Stringhe primitive

Scrivere un algoritmo che, dati un insieme S contenente m stringhe dette *primitive* ed una stringa $X[1 \ldots n]$, conti in quanti modi diversi X è ottenibile dalla concatenazione di stringhe primitive.

Ad esempio, dato $S = \{\texttt{01}, \texttt{10}, \texttt{011}, \texttt{101}\}$:

- $X = \texttt{0111010101}$: ci sono 3 modi
  ($\texttt{011} - \texttt{10} - \texttt{10} - \texttt{101}$, $\texttt{011} - \texttt{10} - \texttt{101} - \texttt{01}$ e $\texttt{011} - \texttt{101} - \texttt{01} - \texttt{01}$)
- $X = \texttt{0110001}$: non c'è modo di ottenere tre 0 consecutivi

Per comodità, supponete che la lunghezza di una stringa s sia $|s|$ e di avere a disposizione una primitiva $\mathsf{check}(X, s, i)$ che ritorna vero se la stringa s è contenuta nella stringa X a partire dalla posizione i. Il costo della chiamata a $\mathsf{check}()$ è $O(|s|)$. Ad esempio, se $X = \texttt{1001}$ e $s = \texttt{00}$, $\mathsf{check}(X, s, 2)$ ritorna vero, per tutti gli altri indici i ritorna falso.

Quadrato binario

Sia A una matrice $n \times n$ di valori booleani 0/1. Scrivere un algoritmo che prenda in input la matrice A e la sua dimensione n, e restituisca la dimensione del più grande quadrato composto da valori 1 contenuto nella matrice. Ad esempio, nella matrice seguente, i quadrati di dimensione massima sono grandi 4×4 (ve ne sono due, di cui uno evidenziato in rosso).

```
1 0 1 0 1 0 0
1 0 1 1 1 1 0
0 1 1 1 1 1 0
0 0 1 1 1 1 0
1 1 1 1 1 1 0
1 1 1 1 1 1 0
```

Limite inferiore per costruzione ABR

Limite inferiore

Sia V un vettore non ordinato contenente n valori interi distinti.

Si consideri il problema di costruire un albero binario di ricerca a partire dai dati in V. Dimostrare che il limite inferiore per questo problema è $\Omega(n \log n)$.

Costruzione ABR di altezza minima (Bonus)

Dato un vettore V di n interi ordinati e distinti, scrivere una procedura che costruisca un albero binario di ricerca di altezza minima. Discutere correttezza e complessità dell'algoritmo proposto

Sequenza specchiata

Dato un vettore V di n interi appartenenti all'insieme $\{0, 1\}$, si scriva un algoritmo che restituisca la lunghezza del più lungo sottovettore contiguo formato da k valori 0 seguiti da k valori 1.

Si noti che è possibile che esistano altri valori 0 prima del sottovettore così individuato, oppure altri valori 1 dopo il sottovettore, ma non è possibile che si verifichino entrambe le estensioni, altrimenti il sottovettore non sarebbe massimale. Discutere correttezza e complessità dell'algoritmo proposto.

Esempio: per l'input `00111111000111`10000, l'algoritmo deve restituire 6.

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 $d_{i,j} = |s_i - s_j|$.

Spoiler alert!

Stringhe primitive (Compito 09/07/2012)

$$DP[i] = \begin{cases} \sum_{s \in S \wedge \mathsf{check}(X,s,i)} DP[i+|s|] & 1 \leq i \leq n \\ 1 & i > n \end{cases}$$

Stringhe primitive

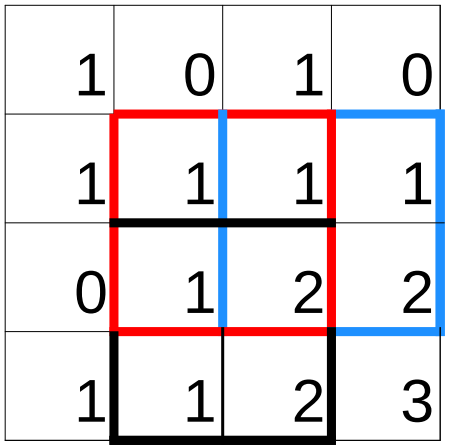
```
int count(int[] X, int n, Set S, int[] DP, int i)
if i > n then
    return 1
else
    if DP[i] < 0 then
        DP[i] = 0
        foreach s ∈ S do
            if check(X, s, i) then
                DP[i] = DP[i] + count(X, n, S, DP, i + |s|)
return DP[i]
```

La chiamata iniziale è $\mathsf{count}(X, n, S, DP, 1)$. Detto $m = \sum_{s \in S} |s|$, la complessità è $O(mn)$.

La soluzione potrebbe essere migliorata con una struttura dati chiamata Trie - si ottiene complessità $O(mn)$, con $m = \max\{|s| : s \in S\}$.

Quadrato binario (Compito 10/09/12)

1	0	1	0
1	1	1	1
0	1	1	1
1	1	1	1



Quadrato binario

$DP[i][j]$ contiene la dimensione del più grande quadrato composto da soli 1 il cui angolo in basso a destra sia nella posizione (i, j)

$$DP[i][j] = \begin{cases} 0 & A[i][j] = 0 \\ 1 & A[i][j] = 1 \wedge (i = 1 \vee j = 1) \\ \min\{DP[i-1][j], & \\ \quad DP[i-1][j-1], & \\ \quad DP[i][j-1]\} + 1 & \text{altrimenti} \end{cases}$$

Quadrato binario

```
int maxSquare(boolean[][] A, int n)
int[][] DP = new int[1...n][1...n]
for i = 1 to n do
    DP[i][1] = A[i][1]
    DP[1][i] = A[1][i]
for i = 2 to n do
    for j = 2 to n do
        if A[i][j] == 0 then
            DP[i][j] = 0
        else
            DP[i][j] = min(DP[i-1][j], DP[i-1][j-1], DP[i][j-1]) + 1
return max(DP, n)                    % Return max value in matrix, Θ(n^2)
```

Limite inferiore per costruzione ABR

Se fosse possibile costruire un albero in $o(n \log n)$, allora sarebbe possibile visitare l'albero tramite una in-visita in tempo $O(n)$, ottenendo i valori ordinati. Avremmo così realizzato un algoritmo di ordinamento che opera in tempo $o(n \log n)$.

Sequenza specchiata

Non sono necessarie particolari tecniche per risolvere questo esercizio; è sufficiente fare una scansione di costo lineare $\Theta(n)$.

int mirror(**int** [] V, **int** n)

```
int count0, count1 = 0
int largest = 0
for i = 1 to n do
    if V[i] == 0 then
        if count1 > 0 then
            count0 = 0
            count1 = 0
        count0 = count0 + 1
    else
        count1 = count1 + 1
        largest = max(largest, 2 · min(count0, count1))
return largest
```

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