UNIVERSITÀ DEGLI STUDI DI MILANO

Algoritmi

"Contagio"

Giugno 2021

Proserpio Lorenzo
matricola N°928121

prof. Cordone Roberto & Goldwurm Massimiliano

Indice

Il problema

Lo scopo principale del progetto è simulare l'andamento del contagio di una malattia e trarre statistiche utili a modellizzare il fenomeno. La simulazione avverrà individuo per individuo, sono noti il numero di individui e l'orizzonte temporale preso in esame. Si presuppone l'esistenza di una struttura ospedaliera con capienza limitata e in cui si da' priorità ai soggetti più fragili. Il virus non viene immediatamente rilevato una volta contratto, ma vi è un periodo di incubazione durante il quale l'ospite risulta comunque infetto. Di ogni soggetto sono noti i legami sociali, ovverosia quali altre persone frequenta giornalmente, presupponiamo amici o parenti. Inoltre sappiamo anche quanto stretto sia il legame tra due persone, cioè quanto si frequentano. Ogni giorno coloro che sono sani o che sono infetti, ma non rilevati, incontrano i loro amici, rischiando, in base alla loro refrattarietà alla malattia, di essere contagiati nel primo caso e di diffondere il contagio nel secondo. Viene adottata una politica di quarantena domestica per tutti quei soggetti infetti e rilevati, ma che non trovano posto in ospedale. Tali individui vengono messi in una lista d'attesa, ordinata sempre per refrattarietà, nel caso si liberi qualche posto e non si presentino casi più gravi (rappresentati da soggetti più fragili). Tutti coloro che sono in quarantena domestica e in ospedale ricevono le cure, ovviamente migliori nel secondo caso, e dopo un periodo di terapia che dipende da individuo ad individuo guariscono e diventano immuni. A fini statistici siamo anche interessati a stimare il diametro "spaziale/temporale" del contagio (cioè il massimo, al variare delle coppie di persone, del minimo numero di persone attraverso cui deve passare il virus al fine che un membro della coppia infetti l'altro) e a determinare i gruppi di individui più strettamente legati fra di loro (che rappresentano possibili focolai). Terminata la simulazione vogliamo possedere anche le seguenti statistiche:

- il numero totale dei contagiati;
- il numero totale di ricoverati in ospedale;
- il numero di individui ancora contagiati alla fine dell'orizzonte temporale;
- il numero degli individui rilevati, ma non subito ricoverati in ospedale;
- il numero totale di giorni di degenza in ospedale;
- il numero totale di giorni in quarantena;

- il tempo medio in giorni di attesa (se ben definito), cioè il rapporto tra il numero totale di giorni in quarantena e il numero di individui rilevati infetti e non subito contagiati.

Modello astratto

Persone

Modellizziamo le persone come oggetti, nel senso matematico del termine, con le seguenti proprietà:

- cognome;
- età dell'individuo $e_i < 100$;
- gli eventuali giorni all'alba dell'orizzonte temporale da cui risulta contagiato;
- un coefficiente numerico che rappresenta lo stato di salute s_i (più è alto più il soggetto è sano).

Inoltre modellizziamo la refrattarietà con un coefficiente che dipende dall'età e dallo stato di salute. Tale coefficiente, chiamato ρ_i, si ottiene, per la persona i, come $s_i(1-e_i/100)^3$ e possiamo vedere che soggetti più fragili (più anziani o meno in salute) possiedono un ρ_i più basso. In termini algoritmici le persone saranno i nodi del nostro grafo a cui ci riferiremo tramite indici o puntatori.

Relazione di amicizia

La relazione di amicizia viene modellizzata come una relazione simmetrica tra due individui, corredata di un coefficiente numerico f_{ij} che modellizza il loro legame sociale (più è alto, più hanno un rapporto stretto). In termine algoritmici ognuna di tale relazioni rappresenta un arco non orientato con peso tra due nodi. Dunque le persone corredate delle rispettive relazioni di amicizia sono un grafo non orientato con archi pesati. La motivazione di tale scelta deriva direttamente dalla simmetria della relazione.

