

algoritmi

Filippo Magi

December 23, 2021

1 Algoritmo senza ottimizzazione

Algorithm 1 Ricerca del massimo flusso a costo minimo senza alcuna ottimizzazione

Require: grafo dei residui $\overset{\leftrightarrow}{G} = \{V(G), E(G) \cup \{\overset{\leftarrow}{e} : e \in E(G)\}\}$.
Ensure: valore del flusso massimo
s \leftarrow SourceNode di $\overset{\leftrightarrow}{G}$
t \leftarrow SinkNode di $\overset{\leftrightarrow}{G}$
loop
 f \leftarrow DoBfs(grafo)
 if f = 0 **then**
 break
 end if
 mom \leftarrow t
 while mom \neq s **do**
 aggiorno capacità o flusso del nodo m a seconda del suo predecessore
 mom \leftarrow predecessore di mom
 end while
end loop
return flusso uscente da s

2 Ottimizzazione sono nelle ultime label

3 Propagazione della malattia

Algorithm 2 Algoritmo DoBfs senza alcuna ottimizzazione

Require: grafo dei residui $\vec{\bar{G}}$
Ensure: quantità di flusso inviata, grafo dei residui $\vec{\bar{G}}$ aggiornato per cammino f aumentante
cancella informazioni precedenti presenti in $V(G)$
coda \leftarrow Coda di nodi
coda.Enqueue(SourceNode di $\vec{\bar{G}}$)
while coda non è vuota **do**
 element \leftarrow coda.Dequeue
 for all arco **edge** che esce dal nodo element **do**
 n \leftarrow nodo che entra da edge
 if nodo n non è già stato visitato AND è possibile inviare(o ritirare) del flusso **then**
 aggiorno dati di n (nodo precedente, label, flusso entrante)
 if n è il SinkNode di $\vec{\bar{G}}$ **then**
 return flusso entrante in n
 else
 coda.Enqueue(n)
 end if
 end if
 end for
end while
return 0

Algorithm 3 Ricerca del massimo flusso a costo minimo con ricalcolo solo nelle ultime label

Require: grafo dei residui $\overset{\leftrightarrow}{G} = \{V(G), E(G) \cup \{\overset{\leftarrow}{e} : e \in E(G)\}\}$.
Ensure: valore del flusso massimo

$s \leftarrow \text{SourceNode di } \overset{\leftrightarrow}{G}$
 $t \leftarrow \text{SinkNode di } \overset{\leftrightarrow}{G}$
 $\text{noCap} \leftarrow \text{null}$

loop
 $f \leftarrow \text{DoBfs}(\text{grafo}, \text{noCap})$
 if $f = 0$ **then**
 break
 end if
 $\text{mom} \leftarrow t$
 while $\text{mom} \neq s$ **do**
 aggiorno capacità o flusso del nodo m a seconda del suo predecessore
 if capacità dell'arco $(m, \text{predecessore di } m) = 0$ **then**
 $\text{noCap} \leftarrow \text{mom}$
 end if
 $\text{mom} \leftarrow \text{predecessore di mom}$
 end while
end loop
return flusso uscente da s

Algorithm 4 Algoritmo DoBfs con ottimizzazione solo nelle ultime label

Require: grafo dei residui \vec{G} , nodo noCap
Ensure: valore del flusso inviato al SinkNode, \vec{G} aggiornato

```
coda  $\leftarrow$  coda vuota di nodi
if noCap = null then
    coda.Enqueue(source nodo di  $\vec{G}$ )
else
    rendo invalido noCap
    provo a riparare noCap /*controllo se c'è un nodo con label = noCap.label-1
    e con capacità o flusso diversa da 0*/
    t  $\leftarrow$  SinkNode di  $\vec{G}$ 
    if noCap è stato riparato AND il flusso entrante nel predecessore di t  $\neq$  0
    AND capacità dell'arco (t, predecessore di t) > 0 then
        return Min(flusso entrante di t, flusso passante per noCap)
    end if
    coda  $\leftarrow$  nodi di  $\vec{G}$  con label = (noCap.label - 1)
    cancello informazioni contenute nei nodi con label  $\leq$  noCap.label
    for all nodo n  $\in$  coda do
        controllo che flusso inviato di n sia legale con predecessori di n { analizzo
        se il predecessore abbia flusso entrate  $\geq$  flusso entrante di n, in questo
        caso termina, altrimenti continua coi i predecessori, da valutare se deve
        arrivare fino a n o se basta che incontri il primo nodo con flusso legale,
        CorrectFlow nel codice}
    end for
end if
while la coda non è vuota do
    element  $\leftarrow$  coda.dequeue()
    for all edge che entrano o escono da element do
        n  $\leftarrow$  nodo dove edge entra
        p  $\leftarrow$  nodo dove edge esce
        if element è valido (cioè la label è corretta) then
            if p = element AND capacità di edge > 0 AND (n non è stato esplorato
            OR n non è valido) then
                aggiorni dati di n {label, flusso entrante e nodo precedente, nel caso
                sia necessario "riparo" il nodo}
                if n è SinkNode di  $\vec{G}$  then
                    return flusso entrante di n
                else
                    coda.Enqueue(n)
                end if
            if n = element AND flusso di edge > 0 AND (p non è stato esplorato
            OR p non è valido) then
                aggiorni i dati di p
                aggiorni edge indicando che deve essere percorso in senso opposto
                if p è SinkNode di  $\vec{G}$  then
                    return flusso entrante di p
                else
                    coda.Enqueue(p)
                end if
            end if
        end if
    end for
end while
return 0
```

