Laboratorio di Algoritmi - Progetto "Bibliometria"

Žana Ilić - 898373

20 Gennaio 2020

Introduzione

Supponiamo che abbiamo un insieme degli scienziati che, naturalmente, scrivono tanti pubblicazioni. Qualche volte le scrivono da solo, qualche volte collaborano con una o più persone. Certo, uno scienziato che ha scritto più articoli ha produttività scientifica magiore di uno che ha scritto meno articoli. Ma questo non è tutto. Dobbiamo anche guardare l'aspetto qualitativo, non solo quantitativo. Se qualche articolo è di buona qualità, altri scienziati lo prenderanno in considerazione nelle loro ricerche, cioè quel'articolo viene citato negli altri articoli. Allora un articolo più citato è più significativo di uno meno citato. Ora, per facilità, scriviamo autore invece di scienzato. Con tutte queste informazioni, possiamo calcolare facilmente il numero totale di articoli scritti da un autore e numero di citazioni totali - numero di volte in cui un suo articolo qualsiasi è citato in qualsiasi altro articolo (non è importante quali sono gli autori di quest'ultimo). Nel nostro caso, vogliamo anche calcolare indice di Hirsch.

Implementazione

Lettura da un file e la struttura `Article`

Iniziamo l'implementazione. Leggiamo il file dal linea di commando.
Durante la lettura del file usiamo la struttura `Article`, che contiene tutte le informazioni di un articolo, i cui attributi sono: numero di identificazione di articolo (`id`), titolo di articolo (`name`), vettore con i nomi degli tutti gli autori di articolo (`autors`) e il numero corispondente (`numberOfAutors`), vettore con i numeri di identificazione di tutti gli articoli citati in articolo corrente (`quotes`) e il numero corispondente (`numberOfQuotes`) e il numero di articoli dove articolo corrente viene citato (`quotedNumber`). Tutti gli articoli sono salvati in una struttura `ArticlesList`, che contiene vettore di articoli `articles` e il numero massimo di identificazione (cioe `id` dell'ultimo articolo - `maxArticleId`).

Questa struttura si può chiamare Lookup Table. Usiamo questa struttura perchè così possiamo facilmente accedere a qualunque articolo e perchè non richiede nessuna iterazione aggiuntiva quando si accede all'elemento della lista degli articoli, perchè articoli sono aggiunti nella lista con i suoi numeri di identificazione. Vantaggio della questa struttura è che il tempo usato per una iterazione è $O(1)$. Svantaggio è che lo spazio si memoria usato è $O(n+1)$ - dove n è numero di strutture `Article` nella lista (più 1 perchè `articles[0]` e sempre vuoto, gli identificatori partendono da 1).

Prima creiamo una nuova vuota lista degli articoli (`createArticleList`), che poi viene riempita con gli articoli letti dal file. Usando le funzioni che si trovano in `readfile.c`, programma legge da file una riga che contiene un articolo (funzione `readOneArticle`). Con `readSingleItem` si leggono `id` e `name` mentre con `readMultipleItems` si leggono `autors` e `quotes` dal file, perchè articolo può avere più autori e può citare più altri articoli. Quando la lettura è andata bene, si crea nuovo articolo con funzione `createNewArticle`, con tutte le informazioni di articolo, cioè vettori di nomi e di citazioni, numero di identificazione ecc.

Leggiamo tutte le rige, una per una, fino a quando non ci sono più rige in file. Aggiungiamo articoli nella lista `articlesList`, fino a quando tutti gli articoli sono messi nella lista. Dopo di questo chiudo il mio file.

La connessione delle due strutture `Article` e `Autor`

Ora voglio creare una nuova vuota lista degli autori, che poi viene riempita. Lo faccio con `createNewAutorsList`.

In questo punto abbiamo bisogno di struttura `Autor` i cui attributi sono: nome di autore (`name`), numero di suoi articoli (`numberOfArticles`) e numero di volte quando è stato citato (`numberOfQuotes`), lista degli tutti i suoi articoli (`articles`), lista di tutti articoli dove è citato (`quotedArticles`) e anche suo indice di Hirsch (`hindex`). Anche in questa struttura abbiamo la lista - questa volta di autori `AutorsList`, che contiene vettore di autori `autors`, il numero di autori `numberOfAutors` e un vettore aggiuntivo `autorsHelperArray`.

