# SickPropagation

Filippo Magi

January 29, 2022

## 1 Propagazione della malattia

**Algorithm 1** Ricerca del massimo flusso con propagazione della malattia

**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**
      $m$.PreviousEdge.AddFlow$(f)$
      **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**
        no$Cap \leftarrow mom$
      **end if**
      $mom \leftarrow mom$.PreviousNode
    **end while**
  **end while**
  **return** flusso uscente da s

**Algorithm 2** DoBfs con propagazione della malattia

**Require:** rete $(G, u, s, t)$, nodo *noCap*
**Ensure:** flusso inviato al nodo t
  *coda* ← coda vuota di nodi
  **if** *noCap* =null **then**
    *coda*.enqueue($s$)
  **else**
    Repair(*noCap*) {controllo se c'è un nodo con label = noCap.label-1 e con capacità o flusso diversa da 0, aggiornando i dati}
    **if** *noCap* è stato riparato **then**
      $f \leftarrow$ GetFlow() {recupero il flusso inviabile nel percorso descritto tra s e t }
      **if** $f \neq 0$ **then**
        **return** $f$
      **end if**
    **end if**
    $v \leftarrow$ SickPropagation($G, noCap, coda$)
    **if** $v \neq 0$ **then**
      **return** $v$
    **end if**
    **for all** $n \in N(G) | n.label \geq noCap.label$ **do**
      $n$.Reset()
    **end for**
    **if** coda è vuota **then**
      $coda \leftarrow n \in G | n.label = (noCap.label - 1)$
    **end if**
  **end if**
  **while** *coda* non è vuota **do**
    $element \leftarrow coda$.Dequeue()
    **for all** $edge \leftarrow element$.Edges **do**
      $n \leftarrow edge$.nextNode
      $p \leftarrow edge$.previousNode
      **if** $n$.visited AND $p$.visited (se il nodo è invalido è considerato come non visitato) **then**
        **continue**
      **end if**
      **if** $p = element$ AND $u(edge) > 0$ AND ($n.label \geq p.label$ OR $\neg n.valid$ OR $noCap$ = null) **then**
        $n$.update(p,edge) (label,visitato, nodo precedente, nel caso riparo il nodo)
        **if** $n = t$ **then**
          **return** GetFlow()
        **else**
          $coda$.enqueue($n$)
        **end if**
      **else if** $n = element$ AND $f(edge) > 0$ AND $\neg p.visited$ **then**
        $p$.update(n,edge)
        **if** $p = t$ **then**
          **return** GetFlow()
        **else**
          $coda$.enqueue($p$)
        **end if**
      **end if**
    **end for**
  **end while**
  **return** 0

**Algorithm 3** SickPropagation

**Require:** rete $(G, u, s, t)$, Nodo*node*, *coda*
**Ensure:** possibile flusso inviato verso t (partendo da n,conoscendo i valori antecedenti a n), 0 altrimenti

```
malati ← coda vuota di nodi
malati.Enqueue(node);
while malati non è vuota do
    m ← malati.Dequeue()
    Repair(m)
    if m non è stato riparato then
        malati.enqueue(n|n.previousNode= m)
    else if m = t then
        return GetFlow()
    else
        coda.enqueue(m)
    end if
end while
return 0
```