Algorithms 2020

Flows (cont)

Recap

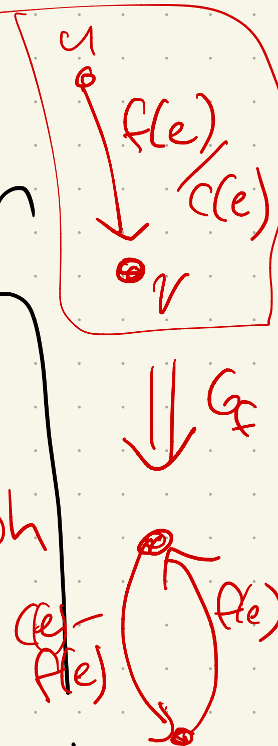
- Finish Ch 10 today, start 11 Wednesday
- HW is due Friday
- Reading as usual

Last time:

Thm: Max-flow = Min cut

$\Updownarrow$

Algorithm: Ford-Fulkerson (1956)



MAXFLOW(G):

  Let $f(e) = 0$ initially $\forall e$

  Compute G_f ← residual graph

  While there is s-t path in G_f:

    Let P be a simple s-t path, ← WFS

    $f' \leftarrow$ augment(f, P) ← + updating edges

    $f \leftarrow f'$

    update G_f

  return f

Push as much flow on that path as we can

Runtime:

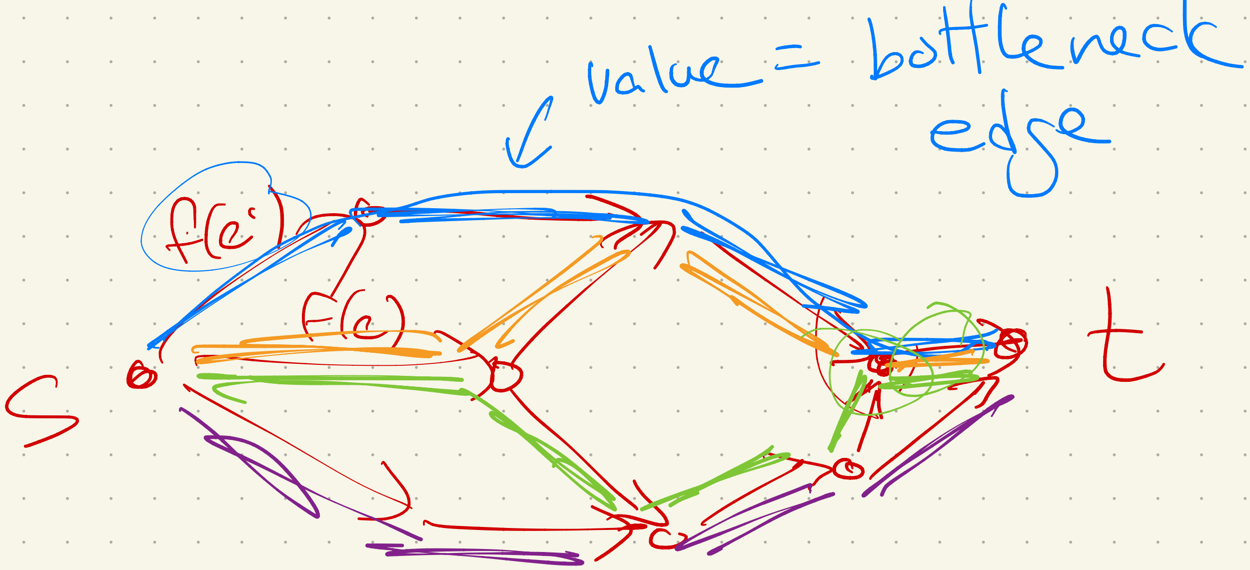
each time get ≥ 1 more flow

So repeats $O(f^*)$ ← max flow

Total: $O((V+E) f^*)$

Next:

Flow decomposes into paths.



[use F-F alg. to see these paths

Faster versions (?)

This is an active area of research!

