Il file `coordinate.txt` contiene una riga di intestazione e le coordinate (latitudine e longitudine) di 10 città.

1: File coordinate.txt

```
latitudine longitudine
12 78
25 84
31 63
44 71
52 55
61 83
68 47
75 66
83 38
91 59
```

In base alle coordinate, è possibile calcolare la distanza tra ciascuna coppia di città. La tabella seguente riporta la matrice delle distanze per le città rappresentate nel file `coordinate.txt`.

0.0	14.3	24.2	32.8	46.1	49.3	64.0	64.1	81.5	81.3
14.3	0.0	21.8	23.0	39.6	36.0	56.7	53.1	74.0	70.6
24.2	21.8	0.0	15.3	22.5	36.1	40.3	44.1	57.7	60.1
32.8	23.0	15.3	0.0	17.9	20.8	33.9	31.4	51.1	48.5
46.1	39.6	22.5	17.9	0.0	29.4	17.9	25.5	35.4	39.2
49.3	36.0	36.1	20.8	29.4	0.0	36.7	22.0	50.1	38.4
64.0	56.7	40.3	33.9	17.9	36.7	0.0	20.2	17.5	25.9
64.1	53.1	44.1	31.4	25.5	22.0	20.2	0.0	29.1	17.5
81.5	74.0	57.7	51.1	35.4	50.1	17.5	29.1	0.0	22.5
81.3	70.6	60.1	48.5	39.2	38.4	25.9	17.5	22.5	0.0

Table 1: Matrice delle distanze tra 10 città

Ad esempio:

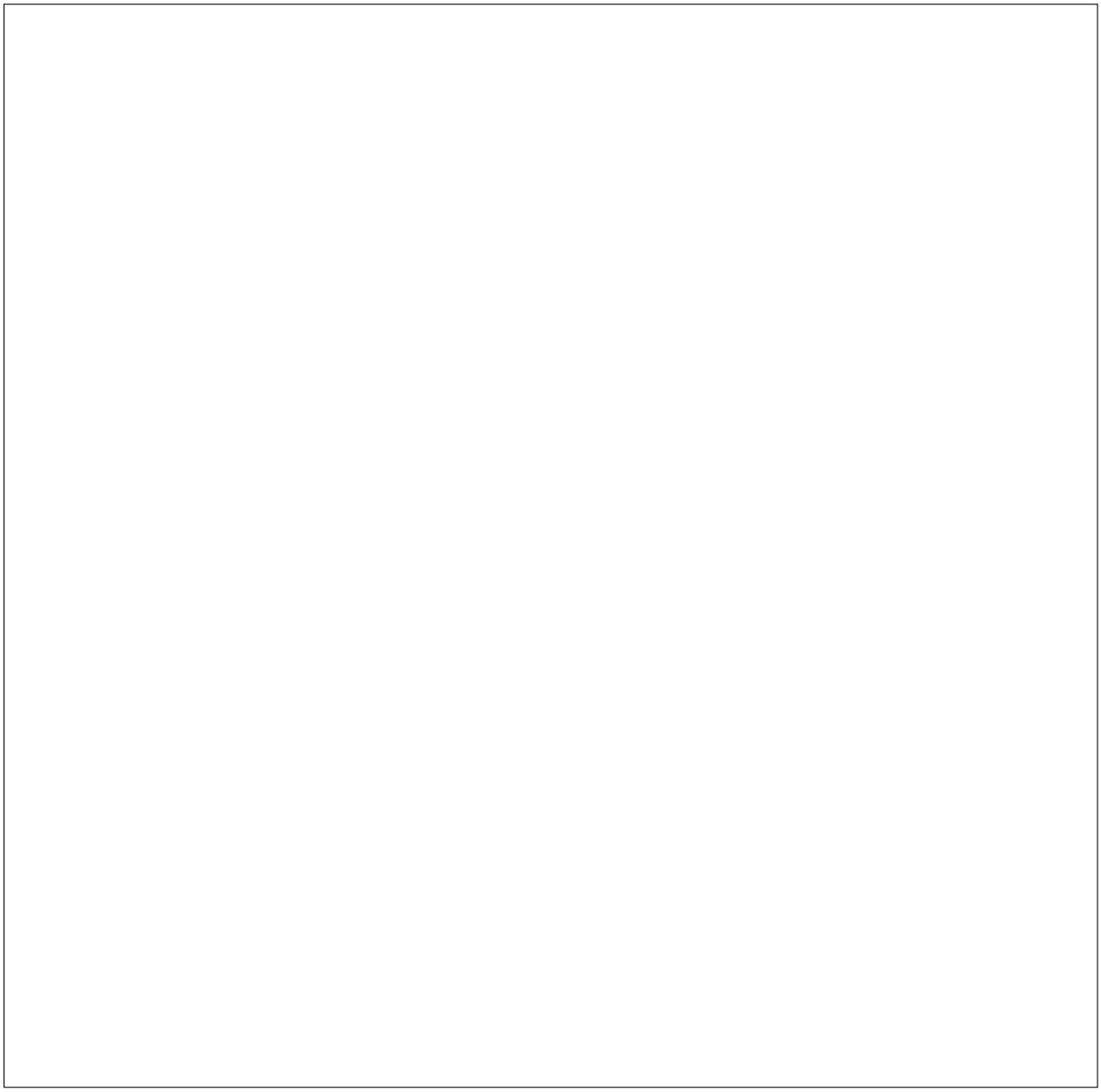
- il valore nella posizione $riga = 0$, $colonna = 1$ è pari a 14.3 e rappresenta la distanza tra la città di indice 0 (avente coordinate $latitudine = 12$, $longitudine = 78$) e la città di indice 1 (avente coordinate $latitudine = 25$, $longitudine = 84$)
- il valore nella posizione $riga = 6$, $colonna = 9$ è pari a 25.9 e rappresenta la distanza tra la città di indice 6 (avente coordinate $latitudine = 68$, $longitudine = 47$) e la città di indice 9 (avente coordinate $latitudine = 91$, $longitudine = 59$)

