CSE 373

DECEMBER 1ST – GRAPH MADNESS

ASSORTED MINUTIAE

- **Project 3**
  - Maximum of 3 late days
  - Resubmission for all 3 parts by next Wednesday
- **Written Assignment**
  - Extra Credit
- **Next week:**
  - Monday office hours: 12:00-2:00 in my office
  - No office hours next Friday: email me to make an appointment

TODAY'S LECTURE

- **Isometric Graphs**
- **Last graphs problem**
  - Network Flow (Disclaimer)
- **Graph problem symmetry**

NEXT WEEK

- Algorithm Design
- Computability and Complexity
- Exam Review

FINAL EXAM

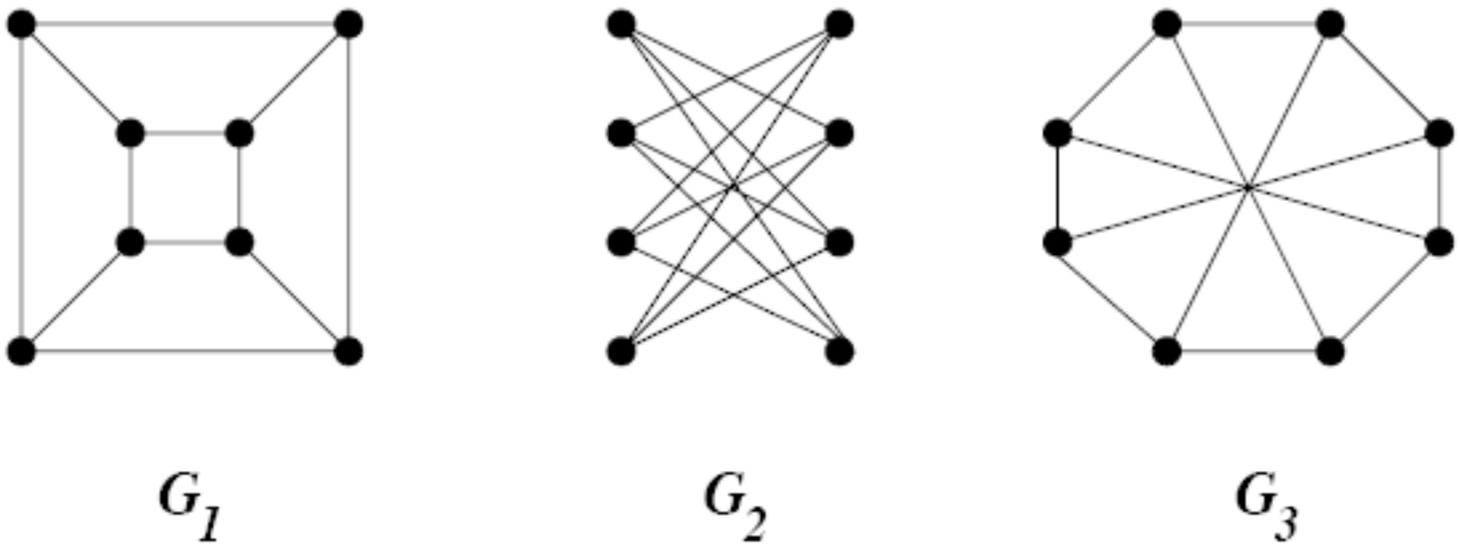
- **Topics list out this weekend**
- **Tue; December 12, 2017, 2:30-4:20**
  - Kane 220
- **Section, exam review**
- **Next Friday, exam review**
- **Practice Exam by next Tuesday**

GRAPHS

- **Talked a lot about graph representations**
- **Runtimes and memory**
- **How difficult can graphs be?**
    - Is it easier or more difficult to understand certain parts?

