

Alberi – Indovina l'albero

Gli ordini di visita di un albero binario di 9 nodi sono i seguenti:

- A, E, B, F, G, C, D, I, H (anticipato)
- B, G, C, F, E, H, I, D, A (posticipato)
- B, E, G, F, C, A, D, H, I (simmetrico).

Si ricostruisca l'albero binario e si descriva la strategia utilizzata.

Per chi ha finito in fretta: scrivere un algoritmo che implementa tale strategia.

Alberi – Albero livello–valore

Scrivere un algoritmo che preso in input un albero binario T i cui nodi sono associati ad un **valore intero** $T.value$, restituisca il numero di nodi dell'albero il cui valore è **uguale al livello del nodo**.

Vi ricordo che il **livello del nodo** è pari al numero di archi che devono essere attraversati per raggiungere la radice dal nodo. Per cui la radice ha livello 0, i suoi figli hanno livello 1, etc.

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 pari a:
 - **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 $c[1 \dots n]$. Discuterne la complessità.

Grafi – Tutte le strade portano a Roma

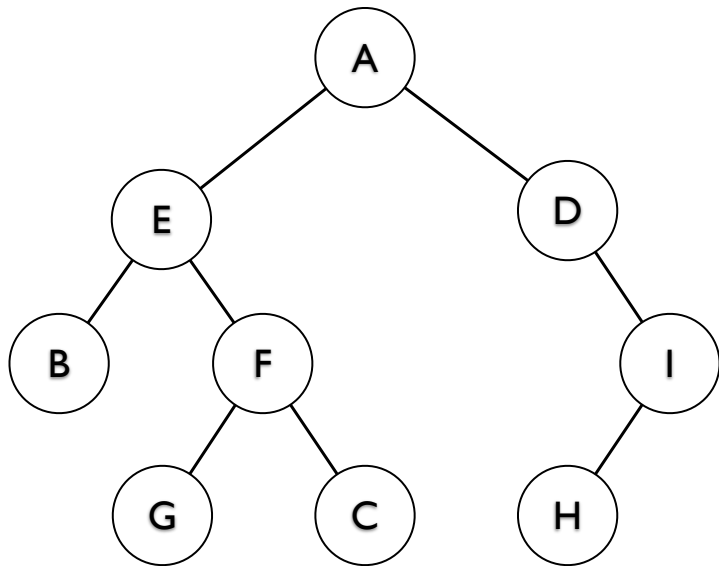
Un vertice v in un grafo orientato G si dice di tipo “Roma” se ogni altro vertice w in G può raggiungere v con un cammino orientato che parte da w e arriva a v .

- 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!

Indovina l'albero



Albero livello-valore

Una semplice visita posticipata risolve il problema. La complessità è ovviamente $O(n)$.

```
int sameLevel(TREE T)
return sameLevelRec(T, 0)

int sameLevelRec(TREE T, int  $\ell$ )
int tot = 0
if  $T \neq \text{nil}$  then
    tot = sameLevelRec(T.right(),  $\ell + 1$ ) + sameLevelRec(T.left(),  $\ell + 1$ )
    if  $T.value == \ell$  then
        tot = tot + 1
return tot
```


Stessa distanza

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

```
int  $dist_1$  = new int[1 ...  $G.n$ ]  
int  $dist_2$  = new int[1 ...  $G.n$ ]  
distance( $G$ ,  $s_1$ ,  $dist_1$ )  
distance( $G$ ,  $s_2$ ,  $dist_2$ )  
int  $counter$  = 0  
foreach  $u \in G.V()$  do  
    if  $dist_1[u] \neq -1$  and  $dist_1[u] == dist_2[u]$  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 $(v_1, v_2), (v_2, v_3) \dots, (v_{k-1}, v_k), (v_k, v_1)$, 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 prende un nodo x e 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.
- Tuttavia, affinché venga assegnato a due partizioni diverse, dovrebbe 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 colorRec(GRAPH G, NODE u, int[] colors, int color)
```

```
  colors[u] = color
```

```
foreach v ∈ G.adj(u) do
```

```
  if colors[v] < 0 then
```

```
    if colorRec(G, v, colors, 1 − color) == false then
```

```
      return false
```

```
  else if colors[v] == color then
```

```
    return false
```

```
return true
```

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 $G^T = \text{transpose}(G)$

boolean[] $id = \text{new int}[1 \dots G.n] = \{0\}$

$\text{ccdfs}(G^T, 1, v, id)$

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

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( $G, v$ )
```

Il costo è $O(m+n)$.