Laboratorio 3

Elementi di Informatica e Programmazione

Esercizio 1: Rimozione doppie

Scrivere il programma `stripDuplicates.py` che legga un'unica riga di testo e, poi, la visualizzi priva di eventuali caratteri consecutivi duplicati Esempio:

- legge `Pippo` e scrive `Pipo`
- legge `pseudo---casuale` e scrive `pseudo-casuale`
- legge `mammmma` e scrive `mama`

Soluzione

Soluzione 1: ri-assegnamo la stringa di input rimuovendo di volta in volta un doppione.

```
string = input("Inserire una stringa: ")

i = 0
while i < len(string) - 1: # attenzione al -1
    if string[i] == string[i+1]: # confronto caratteri successivi
        string = string[:i] + string[i+1:] # cosa fa questa riga?
    else:
        i += 1
print(string)
```

Soluzione 2: converto la stringa in una lista di caratteri e uso il metodo `pop`.

```
string = input("Inserire una stringa: ")
# Converto la stringa in una lista di caratteri
characters = list(string)

i = 0
while i < len(characters) - 1:
```

```
    if characters[i] == characters[i+1]:
        # se caratteri successivi sono uguali, rimuovine uno
        characters.pop(i)
    else:
        i += 1

# unisco tutti i caratteri rimasti nella lista in un'unica stringa
string = "".join(characters)
print(string)
```

Esercizio 2: Tavole Pitagoriche delle potenze

Scrivere il programma `tableOfPowers.py` che visualizzi una tavola pitagorica dopo aver chiesto all'utente i valori massimi della base e dell'esponente, che devono essere numeri interi positivi. Ogni riga della tabella deve contenere le potenze consecutive di una stessa base, con base crescente dall'alto in basso (come in una normale tavola pitagorica). Porre particolare cura nell'impaginazione della tabella, che deve rispettare queste regole:

- In ciascuna riga, due valori consecutivi devono essere separati da almeno uno spazio.
- In ciascuna colonna, tutti i valori devono essere "allineati a destra", cioè la cifra che rappresenta le unità di un valore deve trovarsi nella stessa colonna della cifra che rappresenta le unità degli altri valori.
- La lunghezza delle righe deve essere quella minima che risulta compatibile con le regole precedenti.

Ad esempio, quando il valore massimo della base è 10 e il massimo esponente è 5, la tabella formattata correttamente è la seguente.

```
 1   1    1     1      1
 2   4    8    16     32
 3   9   27    81    243
 4  16   64   256   1024
 5  25  125   625   3125
 6  36  216  1296   7776
 7  49  343  2401  16807
 8  64  512  4096  32768
 9  81  729  6561  59049
10 100 1000 10000 100000
```

Suggerimenti:

- potete usare gli operatori di formattazione delle stringhe. Ricordate che `"%10d"` formatta un numero intero in modo che occupi *almeno* 10 spazi. Se ha meno di 10 cifre allora un numero appropriato di spazi verrà aggiunto *all'inizio*.
- notate che ogni colonna ha una larghezza diversa. Qual è la larghezza di ogni colonna? Usate la risposta a questa domanda per costruire la stringa di formattazione di ogni elemento della tabella.

Soluzione

```
max_base = int(input("Inserire la base massima: "))
max_exponent = int(input("Inserire l'esponente massimo"))

assert max_base >= 1
assert max_exponent >= 1

# Iterazione su tutte le basi valide, a partire da 1
# fino a max_base incluso
for base in range(1, max_base+1):
    # Iterazione su tutti gli esponenti (colonne della tabella) validi,
    # a partire da 1 fino a max_exponent incluso
    for exponent in range(1, max_exponent+1):
        # calcolo la lunghezza massima della colonna corrente
        max_value_len = len(str(max_base**exponent))
        elem = base ** exponent
        # costruisco la stringa di formattazione
        format_str = "%" + str( max_value_len ) + "d"
        # stampo l'elemento della tabella, aggiungendo
        # uno spazio a fine riga
        print(format_str % elem, end=" ")
    # vado a capo alla fine della riga della tabella
    print()
```

Esercizio 3: quadrati magici

Scrivere il programma `isPerfectMagicSquare.py` che verifichi se un quadrato di numeri interi è un "quadrato magico perfetto". Una disposizione bidimensionale di numeri tutti diversi avente dimensione $n \times n$ è un quadrato magico se la somma degli elementi di ogni riga, di ogni colonna e delle due diagonali principali ha lo stesso valore, detto "costante magica" o "somma magica" del quadrato. Se, in aggiunta alle condizioni precedenti, i numeri presenti nel quadrato di dimensione n sono i numeri interi da 1 a n^2, allora il quadrato magico si dice "perfetto". Il quadrato magico di dimensione 1 può essere considerato un caso limite o degenere,

ma il programma deve riconoscerlo come corretto (non esistono, invece, quadrati magici di dimensione 2, né perfetti né imperfetti, come è facile dimostrare matematicamente).

