

algoritmi bidirezionali

Filippo Magi

January 29, 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: vuotoSouce  $\leftarrow s$ 
2: vuotoSink  $\leftarrow t$ 
3: fMax  $\leftarrow 0$ 
4: while TRUE do
5:    $(f, n) \leftarrow \text{DoBfs}(G, \text{vuotoSource}, \text{vuotoSink})$ 
6:   if  $f = 0$  then
7:     break
8:   end if
9:    $fMax \leftarrow fMax + f$ 
10:  vuotoSouce  $\leftarrow \text{null}$ 
11:  vuotoSink  $\leftarrow \text{null}$ 
12:   $n.\text{flussoPassante} \leftarrow n.\text{flussoPassante} + n.\text{flussoPassante}$ 
13:  momSource  $\leftarrow n$ 
14:  momSink  $\leftarrow n$ 
15:  while momSource  $\neq s$  do
16:    momSource.previousEdge.addFlow( $f$ )
17:    if  $u(\text{momSource.previousEdge}) < 0$  OR  $f(\text{momSource.previousEdge}) < 0$  then
18:      vuotoSource  $\leftarrow \text{momsource}$ 
19:      momsource.revert() {faccio tornare i dati a come erano prima
dell'aggiornamento (fMax, valori di flusso passante e archi modificati)}
20:      sourceReverted  $\leftarrow \text{true}$ 
21:       $fMax \leftarrow fMax - f$ 
22:      break
23:    else
24:      if  $u(\text{momSource.previousEdge}) = 0$  then
25:        momSource.valid = false
26:        vuotoSource  $\leftarrow \text{momSource}$ 
27:      end if
28:      momSource.flussoPassante  $\leftarrow \text{momSource.flussoPassante} - f$ 
29:      momSource  $\leftarrow \text{momSource.previousNode}$ 
30:    end if
31:  end while
32:  if  $\neg \text{sourceReverted}$  then
33:    while momSink  $\neq t$  do
34:      momSink.nextEdge.addFlow( $f$ )
35:      if  $u(\text{momSink.nextEdge}) < 0$  OR  $f(\text{momSink.nextEdge}) < 0$  then
36:        vuotoSink  $\leftarrow \text{momSink}$ 
37:        momSink.revert()
38:        break
39:      else
40:        if  $u(\text{momSink.nextEdge}) = 0$  then
41:          momSink.valid  $\leftarrow \text{false}$ 
42:          vuotoSink  $\leftarrow \text{momSink}$ 
43:           $fMax \leftarrow fMax - f$ 
44:        end if
45:        momSink.flussoPassante  $\leftarrow \text{momSink.flussoPassante} - f$ 
46:        momSink  $\leftarrow \text{momSink.nextNode}$ 
47:      end if
48:    end while
49:  end if
50: end while
51: return fMax
```

Algorithm 2 DoBfs con ottimizzazione sugli ultimi livelli

Require: rete (G, u, s, t) , $noCapSource$, $noCapSink$, cioè nodi, rispettivamente della parte sorgente e della parte destinazione, che non sono più raggiungibili dal percorso deciso precedentemente

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: if  $noCapSource \neq \text{null}$  then
4:   Repair( $noCapSource$ )
5:   if riesco a riparare  $noCapSource$  then
6:     if  $noCapSink = \text{null}$  then
7:       for all  $n \in \text{LastSinkNodes} \mid n.valid$  do
8:         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)}
9:         if percorso legale tra  $n$  e  $noCapSource$  AND  $\forall e \mid e \in \text{percorso tra}$ 
             $n$  e  $noCapSource, u(e) > 0$  then
10:          return ( $n.flussoPassante, n$ )
11:        end if
12:      end for
13:    else
14:       $sourceRepaired \leftarrow \text{true}$ 
15:    end if
16:  end if
17:  if  $noCapSource = s$  then
18:     $coda.enqueue(noCapSource)$ 
19:  else if  $noCapSource \in \text{LastSinkNodes}$  then
20:     $codaSource \leftarrow \text{LastSourceNodes}$  {nodi collegati ai nodi di LastSinkN-
      odes}
21:  else
22:     $codaSource \leftarrow$  nodi esplorati da source con label =  $noCapSource.label -$ 
      1
23:    for all  $n \in N(G) \mid$  esplorati da source AND  $n.label \geq$ 
       $noCapSource.label$  do
24:       $n.reset()$ 
25:    end for
26:  end if
27: end if
```

```

28: if noCapSink  $\neq$  null then
29:   Repair(noCapSink)
30:   if riesco a riparare noCapSink then
31:     sinkRepaired  $\leftarrow$  true
32:     if sourceRepaired OR noCapSource = null then
33:       for all  $n \in N(G) | n.\text{valid AND } n \in \text{LastSinkNodes}$  do
34:         if sourceRepaired then
35:           if da  $n$  posso ricorsivamente retrocedere verso noCapSource
           (GetFlow) then
36:             sourceFlow  $\leftarrow n.\text{flussoPassante}$ 
37:           else
38:             continue
39:           end if
40:         else
41:           sourceFlow  $\leftarrow \min(n.\text{previousNode.flussoPassante}, u(n.\text{PreviousEdge})$ 
           {nel caso in cui arco.reversed = true, devo controllare il flusso e non la
           capacit  residua} )
42:         end if
43:         if sourceFlow > 0 AND  $n$  pu  retrocedere ricorsivamente verso
           noCapSink(Getflow) AND  $n.\text{flussoPassante} > 0$  then
44:           return ( $\min(n.\text{flussoPassante}, \text{sourceFlow}), n$ )
45:         end if
46:       end for
47:     end if
48:   end if
49:   if noCapSink =  $t$  then
50:     codaSink.enqueue(noCapSink)
51:   else
52:     codaSink  $\leftarrow$  nodi esplorati da sink con label = noCapSink.label-1
53:     for all  $n \in N(G) |$  esplorati da sink AND  $n.\text{label} \geq \text{noCapSink.label}$ 
     do
54:        $n.\text{reset}()$ 
55:     end for
56:   end if
57: end if

