Presentazione algoritmo di ricerca bidirezionale di flusso con divisione per label

Filippo Magi

February 17, 2022

1 strutture dati

1.1 BiEdge

L'arco BiEdge che collega due nodi, che ha le informazioni sulla capacità residua e sulla quantità di flusso inviata, oltre a un booleano per capire se durante l'invio del flusso deve inviarlo o ritirarlo.

1.2 Node

Il nodo contiene le informazioni sugli archi a lui collegati con una lista di BiEdge. Contiene un intero InFlow, che rappresenta il valore di flusso che è possibile inviare nel percorso predefinito, oltre a un intero Label, che rappresenta la distanza tra quel nodo e s o t, a seconda da quale parte è stato esplorato. Due booleani, il primo mi rappresentano se è stato esplorato da s o da t, il secondo se l'arco che lo collega con il nodo precedente (e quindi più vicino a s/t) ha capacità $= 0$. Inoltre ci sono le informazioni di indirizzamento (nextNode e nextEdge per i nodi esplorati da t, previousNode e previousEdge per i nodi esplorati da s).

1.3 graph

Graph è rappresentato da una lista di insiemi di nodi, cioè i nodi esplorati da s e da t, divisi per label, oltre ad avere due insiemi che contengono i nodi di confine fino a quel momento scoperti, sia dalla parte di s, sia dalla parte di t.

2 Descrizione

La fase di invio del flusso e il tentativo di recuperare il percorso riparando i nodi contenuti in noCapsSource o noCapsSink è come in NodePropagation.

Finché riesco a trovare un percorso s e t, cerco il percorso tramite la Bfs ricercata,e poi inviare il flusso. Nell'invio del flusso, seleziono, se presente, il

nodo che ha come capacità residua = 0,e uso quel nodo per la Bfs (questo per entrambi i lati esplorati).

Dato che analizzo l'InFlow presente, è possibile che non sia adeguatamente aggiornato, quindi nel caso in cui non è possibile inviare il flusso indicato, per i nodi già modificati gli inserisco la differenza, per il resto faccio con il valore aggiornato.

2.1 Bfs con ottimizzazione sugli ultimi livelli

gli passo in input, oltre al grafo, anche due pile di nodi **noCapSource** e **noCapSink**.

2.2 riparazione dei nodi di source

Nel caso in cui **noCapsSource** non sia vuoto, analizzo i nodi ivi contenuti. Provo a riparare il nodo **noCapSource**, ricavato dalla pop di **noCapsSource**, nel caso in cui riesca, aggiorno i dati e procedo con il nodo successivo, altrimenti inserisco nuovamente il nodo nella pila e smetto di smetto di provarli a ripararli. Nel caso in cui sia riuscito a ripararli tutti e **noCapsSink** sia vuota, cerco un "nodo di confine" valido, nel caso raggiunga l'ultimo nodo riparato, restituisco il percorso. Nel caso non sia riuscito a riparare il nodo, analizzo il nodo che non è stato possibile riparare, qui chiamato $notValidNode$:

- nel caso in cui $notValidNode$ sia il nodo sorgente s, lo inserisco nella coda dei nodi da esplorare di source
- nel caso in cui $notValidNode$ sia un nodo di confine, e quindi esplorato da t, inserisco nella coda tutti i nodi di confine, recuperabili attraverso il grafo
- altrimenti inserisco nella coda tutti i nodi $m|m.label+1 = notValidNode.label$, oltre a cancellare tutti le informazioni di indirizzamento di tutti i nodi con label pari o maggiore di $notValidNode$

2.3 riparazione dei nodi di sink

Nel caso in cui **noCapsSink** non sia vuota, analizzo i nodi ivi contenuti. Provo a riparare il nodo **noCapSink**, ricavato dalla pop di **noCapsSink**, nel caso riesca a ripararlo, aggiorno i dati e procedo col nodo successivo, finché la coda non è vuota. Nel caso in cui trovi un nodo non riparabile, lo inserisco di nuovo dentro la pila, e smetto di provarli a ripararli. Nel caso sia riuscito a riparare tutti i nodi di sink e **noCapSource** sia vuoto, cerco un possibile percorso già presente. Per ogni nodo di confine, capisco se è possibile raggiungere l'ultimo nodo riparato, nel caso sia possibile e il valore del flusso inviabile sia maggiore di 0, analizzo quanto flusso potrei inviare da parte di source, analizzando il valore di InFlow del nodo precedente e la capacità/flusso dell'arco. Nel caso siano entrambi positivi, ho trovato un percorso.

Nel caso in cui non sia riuscito a riparare tutti i nodi, analizzo il nodo che non è stato possibile riparare $noValidNode$: se è il nodo destinazione t, lo inserisco nella coda, altrimenti, inserisco nella coda tutti i nodi $m|m.label + 1 = noValidNode.label$,e cancello le informazioni di indirizzamento tutti i nodi con label pari o superiore a $noValidNode$

2.4 ricerca del percorso

Finché entrambe le code non sono vuote, analizzo gli elementi di una coda, analizzo i nodi a loro collegati, aggiornandoli e inserendoli in una coda **buffer** se non sono stati precedentemente esplorati. nel caso in cui trovi un nodo valido esplorato dalla parte opposta , significa che ho trovato un cammino. Una volta analizzati tutti i nodi, eseguo uno swap tra buffer e la coda utilizzata, per proseguire con la coda utilizzata per esplorati i nodi della parte opposta.

2.5 RepairNode

prendo in input un nodo e un booleano per capire da quale parte deve essere considerato nel caso in cui tratto un nodo "di confine"

nel caso un cui node sia stato esplorato da source, cerco tra i nodi a lui collegati un nodo tale che sia valido, con InFlow positivo, con capacità/flusso positivo e label uguale a quella del nodo -1 (nodeInput.label = nodoEsplorato.label + 1) nel caso lo trovo, salvo, aggiorno i dati e confermo che ho riparato il nodo.

nel caso in cui il nodo in input sia stato esplorato da sink, devo controllare anche il booleano nel caso il valore sia di confine.

Se il nodo era di confine e il booleano mi dice che deve esplorare la parte di source, cerco se tra i nodi appartenenti a lastSourceNodes è possibile ripararlo, altrimenti cerco un nodo valido, con InFlow positivo, con flusso/capacità positiva e con label pari a quella del nodo in ingresso -1 (nodeInput.label = nodoEsplorato.label+1)

nel caso riesco a trovarlo, aggiorno i dati e confermo di averlo trovato.