Individui strettamente legati tra loro

Per astrarre e modellizzare il concetto di individui strettamente legati tra loro poniamo la seguente definizione: la persona i e la persona j sono *strettamente legati* fra di loro se $f_{ij} \geq \max\{\rho_i, \rho_j\}$. Dunque i gruppi di persone che

si frequentano assiduamente in modo diretto o indiretto sono rappresentati dalle componenti connesse del grafo del punto precedente privato degli archi che non rispettano la condizione della definizione. La motivazione risiede nel fatto che, in caso di contagio di j, k rischia di essere contagiato tramite i. Poi dovremo stampare in ordine alfabetico tali gruppi, quindi ci servirà un algoritmo di ordinamento.

Potenziale velocità di contagio

Per modellizzare la potenziale velocità di contagio siamo interessati, per ogni coppia, a capire quale sia il numero minimo di passaggi necessari affinchè un membro infetti l'altro. Con passaggi intendo il numero di giorni (che equivale al numero di persone) intercorrono prima che l'altro membro della coppia risulti infettato. Dopodichè siamo interessati al massimo tra tali valori. Questo lavoro corrisponde al trovare la lunghezza di tutti i cammini minimi tra possibili coppie di nodi, assegnando peso ad ogni arco pari ad 1, del grafo di due punti sopra. Dopodichè è necessario trovare il massimo di tali lunghezze e la coppia a cui è associato. Se le coppie sono più di una si sceglierà quella che viene prima in ordine alfabetico.

Ospedale e quarantena

Per descrivere l'aggressività della malattia viene introdotto un parametro g_{max} che modifica il tempo a cui i pazienti si devono sottoporre alle cure (casalinghe o in ospedale). L'ospedale deve essere una struttura che risulti comoda per le seguenti operazioni: ricoverare e dimettere. Tradotto, sebbene abbia una capacità massima, bisogna poter aggiungere persone e togliere persone in modo agile. Inoltre non ci serve mantenere un ordine particolare tra i pazienti, dunque la struttura più adatta è una lista, associata ad un numero che tenga il conto dei posti liberi. Per tenere traccia degli individui in quarantena (che devono essere ordinati rispetto all'indice di refrattarietà) non userò una struttura apposita, ma lo stesso vettore, denominato *controllo*, utilizzato per tenere traccia dei sani, infetti non rilevati, infetti dal giorno successivo, immuni e individui in ospedale. Tale vettore è ordinato con l'ordine con cui si carica il database delle persone da file. E viene visitato in ordine di refrattarietà. In tal modo si evita di dover riordinare sempre la quarantena ogni giorno, perchè ogni individuo aggiunto viene aggiunto già nella posizione giusta della lista di attesa, tale idea verrà spiegata meglio quando verrà commentato il codice.

Scansione della giornata

Ai fini della simulazione discretizziamo il tempo in giorni. Ogni giornata si ripete e si svolge cronologicamente esattamente come segue:

1. ogni paziente in ospedale vede il proprio tempo di decorso della malattia aumentato di due, se tale tempo supera la soglia $g_{max}(1 - \rho_i)$ allora il paziente risulta guarito, viene dimesso dall'ospedale e diventa immune alla malattia;

2. ogni malato in quarantena vede il proprio tempo di decorso della malattia aumentato di uno, se tale tempo supera la soglia $g_{max}(1 - \rho_i)$ allora il paziente risulta guarito e diventa immune alla malattia;

3. i soggetti infetti che vengono rilevati sono segnati nella lista di attesa per l'ospedale scavalcando le altre persone con ρ_i maggiore, viene loro imposta la quarantena domestica, non incontrano più nessuno e vedono il decorso della loro malattia aumentato di uno;

4. in base ai posti liberi in ospedale si scorre la lista di attesa e si mandano gli eventuali candidati in cura;

5. tutti coloro che sono o non contagiati o infetti non rilevati (in realtà vedremo che basterà fare questo processo solo per la prima categoria) incontrano i loro amici/parenti. Se questi sono infetti il "coefficiente di contagio" (posto all'inizio di ogni giornata uguale a zero) viene aumentato del valore f_{ij}, se tale coefficiente supera strettamente ρ_i l'individuo i, inizialmente sano, viene contagiato e risulta infetto non rilevato a partire dal giorno successivo.

