algoritmi bidirezionali

Filippo Magi

February 8, 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 ← s
2: vuotoSink ← t
3: fMax ← 0
4: while TRUE do
5:    (f, n) ← DoBfs(G, vuotoSource,vuotoSink)
6:    if f = 0 then
7:       break
8:    end if
9:    fMax ← fMax + f
10:   vuotoSouce ← null
11:   vuotoSink ← null
12:   n.flussoPassante ← n.flussoPassante + n.flussoPassante
13:   momSource ← n
14:   momSink ← n
15:   while momSource ≠ s do
16:      momSource.previousEdge.addFlow(f)
17:      if u(momSource.previousEdge ) < 0 OR f(momSource.previousEdge )< 0  then
18:         vuotoSource ← momsource
19:         momsource.revert() {faccio tornare i dati a come erano prima dell'aggiornamento (fMax, valori di flusso passante e archi modificati)}
20:         sourceReverted ← true
21:         fMax ← fMax − f
22:         break
23:      else
24:         if u(momSource.previousEdge) = 0 then
25:            momSource.valid = false
26:            vuotoSource ← momSource
27:         end if
28:         momSource.flussoPassante ← momSource.flussoPassante-f
29:         momSource ← momSource.previousNode
30:      end if
31:   end while
32:   if ¬sourceReverted then
33:      while  momSink ≠ t do
34:         momSink.nextEdge.addFlow(f)
35:         if u(momSink.nextEdge)< 0 OR f(momSink.nextEdge)< 0  then
36:            vuotoSink ← momSink
37:            momSink.revert()
38:            break
39:         else
40:            if  u(momSink.nextEdge) = 0 then
41:               momSink.valid ← false
42:               vuotoSink ← momSink
43:               fMax ← fMax − f
44:            end if
45:            momSink.flussoPassante ← momSink.flussoPassante −f
46:            momSink ← 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 raggiungili 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: $buffer \leftarrow$ coda di nodi vuota
4: **if** $noCapSource \neq$ null **then**
5: Repair($noCapSource$
6: **if** riesco a riparare $noCapSource$ **then**
7: **if** $noCapSink =$null **then**
8: **for all** $n \in$ LastSinkNodes $|n.valid$ **do**
9: GetFlow($noCapSource, n$) {da n cerco di retrocedere verso noCapSource, aggiornando ricorsivamente le informazioni dei nodi in modo opportuno (sopratutto per quanto riguarda n)}
10: **if** percorso legale tra n e $noCapSource$ AND $\forall e|e \in$ percorso tra n e $noCapSource, u(e) > 0$ **then**
11: **return** (n.flussoPassante,n)
12: **end if**
13: **end for**
14: **else**
15: $sourceRepaired \leftarrow$ true
16: **end if**
17: **end if**
18: **if** $noCapSource = s$ **then**
19: $coda$.enqueue($noCapSource$)
20: **else if** $noCapSource \in$ LastSinkNodes **then**
21: $codaSource \leftarrow$ LastSourceNodes {nodi collegati ai nodi di LastSinkNodes}
22: **else**
23: $codaSource \leftarrow$ nodi esplorati da source con label = noCapSource.label-1
24: **for all** $n \in N(G)|$esplorati da source AND $n.label \geq noCapSource.label$ **do**
25: n.reset()
26: **end for**
27: **end if**
28: **end if**

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

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

```
95:         else if element = n AND f(edge) > 0 then
96:           if p.visited then
97:             if p.sourceside then
98:               continue
99:             else
100:              f ← min(n.flussoPassante, p.flussoPassante, f(edge))
101:              if f = 0 then
102:                continue
103:              end if
104:              p.update(n,edge)
105:              edge.reversed ← true
106:              return (f, p)
107:            end if
108:          end if
109:          if ¬p.sourceSide AND p ≠ t then
110:            sinkRepaired ← false
111:            for all node ∈ V(G)|¬node.sourceNodeANDnode.label =
    (p.label − 1) do
112:              codaSink.enqueue(node)
113:            end for
114:            for all node ∈ V(G)|¬node.sourceSide AND node.label ≥
    p.label do
115:              node.reset()
116:            end for
117:            continue
118:          end if
119:          p.update(n, edge)
120:          edge.reversed ← true
121:          buffer.enqueue(p)
122:        end if
123:      end for
124:    end while
125:    mom ← codaSource
126:    codaSource ← buffer
127:    buffer ← mom
```

```
128:  while ¬codaSink.isEmptyAND (noCapSink ≠ null OR ¬sinkRepaired) do
129:    element ← codaSink.dequeue()
130:    if element.sourceSide OR ¬element.valid then
131:      continue
132:    end if
133:    for all edge ∈ element.Edges do
134:      p ← edge.previousNode
135:      n ← edge.nextNode
136:      if element = n AND u(edge) > 0 then
137:        if p.visited then
138:          if ¬p.sourceSide then
139:            continue
140:          else
141:            if sourceRepaired AND n può retrocedere ricorsivamente verso noCapSink(GetFlow) AND n.flussoPassante> 0 then
142:              f ← min(n.nextNode.flussoPassante, n.flussoPassante)
143:              f ← min(f, n.PreviousNode.flussoPassante, u(n.nextEdge), u(n.previuosEdge)) {qui se edge.reversed = true va considerato il flusso f e non la capacità residua u}
144:              if f > 0 then
145:                return f
146:              end if
147:            end if
148:            f ← min(n.flussoPassante, p.flussoPassante, u(edge))
149:            if f = 0 then
150:              continue
151:            end if
152:            n.update(p, edge)
153:            edge.reversed ←false
154:            return (f, n)
155:          end if
156:        end if
157:        if p.sourceSide AND noCapSink ≠ t then
158:          continue
159:        end if
160:        p.update(n, edge)
161:        edge.reversed ← false
162:        buffer.enqueue(p)
```

```
163:        else if element = pAND f(edge) > 0 then
164:          if p.visited then
165:            if ¬p.sourceSide then
166:              continue
167:            else
168:              if sourceRepaired AND da p riesco a raggiungere noCap-
    Source(GetFlow) then
169:                return (p.flussoPassante, p)
170:              end if
171:              f ← min(p.flussoPassante, n.flussoPassante, f(edge))
172:              if f = 0 then
173:                continue
174:              end if
175:              p.update(n, edge)
176:              edge.reversed ← true
177:              return (f, p)
178:            end if
179:          end if
180:          if n.sourceSide AND n ≠ t then
181:            continue
182:          end if
183:          n.update(p, edge)
184:          edge.reversed ← true
185:          buffer.enqueue(n)
186:        end if
187:      end for
188:    end while
189:    mom ← codaSink
190:    codaSink ← buffer
191:    buffer ← mom
192:  end while
193:  return (0, null)
```