Esercizio filtro

Scrivere un programma che riceva in input da *riga di comando* una stringa formata da caratteri arbitrari (nel sistema di codifica UNICODE) e stampi su standard output tutte le rotazioni della stringa data, esattamente come da esempi di esecuzione. Per rotazione si intende una stringa ottenuta muovendo il primo carattere della stringa alla fine della stringa. Sulla prima riga dovrà essere stampata la stringa in input e su ogni riga successiva dovrà essere stampata la stringa della riga precedente ruotata di un carattere. Le righe corrisponderanno al numero di caratteri della stringa in input.

Esempio d'esecuzione:

```
$ go run esercizio_filtro.go filtro
filtro
iltrof
ltrofi
trofil
rofilt
ofiltr

$ go run esercizio_filtro.go èòàéùù°
èòàéùù°
òàéùù°è
àéùù°èò
éùù°èòà
ùù°èòàé
ù°èòàéù
°èòàéùù
```

Test automatico

L'esercizio filtro è considerato esatto **solo se** rispetta le specifiche date e **solo se** passa il test automatico fornito

Inizializzazione del test automatico Al fine di poter eseguire il test automatico, è prima necessario eseguire **una volta sola** ed **esclusivamente nella cartella relativa a questo esercizio** il comando: `go mod init esercizio_filtro` ottenendo il seguente output:

```
$ go mod init esercizio_filtro
go: creating new go.mod: module esercizio_filtro
go: to add module requirements and sums:
        go mod tidy
```

Esecuzione del test automatico Successivamente sarà possibile eseguire il test automatico utilizzando il comando `go test`. L'esercizio passa il test

automatico se il comando `go test` restituisce `PASS`, come nel seguente output:

```
$ go test
PASS
ok
```

Invece, nel caso in cui l'output dovesse restituire `FAIL`, come nel seguente esempio, significa che almeno un caso tra quelli riportati nell'esempio d'esecuzione non è stato eseguito in modo corretto, ed il filtro è considerato **errato**. In tal caso `go test` restituirà anche l'output atteso dal programma e l'output effettivo che il programma produce. Questo dato è utile per capire in cosa il risultato del programma differisce dall'output atteso.

```
$ go test
--- FAIL: TestFiltro (0.00s)
        esercizio_filtro_test.go:40:
                ...
```

Esercizio 1

Scrivere un programma che legga da **riga di comando** un valore intero `soglia` e un numero indefinito di stringhe (nel sistema di codifica UNICODE).

Stampare a schermo, una per riga, tutte le coppie di stinghe con una distanza di Hamming minore di `soglia`. Ogni stringa non deve mai comparire in coppia con se stessa. Ogni coppia deve essere stampata a schermo una volta sola. Si assuma che la stessa stringa non si ripeta più volte in input.

La distanza di Hamming può essere calcolata solo su stringhe di uguale lunghezza. Si calcola derivando il numero di posizioni nelle quali i simboli corrispondenti sono diversi.

Ad esempio, le stringhe `ciao` e `cibo` hanno distanza di Hamming 1, in quanto solo in una posizione i caratteri sono diversi.

Il programma deve implementare almeno la seguente funzione:

- `HammingDistance(s1, s2 string) int`: che date due stringhe `s1` e `s2` calcola la distanza di hamming come spiegato precedentemente. La funzione deve restituire `-1` se le stringhe hanno lunghezza diversa.

Ogni coppia di stringhe dovrà essere stampata in ordine lessicografico: Es. `ciao cibo` e non `cibo ciao`. Inoltre, la stampa delle coppie su diverse righe dovrà, anch'essa, essere effettuata rispettando l'ordine lessicografico, come da *Esempio di esecuzione*.

Esempio di esecuzione

```
$ go run esercizio_1.go 1 vitale cibo D̈itale ciao Ćibi giri c̣ibo
ciao cibo
```

```
cibo çibo
vitale Ditale

$ go run esercizio_1.go 3 vitale mirate girate smielate mielate dorate
dorate girate
dorate mirate
girate mirate
girate vitale
mirate vitale

$ go run esercizio_1.go 2 ßañe cane rape ßaÿÿ ßßÚþ ßañëЌ
cane rape
cane ßañe
rape ßañe
ßañe ßaÿÿ

$ go run esercizio_1.go 1 ç°ç°é* ç°çoè+ aaa°° bba°° ciao ciaò
ciao ciaò
```

Test automatico

L'esercizio è considerato completamente esatto se rispetta le specifiche date e se passa il test automatico fornito.