We'll see two faster examples, both (relatively) simple variations on the Ford-Fulkerson algorithm:

1. Edmonds-Karp: choose largest bottleneck edge
   ↳ $O(E^2 \log E \log |f^*|)$

   $\leftarrow$ # of edges

2. shortest augmenting path $\leftarrow$ BFS tree
   ↳ $O(VE^2)$

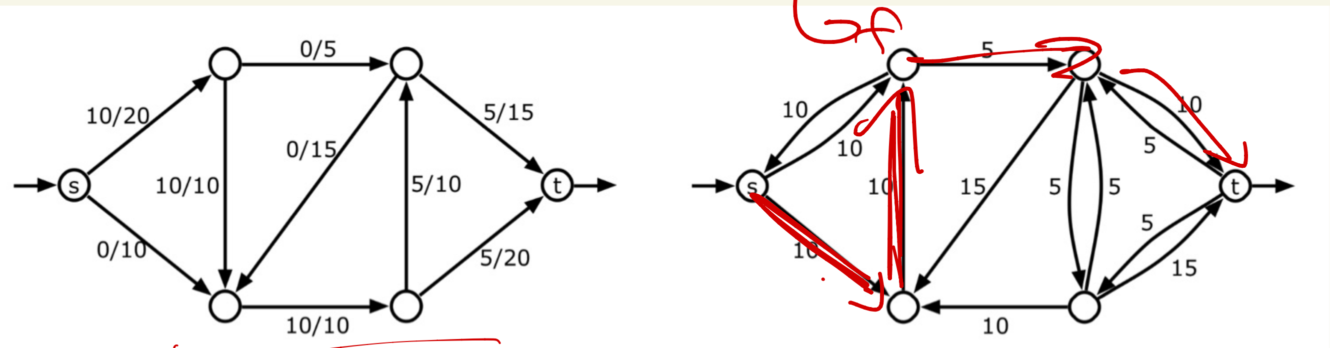
Edmonds-Karp:

Largest bottleneck:

how?

Take G_f:

G_f



A flow f in a weighted graph G and the corresponding residual graph G_f.

Grow a tree from s, adding largest edge out each time

Runtime: use heap

$E \log E$

E-K

~~MAXFLOW~~ (G):

Let f(e)=0 initially ∀e

Construct G_f

While there is s-t path in G_f:

~~Let P be a simple s-t path~~

f' ← augment (f, P)

f ← f'

update G_f

$O(E+V)$

return f

Let P be the largest bottleneck path in G_f

$O(E\log E)$ using heap

# of repetitions in loop?

In FF, went down by ≥ 1.

Here? Hopefully better!

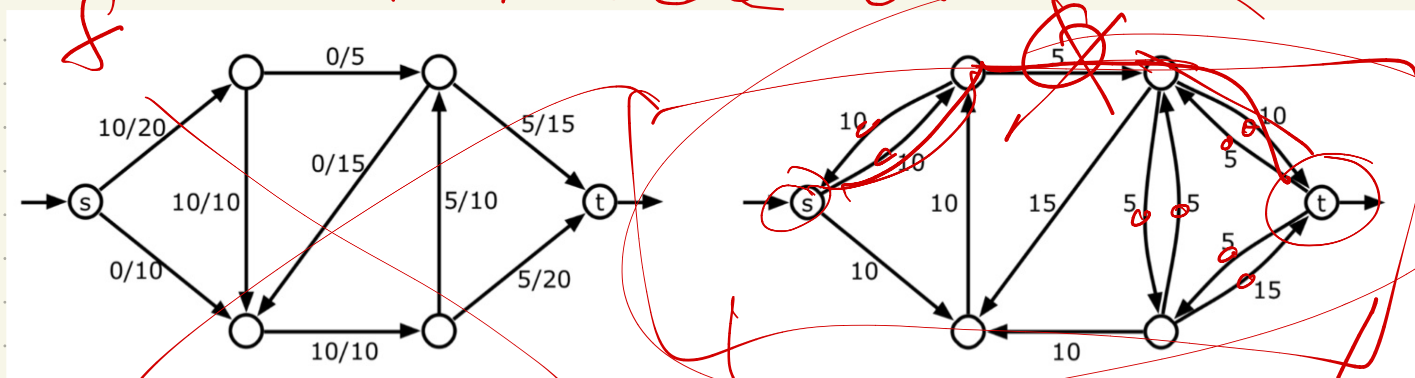
Key: look at current flow

$\#$ loop repetitions:

Consider current flow:

in middle of while

f



A flow f in a weighted graph G and the corresponding residual graph G_f.