Strutture dati e algoritmi

Si commenteranno in modo *top-down* prima le strutture, poi il main e infine ogni libreria necessaria. Il nome di ogni sottosezione coinciderà con il nome del file che sto commentando. In tutti i paragrafi successivi con N intendo il numero di persone che compongono la popolazione.

strutture.h

La struttura più importante è il grafo che è stato implementato come *forward star*, perchè aprendo i file di esempio ho notato che non vi erano molti archi. Il grafo non è altro che un vettore di lunghezza N di strutture chiamate *amici*.

Ognuna di queste strutture deve contenere un puntatore ad una persona (nodo) chiamato *whose*, un vettore di indici di nodi che rappresentano gli amici di *whose* chiamato *who* e un vettore di float che rappresentano i pesi degli archi. Questa struttura occupa uno spazio dell'ordine di $\mathcal{O}(N + E)$, dove con E intendo il numero di archi. A questo punto è necessario definire una struttura che mi permetta di caricare la popolazione. Tale struttura è un vettore di lunghezza N di struct *persona*. La struct persona contiene tutte le informazioni necessarie per ogni persona: nome, età, salute, tempo da cui sono infetti, ρ e tempo necessario per il decorso della malattia. Tale vettore per intero è di dimensioni dell'ordine di $\mathcal{O}(N)$. L'ospedale è implementato come una lista in cui ogni elemento è del tipo: indice del nodo e puntatore all'elemento successivo. Se la capacità dell'ospedale è r allora tale struttura occupa al più uno spazio $2r$, possiamo dire quindi che è dell'ordine di $\mathcal{O}(r)$ (notare che affinchè il tutto sia sensato devo avere $r \leq N$). Infine, per comodità, si definisce il tipo *bool*.

progetto202106.c

Questo file è il *main* del progetto, l'analisi di complessità di ogni funzione che appartiene ad una libreria specifica sarà rimandata alle sezioni successive, per il momento mi occupo solo della complessità temporale e spaziale del *main* vero e proprio. Ho bisogno, nell'ordine, di procedure che:

1. gestiscano il file di input, allochino tutte le strutture necessarie per caricare le persone (libreria *gfileerr.c*);
2. ordinino e stampino degli elenchi secondo diversi criteri (libreria *heapsort.c*);
3. allochino e carichino il grafo da file, calcolino e stampino le componenti connesse (libreria *grafi.c*);
4. calcolino i cammini minimi e ci trovino il massimo (libreria *cammini.c*);
5. allochino le strutture necessarie, eseguano e stampino i risultati della simulazione e le statistiche volute (libreria *simulazione.c*).

Questi punti verranno sviscerati in maniera esaustiva nelle sezioni seguenti. Quello che rimane da commentare del *main* è la dichiarazione delle variabili (la cui maggior parte sono vettori che poi verranno allocati dinamicamente dalle procedure), vari cicli ausiliari e la gestione della deallocazione della memoria. Le variabili sono in numero costante di dimensione costante (prima che siano allocate) e quindi occupano uno spazio dell'ordine di $\mathcal{O}(1)$, per la

precisione, contando che una stringa di mille caratteri occupi mille posti (potrebbe essere ridotta, ma serve per gestire il file di cui a priori non so la lunghezza delle righe), 1027. Per quanto riguarda i cicli ausiliari abbiamo un ciclo per allocare ogni riga del grafo *forward star*, dunque dovremo ricordarci in fase di analisi finale della complessità di moltiplicare il costo spaziale di *alloca_who* per $\mathcal{O}(N)$, un ciclo per caricare il grafo da file quindi il costo spaziale di *carica_who* va moltiplicato per $\mathcal{O}(N)$, un ciclo sul tempo in cui viene richiamata la funzione *day* il cui costo temporale va moltiplicato per $\mathcal{O}(T)$ (con T intendo il numero di giorni presi in considerazione) e un ciclo di lunghezza N per calcolare il numero di soggetti rimasti. Per quanto riguarda le procedure di *free* sono tutte al più cicli di lunghezza N. Come nota generale a quasi tutte le procedure gli argomenti vengono passati per indirizzo, la memoria utilizzata dunque è sempre costante (pari a quella di tot puntatori), nei casi in cui vengano passati oggetti di dimensione non costante indicherò anche quello nella complessità spaziale. Ricordo che i vettori C li passa di default per indirizzo (quello della prima cella del vettore).

gfileerr.c

Procedo ora ad elencare le procedure, con breve descrizione del loro funzionamento ed analisi della loro complessità spaziale e temporale:

- *error_command_line* controlla di aver letto una stringa da riga di comando, termina il programma altrimenti. Complessità spaziale: $\mathcal{O}(1)$. Complessità temporale: $\mathcal{O}(1)$.

