Esercizi di programmazione 2018/2019

Informatica@SEFA 2018/2019 - esame orale

Massimo Lauria <massimo.lauria@uniroma1.it>*

Domenica, 30 Dicembre 2018

Questi esercizi devono essere svolti prima dell'esame orale. Devo essere consegnati a vista, durante l'esame orale, su un singolo file Python `esercizio2018.py`. Il programma deve superare i test preliminari che si trovano sul sito del corso.

Valutazione dell'esercizio

Il file consegnato verrà testato con python3, in ambiente Linux o MacOS.

Autovalutazione

Avete a disposizione un file di test **preliminare**. Il programma deve passare **TUTTI** i test preliminari o non potrete fare continuare con l'esame orale. Il passaggio dei test preliminari **suggerisce solamente** che la soluzione sembra ben impostata, ma non dice nulla sulla sua correttezza.

Il file di test `test_esercizio2018.py` (link cliccabile)

e per essere eseguito, la stessa cartella deve contenere il vostro esercizio `esercizio2018.py`, insieme a tre file presenti sulla pagina del corso (i link seguenti sono cliccabili):

1. il database `registro_automobilistico_db.sqlite`
2. il database `chinook_db.sqlite`

*http://massimolauria.net/courses/infosefa2018/

3. il file di testo `holmes.txt` (codificato `utf-8-sig`)

Valutazione durante l'esame orale

Per verificare la correttezza dell'esercizio verranno eseguiti ulteriori test segreti e verranno fatte domande sul codice durante l'esame orale.

Il voto dell'esame dipenderà **anche** dall'esito di queste prove. È permesso discutere con i colleghi le tracce e gli esercizi, per risolvere incomprensioni e dubbi su cosa viene richiesto dalla traccia. **Non è assolutamente permesso** scrivere i programmi con la collaborazione di altri studenti, pena l'annullamento dell'esame orale e la cancellazione di eventuali scritti. Se non siete sicuri di come si debbano comportare le vostre funzioni nei casi limite, provate ad eseguire i test, usate il buon senso e confrontatevi con i colleghi.

Indice degli esercizi

1. Manipolazione di stringhe
    - `inverti(stringa)`
    - `cesare(stringa,n)`
2. Regine
    - `isbooleansquare(M)`
    - `isbooleanqueen(M)`
    - `isqueen(text)`
3. Elaborazione di testi
    - `conteggiotesto(testo,N)`
    - `conteggiofile(nome_file,N,encoding)`
4. Database e SQL via Python
    - `query1(N)`
    - `query2()`
    - `query3(a,b)`

1 Manipolazione di stringhe

Scrivete una funzione `inverti(stringa)` che prenda una stringa e calcoli la sua inversa, ovvero il primo carattere della nuova stringa dovrà essere l'ultimo della vecchia, e così via. Naturalmente l'esercizio non dovrà utilizzare funzioni python come `reverse` o cose simili. Dovete realizzare l'inversione con il vostro codice.

```
from esercizio2018 import inverti
print( inverti("abcdef") )
print( inverti("123456789 abcdefghi") )
print( inverti("") )
print( inverti("abacaba") )
```

```
fedcba
ihgfedcba 987654321

abacaba
```

Scrivete una funzione `cesare(stringa,n)` che realizzi il cifrario di Cesare. Ovvero data una stringa di testi, tutte le lettere latine non accentate devono essere *spostate in avanti di* `n` *posizioni*. Lo spostamento è definito secondo la seguente regola: ogni lettera viene sostituita con la lettera che si trova `n` posti in avanti nell'alfabeto. Ad esempio se `n` è 3 allora *a* viene trasformata in *d*, *b* in *e* e così via. Notate che lo schema è circolare, nel senso che la lettere *z*, ad esempio, viene trasformata nella lettera *c*. Se `n` è negativo, allora lo spostamento è all'indietro. Ad esempio *f* spostato di -5 diventa *a*.

