Una lezione dal Maestro E. W. Dijkstra

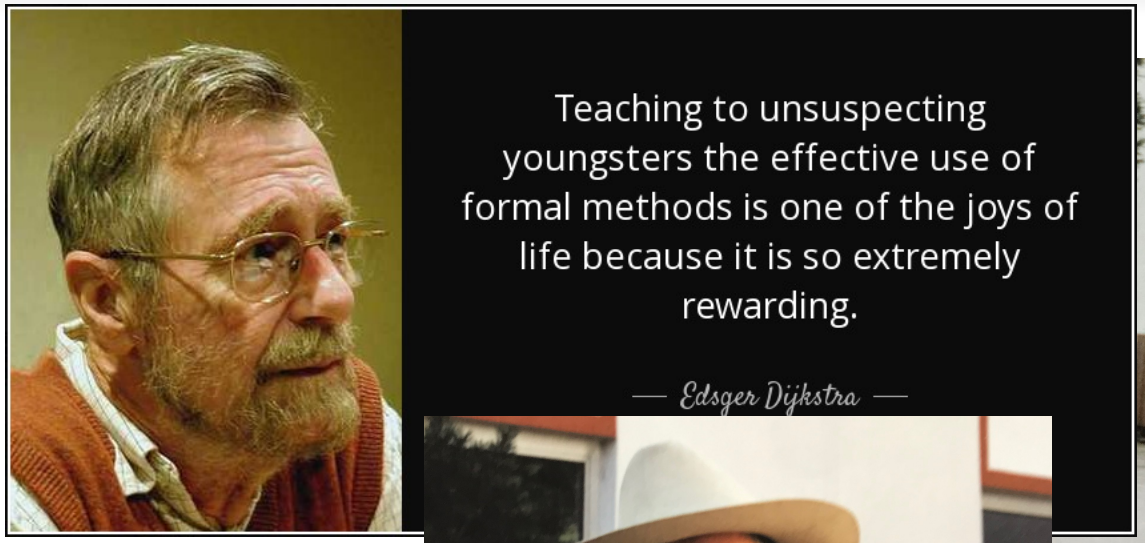
corso di laurea in **Matematica**

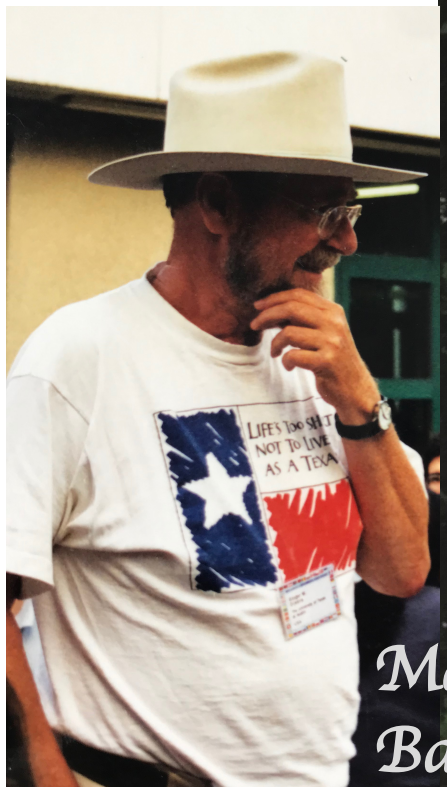
Informatica Generale, Lezione **14(b)**

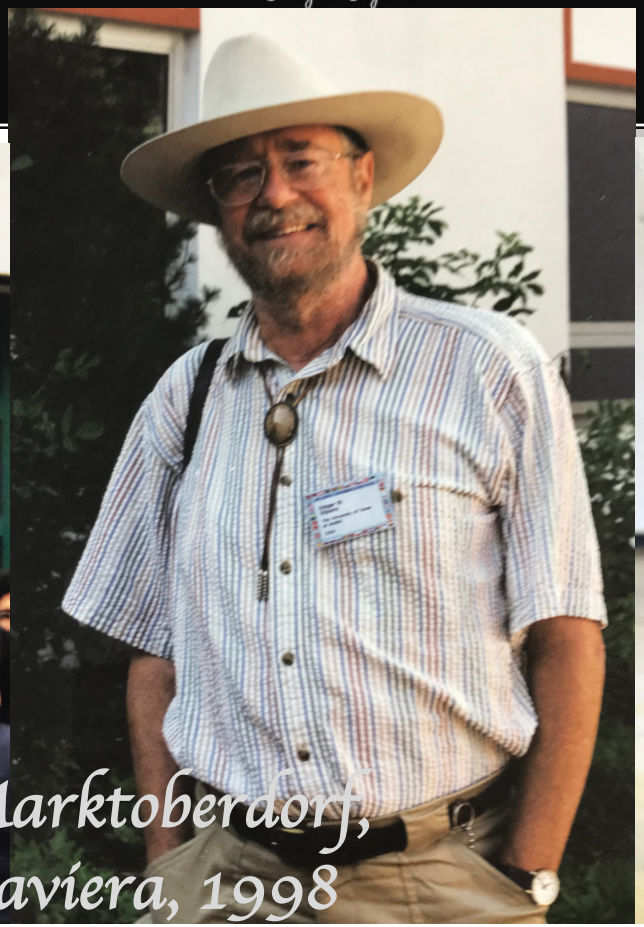
Ivano Salvo

SAPIENZA
UNIVERSITÀ DI ROMA

Un ricordo di Edsgar W. Dijkstra







Probl.: Uguaglianza a meno di shift

Problema: Scrivere un programma che verifica se due vettori u e v di n elementi sono **uguali a meno di shift**, interpretando i **vettori come circolari** e ritornare lo shift che li rende uguali.

Dobbiamo verificare **se esiste k** ($0 \le k < n$) per cui:

$$u[i] = v[i+k \bmod n] \text{ per ogni } i \ (0 \le i < n)$$

[Nella formulazione originaria, si consideravano due funzioni a **dominio finito periodiche**, ma non fa differenza sostanziale].

Non dovrebbe essere difficile trovare una funzione quadratica.

Anche se il programma può fare molte meno operazioni, ci sono dei casi pessimi quadratici: $SA_0 = \underbrace{000\ldots0}_{n-1}1$ e $SB_0 = \underbrace{000\ldots0}_{n-2}11$).

```
def equalShift(u, v):
    n = len(u)
    for k=0 to n-1:
        if equalCircular(u, v, k)
            return k
    return -1
```

```
def equalCircular(u, v, k):
    n = len(u)
    for i=0 to n-1:
        if u[i]!=v[(i+k) % n]
            return False
    return True
```

Lavorare per una specifica "migliore"

Risulta comodo avere una **specifica simmetrica** in cui i due vettori u e v abbiano lo stesso ruolo. D'ora in poi interpreteremo sempre gli indici in modo circolare, per cui $u[i] = u[i \bmod k]$.

Inoltre, conviene semplificare le notazioni denotando con $\mathbf{U}_i$ e $\mathbf{V}_i$ le sequenze ottenute leggendo i vettori u e v in **modo circolare**:

$$U_i = u[i]\ u[i+1]\ \ldots\ u[n-1]\ u[0]\ u[1]\ \ldots\ u[i-2]\ u[i-1]$$

$$V_i = v[i]\ v[i+1]\ \ldots\ v[n-1]\ v[0]\ v[1]\ \ldots\ v[i-2]\ v[i-1]$$

Riscriviamo quindi la specifica in modo **compatto** e **simmetrico**:

$$\exists\, i, j.\ \mathbf{U}_i = \mathbf{V}_j$$

Testare l'uguaglianza di **tutte le coppie** U_i e V_j porta a un programma **almeno quadratico**, sulla falsariga del precedente.

Tuttavia, **trovando un ordine nella ricerca** permette di recuperare il lavoro fatto in una verifica terminata con insuccesso.

Andando con ordine…

Consideriamo gli insiemi di sequenze $\mathbf{U} = \{ U_i \mid 0 \leq i < n \}$ e $\mathbf{V} = \{ V_i \mid 0 \leq i < n \}$ equipaggiati con **l'ordine lessicografico** (ricordiamo, **quello del dizionario**, per intenderci).