G_f

Residual graph: Also a flow graph!

Let f' be max flow in G_f.

G_f has f' flow & at most E paths from s to t. ← decompose into paths

$\Rightarrow$ one of them is "big"

$P_1 \; P_2 \; P_3 \cdots P_E$

sum $= f^*$

$$\boxed{\geq \frac{E}{f'}}$$

if all $< \frac{E}{f^*}$, then sum would be too small.

So: path must ~~adds~~ $\geq \frac{f'}{E}$ to flow

push more flow along path

$\Rightarrow$ next residual graph

$f' - \frac{f'}{E}$

will have $\leq \boxed{\left(1-\frac{1}{E}\right)f'}$

in it.

push at least enough to be below this

$\hookrightarrow \leq \left(1-\frac{1}{|E|}\right)^{\ell} \cdot f^*$ ~~repetitions~~ flow

after ℓ repetitions

What is ℓ?

rounds

Well, $\ell = \boxed{E \cdot \ln f^*}$ repetitions,

$$\boxed{\left(1-\frac{1}{|E|}\right)^{\ell} f^* < 1}$$

math!

Wait: <1??

integrel capacities!

so if <1, must $=0$

$\Rightarrow$ resid. graph is now disconnected

So: total runtime:

EK(G):

Let $f(e)=0$ initially $\forall e$

Construct G_f

While there is s-t path in G_f:

~~Let P be a simple s-t path~~

$f' \leftarrow$ augment(f, P)

$f \leftarrow f'$

update G_f

return f

Let P be the largest bottleneck path

Time: $E \log E + O(V+E)$

repetitions: $E \ln f^*$

Total: $E^2 \log E \lg f^*$

Shortest paths

MAXFLOW(G):

Let $f(e)=0$ initially $\forall e$
Construct G_f

While there is s-t path in G_f
~~Let P be a simple s-t path~~
$f' \leftarrow$ augment(f, P)
$f \leftarrow f'$
update G_f

return f

$O(V+E)$

strange - no mention of amt of flow

Let P = shortest s-t path (# edges, not capacities)

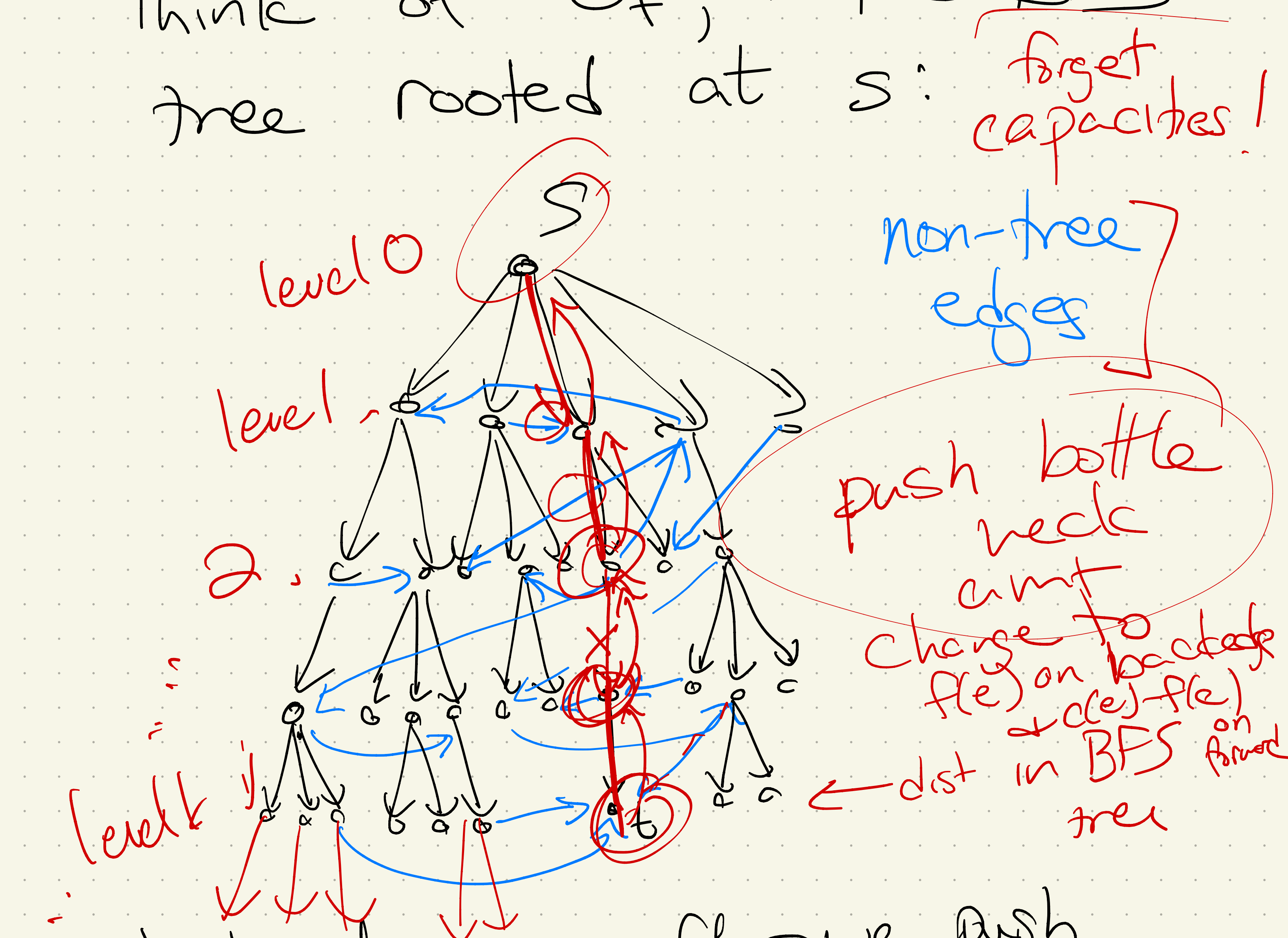
Which traversal?

