Relazione progetto di Algoritmi "Publish-or-perish", gennaio 2023

Alyssa Pezzutti

Gennaio 2023

1 Il problema

Si vuole implementare un algoritmo che gestisca e riordini una sequenza di articoli e autori, e ne calcoli specifici indicatori e sottoinsiemi che descriveremo in dettaglio man mano che ne affrontiamo l'implementazione. Per ora, a grandi linee dobbiamo calcolare:
- il numero di pubblicazioni di un autore;
- il numero di volte che un articolo è stato citato;
- l'*indice di Hirsch*;
- posti gli autori in un grafo di collaborazione, il *diametro* e il *clique number*.

Vorremo poi stampare a video:
- la lista di autori ordinata;
- la lista di autori piu' lontani tra loro nel grafo di collaborazione;
- le cliques massimali.

Avremo quindi bisogno di strutture dati dinamiche, che permettano di calcolare e tenere in memoria diversi tipi di dati, e soprattutto che favoriscano il riordino di elementi secondo vari criteri.

2 L'algoritmo

2.1 Caricamento articoli

Innanzitutto il programma riceve in ingresso una sequenza formattata di articoli e i loro autori. Poiché il numero di articoli è indicato nel file in ingresso, possiamo inserirli in un vettore di elementi di tipo `articolo_st`, dove `articolo_st` è una struttura contenente:
- identificatore dell'articolo
- titolo
- lista di autori (passati per puntatore)
- vettore con le citazioni

- numero di citazioni

Per ciascun articolo in input, il programma ne esegue il parsing: divide la stringa letta in sottostringhe (avendo cura di rimuovere gli spazi con una funzione apposita) e contemporaneamente inserisce gli autori in una lista lineare, bidirezionale, senza sentinella. Scegliamo una struttura dinamica di questo tipo perché a priori non posso conoscere il numero di autori, e questo ci permette di aggiungerli e riordinarli semplicemente spostando dei puntatori. Inoltre risulta comoda per le richieste seguenti, dove il programma dovrà gestire tutto l'insieme di autori in una volta, e qualunque tipo di elenco verrà scorso sempre da capo.

Il programma ha quindi ha disposizione un vettore di articoli e una lista di autori. Ciascun autore è a sua volta una struttura, i cui campi contengono le informazioni che useremo in seguito, ovvero:
- articoli scritti (e il loro numero)
- numero di citazioni ricevute
- collaboratori (inseriti a loro volta in una lista)

Come detto, leggendo il file una riga alla volta riempiamo sia il vettore dei titoli che la lista degli autori. Man mano che incontriamo un autore il programma verifica se è già presente nella lista (operazione che richiede di scorrere la lista da capo ogni volta). In caso affermativo deve solo aggiornarne i campi, altrimenti crea un nuovo autore. In questo modo salva subito in memoria tutte le informazioni presenti nel file per evitare di doverlo rileggere piu' volte. Ogni riferimento successivo a questi elementi (articoli e autori) avviene per indirizzo, cosicché gli elementi sono presenti in memoria una volta sola.

Costo Posti n = numero articoli, m_i = numero autori dell'articolo i-esimo, m = numero autori totale, k_i = numero citazioni ricevute dall'articolo i.
Per ciascun articolo il programma deve scorrere la lista complessiva di autori per verificare se l'autore è già presente ed eventualmente inserirlo. Poichè i titoli si trovano in un vettore il cionteggio delle citazioni avviene in tempo costante.

$T \in \Theta(m \sum_{i=1}^{n} m_i) + \Theta(1)$
nel caso migliore $m_i = 1 \forall i$, mentre nel caso peggiore $m_i = m \ \forall i$, quindi

$\Theta(nm) \leqslant T \leqslant \Theta(nm^2)$

$S \in \Theta(n) + \Theta(m)$

2.1.1 Estrapolazione di autori e citazioni

Il programma deve adesso scorrere nuovamente la lista di autori per riempire le informazioni mancanti: fin'ora abbiamo solo creato la lista autori, ma non riempito tutti i campi di ciascuno. Deve quindi:
- cercare e inserire tutti i collaboratori

- contare quante volte un autore è stato citato
- calcolarne l'indice di Hirsch

Riempimento lista collaboratori Ogni struttura autore contiene un campo con puntatori agli articoli che ha scritto, dunque il programma scorre la lista degli autori, e per ogni autore scorre la lista dei suoi articoli. Per ciascun articolo:
- somma il numero di citazioni ricevute
- scorre la lista dei collaboratori e inserisce ciascun collaboratore nella lista complessiva (già in ordine alfabetico)

L'inserimento ordinato avviene sempre al momento, in modo simile all'InsertionSort, che funziona bene per liste con pochi elementi come queste. Osserviamo che si sarebbe anche potuto porre i collaboratori in un vettore di adiacenza anzichè una lista, cioè un vettore di lunghezza pari al numero di autori con un indice che marcasse se un autore è collaboratore o meno. Sarebbe piu' efficiente di una lista, anche per le operazioni da svolgere successivamente, ma continuiamo a usare una lista per simmetria con quella degli autori già esistente (visto che questi sono a loro volta insiemi di autori).

