lezione08

October 19, 2023

1 Fondamenti di Programmazione

Andrea Sterbini

lezione 8 - 19 ottobre 2023

2 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 vorrebbe una struttura dati con inserimento, cancellazione e minimo in $O(log(k))$

2.0.1 SOLUZIONE: from sortedcontainers import SortedList

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

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

[1]:
```
from sortedcontainers import SortedList
help(SortedList.add)
help(SortedList.__delitem__) # usato da 'del massimi[0]' per eliminare il minimo
help(SortedList.__getitem__) # usato da 'massimi[0]'     per leggere il minimo
```

```
Help on function add in module sortedcontainers.sortedlist:

add(self, value)
    Add `value` to sorted list.

    Runtime complexity: `O(log(n))` -- approximate.

    >>> sl = SortedList()
    >>> sl.add(3)
```

```
    >>> sl.add(1)
    >>> sl.add(2)
    >>> sl
    SortedList([1, 2, 3])

    :param value: value to add to sorted list

Help on function __delitem__ in module sortedcontainers.sortedlist:

__delitem__(self, index)
    Remove value at `index` from sorted list.

    ``sl.__delitem__(index)`` <==> ``del sl[index]``

    Supports slicing.

    Runtime complexity: `O(log(n))` -- approximate.

    >>> sl = SortedList('abcde')
    >>> del sl[2]
    >>> sl
    SortedList(['a', 'b', 'd', 'e'])
    >>> del sl[:2]
    >>> sl
    SortedList(['d', 'e'])

    :param index: integer or slice for indexing
    :raises IndexError: if index out of range

Help on function __getitem__ in module sortedcontainers.sortedlist:

__getitem__(self, index)
    Lookup value at `index` in sorted list.

    ``sl.__getitem__(index)`` <==> ``sl[index]``

    Supports slicing.

    Runtime complexity: `O(log(n))` -- approximate.

    >>> sl = SortedList('abcde')
    >>> sl[1]
    'b'
    >>> sl[-1]
    'e'
    >>> sl[2:5]
    ['c', 'd', 'e']
```

```
    :param index: integer or slice for indexing
    :return: value or list of values
    :raises IndexError: if index out of range
```

[35]:
```
from sortedcontainers import SortedList
def aggiorna_k_massimi_SC(massimi, X, k):
    if len(massimi) < k:
        massimi.add(X)      # O(log(k))
    elif massimi[0] < X:    # O(log(k))
        del massimi[0]      # O(log(k))
        massimi.add(X)      # O(log(k))

# quindi il tempo nel caso peggiore è O(log(k))
```

[36]:
```
def k_massimi_SC_crescente(N, k):
    massimi = SortedList()
    for X in range(N):         # il caso peggiore è la sequenza crescente,␣
 ↪ripetuto N volte
        aggiorna_k_massimi_SC(massimi, X, k)        # O(log(k))
    return massimi

# quindi il tempo peggiore è O(log(k)*N)
```

[16]:
```
%timeit k_massimi_SC_crescente(1_000_000,    30)
%timeit k_massimi_SC_crescente(1_000_000,   300)
%timeit k_massimi_SC_crescente(1_000_000,  3000)
%timeit k_massimi_SC_crescente(1_000_000, 30000)
```

```
986 ms ± 7.37 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
1.04 s ± 6.88 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
1.11 s ± 8.5 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
1.13 s ± 4.28 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

[38]:
```
from math import log2
[ log2(X) for X in [30, 300, 3000, 30000]]
```

[38]:
```
[4.906890595608519, 8.228818690495881, 11.550746785383243, 14.872674880270605]
```

3 PAUSA

4 Tipi in Python

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

Questo perchè i tipi sono dinamici e gli oggetti riferiti dalle variabili "sanno" quali metodi possono essere applicati.

Questo vi fornisce ampia libertà con tutta una serie di controlli eseguiti a run-time.

Nei linguaggi compilati dobbiamo indicare i tipi e questo permette di spostare nella fase di compilazione molti controlli e rendere il codice compilato più veloce.

Esistono tool per controllare i tipi usati nel programma, i due più comuni sono: - **mypy** che fa una analisi statica del codice (ovvero SENZA eseguirlo) e deduce e controlla i tipi - **typeguard** a run-time (eseguendo il codice)

Aggiungere i tipi alle definizioni delle funzioni, variabili ed argomenti ci fa catturare molti più errori

Esistono inoltre dei compilatori di Python (ad es. il progetto **Codon** oppure **Cython**) che lo rendono più efficiente e veloce (al livelo del C/C++) se i tipi vengono indicati (per un sottoinsieme del linguaggio).

4.1 Come si aggiungono i tipi al codice