BFS : $O(E+V)$

Q: loop??

Q: How many times do we need a path?
(ie: how many repetitions of the while loop?)

Think of G_f, + the BFS tree rooted at s:



what changes after we push flow?

Let G_0 = initial G_f

& G_i = residual graph after i repetitions of the loop.

G_i has a BFS T_i tree, so let $\text{level}_i(v)$ = depth in tree

0 to $\leq |V|-1$

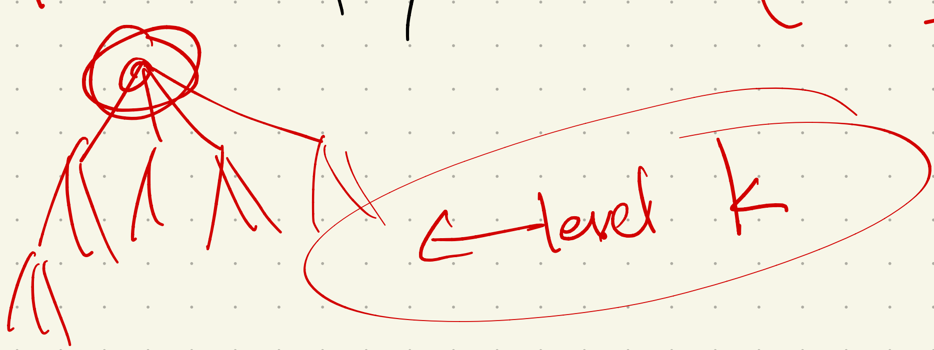
(Note: once s can't reach t, then level(t) = ∞.)

Claim: levels only get bigger ($\geq$) in each round. in G_i

proof: induction on level. (fix i)

