Table of Contents

Fondamenti di Programmazione

Andrea Sterbini

lezione 8 - 27 ottobre 2022

RECAP: estrarre i k massimi da N valori generati a caso

$O(n)$ volte (per tutti i valori letti) aggiorno la lista di **k** migliori elementi

- tempo $O(k)$ se **non** tengo i k migliori ordinati
- oppure $O(k * log(k))$ se li tengo ordinati

Sembra inutile tenerli ordinati!

Avevo dato un suggerimento per migliorare

- trovare in tempo $O(log(k))$ dove inserire X
- inserire X (purtroppo in tempo $O(k)$)

E questo ci riporta di nuovo a $O(k)$ per aggiornare i k elementi

Ci vuole una struttura dati con inserimento/cancellazione in $O(log(k))$

SOLUZIONE:

from sortedcontainers import SortedList (https://grantjenks.com/docs/sortedcontainers/sortedlist.html)

- $O(log(k))$ inserimento di un nuovo valore
- $O(log(k))$ lettura minimo (elemento a indice 0)
- $O(log(k))$ cancellazione del minimo

Risultato finale: tempo $O(n * log(k))$

In [1]:
```
from sortedcontainers import SortedList
def aggiorna_k_massimi(massimi, X, k):
    if len(massimi) < k:
        massimi.add(X)      # O(log(k))
    elif massimi[0] < X:
        del massimi[0]      # O(log(k))
        massimi.add(X)      # O(log(k))

from random import seed, choices
def k_massimi_random(N, k, seme=0):
    seed(seme)
    massimi = SortedList()
    for X in choices(range(-1000000,1000000),k=N):
        aggiorna_k_massimi(massimi, X, k)
    return massimi
```

In [2]:
```
%timeit k_massimi_random(1000000,    30, 0)
%timeit k_massimi_random(1000000,   300, 0)
%timeit k_massimi_random(1000000,  3000, 0)
%timeit k_massimi_random(1000000,  5000, 0)
%timeit k_massimi_random(1000000, 10000, 0)
%timeit k_massimi_random(1000000, 20000, 0)
%timeit k_massimi_random(1000000, 30000, 0)
```

```
856 ms ± 70.9 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
1.1 s ± 456 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
874 ms ± 35.7 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
997 ms ± 32.6 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
1.09 s ± 25.1 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
1.14 s ± 36.9 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
1.29 s ± 74.8 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

Tipi in Python

Python NON dichiara nè controlla a runtime che i tipi dei dati forniti alle funzioni (e ritornati) siano "giusti"

Però ci sono tool per controllarli:

- **mypy** con una analisi statica del codice
- **typeguard** a run-time

Aggiungere i tipi alle definizioni delle funzioni ci fa catturare molti più errori

Sintassi

```
variabile : tipo = valore

def f( arg1 : tipo1, ...) -> return_type :
    codice della funzione
```

Queste "annotazioni" vengono inserite nella definizione della funzione e possono essere controllate con **`mypy`** oppure con **`typeguard`**

Come si descrivono i tipi

- potete usare direttamente i nomi delle corrispondenti classi
  - **bool, int, float, dict, set, tuple, list, ...**

```
pi : float = 3.14
dati : list[int] = [ 3, 6, 12, 45 ]
```

Per i contenitori possiamo indicare **tra parentesi quadre** il tipo degli elementi contenuti

tipo	descrizione
list[int]	lista omogenea di **int** (di lunghezza indefinita)
dict[str, int]	dizionario con chiavi tutte **str** e valori tutti **int**
set[float]	insieme omogeneo di **float**
tuple[int, float]	coppia di valori (**int**, **float**)
tuple[int, ...]	tupla di lunghezza variabile contenente solo **int**

Potete usare sinonimi (alias) al posto di tipi complessi

Questo aiuta a rendere più leggibili le annotazioni

```
# una scheda è un dizionario
# di coppie  'info':'valore'
Scheda = dict[str, str]
# una agendina è una lista di schede
Agenda = list[Scheda]
```

Ci sono tipi speciali

```
from typing import ...
```

- **None** per indicare il valore **None**
- **Any** per indicare che va bene qualsiasi tipo

- **NoReturn** per funzioni non tornano valori (lanciano sempre errore)
- **Union[T1, T2]** oppure **T1 | T2** per indicare che sono validi sia **T1** che **T2**

- **Optional[T]** equivale a **T | None**
- **Callable[[...], ReturnType]**
  per indicare una funzione che riceve gli argomenti **[...]** e torna un **ReturnType**
- **TypeVar** per definire tipi parametrici
- ...

In [5]:
```
## Esempio di annotazioni di tipo
from typing import Sized  # ha __len__

