ ] A, **int** n)

che prenda in input un vettore ordinato A contenente n interi e restituisca **true** se A contiene un valore di maggioranza, ovvero un valore che compare più di $n/2$ volte; restituisca **false** altrimenti. Soluzioni di costo computazionale lineare non verranno prese in considerazione.

Discutere correttezza e complessità dell'algoritmo proposto.

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).

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

Spoiler alert!

Famiglia (24/07/20)

Data la forma della ricorrenza, è possibile utilizzare il Master Theorem, versione base.

- Per $a = 1$, abbiamo $\alpha = \log_2 1 = 0$, $\beta = 0$; siamo nel secondo caso, $T(n) = \Theta(\log n)$.
- Per $a = 2$, abbiamo $\alpha = \log_2 2 = 1$, $\beta = 1$; siamo nel secondo caso, $T(n) = \Theta(n \log n)$.
- Per $a = 3$, abbiamo $\alpha = \log_2 3 < 2$, $\beta = 2$; siamo nel terzo caso, $T(n) = \Theta(n^2)$.
- In generale, è possibile dimostrare che $\log_2 a < a - 1$ per tutti i valori interi $a \geq 3$; siamo nel terzo caso e si ottiene $T(n) = \Theta(n^{a-1})$.

Ricorrenza $T(n) = T(m) - T(n-m) + 1$

- Ricordate che m è un valore costante;
- Il numero di chiamate ricorsive è $\lfloor n/m \rfloor$;
- Ogni chiamata costa $T(m) + 1$, ancora un valore costante.

Supponiamo quindi che $T(n) = \Theta(n)$.

Ricorrenza $T(n) = T(m) - T(n-m) + 1 = O(n)$

Passo induttivo: supponiamo per ipotesi induttiva che $T(k) \leq ck$ per ogni $k < n$. Proviamo che $\exists c > 0 : T(n) \leq cn$.

$$\begin{aligned} T(n) &= T(m) + T(n-m) + 1 \\ &\leq 1 + cn - cm + 1 \\ &\leq cn - cm + 2 \\ &\overset{?}{\leq} cn \end{aligned}$$

La disequazione è vera per $c \geq 2/m$; poichè $m \geq 1$, $c \geq 2$ è un valore accettabile per ogni m.

Ricorrenza $T(n) = T(m) - T(n-m) + 1 = O(n)$

Caso base: consideriamo tutti i casi per cui $1 \leq i \leq m$:

$$T(i) = 1 \leq c \cdot i \Leftrightarrow c \geq \frac{1}{i}$$

Le disequazioni su c derivanti dal caso base possono essere riassunte dal fatto che $c \geq 1$, poichè $1/i \leq 1$ per tutti i valori $i \geq 1$.

Considerando quindi sia la disequazione sul passo ricorsivo che sul caso base, otteniamo che $c \geq 2$.

Ricorrenza $T(n) = T(m) - T(n-m) + 1 = \Omega(n)$

Passo induttivo: supponiamo per ipotesi induttiva che $T(k) \geq ck$ per ogni $k < n$. Proviamo che $\exists c > 0 : T(n) \geq cn$.

$$\begin{aligned} T(n) &= T(m) + T(n-m) + 1 \\ &\geq 1 + cn - cm + 1 \\ &\geq cn \end{aligned}$$

L'ultima disequazione è banalmente vera per ogni $c \leq 2/m$.

Caso base: consideriamo tutti i casi per cui $1 \leq i \leq m-1$:

$$T(i) = 1 \geq c \cdot i \Leftrightarrow c \leq \frac{1}{i}$$

Il valore $\frac{1}{m}$ è il più piccolo fra tutte le disequazioni trovate; per soddisfarle tutte, compresa quella derivante dal passo induttivo, basterà porre $c = 1/m$.

Maggioranza (03/07/20)

Se esiste un valore di maggioranza, deve essere il valore contenuto nella posizione mediana $\lfloor (n+1)/2 \rfloor$.

- Se n è dispari, questo è l'elemento centrale. Qualunque maggioranza deve includere questo elemento e altri $(n-1)/2$ elementi; infatti, a sinistra e destra di tale elemento ci stanno esattamente $(n-1)/2$ elementi.
- Se n è pari, sia $n/2$ che $n/2+1$ devono appartenere alla maggioranza, per lo stesso ragionamento di cui sopra.

Dato questo valore, si effettua ricerca dicotomica che assuma l'esistenza di tale valore e restituisca l'indice più basso in cui tale valore si presenta.

Conoscendo l'indice del primo elemento, è necessario vedere se l'elemento in posizione $A[\mathit{first} + \lfloor n/2 \rfloor]$ è uguale ad esso, nel qual caso esiste una maggioranza di elementi uguali, essendo il vettore ordinato.

Maggioranza (03/07/20)

```
int hasMajority(int[] A, int n)
int candidate = A[⌊(n + 1)/2⌋])
int first = searchFirst(A, 1, n, candidate)
return A[first] == A[first + ⌊n/2⌋]
```

```
int searchFirst(int[] A, int i, int j, int v)
if i == j then
    return i
else
    int m = ⌊(i + j)/2⌋
    if v ≤ A[m] then
        return searchFirst(A, i, m, v)
    else
        return searchFirst(A, m + 1, j, v)
```

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⟩)
```

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, NODE v)

```
GRAPH G^T = transpose(G)
boolean[] id = new int[1...G.n] = {0}
ccdfs(G^T, 1, v, id)
foreach u ∈ G^T.V() do
    if id[u] == 0 then
        return false
return true
```

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

È 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)

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

Il costo è $O(m+n)$.