base case $k=0$: s (level 0), always stays ≥ 0 $(=0)$

G_i



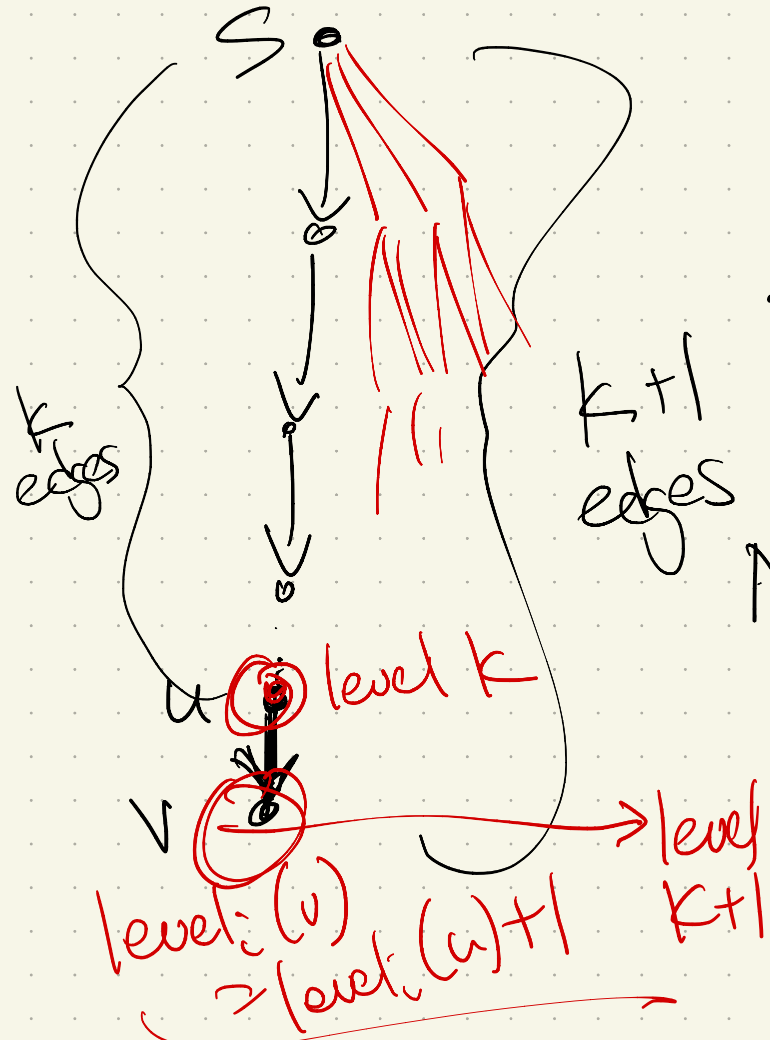
IH: consider levels $\leq k$ in G_i. IH: for any such u on level $\leq k$,

$\text{level}_{i-1}(u) \leq \text{level}_i(u)$

in G_{i-1} in G_i

IS: now take v on level $k+1$ of G_i:

Must be a path $s \rightsquigarrow v$



take u just before v on path

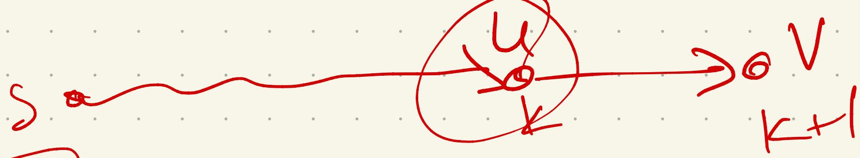
$\Rightarrow$ by IH,

$\text{level}_{i-1}(u) \leq \text{level}_i(u)$

Now: how did we get this path in G_i?

NTS: holds for v

cont:



→ $\boxed{u \to v}$ is an edge in G_{i-1}:

$$level_{i-1}(v) \le level_{i-1}(u)+1$$

$$\boxed{level_i(u) \ge level_{i-1}(u) \quad \therefore \to v \text{ satisfies property}}$$

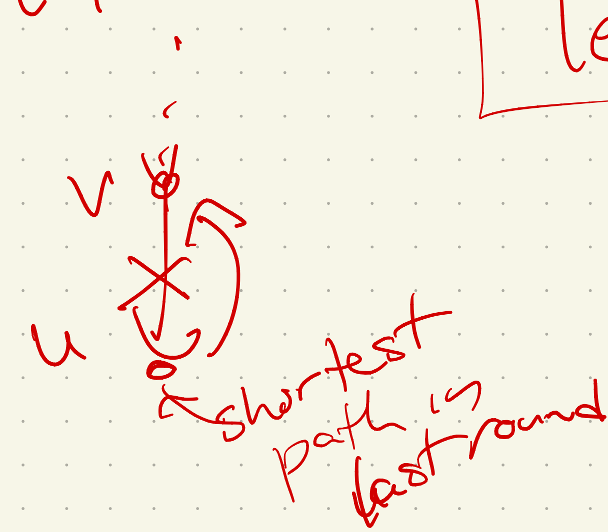
- might have come from pushing in G_{i-1}:

here, it came from pushing a shortest path in the last round, so $\boxed{v \to u}$ was used last round.

