algoritmi bidirezionali

Filippo Magi

February 8, 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:    (f, nodo) ← DoBfs(G,vuotoSource,vuotoSink)
6:    if f = 0 then
7:       break
8:    end if
9:    vuotoSource ← false
10:   vuotoSource ← false
11:   fMax ← fMax + f
12:   mom ← n
13:   while n ≠ s do
14:      n.PreviousEdge.AddFlow(f)
15:      if u(n.PreviousEdge)= 0 then
16:         vuotoSource ← true
17:      end if
18:      n.update(f) {n.InFlow -=f}
19:      n ← n.previousNode
20:   end while
21:   while mom ≠ t do
22:      n.nextEdge.addFlow(f)
23:      if e(n.nextEdge) = 0 then
24:         vuotoSink ← true
25:      end if
26:      n.update(f) {n.InFlow -=f}
27:      n ← n.nextNode
28:   end while
29: end while
30: return fMax
```

Algorithm 2 DoBfs : Ricerca un path tra s e x[], e da x[] a t, dove t[] sono i nodi intermedi dove si incontrano i due path

Require: rete (G, u, s, t), *sourceSide* e *sinkSide*, che sono dei booleani che chiariscono in quale parte si dovrà operare

Ensure: valore del flusso inviabile, nodo intermedio, cioè che tiene in memoria sia il nodo successivo, sia il nodo precedente

1: $codaSource \leftarrow$ coda di nodi
2: $codaSink \leftarrow$ coda di nodi
3: $buffer \leftarrow$ coda di nodi
4: **if** *sourceSide* **then**
5: **for all** $n \in V(G) | n$ è stato esplorato da s **do**
6: n.Reset()
7: **end for**
8: $codaSource$.enqueue(s)
9: **end if**
10: **if** *sinkSide* **then**
11: **for all** $n \in V(G) | n$ è stato esplorato da t **do**
12: n.Reset()
13: **end for**
14: $codaSink$.enqueue(t)
15: **end if**
16: {per motivi prestazioni si controlla nel codice se si hanno sia sinkSide sia sourceSide positivi per non analizzare tutti i nodi 2 volte}
17: **while** $\neg codaSource$.isEmpty OR $\neg codaSink$.iSEmpty **do**
18: **while** $\neg codaSource$.isEmpty **do**
19: $element \leftarrow codaSource$.dequeue()
20: **for all** $edge \in n$.Edges **do**
21: $p \leftarrow edge$.previousNode
22: $n \leftarrow edge$.nextNode
23: **if** $element = p$ AND $u(edge) > 0$ **then**
24: **if** n.visited **then**
25: **if** n è stato esplorato dalla parte di Source **then**
26: **continue**
27: **else**
28: $f \leftarrow \min(u(edge), p.flussoPassante, n.flussoPassante)$
29: **if** $f = 0$ **then**
30: **continue**
31: **end if**
32: n.update($p, edge, n$)
33: $edge$.reversed = false
34: **return** (f, n)
35: **end if**
36: **end if**
37: n.update($p, edge$)
38: $edge$.reversed = false
39: $buffer$.enqueue(n)
40: **end if**

```
41:         if element = n AND f(edge) > 0 then
42:           if p.visited then
43:             if p è stato già esplorato dalla parte di Source then
44:               continue
45:             else
46:               f ← min(n.flussoPassante, p.flussoPassante, f(edge)
47:               if f = 0 then
48:                 continue
49:               end if
50:               p.update(n, edge, p)
51:               edge.reversed = true
52:               return (f, p)
53:             end if
54:           end if
55:           p.update(n, edge)
56:           edge.reversed = true
57:           buffer.enqueue(p)
58:         end if
59:       end for
60:     end while
61:     mom ← codaSource
62:     codaSource ← buffer
63:     buffer ← mom
64:     while ¬codaSink.isEmpty do
65:       element ← codaSink.dequeue()
66:       for all edge ∈ element.Edges do
67:         p ← edge.previousNode
68:         n ← n.nextNode
69:         if element = n AND u(edge) > 0 then
70:           if p.visited then
71:             if p è stato esplorato dalla parte di Sink then
72:               continue
73:             else
74:               f ← min(p.flussoPassante, u(edge), n.flussoPassante)
75:               if f = 0 then
76:                 continue
77:               end if
78:               n.update(p, edge, n)
79:               edge.Reversed = false
80:               return (f, n)
81:             end if
82:           end if
83:           p.update(n, edge)
84:           edge.Reversed = false
85:           buffer.enqueue(p)
86:         end if
```

```
87:          if element = p AND f(edge) > 0 then
88:            if n.visited then
89:              if n è stato esplorato da Sink then
90:                continue
91:              else
92:                f ← min(n.flussoPassante, p.flussoPassante, f(edge)
93:                if f = 0 then
94:                  continue
95:                end if
96:                p.update(n, edge, p)
97:                edge.reversed = true
98:                return (f, p)
99:              end if
100:           end if
101:           n.update(p, edge)
102:           edge.reversed = true
103:           buffer.enqueue(n)
104:         end if
105:       end for
106:     end while
107:     mom ← buffer
108:     codaSink ← buffer
109:     buffer ← mom
110:   end while
111:   return (0, null)
```