algoritmi bidirezionali

Filippo Magi

February 18, 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: $fMax \leftarrow 0$
2: $vuotoSource \leftarrow$ true
3: $vuotoSource \leftarrow$ true
4: **while** TRUE **do**
5: $(f, nodo) \leftarrow$ DoBfs(G,vuotoSource,vuotoSink)
6: **if** $f = 0$ **then**
7: **break**
8: **end if**
9: $vuotoSource \leftarrow$ false
10: $vuotoSource \leftarrow$ false
11: $fMax \leftarrow fMax + f$
12: $mom \leftarrow n$
13: **while** $n \neq s$ **do**
14: n.PreviousEdge.AddFlow(f)
15: **if** $u(n$.PreviousEdge$)= 0$ **then**
16: $vuotoSource \leftarrow$ true
17: **end if**
18: n.update(f) {n.InFlow -=f}
19: $n \leftarrow n$.previousNode
20: **end while**
21: **while** $mom \neq t$ **do**
22: n.nextEdge.addFlow(f)
23: **if** $e(n$.nextEdge$) = 0$ **then**
24: $vuotoSink \leftarrow$ true
25: **end if**
26: n.update(f) $\{n.InFlow- = f\}$
27: $n \leftarrow n$.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 ← coda di nodi
2:  codaSink ← coda di nodi
3:  if sourceSide then
4:    for all n ∈ V(G)|n.sourceSide do
5:      n.Reset()
6:    end for
7:    codaSource.enqueue(s)
8:  end if
9:  if sinkSide then
10:   for all n ∈ V(G)|¬n.sourceSide do
11:     n.Reset()
12:   end for
13:   codaSink.enqueue(t)
14: end if
15: while ¬codaSource.isEmpty ∨ ¬codaSink.isEmpty  do
16:   if codaSource.isEmpty then
17:     element ← codaSource.dequeue()
18:     for all edge ∈ n.Edges do
19:       p ← edge.previousNode
20:       n ← edge.nextNode
21:       if element = p ∧ u_f(edge) > 0 then
22:         if n.flussoPassante ≠ 0 then {vuol dire che è stato precedentemente esplorato}
23:           if n.sourceSide then
24:             continue
25:           else
26:             f ← min(u_f(edge), p.flussoPassante, n.flussoPassante)
27:             if  f = 0 then
28:               continue
29:             end if
30:             n.update(p, edge, n)
31:             edge.reversed ← false
32:             return (f, n)
33:           end if
34:         end if
35:         n.update(p, edge)
36:         edge.reversed ← false
37:         codaSource.enqueue(n)
38:       end if
```

```
39:         if element = n ∧ f(edge) > 0 then
40:           if p.flussoPassante ≠ 0 then
41:             if p.sourceSide then
42:               continue
43:             else
44:               f ← min(n.flussoPassante, p.flussoPassante, f(edge)
45:               if f = 0 then
46:                 continue
47:               end if
48:               p.update(n, edge, p)
49:               edge.reversed ←true
50:               return (f, p)
51:             end if
52:           end if
53:           p.update(n, edge)
54:           edge.reversed ←true
55:           codaSource.enqueue(p)
56:         end if
57:       end for
58:     end if
59:     if ¬codaSink.isEmpty then
60:       element ← codaSink.dequeue()
61:       for all edge ∈ element.Edges do
62:         p ← edge.previousNode
63:         n ← edge.nextNode
64:         if element = n ∧ u_f(edge) > 0 then
65:           if p.flussoPassante ≠ 0 then
66:             if ¬p.sourceSide then
67:               continue
68:             else
69:               f ← min(p.flussoPassante, u_f(edge), n.flussoPassante)
70:               if f = 0 then
71:                 continue
72:               end if
73:               n.update(p, edge, n)
74:               edge.reversed ←false
75:               return (f, n)
76:             end if
77:           end if
78:           p.update(n, edge)
79:           edge.reversed ←false
80:           codaSink.enqueue(p)
81:         end if
```

```
82:         if element = p ∧ f(edge) > 0 then
83:           if n.flussoPassante ≠ 0 then
84:             if ¬n.sourceSide then
85:               continue
86:             else
87:               f ← min(n.flussoPassante, p.flussoPassante, f(edge)
88:               if f = 0 then
89:                 continue
90:               end if
91:               p.update(n, edge, p)
92:               edge.reversed ←true
93:               return (f, p)
94:             end if
95:           end if
96:           n.update(p, edge)
97:           edge.reversed ←true
98:           codaSink.enqueue(n)
99:         end if
100:      end for
101:    end if
102: end while
103: return (0, null)
```