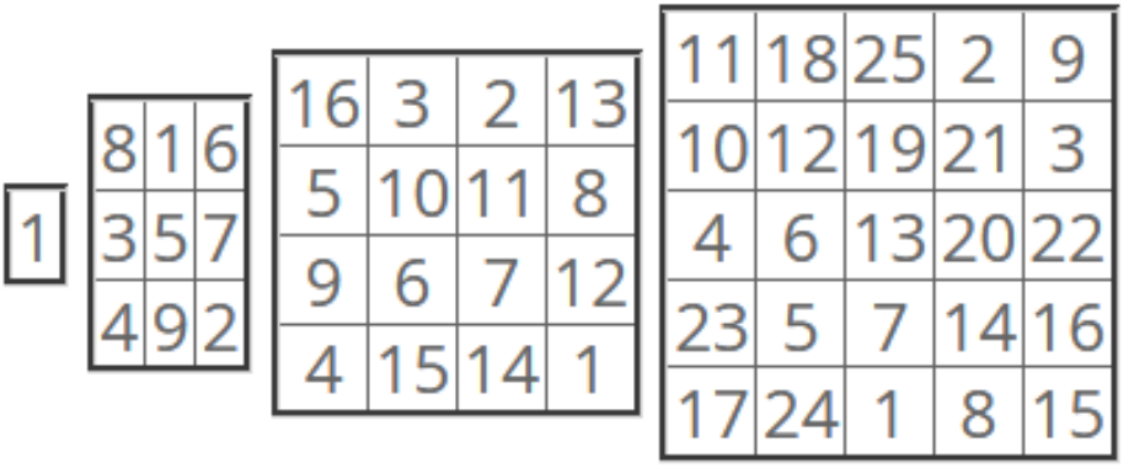

Figure 1: Esempi di quadrati magici

L'utente introduce i numeri del (presunto) quadrato perfetto riga per riga, in sequenza, separati da almeno uno spazio (il numero di righe viene dedotto dal programma esaminando il numero di colonne presenti nella prima riga). A quel punto, il programma deve intraprendere le azioni seguenti:

1. verificare che nei dati forniti in input ci siano tante righe quante colonne e che tutte le righe/colonne abbiano la stessa lunghezza: in caso contrario, il programma termina segnalando un fallimento;
2. verificare che la sequenza di valori introdotta contenga tutti (e soli) i numeri da 1 a n2, senza ripetizioni: in caso contrario, il programma termina segnalando un fallimento;
3. visualizzare la matrice, incolonnata correttamente
4. verificare la validità delle regole del quadrato magico, interrompendo la verifica con la segnalazione di fallimento non appena una regola non sia rispettata;
5. segnalare il successo della verifica. Collaudare il programma con gli esempi riportati sopra riportati.

Soluzione

```
def read_square():
    """
    Questa funzione legge una matrice quadrata dall'input.
    La lettura termina alla prima riga vuota.
```

```
    """
    matrix = []
    while True :
        s = input()
        if s == "" :
            break
        row = [] # riga che costruirò, leggendo i numeri in s
        i = 0
        # ignoro eventuali spazi iniziali
        while i < len(s) and s[i] == ' ' : i += 1
        # cerco i numeri nella stringa s,
        while i < len(s) :
            j = i # forse inizia un numero
            while i < len(s) and s[i].isdigit() : i += 1 # "consuma" cifre
            if i == j : # non c'erano cifre, non era un numero
                exit("Errore nella riga %i" % (1 + len(matrix)))
            # aggiungo il numero alla fine della riga che sto costruendo
            row.append(int(s[j:i]))
            # "consumo" gli spazi seguenti, fino al prossimo numero oppure alla fine
            # della stringa s
            while i < len(s) and s[i] == ' ' : i += 1
        # ho costruito una riga della matrice, la aggiungo alla matrice
        matrix.append(row)
        # se la riga appena aggiunta non è la prima riga, deve essere
        # lunga come le precedenti
        if len(matrix) != 0 :
            if len(row) != len(matrix[0]) :
                if len(row) < len(matrix[0]) :
                    exit("Riga %i troppo corta" % len(matrix))
                else :
                    exit("Riga %i troppo lunga" % len(matrix))
        # se la matrice è quadrata, ho finito di leggere dati
        if len(matrix) == len(matrix[0]) :
            break

    if len(matrix) != len(matrix[0]):
        exit("ERRORE: La matrice non è quadrata")
    return matrix


def print_square(matrix):
    """
```

```
    Stampa una matrice quadrata
    """
    size = len(matrix)
    num = size * size
    num_chars = len(str(num)) # quanti caratteri servono per rappresentare il numero più l
    for row in matrix:
        for x in row:
            format_string = "%" + str(num_chars) + "d"
            print(format_string % x, end=" ")
        print()


def check_all_numbers_present(matrix):
    """
    Questa funzione controlla che i numeri da 1 a N ci siano tutti,
    senza controllare la presenza di eventuali duplicati, perché
    è facile dimostrare che, se in un elenco di N numeri ci sono
    tutti i numeri da 1 a N, non ci possono essere duplicati
    """
    size = len(matrix)
    num = size * size
    for i in range(1, num+1):
        # cerco il numero i in ogni riga
        found = False
        for row in matrix:
            found = i in row
            if found:
                break
        if not found:
            print(i, "not found")
            return False
    return True


def check_magic(matrix):
    """
    Controlla se il quadrato è magico:

    - Somma delle colonne deve essere il numero magico
    - Somma delle righe deve essere il numero magico
    - Somma delle diagonali deve essere il numero magico
    """
```

```
    size = len(matrix)
    num = size * size
    # calcolo il numero magico
    magic_number = sum(matrix[0])
    # ... controllo che le altre righe siano uguali
    for i in range(1, size) : # inutile controllare la prima riga...
        if sum(matrix[i]) != magic_number :
            print("La riga %i è sbagliata" % (i+1))
            return False
    # ... controllo le colonne (sommare una colonna è un po' più complicato)
    for i in range(size) :
        temp = 0
        for j in range(size) :
            temp += matrix[j][i]
        if temp != magic_number :
            print("La colonna %i è sbagliata" % (i+1))
            return False
    # ... controllo le due diagonali principali
    temp = 0
    for i in range(size) :
        temp += matrix[i][i]
    if temp != magic_number :
        print("La diagonale dall'angolo superiore sinistro è sbagliata")
        return False
    temp = 0
    for i in range(size) :
        temp += matrix[i][size - i - 1] # ATTENZIONE al secondo indice...
    if temp != magic_number :
        print("La diagonale dall'angolo superiore destro è sbagliata")
        return False
    return True


# Qui eseguiamo tutto il codice
matrix = read_square()
print_square(matrix)
if check_all_numbers_present(matrix) and check_magic(matrix):
    print("OK")
else:
    exit("Il quadrato non è magico")
```

