Esercizi Divide-et-impera

Chi manca?

Sia dato un vettore ordinato $A[1 \ldots n]$ contenente n elementi interi distinti appartenenti all'intervallo $1 \ldots n+1$. Si scriva un algoritmo, basato sulla ricerca binaria, per individuare in tempo $O(\log n)$ l'unico intero dell'intervallo $1 \ldots n+1$ che non compare in A.

Punto fisso

Progettare un algoritmo che, preso un vettore ordinato A di n interi distinti, determini se esiste un indice i tale che $A[i] = i$ in tempo $O(\log n)$.

Vettori uni-modulari

Un vettore di interi A è detto unimodulare se ha tutti valori distinti ed esiste un indice h tale che $A[1] > A[2] > \ldots > A[h-1] > A[h]$ e $A[h] < A[h+1] < A[h+2] < \ldots < A[n]$, dove n è la dimensione del vettore. Progettare un algoritmo $O(\log n)$ che dato un vettore unimodulare restituisce il valore minimo del vettore.

Per fare un albero (binario di ricerca) ci vuole...

Dato un vettore V di n interi ordinati e distinti, scrivere una procedura che costruisce un albero binario di ricerca di altezza minima.
Discuterne la correttezza e la complessità.

Samarcanda

Nel gioco di Samarcanda, ogni giocatore è figlio di una nobile famiglia della Serenissima, il cui compito è di partire da Venezia con una certa dotazione di denari, arrivare nelle ricche città orientali, acquistare le merci preziose al prezzo più conveniente e tornare alla propria città per rivenderle.

Dato un vettore P di n interi in cui $P[i]$ è il prezzo di una certa merce al giorno i, trovare la coppia di giornate (x, y) con $x < y$ per cui risulta massimo il valore $P[y] - P[x]$. Calcolare la complessità e dimostrare la correttezza.

Vicini-vicini – difficile

Siano dati due vettori $x[1 \dots n]$ e $y[1 \dots n]$ che rappresentano le coordinate cartesiane di n punti nel piano; ovvero il punto i-esimo è in posizione $(x[i], y[i])$. Scrivere un algoritmo che restituisca, fra tutte le coppie, quella con minima distanza.

Note:

- Esiste ovviamente una soluzione banale $O(n^2)$; si può fare di meglio?
- Questo problema ha applicazioni nei software GIS

Spoiler alert!

Esercizi Divide-et-Impera

Una spiegazione più approfondita del funzionamento di missing(), fixedPoint(), vum() si trova nel file `12 - Divide-et-impera` che si trova a questo indirizzo:

`http://cricca.disi.unitn.it/montresor/teaching/asd/materiale/esercizi/esercizi-argomento/`

Chi manca?

```
int missing(int[] A, int i, int j)
if i == j then
    if A[i] == i then
        % Needed if the missing number is n + 1
        return i + 1
    else
        return i
int m = ⌊(i + j)/2⌋
if A[m] == m then
    return missing(A, m + 1, j)
else
    return missing(A, i, m)
```

Punto fisso

```
boolean fixedPoint(int[] A, int i, int j)
if i > j then
    return false
int m = ⌊(i + j)/2⌋
if A[m] == m then
    return true
else if A[m] < m then
    return fixedPoint(A, m + 1, j)
else
    return fixedPoint(A, i, m − 1)
```

Vettori unimodulari

```
int vum(int[] A, int i, int j)
if i == j then
    return A[i]
if j == i + 1 then
    return min(A[i], A[j])
int m = ⌊(i + j)/2⌋
if A[m − 1] > A[m] and A[m + 1] > A[m] then
    return A[m]
if A[m − 1] > A[m] then
    return vum(A, m + 1, j)
else
    return vum(A, i, m − 1)
```

Per fare un albero (binario di ricerca) ci vuole...

```
TREE build-tree-rec(int[] A, int i, int j)
if i ≤ j then
    int m = (i + j)/2
    TREE T = new TREE
    T.left = build-tree-rec(A, i, m − 1)
    T.right = build-tree-rec(A, m + 1, j)
    T.key = V[m]
    return T
else
    return nil
```

L'equazione di ricorrenza è pari a: $T(n) = 2T(n/2) + 1$, che dà origine a $T(n) = \Theta(n)$.

Samarcanda

Struttura della soluzione:

- Si divide il vettore a metà
- Si applica ricorsivamente la soluzione nelle due metà, ottenendo la migliore soluzione nella metà destra e nella metà sinistra
- Da questo approccio, non vengono considerate le soluzioni in cui si acquista nella metà sinistra del vettore, si vende nella metà destra
- Si cerca quindi il minimo a sinistra e il massimo a destra e si applica ricorsivamente la soluzione
- Caso base: un elemento solo, che ovviamente da origine a zero come massimo guadagno

Samarcanda

```
samarcanda(int[] A, int i, int j)
if i ≥ j then
    return 0
m = ⌊(i + j)/2⌋
int maxLeft = samarcanda(L, i, m)
int maxRight = samarcanda(L, m + 1, j)
int ml = min(L, i, m)
int mr = max(L, m + 1, j)
int maxCross = max(0, mr − ml)
return max(maxLeft, maxRight, maxCross)
```

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

$$= \Theta(n \log n)$$

Samarcanda

- La soluzione così proposta non è quella ottima, perché il problema può essere risolto in tempo $\Theta(n)$
- Un possibile approccio consiste nel calcolare il vettore delle differenze fra le quotazioni, e applicare la somma massimale vista alla prima lezione. Costo: $\Theta(n)$.
- Altrimenti, è facile progettare una soluzione ad-hoc per il problema