

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:    $(f, n) \leftarrow \text{DoBfs}(G, \text{vuotiSource}, \text{vuotiSink})$ 
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  $u_f(\text{momSource.previousEdge}) < 0 \vee f(\text{momSource.previousEdge}) < 0$  then
19:      vuotiSource.Clear()
20:      flowError  $\leftarrow \text{GetFlow}(s, n).\text{flussoPassante}$ 
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  $u_f(\text{momSource.previousEdge}) = 0$  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
```

```

37: while momSink  $\neq$  t do
38:   momSink.nextEdge.addFlow(f)
39:   if  $u_f(\textit{momSink.nextEdge}) < 0 \vee f(\textit{momSink.nextEdge}) < 0$  then
40:     vuotiSink.Clear()
41:     flowError  $\leftarrow$  GetFlow(t, n).flussoPassante
42:     mom  $\leftarrow$  n
43:     while mom  $\neq$  momSink.nextNode do
44:       mom.flussoPassante  $\leftarrow$  mom.flussoPassante - flowError
45:       mom.nextEdge.addFlow(flowError)
46:       mom  $\leftarrow$  mom.nextNode
47:     end while
48:     mom  $\leftarrow$  n
49:     while mom  $\neq$  s do
50:       mom.flussoPassante  $\leftarrow$  mom.flussoPassante - flowError
51:       mom.PreviousEdge.addFlow(flowError)
52:       mom  $\leftarrow$  mom.previousNode
53:     end while
54:     vuotiSink.Push(momSink)
55:     momSink.valid  $\leftarrow$  false
56:     f  $\leftarrow$  f + flowError
57:   else if  $u_f(\textit{momSink.nextEdge}) = 0$  then
58:     momSink.valid  $\leftarrow$  false
59:     vuotiSource.push(momSource)
60:   end if
61:   momSink.flussoPassante  $\leftarrow$  momSink.flussoPassante - f
62:   momSink  $\leftarrow$  momSink.nextNode
63: end while
64: fMax  $\leftarrow$  fMax + f
65: end while
66: 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
```

```

19:  if  $\neg noCapsSink.isEmpty \wedge repaired$  then
20:    for all  $n \in LastSinkNodes$   $| n.valid$  do
21:       $GetFlow(noCapSource, n)$  {da n cerco di retrocedere verso noCap-
        Source, aggiornando ricorsivamente le informazioni dei nodi in modo oppor-
        tuno (soprattutto per quanto riguarda n)}
22:      if  $GetFlow$  ha trovato un percorso  $\wedge n.flussoPassante \neq 0$  then
23:        if  $edge.reversed$  then
24:          return  $(\min(n.flussoPassante, f(edge)), n)$ 
25:        else
26:          return  $(\min(n.flussoPassante, u_f(edge)), n)$ 
27:        end if
28:      end if
29:    end for
30:  end if
31:  if  $\neg repaired$  then
32:    if  $noCapSource = s$  then
33:       $codaSource.enqueue(noCapSource)$ 
34:    else if  $noCapSource \in LastSinkNodes$  then
35:       $codaSource \leftarrow LastSourceNodes$  {nodi collegati ai nodi di LastSin-
        kNodes}
36:    else
37:      for all  $n \in V(G) | n.sourceSide \wedge n.label + 1 = noCapSource.label$ 
        do
38:         $codaSource.enqueue(n)$ 
39:      end for
40:      for all  $n \in V(G) | n.SourceSide \wedge n.label \geq noCapSource.label$  do
41:         $n.reset()$ 
42:      end for
43:    end if
44:  end if
45: end if

```

```

46: if  $\neg noCapsSink.isEmpty$  then
47:    $repaired \leftarrow \text{true}$ 
48:    $p \leftarrow \text{null}$ 
49:   while  $\neg noCapsSink.isEmpty$  do
50:      $noCapsSink \leftarrow noCapsSink.pop()$ 
51:      $GetFlow(p, noCapSink)$ 
52:      $p \leftarrow noCapSink$ 
53:      $Repair(noCapSink)$ 
54:     if non riesco a riparare  $noCapSink$  then
55:        $noCapsSink.push(p)$ 
56:        $repaired \leftarrow \text{false}$ 
57:       break
58:     end if
59:   end while
60:   if  $repaired \wedge noCapsSource.isEmpty$  then
61:     for all  $n \in LastSinkNodes|n.valid$  do {nodo di confine valido}
62:       if  $n.previousEdge.reversed$  then
63:          $sourceFlow \leftarrow \min(n.previousNode.inFlow, f(n.previousEdge))$ 
64:       else
65:          $sourceFlow \leftarrow \min(n.previousNode.inFlow, u_f(n.previousEdge))$ 
66:       end if
67:        $GetFlow(p, n)$ 
68:       if è stato trovato un percorso tra  $p$  ed  $n \wedge n.flussoPassante \neq 0 \wedge$ 
 $sourceFlow > 0$  then
69:         return  $(\min(n.flussoPassante, sourceFlow), n)$ 
70:       end if
71:     end for
72:   end if
73:   if  $\neg repaired$  then
74:     if  $noCapSink = t$  then
75:        $codaSink.enqueue(noCapSink)$ 
76:     else
77:       for all  $n \in V(G)|n.label + 1 = noCapSink.label$  do
78:          $codaSink.enqueue(n)$ 
79:       end for
80:       for all  $n \in N(G)|\neg n.sourceSide \wedge n.label \geq noCapSink.label$  do
81:          $n.reset()$ 
82:       end for
83:     end if
84:   end if
85: end if

```