Inizializzazione del test automatico Al fine di poter eseguire il test automatico, è prima necessario eseguire **una volta sola** ed **esclusivamente nella cartella relativa a questo esercizio** il comando: `go mod init esercizio_1` ottenendo il seguente output:

```
$ go mod init esercizio_1
go: creating new go.mod: module esercizio_1
go: to add module requirements and sums:
        go mod tidy
```

E' necessario, successivamente, creare l'eseguibile relativo al codice con il comando:

```
$ go build esercizio_1.go
```

Esecuzione del test automatico Successivamente sarà possibile eseguire il test automatico utilizzando il comando `go test`. L'esercizio passa il test automatico se il comando `go test` restituisce, alla fine:

```
PASS
ok
```

Esercizio 2

Scrivere un programma che legga da **standard input** una sequenza di righe di testo e termini la lettura quando, premendo la combinazione CTRL+D, viene inserito da standard input l'indicatore End-Of-File (EOF).

Esempio di input:

```
<Mario,Rossi>
<Carlo,Verdi>
```

Ogni riga rappresenta nome e cognome di una persona, nel formato `<nome,cognome>`. Si assume che sia nome che cognome siano lunghi almeno 3 caratteri. Sia nome che cognome sono composti da caratteri in formato UNICODE.

Si definisca un nuovo tipo `Persona` contenente un campo per il nome, un campo per il cognome e un campo per il nickname. In particolare, il nickname deve essere generato automaticamente prendendo i primi tre caratteri del nome e i primi tre caratteri del cognome. Ad esempio, il nickname per `Mario Rossi` è `MarRos`. Nel caso il nickname generato sia già presente, deve essere aggiunto in coda un numero progressivo (a partire da uno). Immaginiamo che ci siano tre persone: `Mario Rossi`, `Marina Rostelli`, e `Marco Rospi`. Per la prima persona, il nickname generato è `MarRos`, per la seconda `MarRos1`, mentre per la terza `MarRos2`.

Al termine della lettura, stampare (vedi *Esempio d'esecuzione*):

- tutti i nickname, ordinati in ordine alfabetico
- i soli nickname che non hanno caratteri in comune con altri nickname, in ordine alfabetico. Ad esempio, immaginiamo di avere i seguenti nickname: `MarRos`, `MetBro`, e `GiuFli`. L'unico nickname che non ha caratteri in comune con tutti gli altri è `GiuFli`. Per ognuno dei nickname stampati, mostrare anche il nome e il cognome della persona corrispondente.

Il programma deve implementare almeno la seguente funzione:

- `CaratteriDiversi(n1, n2 string) bool`: che date due stringhe `n1` e `n2` restituisce `true` se le due stringhe non hanno caratteri in comune, `false` altrimenti.

Esempio d'esecuzione:

```
$ go run esercizio_2.go < test1.txt
[ClaBis NedFhe PaoRos PaoRos1 PaoRos2]
NedFhe (Ned, Fhed)

$ go run esercizio_2.go < test2.txt
[AxéROs BăšṬŏń JòhDòè PaoBro ĆenMen ĆĀŔĆŔĀ ĆĀŔĆŔĀ1 ṔāÓḂRÓ]
AxéROs (Axél, ROse)
JòhDòè (Jòhn, Dòè)
```

```
ƤāÓƁRÓ (ƤāÓĻÓ, ƁRÓŜĩÕ)
```

Test automatico

L'esercizio è considerato completamente esatto se rispetta le specifiche date e se passa il test automatico fornito.