Algorithm 5 Ricerca del massimo flusso a costo minimo con propagazione della malattia

Require: grafo dei residui $\vec{\overleftarrow{G}} = \{V(G), E(G) \cup \{\overleftarrow{e} : e \in E(G)\}\}$.

Ensure: valore del flusso massimo

$s \leftarrow \text{SourceNode di } \vec{\overleftarrow{G}}$

$t \leftarrow \text{SinkNode di } \vec{\overleftarrow{G}}$

$\text{noCap} \leftarrow \text{null}$

loop

$f \leftarrow \text{DoBfs}(\text{grafo}, \text{noCap})$

if $f = 0$ **then**

break

end if

$\text{mom} \leftarrow t$

while $\text{mom} \neq s$ **do**

 aggiorno capacità o flusso del nodo m a seconda del suo predecessore

if capacità dell'arco $(m, \text{predecessore di } m) = 0$ **then**

$\text{noCap} \leftarrow \text{mom}$

end if

$\text{mom} \leftarrow \text{predecessore di mom}$

end while

end loop

return flusso uscente da s

Algorithm 6 DoBfs con propagazione della malattia

Require: grafo dei residui \vec{G} , nodo noCap
Ensure: flusso inviato al nodo t, \vec{G} aggiornato
coda \leftarrow coda vuota di nodi
if noCap = null **then**
 coda.enqueue(sourceNode di \vec{G})
else
 t \leftarrow SinkNode di \vec{G}
 provo a riparare noCap {controllo se c'è un nodo con label = noCap.label-1
 e con capacità o flusso diversa da 0}
 if noCap è stato riparato AND il flusso entrante nel predecessore di t $\neq 0$
 AND capacità dell'arco (t, predecessore di t) > 0 **then**
 return Min(flusso entrante di t, flusso passante per noCap)
 end if
 v \leftarrow SickPropagation(grafo,noCap,coda)
 if v $\neq 0$ **then**
 return v
 end if
 cancello il flusso entrante nei nodi con label \leq noCap.label
 if coda è vuota **then**
 coda \leftarrow nodi di \vec{G} con label = (noCap.label - 1)
 end if
 for all nodo n \in coda **do**
 recupera flusso entrante dall'ultimo nodo che lo ha {Dato che è possibile
 che il flusso entrante sia uguale a zero dovuto a SickPropagation, torno
 indietro fino al primo nodo con flusso entrante positivo, quindi, ricorsi-
 vamente, inserisco il flusso entrante per tutti i nodi che ho visitato}
 end for
end if
while la coda non è vuota **do**
 element \leftarrow coda.Dequeue()
 for all edge \leftarrow archi che escono ed entrano in element **do**
 n \leftarrow nodo che entra da edge
 p \leftarrow nodo che esce da edge
 if n ed p sono stati visitati (se il nodo è invalido è considerato non
 visitato) **then**
 continue
 end if
 if p = element AND capacità di edge > 0 AND (n è successivo a p OR
 n non è valido OR noCap = null) **then**
 aggiorno i dati di n (label, flusso entrante, nodo precedente, nel caso
 riparo il nodo)
 if n è SinkNode di \vec{G} **then**
 return flusso entrante in n
 else
 coda.enqueue(n)
 end if
 else if n = element AND flusso di edge > 0 AND p non è stato visitato
 then
 aggiorno i dati di p
 if p è SinkNode di \vec{G} **then**
 return flusso entrante in n
 else
 coda.enqueue(p)
 end if
 end if
 end for
end while

Algorithm 7 SickPropagation

Require: grafo dei residui $\vec{\vec{G}}$, Nodo node, coda di nodi
Ensure: possibile flusso inviato verso t (partendo da n, conoscendo i valori antecedenti a n), 0 altrimenti
malati \leftarrow coda di nodi
malati.Enqueue(node);
while malati non è vuota **do**
 m \leftarrow malati.Dequeue()
 provo a riparare m
 if il nodo non è stato riparato **then**
 malati.enqueue(nodo con previousNode = m)
 else if m è SinkNode di $\vec{\vec{G}}$ **then**
 v \leftarrow ricorsivamente, dal sourceNode di $\vec{\vec{G}}$ ottengo il flusso massimo che posso inviare, salvando il flusso entrante per ogni arco (recuperato tramite previousNode) che attraverso
 return v
 else
 coda.enqueue(m)
 end if
end while
return 0
