Analisi e progetto di un programma

Analisi top-down

ESEMPIO 1 - K MASSIMI

Facciamo un esempio giocattolo:

Problema:

Definire una funzione che riceve come parametri una lista di interi L e una costante positiva K e torna come risultato i K valori più grandi di L.

```
>>> lista = [ 2, 5, 9, 22, 90, 17, 41 ]
>>> k_massimi(lista, 3)
[90, 41, 22]
```

Cominciamo a indicare quali sono i valori in input ed in output per decidere come rappresentarli (in questo caso sono già specificati) ed inoltre se ci sono effetti collaterali (modifiche alle strutture dati fornite, a variabili globali, oppure stampe o modifiche di file).

Input: L: lista di interi K
Output: lista dei K valori massimi contenuti in L
Effetti collaterali: non specificato

Esaminiamo se esistono casi in cui la descrizione dei dati in ingresso e uscita potrebbe essere ambigua….

Esaminiamo se esistono casi in cui la descrizione dei dati in ingresso e uscita potrebbe essere ambigua.

- non viene indicato cosa accade se K è negativo o maggiore degli elementi in L,
  potremmo assumere che ci viene sempre fornita una 0 <= K <= len(L) oppure tornare meno di K elementi se K>len(L) oppure segnalare un errore
- non viene indicato se L è ordinata o meno,
- non viene indicato se i valori tornati devono essere in un ordine particolare,
  potremmo decidere di tornarli in ordine decrescente oppure tornarli in un ordine qualsiasi
- non viene indicato se la lista L può essere modificata o deve rimanere invariata,
  potremmo decidere di: modificarla distruttivamente o evitare ogni modifica
- non viene indicato cosa fare se alcuni dei valori di L sono ripetuti,
  potremmo tornare solo valori diversi tra loro oppure tornare tutti i K massimi anche se ripetuti

Decidiamo ad esempio:

-K non può essere negativo o maggiore del numero di elementi di L,
lanciamo un errore se non si ha 0 <= K <= len(L)

- non viene indicato se L è ordinata o meno,
  assumiamo il caso più generale in cui L è non ordinata
- non viene indicato se i valori tornati devono essere in un ordine particolare,
  torniamo il risultato in ordine decrescente
- non viene indicato se la lista L può essere modificata o deve rimanere invariata,
  la lista L può essere modificata
- non viene indicato cosa fare se alcuni dei valori di L sono ripetuti,
  torniamo anche eventuali ripetizioni

Un semplice algoritmo che ci permette di individuare i K valori massimi potrebbe essere:

```
# per estrarre i primi K valori massimi dalla lista L
        # lancio un errore se K è troppo grande o negativo
        # inizializzo il risultato ad una lista vuota
        # ripeto K volte
                # tolgo il massimo valore X da L
                # aggiungo X al risultato
        # torno il risultato
```

E' scritto come commenti indentati in modo da usarlo anche come documentazione del codice che bisognerà progettare

Importanza di documentare per bene il codice.

```
def k_massimi(L, K):
        ''' funzione che torna i primi K valori massimi dalla lista L '''
        # lancio un errore se K è troppo grande o negativo
        assert 0 <= K <= len(L), "K dev'essere positivo e non superare il numero di elementi di L "
        # inizializzo il risultato ad una lista vuota
        risultato = [ ]
        # ripeto K volte
        for _ in range(K):
                # tolgo il massimo valore X da L
                X = estrai_massimo(L)
                # aggiungo X al risultato
                risultato += [X]
        # torno il risultato
        return risultato
```

Resta da progettare la funzione d'appoggio estrai_massimo( )

Importanza di strutturare il codice (programmazione strutturata)

Un semplice algoritmo di ricerca del massimo e sua cancellazione dalla lista potrebbe essere:

```
# per estrarre il massimo valore dalla lista L
        # inizializzo il valore massimo al primo valore nella lista
        # scandisco uno ad uno i valori di L
                # confronto il valore corrente X con il massimo corrente
                # se X è maggiore allora aggiorno il massimo corrente con X
        # elimino il massimo da L
        # restituisco il valore massimo
```

```
def estrai_massimo(L):
        ''' si estrarre il massimo dalla lista L (rimuovendolo dalla lista) '''
        # inizializzo il valore massimo al primo valore nella lista
        massimo=L[0]
        # scandisco uno per uno i valori di L
        for X in L:
                # confronto il valore corrente X con il massimo corrente
                if X > massimo:
                        # se X è maggiore allora aggiorno il massimo corrente con X
                        massimo=X
        # elimino il massimo da L
        L.remove(massimo)
        # restituisco il valore massimo
        return massimo
```

nota che qui assumiamo che la lista non sia vuota.