Solo i caratteri alfabetici latini non accentati devono essere trasformati, ovvero i caratteri `a`, `b`, . . . , `y`, `z`, `A`, `B`, . . . , `Y`, `Z`. Le lettere maiuscole rimangono maiuscole e quelle minuscole rimangono minuscole.

```
from esercizio2018 import cesare
print( cesare("abcdefghijklmnopqrstuvwxyz",5) )
print( cesare("abcdefghijklmnopqrstuvwxyz",13) )
print( cesare("abcdefghijklmnopqrstuvwxyz",0) )
print( cesare("abcdefghij 1234567890 ;/?/.<> ABCDEFGHIJ",7) )
print( cesare("abcdefghij 1234567890 ;/?/.<> ABCDEFGHIJ",19) )
print( cesare("abcdefghij 1234567890 ;/?/.<> ABCDEFGHIJ",-1) )
print( cesare("abc",-2), cesare("abc",-1),
       cesare("abc", 0), cesare("abc", 1), cesare("abc", 2) )
```

```
fghijklmnopqrstuvwxyzabcde
nopqrstuvwxyzabcdefghijklm
abcdefghijklmnopqrstuvwxyz
hijklmnopq 1234567890 ;/?/.<> HIJKLMNOPQ
tuvwxyzabc 1234567890 ;/?/.<> TUVWXYZABC
zabcdefghi 1234567890 ;/?/.<> ZABCDEFGHI
yza zab abc bcd cde
```

2 Regine

In una scacchiera la regina può muoversi in otto direzioni Nord, Nord-Est, Est, Sud-Est, Sud, Sud-Ovest, Ovest, Nord-Ovest. Dunque può mangiare qualunque pezzo che si trovi in una di quelle otto direzioni se non ci sono altri pezzi in mezzo.

Introduciamo della terminologia: due regine sono in *conflitto* se sono posizionate in modo tale che si possano mangiare l'un l'altra. Un gruppo di regine posizionate sulla scacchiera sono in conflitto se ce ne sono due che lo sono. In sostanza un gruppo di regine è in conflitto se due di loro sono

- sulla stessa riga; oppure
- sulla stessa colonna; oppure
- sulla stessa parallela della diagonale; oppure
- sulla stessa parallela dell'anti-diagonale.

Il classico *problema delle regine* consiste nel porre 8 regine in una scacchiera 8×8 in modo tale che le regine non siano in conflitto. Un esempio di soluzione del problema è in Figura 1. Per esempio se invece ci fossero due regine nelle posizioni `(3,g)` e `(6,d)`, queste regine sarebbero in conflitto.

Una generalizzazione del problema delle regine su scacchiere di grandezza arbitraria è quello di porre n regine in una scacchiera di dimensioni $n \times n$, dove $n > 0$. Il movimento delle regine è lo stesso di quello della scacchiera 8×8, ma esteso naturalmente alla scacchiera più grande.

L'esercizio in questa sezione richiede di **verificare** che un posizionamento delle regine sia la soluzione del problema, ovvero che soddisfi i requisiti richiesti. L'esercizio è diviso in tre parti.

2.1 Rappresentazione del posizionamento di regine

Un modo possibile per rappresentare le regine su una scacchiera $n \times n$ è quello di utilizzare una matrice di n righe ed n colonne. In ogni cella il valore è un booleano: `True` se la cella contiene una regina, `False` altrimenti. Una matrice è facilmente rappresentabile come lista di liste.

La prima funzione `isbooleansquare(M)` da realizzare per questa sezione degli esercizi verifica che la rappresentazione della matrice quadrata booleana sia corretta.