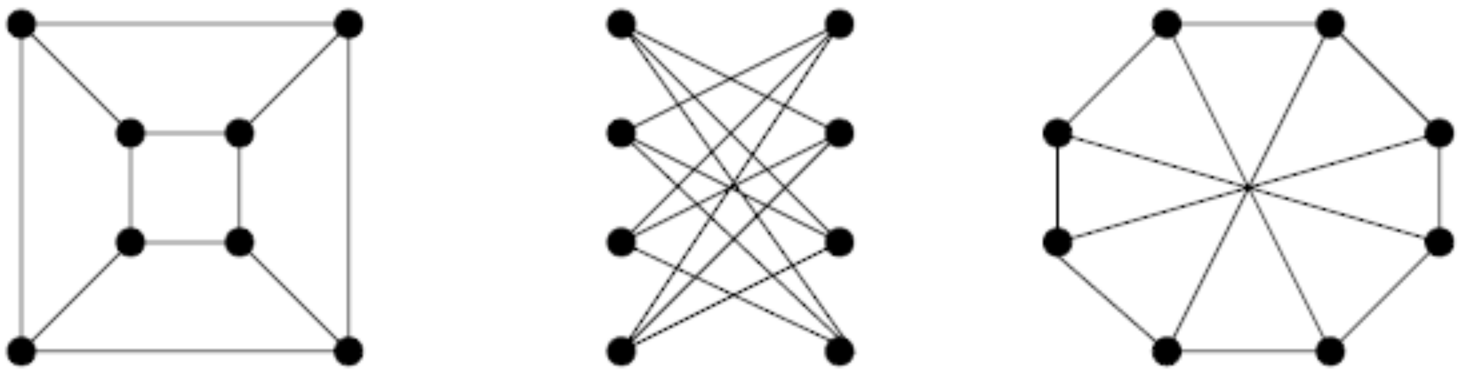
GRAPHS

- Which of these 3 graphs do you think would be easiest to run Dijkstra's algorithm on?



GRAPHS

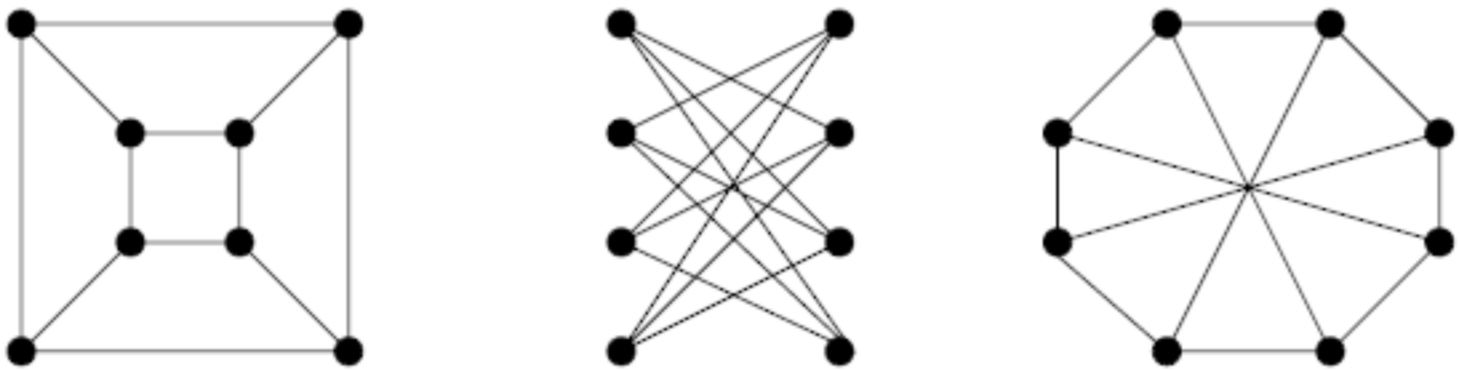
- **Which of these 3 graphs do you think would be easiest (for the computer) to run Dijkstra's algorithm on?**



