algoritmi bidirezionali

Filippo Magi

February 25, 2022

1 Ottimizzazione sugli ultimi livelli

1.1 FlowFordFulkerson

1.2 DoBfs

Algorithm 1 Ricerca del flusso massimo

```
Require: rete $(G, u, s, t)$
Ensure: valore del flusso massimo
 1: $vuotiSouce \leftarrow$ pila di nodi
 2: $vuotiSink \leftarrow$ pila di nodi
 3: $fMax \leftarrow 0$
 4: $vuotiSouce.push(s)$
 5: $vuotiSink.push(t)$
 6: while TRUE do
 7:   $(f, n) \leftarrow$ DoBfs(G, vuotiSource,vuotiSink)
 8:   if $f = 0$ then
 9:     break
10:   end if
11:   $vuotiSouce$.Clear()
12:   $vuotiSink$.Clear()
13:   $n$.flussoPassante $\leftarrow$ $n$.flussoPassante + f
14:   $momSource \leftarrow n$
15:   $momSink \leftarrow n$
16:   while $momSource \neq s$ do
17:     $momSource$.previousEdge.addFlow($f$)
18:     if $u_f(momSource$.previousEdge $) < 0 \vee f(momSource$.previousEdge $)< 0$ then
19:       $vuotiSource$.Clear()
20:       $flowError \leftarrow$GetFlow($s, n$).flussoPassante
21:       $mom \leftarrow n$
22:       while $mom \neq momsource.previousNode$ do
23:         $mom$.flussoPassante $\leftarrow$ $mom$.flussoPassante - flowError
24:         $mom$.PreviousEdge.addFlow($flowError$)
25:         $mom \leftarrow mom$.previousNode
26:       end while
27:       $vuotiSource$.push(momSource)
28:       $momSource.valid \leftarrow$ false
29:       $f \leftarrow f + flowError$
30:     else if $u_f(momSource$.previousEdge$) = 0$ then
31:       $momSource.valid \leftarrow$ false
32:       $vuotiSource.push(momSource)$
33:     end if
34:     $momSource$.flussoPassante $\leftarrow$ $momSource$.flussoPassante-f
35:     $momSource \leftarrow momSource$.previousNode
36:   end while
```

```
  **while** $momSink \neq t$ **do**
    $momSink$.nextEdge.addFlow($f$)
    **if** $u_f$($momSink$.nextEdge)$< 0 \vee f$($momSink$.nextEdge)$< 0$ **then**
      $vuotiSink$.Clear()
      $flowError \leftarrow$GetFlow($t, n$).flussoPassante
      $mom \leftarrow n$
      **while** $mom \neq momsink.nextNode$ **do**
        $mom$.flussoPassante $\leftarrow$ $mom$.flussoPassante - flowError
        $mom$.nextEdge.addFlow($flowError$)
        $mom \leftarrow mom$.nextNode
      **end while**
      $mom \leftarrow n$
      **while** $mom \neq s$ **do**
        $mom$.flussoPassante $\leftarrow$ $mom$.flussoPassante - flowError
        $mom$.PreviousEdge.addFlow($flowError$)
        $mom \leftarrow mom$.previousNode
      **end while**
      $vuotiSink$.Push(momSink)
      $momSink.valid \leftarrow$ false
      $f \leftarrow f + flowError$
    **else if** $u_f$($momSink$.nextEdge) $= 0$ **then**
      $momSink.valid \leftarrow$ false
      $vuotiSource.push(momSource)$
    **end if**
    $momSink$.flussoPassante $\leftarrow$ $momSink$.flussoPassante $-f$
    $momSink \leftarrow momSink$.nextNode
  **end while**
  $fMax \leftarrow fMax + f$
**end while**
**return** $fMax$
```