Ovviamente, $\exists\, i, j.\ U_i = V_j$ implica che che **U** e **V** sono **lo stesso insieme di sequenze**, e in particolare essendo **l'ordine lessicografico** un **ordine totale**, se $\exists\, i, j.\ U_i = V_j$ avrò anche che **i massimi** $max\ \mathbf{U}$ e $max\ \mathbf{V}$ sono uguali.

L'esplorazione di trovare due sequenze uguali, non sarà più completamente brute-force, ma **guidata dall'ordine lessicografico**, sempre verso sequenze maggiori, allo scopo di verificare quale delle seguenti tre proprietà sia soddisfatta:

$$\exists\, i\ U_i > max\ \mathbf{V} \qquad \exists\, i\ V_i > max\ \mathbf{U} \qquad \exists\, i, j\ U_i = V_j$$

e tornando -1 nei primi due casi e $|i - j|$ nel terzo.

...verso il lieto fine...

Consideriamo il seguente frammento di programma:

```
while h < n and u[(i+h) % n] == v[(j+h) % n]): h = h+1
```

Se usciamo da questo ciclo perché $h = n$, ovviamente abbiamo finito, scoprendo che $U_i = V_j$.

Viceversa, sappiamo che ci sono ***h* elementi iniziali comuni** tra U_i e V_j. Occorre capire se sia possibile scegliere un'altra coppia di indici che trae **vantaggio** dalle ***h* uguaglianze trovate**.

Quale informazione possiamo ricavare da $u_{i+h} < v_{j+h}$ nelle **sequenze ordinate**? $U_i < V_j \leq max\ \mathbf{V}$.

Avendo verificato h uguaglianze sappiamo anche che $U_{i+k} < V_{j+k} \leq max\ \mathbf{V}$ per ogni $0 \leq k < h$.

Possiamo di conseguenza ripartire confrontando U_{i+h+1} con V_j **sperando di salire nell'ordine lessicografico...**

Simmetricamente $u_{i+h} > v_{j+h}$ possiamo saltare a confrontare U_i con V_{j+h+1} **scartando sequenze sicuramente non massime**.

Lieto fine I: asserzioni logiche

Riassumendo, nel nostro ciclo sarà sempre vero che:

$P[i, j, h] \equiv \forall k \in [0, h).u_{i+k} = v_{j+k}$: che asserisce che i due vettori coincidono nell'intervallo $[0, h)$: questo viene mantenuto vero perché quando trovo elementi uguali incremento h, mentre riazzero h, quando trovo elementi diversi.

$Q_U[i] \equiv \forall k \in [0, i).\ U_k < max\ \mathbf{V}$: tutte le sequenze **già esaminate** nel vettore u sono minori del massima max **V**: questo perché quando si incrementa i abbiamo $u_{i+h} < v_{j+h}$.

$Q_V[j] \equiv \forall k \in [0, j).\ V_k < max\ \mathbf{U}$: tutte le sequenze **già esaminate** nel vettore v sono minori della massima max **U**: questo perché quando si incrementa j abbiamo $u_{i+h} > v_{j+h}$.

Terminazione: $3n - (i + j + h)$ Ho 3 forme di modifica di i, j e h:

1) $i'=i$, $j'=j$ e $h'=h\mathbf{+1}$ implica che $i'+j'+h'=i+j+h\mathbf{+1}$;

2) $i'=i+h\mathbf{+1}$, $j'=j$ e $h'=0$ implica che $i'+j'+h'=i+j+h\mathbf{+1}$;

3) $i'=i$, $j'=j+h\mathbf{+1}$ e $h'=0$ implica che $i'+j'+h'=i+j+h\mathbf{+1}$;

Osservazione: il +1 **è essenziale** ☺ e si fanno al più $3n$ **iterazioni**.

Lieto fine II: codice algoritmo

Ecco l'algoritmo: usiamo `x` e `y` per comodità per non riscrivere (e **non ricalcolare**) l'espressione `(i+h) % n`.

Se h=n allora abbiamo verificato n uguaglianze.

```
def equalsShift(u, v):
    i, j, h = 0, 0, 0
    while i<n and j<n and h<n:
        x, y = (i+h) % n, (j+h) % n
        if u[x]==v[y]: h = h+1
           else if u[x] < v[y]: i = i + h + 1
                   else j = j + h + 1
                h = 0
    if h < n return False, 42
    return True, abs(i - j)
```