Max Flow with Level Graphs

CMSC 641 Design & Analysis of Algorithms

Defn: Let f be a flow in a flow network G.

The *level graph* LG_f is a breadth-first search graph of the residual graph G_f with back edges & "sideways" edges deleted.

Cross edges from level i to level $i+1$ are kept.

Modified Edmonds-Karp

Given: flow network $G=(V,E)$ and $c: E \to \mathbb{R}$
Initial flow $f=0$

phase

Construct residual graph G_f

Construct level graph LG_f
Stop if t is not reachable from s

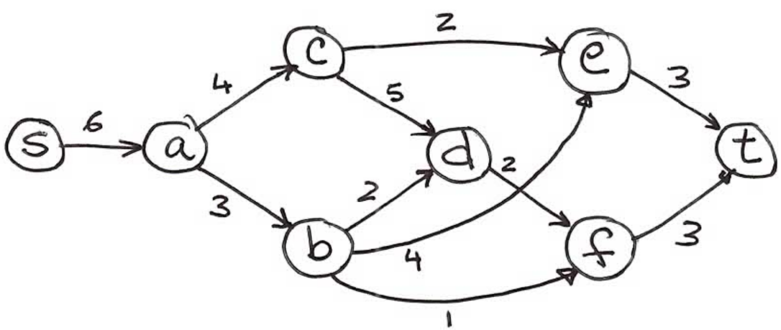
iteration

Add path flow of any shortest path in LG_f
Update capacities in LG_f, delete saturated edges
Update total flow f.
Repeat until t is disconnected from s

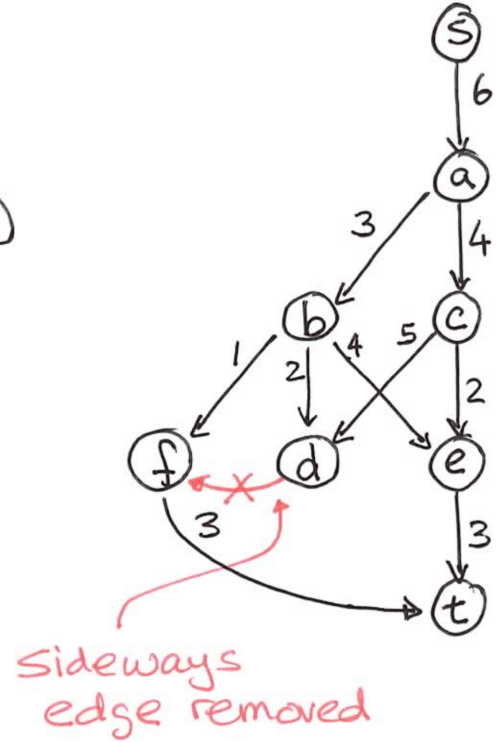
blocking flow

Repeat

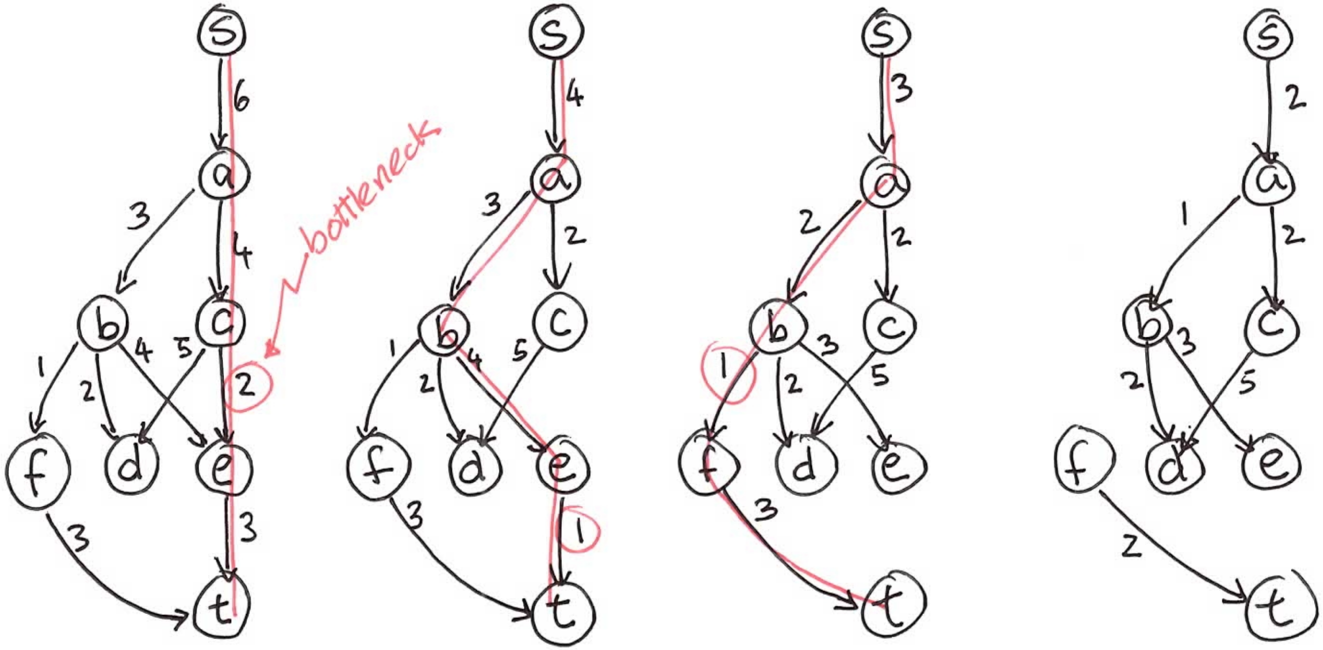
Example:



s
a
b
c
d
e
f
t
6
4
3
2
5
2
2
4
1
3
3

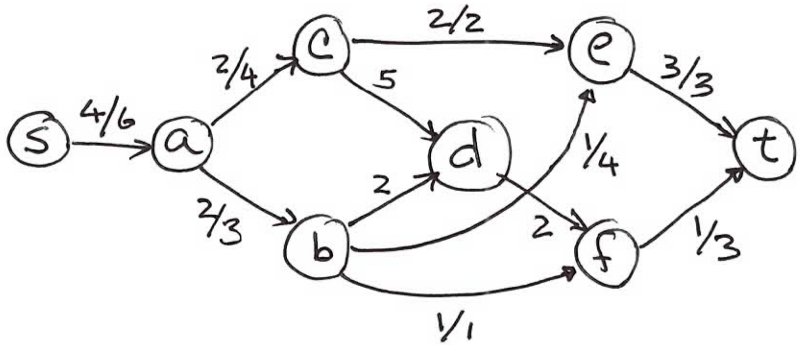


s
a
b
c
d
e
f
t
6
3
4
1
2
4
5
2
3
3
sideways edge removed

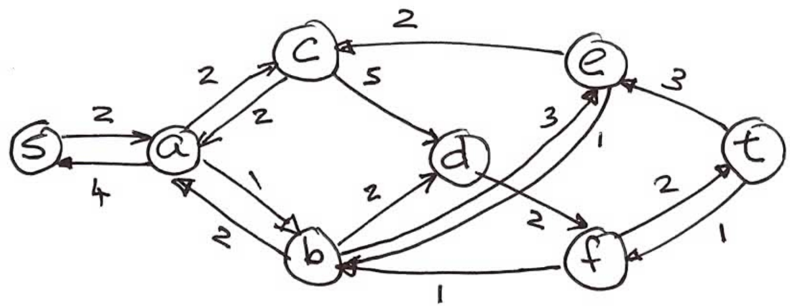


blocking flow

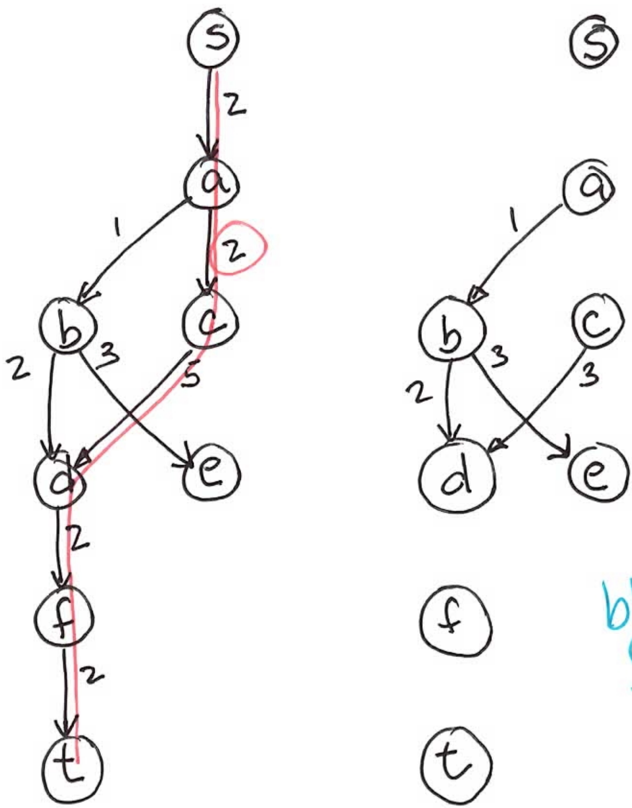
Resulting flow:



Residual Graph:

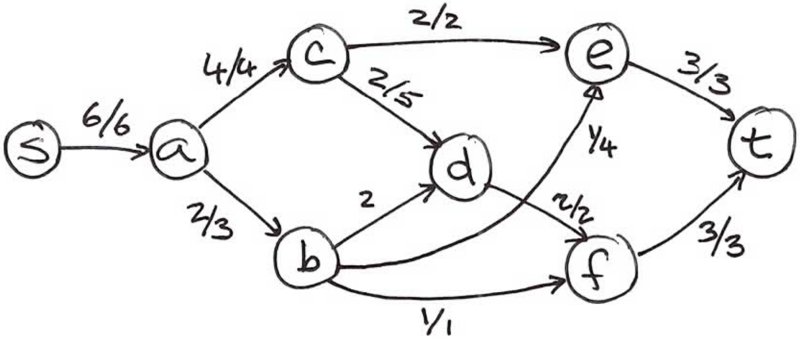


New Level Graph

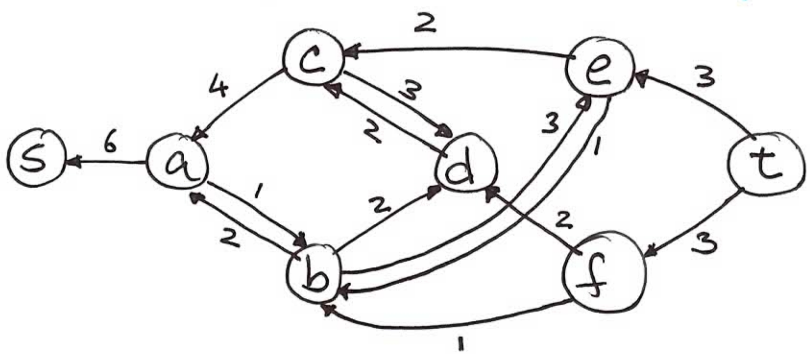


blocking flow

Final flow:



Final Residual Graph:



s cannot reach t. Done!

Defn: Let $\delta_f(a,b)$ = minimum link distance from a to b in G_f

Lemma 1: If f' is obtained from f by augmenting thru shortest paths, then $\forall a \in V$

$$\delta_f(a,t) \leq \delta_{f'}(a,t) \text{ and } \delta_f(s,a) \leq \delta_{f'}(s,a).$$

Pf: (same as before)

previous proof did not depend on starting at source s.

Lemma 2: Let f' be the new flow after a phase that started with flow f. Then, $\delta_{f'}(s,t) \geq \delta_f(s,t) + 1$.

Pf: Let $d = \delta_f(s,t)$.

In $LG_{f'}$, there must be a path from s to t with distance $\geq d$. (By Lemma 1)

Some edge (u,v) on this path must not be in LG_f

$$s \rightsquigarrow u \rightarrow v \rightsquigarrow t$$

(Otherwise, the previous phase did not end with a blocking flow.)

Thus, (u,v) must be a sideways edge or a back edge in the BFS of G_f

Then, $\delta_f(s,u) \geq \delta_f(s,v)$. Therefore,

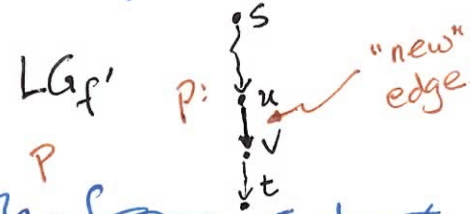
$$\delta_{f'}(s,t) \geq \delta_{f'}(s,u) + 1 + \delta_{f'}(v,t) \geq \delta_f(s,u) + \delta_f(v,t) + 1$$

$$\geq \delta_f(s,v) + \delta_f(v,t) + 1 \geq d+1$$

$\geq$

since $\forall a \in V$ $d \leq \delta_f(s,a) + \delta_f(a,t)$

Lemma 2: Let f' be the new flow after a phase that started with flow f. Then, $\delta_{f'}(s,t) \geq \delta_f(s,t)+1$.



