algoritmi bidirezionali

Filippo Magi

February 18, 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** $mom \neq momsource$ **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**

```
37:   while momSink ≠ t do
38:     momSink.nextEdge.addFlow(f)
39:     if u_f(momSink.nextEdge)< 0 ∨ f(momSink.nextEdge)< 0 then
40:       vuotiSink.Clear()
41:       flowError ←GetFlow(t, n).flussoPassante
42:       mom ← n
43:       while mom ≠ momsink do
44:         mom.flussoPassante ← mom.flussoPassante - flowError
45:         mom.nextEdge.addFlow(flowError)
46:         mom ← mom.nextNode
47:       end while
48:       mom ← n
49:       while mom ≠ s do
50:         mom.flussoPassante ← mom.flussoPassante - flowError
51:         mom.PreviousEdge.addFlow(flowError)
52:         mom ← mom.previousNode
53:       end while
54:       vuotiSink.Push(momSink)
55:       momSink.valid ← false
56:       f ← f + flowError
57:     else if u_f(momSink.nextEdge) = 0 then
58:       momSink.valid ← false
59:       vuotiSource.push(momSource)
60:     end if
61:     momSink.flussoPassante ← momSink.flussoPassante −f
62:     momSink ← momSink.nextNode
63:   end while
64:   fMax ← 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: $sourceRepaired \leftarrow$ false
5: $sinkRepaired \leftarrow$ false
6: **if** $noCapsSource.isEmpty$ **then**
7: $p \leftarrow$ null
8: $repaired \leftarrow$ true
9: **while** $\neg noCapsSource.isEmpty$ **do**
10: $noCapSource \leftarrow noCapsSource$.pop()
11: GetFlow$(p, noCapSource)$
12: $p \leftarrow noCapSource$
13: Repair$(noCapSource)$
14: **if** non riesco a riparare $noCapSource$ **then**
15: $noCapsSource$.Push$(noCapSource)$
16: $repaired \leftarrow$ false
17: **break**
18: **end if**
19: **end while**

```
20:   if ¬noCapsSink.isEmpty ∧ repaired then
21:     for all n ∈ LastSinkNodes |n.valid  do
22:       GetFlow(noCapSource, n) {da n cerco di retrocedere verso noCapSource, aggiornando ricorsivamente le informazioni dei nodi in modo opportuno (sopratutto per quanto riguarda n)}
23:       if GetFlow ha trovato un percorso∧n.flussoPassante ≠ 0 then
24:         if edge.reversed then
25:           return (min(n.flussoPassante, f(edge)), n)
26:         else
27:           return (min(n.flussoPassante, u_f(edge)), n)
28:         end if
29:       end if
30:     end for
31:   end if
32:   sourceRepaired ← repaired
33:   if ¬repaired then
34:     if noCapSource = s then
35:       codaSource.enqueue(noCapSource)
36:     else if noCapSource ∈ LastSinkNodes then
37:       codaSource ← LastSourceNodes {nodi collegati ai nodi di LastSinkNodes}
38:     else
39:       for all n ∈ V(G)|n.sourceSide ∧ n.label + 1 = noCapSource.label do
40:         codaSource.enqueue(n)
41:       end for
42:       for all n ∈ V(G)|n.SourceSide ∧ n.label ≥ noCapSource.label do
43:         n.reset()
44:       end for
45:     end if
46:   end if
47: end if
```

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

```
88: while ¬codaSink.isEmpty ∨ ¬codaSource.isEmpty do
89:   while ¬codaSource.isEmpty ∧ ¬noCapsSource.isEmpty do
90:     element ← codaSource.dequeue()
91:     if ¬element.sourceSide ∨ ¬element.valid then
92:       continue
93:     end if
94:     for all edge ∈ element.Edges do
95:       p ← edge.previousNode
96:       n ← edge.nextNode
97:       if element = p ∧ u_f(edge) > 0 then
98:         if n.flussoPassante ≠ 0 then {il nodo è già stato visitato}
99:           if n.sourceside then
100:            continue
101:          else
102:            f ← min(n.flussoPassante, p.flussoPassante, u_f(edge))
103:            if f = 0 then
104:              continue
105:            end if
106:            n.update(p, edge)
107:            addLast(n) {aggiunge n a LastSinkNodes e i nodi a lui collegati sourceSide li inserisco in LastSourceNodes}
108:            edge.reversed ← false
109:            return (f, n)
110:          end if
111:        end if
112:        n.update(p, edge)
113:        buffer.enqueue(n)
```

```
114:        else if element = n ∧ f(edge) > 0 then
115:          if p.flussoPassante > 0 then
116:            if p.sourceside then
117:              continue
118:            else
119:              f ← min(n.flussoPassante, p.flussoPassante, f(edge))
120:              if f = 0 then
121:                continue
122:              end if
123:              p.update(n, edge)
124:              addLast(p)
125:              edge.reversed ← true
126:              return (f, p)
127:            end if
128:          end if
129:          p.update(n, edge)
130:          edge.reversed ← true
131:          buffer.enqueue(p)
132:        end if
133:      end for
134:    end while
135:    mom ← codaSource
136:    codaSource ← buffer
137:    buffer ← mom
```

```
138:  while ¬codaSink.isEmpty ∧ ¬noCapsSink.isEmpty do
139:    element ← codaSink.dequeue()
140:    if element.sourceSide ∨ ¬element.valid then
141:      continue
142:    end if
143:    for all edge ∈ element.Edges do
144:      p ← edge.previousNode
145:      n ← edge.nextNode
146:      if element = n ∧ u_f(edge) > 0 then
147:        if p.flussoPassante ≠ 0 then
148:          if ¬p.sourceSide then
149:            continue
150:          else
151:            f ← min(n.flussoPassante, p.flussoPassante, u_f(edge))
152:            if f = 0 then
153:              continue
154:            end if
155:            n.update(p, edge)
156:            addLast(n)
157:            edge.reversed ←false
158:            return (f, n)
159:          end if
160:        end if
161:        p.update(n, edge)
162:        edge.reversed ← false
163:        buffer.enqueue(p)
```

```
164:          else if element = p ∧ f(edge) > 0 then
165:            if n.flussoPassante ≠ 0 then
166:              if ¬n.sourceSide then
167:                continue
168:              else
169:                f ← min(p.flussoPassante, n.flussoPassante, f(edge))
170:                if f = 0 then
171:                  continue
172:                end if
173:                p.update(n, edge)
174:                addLast(p)
175:                edge.reversed ← true
176:                return (f, p)
177:              end if
178:            end if
179:            n.update(p, edge)
180:            edge.reversed ← true
181:            buffer.enqueue(n)
182:          end if
183:        end for
184:      end while
185:      mom ← codaSink
186:      codaSink ← buffer
187:      buffer ← mom
188:    end while
189:    return (0, null)
```