```

```

58: while  $\neg codaSink.isEmpty$  OR  $\neg codaSource.isEmpty$  do
59:   if  $\neg codaSource.isEmpty$  AND ( $noCapSource \neq null$  OR
       $\neg sourceRepaired$ ) then
60:      $element \leftarrow codaSource.dequeue()$ 
61:     if  $\neg element.sourceSide$  OR  $\neg element.valid$  then
62:       continue
63:     end if
64:     for all  $edge \in element.Edges$  do
65:        $p \leftarrow edge.previousNode$ 
66:        $n \leftarrow edge.nextNode$ 
67:       if  $element = p$  AND  $u(edge) > 0$  then
68:         if  $n.visited$  then
69:           if  $n.sourceside$  (esplorato da source) then
70:             continue
71:           else{in questo caso ho le due parti che si incontrano}
72:              $f \leftarrow \min(n.flussoPassante, p.flussoPassante, u(edge))$ 
73:             if  $f = 0$  then
74:               continue
75:             end if
76:              $n.update(p, edge)$ 
77:              $edge.reversed \leftarrow false$ 
78:             LastNodesSinkSide.add( $n$ ) {di conseguenza inserisco tutti i
              nodi collegati direttamente a  $n$  che fanno parte di SourceSide in LastN-
              odesSourceSide}
79:             return ( $f, n$ )
80:           end if
81:         end if
82:         if  $\neg n.sourceSide$  AND  $n \neq t$  then
83:            $sinkRepaired \leftarrow false$ 
84:           for all  $node \in N(G) | \neg node.sourceSide$  AND  $node.label =$ 
            ( $n.label - 1$ ) do
85:              $codaSink.enqueue(node)$ 
86:           end for
87:           for all  $node \in N(G) | \neg node.sourceSide$  AND  $node.label \geq$ 
             $n.label$  do
88:              $node.reset()$ 
89:           end for
90:           continue
91:         end if
92:          $n.update(p, edge)$ 
93:          $codaSource.enqueue(n)$ 