Esercizio 4: creare quadrati magici

Scrivere il programma `printPerfectMagicSquare.py` che generi e visualizzi un "quadrato magico perfetto" $n \times n$ con n numero DISPARI fornito dall'utente (la generazione di quadrati con n pari è molto più complessa). Realizzare l'algoritmo seguente (descritto mediante "pseudocodice"):

```
riga = n - 1
colonna = n // 2
for each k in 1, 2, ..., n*n
    scrivi k nella posizione [riga][colonna]
    incrementa riga e colonna
    se riga == n, allora riga = 0
    se colonna == n, allora colonna = 0
    se la posizione [riga][colonna] non è vuota, allora:
        ripristina riga e colonna ai loro valori precedenti
        decrementa riga
```

La richiesta iniziale del valore di n deve avvenire senza visualizzare nessun messaggio all'utente (che, quindi, deve essere un "utente informato"...) e il programma non deve visualizzare alcunché oltre alla matrice di numeri. In questo modo, i dati prodotti in uscita da questo programma possono essere forniti in ingresso al programma precedente (`isPerfectMagicSquare.py`), che può così essere utilizzato per collaudare questo usando la redirezione di output (prima) e di input (poi). Fare questi collaudi con tutti i numeri dispari fino a 19. Esempio (l'utente scrive 5):

```
C:\Users\Desktop\Python>python3 printPerfectMagicSquare.py > magicsquare.txt
5
C:\Users\Desktop\Python>python3 isPerfectMagicSquare.py < magicsquare.txt
11 18 25 2 9
10 12 19 21 3
4 6 13 20 22
23 5 7 14 16
17 24 1 8 15
OK
```

Soluzione

```python
n = int(input())
assert n >= 1
assert n % 2 == 1
```

```
matrix = []
# riempio la matrice di zeri, che durante l'esecuzione dell'algoritmo
# di generazione del quadrato magico significheranno "posizione non
# ancora scritta", perché 0 non è un valore valido nel quadrato
for i in range(n) :
    matrix.append([0]*n) # ogni riga ha n zeri

# eseguo l'algoritmo suggerito
row = n - 1
column = n // 2
for k in range(1, 1 + n * n) :
    matrix[row][column] = k
    # devo memorizzare i valori attuali di row e column per poter
    # "ripristinare i loro valori precedenti", come previsto dall'algoritmo
    oldrow = row
    oldcolumn = column
    row += 1
    column += 1
    if row == n : row = 0
    if column == n : column = 0
    if matrix[row][column] != 0 :
        row = oldrow
        column = oldcolumn
        row -= 1

# visualizzo il quadrato generato, con le spaziature opportune
width = len(str(n*n))
for i in range(n) :
    for j in range(n) :
        print(("%" + str(width) + "i") % matrix[i][j], end=" ")
    print()
```

Esercizio 5: fattorizzazione in numeri primi

Scrivere il programma `factoring.py` che scomponga un numero intero maggiore di 1 nei suoi fattori primi. Il risultato della moltiplicazione di tutti (e soli) i fattori primi visualizzati deve essere uguale al numero che si doveva scomporre, quindi eventuali fattori primi ripetuti devono essere visualizzati più volte, con la molteplicità corretta (es. il numero 300 si scompone in 2, 2, 3, 5, 5)

Soluzione

```
number = int(input("Inserire un numero intero maggiore di 1: "))
assert number > 1

divisor = 2
while divisor <= number:
    if number % divisor == 0:
        print(divisor, end=" ")
        number = number // divisor
    else:
        divisor += 1

print()
```