```
# definizione del tipo di una variabile nell'assegnamento
variabile : tipo = valore
# oppure come commento
variabile = valore # type : tipo

# definizione del tipo degli argomenti e del risultato di una funzione
def funzione( argomento1 : tipo1, ...) -> return_type :
    codice della funzione
# oppure come commento
def funzione( argomento1, ...):
    # type:(tipo1, ...) -> return_type
    codice della funzione
```

Queste "annotazioni" finiscono in un attributo dell'oggetto funzione e possono essere esaminate e controllate con `mypy` oppure con `typeguard`

4.2 Come si descrivono i tipi

- potete usare direttamente i nomi delle classi corrispondenti
  - **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 nel contenitore

esempio	descrizione
list	lista eterogenea
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**) nell'ordine
tuple[int, ...]	tupla di lunghezza variabile contenente solo **int**

4.3 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]
```

Si tendono ad usare nomi maiuscoli

4.4 Ci sono tipi speciali nel modulo `typing`

`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
- ...

[18]:
```
## Esempio di annotazioni di tipo
from typing import Sized  # i Sized hanno il metodo __len__ e se ne può␣
 ↪calcolare la dimensione

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

# i tipi sono inseriti nell'attributo __annotations__ dell'oggetto funzione
criterio.__annotations__
```

[18]:
```
{'elemento': typing.Sized, 'return': tuple[int, typing.Sized]}
```

[21]:
```
# provo a eseguire un test su sorted
def test_sorted():
    L        : list[Sized] = [ '932', '1', '23', '045', 2 ] # <== ATTENZIONE␣
 ↪intero!!!
    expected : list[Sized] = ['1', '2', '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)
Cell In[21], line 9
      6     assert result == expected
      8 if __name__ == '__main__':
----> 9     test_sorted()
     10 # scopriamo l'errore SOLO A RUNTIME!!!

Cell In[21], line 5, in test_sorted()
      3 L        : list[Sized] = [ '932', '1', '23', '045', 2 ] # <== ATTENZIONE
 ↪intero!!!
      4 expected : list[Sized] = ['1', '2', '23', '045', '932']
----> 5 result = sorted(L,key=criterio)
      6 assert result == expected

Cell In[18], line 8, in criterio(elemento)
      7 def criterio(elemento : Sized) -> tuple[int,Sized]:
----> 8         return len(elemento), elemento

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

4.5 come verificare i tipi staticamente? (senza eseguire il codice)

mypy --pretty sorted.py

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

[22]:
```
!mypy --pretty sorted.py
```

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

4.6 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

4.6.1 Run-time type-checking

Non si fa così: **python sorted.py**

(lo scopriamo all'ultimo momento, eseguendo **len**)

[24]:
```
!python sorted.py
```

```
Traceback (most recent call last):
  File "/Users/andrea/Documents/Uni/Didattica/Prog1/2023-
24/Lezioni/lezione08/sorted.py", line 12, in <module>
    test_sorted()
  File "/Users/andrea/Documents/Uni/Didattica/Prog1/2023-
24/Lezioni/lezione08/sorted.py", line 9, in test_sorted
    assert sorted(L, key=criterio_sort) == expected
  File "/Users/andrea/Documents/Uni/Didattica/Prog1/2023-
24/Lezioni/lezione08/sorted.py", line 4, in criterio_sort
    return len(elemento), elemento
TypeError: object of type 'int' has no len()
```

4.6.2 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

[25]:
```
!pytest sorted.py
```

```
============================= test session starts
==============================
platform darwin -- Python 3.10.12, pytest-7.4.2, pluggy-1.3.0
rootdir: /Users/andrea/Documents/Uni/Didattica/Prog1/2023-24/Lezioni/lezione08
plugins: hypothesis-6.88.1, anyio-4.0.0, timeout-2.2.0, json-0.4.0,
typeguard-4.1.5, profiling-1.7.0
collected 1 item


sorted.py F
[100%]

=================================== FAILURES ===================================
_______________________________ test_sorted
________________________________

    def test_sorted() -> None
:
        L        : list[Sized] = [
'932',
'1',
'23',
'045', 2
]
```

