

Algorithms and Programming in Python for Data Science

Dario Comanducci

I. INTRODUZIONE

Il compito richiede l'analisi di un file di testo su degli studenti e docenti di una scuola (che verranno indicati come individui o soggetti) dove ogni riga $t \ i \ j \ r \ s$ indica che l'individuo i della categoria r ha incontrato l'individuo j della categoria s nell'intervallo di tempo $[t-20, t]$, espresso in s (le categorie, o classi di affiliazione, in totale sono '1A', '1B', ..., '5B', 'Teachers'); su tale dataset meetings¹ sono state sviluppate un'animazione dei primi N incontri (con Turtles), e delle funzioni per le domande successive.²

II. ANIMAZIONE

L'animazione delle interazioni è racchiusa nella classe `Viewer`: il suo costruttore accetta le categorie, racchiuse nella tupla `categories`, e il dataframe `meetings` al fine di istanziare e posizionare le tartarughe per ciascun soggetto: in particolare, il metodo `create_Layout()` all'interno del costruttore estrae tramite pandas da `meetings` un dataframe `school` con colonne date dagli id degli individui e dalla classe di appartenenza di ciascun soggetto, al netto di duplicati; tramite `school.create_Layout()` associa al soggetto `id_i` una tartaruga che dispone sull'area del filmato assegnata alla relativa classe `cat_i` con relativo colore (Fig. 1a).

Con `play()` vengono poi scorse le prime N righe³ di `meetings` spostando per ogni incontro la tartaruga interessata: Fig. 1b mostra il caso per N pari a tutte le righe; al termine, la finestra di `Turtles` viene chiusa cliccando sulla sua 'X'.

III. DOMANDE

Per ogni domanda è stata implementata una funzione ad hoc: tutte le funzioni ricevono in ingresso il dataframe `meetings` e arrivano al risultato in tempi lineari $O(n)$ rispetto al numero n di righe di `meetings`. Tab. I riassume i risultati prodotti da ciascuna funzione e la relativa complessità computazionale.

Sebbene varie domande abbiano schema simile, le funzioni sono separate per semplicità. In particolare, le domande 1–5 chiedono le "entità" (soggetti o categorie, o loro coppie) che massimizzano il numero, casomai condizionato, di incontri:

- è quindi previsto un set `argmax` contenente le entità che hanno prodotto il valore di riferimento contenuto nella variabile `maxValue` come stima corrente del numero massimo di incontri, assieme ad un dizionario `counts` con chiavi (entità, contatore) per gli incontri conteggiati;
- ogni volta che un'entità e produce in `counts[e]` il valore di `maxValue`, essa viene aggiunta al set `argmax`;

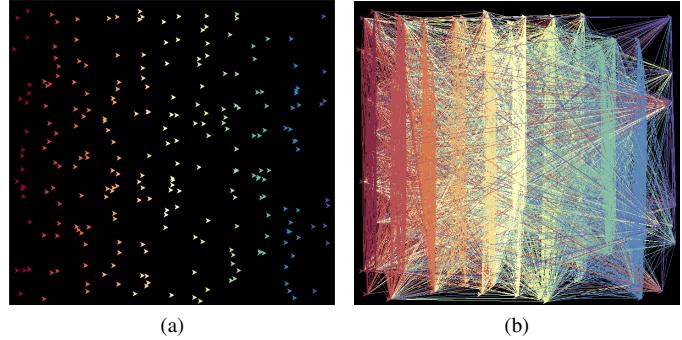


Figura 1: Layout iniziale dell'animazione (a) e suo completamento su tutte le interazioni (b).

- se il valore raggiunto in `counts[e]` supera la stima in `maxValue`, l'algoritmo resetta `argmax` con la sola entità e , aggiornando `maxValue = counts[e]`.

Il meccanismo si ripete ad ogni riga di `meetings`: le domande si differenziano nella natura di e e nella condizione da applicare per decidere d'incrementare `counts[e]`.⁴

A. Domanda 1

La funzione `answer1()` deve indicare l'individuo con più interazioni, perciò cicla su entrambi i soggetti di ogni riga: ciascuno di essi a ruota costituisce l'entità e , incrementando il suo contatore associato `counts[e]` ad ogni incontro di e .