Costo Siano n_j il numero di titoli scritti dal j-esimo autore ($1 \leqslant j \leqslant n$), e m_i il numero di autori dell'articolo i-esimo ($1 \leqslant i \leqslant m$).
Nel caso migliore, $n_j = m_i = 1$, mentre nel caso peggiore $n_j = n, m_i = m$, quindi:

$$\Theta(m) \leqslant T \leqslant \Theta(nm^2)$$

$$\Theta(1) \leqslant S \leqslant \Theta(nm)$$

Calcolo indice di Hirsch L'indice di Hirsch si calcola cosi':
1. si ordinano gli articoli dell'autore per numero di citazioni non crescente;
2. si assegna ad ogni articolo come indice il numero d'ordine, partendo dall'indice
3. si scorrono gli articoli nell'ordine, fermandosi all'ultimo articolo che ha indice non superiore al proprio numero di citazioni;
4. si restituisce l'indice a cui si è arrestato lo scorrimento.

Nota sull'ordinamento: Per ordinare gli articoli è stato scelto QuickSort in quanto è l'algoritmo migliore nel caso medio. Nel complesso pero', QuickSort verrà usato in due istanze diverse, qui per ordinare articoli e successivamente per ordinare autori. I due elementi hanno tipi diversi, dunque per evitare di scrivere due volte le funzioni QuickSort, Partition, Scambia (identiche a meno del tipo del vettore di puntatori passato in ingresso) ci si ispira alla funzione `qsort()` del C per poter passare dei puntatori a void alle funzioni e assegnare i tipi in

seguito. Scriviamo pero' due funzioni di confronto, visto che i criteri diversi per ordinare gli elementi sono diversi, per poi passarle a QuickSort tramite puntatore a funzione.

Costo $T \in \Theta(nlogn) + \Theta(n)$

$S \in \Theta(1)$

Quindi per questa sezione il costo complessivo è:
$\Theta(m) + \Theta(nlogn) + \Theta(n) \leqslant T \leqslant \Theta(nm^2) + \Theta(nlogn) + \Theta(n)$

$\Theta(1) \leqslant S \leqslant \Theta(nm)$

2.2 Ordinamento autori

Il programma deve adesso ordinare gli autori. Nell'ordine, i criteri da seguire sono:
- numero di articoli non crescente
- numero di citazioni non crescente
- ordine alfabetico

Di nuovo, il programma usa QuickSort per l'ordinamento. Poichè essa opera su vettori, bisogna innanzitutto copiare la lista in un vettore per poi copiare tutto nuovamente nella lista, operazioni che si possono eseguire in tempo lineare. è un'operazione che aumenta il costo in tempo e spazio (e che putroppo rifaremo in seguito) che pero' facciamo per mantenere sempre gli autori in una lista anzichè cambiare struttura in mezzo al programma.

Costo $T \in \Theta(mlogm)$

$S \in \Theta(m)$

2.3 Grafo di collaborazione

La prossima operazione da eseguire è il calcolo del diametro del grafo di collaborazione. Di fatto il programma non crea un grafo vero e proprio, ma tratta le liste di collaboratori di ciascun autore come liste di forward star, dato che per ogni autore indicano proprio i nodi adiacenti.

Come sopra, il programma copia la lista di autori in un vettore prima di iniziare per poter utilizzare funzioni viste durante il corso, modificate quanto basta. Il programma alloca inoltre una matrice di adiacenza per semplificare la ricerca e la stampa delle coppie di autori piu' distanti tra loro.

2.3.1 Diametro

Per il calcolo del diametro si sfrutta l'algoritmo di visita in ampiezza: scorrendo la lista autori, fissa l'autore corrente come nodo sorgente e visita tutti gli altri a partire da esso, per poi risalire ogni cammino e calcolarne la lunghezza. Quindi per ogni autore l'algoritmo deve calcolarne la distanza da tutti gli altri, salvarla in un vettore e tenere traccia del massimo. Inoltre, se il grafo è sconnesso il diametro è pari al numero di autori, e le coppie di autori da stampare sono quelle non connesse tra di loro.

Osserviamo che poichè l'implementazione della coda fa uso di un vettore, questo va allocato a priori anche se non possiamo sapere quanti elementi dovremo effettivamente inserirvi, occupando spazio che probabilmente non verrà poi utilizzato. Per i test case forniti per questo progetto una coda di 4096 elementi sembra essere sufficientemente lunga, ma se venissero usati esempi che richiedessero code piu' lunghe, il programma fallirebbe.

