Esercizio 1

Scrivere un programma che legga da **standard input** due interi `n1` e `n2`. Il programma deve stampare a video la sequenza di numeri che vanno da `n1` a `n2` come mostrato nell'**Esempio d'esecuzione**. In particolare:

- ogni numero divisibile per 3 deve essere rimpiazzato dalla parola `"Din"`;
- ogni numero divisibile per 5 deve essere rimpiazzato dalla parola `"Don"`;
- ogni numero divisibile per 7 deve essere rimpiazzato dalla parola `"Dan"`;
- ogni numero divisibile per 15 deve essere rimpiazzato dalla parola `"DinDon"`;
- ogni numero divisibile per 21 deve essere rimpiazzato dalla parola `"DinDan"`;
- ogni numero divisibile per 35 deve essere rimpiazzato dalla parola `"DonDan"`;
- ogni numero divisibile per 105 deve essere rimpiazzato dalla parola `"DinDonDan"`.

Se un numero è divisibile per diversi valori, il numero deve essere rimpiazzato dalla parola associata al divisore più grande. Ad esempio, il numero 12, divisibile per 2, 3 e 6, deve essere rimpiazzato dalla parola `"DinDon"`.

Si assuma che i valore letti da **standard input** siano nel formato corretto, con `n1 < n2`.

Esempio d'esecuzione:

```
$ go run esercizio_1.go
90 110
DinDon Dan 92 Din 94 Don Din 97 Dan Din Don 101 Din 103 104 DinDonDan 106 107 Din
109 Don

$ go run esercizio_1.go
1 7
1 2 Din 4 Don Din Dan

$ go run esercizio_1.go
14 40
Dan DinDon 16 17 Din 19 Don DinDan 22 23 Din Don 26 Din Dan 29 DinDon 31 32 Din 34
DonDan Din 37 38 Din Don

$ go run esercizio_1.go
25 35
Don 26 Din Dan 29 DinDon 31 32 Din 34 DonDan
```

Esercizio 2

Un robot è in grado di muoversi nelle quattro direzioni nord, sud, est e ovest.

In particolare, il robot accetta comandi che consistono in una coppia di valori `d p`, dove `d` è una stringa che indica la direzione e può assumere uno dei quattro valori `NORD`, `SUD`, `EST`, `OVEST`, mentre `p > 0` è un intero che indica il numero di passi che il robot deve compiere in quella direzione.

Scrivere un programma che:

- legga da **standard input** una sequenza di righe di testo;
- termini la lettura quando, premendo la combinazione di tasti `Ctrl+D`, viene inserito da **standard input** l'indicatore End-Of-File (EOF).

Ogni riga di testo consiste in una coppia di valori `d p` che descrivono un comando che il robot deve eseguire.

Dopo aver terminato la fase di lettura, come mostrato nell'**Esempio d'esecuzione**, il programma deve stampare a video:

- il numero totale di passi che deve compiere il robot in ognuna delle quattro direzioni (se il robot non si deve mai muovere in una certa direzione, l'output relativo a tale direzione non va stampato);
- la direzione in cui il robot deve compiere il minor numero totale di passi (a parità di numero di passi, tra due direzioni deve essere selezionata quella che precede l'altra in ordine alfabetico);
- la sequenza di comandi inversi che si dovrebbe impartire al robot per farlo ritornare al punto di partenza lungo lo stesso percorso.

Inoltre, all'interno del programma:

1. deve essere definito il tipo `Comando`:

   ```
   type Comando struct {
       direzione string
       passi int
   }
   ```

2. oltre alla funzione main(), devono essere definite ed utilizzate almeno le seguenti funzioni:
3. una funzione `LeggiComandi() []Comando` che legge da **standard input** un testo su più righe e terminato dall'indicatore EOF, restituendo un valore `[]Comando` in cui è memorizzata la sequenza di comandi specificati nelle righe di testo lette;
4. una funzione `AnalizzaComandi(comandi []Comando) map[string]int` che riceve in input un valore `[]Comando` nel parametro `comandi` e restituisce un valore `map[string]int` in cui, per ogni direzione specificata per almeno un comando presente in `comandi`, è memorizzato il numero totale di passi che il robot deve compiere in quella direzione rispetto alla totalità dei comandi presenti in `comandi`.

Esempio d'esecuzione:

```
$ go run esercizio_2.go
OVEST 2
NORD 1
EST 5
SUD 4
EST 5
NORD 3
SUD 1
OVEST 3

Movimenti totali:
OVEST 5
NORD 4
EST 10
SUD 5
Direzione in cui il robot deve compiere il minor numero totale di passi:
```

```
NORD
Comandi inversi:
EST 3, NORD 1, SUD 3, OVEST 5, NORD 4, OVEST 5, SUD 1, EST 2

$ go run esercizio_2.go
EST 3
NORD 1
NORD 4
EST 2

Movimenti totali:
EST 5
NORD 5
Direzione in cui il robot deve compiere il minor numero totale di passi:
EST
Comandi inversi:
OVEST 2, SUD 4, SUD 1, OVEST 3

$ go run esercizio_2.go
SUD 4
OVEST 3
SUD 2
NORD 6

Movimenti totali:
SUD 6
OVEST 3
NORD 6
Direzione in cui il robot deve compiere il minor numero totale di passi:
OVEST
Comandi inversi:
SUD 6, NORD 2, EST 3, NORD 4
```

Esercizio 3

Come illustrato nell'immagine di seguito riportata, il piano cartesiano è diviso in quattro quadranti: I, II, III e IV quadrante.

```
                  ^
                  |
     II quadrante   |   I quadrante
             -------|------>
     III quadrante  |   IV quadrante
                  |
```

Sul piano cartesiano, ad ogni punto individuato da una coppia di numeri reali, chiamati rispettivamente ascissa e ordinata, può essere associata un'etichetta simbolica, generalmente una lettera maiuscola.

