

algoritmi

Filippo Magi

January 3, 2022

1 Algoritmo senza ottimizzazione

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

Require: grafo dei residui $\vec{G} = \{V(G), E(G) \cup \{\vec{e}^- : e \in E(G)\}\}$.
Ensure: valore del flusso massimo
s \leftarrow SourceNode di \vec{G}
t \leftarrow SinkNode di \vec{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, usato per la variante senza salvare i nodi con label vicina

Algorithm 2 Algoritmo DoBfs senza alcuna ottimizzazione

Require: grafo dei residui $\vec{\vec{G}}$

Ensure: quantità di flusso inviata, grafo dei residui $\vec{\vec{G}}$ aggiornato per cammino f aumentante

cancella informazioni precedenti presenti in $V(G)$

coda \leftarrow Coda di nodi

coda.Enqueue(SourceNode di $\vec{\vec{G}}$)

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 $\vec{\vec{G}}$ **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 $\vec{\bar{G}} = \{V(G), E(G) \cup \{\bar{e} : e \in E(G)\}\}$.
Ensure: valore del flusso massimo

```

s ← SourceNode di  $\vec{\bar{G}}$ 
t ← SinkNode di  $\vec{\bar{G}}$ 
noCap ← null
loop
  f ← DoBfs(grafo,noCap)
  if f = 0 then
    break
  end if
  mom ← t
  while mom ≠ s do
    aggiorno capacità o flusso del nodo m a seconda del suo predecessore
    if capacità dell'arco (m, predecessore di m) = 0 then
      noCap ← mom
    end if
    mom ← predecessore di mom
  end while
end loop
return flusso uscente da s

```

4 Propagazione della malattia, usato per la variante salvando i nodi con label vicina

5 Shortest Augmenting Path

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 $\overset{\leftrightarrow}{G} = \{V(G), E(G) \cup \{\overset{\leftarrow}{e} : e \in E(G)\}\}$.

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 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, aggiornando i dati}
    if noCap è stato riparato then
        f  $\leftarrow$  flusso massimo consentito nel percorso da t a s
        if f  $\neq$  0 then
            return f
        end if
    end if
    v  $\leftarrow$  SickPropagation(grafo,noCap,coda)
    if v  $\neq$  0 then
        return v
    end if
    confermo di non aver esplorato i nodi con label  $\leq$  noCap.label
    if coda è vuota then
        coda  $\leftarrow$  nodi di  $\vec{G}$  con label = (noCap.label - 1)
    end if
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
            aggiorni i dati di n (label,visitato, nodo precedente, nel caso riparo il
            nodo)
            if n è SinkNode di  $\vec{G}$  then
                return flusso massimo da s a t, secondo il percorso scelto (dai vari
                previousNode all'interno dei nodi)
            else
                coda.enqueue(n)
            end if
        else if n = element AND flusso di edge > 0 AND p non è stato visitato
        then
            aggiorni i dati di p
            if p è SinkNode di  $\vec{G}$  then
                return flusso massimo da s a t, secondo il percorso scelto
            else
                coda.enqueue(p)
            end if
        end if
    end for
end while
return 0
```

Algorithm 7 SickPropagation

Require: grafo dei residui $\vec{\overleftarrow{G}}$, 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 $\vec{\overleftarrow{G}}$ **then**
 v \leftarrow ricorsivamente, dal t vado a recuperare il flusso massimo inviabile secondo il percorso deciso
 return v
 else
 coda.enqueue(m)
 end if
end while
return 0

Algorithm 8 Ricerca del massimo flusso a costo minimo con propagazione della malattia

Require: grafo dei residui $\vec{\overleftarrow{G}} = \{V(G), E(G) \cup \{\vec{e} : e \in E(G)\}\}$.
Ensure: valore del flusso massimo
s \leftarrow SourceNode di $\vec{\overleftarrow{G}}$
t \leftarrow SinkNode di $\vec{\overleftarrow{G}}$
noCap \leftarrow null
loop
 f \leftarrow DoBfs(grafo, noCap)
 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
 if capacità dell'arco (m, predecessore di m) = 0 **then**
 noCap \leftarrow mom
 end if
 mom \leftarrow predecessore di mom
 end while
end loop
return flusso uscente da s

Algorithm 9 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  $\leftarrow$  code.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
            aggiorni 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
            aggiorni 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 10 SickPropagation

Require: grafo dei residui \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{G} **then**
 v \leftarrow ricorsivamente, dal sourceNode di \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

Algorithm 11 Shortest Augmenting Path

Require: grafo dei residui \vec{G}
Ensure: flusso massimo inviato, \vec{G} aggiornato
s \leftarrow nodo sorgente di \vec{G}
t \leftarrow nodo destinazione di \vec{G}
eseguo bfs, che parte da t, assegnando a ogni nodo la distanza da t
invio flusso nel percorso deciso dalla bfs
flussoInviato \leftarrow flusso inviato (dal percorso deciso dalla bfs)
while distanza tra t e s \leq numero di nodi di \vec{G} **do**
 f \leftarrow Dfs(\vec{G} , s, $+\infty$)
 if f \neq 0 **then**
 invia flusso f nel grafo nel percorso indicato
 else
 break
 end if
 flussoInviato \leftarrow f + flussoInviato
end while
return flussoInviato

Algorithm 12 Dfs per trovare il flusso massimo in Shortest Augmenting Path

Require: grafo dei residui $\overset{\leftrightarrow}{G}$, Nodo start, valore f
Ensure: valore del flusso inviabile, percorso percorribile per poter inviare il flusso prima indicato

if distanza tra nodo start e t \leq numero dei nodi presenti in $\overset{\leftrightarrow}{G}$ **then**
 for all arco edge che entra o esce nel nodo start **do**
 if edge è un arco uscente da start ed edge è ammissibile (distanza tra i nodi = 1 e capacità residua positiva) **then**
 aggiorno dati di n
 $f \leftarrow \min(f, \text{capacità di edge})$
 if n è il nodo di destinazione di $\overset{\leftrightarrow}{G}$ **then**
 return f
 end if
 return Dfs($\overset{\leftrightarrow}{G}$, n, f)
 else if edge è arco entrante verso start ed edge è ammissibile (distanza tra i nodi = 1 e flusso inviato positivo) **then**
 aggiorno dati di p
 $f \leftarrow \min(f, \text{flusso di edge})$
 if p è il nodo di destinazione di $\overset{\leftrightarrow}{G}$ **then**
 return f
 end if
 return Dfs($\overset{\leftrightarrow}{G}$, p, f)
 end if
 end for
 $\min \leftarrow +\infty$
 for all arco edge che entra in start **do**
 if flusso di edge è positivo **then**
 $\min \leftarrow \min(\min, \text{flusso di edge})$
 end if
 end for
 for all arco edge che esce da start **do**
 if capacità di edge è positiva **then**
 $\min \leftarrow \min(\min, \text{capacità di edge})$
 end if
 end for
 distanza tra start e t $\leftarrow \min$
 if start è nodo sorgente di $\overset{\leftrightarrow}{G}$ **then**
 return Dfs($\overset{\leftrightarrow}{G}$, start, f)
 else
 return Dfs($\overset{\leftrightarrow}{G}$, predecessore di start, f)
 end if
end if
return 0