Complessivamente l'algoritmo esegue un ciclo sugli autori, nel quale:
- azzera il vettore delle distanze
- esegue l'algoritmo BFS, il quale crea e gestisce una coda di autori durante la visita per poi ripercorrere a ritroso il cammino minimo da ogni nodo alla sorgente, incrementando un contatore per la distanza a ogni passo.

L'algoritmo BFS esegue la visita percorrendo il cammino minimo tra la sorgente e ciascun nodo, e, poichè il grafo non è pesato, risulta quindi adatto ai nostri scopi. Calcolate tutte le distanze si cicla di nuovo su di esse per trovarne il massimo.

Per poi risalire alle coppie di autori piu' distanti tra loro il programma sfrutta la stessa struttura dati usata piu' avanti nel calcolo delle clique. Per ora si accenna solo che ciascuna clique è semplicemente una lista di autori, percio' l' elenco di coppie sarà di fatto una lista di liste di due autori ciascuna. Essendo richiesta una stampa ordinata, gli autori che si trovano a distanza pari al diametro vengono inseriti già in ordine alfabetico.

Poichè la matrice è simmetrica il programma ne scorre solo il triangolo superiore per verificare se la distanza è maggiore del diametro. Poste tutte le coppie in una lista, il programma le stampa.

Costo $T \in \Theta(m^2 + m\sum_{j=1}^{m} m_j))$

$S \in \Theta(m^2 + m\sum_{j=1}^{m} m_j))$

2.4 Cliques

Una clique è definita come un insieme di autori che collaborano tutti tra di loro, percio' decidiamo di rappresentarle con delle liste (lineari, bidirezionali senza

sentinella) di liste di autori, corredate di apposita libreria per gestirle (creata modificando la libreria per le liste fornita a lezione.)

Quindi, il programma deve ora:
- calcolare le clique massimali
- calcolarne la cardinalità
- calcolare il numero di sottoinsiemi di tale cardinalità
- stampare il tutto in modo ordinato.

Come suggerito nella traccia, l'algoritmo per il calcolo delle clique massimali si basa sull'utilizzo di una coda.
Inizialmente la coda contiene clique di singoli autori. Finchè la coda non è vuota, si toglie il primo elemento e si verifica se è possibile aggiungere un nuovo collaboratore: in caso affermativo si crea una nuova clique che lo includa e la si rimette in coda, altrimenti la clique è massimale, e la si salva a parte.

Per evitare duplicati si procede in ordine alfabetico, cercando gli eventuali nuovi collaboratori solo tra quelli comuni e successivi in ordine alfabetico (per esempio, se ho già trovato la clique [A1 A2] evito di creare la clique [A2 A1].)

Implementiamo quindi l'algoritmo attraverso funzioni che gestiscano code di clique, ottenute adattando la libreria `intqueue`.

Per ogni elemento della coda il programma deve:
- scorrere gli autori nella clique
- scorrere i loro collaboratori e controllare se possono essere aggiunti
- eventualmente inserirli in ordine alfabetico
- se la clique è massimale inserirla (in ordine) in una lista a parte

Costo Sia q = numero di clique massimali.

$T \in \Theta(m)[\sum_{j=1}^{m}(\sum_{k=1}^{j}\Theta(k^2))+\Theta(j^2)]$, quindi $T \in \Theta(m^5)$ nel caso peggiore

$S \in \Theta(q)$

2.4.1 Clique number, sottoinsiemi

è adesso richiesto il conteggio del clique number, cioè la cardinalità massima delle clique, e il numero di clique con tale numero di autori.

Il programma deve quindi contare il numero di elementi di tutte le clique massimali: la cardinalità della prima clique della lista è proprio il clique number (visto che le clique sono state inserite in modo ordinato). Scorrendo poi le clique massimali, si conta il loro numero di autori e lo si confronta con il clique number. Se corrispondono tale clique viene stampata a video.

Costo Sia q = numero di clique massimali, e q_j il numero di autori della clique j-esima.

$T \in \Theta(m) + \Theta(q \sum_{j=1}^{q} q_j)$ perché si ha un ciclo sulla lista di clique e per ciascun elemento un ciclo di conteggio.

$S \in \Theta(q)$

2.5 Costo complessivo

$T \in \Theta[m^5 + m^3 + m^2 + nm^2 + nlogn + mlogm + n + m + q^3]$

$S \in \Theta[m^3 + m^2 + nm + n + m + q]$

2.6 Note conclusive

Il programma poteva certamente essere svolto in modo piu' efficiente, soprattutto in termini di tempo. Purtroppo mi sono accorta di cio' solo quando era ormai troppo tardi per modificare l'implementazione, e di questo mi scuso. Nella speranza che sia comunque sufficiente, ringrazio dell'attenzione e porgo cordiali saluti.