Grafi – Stessa distanza

- In un grafo orientato G, dati due nodi s e v, si dice che:
  - v è raggiungibile da s se esiste un cammino da s a v;
  - la distanza di v da s è la lunghezza del più breve cammino da s a v (misurato in numero di archi), oppure $+\infty$ se v non è raggiungibile da s
- Scrivere un algoritmo che prenda in input un grafo orientato $G = (V, E)$ e due nodi $s_1, s_2 \in V$, che restituisca il numero di nodi in V tali che:
  - siano raggiungibili sia da s_1 che da s_2, e
  - si trovino alla stessa distanza da s_1 e da s_2.
- Discutere la complessità dell'algoritmo proposto.

Grafi – Grafi bipartiti

- Un grafo non orientato G è bipartito se l'insieme dei nodi può essere partizionato in due sottoinsiemi disgiunti tali che nessun arco del grafo connette due nodi appartenenti allo stesso sottoinsieme.
- $G = (V, E)$ è 2-colorabile se è possibile trovare una 2-colorazione di esso, ovvero un assegnamento $c[u] \in C$ per ogni nodo $u \in V$, dove C è un insieme di "colori" di dimensione 2, tale che:

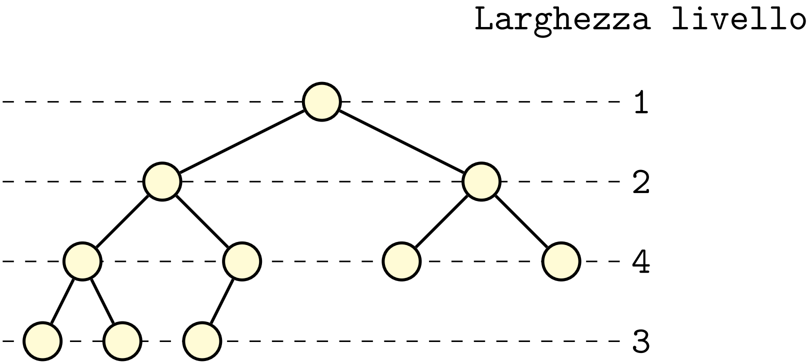
  $$(u, v) \in E \Rightarrow c(u) \neq c(v)$$

- Si dimostri che G è bipartito:
  - se e solo se è 2-colorabile
  - se e solo se non contiene cicli di lunghezza dispari
- Scrivere un algoritmo che prenda in input un grafo bipartito G e restituisca una 2-colorazione di G sull'insieme di colori $C = \{0, 1\}$, espressa come un vettore $c[1 \ldots n]$. Discuterne la complessità.

Alberi – Larghezza albero

La larghezza di un albero T è il numero massimo di nodi di T che stanno tutti al medesimo livello.

Scrivere un algoritmo che restituisca la larghezza di un albero ordinato T contenente n nodi.



SortinoSort

Il professor Sortino ha inventato un nuovo algoritmo di ordinamento.

- Il vettore di input viene diviso in tre parti, di dimensioni circa $n/3$.
- Vengono ordinati ricorsivamente i primi due terzi, i secondi due terzi, e infine di nuovo i primi due terzi.

```
SortinoSort(int[] A, int i, int j)
if j − i + 1 ≤ 6 then
    InsertionSort(A, i, j)
else
    int s = ⌈(j − i + 1)/3⌉
    SortinoSort(A, i, i + 2s − 1)
    SortinoSort(A, i + s, j)
    SortinoSort(A, i, i + 2s − 1)
```

SortinoSort

1. Qual è la complessità di questo algoritmo? Il Prof. Sortino finirà nella prossima edizione del mio libro?
2. (Difficile, Opzionale) Dimostrare per induzione che questo algoritmo è corretto. Per comodità, assumete pure che tutti i valori siano distinti.

Ricorrenza

Trovare i limiti superiore e inferiori più stretti possibili per la seguente equazione di ricorrenza:

$$T(n) = \begin{cases} 2T(\lfloor n/2 \rfloor) + 4T(\lfloor n/4 \rfloor) + 15T(\lfloor n/8 \rfloor) + n^2 & n > 8 \\ 1 & n \leq 8 \end{cases}$$

Spoiler alert!

Stessa distanza

int sameDistance(GRAPH G, NODE s_1, NODE s_2)

```
int dist1 = new int[1 ... G.n]
int dist2 = new int[1 ... G.n]
distance(G, s1, dist1)
distance(G, s2, dist2)
int counter = 0
foreach u ∈ G.V() do
    if dist1[u] ≠ −1 and dist1[u] == dist2[u] then
        counter = counter + 1
return counter
```

Grafi – Grafi bipartiti

- Se G è bipartito, è 2-colorabile. Diamo colore 0 a tutti i nodi in una partizione, diamo colore 1 a tutti i nodi nell'altra. Non essendoci archi fra i nodi di una partizione, la colorazione è valida.
- Se G è 2-colorabile, non contiene cicli di lunghezza dispari. Supponiamo per assurdo che esista un ciclo $(v_1, v_2), (v_2, v_3) \ldots, (v_{k-1}, v_k), (v_k, v_1)$, con k dispari. Se il nodo v_1 ha colore 0, il nodo v_2 deve avere colore 1; il nodo v_3 deve avere colore 0, e così via fino al nodo v_k, che deve avere colore 0. Poichè v_1 è successore di v_k, v_1 deve avere colore 1, assurdo.

