algoritmi bidirezionali

Filippo Magi

February 14, 2022

1 Senza alcuna ottimizzazione

1.1 FlowFordFulkerson

1.2 DoBfs

Algorithm 1 Ricerca del flusso massimo

Require: rete (G, u, s, t)
Ensure: valore del flusso massimo

```
1:  fMax ← 0
2:  vuotoSource ← true
3:  vuotoSource ← true
4:  while TRUE do
5:     nodo ← DoBfs(G,vuotoSource,vuotoSink)
6:     if nodo = null then
7:        break
8:     end if
9:     f ← GetFlow(nodo) {ripercorre da n verso s e t per recuperare il flusso}
10:    if f = 0 then
11:       break
12:    end if
13:    vuotoSource ← false
14:    vuotoSink ← false
15:    fMax ← fMax + f
16:    mom ← n
17:    while n ≠ s do
18:       n.PreviousEdge.AddFlow(f)
19:       if u(n.PreviousEdge)= 0 then
20:          vuotoSource ← true
21:       end if
22:       n ← n.previousNode
23:    end while
24:    while mom ≠ t do
25:       n.nextEdge.addFlow(f)
26:       if u(n.nextEdge) = 0 then
27:          vuotoSink ← true
28:       end if
29:       n.update(f) {n.InFlow -=f}
30:       n ← n.nextNode
31:    end while
32: end while
33: return fMax
```

Algorithm 2 DoBfs

Require: rete (G, u, s, t), booleano *sourceSide* , booleano *sinkSide*, per capire in quale parte del grafo devo operare

Ensure: nodo dove si incontrano i nodi esplorati da sink e quelli esplorati da source

1: $codaSource \leftarrow$ coda vuota di nodi
2: $codaSink \leftarrow$ coda vuota di nodi
3: $codaEgeSource \leftarrow$ coda vuota di archi
4: $codaEdgeSink \leftarrow$ coda vuota di archi
5: **if** $sourceSide \wedge sinkSide$ **then**
6: **for all** $n \in V(G)$ **do**
7: n.reset
8: **end for**
9: $codaSource$.enqueue(s)
10: $codaSink$.enqueue(t)
11: **else if** *souceSide* **then**
12: $codaSource$.enqueue(s)
13: **for all** $n \in V(G)|n.sourceSide$ **do**
14: n.Reset()
15: **end for**
16: $codaEdgeSink$.enqueue($null$)
17: **else if** *sinkSide* **then**
18: $codaSink$.enqueue(t)
19: **for all** $n \in V(G)|\neg n.sourceSide$ **do**
20: n.Reset()
21: **end for**
22: $codaEdgeSource$.enqueue($null$)
23: **end if**
24: **while** $\neg codaSink.isEmpty \vee \neg codaSource.isEmpty$ **do**
25: **if** $(\neg codaSource.isEmpty \wedge (codaEdgeSource.isEmpty \vee (codaSink.isEmpty \wedge codaEdgeSink.isEmpty))$ **then**
26: $elementSource \leftarrow codaSource$.dequeue()
27: **for all** $n \in V(G)|n$è esplorabile da $elementSource$ **do** { arco $x|(x.PreviousNode == elementSource \wedge x.Capacity > 0 \wedge (!x.NextNode.Visited \vee !x.NextNode.SourceSide)) \vee (x.NextNode == elementSource \wedge x.Flow > 0 \wedge (!x.PreviousNode.Visited \vee !x.PreviousNode.SourceSide)$}
28: $codaEdgeSource$.enqueue(n)
29: **end for**
30: **end if**
31: **if** $(\neg codaSink.isEmpty \wedge (codaEdgeSink.isEmpty \vee (codaSource.isEmpty \wedge codaEdgeSink.isEmpty)$ **then**
32: $elementSink \leftarrow codaSink.dequeueu$
33: **for all** $n \in V(G)|n$ è esplorabile da $elementSink$ **do** { arco $x|(x.NextNode == elementSink \wedge x.Capacity > 0 \wedge (!x.PreviousNode.Visited \vee x.PreviousNode.SourceSide)) \vee (x.PreviousNode == elementSink \wedge x.Flow > 0 \wedge (!x.NextNode.Visited \vee x.NextNode.SourceSide))$}
34: $codaEdgeSink$.enqueue(n)
35: **end for**
36: **end if**

```
37:  while ¬codaEdgeSource.isEmpty ∧ ¬codaEdgeSink.isEmpty do
38:    if sourceSide then
39:      sourceEdge ← codaEdgeSource.dequeue
40:      p ← sourceEdge.previousNode
41:      n ← sourceEdge.nextNode
42:      if elementSource = p ∧ u_f(sourceEdge) > 0 then
43:        if n.visited then
44:          if ¬n.sourceSide then
45:            n.update(p, sourceEdge)
46:            sourceEdge.Reversed← false
47:            return n
48:          end if
49:        else
50:          n.update(p, sourceEdge)
51:          sourceEdge.Reversed← false
52:          codaSource.enqueue(n)
53:        end if
54:      else if elementSource = n ∧ f(sourceEdge) > 0 then
55:        if p.visited then
56:          if ¬p.sourceSide then
57:            p.update(n, sourceEdge)
58:            sourceEdge.reversed ← false
59:            return p
60:          end if
61:        else
62:          p.update(n, sourceEdge)
63:          sourceEdge.reversed ← false
64:          codaSource.enqueue(p)
65:        end if
66:      end if
67:    end if
```

```
68:       if sinkSide then
69:         edgeSink ← codaEdgeSink.dequeue
70:         p ← edgeSink.previousNode
71:         n ← edgeSink.nextNode
72:         if elementSink = n ∧ u_f(edgeSink) > 0 then
73:           if p.visited then
74:             if ¬p.sourceSide then
75:               continue
76:             else
77:               n.update(p, edgeSink)
78:               edgeSink.reversed ←false
79:               return n
80:             end if
81:           end if
82:           p.update(n, edgeSource)
83:           edgeSink.reversed ← false
84:           codaSink.enqueue(p)
85:         else if elementSink = p ∧ f(elementSink) > 0 then
86:           if n.visited then
87:             if ¬n.sourceSide then
88:               continue
89:             else
90:               p.update(n, edgeSink)
91:               return p
92:             end if
93:           end if
94:           n.update(p, edgeSink)
95:           edgeSink.reversed ← true
96:           codaSink.enqueue(n)
97:         end if
98:       end if
99:     end while
100:  end while
101:  return null
```