algoritmi bidirezionali

Filippo Magi

January 15, 2022

1 Senza alcuna ottimizzazione

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

```
1: fMax ← 0
2: vuotoSource ← true
3: vuotoSource ← true
4: loop
5:    (f, nodo) ← DoBfs(G↔,vuotoSource,vuotoSink)
6:    if f = 0 then
7:       break
8:    end if
9:    vuotoSource ← false
10:   vuotoSource ← false
11:   fMax ← fMax + f
12:   mom ← n
13:   while n non è il sourceNode di G↔ do
14:      invio(o richiedo) su n.PreviousEdge (arco usato per arrivare a n) una
      quantità di flusso pari a f
15:      if capacità residua = 0 then
16:         vuotoSource ← true
17:      end if
18:      aggiorno flusso passante per n {n.InFlow -=f}
19:      n ← nodo antecedente di n
20:   end while
21:   while mom non è il sinkNode di G↔ do
22:      invio(o richiedo) su n.NextEdge (arco usato per arrivare a n) una quan-
      tità di flusso pari a f
23:      if capacità residua = 0 then
24:         vuotoSink ← true
25:      end if
26:      aggiorno flusso passante per n {n.InFlow -=f}
27:      n ← nodo successore di n
28:   end while
29: end loop
30: return fMax
```

Algorithm 2 DoBfs : Ricerca un path tra s e x[], e da x[] a t, dove t[] sono i nodi intermedi dove si incontrano i due path

Require: Grafo dei residui $\overleftrightarrow{G}$, sourceSide e sinkSide, che sono dei booleani che chiariscono in quale parte si dovrà operare

Ensure: valore del flusso inviabile, nodo appartenente x[], che tiene in memoria sia il nodo successivo, sia il nodo precedente

```
1: codaSource ← coda di nodi
2: codaSink ← coda di nodi
3: if sourceSide then
4:    elimino informazioni dai nodi tra s e x[]
5:    codaSource.enqueue(nodo source di G↔)
6: end if
7: if sinkSide then
8:    elimino informazioni dai nodi tra x[] e t
9:    codaSource.enqueue(nodo sink di G↔)
10: end if
11: {per motivi prestazioni si controlla nel codice se si hanno sia sinkSide sia
    sourceSide positivi per non analizzare tutti i nodi 2 volte}
12: while né codaSource né codaSink sono vuote do
13:    if codaSource non è vuota then
14:       element ← codaSource.dequeue()
15:       for all arco edge in n.Edges do
16:          p ← nodo precedente di edge
17:          n ← nodo successore di edge
18:          if element = p AND capacità di edge> 0 then
19:             if n è già stato esplorato then
20:                if n è stato esplorato dalla parte di Source then
21:                   continue
22:                else
23:                   f ← flusso massimo inviabile dato le informazioni di n,p,edge
24:                   if  f = 0 then
25:                      continue
26:                   end if
27:                   il percorso è stato trovato, aggiorno i dati di n (e il fatto che
    edge deve diminuire la capacità)
28:                   return (f,n)
29:                end if
30:             end if
31:             aggiorno di dati di n
32:             codaSource.enqueue(n)
33:          end if
```

```
34:         if element = n AND flusso di edge> 0 then
35:           if p è già stato esplorato then
36:             if p è stato già esplorato dalla parte di Source then
37:               continue
38:             else
39:               f ← flusso massimo inviabile dato le informazioni di n,p,edge
40:               if f = 0 then
41:                 continue
42:               end if
43:               percorso trovato, aggiorno i dati di p (e il fatto che edge deve
    diminuire il flusso) (f,p)
44:             end if
45:           end if
46:           aggiorno dati di p
47:           codaSource.enqueue(p)
48:         end if
49:       end for
50:     end if
51:     if codaSink non è vuota then
52:       element ← codaSink.dequeue()
53:       for all arco edge degli archi di element (element.Edges) do
54:         p ← nodo precedente di edge
55:         n ← nodo successivo di edge
56:         if element = n AND capacità di edge > 0 then
57:           if p è stato esplorato then
58:             if p è stato esplorato dalla parte di Sink then
59:               continue
60:             else
61:               f ← flusso massimo inviabile dato le informazioni di n,p,edge
62:               if f = 0 then
63:                 continue
64:               end if
65:               percorso trovato, aggiorno i dati di n (e il fatto che edge deve
    diminuire la capacità)
66:               return (f,n)
67:             end if
68:           end if
69:           aggiorno di dati di p
70:           codaSink.enqueue(p)
71:         end if
```

```
72:         if element = p AND flusso di edge > 0 then
73:           if n è già stato esplorato then
74:             if n è stato esplorato da Sink then
75:               continue
76:             else
77:               f ← flusso massimo inviabile dato le informazioni di n,p,edge
78:               if f = 0 then
79:                 continue
80:               end if
81:               percorso trovato, aggiorno i dati di p (e il fatto che edge deve
    diminuire il flusso)
82:               return (f,p)
83:             end if
84:           end if
85:           aggiorno dati di n (ed edge)
86:           codaSink.enqueue(n)
87:         end if
88:       end for
89:     end if
90: end while
```