Inizializzazione del test automatico Al fine di poter eseguire il test automatico, è prima necessario eseguire **una volta sola** ed **esclusivamente nella cartella relativa a questo esercizio** il comando: `go mod init esercizio_2` ottenendo il seguente output:

```
$ go mod init esercizio_2
go: creating new go.mod: module esercizio_2
go: to add module requirements and sums:
        go mod tidy
```

E' necessario, successivamente, creare l'eseguibile relativo al codice con il comando:

```
$ go build esercizio_2.go
```

Esecuzione del test automatico Successivamente sarà possibile eseguire il test automatico utilizzando il comando `go test`. L'esercizio passa il test automatico se il comando `go test` restituisce, alla fine:

```
PASS
ok
```

Esercizio 3

Scrivere un programma che legga da **riga di comando** una sequenza di 4 valori interi (due coppie). Si assuma ciascuna coppia rappresenti coefficiente ed esponente di un termine nella variabile x. Ad esempio, inserendo `2 3 -1 1`, i valori `2 3` rappresenteranno il termine `2x3` e i valori `-1 1` rappresenteranno il termine `-1x`, formando quindi, il polinomio `-1x+2x3`.

Il programma deve stampare su righe diverse, come da esempio di esecuzione, le seguenti informazioni:

- La stringa corrispondente al polinomio letto, in ordine crescente di grado
- Il grado del polinomio (massimo esponente dei termini che costituiscono il polinomio)
- Il coefficiente del termine di grado 1

NOTA nella sequenza di input lo stesso esponente può comparire più volte: in tal caso i rispettivi coefficienti andranno sommati. Ad esempio, i valori `2 3 1 3`

corrisponderanno rispettivamente ai termini `2x3` e `1x3` (entrambi di grado 3). Il polinomio risultante dovrà quindi essere `3x3`.

Si definisca un tipo `Polinomio` che rappresenti polinomi in una variabile a coefficienti interi.

Oltre alla funzione `main()`, devono essere definite ed utilizzate almeno le seguenti funzioni:

- Una funzione `Crea(coeff int, esp uint) Polinomio` che restituisce un Polinomio elementare rappresentato dal coefficiente `coeff` e dall'esponente `esp`.
- una funzione `Coeff(p Polinomio, esp uint) int` che dato un polinomio restituisce il coefficiente del termine del polinomio con l'esponente specificato nella variabile `esp`.
- una funzione `Grado(p Polinomio) uint` che dato un polinomio ne restituisce il grado, ovvero il più alto esponente dei termini che compongono il polinomio.
- una funzione `Somma(p1, p2 Polinomio) (p Polinomio)` che esegue la somma tra polinomi e restituisce il polinomio somma `p`.
- una funzione `StringPoli(p Polinomio) (s string)` che riceve in input un polinomio `p` e ne restituisce una rappresentazione sotto forma di stringa (Esempio: `-1-7x+2x3`). **Suggerimento:** La funzione `FormatUint` del pacchetto `strconv` permette di convertire un intero senza segno in stringa. Nello specifico,

  ```
  - i valori che precedono e seguono `x` sono il coefficiente
    e l'esponente della variabile, rispettivamente
  - i termini devono comparire in ordine crescente di
    grado (esponente)
  - termini con coefficiente 0 non devono comparire, tranne
    nel caso particolare di polinomio nullo
  - gli esponenti 0 e 1 sono omessi; nel caso di esponente 0,
    anche il simbolo `x` va omesso
  - se il primo termine è positivo il segno '+' va omesso
  - non devono esserci spazi nella stringa
  ```

Si assuma che i valori letti da **riga di comando** siano specificati nel formato corretto.

Esempio d'esecuzione:

```
$ go run esercizio_3.go 2 3 -1 1
Polinomio: -1x+2x3
Grado: 3
Coefficiente termine grado 1: -1
```

Test automatico

L'esercizio è considerato completamente esatto se rispetta le specifiche date e se passa il test automatico fornito.

Inizializzazione del test automatico Al fine di poter eseguire il test automatico, è prima necessario eseguire **una volta sola** ed **esclusivamente nella cartella relativa a questo esercizio** il comando: `go mod init esercizio_3` ottenendo il seguente output:

```
$ go mod init esercizio_3
go: creating new go.mod: module esercizio_3
go: to add module requirements and sums:
        go mod tidy
```

Esecuzione del test automatico Successivamente sarà possibile eseguire il test automatico utilizzando il comando `go test`. L'esercizio passa il test automatico se il comando `go test` restituisce `PASS`, come nel seguente output:

```
$ go test
PASS
ok
```