L'algoritmo:

```
def k_massimi(L, K):
        ''' funzione che torna i primi K valori massimi dalla lista L '''
        assert 0 <= K <= len(L), "K dev'essere positivo e non superare il numero di elementi di L "
        risultato = [ ]
        for _ in range(K):
                X = estrai_massimo(L)
                risultato += [X]
        return risultato
```

```
def estrai_massimo(L):
        ''' si estrarre il massimo dalla lista L (rimuovendolo dalla lista) '''
        massimo=L[0]
        for X in L:
                if X > massimo:
                massimo=X
        L.remove(massimo)
        return massimo
```

Una volta implementato l'algoritmo, questo va testato per verificarne
- la correttezza
- l'efficienza.

C'è bisogno dunque di istanze di test, che possono essere prese dalla vita reale o generate casualmente.

Nel caso del nostro problema l'istanza è data da una lista ed un intero.

Ho bisogno di un metodo per generare liste di interi casuali.

Come generare una lista di N valori casuali.

Per generare numeri casuali uso il modulo 'random' che mette a disposizione diverse funzioni utili:

# per il valore iniziale interno da cui generare una sequenza di valori pseudo-casuali
random.seed(numero)

# per generare un intero casuale nell'intervallo [inizio, fine] inclusi
random.randint(inizio, fine)

# per estrarre a caso N elementi di una lista (è un campionamento quindi estrazione con ripetizione)
random.sample(lista, N)

Genero una lista di 1000 elementi presi a caso nell'intervallo 0-9999

```
lista = random.sample(range(10000), 1000)
```

La funzione %time in IPython

```
In[1]: lista = [ 2, 5, 9, 22, 90, 17, 41 ]

In[2]: %time k_massimi(lista,3)
CPU times: user 10 µs, sys: 0 ns, total: 10 µs
Wall time: 14.1 µs
Out[2]: [90, 41, 22]
```

Il numero di confronti richiesto dall'algoritmo è circa K x N

```
In[3]: import random

In[4]: random.seed(0)
In[5]: lista=random.sample(range(100000), 10000)
In[6]: %time k_massimi(lista,100)
CPU times: user 51.1 ms, sys: 2.85 ms, total: 54 ms
Wall time: 52.5 ms
Out[6]: [99996, 99995, 99992,...]

In[7]: random.seed(0)
In[8]: lista=random.sample(range(100000), 10000)
In[9]: %time k_massimi(lista,1000)
CPU times: user 375 ms, sys: 2.57 ms, total: 378 ms
Wall time: 376 ms
Out[9]: [99996, 99995, 99992,...]

In[10]: random.seed(0)
In[11]: lista=random.sample(range(100000), 10000)
In[12]: %time k_massimi(lista,10000)
CPU times: user 2.07 s, sys: 5.11 ms, total: 2.07 s
Wall time: 2.08 s
Out[12]: [99996, 99995, 99992,...]
```

Per uno stesso problema possono esserci diversi algoritmi.

Secondo algoritmo.

Potremmo immaginare un secondo algoritmo chiedendoci se avremmo potuto fare meglio se L fosse stata ordinata (in ordine decrescente).

In questo caso i K massimi che cerchiamo sono proprio i primi K valori di L, e quindi è sufficiente copiarli così come sono nel risultato. Ma L non è ordinata, quindi va ordinata.