$i-1$

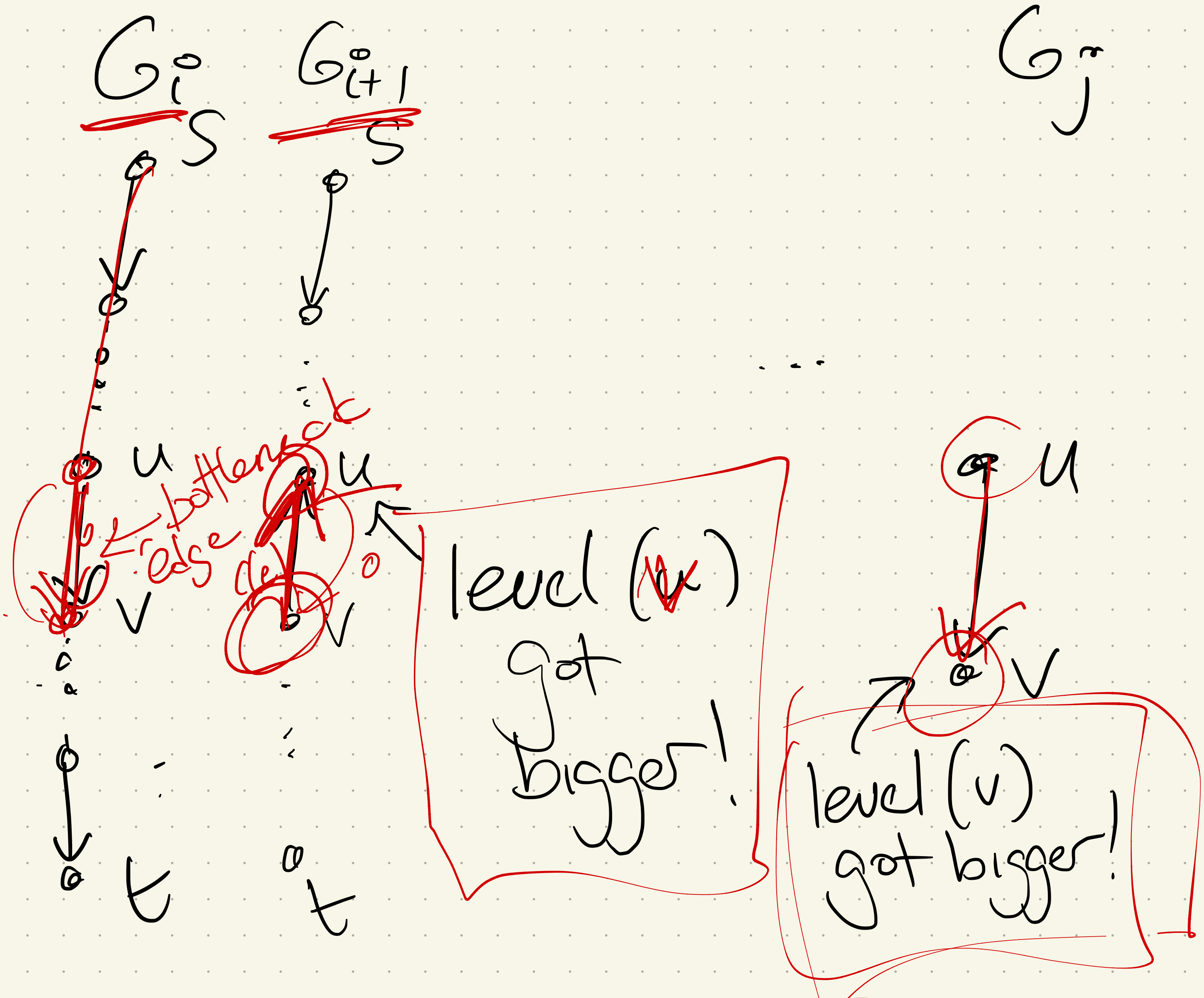
$$\boxed{level_{i-1}(u) = level_{i-1}(v)} + 1$$

$$\le level_{i-1}(u)+1$$



Then: Edges can disappear & reapper.