Algorithm 2 DoBfs con ottimizzazione sugli ultimi livelli

```
Require: rete $(G, u, s, t)$, $noCapsSource, noCapsSink$, cioè pile di nodi contenenti nodi non più raggiungibili attraverso il cammino trovato
Ensure: valore del flusso inviabile, nodo appartenente LastSinkNodes, cioè tutti i nodi che sono intermedi che fanno da ponte tra le due ricerche.
 1: $codaSource \leftarrow$ coda di nodi vuota
 2: $codaSink \leftarrow$ coda di nodi vuota
 3: $buffer \leftarrow$ coda di nodi vuota
 4: $sinkRepaired \leftarrow$ false
 5: if $\neg noCapsSource.isEmpty$ then
 6:   $p \leftarrow$ null
 7:   $repaired \leftarrow$ true
 8:   while $\neg noCapsSource.isEmpty$ do
 9:     $noCapSource \leftarrow noCapsSource$.pop()
10:     GetFlow($p, noCapSource$)
11:     $p \leftarrow noCapSource$
12:     Repair($noCapSource$)
13:     if non riesco a riparare $noCapSource$ then
14:       $noCapsSource$.Push($noCapSource$)
15:       $repaired \leftarrow$ false
16:       break
17:     end if
18:   end while
```

```
    **if** $\neg noCapsSink.isEmpty \wedge repaired$ **then**
        **for all** $n \in$ LastSinkNodes $|n.valid$ **do**
            GetFlow($noCapSource, n$) {da n cerco di retrocedere verso noCapSource, aggiornando ricorsivamente le informazioni dei nodi in modo opportuno (sopratutto per quanto riguarda n)}
            **if** GetFlow ha trovato un percorso$\wedge n.flussoPassante \neq 0$ **then**
                **if** $edge.reversed$ **then**
                    **return** $(\min(n.flussoPassante, f(edge)), n)$
                **else**
                    **return** $(\min(n.flussoPassante, u_f(edge)), n)$
                **end if**
            **end if**
        **end for**
    **end if**
    **if** $\neg repaired$ **then**
        **if** $noCapSource = s$ **then**
            $codaSource$.enqueue($noCapSource$)
        **else if** $noCapSource \in LastSinkNodes$ **then**
            $codaSource \leftarrow$ LastSourceNodes {nodi collegati ai nodi di LastSinkNodes}
        **else**
            **for all** $n \in V(G)|n.sourceSide \wedge n.label + 1 = noCapSource.label$ **do**
                $codaSource$.enqueue($n$)
            **end for**
            **for all** $n \in V(G)|n.SourceSide \wedge n.label \geq noCapSource.label$ **do**
                $n$.reset()
            **end for**
        **end if**
    **end if**
**end if**
```

```
**if** $\neg noCapsSink.isEmpty$ **then**
    $repaired \leftarrow$ true
    $p \leftarrow$ null
    **while** $\neg noCapsSink.isEmpty$ **do**
        $noCapsSink \leftarrow noCapsSink.$pop()
        GetFlow($p, noCapSink$)
        $p \leftarrow noCapSink$
        Repair($noCapSink$)
        **if** non riesco a riparare $noCapSink$ **then**
            noCapsSink.push($p$)
            $repaired \leftarrow$ false
            **break**
        **end if**
    **end while**
    **if** $repaired \wedge noCapsSource.isEmpty$ **then**
        **for all** $n \in LastSinkNodes | n.valid$ **do** {nodo di confine valido}
            **if** $n.previousEdge.reversed$ **then**
                $sourceFlow \leftarrow \min(n.previousNode.inFlow, f(n.previousEdge))$
            **else**
                $sourceFlow \leftarrow \min(n.previousNode.inFlow, u_f(n.previousEdge))$
            **end if**
            GetFlow($p, n$)
            **if** è stato trovato un percorso tra $p$ ed $n$ $\wedge n.flussoPassante \neq 0 \wedge sourceFlow > 0$ **then**
                **return** $(\min(n.flussoPassante, sourceFlow), n)$
            **end if**
        **end for**
    **end if**
    **if** $\neg repaired$ **then**
        **if** $noCapSink = t$ **then**