# qui ho un criterio di ordinamento che
# - accetta cose che hanno una lunghezza
# - e produce coppie (int,Sized)
def criterio(elemento : Sized) -> tuple[int,Sized]:
        return len(elemento), elemento

criterio.__annotations__

```

Out[5]:
```
{'elemento': typing.Sized, 'return': tuple[int, typing.Sized]}
```

In [6]:
```
# provo a eseguire un test su my_sorted
def test_sorted():
    L        : list[Sized] = [ '932', '1', '23', '045', 2 ] # <== intero!!!
    expected : list[Sized] = ['1', '23', '045', '932']
    result = sorted(L,key=criterio)
    assert result == expected

if __name__ == '__main__':
    test_sorted()
# scopriamo l'errore SOLO A RUNTIME!!!
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/tmp/ipykernel_39615/2702506111.py in <cell line: 8>()
      7
      8 if __name__ == '__main__':
----> 9     test_sorted()
     10 # scopriamo l'errore SOLO A RUNTIME!!!

/tmp/ipykernel_39615/2702506111.py in test_sorted()
      3     L        : list[Sized] = [ '932', '1', '23', '045', 2 ] # <== intero!!!
      4     expected : list[Sized] = ['1', '23', '045', '932']
----> 5     result = sorted(L,key=criterio)
      6     assert result == expected
      7
```

come verificare i tipi staticamente

```
mypy --pretty sorted.py
```

```
sorted.py:7: error: List item 4
has incompatible type "int"; expected "Sized"
L : list[Sized] = [ '932', '1', '23', '045', 2 ]
                                              ^
Found 1 error in 1 file (checked 1 source file)
```

mypy scopre l'errore di tipo già dall'assegnamento!!!

MA è anche utile verificare i tipi a runtime

- possono dipendere da dati esterni
- **mypy** non è detto che abbia tutte le info necessarie
  - moduli non tipati o tipati male
  - alcune deduzioni ancora non sono implementate

Run-time type-checking

Non si fa così: **`python sorted.py`**

```
TypeError: object of type 'int'
           has no len()
```

L'abiamo scoperto all'ultimo momento, eseguendo **len**

Run-time type-checking

NON si fa così: **`pytest sorted.py`**

```
E   TypeError: object of type 'int'
               has no len()
```

Si è scoperto solo perchè **len(int)** non funziona

Molto meglio: con typeguard

```
conda install typeguard -c conda-forge
```

Come si usa:

```
pytest sorted.py --typeguard-packages=sorted
```

```
ERROR sorted.py - TypeError: type of
argument "elemento" must be
collections.abc.Sized;
got int instead
```

BENE: adesso **typeguard** si è accorto che **2** non è di tipo **Sized**

In [8]:

```python
### Oppure (sconsigliato): MODIFICANDO IL CODICE
from typeguard import typechecked
from typing import Sized

@typechecked
def criterio(el : Sized) -> tuple[int,Sized]:
    return len(el), el

L : list[Sized] = [ 'uno', 2 ]
sorted(L, key=criterio)     # <== viene scovato qui
```

```
---------------------------------------------------------------
---------------
TypeError                                 Traceback (most r
ecent call last)
/tmp/ipykernel_39615/1097316211.py in <cell line: 10>()
      8
      9 L : list[Sized] = [ 'uno', 2 ]
---> 10 sorted(L, key=criterio)     # <== viene scovato qui

/home/opt/miniconda3/envs/F22/lib/python3.10/site-packages/
typeguard/__init__.py in wrapper(*args, **kwargs)
   1030     def wrapper(*args, **kwargs):
   1031         memo = _CallMemo(python_func, _localns, arg
s=args, kwargs=kwargs)
-> 1032         check_argument_types(memo)
   1033         retval = func(*args, **kwargs)
   1034         try:

/home/opt/miniconda3/envs/F22/lib/python3.10/site-packages/
typeguard/__init__.py in check_argument_types(memo)
    873                 check_type(description, value, expe
cted_type, memo)
    874             except TypeError as exc:  # suppress un
necessarily long tracebacks
--> 875                 raise TypeError(*exc.args) from Non
e
    876
    877     return True