```

86: while  $\neg codaSink.isEmpty \vee \neg codaSource.isEmpty$  do
87:   if  $\neg codaSource.isEmpty \wedge \neg noCapsSource.isEmpty$  then
88:      $element \leftarrow codaSource.dequeue()$ 
89:     if  $\neg element.sourceSide \vee \neg element.valid$  then
90:       continue
91:     end if
92:     for all  $edge \in element.Edges$  do
93:        $p \leftarrow edge.previousNode$ 
94:        $n \leftarrow edge.nextNode$ 
95:       if  $element = p \wedge u_f(edge) > 0$  then
96:         if  $n.flussoPassante \neq 0$  then
97:           if  $n.sourceside$  then {esplorato da source}
98:             continue
99:           else
100:             $f \leftarrow \min(n.flussoPassante, p.flussoPassante, u_f(edge))$ 
101:            if  $f = 0$  then
102:              continue
103:            end if
104:             $n.update(p, edge)$ 
105:             $edge.reversed \leftarrow false$ 
106:             $LastNodesSinkSide.add(n)$  {di conseguenza inserisco tutti i
            nodi collegati direttamente a n che fanno parte di SourceSide in LastN-
            odesSourceSide}
107:            return  $(f, n)$ 
108:          end if
109:        end if
110:         $n.update(p, edge)$ 
111:         $codaSource.enqueue(n)$ 

```

```

112:      else if  $element = n \wedge f(edge) > 0$  then
113:          if  $p.flussoPassante \neq 0$  then
114:              if  $p.sourceside$  then
115:                  continue
116:              else
117:                   $f \leftarrow \min(n.flussoPassante, p.flussoPassante, f(edge))$ 
118:                  if  $f = 0$  then
119:                      continue
120:                  end if
121:                   $p.update(n, edge)$ 
122:                   $edge.reversed \leftarrow \text{true}$ 
123:                  return  $(f, p)$ 
124:              end if
125:          end if
126:           $p.update(n, edge)$ 
127:           $edge.reversed \leftarrow \text{true}$ 
128:           $codaSource.enqueue(p)$ 
129:      end if
130:  end for
131: end if

```

```

132: if  $\neg codaSink.isEmpty \wedge \neg noCapsSink.isEmpty$  then
133:    $element \leftarrow codaSink.dequeue()$ 
134:   if  $element.sourceSide \vee \neg element.valid$  then
135:     continue
136:   end if
137:   for all  $edge \in element.Edges$  do
138:      $p \leftarrow edge.previousNode$ 
139:      $n \leftarrow edge.nextNode$ 
140:     if  $element = n \wedge u_f(edge) > 0$  then
141:       if  $p.flussoPassante \neq 0$  then
142:         if  $\neg p.sourceSide$  then
143:           continue
144:         else
145:            $f \leftarrow \min(n.flussoPassante, p.flussoPassante, u_f(edge))$ 
146:           if  $f = 0$  then
147:             continue
148:           end if
149:            $n.update(p, edge)$ 
150:            $edge.reversed \leftarrow \text{false}$ 
151:           return  $(f, n)$ 
152:         end if
153:       end if
154:        $p.update(n, edge)$ 
155:        $edge.reversed \leftarrow \text{false}$ 
156:        $codaSink.enqueue(p)$ 

```

```

157:     else if  $element = p \wedge f(edge) > 0$  then
158:         if  $p.flussoPassante \neq 0$  then
159:             if  $\neg p.sourceSide$  then
160:                 continue
161:             else
162:                  $f \leftarrow \min(p.flussoPassante, n.flussoPassante, f(edge))$ 
163:                 if  $f = 0$  then
164:                     continue
165:                 end if
166:                  $p.update(n, edge)$ 
167:                  $edge.reversed \leftarrow \text{true}$ 
168:                 return  $(f, p)$ 
169:             end if
170:         end if
171:          $n.update(p, edge)$ 
172:          $edge.reversed \leftarrow \text{true}$ 
173:          $codaSink.enqueue(n)$ 
174:     end if
175: end for
176: end if
177: end while
178: return  $(0, null)$ 

```
