Presentazione algoritmo di ricerca di flusso massimo con propagazione della malattia, esplorata tramite "propagazione dei nodi"

Filippo Magi

February 28, 2022

1 Strutture dati

1.1 BiEdge

BiEdge rappresenta l'arco che collega tra di loro due nodi. Ha i seguenti attributi

- **Flow**, rappresentato nello pseudo-codice come funzione $f(e)$, dove e è un arco generico, rappresenta la quantità di flusso inviata in quell'arco
- **Capacity**, rappresentato nello pseudo-codice come funzione $u_f(e)$, dove e è un arco generico, rappresenta la capacità residua di quell'arco
- **Reversed**, utile durante il l'invio del flusso, mi indica se devo inviare del flusso oppure diminuirlo

BiEdge ha un metodo a lui associato : **SendFlow**.

SendFlow riceve in input il valore del flusso che deve inviare e va a aumentare e diminuire il valore del flusso e della capacità residua, nel caso in cui reversed sia falso, altrimenti va a aumentare la capacità residua e diminuire il flusso inviato.

Ritorna due valori booleani, il primo mi indica se la capacità residua è pari a 0, quindi che quell'arco è diventato un arco bottleneck.

Il secondo mi indica che il valore del flusso o della capacità sono negativi, quindi c'è stato un errore, in quel caso aggiorno il valore del flusso da inviare per tutti i nodi e correggo l'errore dove ho già inviato il flusso.

1.2 Node

Node rappresenta, appunto, i nodi. Un generico nodo *nodo* è rappresentato dai seguenti attributi

- **Name**, usato per comprendere di quale nodo si sta parlando

- **Edges**, è la lista di archi che entrano ed escono *nodo*, nello pseudo-codice è indicato anche come funzione $\delta(node)$, inoltre vengono usati anche $\delta^+(node)$ per rappresentare i nodi uscenti da *node* e $\delta^-(node)$ per rappresentare i nodi entranti in *node*
- **SourceSide**, è un valore booleano che mi indica se *nodo* è stato esplorato partendo dal nodo sorgente s o dal nodo destinazione t
- **Label**, rappresenta la distanza tra *nodo* e il nodo sorgente s(nel caso in cui il nodo sia SourceSide) o dal nodo destinazione t
- **PreviousNode**, rappresenta il nodo predecessore, cioè, il nodo,esplorato a partire da s, che è stato usato per esplorare *nodo*, questo attributo è usato solo nei nodi esplorati a partire da s e nei nodi di confine, cioè i nodi dove i nodi esplorati da s e quelli esplorati da t si incontrano
- **PreviousEdge**, arco che collega *nodo* e il suo PreviousNode, valgono le stesse valutazioni di quest'ultimo
- **NextNode**, rappresenta il nodo successore, cioè il nodo,esplorato a partire da t, che è stato usato per esplorare *nodo*, questo attributo è usato solo nei nodi esplorati a partire da t
- **NextEdge**, rappresenta l'arco che collega *node* con il suo NextNode, valgono le stesse valutazioni di quest'ultimo
- **FlussoPassante** o **InFlow**, rappresenta la quantità di flusso che, con le informazioni ricevute fino a quel momento, è possibile inviare attraverso il percorso descritto attraverso NextNode e PreviousEdge
- **SourceValid** rappresenta se *nodo* è considerato valido nella parte esplorata a partire da s, quindi anche i nodi di confine
- **SinkValid** rappresenta se *nodo* è considerato valido nella parte esplorata a partire da t

Con nodo valido si intende che nel percorso descritto non vi sono i cosiddetti archi bottleneck, cioè archi dove non è più possibile inviare il flusso. I metodi a lui associati sono metodi per cambiare i valori e per inserire gli archi, nello pseudo-codice sono riassunti come segue

- **update**, dati un nodo e un arco, dalle informazioni ivi contenute, aggiorno il valore di SourceSide, Label (attraverso il metodo di Graph ChangeLabel) e FlussoPassante. Nel caso in cui il nodo dato sia stato esplorato a partire da t aggiorno NextEdge, NextNode e SinkValid, in caso contrario aggiorno PreviousEdge, PreviousNode e SourceValid
- **updatePath**, dati un nodo e un arco, aggiorno i valori di PreviousEdge, PreviousNode e SourceValid, inoltre lo dichiaro nodo di confine (utilizzando il metodo di Graph AddLast)

- **AddEdge**, aggiunge l'arco ricevuto in input in Edges
- **Reset**, azzero il valore di FlussoPassante di *node*, in maniera tale che possa essere nuovamente esplorato

1.3 Graph

Graph rappresenta il grafo, quindi l'insieme dei nodi e, di conseguenza, degli archi. È rappresentato come segue