Non possiamo, come nel caso di struttura `Article`, usare solo Lookup Table perchè non ci sono identificatori come numeri interi di autori, ci sono solo i nomi degli autori. Per questo, abbiamo anche un vettore aggiuntivo `autorsHelperArray` che serve solo per memorizzare nomi degli autori e per ogni nome abbiamo un indice `i` che va da 0 a $m - 1$, dove m è il numero di autori. Li memorizziamo in ordine di caricamento. Poi, grazie a funzione `getAutorByName` possiamo trovare autore in `autorsHelperArray` per qualche nome e grazie a funzione `getKeyOfAutorArray` lo aggiungiamo in lista `autors` come intera struttura `Autor`. Per questo ci serve un po più di tempo. In `autorsHelperArray` uso $O(n)$ tempo più $O(1)$ per una iterazione in `AutorsList`. Lo spazio si memoria usato è $O(n + 1)$ - dove n è numero di strutture `Autor` nella lista più $O(m)$ - dove m è numero di autori.

Ora vogliamo creare un nuovo autore e determinare per ogni autore il suo numero di articoli, il suo numero di citazioni totali e il suo Hindex. Passiamo attraverso tutti gli articoli in `articlesList`, per creare autori. Durante questo, usiamo tre funzioni:

`modifyArticlesQuotedNumber` - questa funzione modifica lista degli articoli, aggiungendo numero di articoli dove articolo corrente è stato citato, cioè aggiunge per ogni articolo il suo `quotedNumber`.

`modifyAllArticlesAutors` - questa funzione aggiunge e modifica autori, di un array di articoli. Usiamo la funzione `addAutors`.

`addAutors` - va attraverso lista degli articoli e aggiunge autori in `autorsList`, passando tra tutti gli autori per articolo corrente. Ogni autore proviamo a trovare in lista degli autori e ci sono due possibilità - si già trova o non si trova nella lista. Se per esempio un autore già esiste nella lista degli autori allora non lo aggiungiamo di nuovo.
- Se un autore non si trova nella lista degli autori, usiamo la funzione `createNewAutor`, durante il cui usiamo funzione `insertSortedAutor` così si ordinano gli articoli dell'autore per numero di citazioni non crescente, che successivamente sarà utile per il calcolo di indice di Hirsch. Una condizione che utilizziamo per l'ordinamento è `checkQuotednumber` - numero di articoli dove articolo è stato citato.
- Se un autore si trova nella lista degli autori, usiamo la funzione `updateAutorArticles` che aggiunge articolo a autore in ordine.

`modifyAllArticlesQuotedAutors` - questa funzione calcola quante volte un autore è stato citato. Andiamo attraverso gli articoli in `articlesList`. In primo ciclo `for` prendiamo tutti gli articoli che articolo corrente è citato. Prendiamo un tale articolo e lo chiamiamo `quotedArticle`. In secondo ciclo `for` troviamo autori di `quotedArticle`, e per ogni autore aggiorniamo il numero di volte quando è stato citato.

In questo punto abbiamo caricati anche la lista degli articoli e la lista degli autori contenenti tutte le informazioni.

L'algoritmo `QuickSort` e calcolo di indice di Hirsch

L'algoritmo di ordinamento `QuickSort` Vogliamo ora ordinare gli autori per numero di articoli non crescente. Se due autori hanno uguali numeri di articoli, allora per numero di citazioni, non crescente. Se hanno numero di citazioni uguali, per ordine alfabetico. Per questo, usiamo l'algoritmo `QuickSort`. Questo è un algoritmo di ordinamento ricorsivo. n rappresenta numero di autori. Lo spazio di memoria usato è $O(\log(n))$. Il tempo usato in caso peggiore - si succede quando una partizione contiene 0 elementi e l'altra $n-1$ e in questo caso abbiamo:

$$T(n) = T(n-1) + O(n)_{partition} => T(n) = \sum_{k=0}^{n} k = O(n^2)$$

Il tempo usato in caso medio - si valuta il numero medio di confronti tra elementi del vettore di ingresso eseguiti dall'algoritmo e in questo caso abbiamo:

$$T(n) = 2T(n/2) + O(n) = O(n \log_2 n)$$

In ogni caso dobbiamo anche aggiungere il tempo $O(n)$ perchè usiamo anche un vettore aggiuntivo `autorsHelperArray`.

