algoritmi bidirezionali

Filippo Magi

February 18, 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** $f \neq 0 \wedge d_s(t) > \#V(G) \wedge d_t(s) > \#V(G)$ **do**
11: $(fso, fsi, startSource, startSink) \leftarrow$ Dfs $(G, startSource, startSink, fso, fsi, codaSource, codaSink)$
12: **if** $startSink = startSource \wedge startSink \neq null$ **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: **while** $\neg codaSource.isEmpty$ **do**
18: $codaSource$.dequeue().Reset()
19: **end while**
20: **while** $\neg codaSink.isEmpty$ **do**
21: $codaSink$.dequeue().Reset()
22: **end while**
23: **else if** $startSink = s$ **then**
24: $f \leftarrow fsi$
25: sendFlow($startSink, f$)
26: $fsi \leftarrow +\infty$
27: $startSink \leftarrow t$
28: **while** $\neg codaSink.isEmpty$ **do**
29: $codaSink$.dequeue().Reset()
30: **end while**
31: **else if** $startSource = t$ **then**
32: $f \leftarrow fso$
33: sendFlow($startSource, f$)
34: $fso \leftarrow +\infty$
35: $startsource \leftarrow s$
36: **while** $\neg codaSource.isEmpty$ **do**
37: $codaSource$.dequeue().Reset()
38: **end while**
39: **else**
40: **break**
41: **end if**
42: $fMax \leftarrow f + fMax$
43: **end while**
44: **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,$codaSource$ e $codaSink$, che mi salvano i nodi esplorati a partire da s e da t, per poi cancellare tutti i dati al loro interno

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** $d_s(startSink) < \#V(G) \wedge d_t(startSource) < \#V(G)$ **then**
5: **for all** $edge \in startSource.Edges$ **do**
6: $n \leftarrow edge.NextNode$
7: $p \leftarrow edge.PreviousNode$
8: **if** $startSource = p \wedge d_t(n) = d_t(p) - 1 \wedge u_f(edge) > 0$ **then**
9: $sourceFlow \leftarrow \min(sourceFlow, u_f(edge))$
10: $n.previousEdge \leftarrow edge$ {salvo anche il nodo precedente}
11: $codaSource$.enqueue(n)
12: **if** $n = t$ **then**
13: **return** $(sourceFlow, sinkFlow, n, startSink)$
14: **end if**
15: **if** $n.nextEdge \neq null$ **then** {n è già stato precedentemente esplorato dalla parte di Sink}
16: **return** $(sourceFlow, sinkFlow, n, n)$
17: **end if**
18: **return** SinkDfs$(G, n, startSink, sourceFlow, sinkFlow, codaSource, codaSink)$
19: **end if**
20: **end for**
21: $minDistance \leftarrow +\infty$
22: **for all** $edge \in startSource.Edges$ **do**
23: **if** $edge.PreviousNode = startSource \wedge u_f(edge) > 0$ **then**
24: $minDistance \leftarrow \min(minDistance, d_t(edge.nextNode))$
25: **end if**
26: **end for**
27: $d_t(startSource) \leftarrow minDistance + 1$
28: **if** $startSource = s$ **then**
29: $mom \leftarrow startsource$
30: **else**
31: $mom \leftarrow startSource.previousNode$
32: **end if**
33: $startSource.Reset()$
34: **return** Dfs$(G, mom, startSink, sourceFlow, sinkFlow, codaSource, codaSink)$
35: **end if**
36: **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,$codaSource$ e $codaSink$, che mi salvano i nodi esplorati a partire da s e da t, per poi cancellare tutti i dati al loro interno

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