Efficient implementation of Dinic's algorithm for maximum flow

Goal

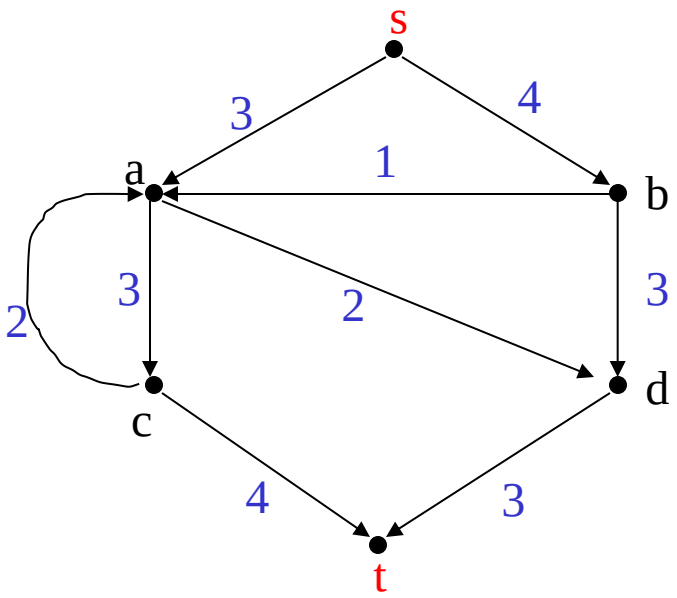
An algorithm that find maximum flow from s to t in a directed graph in $O(mn\log(n))$ time (and with additional effort in $O(mn\log(n^2/m))$

There are 2 algorithms that achieve that, which are quite different.

One of them you actually have already seen…

Definitions

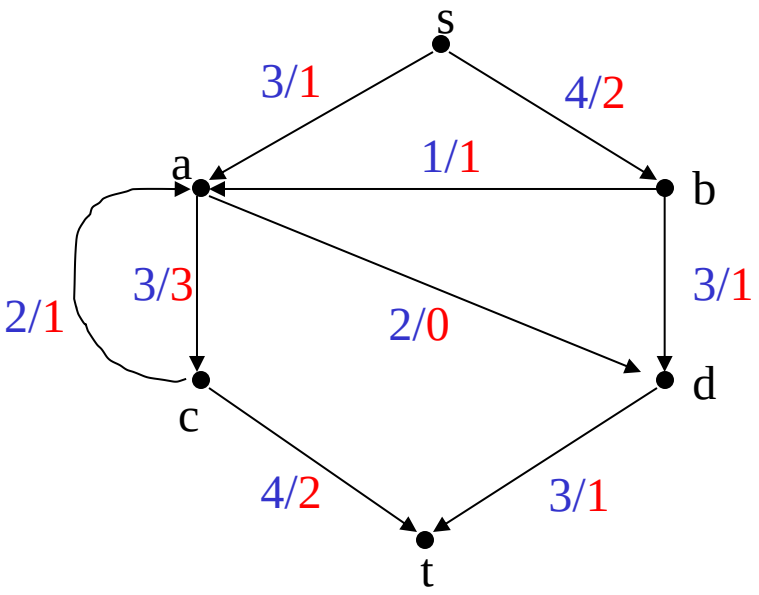
- G=(V,E) is a directed graph
- capacity u(v,w) for every arc (v,w)∈E
- Two distinguished vertices s and t