G_1 G_2 G_3

GRAPHS

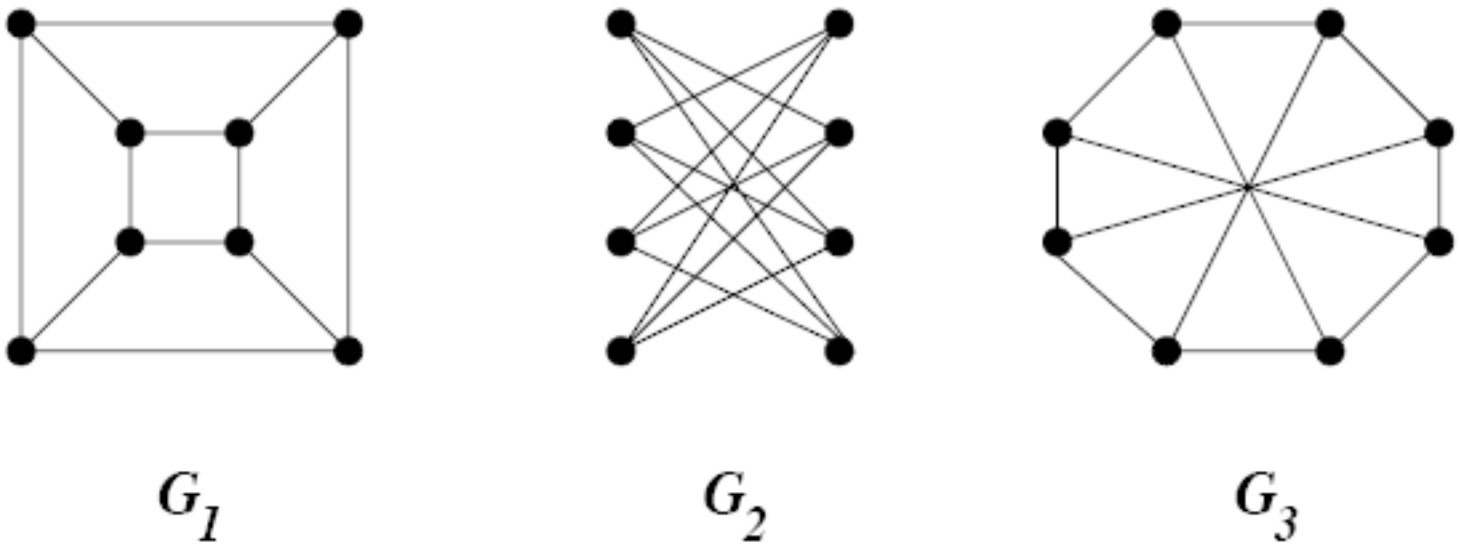
- G_1 and G_2 are the same graph, i.e. they are *isomorphic*



G_1 G_2 G_3

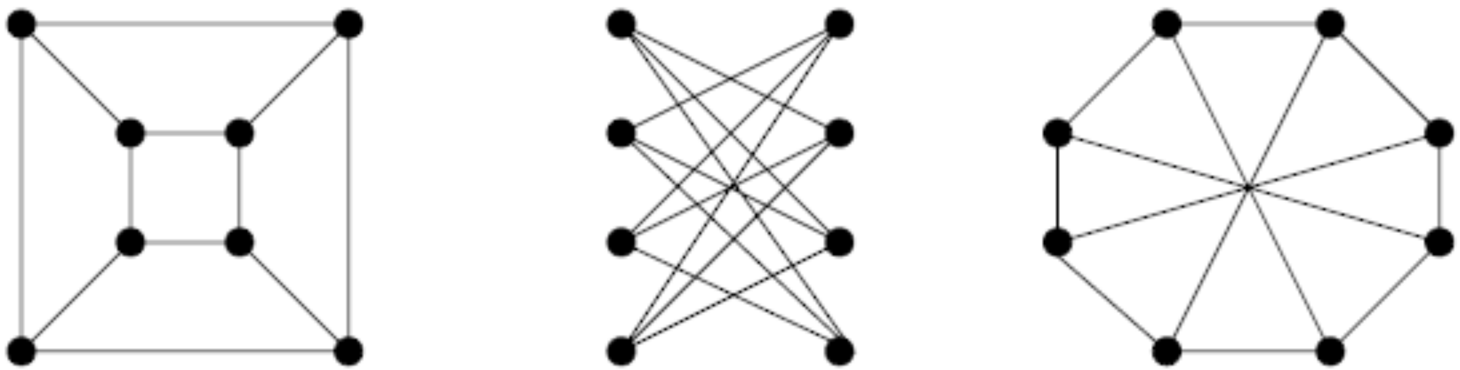
GRAPHS

- G_1 and G_2 are the same graph, i.e. they are *isomorphic*
- G_3 is not



GRAPHS

- G_1 and G_2 are the same graph, i.e. they are *isomorphic*
- G_3 is not. Can you prove it?