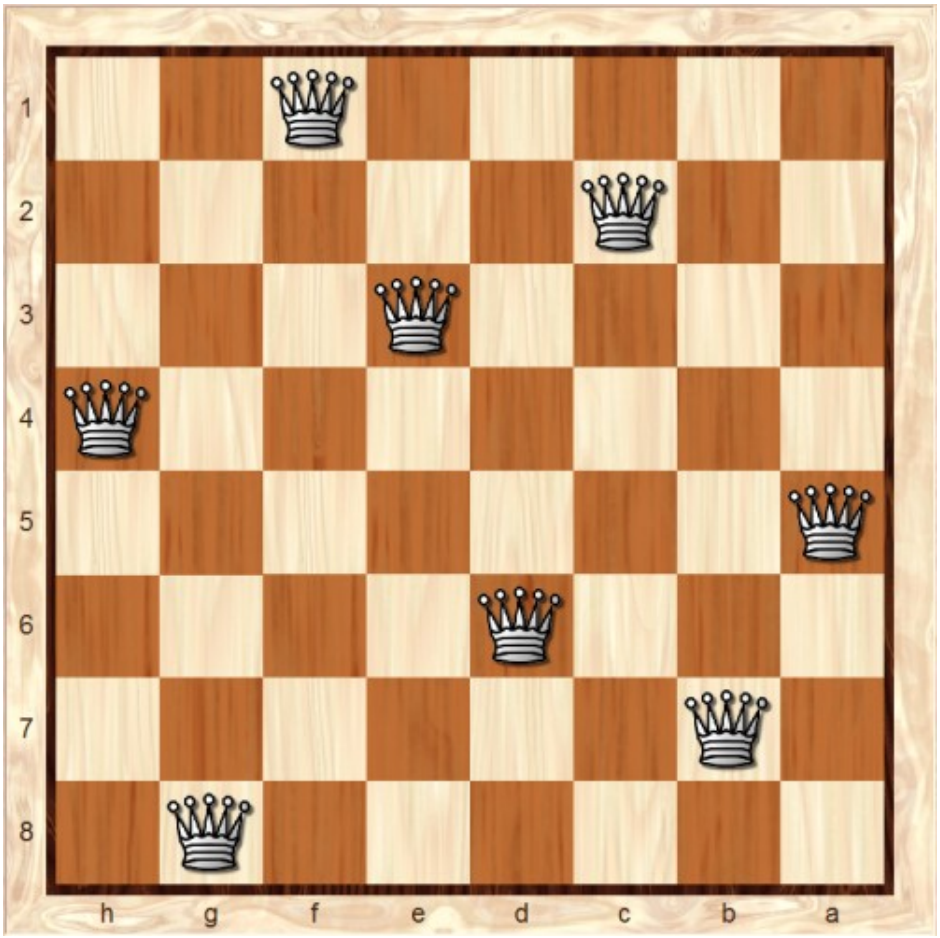

Figura 1: Una soluzione (fonte immagine: scacchi64.com)

La funzione `isbooleansquare(M)` deve restituite `True` se `M` è una lista contenente $n > 0$ liste, ognuna contenente esattamente n valori booleani (ovvero `True` o `False`). La funzione deve restituire `False` in tutti gli altri casi. Questa matrice di valori booleani rappresenta il posizionamento delle regine su una scacchiera $n \times n$.

```
from esercizio2018 import isbooleansquare

print(isbooleansquare([[True]]))

m=[[False,False,False,True ],
   [False,True, False,False],
   [False,False,False,False],
   [True ,False,False,False]]
print(isbooleansquare(m))

# Non quadrata
m=[[False,True, False,False],
   [False,False,False,False],
   [True ,False,False,False]]
print(isbooleansquare(m))

# Righe difformi
m=[[False,True,False],
   [False,False],
   [True ,False,False]]
print(isbooleansquare(m))

# Non booleana
m=[[1,0,0],
   [0,0,1],
   [0,1,0]]
print(isbooleansquare(m))
```

```
True
True
False
False
False
```

2.2 Verifica della soluzione

Questo esercizio consiste nello scrivere una funzione `isbooleanqueen(M)` che prende in input una matrice quadrata di valori booleani che rappresenta le posizioni delle regine, secondo lo schema descritto per l'esercizio precedente.

