

### Come calcolare la distanza di Levenshtein in Go

La distanza di Levenshtein è una misura della differenza tra due stringhe, calcolata come il numero minimo di operazioni necessarie per trasformare una stringa nell'altra. Le operazioni possibili sono l'inserimento, la cancellazione e la sostituzione di un carattere. Questo algoritmo è utile in applicazioni di ricerca testuale, correzione ortografica, bioinformatica, e altri campi che richiedono il confronto di stringhe. In questo articolo, vedremo come implementare l'algoritmo della distanza di Levenshtein in Go.

La distanza di Levenshtein tra due stringhe  $s_1$  e  $s_2$  si basa su tre operazioni:

1. **Inserimento** di un carattere.
2. **Cancellazione** di un carattere.
3. **Sostituzione** di un carattere.

Ad esempio, la distanza di Levenshtein tra "gatto" e "atto" è 1 (basta eliminare la 'g'), mentre tra "casa" e "cara" è 2 (sostituire 's' con 'r' e viceversa).

Per calcolare la distanza di Levenshtein, possiamo usare una matrice per memorizzare i risultati intermedi, ottimizzando così l'algoritmo con la **programmazione dinamica**.

L'algoritmo:

1. Creare una matrice  $dp$  con  $m + 1$  righe e  $n + 1$  colonne, dove  $m$  e  $n$  sono le lunghezze delle stringhe  $s_1$  e  $s_2$ .
2. Inizializzare la prima riga e la prima colonna con gli indici, in quanto rappresentano il numero di operazioni necessarie per convertire una stringa vuota in una stringa con  $i$  caratteri (inserimento) o da una stringa con  $j$  caratteri (cancellazione).
3. Riempire la matrice:
  - Se il carattere di  $s_1$  in posizione  $i-1$  è uguale a quello di  $s_2$  in posizione  $j-1$ , allora il costo è 0.
  - Altrimenti, calcoliamo il costo come il minimo tra:
    - Costo di cancellazione,
    - Costo di inserimento,
    - Costo di sostituzione.

Il valore in basso a destra della matrice rappresenta la distanza di Levenshtein.

Vediamo ora come implementare questo algoritmo in Go.

```
package main

import (
    "fmt"
)

// Funzione per calcolare la distanza di Levenshtein tra due stringhe
func Levenshtein(s1, s2 string) int {
    m, n := len(s1), len(s2)

    // Inizializza la matrice dp di dimensioni (m+1)x(n+1)
    dp := make([][]int, m+1)
    for i := range dp {
        dp[i] = make([]int, n+1)
    }
```

```

// Riempi la prima riga e la prima colonna
for i := 0; i <= m; i++ {
    dp[i][0] = i
}
for j := 0; j <= n; j++ {
    dp[0][j] = j
}

// Riempi la matrice dp con i costi minimi
for i := 1; i <= m; i++ {
    for j := 1; j <= n; j++ {
        if s1[i-1] == s2[j-1] {
            dp[i][j] = dp[i-1][j-1]
        } else {
            dp[i][j] = min(
                dp[i-1][j]+1,    // Cancellazione
                dp[i][j-1]+1,    // Inserimento
                dp[i-1][j-1]+1,  // Sostituzione
            )
        }
    }
}

return dp[m][n]
}

// Funzione di utilità per trovare il minimo tra tre numeri
func min(a, b, c int) int {
    if a < b && a < c {
        return a
    }
    if b < c {
        return b
    }
    return c
}

// Esempio d'uso della funzione Levenshtein
func main() {
    s1 := "gatto"
    s2 := "atto"
    fmt.Printf("La distanza di Levenshtein tra '%s' e '%s' è: %d\n", s1, s2, Levenshtein(s1,
s2))
}

```

Spiegazione:

- 1. Inizializzazione della Matrice:** La matrice dp di dimensioni  $(m+1) \times (n+1)$  viene creata con il pacchetto make, e successivamente ogni cella viene inizializzata con valori appropriati (indice di riga e colonna).
- 2. Popolamento della Matrice:** La funzione itera su ciascun carattere di entrambe le stringhe. Se i caratteri corrispondenti sono uguali, il valore della cella viene copiato dalla diagonale (costo zero). Se sono diversi, viene calcolato il minimo costo tra le tre operazioni: inserimento, cancellazione e sostituzione.
- 3. Risultato Finale:** La cella in basso a destra di dp (dp[m][n]) contiene la distanza di Levenshtein tra le due stringhe.

L'algoritmo attuale ha complessità  $O(m * n)$  sia in termini di spazio che di tempo, dove m e n sono le lunghezze delle due stringhe. Se si desidera ridurre la complessità in spazio, si può utilizzare un array unidimensionale, tenendo traccia solo della riga o colonna corrente e precedente, poiché a ogni iterazione utilizziamo solo questi valori.