            $codaSink.$enqueue($noCapSink$)
        **else**
            **for all** $n \in V(G) | n.label + 1 = noCapSink.label$ **do**
                $codaSink.$enqueue($n$)
            **end for**
            **for all** $n \in N(G) | \neg n.sourceSide \wedge n.label \geq noCapSink.label$ **do**
                $n.$reset()
            **end for**
        **end if**
    **end if**
**end if**
```

```
while $\neg codaSink.isEmpty \lor \neg codaSource.isEmpty$ do
  if $\neg codaSource.isEmpty \land \neg noCapsSource.isEmpty$ then
    $element \leftarrow codaSource$.dequeue()
    if $\neg element.sourceSide \lor \neg element.valid$ then
      continue
    end if
    for all $edge \in element.Edges$ do
      $p \leftarrow edge$.previousNode
      $n \leftarrow edge$.nextNode
      if $element = p \land u_f(edge) > 0$ then
        if $n.flussoPassante \neq 0$ then
          if $n.sourceside$ then {esplorato da source}
            continue
          else
            $f \leftarrow \min(n.flussoPassante, p.flussoPassante, u_f(edge))$
            if $f = 0$ then
              continue
            end if
            $n$.update($p, edge$)
            $edge.reversed \leftarrow$ false
            $LastNodesSinkSide$.add($n$){di conseguenza inserisco tutti i nodi collegati direttamente a n che fanno parte di SourceSide in LastNodesSourceSide}
            return $(f, n)$
          end if
        end if
        $n$.update($p, edge$)
        $codaSource$.enqueue($n$)
      **else if** $element = n \wedge f(edge) > 0$ **then**
        **if** $p.flussoPassante \neq 0$ **then**
          **if** $p.sourceside$ **then**
            **continue**
          **else**
            $f \leftarrow \min(n.flussoPassante, p.flussoPassante, f(edge))$
            **if** $f = 0$ **then**
              **continue**
            **end if**
            $p$.update($n, edge$)
            $edge.reversed \leftarrow$ true
            **return** $(f, p)$
          **end if**
        **end if**
        $p$.update($n, edge$)
        $edge.reversed \leftarrow$ true
        $codaSource$.enqueue($p$)
      **end if**
    **end for**
  **end if**
```

```
if $\neg codaSink.isEmpty \land \neg noCapsSink.isEmpty)$ then
  $element \leftarrow codaSink$.dequeue()
  if $element.sourceSide \lor \neg element.valid$ then
    continue
  end if
  for all $edge \in element.Edges$ do
    $p \leftarrow edge$.previousNode
    $n \leftarrow edge$.nextNode
    if $element = n \land u_f(edge) > 0$ then
      if $p.flussoPassante \neq 0$ then
        if $\neg p.sourceSide$ then
          continue
        else
          $f \leftarrow \min(n.flussoPassante, p.flussoPassante, u_f(edge))$
          if $f = 0$ then
            continue
          end if
          $n$.update$(p, edge)$
          $edge.reversed \leftarrow$false
          return $(f, n)$
        end if
      end if
      $p$.update$(n, edge)$
      $edge.reversed \leftarrow$ false
      $codaSink$.enqueue$(p)$
      **else if** $element = p \land f(edge) > 0$ **then**
        **if** $p.flussoPassante \neq 0$ **then**
          **if** $\neg p$.sourceSide **then**
            **continue**
          **else**
            $f \leftarrow \min(p.flussoPassante, n.flussoPassante, f(edge))$
            **if** $f = 0$ **then**
              **continue**
            **end if**
            $p$.update($n, edge$)
            $edge.reversed \leftarrow$ true
            **return** $(f, p)$
          **end if**
        **end if**
        $n$.update($p, edge$)
        $edge.reversed \leftarrow$ true
        $codaSink$.enqueue($n$)
      **end if**
    **end for**
  **end if**
**end while**
**return** $(0, null)$
```