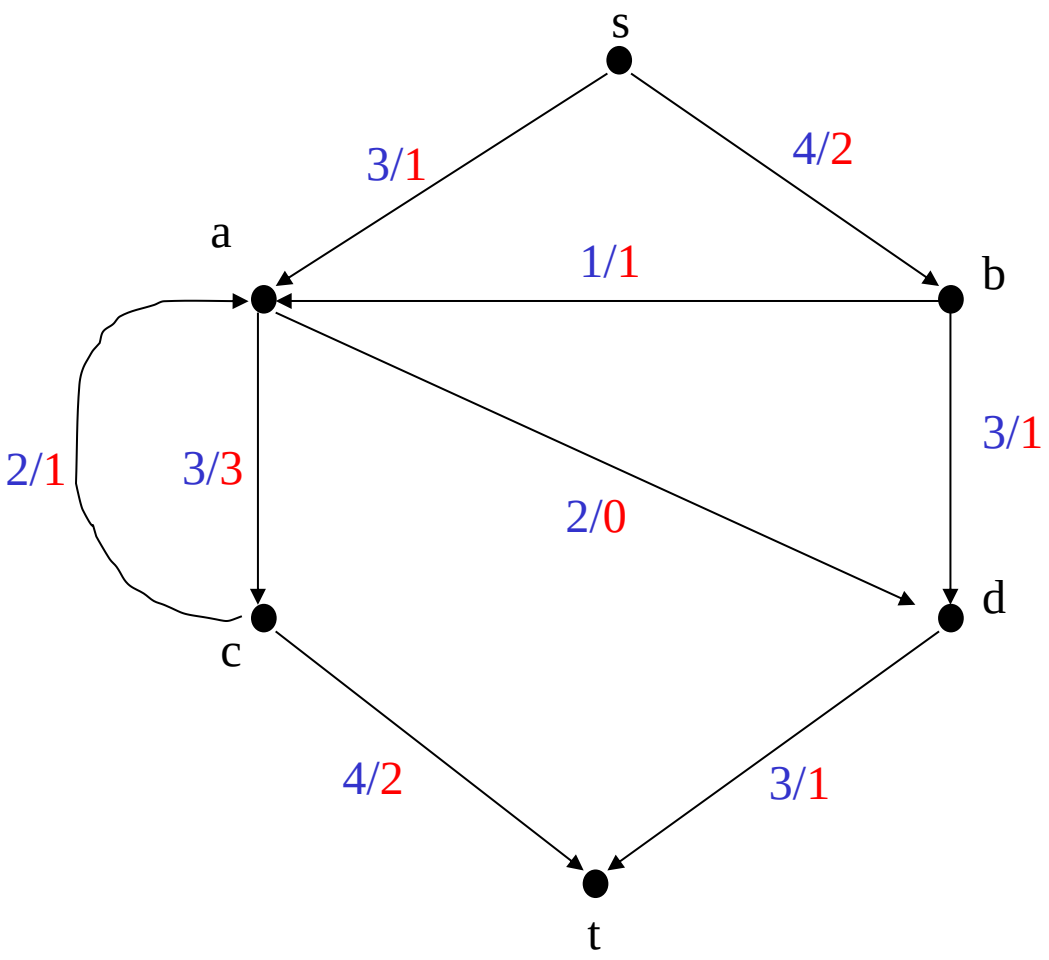
Flow

A function f on the arcs (E):

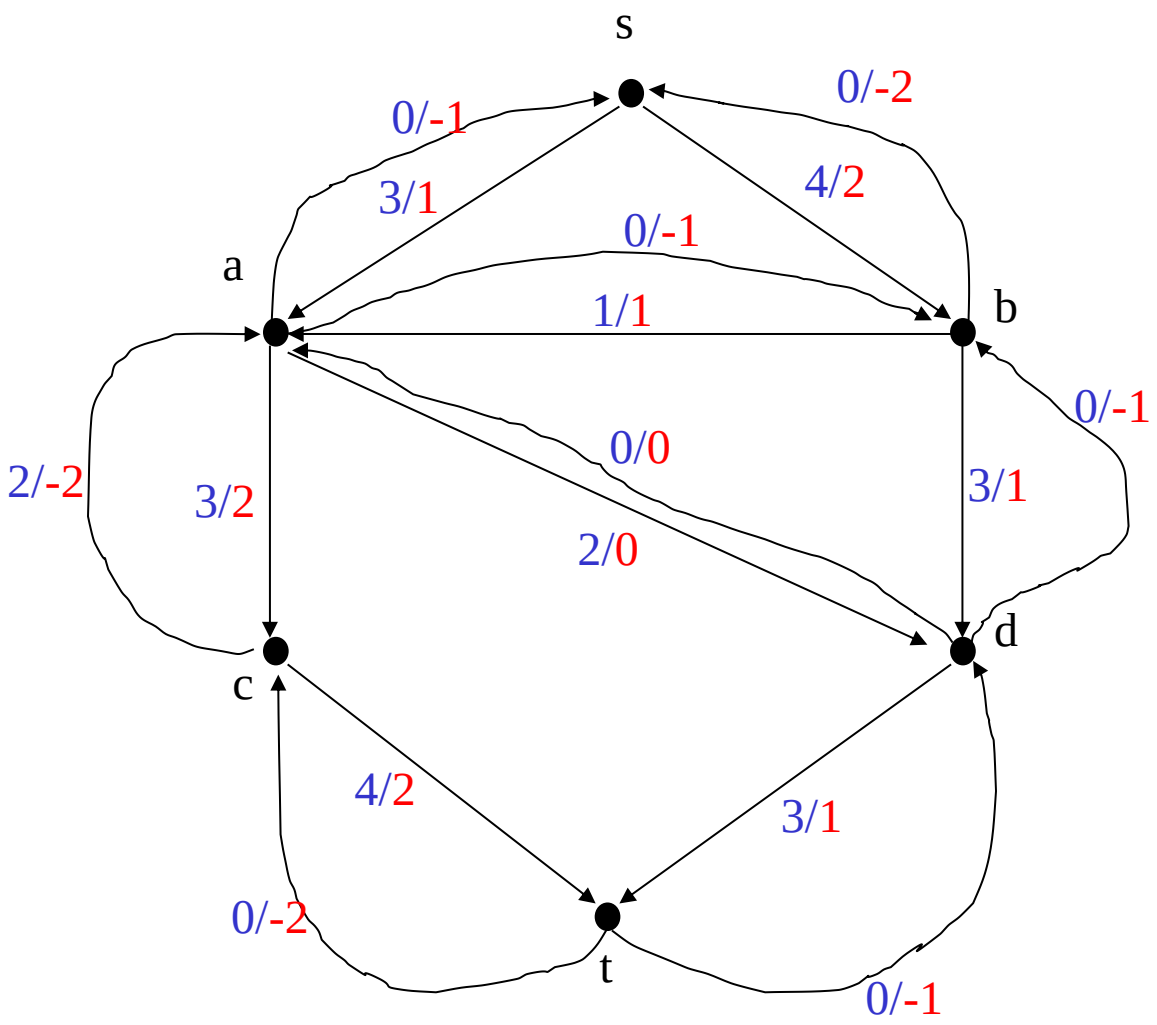
- $0 \leq f(v,w) \leq u(v,w) \quad \forall (v,w) \in E$
- $\sum_{(v,w)\in E} f(v,w) - \sum_{(w,v)\in E} f(w,v) = 0 \quad \forall v \in V \backslash \{s,t\}$