Un percorso completo visita tutte le città una ed una sola volta e può essere rappresentato come una lista di indici. Ad esempio, il percorso rappresentato dalla lista [3, 2, 1, 0, 4, 6, 8, 9, 7, 5], prevede la partenza dalla città di indice 3, poi la visita della città di indice 2, poi la visita della città di indice 1, e così via. Si noti che non è previsto il ritorno alla città di partenza.

1. Definire la funzione `leggi_coordinate`

   - **Parametri di ingresso**:
     - `percorso_file`: stringa
   - **Restituisce**: una lista di tuple di due interi.
   - **Descrizione**: la funzione apre il file il cui percorso è fornito in ingresso, salta la riga di intestazione ed elabora il resto del contenuto come segue: i due valori numerici contenuti in ciascuna riga vengono memorizzati come una tupla di due interi; infine, la funzione restituisce la lista di tutte le tuple lette.
   - **Esempio**: con riferimento al file `coordinate.txt`, la funzione restituisce la seguente lista: [(12, 78), (25, 84), (31, 63), (44, 71), (52, 55), (61, 83), (68, 47), (75, 66), (83, 38), (91, 59)]

2. Definire la funzione `calcola_distanza`
   - **Parametri di ingresso**:
     - `tupla_1`: tupla di due interi
     - `tupla_2`: tupla di due interi
   - **Restituisce**: un float.
   - **Descrizione**: la funzione restituisce la distanza euclidea tra i punti `tupla_1` e `tupla_2`.
   - **Esempio**: dati `tupla_1`=$(12, 78)$, `tupla_2`=$(25, 84)$, la funzione restituisce 14.31782....



3. Definire la funzione `calcola_matrice_distanze`

   - **Parametri di ingresso**:
     - `coordinate`: lista di tuple di due interi
   - **Restituisce**: una lista di liste di float
   - **Descrizione**: la funzione costruisce una matrice quadrata di dimensione $n \times n$ dove n è la lunghezza della lista in ingresso. L'elemento in posizione $[i][j]$ contiene il valore restituito dalla funzione `calcola_distanza` applicata all'i-esimo e al j-esimo elemento della lista.
   - **Esempio**: la matrice delle distanze ottenuta a partire dal file `coordinate.txt` è rappresentata in Tabella 1.