Se le regine sono posizionate senza conflitti la funzione `isbooleanqueen(M)` deve restituire `True`, altrimenti deve restituire `False`. Se `M` non è una rappresentazione corretta del posizionamento di regine in una scacchiera

secondo la specifica dell'esercizio precedente, allora la funzione deve sollevare `ValueError`. Nota che `isbooleanqueen` deve restituire `True` anche se ci sono meno di n regine sulla scacchiera, se queste sono posizionate senza conflitti.

```
from esercizio2018 import isbooleanqueen

print(isbooleanqueen([[True]]))

M=[[False,False,False,True ],
   [False,True, False,False],
   [False,False,False,False],
   [True ,False,False,False]]
print(isbooleanqueen(M))

M=[[False,False,False,True ],
   [False,True, False,False],
   [False,False,False,False],
   [False,False,True ,False]]
print(isbooleanqueen(M))
```

```
True
False
True
```

2.3 Rappresentazione testuale delle soluzioni

Una matrice di booleani è una rappresentazione un po' scomoda, e questo esercizio è una versione dei due precedenti che però utilizza una rappresentazione testuale della posizione delle regine. Scrivete una funzione `isqueen(text)` che ha in input una stringa. Se l'argomento non è una stringa allora la funzione deve sollevare `TypeError`, se è una stringa allora deve essere formattata nel modo seguente:

- una serie di (zero o più) righe vuote o costituite da soli spazi.
- n righe consecutive di n caratteri ciascuna, ogni carattere deve essere un `.` oppure una `X`. Il punto indica una posizione senza regina, mentre `X` indica una posizione con una regina. In ognuna di queste righe gli n caratteri possono essere preceduti e/o seguiti da un numero arbitrario di spazi e tabulazioni.
- una serie di (zero o più) righe vuote o costituite da soli spazi.

Se la stringa non ha il formato descritto sopra allora `isqueen` deve sollevare un `ValueError`, altrimenti deve restituire `False` o `True`, rispettivamente se ci sono due regine che si possono mangiare o meno.

```
from esercizio2018 import isqueen

ex1='''
...X
.X..
....
..X.
'''

ex2='''
.X
X.
'''

ex3='''
X......
..X....
    ....X..
......X
.X.....
...X...
.....X.
'''

ex4='''
X......
..X....
....X..
.....X.
.X.....
...X...
......X
'''

ex5='''

X....
..X..
.....
X....
.X...

'''
print(isqueen(ex1))
print(isqueen(ex2))
print(isqueen(ex3))
print(isqueen(ex4))
print(isqueen(ex5))
```

```
True
False
True
False
False
```

3 Elaborazione di testi

Questa parte riguarda l'analisi del testo, e in particolare il calcolo di qualche statistica sulle parole in esso contenute. Scriverete due funzioni: la prima

```
def conteggiotesto(testo,N):
    ...
```

deve restituire il numero di parole **distinte** all'interno della stringa `testo` che hanno lunghezza `N` caratteri . Parole uguali ma con differenti maiuscole e minuscole devono essere considerate come ripetizioni della **stessa** parola. La seconda funzione

```
def conteggiofile(nome_file,N,encoding)
    ...
```

deve fare la stessa cosa, ma invece di effettuare il conteggio in una stringa, deve farlo all'interno del file di nome `nome_file`. Il file deve essere aperto con l'encoding `encoding`. Vediamo degli esempi:

```
from esercizio2018 import conteggiofile,conteggiotesto
print(conteggiotesto('Casa caSa, casa gatto cane.',4))
print(conteggiotesto('Casa caSa, casa gatto cane.',5))
print(conteggiotesto('Casa caSa, casa gatto cane.',-3))
print(conteggiofile('holmes.txt',5, 'utf-8-sig'))
print(conteggiofile('holmes.txt',12,'utf-8-sig'))
print(conteggiofile('holmes.txt',16,'utf-8-sig'))
```

```
2
1
0
1134
154
1
```

4 Database e SQL via Python

Questa parte del compito prevede che si scrivano delle funzioni python che si interfaccino con i database SQLite e che eseguano certe query. Tutte le funzioni realizzate per questo esercizio devono restituire la lista di tuple che costituisce la risposta alla query, come visto in una delle esercitazioni. Le funzioni che accettano argomenti devono verificarne la correttezza.