Flow (equivalent def)



Flow (equivalent def)



Flow (equivalent def)

For each arc $(v,w) \in E$ if $(w,v) \notin E$ add (w,v) with $u(w,v) = 0$

A flow is a function on the arcs which satisfying

- $f(v,w) = -f(w,v)$ skew symmetry
- $f(v,w) \leq u(v,w)$
- For every v except s and t $\Sigma_w f(v,w) = 0$

Flow (equivalent def)

From f satisfying the first definition we get f’ satisfying the second by $f'(v,w) = f(v,w) - f(w,v)$

From f satisfying the second definition we get f’ satisfying the first by $f'(v,w) = \max\{0, f(v,w)\}$

The value of f

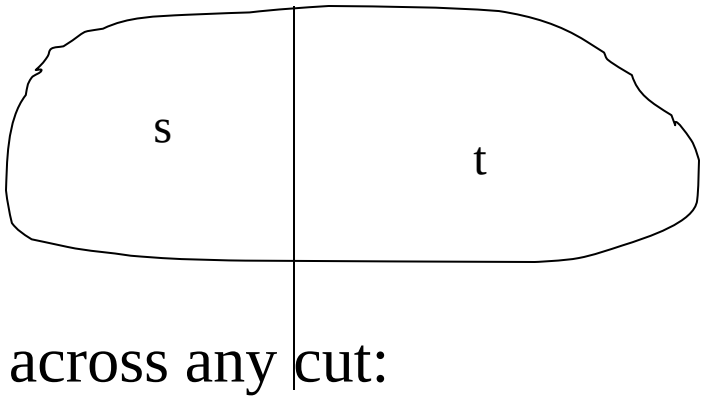
The value of the flow $|f| = \Sigma_w f(s,w)$

It is not hard to show that $|f| = \Sigma_w f(w,t)$

The maxflow problem is to find f with maximum value

Flows and s-t cuts

Let (X,X') be a cut such that s $\in$X, t $\in$X'.



Flow is the same across any cut:

$$f(X,X') = \sum_{\substack{v \in X,\\ w \in X'}} f(v,w) = \sum_{\substack{v \in X,\\ w \in V}} f(v,w) - \sum_{\substack{v \in X,\\ w \in X}} f(v,w) = |f| - 0 = |f|$$

so

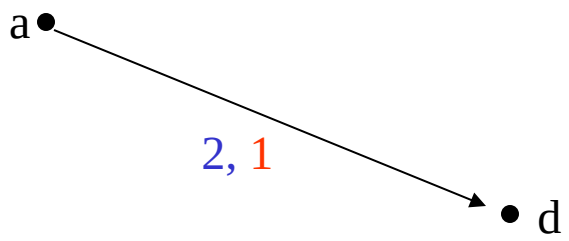
$$|f| \leq cap(X,X') = \Sigma u(v,w)$$

The value of the maximum flow is smaller than the minimum capacity of a cut.

More definitions

The residual capacity of a flow is a function r on the edges such that

$r(v,w) = u(v,w) - f(v,w)$



Interpretation: We can push r(v,w) more flow from v to w by increasing f(v,w) and decreasing f(w,v)

More definitions (cont)

