algoritmi bidirezionali

Filippo Magi

February 9, 2022

1 Ottimizzazione sono nelle ultime label

Algorithm 1 Ricerca del massimo flusso ricalcolando solo nelle ultime label

Require: Rete (G, u, s, t) .
Ensure: valore del flusso massimo
$noCap \leftarrow$ null
while TRUE **do**
$f \leftarrow$ DoBfs$(G, noCap)$
if $f = 0$ **then**
break
end if
$mom \leftarrow t$
while $mom \neq s$ **do**
$mom.PreviousEdge.AddFlow(t.flussoPassante)$
if $u(mom.PreviousEdge) < 0$ OR $f(mom.PreviousEdge) < 0$ **then**
mom.Reverse(t) {da t faccio tornare come le capacità e il flusso come prima di inviarle fino a mom}
$noCap \leftarrow mom$
break
end if
if $u(mom.PreviousEdge) = 0$ **then**
$noCap \leftarrow mom$
end if
$mom \leftarrow mom$.previousNode
end while
end while
return s.FlussoUscente

Algorithm 2 Algoritmo DoBfs con ottimizzazione solo nelle ultime label

Require: rete (G, u, s, t), nodo *noCap*
Ensure: valore del flusso inviato a t, $N(G)$ aggiornati con informazioni sul percorso da fare

```
coda ← coda vuota di nodi
if noCap =null then
  coda.Enqueue(s)
else
  noCap.Valid = false
  Repair(noCap){controllo se c'è un nodo con label = noCap.label-1 e con capacità o flusso diversa da 0}
  f ← GetPathFlow(t)
  if noCap.Valid ∧f ≠ 0 then
    return f
  end if
  coda ← n ∈ N(G)‖n.label = (noCap.label − 1)
  for all n ∈ N(G)‖n.label ≥ noCap.label do
    n.Reset()
  end for
end if
while la coda non è vuota do
  element ← coda.dequeue()
  if element.valid then {label valida}
    for all edge ∈ element.Edges do
      n ← edge.nextNode
      p ← edge.previousNode
      if  p = element AND u(edge) > 0 AND (¬n.visited OR ¬n.valid) then
        n.update(p,edge) {label, flusso entrante e nodo precedente, nel caso sia necessario "riparo" il nodo}
        edge.Reversed = false {per capire se devo inviare o ricevere flusso}
        if n = t then
          return n.flussoPassante
        else
          coda.Enqueue(n)
        end if
      else if n = element AND f(edge) > 0 AND (¬p.visited OR ¬p.valid) then
        p.update(n,edge)
        edge.reversed = true
        if p = t then
          return p.flussoPassante
        else
          coda.Enqueue(p)
        end if
      end if
    end for
  end if
end while
return 0
```