Alberi – Sopra e sotto

Scrivere un algoritmo che prende in input un albero binario T e restituisca in output il numero di nodi di tale albero il cui numero di ascendenti (il padre, il nonno (padre del padre), il bisnonno (padre del nonno), etc) è uguale al numero di discendenti (i figli, i nipoti (figli dei figli), i bisnipoti (figli dei nipoti), etc).
Discutere correttezza e complessità dell'algoritmo proposto.

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
```

Esercizi di programmazione

Cerca la coppia - Versione 1

Dato un vettore $A[1 \ldots n]$ di interi e un intero v, scrivere un algoritmo che determini se esistono due elementi in A la cui somma sia esattamente v.

Cerca la coppia - Versione 2

Dato un vettore $A[1 \ldots n]$ di interi, scrivere un algoritmo che determini se esistono due elementi in A la cui somma sia esattamente 17.

Cerca la coppia - Versione 3

Dato un vettore $A[1 \ldots n]$ di interi positivi, scrivere un algoritmo che determini se esistono due elementi in A la cui somma sia esattamente 17.

Il gioco delle coppie (2-Partition)

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.

Confrontatelo con il problema 3-partition discusso a lezione.

Vito's Family

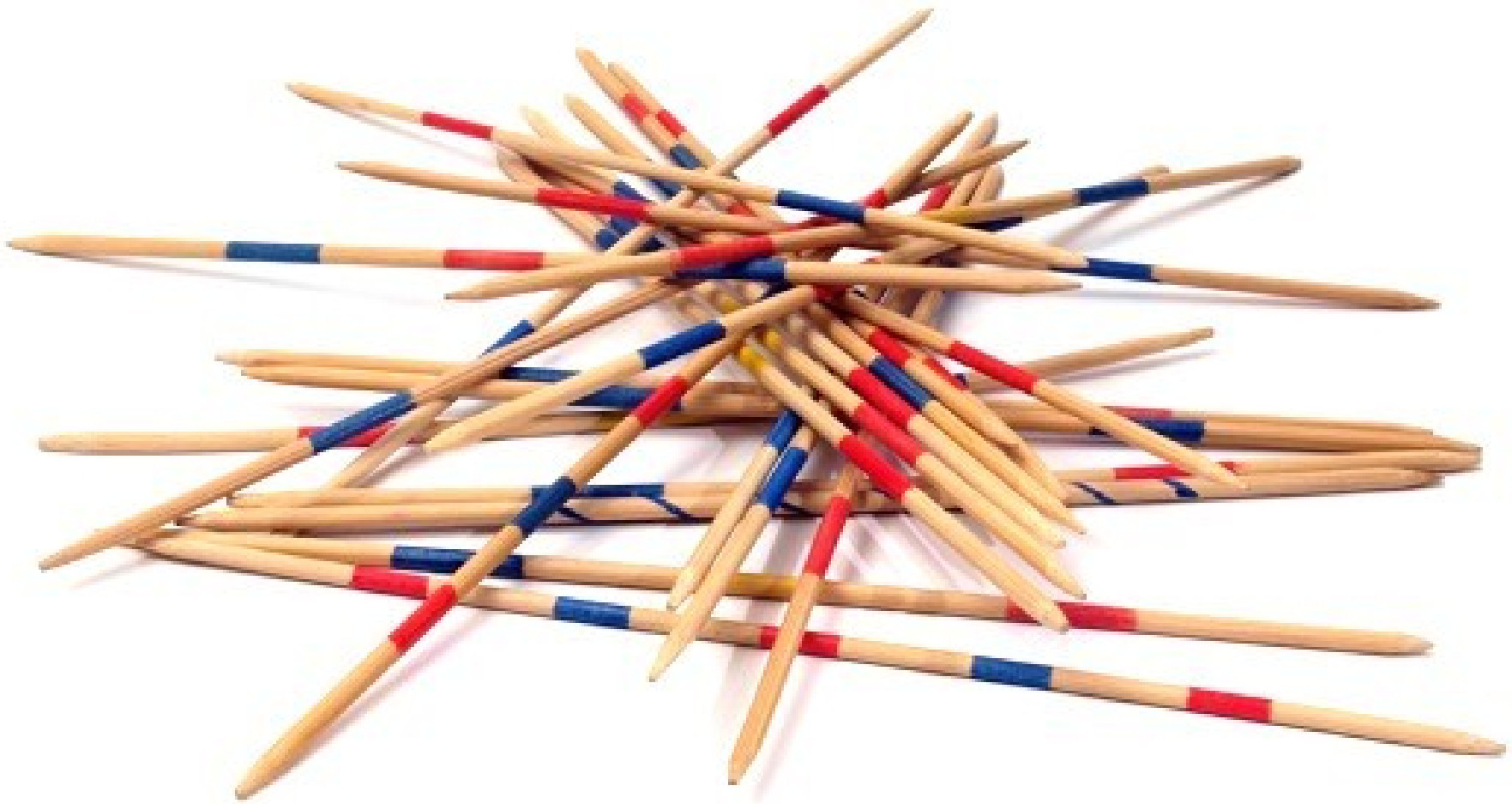
The famous gangster Vito Deadstone is moving to New York. He has a very big family there, all of them living on Lamafia Avenue. Since he will visit all his relatives very often, he wants to find a house close to them. Indeed, Vito wants to minimize the total distance to all of his relatives and has blackmailed you to write a program that solves his problem.

Input For each test case you will be given the integer number of relatives r ($0 < r < 500$) and the street numbers (also integers) $s_1, s_2, ..., s_i, ..., s_r$ where they live ($0 < s_i < 30000$). Note that several relatives might live at the same street number.