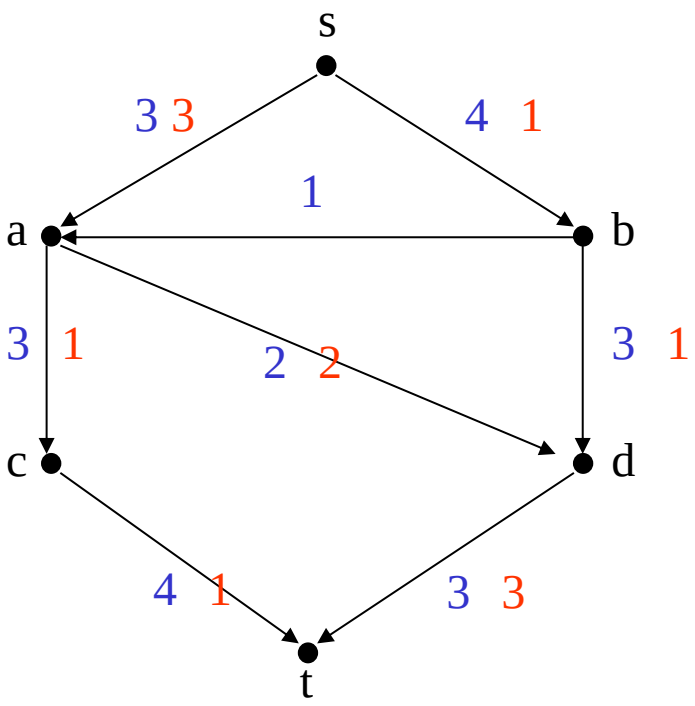
We define the residual graph R on V such that there is an arc from v to w with capacity r(v,w) for every v and w such that r(v,w) > 0

An augmenting path $p \in R$ is a path from s to t in R

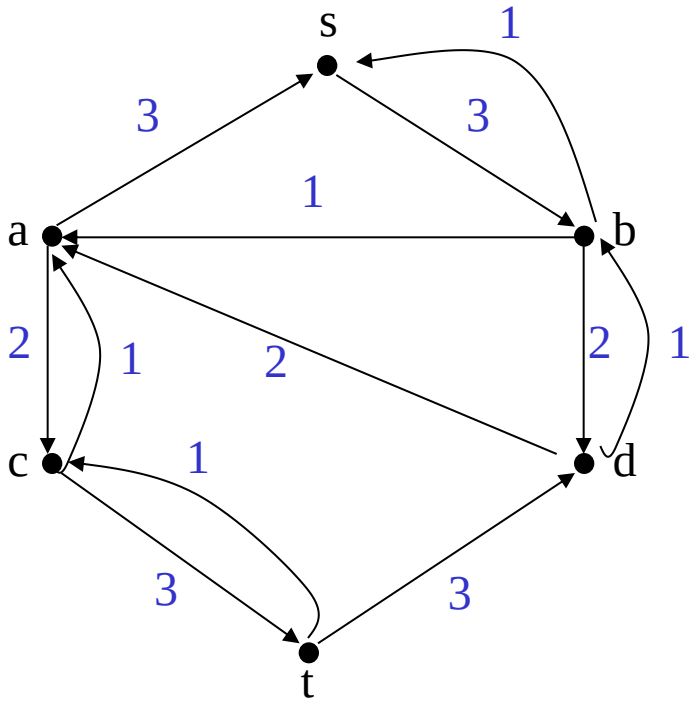
$$r(p) = \min_{(v,w) \in p} r(v,w)$$

We can increase the flow by r(p)

Example



A flow



The residual network

Basic theorem

(1) f is max flow <==>

(2) There is no augmenting path in R <==>

(3) |f| = cap(X,X') for some X

Proof. (3) ==> (1), (1) ==> (2) obvious

To prove (2) ==>(3) let X be all vertices reachable from s in R. By assumption t $\notin$ X. So (X,X') is an s-t cut. Since there is no edge from X to X' in R

|f| = f(X,X') = Σf(v,w) = Σu(v,w) = cap(X,X')

□

Basic Scheme

Repeat the following step:

Find an augmenting path in R, increase the flow, update R

Stop when s and t are disconnected in R.

Need to be careful about how you choose those augmenting paths !

Edmonds and Karp

Choose a shortest augmenting path

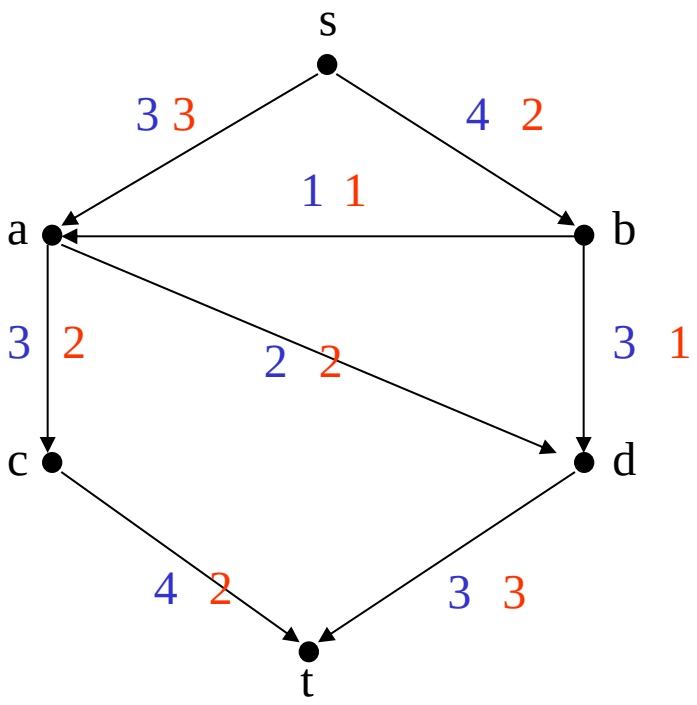
==> $O(m^2n)$ time algorithm

Dinic

Def: A flow f is blocking if every path from s to t contains a saturated edge.