Pf: Let $d = \delta_f(s,t)$

In $LG_{f'}$, there must be a path p from s to t with distance $\geq d$. (Lemma 1.)

Claim A: Some edge (u,v) on this path p is not an edge in LG_f.

Claim B: In LG_f, $\delta_f(s,u) \geq \delta_f(s,v)$

Proof of Claim A: (by contradiction)

prior flow's level graph

Suppose every edge on path p in an edge in LG_f.

Then, p is a path in LG_f.

Note: every path in LG_f is a $\overbrace{\text{shortest path}}^{\text{min-link}}$.

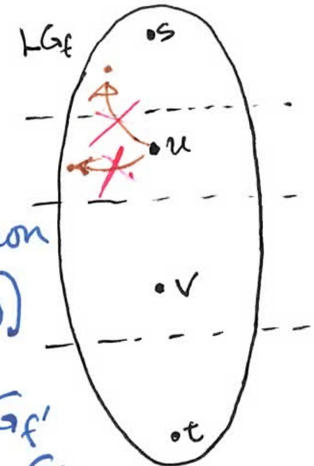
Also, flow f' does not saturate edges along p.

I.e., for each edge (u,v) on p, $f'(u,v) < c(u,v)$.

Because if $f'(u,v) = c(u,v)$, then (u,v) is not an edge in LG_f! Thus, previous phase did not end with a blocking-flow. $\Rightarrow\!\Leftarrow$

Proof of Claim B: (by contradiction)

Let (u,v) be an edge in p that is not in LG_f.



Suppose $\delta_f(s,u) < \delta_f(s,v)$.

Then (u,v) was not a cross edge or back edge removed in the construction of LG_f. (That would mean $\delta_f(s,u) \geq \delta_f(s,v)$)

But, the only way for (u,v) to "appear" in $LG_{f'}$ when it was not an edge in LG_f, is if flow was sent from v to u and $LG_{f'}$ has the option to send it back.

However, an augmenting flow in LG_f never sends flow from bigger $\delta_f(\,)$ to smaller $\delta_f(\,)$. $\Rightarrow\Leftarrow$

Finish proof of Lemma 2:

We know (u,v) is an edge on path p of $LG_{f'}$ **and** (u,v) is not in LG_f **and** $\delta_f(s,u) \geq \delta_f(s,v)$.

$\delta_f(\,)$ not $\delta_{f'}(\,)$

$$\delta_{f'}(s,t) = \delta_{f'}(s,u) + 1 + \delta_{f'}(v,t)$$

go from s to u, (u,v) and then from v to t.

= length of shortest path because p is a path in $LG_{f'}$, where all paths are shortest paths

$$\geq \delta_f(s,u) + 1 + \delta_f(v,t) \text{, Lemma 1.}$$

$$\geq \delta_f(s,v) + 1 + \delta_f(v,t) \text{, Claim B.}$$

$$= \delta_f(s,v) + \delta_f(v,t) + 1 \text{, regrouping}$$

$$\geq \delta_f(s,t) + 1$$

length of shortest path is ≤ length of some path

$$= d+1$$

*** Recall that δ_f is distance in residual graph G_f, not in LG_f

Running Time for modified Edmonds-Karp

Within each iteration:

- $O(E)$ time to find an augmenting path
- $O(E)$ time to update capacities & flow

# of iterations per phase is $O(E)$

- each iteration saturates and deletes at least 1 edge
- $|E_f| \leq 2|E|$ always!

# of phases $\leq |V|$

- $\delta_f(s,t)$ increases by 1 after each phase

Total time $O(VE^2) \approx O(V^5)$

Dinitz Algorithm:

Idea: find multiple augmenting paths to save time.

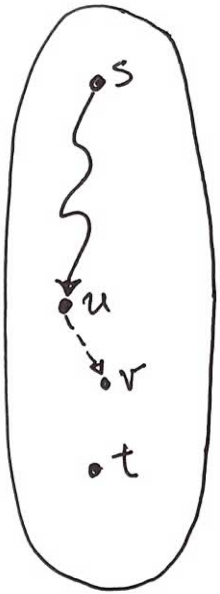
4 operations:

Initialize: set up level graph

Advance: make path p longer by adding 1 vertex

Retreat: process dead ends

Augment: reached sink t. Update flow.