Un algoritmo completo potrebbe essere:

```
# per estrarre i primi K massimi da una lista di interi L
        # controllo che sia 0 <= K <= len(L) oppure lancio un errore
        # ordino L in ordine decrescente
        # torno una lista che contiene solo i primi K elementi di L ordinata
```

```
def k_massimi2(L, K):
    '''torna la lista dei primi K massimi da una lista di interi L'''
    # controllo che sia 0 <= K <= len(L) oppure lancio un errore
    assert 0 <= K <= len(L), "K dev'essere positivo e non superare il numero di elementi di L"
    # ordino L in ordine decrescente
    lista_ordinata = sorted(L, reverse=True)
    # torno una lista che contiene solo i primi K elementi di L ordinata
    return lista_ordinata[:K]
```

L'algoritmo:

```
def k_massimi2(L, K):
    '''torna la lista dei primi K massimi da una lista di interi L'''
    assert 0 <= K <= len(L), "K dev'essere positivo e non superare il numero di elementi di L"
    lista_ordinata = sorted(L, reverse=True)
    return lista_ordinata[ :K]
```

Il numero di confronti per ordinare N elementi è N x log N

```
In[1]: random.seed(0)
In[2]: lista=random.sample(range(100000), 10000)

In[4]: %time k_massimi2(lista,100)
CPU times: user 6.09 ms, sys: 353 µs, total: 6.44 ms
Wall time: 6.42 ms
Out[4]: [99996, 99995, 99992,…]

In[5]: %time k_massimi2(lista,1000)
CPU times: user 4.41 ms, sys: 231 µs, total: 4.64 ms
Wall time: 4.54 ms
Out[5]: [99996, 99995, 99992,…]

In[6]: %time k_massimi2(lista,10000)
%time k_massimi2(lista,10000)
CPU times: user 4.88 ms, sys: 7 µs, total: 4.89 ms
Wall time: 4.92 ms
```

Quando K> log N conviene il secondo algoritmo

```
In[3]: import random

In[4]: random.seed(0)
In[5]: lista=random.sample(range(100000), 10000)
In[6]: %time k_massimi(lista,100)
CPU times: user 51.1 ms, sys: 2.85 ms, total: 54 ms
Wall time: 52.5 ms
Out[6]: [99996, 99995, 99992,...]

In[7]: random.seed(0)
In[8]: lista=random.sample(range(100000), 10000)
In[9]: %time k_massimi(lista,1000)
CPU times: user 375 ms, sys: 2.57 ms, total: 378 ms
Wall time: 376 ms
Out[9]: [99996, 99995, 99992,...]

In[10]: random.seed(0)
In[11]: lista=random.sample(range(100000), 10000)
In[12]: %time k_massimi(lista,10000)
CPU times: user 2.07 s, sys: 5.11 ms, total: 2.07 s
Wall time: 2.08 s
Out[12]: [99996, 99995, 99992,...]
```

ESEMPIO 2 - K MASSIMI CON LIMITI DI MEMORIA

Cerchiamo di esaminare un problema simile che ha un vincolo in più, supponiamo che i valori di L non siano inizialmente tutti disponibili ma che possano essere moltissimi e che potremmo non aver abbastanza memoria né per costruire la lista iniziale né per costruire la lista ordinata.

In questo caso dobbiamo inventare un algoritmo più furbo, che esamina un solo valore alla volta ed aggiorna una struttura dati che alla fine deve contenere solo i K valori maggiori esaminati.

Ci è sufficiente ricordare via via i K migliori valori e per ciascun nuovo valore:

- se non abbiamo ancora raccolto K valori lo aggiungiamo senza problemi
- se è minore o uguale del k-esimo massimo lo possiamo ignorare
- se è maggiore del k-esimo massimo lo inseriamo tra i k massimi e buttiamo il valore più piccolo

Conviene che i K valori via via vengano mantenuti ordinati in modo da rendere più semplici e veloci i confronti di cui sopra.

Un possibile algoritmo:

```
# per estrarre i K maggiori valori tra N
        # inizialmente i K valori massimi sono la lista vuota
        # per N volte
                # leggo ciascun valore X
                # aggiorno i K maggiori valori inserendoci X se necessario
        # torno i K valori massimi
```