Grafi – Grafi bipartiti

- Se non esistono cicli di lunghezza dispari, il grafo è bipartito. Dimostriamo questa affermazione costruttivamente. Si prenda un nodo x lo si assegna alla partizione S_1. Si prendono poi tutti i nodi adiacenti a nodi in S_1 e li si assegna alla partizione S_2. Si prendono tutti i nodi adiacenti a nodi in S_2 e li si assegna alla partizione S_1. Questo processo termina quando tutti i nodi appartengono ad una o all'altra partizione. Un nodo può essere assegnato più di una volta se e solo se fa parte di un ciclo. Ma affinché venga assegnato a due colori diversi, deve far parte di un ciclo di lunghezza dispari, e questo non è possibile.

Grafi – Grafi bipartiti

Questa funzione ritorna un vettore di colori se il grafo è 2-colorabile, **nil** altrimenti.

```
int[] color(GRAPH G)
----------------------------------------
int[] colors = new int[1 ... G.n]
for u ∈ G.V() do
    colors[u] = −1
foreach u ∈ G.V() do
    if colors[u] < 0 then
        if not colorRec(G, u, colors, 0) then
            return nil
return colors
```

Grafi – Grafi bipartiti

Questa funzione ritorna **true** se il grafo è 2-colorabile, **false** altrimenti.

boolean $\mathsf{colorRec}(\textsc{Graph}\ G, \textsc{Node}\ u, \textbf{int}[\,]\ \mathit{colors}, \textbf{int}\ \mathit{color})$

$\mathit{colors}[u] = \mathit{color}$
foreach $v \in G.\mathsf{adj}(u)$ **do**
    if $\mathit{colors}[v] < 0$ **then**
        if $\mathsf{colorRec}(G, v, \mathit{colors}, 1 - \mathit{color}) == \textbf{false}$ **then**
            return false
    else if $\mathit{colors}[v] == \mathit{color}$ **then**
        return false
return true

Larghezza (1)

```
int breadth(TREE t)

int breadth = 0
int level = 1
int count = 1
QUEUE Q = Queue()
Q.enqueue(t)
while not Q.isEmpty() do
    TREE u = Q.dequeue()
    if u.level ≠ level then
        level = u.level
        count = 0
    count = count + 1
    breadth = max(breadth, count)
    TREE v = u.leftmostChild()
    while v ≠ nil do
        v.level = u.level + 1
        Q.enqueue(v)
        v = v.rightSibling()
return breadth
```

Larghezza (2)

```
int breadth(TREE t)
----------------------------------------------------------------
int count = 1          % # nodi nel livello corrente da visitare; radice
int breadth = 1        % Massima larghezza trovata finora; radice
QUEUE Q = Queue()
Q.enqueue(t)
while not Q.isEmpty() do
    TREE u = Q.dequeue()
    TREE v = u.leftmostChild()
    while v ≠ nil do
        Q.enqueue(v)
        v = v.rightSibling()
    count = count − 1
    if count == 0 then                                  % Nuovo livello
        count = Q.size()
        breadth = max(breadth, count)
return breadth
```

Ricorrenza

E' facile vedere che la ricorrenza è $\Omega(n^2)$, per via della sua componente non ricorsiva. Proviamo quindi a dimostrare che $T(n) = O(n^2)$.

- Caso base: $T(n) = 1 \leq cn^2$, per tutti i valori di n compresi fra 1 e 8. Tutte queste disequazioni sono soddisfatte da $c \geq 1$.
- Ipotesi induttiva: $T(k) \leq ck^2$, per $k < n$
- Passo induttivo:

$$\begin{aligned} T(n) &= 2T(\lfloor n/2 \rfloor) + 4T(\lfloor n/4 \rfloor) + 15T(\lfloor n/8 \rfloor) + n^2 \\ &\leq 2c\lfloor n/2 \rfloor^2 + 4c\lfloor n/4 \rfloor^2 + 15\lfloor n/8 \rfloor^2 + n^2 \\ &\leq 2cn^2/4 + 4cn^2/16 + 15cn^2/64 + n^2 \\ &\leq 63/64cn^2 + n^2 \leq cn^2 \end{aligned}$$

  L'ultima disequazione è rispettata per $c \geq 64$.

Abbiamo quindi dimostrato che $T(n) = \Theta(n^2)$.

SortinoSort – 12/09/10

La complessità è rappresentata dalla seguente equazione di ricorrenza:

$$T(n) = \begin{cases} 3T(\frac{2}{3}n) + 1 & n > 6 \\ 1 & n \leq 6 \end{cases}$$

Utilizzando il teorema delle ricorrenze lineari con partizione bilanciata, si ottiene che $a = 3$, $b = 3/2$, da cui $\alpha = \log_{3/2} 3$; inoltre, $\beta = 0$. Siamo quindi nel caso $T(n) = n^\alpha$.

Non avete una calcolatrice e non sapete quanto sia $\log_{3/2} 3$?

SortinoSort – 12/09/10

La complessità è rappresentata dalla seguente equazione di ricorrenza:

$$T(n) = \begin{cases} 3T(\frac{2}{3}n) + 1 & n > 6 \\ 1 & n \leq 6 \end{cases}$$

Utilizzando il teorema delle ricorrenze lineari con partizione bilanciata, si ottiene che $a = 3$, $b = 3/2$, da cui $\alpha = \log_{3/2} 3$; inoltre, $\beta = 0$. Siamo quindi nel caso $T(n) = n^\alpha$.
Non avete una calcolatrice e non sapete quanto sia $\log_{3/2} 3$?

E' semplice: $\left(\frac{3}{2}\right)^2 = \frac{9}{4} < 3$, mentre $\left(\frac{3}{2}\right)^3 = \frac{27}{8} > 3$. Quindi α è compreso fra 2 e 3, e quindi questo algoritmo è addirittura peggiore di Insertion Sort. Il prof. Sortino non finirà nel mio libro.