algoritmi

Filippo Magi

December 29, 2021

1 Algoritmo senza ottimizzazione

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

Require: grafo dei residui $\overleftrightarrow{G} = \{V(G), E(G) \cup \{\overleftarrow{e} : e \in E(G)\}$.
Ensure: valore del flusso massimo
  s $\leftarrow$ SourceNode di $\overleftrightarrow{G}$
  t $\leftarrow$ SinkNode di $\overleftrightarrow{G}$
  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

4 Shortest Augmenting Path

Algorithm 2 Algoritmo DoBfs senza alcuna ottimizzazione

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

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

```
cancella informazioni precedenti presenti in V(G)
coda ← Coda di nodi
coda.Enqueue(SoruceNode di G↔)
while coda non è vuota do
  element ← coda.Dequeue
  for all arco edge che esce dal nodo element do
    n ← 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 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 $\overleftrightarrow{G} = \{V(G), E(G) \cup \{\overleftarrow{e} : e \in E(G)\}$.
Ensure: valore del flusso massimo

```
s ← SourceNode di G↔
t ← SinkNode di 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,predecossore di m) = 0 then
      noCap ← mom
    end if
    mom ← 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 $\overleftrightarrow{G}$, nodo noCap
Ensure: valore del flusso inviato al SinkNode, $\overleftrightarrow{G}$ aggiornato
coda ← coda vuota di nodi
if noCap = null **then**
coda.Enqueue(source nodo di $\overleftrightarrow{G}$
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 ← SinkNode di $\overleftrightarrow{G}$
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 ← nodi di $\overleftrightarrow{G}$ 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 precedessori 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 ← coda.dequeue()
for all edge che entrano o escono da element **do**
n ← nodo dove edge entra
p ← 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**
aggiorno dati di n {label, flusso entrante e nodo precedente, nel caso sia necessario "riparo" il nodo}
if n è SinkNode di $\overleftrightarrow{G}$ **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**
aggiorno i dati di p
aggiorno edge indicando che deve essere percorso in senso opposto
if p è SinkNode di $\overleftrightarrow{G}$ **then**
return flusso entrante di p
else
coda.Enqueue(p)
end if
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 $\overleftrightarrow{G} = \{V(G), E(G) \cup \{\overleftarrow{e} : e \in E(G)\}$.
Ensure: valore del flusso massimo

```
s ← SourceNode di G↔
t ← SinkNode di 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,predecossore di m) = 0 then
      noCap ← mom
    end if
    mom ← predecessore di mom
  end while
end loop
return flusso uscente da s
```

Algorithm 6 DoBfs con propagazione della malattia

Require: grafo dei residui $\overleftrightarrow{G}$, nodo noCap
Ensure: flusso inviato al nodo t, $\overleftrightarrow{G}$ aggiornato
coda ← coda vuota di nodi
if noCap = null **then**
coda.enqueue(sourceNode di $\overleftrightarrow{G}$)
else
t ← SinkNode di $\overleftrightarrow{G}$
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 ← 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 ← nodi di $\overleftrightarrow{G}$ 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, ricorsivamente, inserisco il flusso entrante per tutti i nodi che ho visitato}
end for
end if
while la coda non è vuota **do**
element ← code.Dequeue()
for all edge ← archi che escono ed entrano in element **do**
n ← nodo che entra da edge
p ← 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 $\overleftrightarrow{G}$ **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 $\overleftrightarrow{G}$ **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 $\overleftrightarrow{G}$, Nodo node, coda di nodi
Ensure: possibile flusso inviato verso t (partendo da n,conoscendo i valori antecedenti a n), 0 altrimenti
malati ← coda di nodi
malati.Enqueue(node);
while malati non è vuota **do**
m ← 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 $\overleftrightarrow{G}$ **then**
v ← ricorsivamente, dal sourceNode di $\overleftrightarrow{G}$ 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 8 Shortest Augmenting Path

Require: grafo dei reisidui $\overleftrightarrow{G}$
Ensure: flusso massimo inviato, $\overleftrightarrow{G}$ aggiornato
s ← nodo sorgente di $\overleftrightarrow{G}$
t ← nodo destinazione di $\overleftrightarrow{G}$
eseguo bfs, che parte da t, assegnando a ogni nodo la distanza da t
invio flusso nel percorso deciso dalla bfs
flussoInviato ← flusso inviato (dal percorso deciso dalla bfs)
while distanza tra t e s ¡ numero di nodi di $\overleftrightarrow{G}$ **do**
f ← Dfs($\overleftrightarrow{G}, s, +\infty$)
if f $ne0$ **then**
invia flusso f nel grafo nel percorso indicato
else
break
end if
flussoInviato ← f + flussoInviato
end while
return flussoInviato

Algorithm 9 Dfs per trovare il flusso massimo in Shortest Augmenting Path

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

```
if distanza tra nodo start e t ¡ numero dei nodi presenti in 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 ← min(f,capacità di edge)
      if n è il nodo di destinazione di G↔ then
        return f
      end if
      return Dfs(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 ← min(f, flusso di edge)
      if p è il nodo di destinazione di G↔ then
        return f
      end if
      return Dfs(G↔,p,f)
    end if
  end for
  min ← +∞
  for all arco edge che entra in start do
    if flusso di edge è positivo then
      min ← min(min, flusso di edge)
    end if
  end for
  for all arco edge che esce da start do
    if capacità di edge è positiva then
      min ← min(min, capacità di edge)
    end if
  end for
  distanza tra start e t ← min
  if start è nodo sorgente di G↔ then
    return Dfs(G↔,start,f)
  else
    return Dfs(G↔,predecessore di start,f)
  end if
end if
return 0
```