Per simulare la presenza di moltissimi dati usiamo un generatore casuale di valori interi e generiamo N numeri con l'accortezza di voler ottenere la stessa sequenza di valori fissando inizialmente il seed da cui far iniziare il generatore casuale.

```
random.seed(S)
for i in range(N):
        X = random.randint ( 1, 1000000000)
        ……
```

il frammento di programma precedente per un dati due interi S ed N genera in X uno dopo l'altro tutti gli N elementi di una lista contenente interi tra 1 e 1000000000

```
# per estrarre i K maggiori valori da N generati a caso a partire dal seed S
        # inizializzo il seed del generatore con S
        # inizialmente i K valori massimi sono la lista vuota
        # genero un valore casuale X alla volta per N volte e ogni volta:
                # aggiorno i K maggiori valori inserendoci X se necessario
        # torno i K valori massimi
```

```
import random

def k_massimi_casuali(S, N, K):
'''estrae i K maggiori valori interi tra N generati a caso a partire dal seed S'''
        # inizializzo il seed del generatore con S
        random.seed(S)
        # inizialmente i K valori massimi sono la lista vuota
         massimi = []
         # per N volte
         for i in range(N):
                # genero un valore casuale X
                X = random.randint(1, 1000000000)
                # aggiorno i K maggiori valori inserendoci X se necessario
                aggiorna_K_massimi(massimi, X, K)
        # torno i K valori massimi
        return massimi
```

```
# per aggiornare i K massimi aggiungendo un nuovo valore V
        # se massimi contiene meno di K valori aggiungiamo V e terminiamo
        # se V è minore o uguale del minimo contenuto in massimi lo ignoriamo e terminiamo
        # altrimenti eliminiamo il minimo ed aggiungiamo V alla lista
```

```
def aggiorna_K_massimi(massimi, V, K):
    '''aggiorna i K massimi aggiungendo eventualmente un nuovo valore V'''
    # se massimi contiene meno di K valori aggiungiamo V e terminiamo
    if len(massimi) < K:
        massimi.append(V)
        return
    # se invece V è minore o uguale al minimo contenuto in massimi lo ignoriamo
    minimo = min(massimi)
    if V <= minimo:
        return
    # altrimenti eliminiamo il minimo ed aggiungiamo V alla lista
    massimi.remove(minimo)
    massimi.append(V)
```

L'algoritmo:

```
def k_massimi_casuali(S, N, K):
        '''estrae i K maggiori valori interi tra N generati a caso a partire dal seed S'''
        random.seed(S)
        massimi = []
        for i in range(N):
                X = random.randint(1, 1000000000)
                aggiorna_K_massimi(massimi, X, K)
        return massimi

                                def aggiorna_K_massimi(massimi, V, K):
                                        '''aggiorna i K massimi aggiungendo eventualmente un nuovo valore V'''
                                        if len(massimi) < K:
                                                massimi.append(V)
                                                return
                                        minimo = min(massimi)
                                        if V <= minimo:
                                                return
                                        massimi.remove(minimo)
                                        massimi.append(V)
```

Proviamo a vedere quanto ci mette nei tre casi in cui vogliamo 1, 10, 100 o 1000 massimi in un file con N=1000000 di interi :

```
In[1]: %time k_massimi_casuali(0,1000000,1)
CPU times: user 1.53 s, sys: 3.86 ms, total: 1.54 s
Wall time: 1.54 s
Out[1]: [999999707]

In[2]: %time k_massimi_casuali(0,1000000,10)
CPU times: user 1.7 s, sys: 3.77 ms, total: 1.7 s
Wall time: 1.7 s
Out[2]: [999997022, 999998633, 999993012,...]

In[3]: %time k_massimi_casuali(0,1000000,100)
CPU times: user 3.03 s, sys: 3.67 ms, total: 3.03 s
Wall time: 3.03 s
Out[3]: [999990156, 999910395, 999962882, ...]

In[4]:  %time k_massimi_casuali(0,1000000,1000)
CPU times: user 17.2 s, sys: 10 ms, total: 17.2 s
Wall time: 17.2 s
Out[4]: [999990156, 999910395, 999473447, 999962882]
```

Il tempo cresce molto velocemente, si può migliorare ad esempio tenendo la lista dei massimi ordinata in modo crescente (in questo caso trovare il minimo sarebbe molto piu' efficiente …

Le tracce dei 3 esercizi nell'homework sono nei 3 file:
program01.py
program02.py
program03.py
una volta risolto l'esercizio si può attivare il grader corrispondente eseguendo il file di test corrispondente,

Ad esempio per il program01.py bisogna eseguire test_01.py
da terminale :
python test_01.py
la prima volta bisogna eseguire il comando
conda install -c conda-forge ddt