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 ← 0
2:  vuotoSource ← true
3:  vuotoSource ← true
4:  while TRUE do
5:     n ← DoBfs(G, vuotoSource, vuotoSink)
6:     f ←GetFlow(n) {mi calcola il flusso massimo invabile nel percorso}
7:     if f = 0 then
8:        break
9:     end if
10:    vuotoSource ← false
11:    vuotoSource ← false
12:    fMax ← fMax + f
13:    mom ← n
14:    while n ≠ s do
15:       n.PreviousEdge.AddFlow(f)
16:       if u_f(n.PreviousEdge)= 0 then
17:          vuotoSource ← true
18:       end if
19:       n ← n.previousNode
20:    end while
21:    while mom ≠ t do
22:       mom.nextEdge.addFlow(f)
23:       if u_f(mom.nextEdge) = 0 then
24:          vuotoSink ← true
25:       end if
26:       mom ← mom.nextNode
27:    end while
28: end while
29: 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

```
codaSource ← coda di nodi
codaSink ← coda di nodi
buffer ← coda di nodi
if sourceSide then
    for all n ∈ V(G)|n è stato esplorato da s do
        n.Reset()
    end for
    codaSource.enqueue(s)
end if
if sinkSide then
    for all n ∈ V(G)|n è stato esplorato da t do
        n.Reset()
    end for
    codaSink.enqueue(t)
end if
while ¬codaSource.isEmpty ∨ ¬codaSink.iSEmpty  do
    while ¬codaSource.isEmpty do
        element ← codaSource.dequeue()
        for all edge ∈ n.Edges do
            p ← edge.previousNode
            n ← edge.nextNode
            if element = p ∧ u_f(edge) > 0 then
                if n.visited then
                    if n.sourceSide then
                        continue
                    else
                        n.update(p, edge, n)
                        edge.reversed ← false
                        return (f, n)
                    end if
                end if
                n.update(p, edge)
                edge.reversed ← false
                buffer.enqueue(n)
            end if
```

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

```
74:         if element = p ∧ f(edge) > 0 then
75:           if n.visited then
76:             if ¬n.sourceSide then
77:               continue
78:             else
79:               p.update(n, edge, p)
80:               edge.reversed ←true
81:               return (f, p)
82:             end if
83:           end if
84:           n.update(p, edge)
85:           edge.reversed ←true
86:           buffer.enqueue(n)
87:         end if
88:       end for
89:     end while
90:     mom ← buffer
91:     codaSink ← buffer
92:     buffer ← mom
93:   end while
94:   return (0, null)
```