Scrivere un programma che legga da **riga di comando** una serie di valori. Ogni tripletta di valori specifica un punto sul piano cartesiano definito da:

1. *etichetta*: una stringa che specifica l'etichetta simbolica associata al punto (ad es.: "A", "B", ...);
2. *x*: un valore reale che specifica l'ascissa del punto;
3. *y*: un valore reale che specifica l'ordinata del punto.

Ogni coppia di punti descrive un segmento che ha per estremi i punti stessi.

Si ipotizzi che vengano inseriti da riga di comando i seguenti valori:

```
A 10.0 2.0 B 10.0 -2.0 C -10.0 -2.0
```

La coppia di punti:

```
A 10.0 2.0
B 11.5 3.0
```

specifica il segmento `AB`.

La coppia di punti:

```
A 10.0 2.0
C 8.0 1.0
```

specifica il segmento `AC`.

La coppia di punti:

```
B 11.5 3.0
C 8.0 1.0
```

specifica il segmento `BC`.

La lunghezza di ciascun segmento è pari alla distanza euclidea tra gli estremi del segmento.

Per esempio, la lunghezza del primo segmento, quello con estremi il punto `A` ed il punto `B`, è pari alla distanza euclidea tra i punti `A` e `B`: $((x_A-x_B)^2 + (y_A-y_B)^2)^{1/2}$.

Il programma inoltre deve chiedere, da **standard input**, un valore reale soglia

Una volta terminata la fase di lettura, il programma deve stampare a video (come mostrato nell'**Esempio di esecuzione**) la descrizione di ogni segmento definibile a partire dai punti specificati nelle triplette di valori letti

da **riga di comando** tale che:

1. i punti che lo definiscono siano simmetrici rispetto all'asse y;
2. il segmento sia contenuto nel terzo e quarto quadrante
3. il segmento abbia una lunghezza maggiore del valore `soglia` letto da **standard input**.

Se non esistono segmenti che soddisfano le condizioni 1, 2 e 3, il programma non deve stampare nulla.

Si assuma che:

- le triplette di valori lette da **riga di comando** siano nel formato corretto;
- vengano lette da **riga di comando** almeno 2 triplette di valori;
- la tripletta di valori specifichi correttamente un punto sul piano cartesiano;
- il valore reale specificato a **standard input** sia nel formato corretto.

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

- una funzione `Distanza(p1, p2 Punto) float64` che riceve in input due instanze del tipo `Punto` nei parametri `p1` e `p2` e restituisce un valore `float64` pari alla distanza euclidea tra i punti rappresentati da `p1` e `p2`;
- una funzione `StringPunto(p Punto) string` che riceve in input un'instanza del tipo `Punto` nel parametro `p` e restituisce un valore `string` che corrisponde alla rappresentazione `string` di `p` nel formato `ETICHETTA = (X, Y)`, dove `ETICHETTA` è il valore `string` che specifica l'etichetta simbolica di `p`, mentre `X` ed `Y` sono i valori `float64` che specificano rispettivamente l'ascissa e l'ordinata di `p`;
- una funzione `StringSegmento(s Segmento) string` che riceve in input un'instanza del tipo `Segmento` nel parametro `s` e restituisce un valore `string` che corrisponde alla rappresentazione `string` di `s` nel formato `Segmento con estremi ESTREMO_1 e ESTREMO_2.`, dove `ESTREMO_1` ed `ESTREMO_2` sono le rappresentazioni `string` delle istanze del tipo `Punto` che rappresentano gli estremi di `s`.

Esempio d'esecuzione:

```
$ go run esercizio_3.go A 10 2 B 10 -2 C -10 -2 D -10 2
1
Segmento con estremi B = (10.00, -2.00) e C = (-10.00, -2.00).

$ go run esercizio_3.go A 10 2 B 10 -2 C -10 -2 D -10 2
20

$ go run esercizio_3.go A 3 4 C -5 1 B 3 -4 D 5 1 E -3 -4 F -3 4 G -5 -1 H 5 -1
5
Segmento con estremi B = (3.00, -4.00) e E = (-3.00, -4.00).
Segmento con estremi G = (-5.00, -1.00) e H = (5.00, -1.00).

$ go run esercizio_3.go A 3 4 C -5 1 B 3 -4 D 5 1 E -3 -4 F -3 4 G -5 -1 H 5 -1
7.0
Segmento con estremi G = (-5.00, -1.00) e H = (5.00, -1.00).
```