## Conclusione

Implementare la distanza di Levenshtein in Go è un processo piuttosto semplice grazie al supporto per array multidimensionali e alle operazioni su stringhe. Questo approccio può essere usato in una varietà di applicazioni in cui il confronto tra stringhe è cruciale.

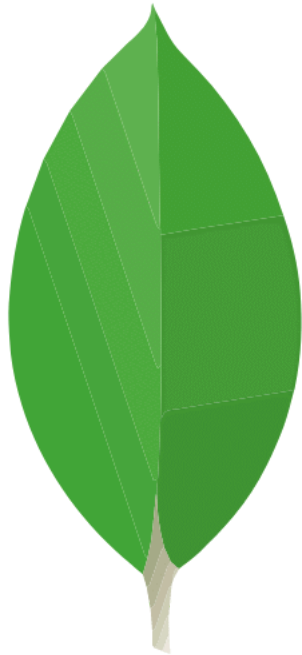
## Applicazioni Correlate



- 

### Go Placeholder Image

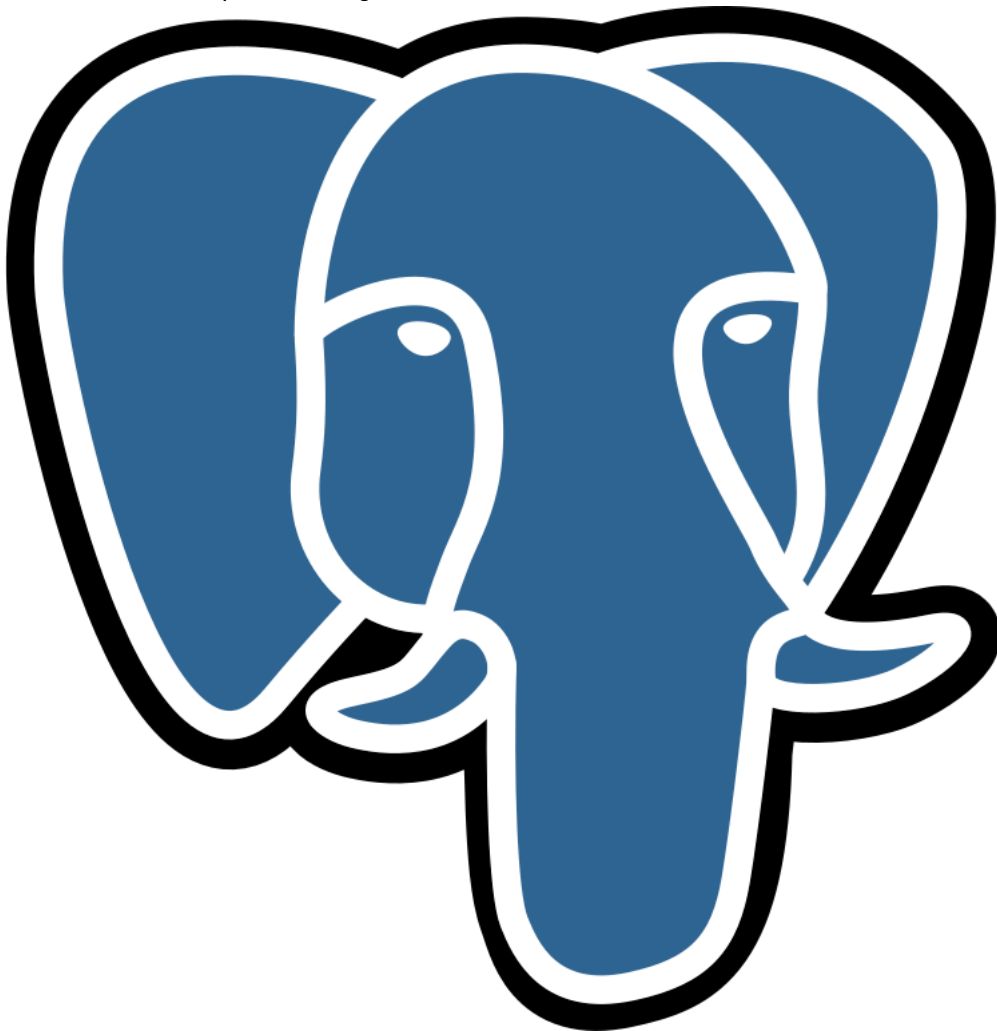
Applicazione in Go per la creazione di immagini segnaposto.  
DockerDocker ComposeGoJavaScript



•

### Go MongoDB App

Applicazione basata su MongoDB ed implementata in Go con il driver ufficiale.  
DockerDocker ComposeGoMongoDB



•

## **Go PostgreSQL App**

Applicazione basata su PostgreSQL e sviluppata in Go.

Docker Docker Compose Go PostgreSQL