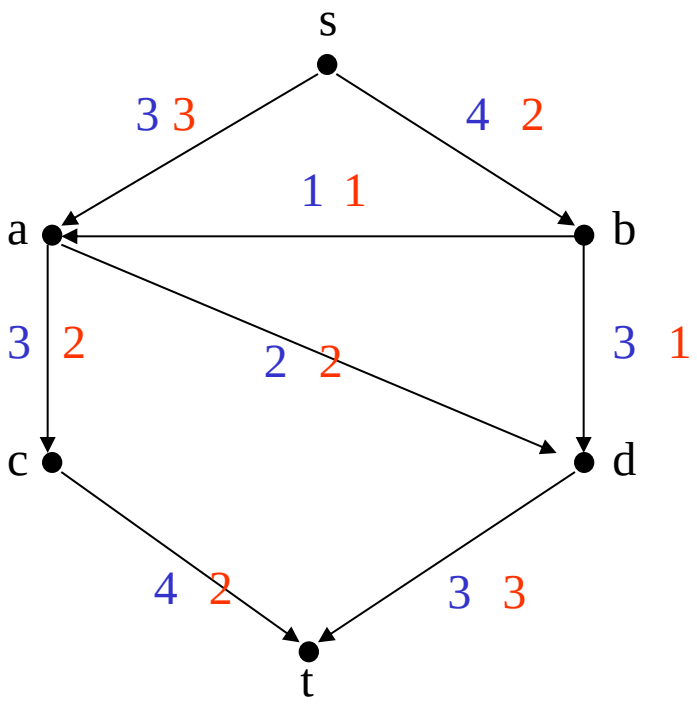
Note that a blocking flow need not be a maximum flow!

Example



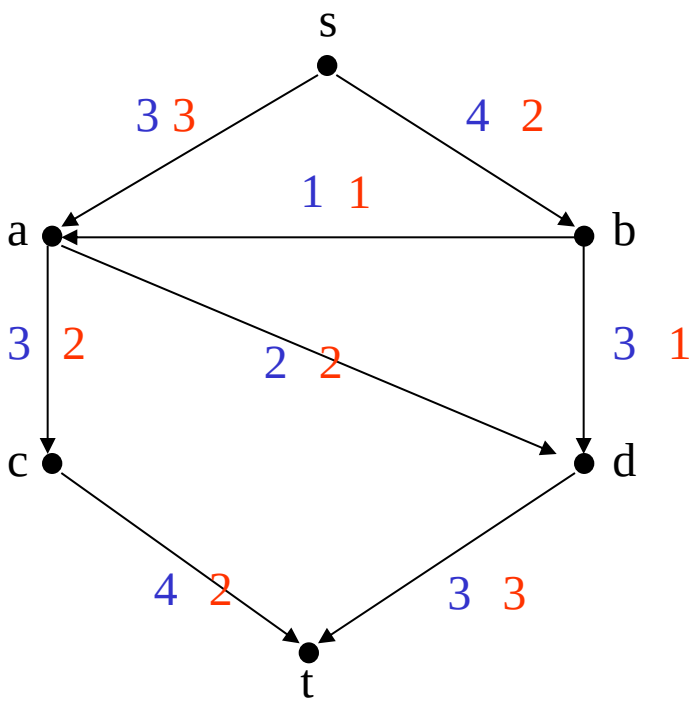
Is this a blocking flow ?

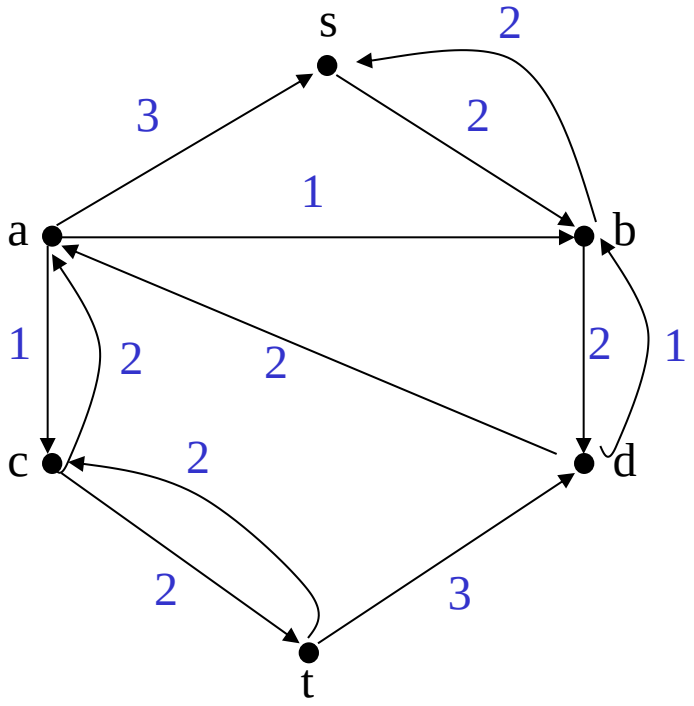
Example



Is this a maximum flow ?