Operations in Dinitz:

Initialize

Construct a new level graph LG

$u := s$ // s = source

$p := [s]$ // path p w/ one vertex s

Goto Advance

Advance

If u has no edges out, goto Retreat

O.w. let (u,v) be an edge

$p := p \cdot [v]$ // add v to end of path p

$u := v$ // update "current" vertex

If $v \neq t$, goto Advance. If $v = t$, goto Augment

Retreat

If $u=s$, halt.

O.w. delete u and all adjacent edges in LG_f

remove u from p

let $u=$ last vertex on p.

Goto Advance

Augment

- let Δ be the bottleneck capacity along p.
- Augment flow using p as augmenting path
- Adjust residual capacities along p.
- Delete any newly saturated edges.
- Let u = last vertex on path p still reachable from s.
- Goto Advance

Claim: Each phase of Dinitz Algorithm takes $O(VE)$ time.

Corollary: Total time for Dinitz Algorithm is $O(VE) \times V$ phases $= O(V^2 E) = O(V^4)$

for dense graphs

Pf: Potential function $\Phi = 2 \cdot n(G) \cdot e(G) - |P|$

# of vertices

# of edges

length of path from s to current vertex

Initialize: $O(VE)$ amortized time ← Initialize pays for everything

$|V|$ vertices created

$|E|$ edges created

initial path length $= 0$

Advance: 0 amortized time ← yes, we want 0

looks for one outgoing edge. $O(1)$ real time.

$|P|$ increases by 1

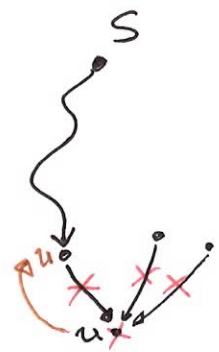
Φ decreases by 1, release 1 credit

to do $O(1)$ work.

Recall: $\Phi = 2 \cdot n(G) \cdot e(G) - |P|$

Retreat: $\emptyset$ amortized time

- Real time proportional to $\text{indeg}(u)$.
- At least 1 edge removed, releasing $2 \cdot n(G)$ credits to pay for $c \cdot \text{indeg}(u)$ work.



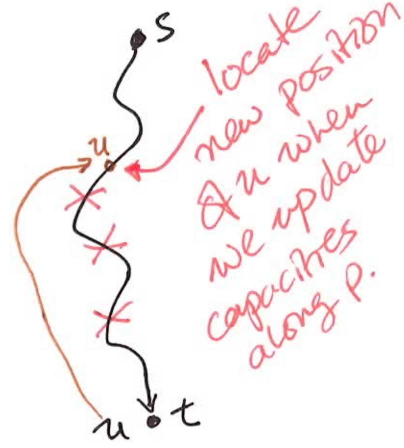
Recall: $\Phi = 2 \cdot n(G) \cdot e(G) - |p|$

Augment: $\emptyset$ amortized time

- Real time proportional to $|p|$
- Path length shortened, increasing the potential by $n(G)$ in the worst case
- At least 1 edge is saturated and removed, releasing $2 \cdot n(G)$ credits.

  Use $n(G)$ to offset reduction in path length

  Use $n(G)$ to update capacities along P.

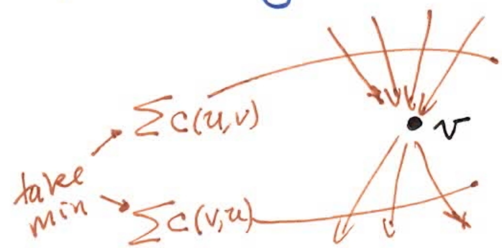


MPM Algorithm (1978)

Malhorta, Pramodh-Kumar, Maheshwari

$O(n^3)$ time matches preflow-push alg in textbook

Uses Fibonacci Heaps.