Algorithm 1 Algoritmo di QuickSort - pseudocodice

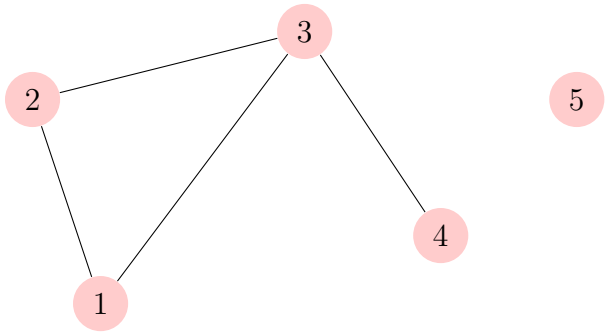
procedure Quicksort($a, low, high$)
if $low < high$ **then**
$p =$ Partition($a, low, high$)
Quicksort($a, low, p - 1$)
Quicksort($a, p + 1, high$)
endif
end procedure
procedure Partition($a, low, high$)
$pivot = a[high]$
$i = low - 1$
for $j = low$ **to** $high - 1$ **do**
if $a[j] < pivot$ **then** $i = i + 1$
swap $a[i]$ with $a[j]$
endif
swap $a[i]$ with $a[high]$
endfor
end procedure

Indice di Hirsch L'ultima cosa che ci serve è l'indice di Hirsch per autori. Come si calcola indice di Hirsch per un autore? Prima calcoliamo per ogni articolo scritto da un autore il suo numero di citazioni totale, cioè quante volte articolo viene citato in altri articoli. Ciò fatto, mettiamo gli articoli in ordine non crescente rispetto a numero di citazioni e assegniamo ad ogni articolo come indice il numero d'ordine. Lo facciamo con l'algoritmo `QuickSort`. Poi, confrontiamo indice e numero di citazioni. Ricordiamo indice di ultima volta quando indice è un numero maggiore di numero di citazioni. Questo indice è indice di Hirsch. Per quest'ultimo abbiamo usato la funzione `calculateHindexForAutor`.

Connesione tra autori che collaborano fra loro

Abbiamo detto che un autore può scrivere un articolo da solo oppure con una o con più altre persone. Così si creano diversi gruppi di autori che collaborano fra loro. Questi gruppi si possono rappresentare con un grafo non orientato, dove un lato collega due autori che hanno almeno una collaborazione. Usando il grafo, possiamo trovare due cose: il diametro e il clique number.

Per esempio, se abbiamo cinque autori {`A1,A2,A3,A4,A5`} dove `A1`, `A2` e `A3` hanno scritto un articolo insieme, anche `A3` e `A4` hanno una collaborazione e `A5` ha scritto tutti gli suoi articoli da solo, allora il nostro grafo si presenta così:



Ora vogliamo creare un grafo di collaborazioni non orientato. Non abbiamo usato grafo all'inizio perchè non ci servono tutte le informazioni degli autori - vogliamo sapere solo con cui un autore collabora e quindi non dobbiamo ricordare informazioni come indice di Hirsch, numero di articoli ecc. Così salviamo un pò di memoria.

Costruiamo un grafo con la struttura `GraphNode` che ha un nome di autore `autorName`, un puntatore al prossimo elemento nella lista dei nodi (con cui non deve, ma può essere collegato) `GraphNode* next` e puntatore a un nodo di suo vicino, con cui è collegato `GraphNeighbour* neighbour`. La struttura `GraphNeighbour` ha puntatore al suo nodo `GraphNode* node`, che si trova in grafo e ha un puntatore al prossimo vicino di nodo corrente `GraphNeighbour* link`. La struttura `Graph` ricorda solo il primo nodo `GraphNode* start` e numero totale dei nodi `numberOfNodes`. Quindi possiamo facilmente passare attraverso il grafo, partendo da primo nodo.

