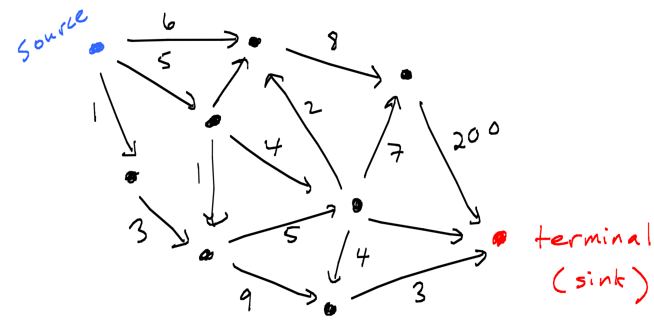
lecture 10

Network Flows 1:

- definitions
- residual graphs & augmenting paths
- Ford - Fulkerson algorithm

"Flow Network" - Example

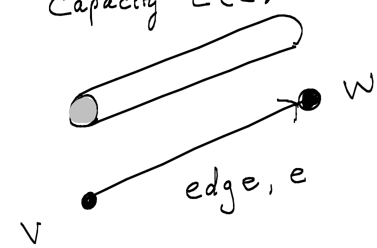


The edge weights are capacities for transporting stuff from source to sink via a steady state flow.

Flow Network definition

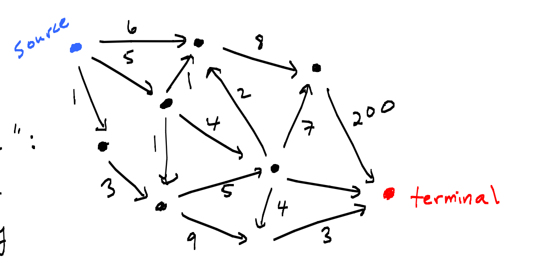
- $G = (V, E)$ is a directed weighted graph.
- Source vertex s in V has no incoming edges.
- Terminal vertex t in V has no outgoing edges.
- For each edge e in E, the edge weight is an integer valued capacity $c(e) \geq 0$.

Think of the edges as pipes: empty cylinders that could allow a steady flow through, up to $\underbrace{c(e)}_{\text{units not specified}}$:



"Flow network":

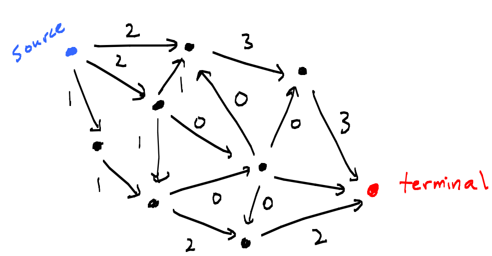
edge weight is a capacity $c(e)$



$\neq$

"Network flow":

edge weight is a flow $f(e)$



Network Flow definition

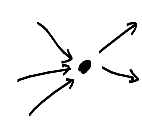
$f: E \rightarrow \{0, 1, 2, \dots\}$

This can be generalized to non-integers.

Capacity Constraint:

for any $e \in E$, $0 \leq f(e) \leq c(e)$

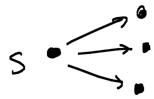
Conservation Constraint:



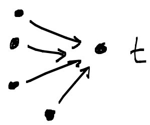
for any $u \in V \setminus \{s, t\}$,

$$\sum_{v \in V} f(u,v) = \sum_{v \in V} f(v,u)$$

$$f^{out}(s) \equiv \sum_{v \in V} f(s,v)$$



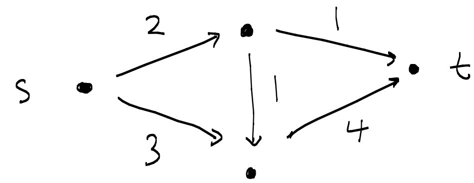
$$f^{in}(t) \equiv \sum_{v \in V} f(v,t)$$



Exercise:

Show $f^{out}(s) = f^{in}(t)$, which is called the value of the flow.

Example (flow)



$f^{out}(s) = 5$ $\qquad$ $f^{in}(t) = 5$

Note conservation conditions for $V \setminus \{s,t\}$

Given a flow network (G, s, t, c), there may be many "admissable" i.e. allowable network flows.

The maximum flow is the flow that has the largest value.

How can we find the maximum flow?

(Note there may be two flows have the same value. That's ok.)