<u>Defn</u>: capacity of a vertex

$$= \mathrm{cap}(v) = \min\left(\sum_{u \in V} c(u,v) \;,\; \sum_{u \in V} c(v,u) \right)$$

Build LG_f as before. In each phase:

Construct LG_f
Delete vertices not on any path from s to t (use DFS)
Calculate cap(v), store in Fibonacci Heap

continue until flow is blocked

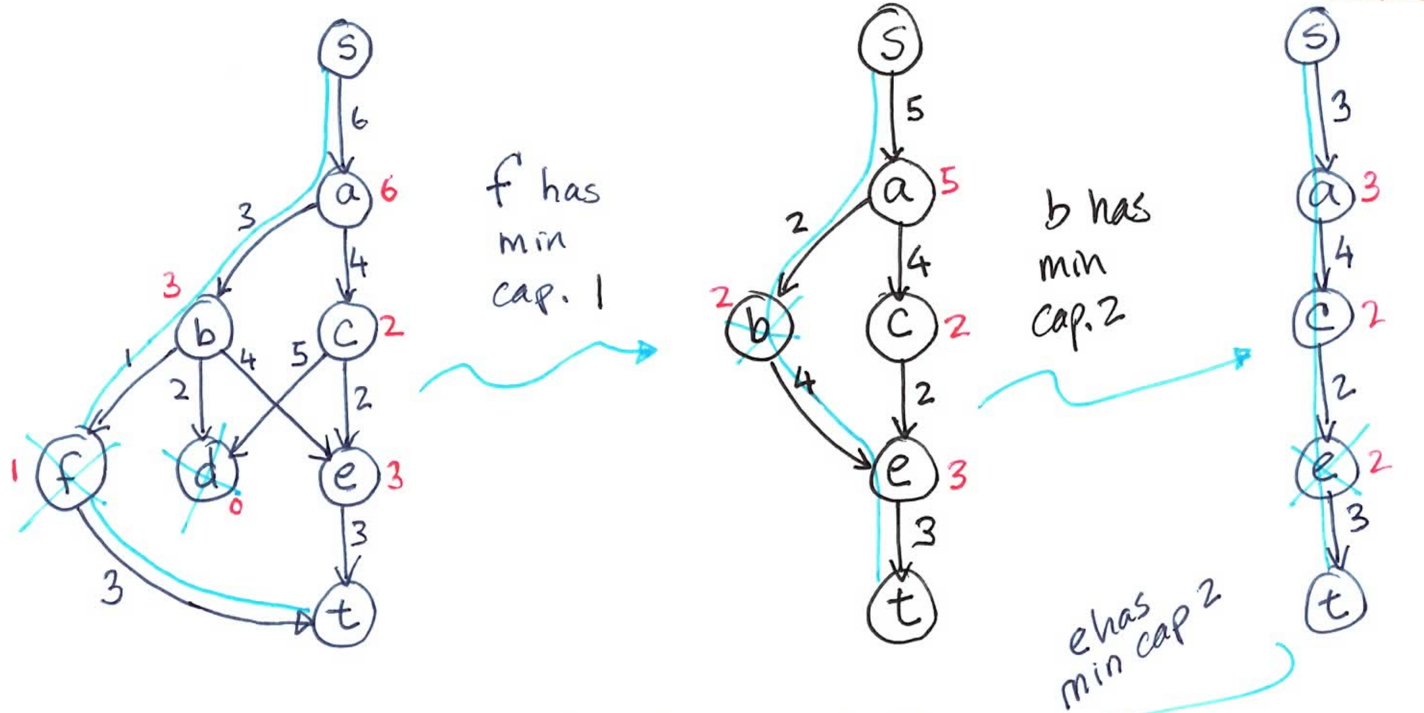
Pick vertex v with min capacity d

construct flow from s to v with d units — Start at v, pulling flow from each incoming edge. Saturate edges in turn, leaving only 1 partially filled edge at each vertex. Continue until s is reached.

construct flow from v to t with d units — Same as above, except push flow from v, instead of pulling.

Delete saturated edges, vertex v & edges incident on v
Update capacities of affected vertices (decrease key)

In this simple example, all augmentations are along paths.
In general, flow is augmented thru many paths at the same time.



No path from s to t, all vertices removed.

Running Time:

Amortized time per phase

$O(E+V)$ to compute LG_f, initialize Fib Heap, etc

$O(V \log V)$ for V calls to delete min

$O(E)$ to delete each saturated edge & to perform decrease key on affected vertices in $O(1)$ amort. time.

V^2 visits to partially filled edges. 1 edge per vertex per iteration.
\# of iterations $\leq V$, since one vertex is deleted each iteration.
Update edge capacities & vertex capacities (decrease key) in $O(1)$ time per visit. $V^2 \times O(1) = O(V^2)$

Total time per phase $O(V^2) \times V$ phases $= O(V^3)$ total time.