G_1 G_2 G_3

GRAPHS

- **Graphs have a sneaky way of appearing different all the time**
  - This isn't just true of the graph itself, but it can also be true of graph problems that we want to solve

GRAPHS

- **Graphs have a sneaky way of appearing different all the time**
  - This isn't just true of the graph itself, but it can also be true of graph problems that we want to solve
  - Makes graph theory incredibly interesting, but difficult to discuss

NETWORK FLOW

- Determine the maximum flow from a source vertex to a sink in a graph

NETWORK FLOW

- **Determine the maximum flow from a source vertex to a sink in a graph**
  - Graph: G(V,E)
  - Source vertex, *s*
  - Sink vertex, *t*
  - Each edge's weight represents the traffic a particular edge can carry (must be non-negative)

MAXIMUM FLOW

- **Consider breaking graph into two subgraphs**
  - $G(V,E_1)$ and $G(V,E_2)$ where $|E_1| = |E_2|$, but their weights are different
  - For each weight in E, $E_{1w}+E_{2w} = E_w$
  - The first is the **flow graph** and the second is the **residual graph**

MAXIMUM FLOW

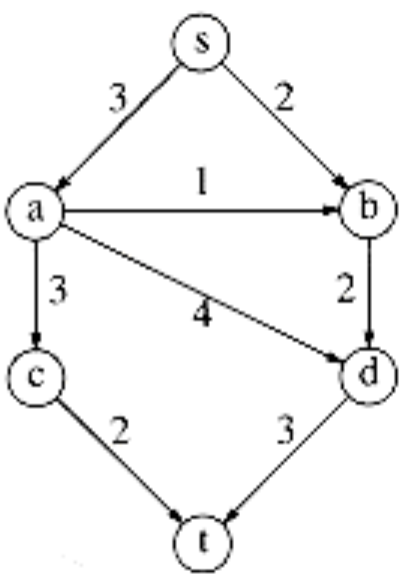
- **Consider breaking graph into two subgraphs**
  - $G(V,E_1)$ and $G(V,E_2)$ where $|E_1| = |E_2|$, but their weights are different
  - For each weight in E, $E_{1w}+E_{2w} = E_w$
  - The first is the **flow graph** and the second is the **residual graph**

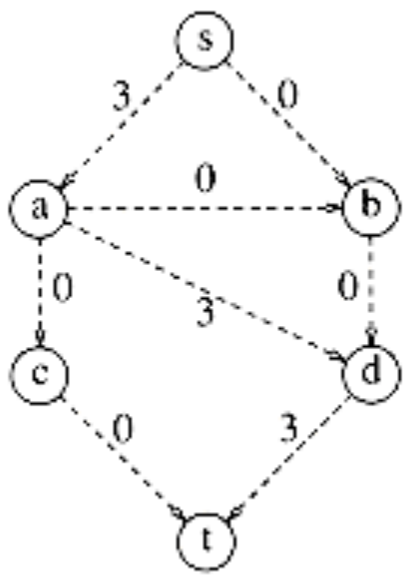
MAXIMUM FLOW

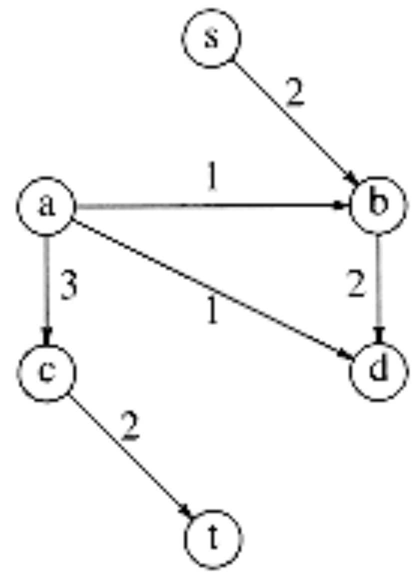
- **Consider breaking graph into two subgraphs**
  - For the flow graph, except the source and sink, the weights of all edges in must equal the weight of edges out
  - The residual graph can never have negative weights

MAXIMUM FLOW

Graph | **Flow** | **Residual**







NAÏVE ALGORITHM

- **Start where the the residual is the graph and the flow is empty**
- **While there is a path from *s* to *t* in the residual**
  - Find the minimum edge weight along the path
  - For each in the path
    - Add the minimum weight for each edge in the path to the flow
    - Subtract the minimum weight for each edge from the residual

EXAMPLE

EXAMPLE

- **What went wrong?**

EXAMPLE

- **What went wrong?**
  - If we select paths in the wrong order, we might not get the correct solution
  - This is an example of a greedy-first algorithm
  - Need to have an opportunity to back-track
  - Well, let’s add a reversal (augmenting) edge into the residual!