- *carica_par* legge i parametri N, T, g, g_{max} e r. Complessità spaziale: $\mathcal{O}(1)$. Complessità temporale: $\mathcal{O}(1)$.

- *alloca_person* alloca il vettore di struct *persona* e un vettore contenente puntatori a struct *persona*. Complessità spaziale: $\mathcal{O}(N)$. Complessità temporale: $\mathcal{O}(1)$.

- *carica_pop* finchè non arriva in fondo al file legge una riga per volta e carica i campi delle struct *persona*; inoltre, calcola l'indice di refrattarietà e il tempo necessario al decorso della malattia. Complessità spaziale: $\mathcal{O}(1)$. Complessità temporale: $\mathcal{O}(N)$ (le righe del file sono tante quante le persone).

- *set_nodi* riempie il vettore di puntatori a struct *persona* con gli indirizzi delle persone. Lo userò poi negli algoritmi di ordinamento, perchè scambiare due puntatori è più efficiente che scambiare due struct. Complessità spaziale: $\mathcal{O}(1)$. Complessità temporale: $\mathcal{O}(N)$.

heapsort.c

Questa libreria contiene le funzioni necessarie per ordinare i nodi e stampare gli elenchi. Come algoritmo di ordinamento ho scelto *heapsort* perchè a priori non posso supporre nulla sui dati, ha complessità costante nel caso migliore, peggiore e medio, inoltre è facilmente modificabile per le richieste del problema in caso di parità di coefficienti. Ho implementato due tipi leggermenti diversi di *heapsort*. Il primo per soddisfare la richiesta di stampare un elenco ordinato per età decrescente (in caso di parità salute crescente e poi ordine alfabetico), elenco che poi non ci serve più quindi ho preferito usare il vettore *aux_nodi* che viene risettato come all'inizio una volta effettuata la stampa, il secondo per soddisfare la richiesta di ordinare per ρ decrescenti (e in caso di parità alfabetico) conservando però il vettore delle permutazioni, cio' ci servirà nella fase di simulazione. Analizzo in blocco le procedure simili:

- *alloca_permutation* alloca i vettori che tengono traccia delle permutazioni dei nodi, uno ci servirà per l'ordine alfabetico l'altro rispetto a ρ. Complessità spaziale: $\mathcal{O}(N)$. Complessità temporale: $\mathcal{O}(1)$.

- *set_permutation* setta i vettori di permutazione con un ordinamento naturale. Complessità spaziale: $\mathcal{O}(1)$. Complessità temporale: $\mathcal{O}(N)$.

- *Swap_int* e *Swap* sono semplici funzioni di scambio di interi o di puntatori a struct *persona*. Complessità spaziale: $\mathcal{O}(1)$. Complessità temporale: $\mathcal{O}(1)$.

- *HeapSort_eta, HeapSort_salute, HeapSort_alfabetico, HeapSort_rho_perm* e *HeapSort_alfabetico_perm* con le rispettive funzioni di aggiornamento dell'*heap* hanno complessità spaziale: $\mathcal{O}(1)$. Complessità temporale: $\mathcal{O}(N \log(N))$.

- *Parita_salute, Parita_alfabetico_salute* e *Parita_alfabetico_rho* gestiscono i casi in cui vi sia una parità nel coefficiente primario di ordinamento. Di fatto si limitano a svolgere un *heapsort* su sottosequenze dentro il vettore dei nodi laddove sia necessario. Complessità spaziale: $\mathcal{O}(1)$. Complessità temporale: $\mathcal{O}(N \log(N))$.