Applications

The formulation of the maximum flow problem suggests obvious applications.

e.g. transportation networks.

But there are also many applications that have nothing to do with physical flows.

e.g. matching problems.

Kleinberg Tardos textbook has several sections on applications which are typically covered in COMP 360. e.g. http://cs.mcgill.ca/~hatami/comp360-2014/ spends first $3\frac{1}{2}$ weeks on network flows.

Algorithm 1: how to find maximum flow from s to t?

Initialize $f = 0$

while true {

if there is a path P from s to t, such that all edges on that path have a flow that is strictly less than the capacity

then increase the flow on that path by as much as possible

else break

}

Algorithm 1: how to find maximum flow from s to t?

Initialize $f = 0$

while true {

if there is a path P from s to t, such that, for all edges e in P, $f(e) < c(e)$ {

$\beta = \min \{ c(e) - f(e) : e \text{ in } P \}$

For all $e \in P$,

$f(e) = f(e) + \beta$

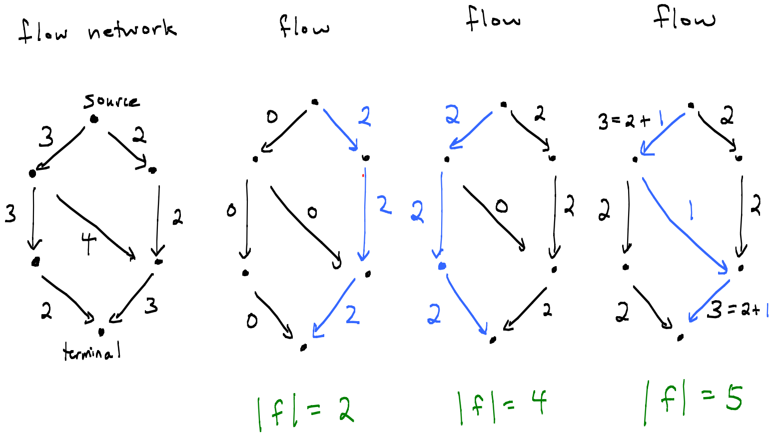
}

else break

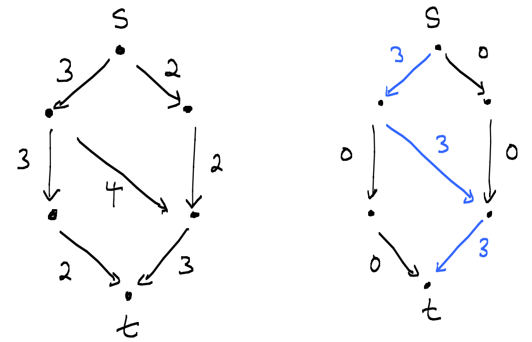
}

// β stands for "bottleneck"

Example where Algorithm 1 works :



Example where Algorithm 1 fails :

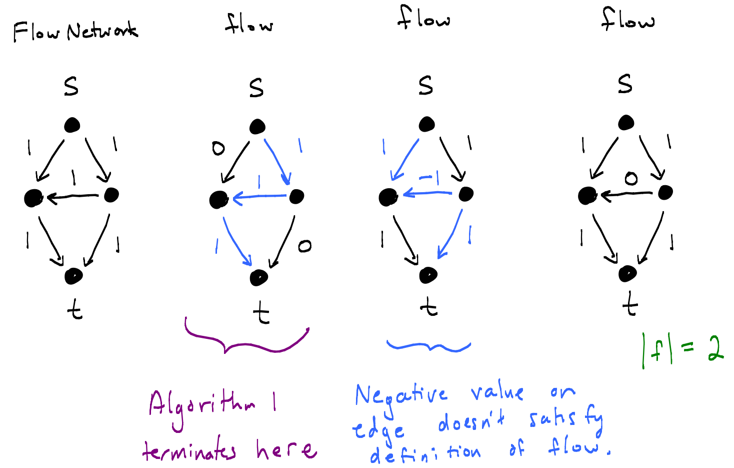


|f| = 3 and algorithm terminates

How to choose paths so that we

- don't get stuck
- are guarenteed to find the maximum flow
- are efficient (will be covered in COMP 360)

Algorithm 2 Motivation : if we could subtract flow, then we could redirect it.



Using negative numbers on directed edges is possible, but I will present an alternative representation which is based on a "residual graph", and use that in the second algorithm (later).