How many times?



If level increases by $\boxed{+2}$:

after $V/2$ changes, must be ∞

In each iteration of the loop — some edge disappears!

$\Rightarrow \boxed{\frac{V}{2}} \cdot E$ repetitions

Total: $\frac{V}{2} \cdot E\,(V+E)\ \cancel{\Theta}(E^2 V)$

```
MAXFLOW(G):
    Let f(e)=0 initially ∀e
    Construct G_f

    While there is s-t path in G_f
        ~~Let P be a simple s-t path~~
        f' ← augment(f, P)
        f ← f'
        update G_f

    return f
```

Let P = shortest s-t path (# edges, not capacities)

And... not done!

Technique	Direct	With dynamic trees	Source(s)
Blocking flow	$O(V^2E)$	$O(VE\log V)$	[Dinitz; Karzanov; Even and Itai; Sleator and Tarjan]
Network simplex	$O(V^2E)$	$O(VE\log V)$	[Dantzig; Goldfarb and Hao; Goldberg, Grigoriadis, and Tarjan]
Push-relabel (generic)	$O(V^2E)$	—	[Goldberg and Tarjan]
Push-relabel (FIFO)	$O(V^3)$	$O(VE\log(V^2/E))$	[Goldberg and Tarjan]
Push-relabel (highest label)	$O(V^2\sqrt{E})$	—	[Cheriyan and Maheshwari; Tunçel]
Push-relabel-add games	—	$O(VE\log_{E/(V\log V)} V)$	[Cheriyan and Hagerup; King, Rao, and Tarjan]
Pseudoflow	$O(V^2E)$	$O(VE\log V)$	[Hochbaum]
Pseudoflow (highest label)	$O(V^3)$	$O(VE\log(V^2/E))$	[Hochbaum and Orlin]
Incremental BFS	$O(V^2E)$	$O(VE\log(V^2/E))$	[Goldberg, Held, Kaplan, Tarjan, and Werneck]
Compact networks	—	$\boxed{O(VE)}$ dt	[Orlin]

Figure 10.10. Several purely combinatorial maximum-flow algorithms and their running times.

use this

Many use very different techniques

- linear programming
- complex data structures
- not residual graphs

Still active!

Showing 1–4 of 4 results

Search v0.5.6 released 2020-02-24 Feedback?

Query: order: -announced_date_first; size: 50; classification: Computer Science (cs); include_cross_list: True; terms: AND title=Max flow algorithm

Simple Search

Refine query New search

50 results per page. Sort results by Announcement date (newest first) Go

1. arXiv:2009.03260 [pdf, ps, other] cs.DS math.OC

   A Potential Reduction Inspired Algorithm for Exact Max Flow in Almost $\widetilde{O}(m^{4/3})$ Time

   Authors: Tarun Kathuria

   Abstract: We present an algorithm for computing s-t maximum flows in directed graphs in $\widetilde{O}(m^{4/3+o(1)}U^{1/3})$ time. Our algorithm is inspired by potential reduction interior point methods for linear programming. Instead of using scaled gradient/Newton steps of a potential function, we take the step which maximizes the decrease in the potential value subject to advancing a certain amount o... ▽ More

   Submitted 7 September, 2020; **originally announced** September 2020.

2. arXiv:1910.04848 [pdf, other] cs.DS cs.CC

   A Fast Max Flow Algorithm

   Authors: James B. Orlin, Xiao-Yue Gong

   Abstract: In 2013, Orlin proved that the max flow problem could be solved in $O(nm)$ time. His algorithm ran in $O(nm + m^{1.94})$ time, which was the fastest for graphs with fewer than $n^{1.06}$ arcs. If the graph was not sufficiently sparse, the fastest running time was an algorithm due to King, Rao, and Tarjan. We describe a new variant of the excess scaling algorithm for the max flow problem whose runn... ▽ More

   Submitted 10 October, 2019; **originally announced** October 2019.