Con `createGraph` creiamo un nuovo grafo, inizialmente vuoto, che poi viene riempito grazie a funzione `fillGraphWithAutors`. Guardiamo tale funzione. Vogliamo mettere tutti gli autori e i suoi collaboratori (vicini) in grafo. Lo facciamo così: andiamo attraverso tutti gli articoli e aggiungiamo autori come nodi con funzione `addNodesByAutorsName`. Funzione `findNodeInGraphByName` controlla se c'è ancora un nodo con lo stesso nome, e se già esiste, non lo aggiunge. Se non esiste, si crea nuovo nodo (con `createGraphNode`) e si aggiunge in grafo (con `addNodeToGraph`). Ora tutti gli autori sono messi in grafo, ma non sono ancora collegati.

Ora vogliamo trovare tutti i collaboratori per ogni autore. Usiamo la funzione `addNeighboursForAutor`. Prima aggiungiamo solo quegli autori con i quali l'autore ha lavorato sull'articolo corrente, facciamo così per tutti gli articoli. Con funzione `findNeighbourForNode` chiediamo se il nodo si già trova nella lista degli vicini per nodo corrente. Se non si trova lo creiamo (con `createGraphNeighbour`) e aggiungiamo a nodo corrente (con `addNeighbourToNode`). Così aggiungiamo tutti gli collaboratori ad ognuno autore (se non è già aggiunto) - con due cicli for.

In questo punto sono creati tutti gli collaboratori. Nodi sono conessi con i suoi vicini.

Algoritmo di Floyd-Warshall per calcolare diametro

Diametro è la lunghezza massima del cammino più breve fra due autori, calcolata come numero di lati. Infatti, vogliamo calcolare la distanza tra due autori che sono più lontani fra loro. Se ci sono più diverse "strade" per arrivare da un autore a un altro, scegliamo la "strada" più breve.

Per esempio, se guardiamo grafo di collaborazioni sopra e se escludiamo `A5` (in questo caso, diametro è uguale a 5, che è il numero di vertici, perchè grafo non è conesso), troviamo che `A1` e `A4` sono due autori più lontani fra loro. Ci sono due cammini da uno ad altro, scegliamo quello più breve - diametro vale 2.

Per calcolare il diametro usiamo il algoritmo di Floyd-Warshall. Usiamo la funzione `calculateDiametroForGraph`. All'inizio costruiamo una matrice nxn dove n è il numero dei nodi. Lo facciamo con `matrixInitialization`. Questo occupa $O(n^2)$ spazio di memoria. All'inizio tutti gli elementi della matrice sono uguali a n, come per un grafo non conesso (per definizione). Abbiamo bisogno di un vettore aggiuntivo, che ricorda i nomi di nodi. Inizializzaziamolo con `helpersArrayInitialization`.

Controlliamo poi se due nodi sono vicini o no e modifichiamo la matrice. Assegniamo numero 1 in posizione (i, j) se due nodi i e j sono vicini oppure resta il numero n se non ci sono. Dopo di questo, usiamo il algoritmo di Floyd-Warshall che trova il cammino minimo tra due nodi per tutte le possibili coppie e modifica la matrice cioè scrive le valori di cammini minimi tra due nodi. Ha un costo uniforme, il tempo usato ci sarà $O(n^3)$ e lo spazio di memoria usato $O(n^2)$, sempre con n uguale a numero di nodi. In particolare, lo spazio di memoria usato è pari a dimensione di una matrice nxn. Alla fine, calcoliamo il diametro - massimo tra tutti i cammini minimi calcolati con funzione `getMaximumFromMatrix`.