```
        expected : list[Sized] = [
'1',
'23',
'045',
'932' ]
>       assert sorted(L, key=criterio_sort) ==
expected

sorted.py:9:
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

elemento = 2

    def criterio_sort(elemento : Sized) ->
tuple[int, Sized]:
>           return len(elemento),
elemento
E           TypeError: object of type 'int' has no len()

sorted.py:4: TypeError
============================ short test summary info

============================
FAILED sorted.py::test_sorted - TypeError: object of type 'int'
has no len()
============================== 1 failed in 0.05s

==============================
```

4.6.3 Molto meglio: col modulo typeguard

Che si installa nell'Anaconda prompt col comendo

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

4.6.4 Si usa eseguendo in Anaconda prompt il comando

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

```
E              typeguard.TypeCheckError : argument "elemento" (int) is not an
instance of collections.abc.Sized
```

BENE: adesso **typeguard** si è accorto che **2** non è di tipo **Sized** quando ha controllato l'assegnamento

[ ]:
```
!pytest sorted.py --typeguard-packages=sorted
```

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

```python
@typechecked          # decoratore che controlla i tipi in ingresso e uscita
def criterio(el : Sized) -> tuple[int,Sized]:
    return len(el), el

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

4.7 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

4.8 Come usiamo i tipi negli HW obbligatori

- tutte le funzioni fornite sono annotate coi tipi
- uno dei test dello HW verifica i tipi a runtime (e che non li togliate)

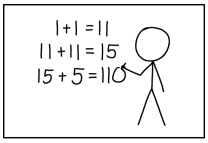
5 INTERVALLO

[30]:
```
%%HTML
<div align="middle">
<video width="80%" controls>
      <source src="Intervallo.webm" type="video/webm">
</video></div>
```

```
<IPython.core.display.HTML object>
```

6 Homework 2 (obbligatorio - AA 22-23)

6.1 dal blog XKCD di Randall Munroe



I+I=II II+II=IV
IV+V=IX

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

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

lettera	peso	lettera	peso	lettera	peso	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 di ciascuna lettera

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

Chiamiamo questo formato "formato XKCD" degli interi

6.2 Obiettivo dello HW

Bisognava implementare le 4 funzioni:

```
def decode_XKCD_tuple( xkcd_values : tuple[str, ...], k : int) -> list[int]:
    'decodifica una tupla di interi in formato XKCD e ne estrae i K massimi'
def decode_value( xkcd : str ) -> int:
    'decodifica un valore dal formato XKCD all\'intero corrispondente'
def xkcd_to_list_of_weights( xkcd : str) -> list[int]:
    'trasforma un valore in formato XKCD nella lista di pesi interi'
def list_of_weights_to_number( weigths : list[int] ) -> int:
    'data una lista di pesi ottiene il numero intero corrispondente'
```

E poi estrarre da una lista di stringhe in "formato XKCD" i **k** valori massimi. - convertendo ciascuna stringa nella lista di pesi - convertendo ciascuna lista di pesi nel valore intero

6.2.1 Perchè questo HW era così semplice?

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

Gli HW seguenti NON vengono specificati a questo livello di dettaglio (l'analisi del problema è a carico vostro)

6.2.2 Realizziamolo

- **decode_XKCD_tuple**: per decodificare la tupla e ottenere i k massimi
  - trasformiamo ciascuna stringa nell'intero e gestiamo i k massimi come si è visto
- **decode_value**: per trasformare una stringa in un intero
  - trasformiamo la stringa in una lista di pesi
  - otteniamo l'intero dalla lista di pesi
- **xkcd_to_list_of_weights**: per trasformare la stringa in una lista di pesi
  - scandiamo la stringa cercando i pesi in ordine di lunghezza decrescente (da '1000' a '1')
  - oppure osserviamo che ciascun peso inizia con 5 o con 1

Esempio: 1961: '1000100100050101' -> [1000, 100, 1000, 50, 1]

- **list_of_weights_to_number**: per trasformare la lista di pesi in un intero
  - applichiamo le regole dei numeri romani:
  - ovvero se un peso è seguito da un peso maggiore (es. 9 = IX)
    - va sottratto
    - altrimenti va sommato

Esempio: [1000, 100, 1000, 50, 1] -> (1000 - 100 + 1000 + 50 + 1) = 1961

[39]:
```
# carico un modulo che applica mypy alle celle di questo notebook
%load_ext nb_mypy
# attivo la verifica ad ogni salvataggio
%nb_mypy On
```

```
Version 1.0.5
```