- **LabeledNodeSourceSide**, è una lista di insiemi, dove ogni insieme contiene i nodi con una certa label, vi sono contenuti tutti i nodi esplorati a partire da s. Durante la fase di creazione del grafo, tutti i nodi tranne t sono qui inseriti, in attesa che vengano esplorati per la prima volta.
- **LabeledNodeSinkSide** , è una lista di insiemi, dove ogni insieme contiene i nodi esplorati a partire da t con una determinata label. Il nodo t è qui inserito durante la fase di creazione del grafo.
- **LastNodesSinkSide**, è un insieme di nodi che contiene i nodi che abbiamo chiamato "di confine".
- **LastNodesSourceSide**, è un insieme di nodi che contengono i nodi esplorati a partire da s che sono collegati con i cosiddetti nodi di confine

I metodi contenuti in Graph sono i seguenti:

- **ResetSourceSide**, prende in input una intero, per tutti i nodi con label pari o superiore che sono stati esplorati a partire da s (tranne i nodi di confine), indico che hanno FlussoPassante pari a 0, intendendo che non sono stati esplorati
- **ResetSinkSide**,prende in input una intero, per tutti i nodi con label pari o superiore che sono stati esplorati a partire da t , indico che hanno FlussoPassante pari a 0, intendendo che non sono stati esplorati.
- **ChangeLabel**, prende in input il nodo da spostare di label, un booleano che mi indica se è stato esplorato a partire da s o da t, e la label. Rimuovo dall'opportuno insieme il nodo indicato e lo inserisco in quello indicatomi, aggiornando i dati del nodo stesso
- **AddLast**, prende in input un nodo che deve essere stato esplorato a partire da t, lo inserisce in LastNodesSinkSide e inserisce tutti i nodi a lui collegati esplorati a partire da s in LastNodesSinkSide.

Nello pseudo-codice si è preferito indicare con formule matematiche le azioni da avvenire, ma si rifanno a queste.

2 Descrizione algoritmo

2.1 FlowFordFulkerson

Ricevuta in input una rete, inizializzo inserendo nelle due pile di nodi *vuotiSource* e *vuotiSink* i nodi s e t.

Ripeto i seguenti passi finché o non trovo un percorso o il flusso inviabile attraverso quel percorso è pari a 0:

1. inizio una ricerca di un nuovo cammino attraverso **DoBfs**
2. salvo il nodo di confine n e il flusso inviabile s da **DoBfs**
3. cancello le informazioni contenute in *vuotiSource* e in *vuotiSink*
4. retrocedo da n verso s inviando il flusso f (secondo le indicazioni dell'arco), attraverso il percorso descritto da PreviousNode
5. nel caso un arco,dopo l'invio del flusso, abbia capacità pari a 0, lo inserisco nella pila *vuotiSource*
6. da n avanzo fino a t, inviando il flusso f (secondo le indicazioni dell'arco), attraverso il percorso descritto da NextNode
7. nel caso un arco,dopo l'invio del flusso, abbia capacità pari a 0, lo inserisco nella pila *vuotiSink*
8. aggiorno la quantità di flusso inviata

2.2 DoBfs

Ricevo in input il grafo, la pila di nodi senza capacità ottenuti durante l'ultimo invio del flusso sia della parte di source *noCapsSource* sia della parte di sink *noCapsSink*.

1. Inizializzo tre code : *codaSource*, *codaSink*, *malati*, e due booleani : *sourceRepaired*, *sinkRepaired*, che, per il momento, mi indicano se la pila corrispondente è vuota o meno;
2. cerco di riparare ogni nodo presente in *noCapsSource*, aggiornando il FlussoPassante tra il nodo precedentemente esplorato e quello che quello che dovrò provare a riparare. Inserisco tutti i nodi non riparati in *malati*,e salvo mi salvo il primo nodo non riparato in *noCapSource*;
3. nel caso sia riuscito a riparare tutti i nodi e *noCapsSink* è vuota, tra i nodi di confine cerco un percorso con l'ultimo nodo riparato, nel caso lo trovi con flussoPassante positivo, significa che ho trovato un cammino e restituisco i dati;
4. per ogni nodo contenuto in *malati*, eseguo sickPropagation (della parte opportuna);

5. se sickPropagation mi restituisce un nodo di confine, e tale nodo può anche raggiungere t, indico che ho trovato un nuovo cammino e restituisco i valori;

6. se non ho nodi in $noCapsSink$(quindi $sinkRepaired = true$),per ogni nodo di confine che è SourceValid e SinkValid (con ulteriori controlli nel codice), cerco di raggiungere il nodo source, se lo raggiunge, ho trovato un nuovo cammino;