Algorithm 2 Algoritmo di Floyd-Warshall - Inizializzione

for $i = 1, 2, \ldots, n$ **do**
for $j = 1, 2, \ldots, n$ **do**
if $i = j$ **then**
$c_{ii} := 0$, $b_{ij} := i$
else if $(v_i, v_j) \in E$ **then**
$c_{ij} := w(v_i, v_j)$, $b_{ij} := i$
else if $(v_i, v_j) \notin E$ **then**
$c_{ij} := \infty$, $b_{ij} := \perp$
endif
endif
endfor
endfor

Algorithm 3 Algoritmo di Floyd-Warshall - Implementazione

for $k = 1, 2, \ldots, n$ **do**
for $i = 1, 2, \ldots, n$ **do**
for $j = 1, 2, \ldots, n$ **do**
if $c_{ik} + c_{kj} < c_{ij}$ **then**
$c_{ij} := c_{ik} + c_{kj}$, $b_{ij} := b_{kj}$
endif
endfor
endfor
endfor

Nel algoritmo c_{ij} rappresenta peso (o costo) di cammino minimo tra nodi i e j. Costo di cammino è la somma dei suoi lati: $\sum_{i=1}^{n-1} w(v_i, v_{i+1})$. Nel nostro caso non si usa mai b_{ij}.

Calcolo di clique massimali, clique number e numero di sottoinsiemi

Clique number è la cardinalità massima di un sottoinsieme di autori che collaborano tutti fra loro, cioè hanno tutti almeno un articolo in collaborazione con ciascuno degli altri autori che si trovano nello sottoinsieme stesso. Un altra cosa interessante sono le clique massimali. Sono quelle che non sono contenute strettamente in altre clique.

Prima di tutto, costruiamo due strutture. Abbiamo una struttura `QueueElement` i cui attributi sono: numero di tutti autori - `numberOfAutors` e un vettore con i nomi di autori - `autors`. In particolare, `QueueElement` rappresenta vettore degli tutti autori che collaborano fra loro e il numero di autori in tale vettore. Quando si aggiungono elementi in vettore si aggiungano in ordine alfabetico grazie a funzione `addAutorToQueueElement`.

C'è un altra struttura `Queue` con attributi: indice di primo elemento aggiunto - `front`, indice di ultimo elemento aggiunto - `rear`, numero di elementi - `size`, capacità - numero massimo di elementi che possono esistere in coda (`capacity`) e anche un vettore di tipo `struct QueueElement**`.
Per lavoro con le code utilizziamo le funzioni standard come: `enqueue` - che aggiunge elemento in coda; `dequeue` - che rimuove elemento da coda ecc.

Ora possiamo determinare clique number e clique massimali. Procceso è il seguente: dopo creazione di una nuova coda vuota con `createQueue`, vogliamo mettere in tale coda tutte le coppie di autori che collaborano fra loro ad almeno una pubblicazione. Lo facciamo con funzione `fillQueueWithPairOfAutors` - andiamo attraverso grafo e per ogni nodo aggiungiamo i suoi vicini, ma in modo che ci sono solo due nodi in `QueueElement`.

Poi, con funzione `addNewAutorsConnectionToQueue` si va attraverso tutte le coppie di autori che già esistono e li aggiungiamo tutti possibili nodi di grafo. Lo facciamo cosi: prima creiamo una coda aggiuntiva - `helpQueue`. Poi, rimuoviamo un elemento dalla testa della coda e controlliamo se nodo (autore) è connesso con la coppia cioè con elemento dato (uso la funzione `checkIfAutorisConnectedWithGivenAutors`).
- Se sono connessi allora aggiungiamo autore in `QueueElement` e mettiamo elemento in coda aggiuntiva `helpQueue`.

- Se non sono connessi allora solo torniamo elemento in coda aggiuntiva `helpQueue`.

Alla fine `queue` diventa `helpQueue`. Usiamo anche la funzione `sortQueue` per ordinare elementi di coda. Prima li ordiniamo rispetto a cardinalità decrescente e, a parità di numero di autori, in ordine alfabetico. Con questa funzione anche rimuovo duplicati, perche processo descrito sopra produce duplicati e vogliamo rimuoverli tutti.

Ora abbiamo tutte le clique massimali. Usando la funzione `getCliqueMaximumNumber` troviamo il numero di clique massimali cioè clique number. Poi, con funzione `calculateSubsetNumberAndPrintQueue` troviamo numero di elementi di cardinalità massima (numero di "sottoinsiemi") e stampiamo tutte le clique massimali.