algoritmi bidirezionali

Filippo Magi

January 17, 2022

1 Ottimizzazione sugli ultimi livelli

1.1 FlowFordFulkerson

1.2 DoBfs

Algorithm 1 Ricerca del flusso massimo

Require: grafo dei residui $\overleftrightarrow{G} = \{V(G), E(G) \cup \{\overleftarrow{e} : e \in E(G)\}$.

Ensure: valore del flusso massimo di $\overleftrightarrow{G}, E(G)$ aggiornato.

vuotoSouce ← nodo sorgente s di $\overleftrightarrow{G}$
vuotoSink ← nodo destinazione t di $\overleftrightarrow{G}$
fMax ← 0
loop
  (f,n) ← DoBfs($\overleftrightarrow{G}$, vuotoSource,vuotoSink)
  if f = 0 **then**
    break
  end iffMax ← fMax + f
  vuotoSouce ← null
  vuotoSink ← null
  aggiorno n per far sì che il doppio aggiornamento alla fine come se ce ne fosse stato solo uno
  momSource ← n
  momSink ← n
  while momSource non è il sorgente di $\overleftrightarrow{G}$ **do**
    aggiungo(o rimuovo) all'arco predecessore di momSource una quantità di flusso pari a f
    if la capacità (o il flusso) è negativo **then**
      vuotoSource ← momsource
      faccio tornare i dati a come erano prima dell'aggiornamento (fMax, valori di flusso passante e archi modificati)
      break
    else
      if capacità dell'arco indicato = 0 **then**
        momSource diventa non valido
        vuotoSource ← momSource
      end if
      aggiorno flusso passante per momSource
      momSource ← nodo predecessore di momSource
    end if
  end while
  while momSink non è il nodo destinazione t di $\overleftrightarrow{G}$ **do**
    aggiungo o rimuovo all'arco successore di momSink una quantità di flusso pari a f
    if capacità o flusso risultante negativa **then**
      vuotoSink ← momSink
      faccio tornare i dati a come erano prima dell'aggiornamento (fMax, valori di flusso passante e archi modificati)
      break
    else
      if capacità dell'arco indicato = 0 **then**
        momSink diventa non valido
        vuotoSink ← momSink
      end if
      aggiorno flusso passante per momSink
      momSink ← nodo successivo di momSink
    end if
  end while
end loop
return fMax

Algorithm 2 DoBfs con ottimizzaione sugli ultimi livelli

Require: grafo dei residui $\overleftrightarrow{G}$, noCapSource, noCapSink, cioè nodi,rispettivamente della parte sorgente e della parte destinazione, che non sono più raggiungili dal percorso deciso precedentemente

Ensure: valore del flusso inviabile, nodo appartenente LastSinkNodes, cioè tutti i nodi che sono intermedi che fanno da ponte tra le due ricerche.

codaSource ← coda di nodi vuota
codaSink ← coda di nodi vuota
if noCapSource ≠ null **then**
  provo a riparare noCapSource, cioè esplorando gli archi a lui connessi, cerco un nodo con tale che noCapSource.Label = (nodoTrovato.label + 1)
  if riesco a riparare noCapSource **then**
    if noCapSink = null **then**
      for all nodo n tale che è valido (cioè che ha l'arco precedente e successivo con capacità positiva) ed è appartenente a LastSinkNodes **do**
        da n cerco di retrocedere verso noCapSource, aggiornando ricorsivamente le informazioni dei nodi in modo opportuno (sopratutto per quanto riguarda n)
        if ho trovato il percorso tra n e noCapSource AND il possibile flusso inviabile è > 0 **then**
          return (flusso passante per n,n)
        end if
      end for
    else
      sourceRepaired ← *true*
    end if
  end if
  if noCapSource è il nodo sorgente di $\overleftrightarrow{G}$ **then**
    coda.enqueue(noCapSource)
  else if noCapSource non è stato esplorato dalla parte di source(è un nodo "di confine") **then**
    codaSource ← coda dei ultimi nodi esplorati dalla parte di source, esclusi quelli "di confine" (LastNodesSourceSide)
  else
    codaSource ← nodi esplorati da source con label = noCapSource.label-1
    cancello le informazioni da tutti i nodi esplorati da Source con label ≥ noCapSource.label
  end if
end if