TypeError: type of argument "el" must be collections.abc.Si
zed; got int instead
```

Cosa vi consiglio di fare

Definite i tipi degli argomenti e i risultati delle vostre funzioni (e le vostre variabili)

Chiamate

`mypy --pretty programma.py`

`pytest programma.py --typeguard-packages=programma`

Come usiamo i tipi negli HW

- tutte le funzioni fornite sono annotate coi tipi
- uno dei test dello HW verifica i tipi a runtime

Homework 2 obbligatorio

Dato un numero romano ('MCMLXI' = 1961) composto dei simboli

lettera	peso	lettera	peso	lettera	peso	lettera	peso
M	1000	D	500	C	100	L	50
X	10	V	5	I	1		

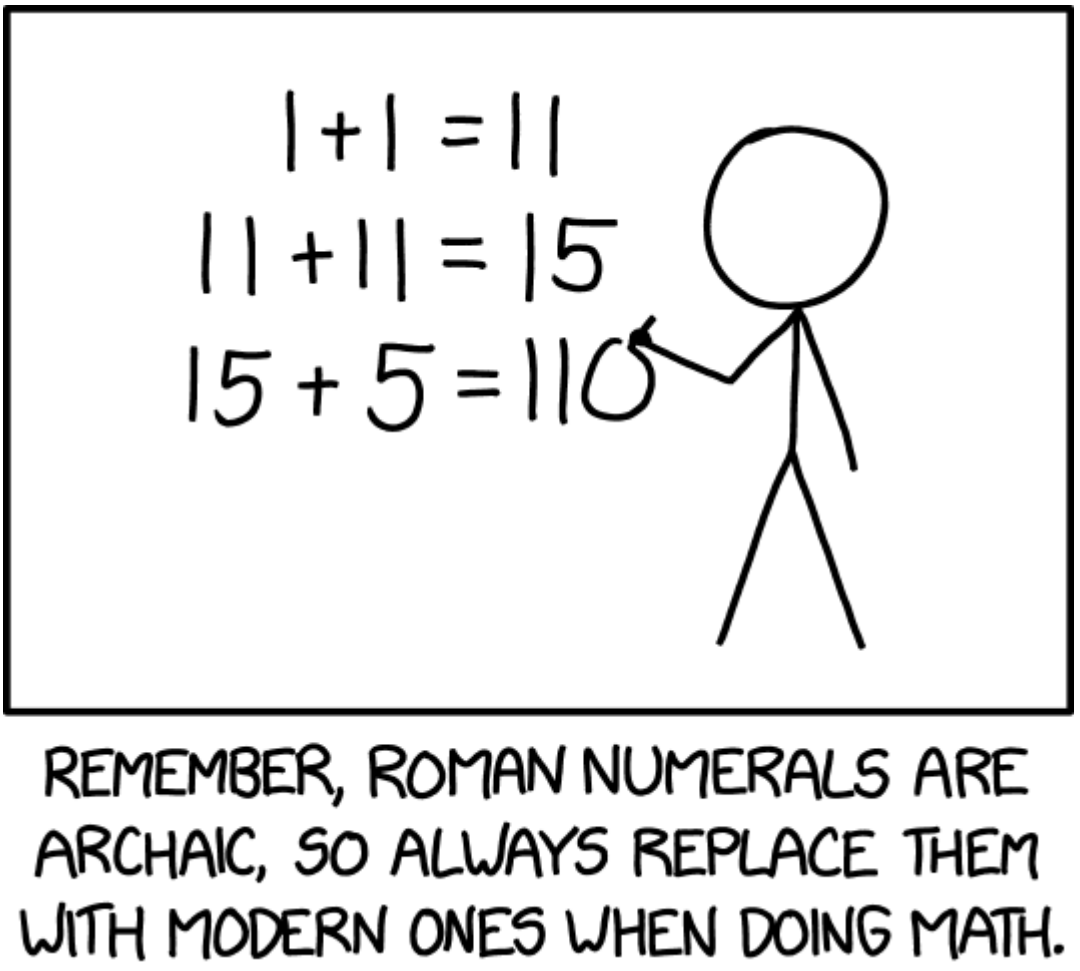
Supponiamo di scriverlo concatenando direttamente i valori

```
'MCMLXI' => 1000 100 1000 50 10 1
         => '1000100100050101'
```

Chiamiamo questo formato "formato XKCD" (http://m.xkcd.com/2637)

Il blog XKCD di Randall Munroe

1+1=2, 2+2=4, 4+5=9



Obiettivo dello HW

Dovete implementare le 4 funzioni:

```
def decode_XKCD_tuple(
    xkcd_values : tuple[str, ...],
    k : int) -> list[int]:
def decode_value(
    xkcd : str ) -> int:
def xkcd_to_list_of_weights(
    xkcd : str) -> list[int]:
def list_of_weights_to_number(
    weigths : list[int] ) -> int:
```

Perchè questo HW è così semplice?

- vi fornisce già l'analisi (per chi è alle prime armi)
- controlla che usiate i dati giusti con **typeguard**
- controlla che non scriviate codice intricato con **radon**

I prossimi HW NON verranno specificati a questo livello di dettaglio

Analisi di problemi : istogramma

Dato un elenco di dati vogliamo calcolarne la frequenza
e produrre una stampa in cui visualizziamo le frequenze
su righe successive, come barre di asterischi

Esempio:

```
1  ****
2  **
3
4  *******
```

In [ ]:

```
'''
istogramma di una serie di dati

Input:
    - lista di dati numerici (interi/float?)
    - definizione degli intervalli (implicita/esplicita
Output:
    - stringa con più righe,
        una per ogni valore
        ciascuna riga contiene il valore, spazio e
        tanti asterischi quante volte quel valore appar
    (i valori con conteggio 0 devono apparire?)
'''
```

Istogramma con estremi e dati interi

In [18]:

```
'''
# per ottenere l'istogramma conoscendo A,B e i dati
    # calcolo le frequenze dei dati
        # opzione uno: usiamo una lista se:
            # i valori sono interi
            # conosciamo l'intervallo di valori possibi
    # dalle frequenze produco la stringa
'''
def istogramma_con_estremi_e_dati_interi(A : int,
                                         B : int,
                                         dati : list[in
    frequenze = calcola_frequenze(A, B, dati)
    return produci_istogramma(A, frequenze)

def calcola_frequenze(A : int, B : int, dati : list[int
    'attenzione agli indici della lista per dati positi
    # usiamo un offset
    conteggi : list[int] = [0] *(B-A+1)
    for valore in dati:
        if A <= valore <= B: # controllo che il valore
            conteggi[valore-A] += 1
    return conteggi
def produci_istogramma(A : int, frequenze : list[int])
    testo = ''
    for indice, frequenza in enumerate(frequenze):
        testo += f"{indice+A}\t" + ("*"*frequenza) + '\
    return testo

L = [ 2, 17, 1, 4, 2, 8, 11, 2, 4, 8, 11, 3 ]
print(istogramma_con_estremi_e_dati_interi(-5, 20, L))
```

```
-5
-4
-3
-2
-1
0
1       *
2       ***
3       *
4       **
5
6
7
8       **
9
10
11      **
12
13
14
15
16
17      *
18
19
20
```

Istogramma con estremi e dati float

In [20]:

```
## WHAT IF i dati sono float?
def istogramma_con_estremi_e_dati_float(A : int, B : in
                                        dati : list[flo
    frequenze = calcola_frequenze_float(A,B, dati)
    return produci_istogramma(A, frequenze)

def calcola_frequenze_float(A : int, B : int,
                            dati : list[float]) -> list
    dati_interi = [ int(X) for X in dati ]
    return calcola_frequenze(A,B, dati_interi)

LF = [ 1.3, 5.7, 2.1, 5.8, 2.4, 4.2, 1 ]
print(istogramma_con_estremi_e_dati_float(-2,9, LF))
```

```
-2
-1
0
1       **
2       **
3
4       *
5       **
6
7
8
9
```

Istogramma usando dizionari

In [31]:

```python
# se vogliamo usare meno memoria possiamo usare un dizi
def calcola_frequenze_con_diz_int(dati : list[int]) ->
    frequenze = {}
    for valore in dati:
        if valore in frequenze:
            frequenze[valore] += 1
        else:
            frequenze[valore]  = 1
    return frequenze

def produci_istogramma_diz_con_buchi(istogramma : dict[
    # ignoro i valori con conteggio 0
    testo = ''
    for valore, frequenza in sorted(istogramma.items())
        testo += f"{valore}\t" + ('*'*frequenza) + '\n'
    return testo
def produci_istogramma_diz_senza_buchi(istogramma : dic
    # generare anche i dati con conteggio 0
    # aggiungiamo al dizionario i valori con conteggio
    m = min(istogramma)
    M = max(istogramma)
    for X in range(m,M):
        if not X in istogramma:
            istogramma[X] = 0
    return produci_istogramma_diz_con_buchi(istogramma)

from random import choices
L = choices(range(-10,10), k=30)
istogramma = calcola_frequenze_con_diz_int(L)
%pprint
L
print(produci_istogramma_diz_senza_buchi(istogramma))
```

```
Pretty printing has been turned OFF
-10     *
-9      ****
-8      *
-7
-6      *
-5      *
-4      **
-3
-2
-1      **
0       ***
1       *
2
3       *
4       ***
5       **
6       **
7       ***
8       **
9       *
```