4. Definire la funzione `trova_percorso`

   - **Parametri di ingresso**:
     - `indice_partenza`: un intero.
     - `matrice`: lista di liste di float
   - **Restituisce**: una lista di interi.
   - **Descrizione**: si assuma che `matrice` sia una matrice quadrata $n \times n$, dove l'elemento in posizione $[i][j]$ rappresenta la distanza tra l'elemento di indice i e l'elemento di indice j. La funzione restituisce una lista di n indici che rappresenta l'ordine di visita degli elementi. Il primo elemento della lista restituita deve essere `indice_partenza`. Successivamente, a ogni passo, la funzione deve scegliere l'elemento non ancora visitato che ha distanza minima dall'elemento visitato più di recente. Ogni indice deve comparire nella lista restituita esattamente una volta.
   - **Esempio**: data la matrice riportata in Tabella 1 e il valore `indice_partenza`=3, la funzione restituisce la seguente lista: [3, 2, 1, 0, 4, 6, 8, 9, 7, 5].
     La città di partenza ha indice 3. La successiva città visitata ha indice 2 (distanza = 15.3). La successiva città visitata ha indice 1 (distanza = 21.8): si noti che la città a distanza minima ha indice 3 (distanza = 15.3) ma non viene considerata in quanto già visitata.

Legenda

`s, s1`: stringa `a, b`: numero `i, j, k, n`: intero `x`: elemento generico
`l, l1`: lista `d, d1`: dizionario `r, r1`: set `t, t1`: tupla

Principali funzioni built-in (ordine alfabetico)

`abs(a)`: restituisce il valore assoluto di un numero

`enumerate(iterable)`: restituisce un iteratore i cui elementi sono tuple contenenti il conteggio e il valore ottenuto dall'iterazione su iterable

`input(s)`: scrive il prompt `s` sullo standard output e restituisce stringa acquisita

`isinstance(object, classinfo)`: restituisce True se `object` è del tipo `classinfo` o di una sottoclasse di `classinfo`. Altrimenti, restituisce False

`len(x)`: restituisce la lunghezza (numero di elementi) di `x`. `x` può essere una sequenza (ad es.: string, list, tuple, range) o un contenitore (ad es.: dict, set)

`max(iterable, *, key=None), max(arg1, arg2, *args, key=None)`: restituisce il massimo nell'iterabile, o il massimo tra due o più argomenti. La funzione `key` serve a calcolare il valore da usare per confrontare gli elementi

`min`: analogo a `max`

`print(*x, sep=' ', end='\n')`: stampa oggetti, separati da `sep` e seguiti infine da `end`

`range(j), range(i, j, k)`: restituisce una sequenza immutable per progressione aritmentica

`reversed(seq)`: restituisce un iteratore che percorre `seq` in ordine inverso

`round(a, n)`: arrotonda il valore di `a` all'intero più vicino o ad `n` cifre decimali

`sorted(iterable, key=None, reverse=False)`: restituisce una nuova lista ordinata degli elementi di `iterable`. La funzione `key` serve a calcolare il valore da usare per confrontare gli elementi. `reverse=True` inverte l'ordine

`sum(iterable)`: restituisce la somma dei valori dell'iterabile

`zip(*iterables)`: itera in parallelo su diversi iterabili e restituisce un iteratore costituito da tuple. La i-esima tupla è costituita dall'i-esimo elemento di ciascuno degli iterabili

Indexing e Slicing

`seq[i]`: restituisce l'elemento di indice `i` della sequenza

`seq[i:j]`: restituisce una sottosequenza di `seq` dall'indice `i` all'indice `j` (escluso)

`seq[i:j:k]`: restituisce una sottosequenza di `seq` da indice `i` a indice `j` (escluso) con step `k`

Modulo Random

`choice(seq)`: restituisce un elemento casuale dalla sequenza `seq`

`choices(seq, k=1)`: restituisce una lista di `k` elementi scelti da `seq` con reinserimento

`random()`: restituisce un float casuale in [0,1)

`randint(i, j)`: restituisce un intero casuale tra `i` e `j` (inclusi)