26: **if** noCapSink $\neq$ null **then**
27: cerco di riparare noCapSink
28: **if** riesco a riparare noCapSink **then**
29: sinkRepaired $\leftarrow$ true
30: **for all** nodo n tale che è valido e appartenenti a LastSinkNodes **do**
31: **if** noCapSource è stato riparato AND da n posso ricorsivamente retrocedere verso noCapSource(aggiornando i dati) **then**
32: sourceFlow $\leftarrow$ flusso passante per n
33: **else**
34: sourceFlow $\leftarrow$ min(flusso passabile attraverso il precessore di n, capacità/flusso inviabile tramite l'arco che collega quel nodo a n)
35: **end if**
36: **if** sourceFlow > 0 AND n può retrocedere ricorsivamente verso noCapSink(aggiornando i dati) AND il flusso passabile per $n > 0$ **then**
37: **retrun** (min(flusso passabile per n, sourceFlow),n)
38: **end if**
39: **end for**
40: **end if**
41: **if** noCapSink è il nodo destinazione t **then**
42: codaSink.enqueue(noCapSink)
43: **else**
44: codaSink $\leftarrow$ coda dei nodi $\in \overleftrightarrow{G}$ esplorati da sink con label = (noCapSink.label-1)
45: cancello le informazioni da tutti i nodi esplorati da Sink con label $\geq$ noCapSink.label
46: **end if**
47: **end if**

```
48: while codaSink o codaSource non sono vuote do
49:   if codaSource non è vuoto AND (noCapSource ≠ null OR non sono riuscito a riparare noCapSource then
50:     element ← codaSource.dequeue()
51:     if element non è della parte di source OR element non è valido then
52:       continue
53:     end if
54:     for all arco edge in archi che entrano ed escono da element do
55:       p ← nodo precedente di edge
56:       n ← nodo successivo di edge
57:       if element = p AND capacità di edge > 0 then
58:         if n è già stato esplorato then
59:           if n è parte di sourceside (esplorato da source) then
60:             continue
61:           else{in questo caso ho le due parti che si incontrano}
62:             f ← flusso invabile date le informazioni di p,n,edge
63:             if f = 0 then
64:               continue
65:             end if
66:             aggiorno i dati di n e di edge
67:             aggiungo n a LastNodesSinkSide, inserisco tutti i nodi collegati direttamente a n che fanno parte di SourceSide in LastNodesSourceSide
68:             return (f,n)
69:           end if
70:         end if
71:         if n è sinkSide, ma non è il nodo destinazione t then
72:           sinkRepaired ← false
73:           for all nodo node che di sinkSide con label = (n.label - 1) do
74:             codaSink.enqueue(n)
75:           end for
76:           cancello di dati su sinkside per ogni nodo con label ≥ n.label
77:           continue
78:         end if
79:         aggiorno di dati di n e di edge
80:         codaSource.enqueue(n)
```

```
81:         else if element = n AND flusso passante per e > 0 then
82:           if p è stato già esplorato then
83:             if p è stato eslorato da source then
84:               continue
85:             else
86:               f ← flusso inviabile date le informazioni di n,p,edge
87:               if f = 0 then
88:                 continue
89:               end if
90:               aggiorno le informazioni di p ed edge
91:               return (f,p)
92:             end if
93:           end if
94:           if p è stato esplorato da Sink AND p non è il nodo destinatario t
  then
95:             sinkRepaired ← false
96:             for all nodo node epsplorto da sinkNode con label = (p.label-1)
  do
97:               codaSink.enqueue(node)
98:             end for
99:             cancello le informazioni di tutti i nodi esplorati da Sink con Label
  ≥ n.label
100:              continue
101:          end if
102:          aggiorno le informazioni di p ed di edge
103:          codaSource.enqueue(p)
104:        end if
105:      end for
106:    end if
```

```
107:  if codaSink non è vuota AND (noCapSink ≠ null OR not sinkRepaired) then
108:      element ← codaSink.dequeue()
109:      if element è sourceSide OR element non è valido then
110:        continue
111:      end if
112:      for all arco edge in element.Edges do
113:        p ← nodo precedente di edge
114:        n ← nodo successore di edge
115:        if element = n AND capacità di edge > 0 then
116:          if p è stato esplorato then
117:            if p è stato esplorato da source then
118:              continue
119:            else
120:              if source è stato riparato AND n può retrocedere ricorsivamente verso noCapSink(aggiornando i dati) AND il flusso passabile per n > 0 then
121:                f ← valore di flusso inviabile date le ottenute
122:                if f > 0 then
123:                  return f
124:                end if
125:              end if
126:              f ← flusso inviabile dato p,n ed edge
127:              if f = 0 then
128:                continue
129:              end if
130:              aggiorno di dati di n ed edge
131:              return (f,n)
132:            end if
133:          end if
134:          if p è stato esplorato da source, se non per L'Inizializzazione (cioè noCapSink è il nodo destinazione t) then
135:            continue
136:          end if
137:          aggiorno informazioni di p ed edge
138:          codaSink.enqueue(p)
```

```
139:          else if element = n AND flusso passante per edge > 0 then
140:            if n è stato esplorato then
141:              if n è stato esplorato da source (sourceSide) then
142:                continue
143:              else
144:                if sourceRepaired AND da p riesco a raggiungere noCap-
    Source(aggiornando ricorsivamente i dati) then
145:                  return (flusso passabile attraverso p,p)
146:                end if
147:                f ← flusso passante dati i dati di p,n ed edge
148:                if f = 0 then
149:                  continue
150:                end if
151:                aggiorno i dati di p ed edge
152:                return  (f,p)
153:              end if
154:            end if
155:            if n è sourceSide ed non è il nodo sorgente s then
156:              continue
157:            end if
158:            aggiorno di dati di n
159:            codaSink.enqueue(n)
160:          end if
161:        end for
162:      end if
163: end while
164: return (0,null)
```