- *Stampa_ordine_eta* e *Stampa_ordine_rho* sono delle semplici funzioni di stampa. Complessità spaziale: $\mathcal{O}(1)$. Complessità temporale: $\mathcal{O}(N)$.

grafi.c

Ho già discusso i motivi per cui ho implementato il grafo come *forward star*, per calcolare le componenti connesse mi sono servito del classico algoritmo che sfrutta la visita in profondità del grafo ed un vettore di controllo che viene usato per marcare a quale componente appartenga un nodo. Per ottimizzare lo spazio non ho modificato il grafo costruendone un altro, ma mi sono limitato ad imporre che venga soddisfatta la condizione $f_{ij} \geq \max\{\rho_i, \rho_j\}$ affinchè un arco sia percorribile. Le procedure sono, raggruppando quelle ausiliarie e valutandole in blocco:

- *alloca_friends* e *alloca_who* servono ad allocare lo spazio necessario per il grafo. Complessità spaziale: $\mathcal{O}(N+E)$. Complessità temporale (per ogni singola chiamata, poi andrà moltiplicata nella parte finale): $\mathcal{O}(1)$.
- *alloca_formato* e *flush_amici* servono per leggere correttamente i dati da file. Complessità spaziale: $\mathcal{O}(1)$. Complessità temporale: $\mathcal{O}(N)$.
- *cerca_nome* associa il nome della persona al rispettivo indice numerico. Complessità spaziale: $\mathcal{O}(1)$. Complessità temporale: $\mathcal{O}(N)$.
- *carica_who* carica la lista di adiacenza con i rispettivi pesi per ogni nodo. Complessità spaziale: $\mathcal{O}(1)$ (perchè libero ad ogni ciclo *formato*). Complessità temporale: $\mathcal{O}(N)$ (gli amici sono al più $N-1$).
- *alloca_controllo* alloca il vettore usato per marcare. Complessità spaziale: $\mathcal{O}(N)$. Complessità temporale: $\mathcal{O}(1)$.
- *massimo* trova il massimo tra due float. Complessità spaziale: $\mathcal{O}(1)$. Complessità temporale: $\mathcal{O}(1)$.
- *visita_amici* e *componenti_connesse* contengono l'algoritmo classico *depth-first* per il calcolo delle componenti connesse di un grafo. Complessità spaziale: $\mathcal{O}(1)$. Complessità temporale: $\mathcal{O}(N+E)$.
- *carica_stampa_cc* salva temporaneamente le componenti connesse in un grafo *forward star*, le ordina in modo alfabetico, ordina i primi elementi in modo alfabetico e stampa rispettando l'ordine. Complessità spaziale: $\mathcal{O}(N)$ (in tutti i casi). Complessità temporale: $\mathcal{O}(N \log(N))$.

cammini.c

Al fine di trovare i cammini minimi ho usato l'algoritmo standard che permette di trovari i cammini minimi in un grafo pesato. Ad ogni arco ho dato

peso pari a 1. Ci interessa inoltre solo la matrice dei costi e non quella dei predecessori, questo ci porterà un risparmio di risorse. Le procedure in questa libreria sono:

- *is_edge* mi dice se esiste un arco. Complessità spaziale: $\mathcal{O}(1)$. Complessità temporale: $\mathcal{O}(N)$.
- *min* calcola il minimo tra due interi. Complessità spaziale: $\mathcal{O}(1)$. Complessità temporale: $\mathcal{O}(1)$.
- *alloca_matpath* alloca la matrice dei costi dei cammini. Complessità spaziale: $\mathcal{O}(N^2)$. Complessità temporale: $\mathcal{O}(N)$.
- *load_matpath* svolge l'algoritmo per i cammini minimi. Complessità spaziale: $\mathcal{O}(1)$. Complessità temporale: $\mathcal{O}(N^3)$.
- *ordina_coppia* ordina in modo alfabetico una coppia di persone (viste come indici). Complessità spaziale: $\mathcal{O}(1)$. Complessità temporale: $\mathcal{O}(1)$.
- *maxpath* cerca il massimo tra i cammini e restituisce la coppia ordinata e il valore della cella corrispondente al massimo. Complessità spaziale: $\mathcal{O}(1)$. Complessità temporale: $\mathcal{O}(N^2)$.
- *stampa_matpath* stampa il risultato di *maxpath*. Complessità spaziale: $\mathcal{O}(1)$. Complessità temporale: $\mathcal{O}(1)$.

simulazione.c

Questa libreria contiene le procedure della simulazione vera e propria, tutto il necessario per la gestione dell'ospedale,la stampa dei risultati e delle statistiche. Le procedure, accorpando quelle simili e/o ausiliarie sono:

- *alloca_mat_sim* e *set_mat_sim* alloca e setta con un carattere arbitrario la matrice che conterrà i risultati della simulazione. Complessità spaziale: $\mathcal{O}(N \cdot T)$. Complessità temporale: $\mathcal{O}(N \cdot T)$.
- *first_ospedale* alloca il primo elemento dell'ospedale (è ausiliario). Complessità spaziale: $\mathcal{O}(1)$. Complessità temporale: $\mathcal{O}(1)$.
- *azzera_contatti* azzera i contatti per chi entra in quarantena o in ospedale. Complessità spaziale: $\mathcal{O}(1)$. Complessità temporale: $\mathcal{O}(N + E)$.

- *add_hosp_direct, add_hosp_from_quarantine* e *rem_hosp* servono, rispettivamente, per inserire un individuo infetto direttamente in ospedale (ci servirà nella funzione *day_zero*), per inserire un individuo in ospedale dalla quarantena e per rimuovere un individuo dall'ospedale. Complessità spaziale: $\mathcal{O}(1)$. Complessità temporale: $\mathcal{O}(1)$.

- *traduci_carattere* serve semplicemente per tradurre un intero che corrisponde allo stato dell'individuo e convertirlo in modo che possa essere inserito nella *mat_simulation*. Complessità spaziale: $\mathcal{O}(1)$. Complessità temporale: $\mathcal{O}(1)$.

- *day_zero* gestisce il giorno zero, cioè quando gli individui vengono caricati, inoltre setta il vettore *controllo* con gli stati corretti (0 per individuo libero, 1 per infetto non rilevato, 2 per quarantena, 3 per ospedale, 4 per immune, 5 per infetto il giorno successivo). Manda in ospedale o quarantena gli individui che rispettano le condizioni e azzera i loro contatti. Nel caso peggiore ha complessità temporale di ordine $\mathcal{O}(N \cdot (N + E))$ e complessità spaziale pari a r (che volendo possiamo contare come $\mathcal{O}(N)$).

- *day* svolge un giorno di simulazione esattamente come descritto nella sezione *Scansione della giornata*. L'unica nota da fare è che visito i nodi del grafo in ordine di refrattarietà, il che mi permette di mettere ogni nuovo individuo che entra in quarantena nel posto giusto della lista di attesa, i primi ad entrare in ospedale in caso di posti liberi saranno gli individui il cui controllo è pari a 2 e quelli con refrattarietà minore vengono visitati necessariamente prima. Complessità spaziale: r (che volendo possiamo contare come $\mathcal{O}(N)$). Complessità temporale: $\mathcal{O}(N \cdot (N + E))$.

- *stampa_sim* stampa la matrice che contiene la simulazione. Complessità spaziale: $\mathcal{O}(1)$. Complessità temporale: $\mathcal{O}(N \cdot T)$.

- *stampa_statistiche* stampa le statistiche richieste. Complessità spaziale: $\mathcal{O}(1)$. Complessità temporale: $\mathcal{O}(1)$.

- *del_ospedale* libera un posto in ospedale, serve perchè nel main verrà usato per liberare lo spazio allocato. Complessità spaziale: $\mathcal{O}(1)$. Complessità temporale: $\mathcal{O}(1)$.

Analisi della complessità totale

Riprendiamo il *main* e valutiamo la complessità totale facendo attenzione a quali procedure chiamiamo più volte nel main. La complessità spaziale totale risulta: $\mathcal{O}(N^2) + \mathcal{O}(N \cdot T)$. La complessità temporale totale risulta: $\mathcal{O}(N^3) + \mathcal{O}(T \cdot N \cdot (N + E))$. Se ricordiamo che $E < N^2$ (siccome al più ogni persona può essere amica di tutte le altre meno se' stessa...) abbiamo che la complessità temporale risulta: $\mathcal{O}(T \cdot N^3)$. In conclusione direi che il tutto funziona in tempo e spazio polinomiale, quindi è un problema *trattabile* per Cobham. Come nota finale aggiungo che si sarebbe potuto sfruttare la simmetria della relazione di amicizia (qua e là nel codice è stato fatto), ma non porta guadagni in complessità asintotica, siccome l'ordine resta lo stesso.