Example (cont)





Dinic's algorithm

Let level(v) be the length of the shortest path from s to v in R.

Let L be the subgraph of R containing only edges (v,w) such that level(w) = level(v) + 1

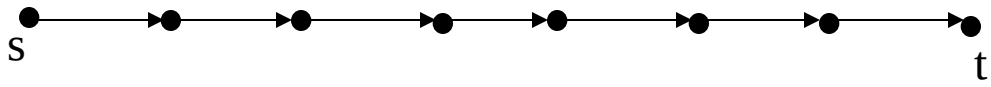
Dinic: find a blocking flow f' in L let f := f + f' and repeat.

Dinic's algorithm (analysis)

Theorem: f is a maximum flow after at most n-1 blocking flow computations.

Proof. Each edge in R' is either an edge in R or the reverse of an edge in R.

Look at a shortest path from s to t in R'



The level in R increases by at most one at each step but cannot increase by exactly one at every step

□

Dinic's algorithm (analysis)

How fast can we find a blocking flow ?

Using DFS in a straightforward manner would give us O(nm).

==> $O(n^2m)$ for the whole max flow algorithm.

One can do it using dynamic trees in O(mlog(n))

When all capacities are 1

How fast can we find a blocking flow ?

When you augment along a path all edges are saturated.

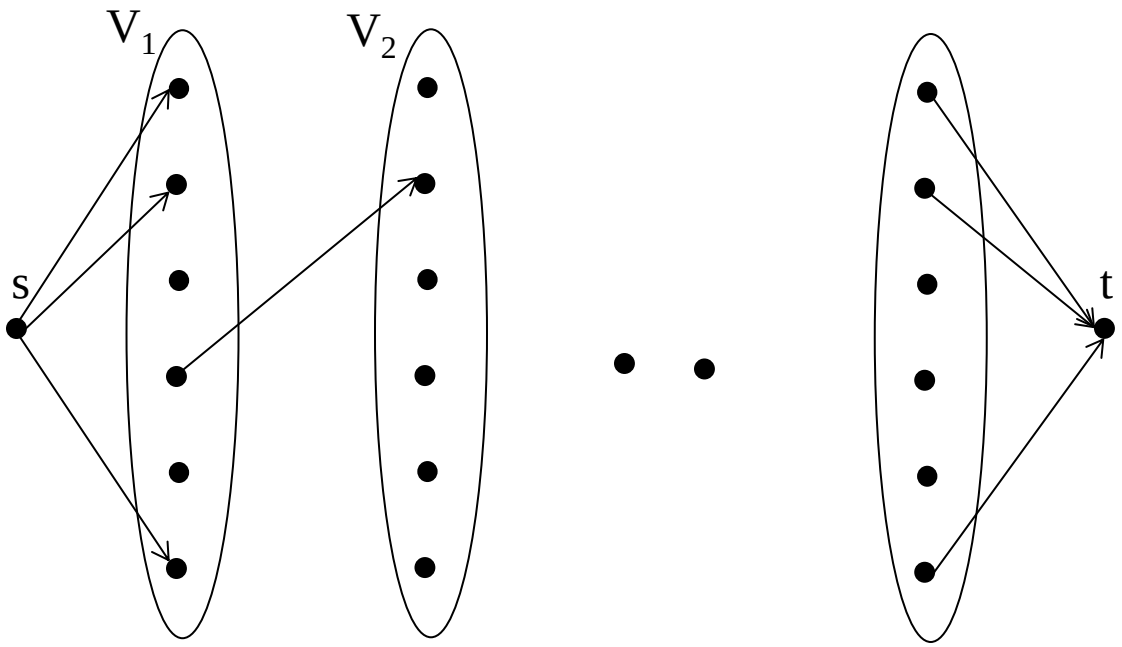
➔ We find a blocking flow in O(m) time

How many phases do we have ?

When all capacities are 1

After √m blocking flows there are √m blocking flows left.