The residual graph is a weighted graph whose edge weights represent how we can change the flow f.

Residual Graph

Given a flow network $G = (V, E)$ with edge capacities c, and given a flow f, define the residual graph G_f :

- G_f has the same vertices as G
- The edges E_f have capacities c_f (called 'residual capacities') that allow us to change the flow f, either by:
  1) adding flow to an edge e in E
  2) subtracting flow from an edge e in E

For each edge $e = (u,v)$ in E

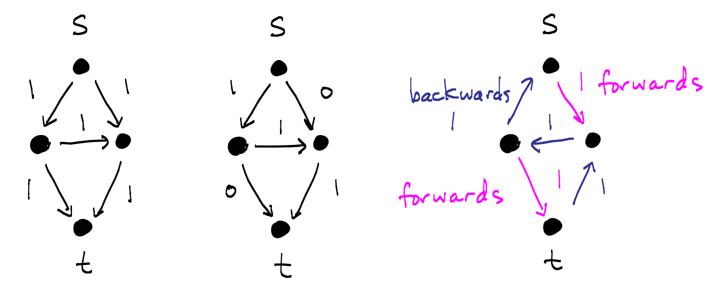
if $f(e) < c(e)$

then put a 'forward edge' (u,v) in E_f
with residual capacity
$$c_f(e) = c(e) - f(e)$$

if $f(e) > 0$
then put a 'backwards edge' (v,u) in E_f
with residual capacity
$$c_f(e) = f(e)$$

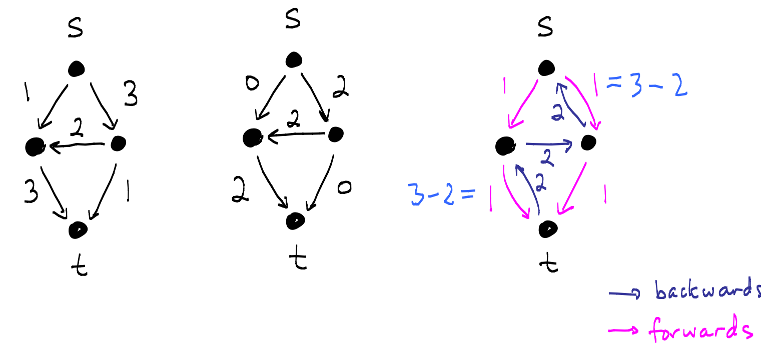
Example 1 (of 3)

G, c	f	G_f, c_f
flow network	flow	residual graph



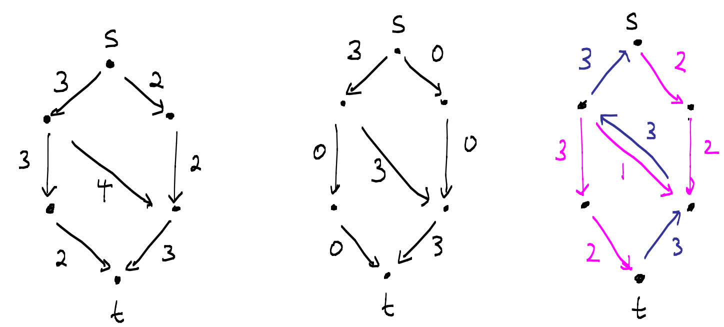
Example 2

G, c	f	G_f, c_f
flow network	flow	residual graph

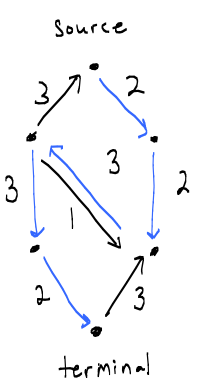


Example 3 (from earlier)

Flow Network	Flow	Residual Graph
G, c	f	G_f, c_f



G_f, c_f

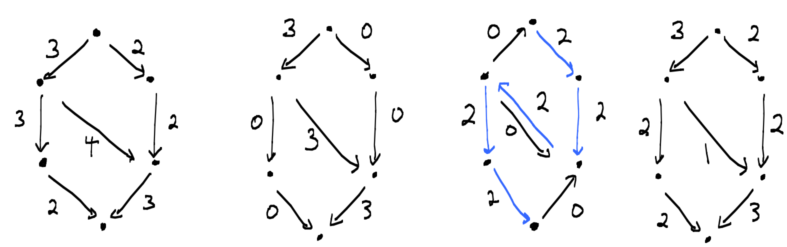


An augmenting path is an "s–t path" (a path from s to t) in the residual graph G_f that allows us to increase the flow.