FORD-FULKERSON

Algorithm [edit]

Let $G(V,E)$ be a graph, and for each edge from u to v, let $c(u,v)$ be the capacity and $f(u,v)$ be the flow. We want to find the maximum flow from the source s to the sink t. After every step in the algorithm the following is maintained:

Capacity constraints:	$\forall (u,v) \in E \; f(u,v) \le c(u,v)$	The flow along an edge can not exceed its capacity.
Skew symmetry:	$\forall (u,v) \in E \; f(u,v) = -f(v,u)$	The net flow from u to v must be the opposite of the net flow from v to u (see example).
Flow conservation:	$\forall u \in V : u \neq s \text{ and } u \neq t \Rightarrow \sum_{w \in V} f(u,w) = 0$	That is, unless u is s or t. The net flow to a node is zero, except for the source, which "produces" flow, and the sink, which "consumes" flow.
Value(f):	$\sum_{(s,u) \in E} f(s,u) = \sum_{(v,t) \in E} f(v,t)$	That is, the flow leaving from s must be equal to the flow arriving at t.

This means that the flow through the network is a *legal flow* after each round in the algorithm. We define the **residual network** $G_f(V,E_f)$ to be the network with capacity $c_f(u,v) = c(u,v) - f(u,v)$ and no flow. Notice that it can happen that a flow from v to u is allowed in the residual network, though disallowed in the original network: if $f(u,v) > 0$ and $c(v,u) = 0$ then $c_f(v,u) = c(v,u) - f(v,u) = f(u,v) > 0$.

Algorithm Ford–Fulkerson

Inputs Given a Network $G = (V,E)$ with flow capacity c, a source node s, and a sink node t

Output Compute a flow f from s to t of maximum value

1. $f(u,v) \leftarrow 0$ for all edges (u,v)
2. While there is a path p from s to t in G_f, such that $c_f(u,v) > 0$ for all edges $(u,v) \in p$:
    1. Find $c_f(p) = \min\{c_f(u,v) : (u,v) \in p\}$
    2. For each edge $(u,v) \in p$
        1. $f(u,v) \leftarrow f(u,v) + c_f(p)$ (*Send flow along the path*)
        2. $f(v,u) \leftarrow f(v,u) - c_f(p)$ (*The flow might be "returned" later*)

The path in step 2 can be found with for example a breadth-first search or a depth-first search in $G_f(V,E_f)$. If you use the former, the algorithm is called Edmonds–Karp.

EXAMPLE

- **Oh boy, that got complicated really quickly**
  - $O(|V||E|^2)$

EXAMPLE

- **Oh boy, that got complicated really quickly**
  - $O(|V||E|^2)$
- **Can we solve this problem a different way?**