Consider the layered network L at that point:

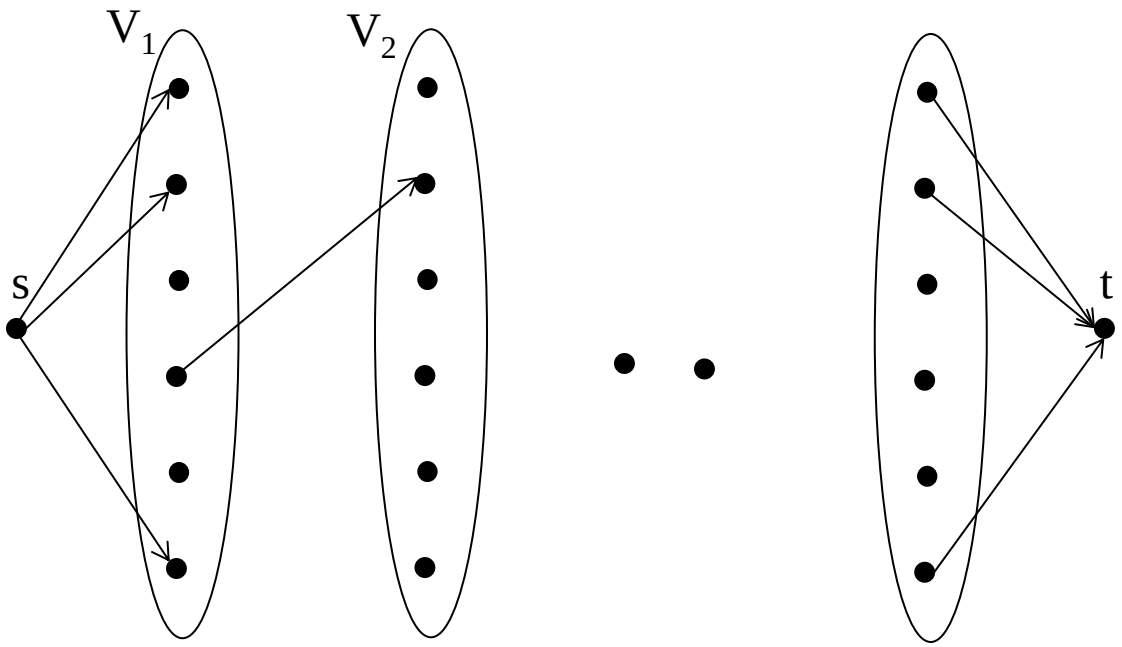


When all capacities are 1

The distance from s to t $\geq \sqrt{m}$

➔ $\exists i$ s.t. the number of arcs from V_i to V_{i+1} is $\leq \sqrt{m}$

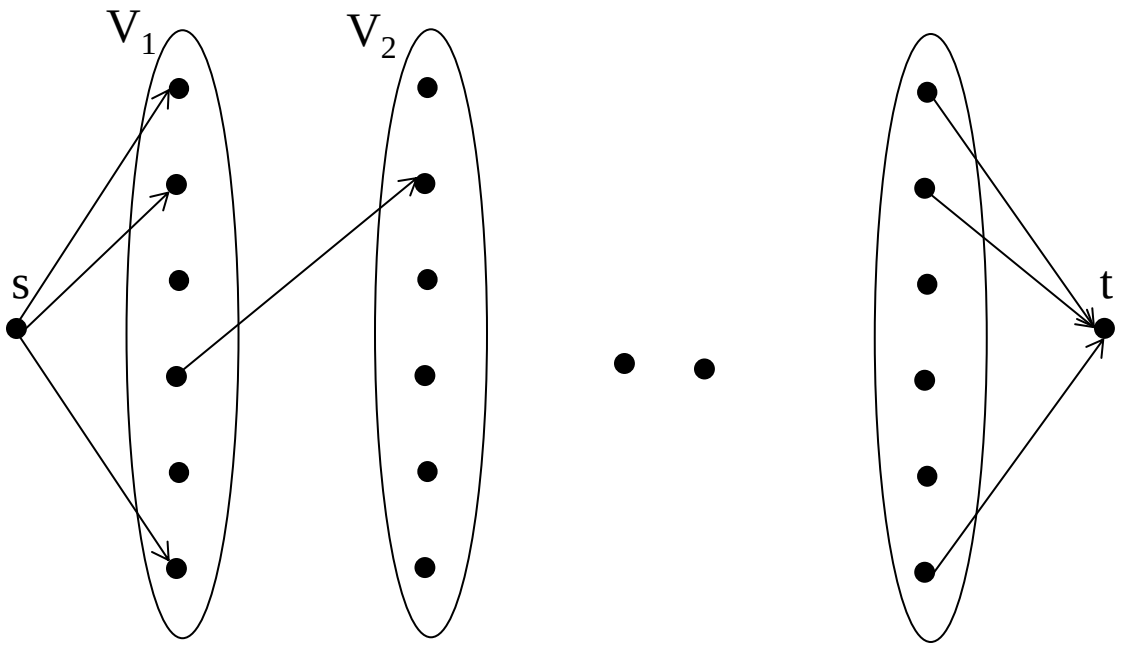
➔ The number of phases left $\leq \sqrt{m}$



When all capacities are 1 (2^{nd} bound)

After $2n^{2/3}$ blocking flows there are $n^{2/3}$ blocking flows left.