[47]:
```
def decode_XKCD_tuple( xkcd_values : tuple[str, ...], k : int) -> list[int]:
    'decodifica una tupla di interi in formato XKCD e ne estrae i K massimi'
    interi = [ decode_value(X) for X in xkcd_values ]
    return sorted(interi, reverse=True)[:k]    # semplice ma non la più veloce
# Esempio
decode_XKCD_tuple(('1000100100050101','1101000','150'), 6)
```

[47]:
```
[1961, 989, 49]
```

[44]:
```
def decode_value( xkcd : str ) -> int:
    'decodifica un valore dal formato XKCD all\'intero corrispondente'
    pesi = xkcd_to_list_of_weights(xkcd)
    return list_of_weights_to_number(pesi)
```

[43]:
```
def xkcd_to_list_of_weights( xkcd : str) -> list[int]:
    '''trasforma un valore in formato XKCD nella lista di pesi interi'''
    pesi : list[str] = ['1000','500','100','50','10','5','1']
    risultato : list[int] = []
    while xkcd:
        for peso in pesi:
            if xkcd.startswith(peso):        # se i primi caratteri␣
 ↪corrispondono
                risultato.append(int(peso))  # aggiungo il peso al risultato␣
 ↪come intero
                xkcd = xkcd[len(peso):]      # accorcio la stringa togliendo la␣
 ↪parte trovata
                break                        # smetto il for e continuo il while
    return risultato
# Esempio
xkcd_to_list_of_weights('1000100100050101')
```

[43]:
```
[1000, 100, 1000, 50, 10, 1]
```

[48]:
```
# altrimenti se notiamo che tutti i pesi iniziano per 1 oppure per 5
# inserisco spazio prima di 1 e di 5 e uso split
def xkcd_to_list_of_weights( xkcd : str) -> list[int]:
    '''trasforma un valore in formato XKCD nella lista di pesi interi'''
    spaziato : str = xkcd.replace('1', ' 1').replace('5', ' 5') # inserisco␣
 ↪spazio prima di 1 e 5
```

```
    return [ int(X) for X in spaziato.split() ]  # e spezzo la stringa sugli␣
 ↪spazi
# Esempio
xkcd_to_list_of_weights('1000100100050101')
```

[48]: [1000, 100, 1000, 50, 10, 1]

[41]:
```
def list_of_weights_to_number( weigths : list[int] ) -> int:
    'data una lista di pesi ottiene il numero intero corrispondente'
    somma = 0
    for i in range(len(weigths)-1): # per ciascun valore fino al penultimo
        if weigths[i]<weigths[i+1]: # se è *seguito* da un valore maggiore
            somma -= weigths[i]     # va sottratto
        else:                       # altrimenti
            somma += weigths[i]     # va sommato
    return somma + weigths[-1]      # aggiungo l'ultimo
# Esempio
list_of_weights_to_number([1000, 100, 1000, 50, 10, 1])
```

[41]: 1961

[49]:
```
# oppure sfrutto zip per scandire i pesi e i pesi sfalzati di 1 posto
def list_of_weights_to_number( weigths : list[int] ) -> int:
    'implementazione usando una list comprehension e zip'
    return sum( -X if X<Y else X              # sommo X oppure lo sottraggo se ␣
 ↪minore del seguente
                for X,Y in zip(weigths,             # scandendo in parallelo␣
 ↪i pesi
                               weigths[1:]+[0]) )   # e i pesi sfalzati di 1␣
 ↪e con uno 0 in fondo
list_of_weights_to_number([1000, 100, 1000, 50, 10, 1])
```

[49]: 1961

7 PAUSA

8 Ancora analisi di problemi : istogramma

Dato un elenco di dati vogliamo calcolarne le frequenze (quante volte appare ciascun valore) e produrre una stampa in cui visualizziamo le frequenze su righe successive, come **barre di asterischi**

Esempio: da [1, 4, 2, 4, 1, 4, 4, 2, 4, 4, 1, 1, 4] otteniamo

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

8.0.1 Problema: stampare l'istogramma di una serie di dati

Input: - la lista di dati numerici (interi/float?) - la definizione degli intervalli (implicita/esplicita?)