Q: In the example here, by how much can we increase the flow by using this path?

A: 2

Flow Network G, c	Flow in G	Flow in G_f	$\Rightarrow$	Flow in G



$|f| = 3$
value of flow
Algorithm 1 got stuck here.

$\beta = 2$
bottleneck

$|f| = 5$
value of flow

How to increase flow f ?

- Compute the residual graph. G_f, c_f
- Find an s-t path P
- Augment the flow f along the path P
  - Let β be the bottleneck $\equiv$ the smallest residual capacity $c_f(e)$ of edges e on the path P.
  - Add β to the flow $f(e)$ on each edge e of the path P.

f. augment (P) {

  $\beta = \min \{ c_f(e) : e \in P \}$

  for each edge $e = (u,v) \in P$

    if e is forward edge

      // $f(e) < c(e)$

      $f(e) = f(e) + \beta$

    else // e is a backwards edge,

      // so $f(e) > 0$

      $f(e) = f(e) - \beta$

}

Algorithm 2 :
computing maximum flow
(Ford - Fulkerson 1954)

$f = 0$

$G_f = G$

while there is an s-t path P in G_f {

  f. augment (P)

  update G_f based on new f

}

Claim: The Ford-Fulkerson algorithm terminates.

Proof:

The capacities and flows are integers > 0.

The sum of capacities leaving s is finite.

Bottleneck values β are positive integers.

The flow increases by the bottleneck β in each pass through main loop.

$\Rightarrow$ Flow is an increasing sequence of integers, bounded above.

How long does F-F take?

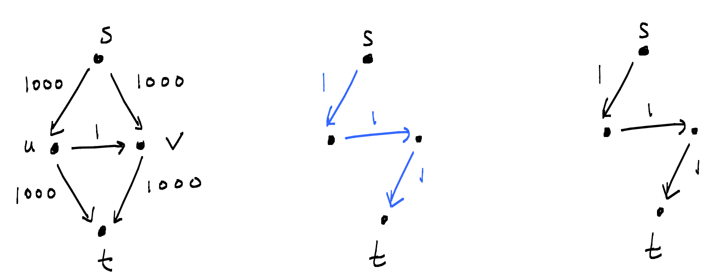
Let $C = \sum_{e \text{ outgoing from } s} c(e)$.

Finding a path from s to t in G_f takes $O(|E|)$ eg. DFS or BFS.

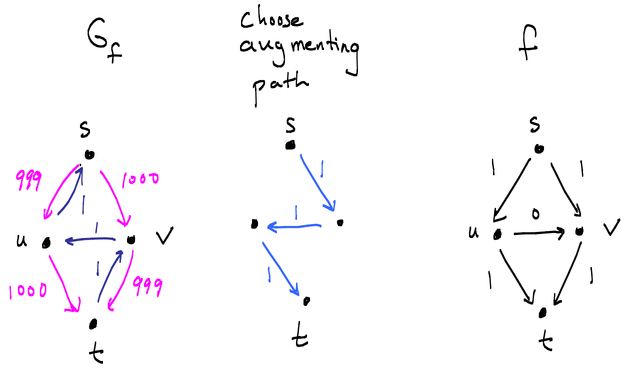
Since the flow increases by at least 1 in each pass, the algorithm is $O(C\,|E|)$

Worst case of F-F

$G_f = G$ find augmenting path f

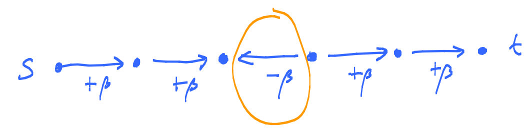


Worst case of F-F (continued)



Repeating this will take 1000 × 2 iterations to find a flow of 2000. A better choice would have found that flow in 2 iterations.

ASIDE: many implementations of F-F (such as Sedgewick's) don't construct a residual graph G_f. Instead, they find a "path" in the original graph G (where some edges may be "backward", and they subtract flow on these edges.)



Next lecture

network flows 2

- max flow = min cut

COMP 360

- efficient network flow (how to choose a good augmenting path)
- applications

Announcements

- midterm | grading - see updated solutions PDF
- A1 grading: late penalties and fairness
- A3 will be posted end of next week (or later) and due in early/mid March

(now is the time to catchup)