Il costo complessivo vale in media $O(n)$, in quanto scandendo le n righe di `meetings` sono eseguite istruzioni $O(1)$ per ogni riga (in particolare l'operazione più "critica", la verifica di una chiave presente in `counts`, vale in media $O(1)$) [1].

B. Domanda 2

Qui l'entità e è l'individuo che ha interagito di più con soggetti diversi, contando solo il primo incontro di una persona con le altre: per annotare gli incontri di e , in `answer2()` è stato necessario anche un dizionario `meets`, con coppia chiave-valore data da e e dal set di persone viste da e : per ogni riga `counts[i]` del soggetto i non è incrementato se ha già incontrato la persona j (j in `meets[i]`).⁵

Come nel caso precedente, la complessità computazionale vale $O(n)$: l'ulteriore verifica di appartenenza da parte di j al set in `meets[i]` costa anch'esso $O(1)$ in media [1].

⁴Il ciclo `for _, row in meetings.iterrows()` scorre le righe di `meetings` sull'iteratore `iterrows`, così da accedere ai dati in ogni riga attraverso le varie colonne (ad esempio `row.i`); non essendo d'interesse, l'indice numerico della riga è rimpiazzato col segnaposto `_`.

⁵L'algoritmo cicla sugli individui i e j coinvolti, scambiandoli nel "ruolo" i e nella riga `for p,q in [(row.i, row.j), (row.j, row.i)]`.

¹Caricato come dataframe con `pd.readTable()` da file in locale.

²Codice sviluppato con Python 3.11.9.

³N immesso a runtime dall'utente.

Tabella I: Funzioni per le domande 1...7, con costo computazionale rispetto al numero n di righe in `primaryschool.tsv`.

funzione	risultato	costo	analisi	funzione	risultato	costo	analisi
<code>answer1()</code>	$\arg \max(2594) = 1695$	$O(n)$	c. medio	<code>answer5()</code>	$\arg \max(368) = (3A, 3B)$	$O(n)$	c. medio
<code>answer2()</code>	$\arg \max(134) = 1551$	$O(n)$	c. medio	<code>answer6()</code>	Fig. 2	$O(n)$	c. medio
<code>answer3()</code>	$\arg \max(11) = 105$ individui	$O(n)$	c. medio	<code>answer7()</code>	$\mathbb{E}[\text{atRisk}()] = 240.31 (*)$	$O(n(T+1))$	c. medio
<code>answer4()</code>	$\arg \max(764) = (1437, 1563)$	$O(n)$	c. medio	$*: T=100 \text{ trials con reimbussolamento, } t0=0$			

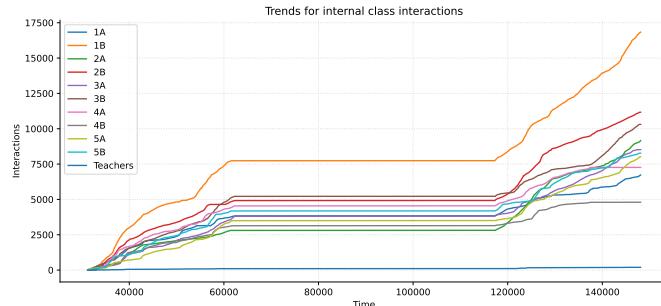


Figura 2: Interazioni nel tempo all'interno di ciascuna classe.

C. Domanda 3

Per il soggetto con più interazioni con classi diverse, `answer3()` ha lo stesso schema di `answer2()`: in `for p,q in [(row.i, row.j), (row.j, row.i)]` nel secondo membro delle tuple i soggetti cambiano con le categorie (`for p,cq in [(row.i, row.s), (row.j, row.r)]`). Essendo la stessa impostazione di `answer2()`, anche qui la complessità media è $O(n)$.

D. Domanda 4

Per la coppia di soggetti con più incontri, `answer4()` usa la tupla ordinata `pair = (min(row.i, row.j), max(row.i, row.j))`, ricalcando `answer1()`, come chiave nel dizionario `counts`: il costo medio è quindi $O(n)$.