`sample(seq, k)`: restituisce una lista di `k` elementi unici scelti da `seq` senza reinserimento

`shuffle(seq)`: rimescolamento della sequenza `seq` in-place

`uniform(a, b)`: restituisce un float casuale tra `a` e `b` (inclusi)

Principali funzioni del modulo math

`math.cos(a)`, `math.sin(a)`, `math.exp(a)`, `math.log(a)`, `math.log2(a)`, `math.sqrt(a)`

Principali eccezioni

`FileNotFoundError`, `IndexError`, `KeyError`, `ValueError`, `Exception` (eccezione generica)

Stringhe e principali metodi

`str()`, **Literal:** `"abc"` o `'abc'`

Trasformazioni (non in-place): `s.lower()`, `s.upper()`, `s.replace(s1,s2)`, `s.strip()`
Controlli: `s.startswith(s1)`, `s.endswith(s1)`, `s.islower()`, `s.isupper()`, `s.isdigit()`
Da lista a stringa: `s.join(l)`, Da stringa a lista: `s.split(sep)`
Altro: `s.count(s1)`, `s.index(s1)`; Conversione: a intero `int(s)`, a float `float(s)`
Concatenazione: `s1 + s2` Replicazione: `s1 * n` Membership: `s in s1`

Liste e principali metodi

`list()`, **Literal:** `[x1, x2, ...]`

Modifiche (in-place): `l.append(x)`, `l.extend(l1)`, `l.insert(i, x)`, `l.pop(i)`, `l.remove(x)`, `l.reverse()`, `l.sort(key=None, reverse=False)`
Altro: `l.count(x)`, `l.index(x)`
List comprehension: `l = [`*expression* `for` *item* `in` *iterable*`]`
Concatenazione: `l1 + l2` Replicazione: `l1 * n` Membership: `x in l`

Tuple e principali metodi

`tuple()`, **Literal:** `(x1, x2, ...)`

Altro: `t.count(x)`, `t.index(x)`
Concatenazione: `t1 + t2` Replicazione: `t1 * n` Membership: `x in t`

Insiemi e principali metodi

`set()`, **Literal:** `{x1, x2, ...}`

Modifiche (in-place): `r.add(x)`, `r.remove(x)`, `r.discard(x)`, `r.clear()`
Operazioni: `r.difference(r1)` (oppure `r - r1`), `r.symmetric_difference(r1)` (oppure `r^r1`), `r.union(r1)` (oppure `r|r1`), `r.intersection(r1)` (oppure `r&r1`)
Controlli: `r.issubset(r1)`, `r.issuperset(r1)`, `r.isdisjoint(r1)`
Membership: `x in r1`

Dizionari e principali metodi

`dict()`, **Literal:** `{k1:v1, ...}`

Modifiche (in-place): `d1.update(d2)`, `d.pop(key)`
Viste: `d.keys()`, `d.values()`, `d.items()`
Accesso: `d[key]`, `d.get(key, x=None)`
Aggiunta o modifica: `d[key] = val`
Membership: `key in d1`
Dict comprehension: `d = {`*key:val* `for` *item* `in` *iterable*`}`

f-string

`f"abc {`*expression*`:`*format*`}"`

Esempio numeri (cifre decimali): `f"{3.14:.1f}"` → `'3.1'`
Esempio stringa con allineamento (`<`, `>`, `^`): `f"{'ciao':<10}"` → `'ciao␣␣␣␣␣␣'`

```
f"{'Trieste':<10} {10:>3}"   → 'Trieste␣␣␣ ␣10'
f"{'Udine':<10} {7:>3}"      → 'Udine␣␣␣␣␣ ␣␣7'
```

File

Apertura: `open(file, mode, encoding)` apre un file e restituisce un file object `f`
Chiusura: `f.close()`; automatica se il file è aperto con il `with` statement
Lettura: `f.read()` (restituisce una stringa con l'intero contenuto del file), `f.readline()` (restituisce una stringa, fino al prossimo \n), `f.readlines()` (restituisce una lista di stringhe)
Scrittura: `f.write(s)`, `f.writelines(l)`