```

```

94:      else if element = n AND  $f(\textit{edge}) > 0$  then
95:          if p.visited then
96:              if p.sourceside then
97:                  continue
98:              else
99:                   $f \leftarrow \min(n.\textit{flussoPassante}, p.\textit{flussoPassante}, f(\textit{edge}))$ 
100:                  if  $f = 0$  then
101:                      continue
102:                  end if
103:                  p.update(n, edge)
104:                  edge.reversed  $\leftarrow$  true
105:                  return (f, p)
106:              end if
107:          end if
108:          if  $\neg p.\textit{sourceSide}$  AND  $p \neq t$  then
109:              sinkRepaired  $\leftarrow$  false
110:              for all node  $\in V(G) | \neg \textit{node.sourceNode}$  AND node.label =
111:                  (p.label - 1) do
112:                      codaSink.enqueue(node)
113:              end for
114:              for all node  $\in V(G) | \neg \textit{node.sourceSide}$  AND node.label  $\geq$ 
115:                  p.label do
116:                      node.reset()
117:              end for
118:              continue
119:          end if
120:          p.update(n, edge)
121:          edge.reversed  $\leftarrow$  true
122:          codaSource.enqueue(p)
123:      end if

```

```

124:  if  $\neg codaSink.isEmpty$  AND ( $noCapSink \neq \text{null}$  OR  $\neg sinkRepaired$ )
      then
125:       $element \leftarrow codaSink.dequeue()$ 
126:      if  $element.sourceSide$  OR  $\neg element.valid$  then
127:          continue
128:      end if
129:      for all  $edge \in element.Edges$  do
130:           $p \leftarrow edge.previousNode$ 
131:           $n \leftarrow edge.nextNode$ 
132:          if  $element = n$  AND  $u(edge) > 0$  then
133:              if  $p.visited$  then
134:                  if  $\neg p.sourceSide$  then
135:                      continue
136:                  else
137:                      if  $sourceRepaired$  AND  $n$  può retrocedere ricorsivamente
                          verso  $noCapSink(GetFlow)$  AND  $n.flussoPassante > 0$  then
138:                           $f \leftarrow \min(n.nextNode.flussoPassante, n.flussoPassante)$ 
139:                           $f \leftarrow \min(f, n.PreviousNode.flussoPassante, u(n.nextEdge), u(n.previousEdge))$ 
                          {qui se  $edge.reversed = \text{true}$  va considerato il flusso  $f$  e non la capacità
                          residua  $u$ }
140:                          if  $f > 0$  then
141:                              return  $f$ 
142:                          end if
143:                      end if
144:                       $f \leftarrow \min(n.flussoPassante, p.flussoPassante, u(edge))$ 
145:                      if  $f = 0$  then
146:                          continue
147:                      end if
148:                       $n.update(p, edge)$ 
149:                       $edge.reversed \leftarrow \text{false}$ 
150:                      return  $(f, n)$ 
151:                  end if
152:              end if
153:              if  $p.sourceSide$  AND  $noCapSink \neq t$  then
154:                  continue
155:              end if
156:               $p.update(n, edge)$ 
157:               $edge.reversed \leftarrow \text{false}$ 
158:               $codaSink.enqueue(p)$ 

```

```

159:      else if element = p AND  $f(\textit{edge}) > 0$  then
160:          if p.visited then
161:              if  $\neg p.\textit{sourceSide}$  then
162:                  continue
163:              else
164:                  if sourceRepaired AND da p riesco a raggiungere noCap-
Source(GetFlow) then
165:                      return (p.flussoPassante, p)
166:                  end if
167:                   $f \leftarrow \min(p.\textit{flussoPassante}, n.\textit{flussoPassante}, f(\textit{edge}))$ 
168:                  if  $f = 0$  then
169:                      continue
170:                  end if
171:                  p.update(n, edge)
172:                  edge.reversed  $\leftarrow$  true
173:                  return (f, p)
174:              end if
175:          end if
176:          if n.sourceSide AND  $n \neq t$  then
177:              continue
178:          end if
179:          n.update(p, edge)
180:          edge.reversed  $\leftarrow$  true
181:          codaSink.enqueue(n)
182:      end if
183:  end for
184:  end if
185: end while
186: return (0, null)

```