E. Domanda 5

Stavolta l'entità è la coppia di classi con più interazioni: `answer5()` impiega sia la tupla ordinata `pair_p` delle persone (come in `answer4()`) che la tupla ordinata delle classi `pair_c = (min(row.r, row.s), max(row.r, row.s))`: ogni volta che due soggetti di classi diverse s'incontrano si verifica se la coppia ordinata delle due persone non sia presente nel set degli incontri per la coppia di classi in esame (`meets[pair_c]`) affinché si incrementi il contatore `counts[pair_c]`. Viceversa, quando è il primo incontro tra individui di classi diverse, si procede all'inizializzazione del contatore `counts[pair_c]=1` e delle persone che si sono incontrate (`meets[pair_c] = {pair_p}`).

Il costo di `answer5()` è $O(n)$, dovendo ciclare n volte su istruzioni con costo $O(1)$ (costo medio, [1]).

F. Domanda 6

Fig. 2 riporta il grafico di `answer6()`: essendo il dataset già ordinato in slot successivi da 20 s, basta tenere un contatore parziale degli incontri `counts` (un dizionario con le classi

come chiavi) resettato in ogni slot, ed aggiornare il totale degli incontri interni al cambio del tempo (`tLast != tCur`), resettando poi `counts[cat]=0`; in ogni riga il contatore in `counts` è aggiornato solo per soggetti della stessa classe.⁶

Il costo medio è $O(n)$: le operazioni in lettura ai dizionari e di append alle liste hanno infatti costo medio $O(1)$ [1].

G. Domanda 7

Per contare i soggetti a rischio in caso d'influenza, `answer7()` impiega la funzione `atRisk(patient0, t0)` per sapere quante persone lo diventano a partire dal paziente zero `patient0`, contagioso dal tempo `t0`, campionato dalla lista `people` degli individui nella scuola.⁷

Per `atRisk(patient0, t0)`, si crea un dizionario `contactTime`, inizializzato dal paziente zero al tempo `t0`, che riporta il tempo d'incontro per ogni persona esposta in contatti a rischio.

Ad ogni riga `row` di `meetings` si valuta prima per il soggetto `row.i` e poi `row.j` se sono già presenti in `contactTime` (ossia se sono già stati esposti) e, nel caso, se è già passato il tempo d'incubazione rispetto al tempo attuale d'incontro: a questo punto, se l'altra persona non è stata ancora esposta ad un contatto a rischio, viene anch'essa aggiunta a `contactTime` con tempo `row.t-tSlot` (`tSlot = 20 s`) ed accodata nella lista `riskyPeople`. La persona viene aggiunta subito alla lista delle persone a rischio, pur non essendo passato il tempo di incubazione, perché da quel momento è possibile valutare dallo scarto temporale con il suo tempo di contatto la sua capacità di contagiare in funzione del tempo d'incubazione `tInc=1`: che stia incubando o sia potenzialmente contagiosa, è comunque una persona a rischio.

Ripetendo in T trials il ciclo per n righe su operazioni con costo $O(1)$ (alcune in media [1]), la complessità vale $O(Tn)$.

Su 100 trials casuali la media stimata è stata pari a 240.31, ipotizzando che il paziente zero fosse contagioso da subito (Tab. I). Per una stima "deterministica", possiamo valutare il numero di contagiati causato come paziente zero da ciascuna persona in `people` dall'inizio delle interazioni, trovando che il numero di persone a rischio presenta le seguenti frequenze: a) 137 casi per 241 soggetti a rischio; b) 99 casi per 240; c) in 4 casi, 237; d) solo 2 casi per 227: la media pesata porta a $241\frac{137}{242} + 240\frac{99}{242} + 237\frac{4}{242} + 227\frac{2}{242} = 58179/242 = 240.409$.

RIFERIMENTI BIBLIOGRAFICI

[1] <https://wiki.python.org/moin/TimeComplexity>

⁶Per il totale, `totCounts[cat]` è una lista che cresce nel tempo per la chiave `cat`; il tempo `tLast` è invece accodato alla lista `time`. Qualora i docenti non fossero da valutare, basta evitare di aggiungere la loro categoria in `counts` e `totCounts`, senza ledere la complessità lineare dell'algoritmo.

⁷Serve un giro su `meetings` per popolare `people` (costo medio $O(n)$).