Output: - stringa formata da più righe, una per ogni valore: - ciascuna riga contiene il **valore, spazio** e tanti **asterischi** quante volte quel valore appare nei dati

Da decidere: i valori con conteggio 0 devono apparire? '''

8.0.2 Istogramma con estremi e dati interi

[32]:
```
'''
# per ottenere l'istogramma conoscendo gli estremi 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 possibili [A, B]
            # ciascun 'bin' è largo 1
    # dalle frequenze produco la stringa
'''
def istogramma_con_estremi_e_dati_interi(A : int,
                                         B : int,
                                         dati : list[int]) -> str:
    frequenze : list[int] = calcola_frequenze(A, B, dati)
    return produci_istogramma(A, frequenze)
```

[33]:
```
# per calcolare le frequenze come lista di conteggi
def calcola_frequenze(A : int, B : int, dati : list[int]) -> list[int] :
    'attenzione agli indici della lista per dati positivi e negativi!'
    # usiamo un offset
    conteggi : list[int] = [0] *(B-A+1)      # devo contenere tutti i valori da␣
 ↪A a B COMPRESI
    for valore in dati:
        if A <= valore <= B: # controllo che il valore sia in [A,B]
            conteggi[valore-A] += 1          # sottraggo A per allinearlo␣
 ↪all'indice 0
    return conteggi
```

[ ]:
```
def produci_istogramma(A : int, frequenze : list[int]) -> str :
    testo = ''
    for indice, frequenza in enumerate(frequenze):
        valore     = indice + A          # devo riaggiungere A all'indice
        asterischi = "*"*frequenza
        testo += f"{valore:>4}\t{asterischi}\n"
    return testo

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

8.0.3 Istogramma con estremi e dati float

- in questo caso posso troncare i valori
- oppure li potrei arrotondare all'intero più vicino con **round**

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

def calcola_frequenze_float(A : int, B : int,
                            dati : list[float]) -> list[int]:
    dati_interi = [ int(X) for X in dati ]          # tronco i valori
    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))
```

8.0.4 Istogramma usando dizionari

Se i dati sono sparsi e ci sono molte frequenze nulle posso usare meno memoria

[ ]:
```
# se vogliamo usare meno memoria possiamo usare un dizionario
# O(n)
def calcola_frequenze_con_diz_int(dati : list[int]) -> dict[int, int] :
    frequenze = {}
    for valore in dati:            # N volte
        if valore in frequenze:
            frequenze[valore] += 1
        else:
            frequenze[valore]  = 1
    return frequenze

# Esempio
from random import choices
L = choices(range(-10,10), k=30)
istogramma = calcola_frequenze_con_diz_int(L)
print(L)
print(istogramma)
```

[ ]:
```
# O(n^2)
def calcola_frequenze_con_diz_int2(dati : list[int]) -> dict[int, int] :
    frequenze = {}
    for valore in dati:                          # N volte
        frequenze[valore] = dati.count(valore)   # O(N)
    return frequenze
    # oppure con una list-comprehension
```

```
    '''return { dati.count(valore) for valore in dati }'''
```

[ ]:
```
# ricordiamoci di ordinare le chiavi
def produci_istogramma_diz_con_buchi(istogramma : dict[int,int]) -> str :
    # ignoro i valori con conteggio 0
    testo = ''
    for valore, frequenza in sorted(istogramma.items()):
        testo += f"{valore}\t" + ('*'*frequenza) + '\n'
    return testo

# Esempio
print(produci_istogramma_diz_con_buchi(calcola_frequenze_con_diz_int2(L)))
```

[ ]:
```
# per visualizzare anche i valori con frequenza 0 li posso aggiungere al␣
 ↪dizionario
def produci_istogramma_diz_senza_buchi(istogramma : dict[int,int]) -> str :
    # generare anche i dati con conteggio 0
    # aggiungiamo al dizionario i valori con conteggio 0
    m = min(istogramma)     # minima chiave
    M = max(istogramma)     # massima chiave
    for X in range(m,M+1):
        if not X in istogramma:
            istogramma[X] = 0
    return produci_istogramma_diz_con_buchi(istogramma)

# Esempio
print(produci_istogramma_diz_senza_buchi(calcola_frequenze_con_diz_int2(L)))
```

[ ]:
```
# oppure ne posso tenere conto nella generazione del testo
# come frequenza = 0 se la chiave non è presente
def produci_istogramma_diz_senza_buchi2(istogramma : dict[int,int]) -> str :
    m = min(istogramma)     # minima chiave
    M = max(istogramma)     # massima chiave
    testo = ''
    for X in range(m,M+1):
        frequenza = istogramma.get(X, 0)     # se X non è nel dizionario torna 0
        asterischi = '*'*frequenza
        testo += f"{X:>4}\t{asterischi}\n"
    return testo

print(produci_istogramma_diz_senza_buchi2(istogramma))
```