Consider the layered network L at that point:

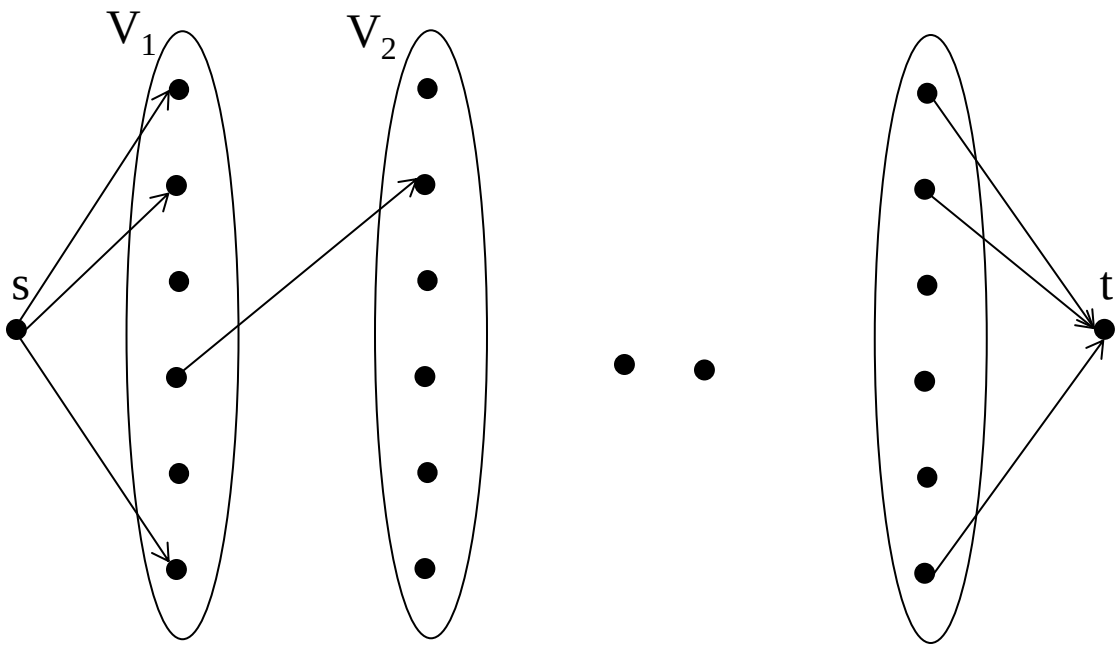


When all capacities are 1 (2^{nd} bound)

The distance from s to t is $\geq 2n^{2/3}$

➔ $\exists i$ s.t. $|V_i| \geq n^{1/3}$ or $|V_{i+1}| \geq n^{1/3}$

➔ The number of remaining blocking flow $\leq n^{2/3}$



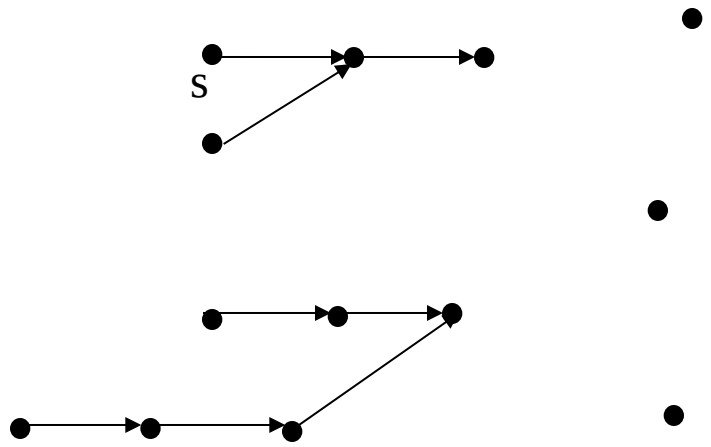
Summary: When all capacities are 1

Dinic's algorithm runs in $O(\min(\sqrt{m}, n^{2/3})m)$ time

Finding a blocking flow using dynamic trees

Spend less time for each edge saturation

Maintain a collection of trees (subgraph of L) where each vertex v has at most one unsaturated edge going out of v. Each edge has weight which is its current residual capacity



Start such that each vertex is in a separate tree.

Dinic (fast implementation)

Advance: v = findroot(s), if v=t goto augment.
if there is no edge going out of v goto retreat
else choose an edge (v, w) going out of v and perform link(v,w,r(v,w))
goto advance

Augment: (v,c) = mincost(s), addcost(s,-c), goto delete.

delete: delete the edge (v, p(v)) from the graph. f(v,p(v)) = u(v,p(v)).
cut(v).
(v,c) = mincost(s). If (c=0) goto delete else goto advance.

retreat: (All paths from v to t are blocked)
if (v=s) halt.
Delete every edge (u,v) from the graph.
If ($v \neq p(u)$) then f(u,v) = 0.
Otherwise, let (u, Δ) = mincost(u), f(u,v) = u(u,v) - Δ, cut(u).
Goto advance.

How many such operations do we do?

We can associate each operation that we do with an edge that is added to or deleted from the forest.

Since each edge is added or deleted once from the forest we do O(m) operations on the trees.

We know how to do each such operation in O(log(n)) time

So we get an O(nmlog(n)) implementation of Dinic's algorithm.