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 illustri *brevemente* il ragionamento.

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.

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 stampa 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
```

Il gioco delle coppie

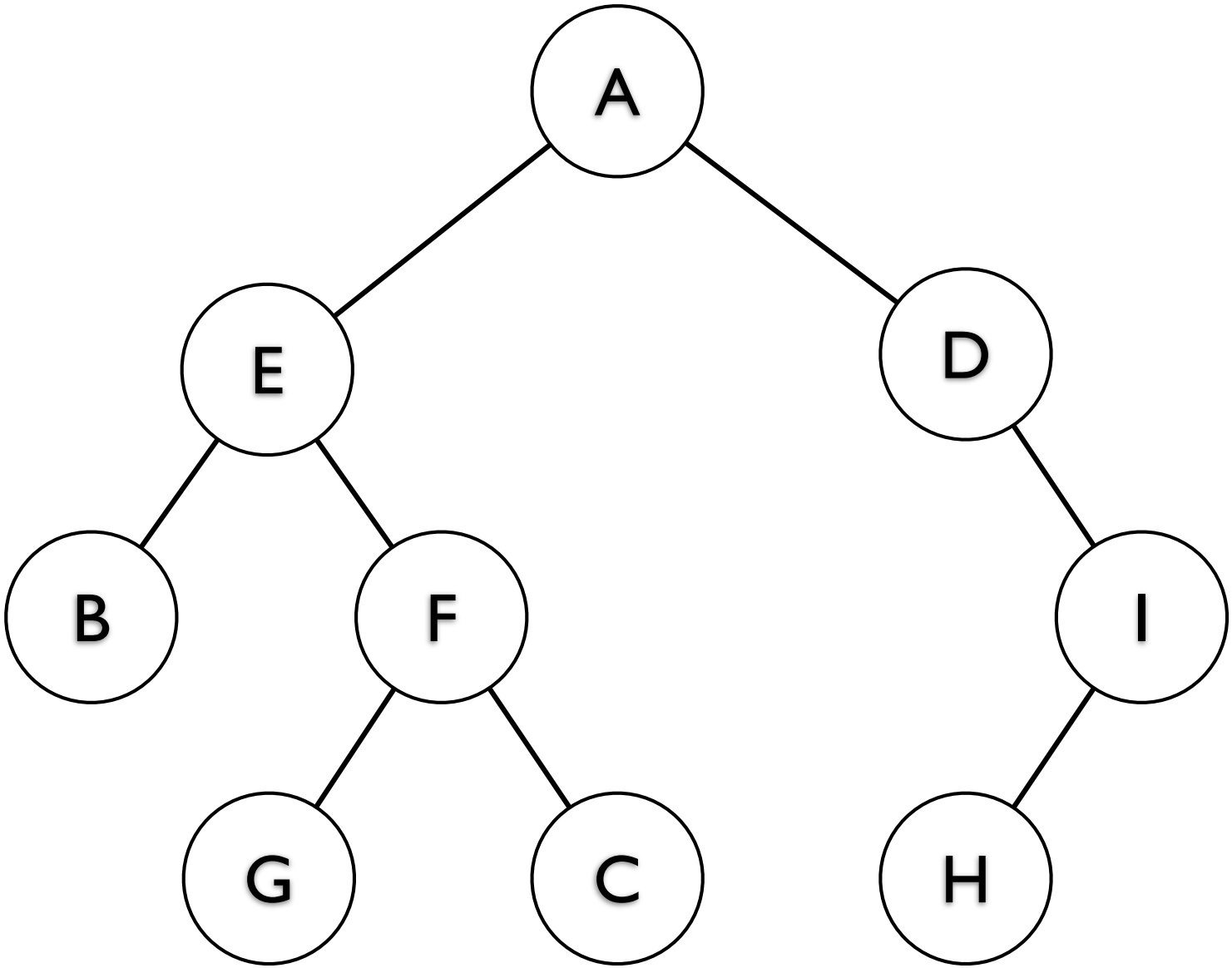
Scrivere un algoritmo che, dato un vettore A di n interi distinti (n pari), ritorna **true** se è possibile partizionare A in coppie di elementi che hanno tutte la stessa somma (intesa come la somma degli elementi della coppia), **false** altrimenti. Ad esempio:

$$7, 4, 5, 2, 3, 6$$

può essere partizionato in $7 + 2 = 4 + 5 = 3 + 6$.
Discutere la complessità e la correttezza – per questo esercizio, la dimostrazione di correttezza è importante e va scritta bene.

Spoiler alert!

Indovina l'albero



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

isRoma(GRAPH G, NODE v)

integer $status \leftarrow$ **new integer**$[1 \ldots G.N]$
foreach $v \in G.\mathsf{V}() - \{r\}$ **do**
    $status[v] \leftarrow$ UNVISITED
$status[r] \leftarrow$ REACH-ROMA
foreach $v \in G.\mathsf{V}() - \{r\}$ **do**
    if $status[v] =$ UNVISITED **then**
        visitRoma$(G, v, status)$
    if $status[v] =$ VISITED **then**
        return false
return true

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

visitRoma(GRAPH G, NODE v, **integer**[ ]$status$)

$status[v] \leftarrow$ VISITED
foreach $w \in G.\mathsf{adj}(v)$ **do**
    if $status[w] =$ UNVISITED **then**
        visitRoma$(G, w, status)$
    if $status[w] =$ REACH-ROMA **then**
        $status[v] \leftarrow$ REACH-ROMA

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.

---

isRoma(Graph G^T, Node v)

---

boolean$[\,]id \leftarrow$ **new integer**$[1 \ldots G^T.n]$
foreach $u \in G^T.\mathsf{V}()$ **do**
  $id[u] \leftarrow 0$
ccdfs$(G^T, 1, v, id)$
foreach $u \in G^T.\mathsf{V}()$ **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: solo il primo nodo può essere di tipo Roma, quindi basta richiamare Roma() (nella versione con grafo trasposto) a partire da esso.

Roma(GRAPH G^T)

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

Il costo è $O(m+n)$.

Anagrammi – 2011/09/07

```
anagrams(Item [][] S, integer 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 x ∈ H do
    Set S ← H.lookup(x)
    print S
```

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

Il gioco delle coppie – 2012/05/03

checkPairs(**integer**[ ] A, **integer** n)

```
sort(A, n)
integer pairSum ← A[1] + A[n]
for i ← 2 to n/2 do
    if A[i] + A[n − i + 1] ≠ pairSum then
        return false
return true
```

Dimostrazione: supponiamo per assurdo che esista un insieme di coppie che rispetti le condizioni per restituire **true**, in cui l'elemento maggiore M sia associato ad un elemento M' diverso dal minore m ($m < M'$). Quindi il minore m è associato ad un elemento m' diverso dal massimo M ($m' < M$). Allora $m + m' < M + M'$, il che contraddice l'ipotesi che tale insieme di coppie rispetti le condizioni per restituire **true**.

Il gioco delle coppie - $O(n)$, hash set

checkPairs(**integer**[ ] A, **integer** n)

```
integer tot ← sum(A, n)                    % Sum all elements, O(n)
real pairSum ← tot/(n/2)
if pairSum ≠ ⌊pairSum⌋ then
    return false
SET set ← Set()                            % Based on a hash table
for i ← 1 to n do
    set.insert(A[i])
for i ← 1 to n do
    if not set.contains(pairSum − A[i]) then
        return false
return true
```