Output For each test case, your program must write the minimal sum of distances from the optimal Vito's house to each one of his relatives. The distance between two street numbers s_i and s_j is $d_{i,j} = |s_i - s_j|$.

Avanzati dalle volte precedenti

Shangai

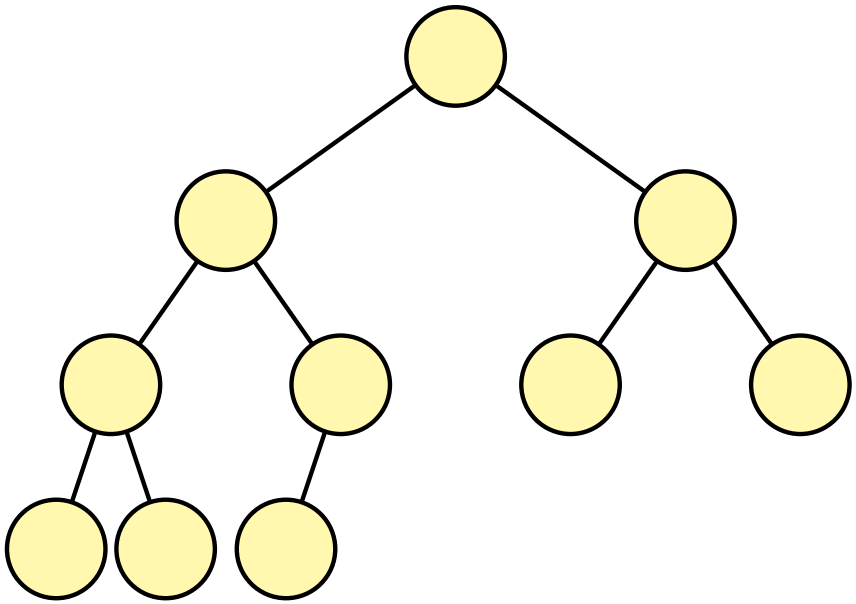


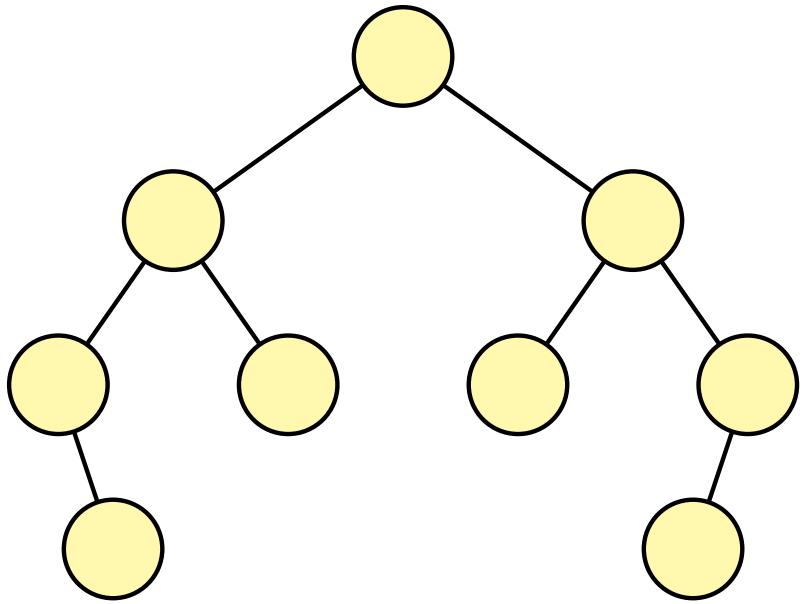
Shangai

- 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.

Alberi simmetrici

Scrivere un algoritmo **boolean** isSymmetric(Tree T) che prenda in input un albero binario T non vuoto e restituisca **true** se T è simmetrico, **false** altrimenti. Un albero binario è simmetrico se il sottoalbero sinistro della radice è un'immagine *speculare* del sottoalbero destro della radice. L'albero binario a sinistra non è simmetrico, mentre lo è quello di destra.





Discutere correttezza e complessità computazionale dell'algoritmo

Difficili

Doppio mediano

Siano $X[1 \ldots n]$ e $Y[1 \ldots n]$ due vettori, ciascuno contenente n interi già ordinati. Scrivere un algoritmo che trovi i valori mediani dei $2n$ elementi dei vettori X e Y presi insieme. Usiamo il plurale perchè essendo $2n$ pari, è possibile definire *due* valori mediani. Discutere correttezza e complessità.

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!

Sopra e sotto

(**int**, **int**) countTreeRec(TREE t, **int** $ancestors$)

---

if $t == \textbf{nil}$ **then**
  return (0,0)
else
  $pred_L, count_L = \text{countTreeRec}(t.left, ancestors + 1)$
  $pred_R, count_R = \text{countTreeRec}(t.right, ancestors + 1)$
  return $(pred_L + pred_R + 1, count_L + count_R + \text{iif}(ancestors = pred_L + pred_R, 1, 0))$