Database `registro_automobilistico_db.sqlite`

- Scrivere una funzione `query1(N)` che restituisca i nomi di tutte le fabbriche che producono in totale `N` versioni di modelli di auto.

```
from esercizio2018 import query1
print(query1(3))
```

```
[('Ford',), ('Honda',)]
```

- Scrivere una funzione `query2()` che restituisca la lista dei proprietari di auto nel registro, specificando cognome, nome, e la data del primo acquisto di un veicolo. La lista deve essere ordinata per data, cognome e nome in modo crescente.

```
from esercizio2018 import query2
print(query2())
```

```
[('Bernocchi', 'Giuseppina', '1990/9/16'),
 ('Giovanolla', 'Filippo', '1993/11/30'),
 ('Spoldi', 'Diego', '1994/2/29')
{...omissis...}
```

Database `chinook_db.sqlite`.

- Scrivete una funzione `query3(a,b)` che produca i nomi degli artisti ed il loro numero di tracce nell'archivio, ma solo per quegli artisti che hanno almeno `a` e al massimo `b` tracce. Le tuple devono essere ordinate in modo decrescente rispetto al numero di tracce e (come ordine secondario) in modo crescente rispetto al nome.

```
from esercizio2018 import query3
from pprint import pprint
pprint(query3(11,13))
pprint(query3(16,20))
pprint(query3(6,10))
pprint(query3(10,7))
pprint(query3(120,300))
```

```
[('Alanis Morissette', 13),
 ('Incognito', 13),
 ('JET', 13),
 ('Velvet Revolver', 13),
 ('Alice In Chains', 12),
 ('BackBeat', 12),
 ('David Coverdale', 12),
 ('Godsmack', 12),
 ('Jota Quest', 12),
 ('Mônica Marianno', 12),
 ('Page & Plant', 12),
 ('Scorpions', 12),
 ('Stone Temple Pilots', 12),
 ('Bruce Dickinson', 11),
 ('Buddy Guy', 11),
 ('R.E.M. Feat. Kate Pearson', 11),
 ('System Of A Down', 11),
 ('The Doors', 11)]
[('Battlestar Galactica', 20),
 ('James Brown', 20),
 ('The Who', 20),
 ('House Of Pain', 19),
 ('The Black Crowes', 19),
 ('Zeca Pagodinho', 19),
 ('AC/DC', 18),
 ('Black Label Society', 18),
 ('Marisa Monte', 18),
 ('Marvin Gaye', 18),
 ('The Clash', 18),
 ('Black Sabbath', 17),
 ('Body Count', 17),
 ('Jimi Hendrix', 17),
 ('Marcos Valle', 17),
 ('Mötley Crüe', 17),
 ('O Rappa', 17),
 ('Soundgarden', 17),
 ('Def Leppard', 16),
 ('Funk Como Le Gusta', 16),
 ('Judas Priest', 16),
 ('Planet Hemp', 16)]
[('Cláudio Zoli', 10),
 ('Joe Satriani', 10),
 ('Marillion', 10),
 ('Men At Work', 10),
 ("Paul D'Ianno", 10),
 ('Raimundos', 10),
 ('Stevie Ray Vaughan & Double Trouble', 10),
 ('Temple of the Dog', 10),
 ('Dennis Chambers', 9),
 ('Frank Zappa & Captain Beefheart', 9),
 ('Pink Floyd', 9),
 ('Apocalyptica', 8),
 ('Billy Cobham', 8),
 ('Terry Bozzio, Tony Levin & Steve Stevens', 7)]
[]
[('Iron Maiden', 213), ('U2', 135)]
```