7. aggiorno sourceRepaired con il valore e inizializzo la coda $codaSource$, aggiungendo o il nodo sorgente s, o i nodi appartenenti a $LastNodesSourceSide$, oppure i nodi con label inferiore di uno a $noCapSource$, in tal caso per ogni nodo con label pari o superiore, esplorato a partire da s, lo indico come non esplorato, attraverso la funzione Reset;

8. eseguo gli ultimi sei punti per i nodi presenti in $noCapsSink$, attraverso le variabile dedicate ai nodi esplorati a partire da t;

9. inizio la ricerca bidirezionale, che finirà o quando ho trovato un cammino, o quando entrambe le code $codaSource$ e $codaSink$ saranno vuote;

10. se è presente almeno un elemento in $codaSource$ e la parte di source non è già stata totalmente riparata ($sourceRepaired = true$), estraggo un nodo $element$ da quest'ultima coda;

11. nel caso $element$ sia un nodo valido e già esplorato, esploro tutti i suoi nodi adiacenti;

12. nel caso il nodo adiacente che sto esplorando non sia stato esplorato precedentemente e non è stato precedentemente esplorato da t, aggiorno i suoi dati (funzione update) e lo inserisco in $codaSource$;

13. nel caso il nodo adiacente che sto esplorando sia già stato esplorato a partire da t e sia valido, indico che ho trovato un cammino,quindi aggiorno i dati (updatePath) e lo restituisco;

14. nel caso il nodo adiacente che sto esplorando sia stato esplorato a partire da t,non potrà essere esplorato durante questa "iterazione"(dato che $sinkRepaired = true$) e ha flussoPassante = 0, allora cerco di ripararlo,senza la limitazione della label, nel caso riesca ho trovato un nuovo cammino, altrimenti indico che la parte di sink non è stata riparata e inizializzo la coda a nodo t;

15. procedo a esplorare la parte di Sink, controllando che non sia stata già totalmente riparata e che ci sia almeno un nodo in $codaSink$;

16. estraggo dalla coda il nodo $element$, controllo che sia stato esplorato ($flussoPassante > 0$), sia SinkValid e che sia stato esplorato a partire da t ($\neg SourceSide$), in caso positivo procedo ad analizzare i suoi nodi adiacenti;

17. nel caso il nodo adiacente che sto esplorando sia già stato esplorato ($flussoPassante > 0$) ed è sia stato esplorato da s sia SourceValid, indico che ho trovato un percorso,quindi aggiorno i dati (updatePath) e restituisco il nodo *element*;

18. nel caso il nodo adiacente che sto esplorando non sia già stato esplorato, aggiorno i dati (metodo update).

2.3 RepairNode

Questa funzione mi permette di cercare di riparare un nodo n, con una indicazione che mi indica, nel caso sia un nodo di confine, se devo considerare i nodi esplorati da s o quelli esplorati da t . Nel caso riceva in input il nodo s o il nodo t, indico fin da subito che non sarà possibile ripararli. Controllo successivamente che il nodo non sia già sano, cioè che non c'è bisogno di ripararlo. di riparare il nodo n, cerco tra i suoi nodi adiacenti un nodo r con le seguenti proprietà:

- il nodo n deve poter ricevere (o inviare) del flusso dal nodo r e dall'arco che li collega;
- il nodo r deve essere valido, quindi SinkValid o SourceValid a seconda della parte che esploro;
- il nodo r deve avere label precedente a quella di n, cioè $r.label+1 = n.label$, questo non vale se sto trattando un nodo di confine e devo esplorare i nodi dalla parte di source.

Se il nodo r ha tutte queste tre proprietà,allora posso procedere a aggiornare i dati e indicare che il nodo è guarito, altrimenti indico che il nodo non è valido per la parte esplorata.

2.4 SickPropagation

Si divide in quella per i nodi di source e in quella per i nodi di sink. Le differenze sono minime : in quale coda vado a inserire i nodi, come si deve comportare la repair e quali nodi devo analizzare, dalla parte di source i nodi esplorati da s e i nodi di confine, dall'altra parte i nodi esplorati a partire da t. Ricevo in input il nodo non valido che deve propagare la malattia e la coda (o meglio, il puntatore della coda). creo una coda di nodi, dove andrò a inserire i nodi malati. Inserisco per primo il nodo ricevuto in input. Finché la coda non è vuota,procedo come segue:

1. estraggo il nodo malato dalla coda;
2. provo a ripararlo;
3. nel caso riesca a ripararlo, controllo se è un nodo di confine o meno, nel caso non lo sia, lo inserisco nella coda ricevuta in input, altrimenti lo restituisco;

4. se non riesco a ripararlo, inserisco i nodi che lo avevano come predecessore (esplorati da s), o come successore (esplorati da t), nella coda dei nodi malati.