L'algoritmo viene invocato dalla seguente funzione wrapper:

---

countTree(TREE t)

$pred, count = \text{countTreeRec}(t, 0)$
return $count$

Anagrammi – 2011/09/07

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

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

Esercizi di programmazione

`Cerca la coppia - Versione 1`

Dato un vettore $A[1 \ldots n]$ di interi e un intero v, scrivere un algoritmo che determini se esistono due elementi in A la cui somma sia esattamente v.

`Soluzione -` $O(n \log n)$

Si ordina il vettore. Per ogni elemento $A[i]$, si cerca il valore $v - A[i]$ tramite ricerca dicotomica.

Codice lasciato per esercizio. Si consiglia di provare a implementarlo nel proprio linguaggio preferito.

Esercizi di programmazione

`Cerca la coppia - Versione 2`

Dato un vettore $A[1 \ldots n]$ di interi, scrivere un algoritmo che determini se esistono due elementi in A la cui somma sia 17.

`Soluzione -` $O(n \log n)$

Si richiama l'algoritmo precedente con $v = 17$.

Codice lasciato per esercizio. Si consiglia di provare a implementarlo nel proprio linguaggio preferito.

Esercizi di programmazione

`Cerca la coppia - Versione 3`

Dato un vettore $A[1 \ldots n]$ di interi positivi, scrivere un algoritmo che determini se esistono due elementi in A la cui somma sia 17.

Soluzione - $O(n)$

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

boolean[ ] $present$ = **new boolean**$[1 \dots n]$ = { **false** }
for $i = 1$ **to** n **do**
    if $1 \leq A[i] \leq 16$ **then**
        $present[A[i]]$ = **true**
int $i = 1$
int $j = 16$
while $i \leq 8$ **and not** ($A[i]$ **and** $A[j]$) **do**
    $i = i + 1$
    $j = j - 1$
return $i \leq 8$

Il gioco delle coppie – 2012/05/03 ($O(n \log n)$)

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

