algoritmi bidirezionali

Filippo Magi

January 29, 2022

1 ShortestAugmentingPath

1.1 FlowFordFulkerson

1.2 Dfs

Algorithm 1 Ricerca del flusso massimo

Require: rete $(G, u, s, t$

Ensure: valore del flusso massimo

1: $fMax \leftarrow$ Bfs(s){faccio partire da s una bfs, cercando un percorso e sopratutto indicando la distanza d_s}
2: sendFlow($t, fMax$) {invio il flusso dal percorso indicato tramite previousNode da t verso s con il valore fMax,nel mentre che procedo cancello le informazioni nei nodi esplorati (tranne la distanza)}
3: $f \leftarrow$ Bfs(t) {bfs da t verso s, trovo un percorso salvato da NextNode e sopratutto trovo la distanza d_t}
4: sendFlow(s, f)
5: $fMax \leftarrow f + fMax$
6: **for all** $n \in V(G)$ **do**
7: n.Reset() {cancello indicazioni su un possibile percorso da fare}
8: **end for**
9: $fso \leftarrow +\infty, fsi \leftarrow +\infty$
10: **while** $f \neq 0$ AND $d_s(t) > \#V(G)$ AND $d_t(s) > \#V(G)$ **do**
11: $(fso, fsi, startSource, startSink) \leftarrow$ Dfs $(G, startSource, startSink, fso, fsi)$
12: **if** $startSink = startSource$ AND $startSink \neq null$ **then**
13: $f \leftarrow \min(fso, fsi)$
14: sendFlow($startSink, f$)
15: $fso \leftarrow +\infty, fsi \leftarrow +\infty$
16: $startSource \leftarrow s, startSink \leftarrow t$
17: **else if** $startSink = s$ **then**
18: $f \leftarrow fsi$
19: sendFlow($startSink, f$)
20: $fsi \leftarrow +\infty$
21: $startSink \leftarrow t$
22: **else if** $startSource = t$ **then**
23: $f \leftarrow fso$
24: sendFlow($startSource, f$)
25: $fso \leftarrow +\infty$
26: $startsource \leftarrow s$
27: **else**
28: **break**
29: **end if**
30: $fMax \leftarrow f + fMax$
31: {valutare se serve fare un reset per ogni nodo presente nel grafo}
32: **end while**
33: **return** fMax

Algorithm 2 Dfs

Require: rete (G, u, s, t), *startSource* e *startSink* rispettivamente il nodo di partenza della parte di Source e di Sink, *sourceFlow* e *sinkFlow* rispettivamente il valore del flusso massimo inviabile dalla parte di Source e di Sink

Ensure: (massimo valore di flusso inviabile da parte di source, massimo valore di flusso inviabile da parte di Sink, Nodo di arrivo dalla parte di Source, nodo di arrivo dalla parte di Sink)

1: **if** $startSource = startSink$ **then**
2: **return** $(sourceFlow, sinkFlow, startSource, startSink)$
3: **end if**
4: **if** $d_s(startSink) < \#V(G)$ AND $d_t(startSource) < \#V(G)$ **then**
5: **for all** $edge \in startSource.Edges$ **do**
6: $n \leftarrow edge.NextNode$
7: $p \leftarrow edge.PreviousNode$
8: **if** $startSource = p$ AND $d_t(n) = d_t(p) - 1$ AND $u(edge) > 0$ **then**
9: $sourceFlow \leftarrow \min(sourceFlow, u(edge))$
10: $n.previousEdge \leftarrow edge$ {salvo anche il nodo precedente}
11: **if** $n = t$ **then**
12: **return** $(sourceFlow, sinkFlow, n, startSink)$
13: **end if**
14: **if** n è già stato precedentemente esplorato dalla parte di Sink **then**
15: **return** $(sourceFlow, sinkFlow, n, n)$
16: **end if**
17: **return** SinkDfs$(G, n, startSink, sourceFlow, sinkFlow)$
18: **end if**
19: **end for**
20: $minDistance \leftarrow +\infty$
21: **for all** $edge \in startSource.Edges$ **do**
22: **if** $edge.PreviousNode = startSource$ AND$u(edge) > 0$ **then**
23: $minDistance \leftarrow \min(minDistance, d_t(edge.nextNode))$
24: **end if**
25: **end for**
26: $d_t(startSource) \leftarrow minDistance + 1$
27: **if** $startSource = s$ **then**
28: $mom \leftarrow startsource$
29: **else**
30: $mom \leftarrow startSource.previousNode$
31: **end if**
32: $startSource.Reset()$
33: **return** Dfs$(G, mom, startSink, sourceFlow, sinkFlow)$
34: **end if**
35: **return** $(0, 0, null, null)$

Algorithm 3 SinkDfs

Require: rete (G, u, s, t), $startSource$ e $startSink$ rispettivamente il nodo di partenza della parte di Source e di Sink, $sourceFlow$ e $sinkFlow$ rispettivamente il valore del flusso massimo inviabile dalla parte di Source e di Sink

Ensure: (massimo valore di flusso inviabile da parte di source, massimo valore di flusso inviabile da parte di Sink, Nodo di arrivo dalla parte di Source, nodo di arrivo dalla parte di Sink)

if $startSource = startSink$ **then**
  return $(sourceFlow, sinkFlow, startSource, startSink)$
end if
if $d_s(startSink) < \#V(G)$ AND $d_t(startSource) < \#V(G)$ **then**
  for all $edge \in startSink.Edges$ **do**
    $n \leftarrow edge.NextNode$
    $p \leftarrow edge.PreviousNode$
    if $startSink = n$ AND $d_s(p) = d_s(n) - 1$ AND $u(edge) > 0$ **then**
      $sourceFlow \leftarrow \min(sinkFlow, u(edge))$
      $p.nextEdge \leftarrow edge$ {salvo anche il nodo precedente}
      if $p = s$ **then**
        return $(sourceFlow, sinkFlow, startSource, p)$
      end if
      if p è già stato precedentemente esplorato dalla parte di Source **then**
        return $(sourceFlow, sinkFlow, p, p)$
      end if
      return Dfs$G, startSource, p, sourceFlow, sinkFlow, s, t)$
    end if
  end for
  $minDistance \leftarrow +\infty$
  for all $edge \in startSink.Edges$ **do**
    if $edge.NextNode = startSink$ AND $u(edge) > 0$ **then**
      $minDistance \leftarrow \min(minDistance, d_s(edge.previousNode)$
    end if
  end for
  $d_s(startSink) \leftarrow minDistance + 1$
  if $startSink = t$ **then**
    $mom \leftarrow startSink$
  else
    $mom \leftarrow startSink.nextNode$
  end if
  $startSink.Reset()$
  return SinkDfs$(G, startSource, mom, sourceFlow, sinkFlow)$
end if
return $(0, 0, null, null)$