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

```
fMax ← 0
vuotoSource ← true
vuotoSource ← true
while TRUE do
    nodo ← DoBfs(G,vuotoSource,vuotoSink)
    if nodo = null then
        break
    end if
    f ← GetFlow(nodo) {ripercorre da n verso s e t per recuperare il flusso}
    if f = 0 then
        break
    end if
    vuotoSource ← false
    vuotoSink ← false
    fMax ← fMax + f
    mom ← n
    while n ≠ s do
        n.PreviousEdge.AddFlow(f)
        if u(n.PreviousEdge)= 0 then
            vuotoSource ← true
        end if
        n ← n.previousNode
    end while
    while mom ≠ t do
        n.nextEdge.addFlow(f)
        if u(n.nextEdge) = 0 then
            vuotoSink ← true
        end if
        n.update(f) {n.InFlow -=f}
        n ← n.nextNode
    end while
end while
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

```
codaSource ← coda vuota di nodi
codaSink ← coda vuota di nodi
codaEgeSource ← coda vuota di archi
codaEdgeSink ← coda vuota di archi
if sourceSide ∧ sinkSide then
    for all n ∈ V(G) do
        n.reset
    end for
    codaSource.enqueue(s)
    codaSink.enqueue(t)
else if souceSide then
    codaSource.enqueue(s)
    for all n ∈ V(G)|n.sourceSide  do
        n.Reset()
    end for
    codaEdgeSink.enqueue(null)
else if sinkSide then
    codaSink.enqueue(t)
    for all n ∈ V(G)|¬n.sourceSide do
        n.Reset()
    end for
    codaEdgeSource.enqueue(null)
end if
while ¬codaSink.isEmpty ∨ ¬codaSource.isEmpty do
    if (¬codaSource.isEmpty ∧ codaEdgeSource.isEmpty) ∨ (codaSink.isEmpty ∧ codaEdgeSink.isEmpty then
        elementSource ← codaSource.dequeue()
        codaEdgeSource.enqueue(δ⁺(elementSource))
    end if
    if (¬codaSink.isEmpty ∧ codaEdgeSink.isEmpty) ∨ (codaSource.isEmpty ∧ codaEdgeSink.isEmpty) then
        elementSink ← codaSink.dequeueu
        codaEdgeSink.enqueue(δ⁻(elementSink)
    end if
```

```
while ¬codaEdgeSource.isEmpty ∧ ¬codaEdgeSink.isEmpty do
  if sourceSide then
    sourceEdge ← codaEdgeSource.dequeue
    p ← sourceEdge.previousNode
    n ← sourceEdge.nextNode
    if elementSource = p ∧ u_f(sourceEdge) > 0 then
      if n.visited then
        if ¬n.sourceSide then
          n.update(p, sourceEdge)
          sourceEdge.Reversed← false
          return n
        end if
      else
        n.update(p, sourceEdge)
        sourceEdge.Reversed← false
        codaSource.enqueue(n)
      end if
    else if elementSource = n ∧ f(sourceEdge) > 0 then
      if p.visited then
        if ¬p.sourceSide then
          p.update(n, sourceEdge)
          sourceEdge.reversed ← false
          return p
        end if
      else
        p.update(n, sourceEdge)
        sourceEdge.reversed ← false
        codaSource.enqueue(p)
      end if
    end if
  end if
```

```
    if sinkSide then
      edgeSink ← codaEdgeSink.dequeue
      p ← edgeSink.previousNode
      n ← edgeSink.nextNode
      if elementSink = n ∧ u_f(edgeSink) > 0 then
        if p.visited then
          if ¬p.sourceSide then
            continue
          else
            n.update(p, edgeSink)
            edgeSink.reversed ←false
            return n
          end if
        end if
        p.update(n, edgeSource)
        edgeSink.reversed ← false
        codaSink.enqueue(p)
      else if elementSink = p ∧ f(elementSink) > 0 then
        if n.visited then
          if ¬n.sourceSide then
            continue
          else
            p.update(n, edgeSink)
            return p
          end if
        end if
        n.update(p, edgeSink)
        edgeSink.reversed ← true
        codaSink.enqueue(n)
      end if
    end if
  end while
end while
return null
```