SECONDA PROVA INTERMEDIA DEL CORSO DI

FONDAMENTI DI INFORMATICA 1

CORSI DI LAUREA IN INGEGNERIA CHIMICA ED INGEGNERIA MECCANICA

26/05/2016

MOTIVARE IN MANIERA CHIARA LE SOLUZIONI PROPOSTE A CIASCUNO DEGLI ESERCIZI SVOLTI

NOME: _____ **COGNOME:** _____ **MATRICOLA:** _____

ESERCIZIO 1: RETI DI CALCOLATORI (5 punti)

Descrivere in modo chiaro e sintetico il modello ISO/OSI.

ESERCIZIO 2: ALGORITMI E LINGUAGGI (5 punti)

Definizione di algoritmo.

ESERCIZIO 3: PYTHON (23 punti)

(1) (1 punto) Scrivere una istruzione che, data una lista di N valori qualunque, stampi la sottolista con indici pari.

(2) Un gruppo di studenti universitari decide di fare il punto della situazione per capire quali esami del corso di Laurea da essi seguito risultano più "abbordabili". Per far questo, dopo un rapido censimento, stilano su file "esami.txt" una tabella organizzata nel mondo seguente (è un possibile **esempio**):

```
Analisi I; 200; 120; 23
Geometria; 150; 110; 28
Fondamenti di Informatica; 200; 150; 27
…
```

Dove la prima colonna corrisponde al nome dell'esame, la seconda al numero di studenti presenti all'ultimo appello, la terza al numero di studenti promossi all'ultimo appello, e la quarta alla media con la quale gli studenti sono stati promossi all'ultimo appello.
A questo punto, avendo molti esami da gestire e non sapendo programmare in Python, il gruppo chiede il vostro aiuto per scrivere un programma che, dalla tabella stilata, stampi a video l'esame o gli esami più abbordabili in funzione di un "indice di abbordabilità" calcolato come media aritmetica tra il punteggio normalizzato a 30 del voto ottenuto all'ultimo appello e la frazione di studenti promossi rispetto ai presenti.
Ragionando sul problema, capite che la struttura dati più adatta è il dizionario le cui chiavi sono fornite dai nomi degli esami. Ciascun valore associato sarà una lista dei valori numerici rimanenti. Nel caso dell'esempio, il dizionario prodotto sarà: `{"Analisi I": [200, 120, 23], "Geometria": [150, 110, 28],…}`.
Per ottimizzare la scrittura del codice, decidete inoltre l'implementazione delle seguenti funzioni:

- (5 punti) Funzione creaTabella(nomeFile): che riceve in ingresso il nome del file formattato come nell'esempio, lo apre, e legge il file memorizzandolo in un dizionario restituito in uscita secondo le specifiche decise sopra.
- (3 punti) Funzione generaChiave(indice,dizionario): che riceve in ingresso un indice corrispondente ad una data chiave nel dizionario e restituisce la chiave (Es. la chiave "Analisi I" ha indice 0).
- (2 punti) Funzione calcolaAbbordabilita(Lista): che ricevendo in ingresso una lista di valori relativi a studenti presenti, passati, e media, calcola il relativo indice di abbordabilità.
- (7 punti) Funzione massimaAbbordabilita(Lista): che ricevendo in ingresso una lista di valori relativi agli indici di abbordabilità, uno per ogni esame, restituisce la lista degli indici relativi ai valori massimi. Ad esempio, il primo elemento della lista restituita corrisponderà all'indice di abbordabilità dell'esame "Analisi I".
- (5 punti) Funzione main(): che leggendo un file formattato come nell'esempio, e memorizzandolo su un dizionario, stampa a video l'esame o gli esami più abbordabili.

Soluzioni

ESERCIZIO 1: RETI DI CALCOLATORI (5 punti)

Vedi dispense, Cap. 5.

ESERCIZIO 2: ALGORITMI E LINGUAGGI (5 punti)

Vedi dispense, Cap.6.

ESERCIZIO 3: PYTHON (23 punti)

Soluzione domanda 1

```
print lista[0:N:2]
```

Soluzione domanda 2

```
def creaTabella(nomeFile):
    inFile=open(nomeFile,"r")

    d={}
    linea=inFile.readline()
    while linea!="":
        istanze=linea.split(";")
        d[istanze[0]]=[float(istanze[1]), float(istanze[2]), float(istanze[3])]
        linea=inFile.readline()

    inFile.close()

    return d

def generaChiave(indice,d):
    chiave=d.keys()
    return chiave[indice]

def calcolaAbbordabilita(tupla):
    i1=tupla[1]/tupla[0]
    i2=tupla[2]/30.
    return 0.5*(i1+i2)

#Versione n.1
def massimaAbbordabilita(lista):
    massima=lista[0]
    for v in lista[1:]: #ricerca del massimo
        if v>massima:
            massima=v

    n=len(lista)        #ricerca sequenziale sui valori
    indici=[]           #corrispondenti al massimo
    i=0
    while i<n:
        if lista[i]==massima:
            indici=indici+[i]
        i=i+1

    return indici
```

```
#Versione n.2 modulare
def trova_massima(lista):
    massima=lista[0]
    for v in lista[1:]: #ricerca del massimo
        if v>massima:
            massima=v
    return massima

def ricerca_massimi_indici(lista,massima):
    n=len(lista)        #ricerca sequenziale sui valori
    indici=[]           #corrispondenti al massimo
    i=0
    while i<n:
        if lista[i]==massima:
            indici=indici+[i]
        i=i+1

    return indici

def massimaAbbordabilita(lista):
    max_abb=trova_massima(lista)
    indici=ricerca_massimi_indici(lista,max_abb)
    return indici

#Versione n.3 integrata
def massimaAbbordabilita(lista):
    n=len(lista)
    massima=lista[0]        #presumo che la lista sia fatta di almeno un elemento
    indici=[0]
    i=1
    while i<n:
        if lista[i]>massima:            #ricerca del massimo…
            massima=lista[i]
            indici=[i]                  #re-inizializzo gli indici per ogni nuovo max
        else:
            if lista[i]==massima:       #...integrato con la ricerca degli indici
                indici=indici+[i]
        i=i+1
    return indici


def main(): #si può modularizzare ulteriormente
    d=creaTabella("esami.txt")
    abbordabilita=[]
    for chiave in d:
        t=d[chiave]
        a=calcolaAbbordabilita(t)
        abbordabilita=abbordabilita+[a]

    maxAbbordabili=massimaAbbordabilita(abbordabilita)
    n=len(maxAbbordabili)
    i=0
    while i<n:
        print generaChiave(maxAbbordabili[i],d)
        i=i+1

main()
```