```
sort(A, n)
int 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

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

int $tot = \mathsf{sum}(A, n)$ % Sum all elements, $O(n)$
float $pairSum = tot/(n/2)$
if $pairSum \neq \lfloor pairSum \rfloor$ **then**
    return false
SET $set = \mathsf{Set}()$ % 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

Vito

Il numero civico ottimale è la mediana.

Si consideri il caso di n dispari e quindi di una singola mediana m; m ha una quantità $(n-1)/2$ di numeri civici sia alla destra che alla sinistra. Il caso con n pari e quindi due valori mediani è simile.

Si consideri ogni altra soluzione m' diversa dalla mediana e assumiamo che $m < m'$ (il caso $m > m'$ è simmetrico).

Sia $d = m' - m$ la differenza fra questi due numeri civici. Nella soluzione in cui abbiamo scelto m', tutti i civici che si trovano a sinistra di m' costano d unità in più rispetto alla soluzione in cui abbiamo scelto m. Tutti i civici a destra di m' (m' incluso) costano d unità in meno rispetto alla soluzione in cui abbiamo scelto m.

Poiché i numeri civici a sinistra di m' sono di più dei numeri civici a destra di m' (m' incluso), la soluzione m' costa più della soluzione m.

Complessità: $O(n)$ per il calcolo della mediana

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** hasCycleRec$(G, u, \&clock, dt, ft)$ **then**
        return true
return false

Alberi simmetrici (24/07/20)

Un sottalbero è simmetrico se i suoi sottoalberi destro e sinistro sono speculari.

Due sottoalberi t_1, t_2 sono speculari se e solo se:

- il sottoalbero sinistro di t_1 è speculare al sottoalbero destro di t_2
- il sottoalbero destro di t_1 è speculare al sottoalbero sinistro di t_1.

Il caso base è dato due nodi **nil** (si ritorna **true**) o da uno nodo **nil** e un nodo non **nil** (si ritorna **false**).

La procedura effettua una visita su entrambi gli alberi, e quindi ha complessità $\Theta(n)$.

Alberi simmetrici (24/07/20)

boolean isSymmetric(TREE T)

return isMirror($T.left, T.right$)

boolean isMirror(TREE t_{L}, TREE t_{R})

```
if t_L == nil and t_R == nil then
    return true
else if t_L ≠ nil and t_R ≠ nil then
    return isMirror(t_L.right, t_R.left) and isMirror(t_L.left, t_R.right)
else
    return false
```

Doppio mediano

---

mediana(**int**[ ] X, **int**[ ] Y, **int** b_x, e_x, b_y, e_y)

---

if $e_x - b_x = 1$ **then** **return** mediana4$(X, Y, b_x, e_x, b_y, e_y)$

int $m_x = \lfloor (b_x + e_x)/2 \rfloor$
int $m_y = \lceil (b_y + e_y)/2 \rceil$
if $X[m_x] < Y[m_y]$ **then** **return** mediana$(X, Y, m_x, e_x, b_y, m_y)$

if $Y[m_y] < X[m_x]$ **then** **return** mediana$(X, Y, b_x, m_x, m_y, e_y)$

return $(X[m_x], Y[m_y])$

Pozzo universale – Esercizio 1.9 degli esercizi su grafi

Dati due vertici i, j:

- se $A[i][j] = 1$, allora i non è un pozzo universale (c'è un arco uscente da i);
- se $A[i][j] = 0$, allora j non è un pozzo universale (manca arco entrante in j).

Si noti inoltre che può esistere un solo pozzo universale.
Partiamo dalla prima riga: $i = 1$.

1. Cerchiamo il minore indice j tale $j > i$ e $A[i][j] = 1$.
2. Se tale vertice non esiste, i non ha archi uscenti ed è l'unico candidato per essere un pozzo universale
3. Se invece tale j esiste, tutti i vertici h tali che $1 \leq h < j$ non possono essere pozzi universali, perché manca un arco da i. Quindi ci spostiamo nella riga $i = j$, e torniamo al passo 1.

Si noti che un possibile candidato viene trovato sempre; al limite è dato dall'ultima riga. A quel punto, si verifica che sia effettivamente un pozzo universale.
Il costo dell'algoritmo è pari a $\Theta(n)$.

Pozzo universale – Esercizio 1.9 degli esercizi su grafi

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

```
int i = 1
int candidate = −1
while i < n and candidate < 0 do
    j = i + 1
    while j ≤ n and A[i][j] == 0 do
        j = j + 1
    if j > n then
        candidate = i
    else
        i = j
rowtot = Σ_{j∈{1...n}} A[candidate][j]
coltot = Σ_{j∈{1...n}} A[j][candidate]
return rowtot == 0 and coltot == n − 1
```