   Comments: 35 pages

3. arXiv:1901.01412 [pdf, other] cs.DS

   New Algorithms and Lower Bounds for All-Pairs Max-Flow in Undirected Graphs

   Authors: Amir Abboud, Robert Krauthgamer, Ohad Trabelsi

   Abstract: We investigate the time-complexity of the All-Pairs Max-Flow problem: Given a graph with n nodes and m edges, compute for all pairs of nodes the maximum-flow value between them. If Max-Flow (the version with a given source-sink pair s, t) can be solved in time $T(m)$, then an $O(n^2) \cdot T(m)$ is a trivial upper bound. But can we do better? For directed graphs, recent results in fine-grai... ▽ More

   Submitted 9 July, 2019; **v1** submitted 5 January, 2019; **originally announced** January 2019.

4. arXiv:1304.2338 [pdf, other] cs.DS

   An Almost-Linear-Time Algorithm for Approximate Max Flow in Undirected Graphs, and its Multicommodity Generalizations

   Authors: Jonathan A. Kelner, Yin Tat Lee, Lorenzo Orecchia, Aaron Sidford

   Abstract: In this paper, we introduce a new framework for approximately solving flow problems in capacitated, undirected graphs and apply it to provide asymptotically faster algorithms for the maximum s-t flow and maximum concurrent multicommodity flow problems. For graphs with n vertices and m edges, it allows us to find an ε-approximate maximum s-t flow in time $O(m^{1+o(1)}\varepsilon^{-2})$, improvi... ▽ More

   Submitted 23 September, 2013; **v1** submitted 8 April, 2013; **originally announced** April 2013.

Even more known about "special" graphs:

Computer Science > Data Structures and Algorithms

Minimum Cuts in Surface Graphs

Erin W. Chambers, Jeff Erickson, Kyle Fox, Amir Nayyeri

(Submitted on 9 Oct 2019)

We describe algorithms to efficiently compute minimum (s, t)-cuts and global minimum cuts of undirected surface-embedded graphs. Given an edge-weighted undirected graph G with n vertices embedded on an orientable surface of genus g, our algorithms can solve either problem in $g^{O(g)} n \log\log n$ or $2^{O(g)} n \log n$ time, whichever is better. When g is a constant, our $g^{O(g)} n \log\log n$ time algorithms match the best running times known for computing minimum cuts in planar graphs.
Our algorithms for minimum cuts rely on reductions to the problem of finding a minimum-weight subgraph in a given $\mathbb{Z}_2$-homology class, and we give efficient algorithms for this latter problem as well. If G is embedded on a surface with b boundary components, these algorithms run in $(g+b)^{O(g+b)} n \log\log n$ and $2^{O(g+b)} n \log n$ time. We also prove that finding a minimum-weight subgraph homologous to a single input cycle is NP-hard, showing it is likely impossible to improve upon the exponential dependencies on g for this latter problem.

A note from Ch 11

Using flows:

A few common themes:

1. Matchings:

   Identify a way to pair up items.

   More pairs $\Rightarrow$ larger flow

2. Disjoint paths:

   Find paths that avoid each other.

3. "Tuple" Selection

Example:

Sham-Poobanana University has hired you to write an algorithm to schedule their final exams. There are n different classes, each of which needs to schedule a final exam in one of r rooms during one of t different time slots. At most one class's final exam can be scheduled in each room during each time slot; conversely, classes cannot be split into multiple rooms or multiple times. Moreover, each exam must be overseen by one of p proctors.[4] Each proctor can oversee at most one exam at a time; each proctor is available for only certain time slots; and no proctor is allowed oversee more than 5 exams total. The input to the scheduling problem consists of three arrays:

- An integer array $E[1..n]$ where $E[i]$ is the number of students enrolled in the ith class.
- An integer array $S[1..r]$, where $S[j]$ is the number of seats in the jth room. The ith class's final exam *can* be held in the jth room if and only if $E[i] \le S[j]$.
- A boolean array $A[1..t, 1..p]$ where $A[k, \ell] = \text{True}$ if and only if the ℓth proctor is available during the kth time slot.[5]

Idea: Build a graph!

any valid assignment $\Rightarrow$ path in G

path in G will be a "valid" assignment