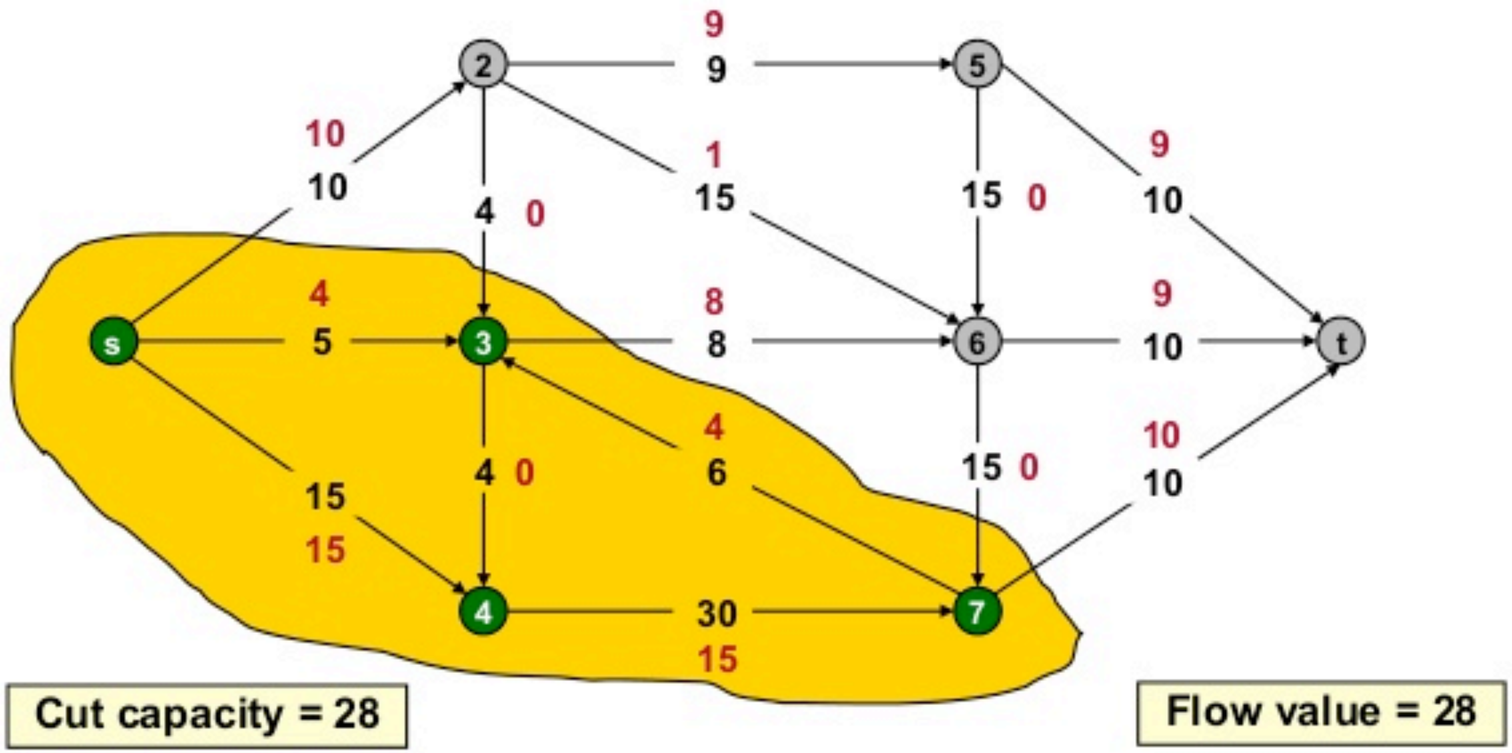
EXAMPLE

- **Oh boy, that got complicated really quickly**
  - $O(|V||E|^2)$
- **Can we solve this problem a different way?**

Max-Flow Min-Cut Theorem

MAX-FLOW MIN-CUT THEOREM (Ford-Fulkerson, 1956): In any network, the value of the max flow is equal to the value of the min cut.

- "Good characterization."
- Proof IOU.



PROBLEM SYMMETRY

- **Solving max-flow is the same as solving the min-cut**
  - What algorithm do we use to solve the min-cut?

FORD-FULKERSON

Algorithm [edit]

Let $G(V,E)$ be a graph, and for each edge from u to v, let $c(u,v)$ be the capacity and $f(u,v)$ be the flow. We want to find the maximum flow from the source s to the sink t. After every step in the algorithm the following is maintained:

Capacity constraints:	$\forall (u,v) \in E\ f(u,v) \le c(u,v)$	The flow along an edge can not exceed its capacity.
Skew symmetry:	$\forall (u,v) \in E\ f(u,v) = -f(v,u)$	The net flow from u to v must be the opposite of the net flow from v to u (see example).
Flow conservation:	$\forall u \in V : u \neq s \text{ and } u \neq t \Rightarrow \sum_{w \in V} f(u,w) = 0$	That is, unless u is s or t. The net flow to a node is zero, except for the source, which "produces" flow, and the sink, which "consumes" flow.
Value(f):	$\sum_{(s,u) \in E} f(s,u) = \sum_{(v,t) \in E} f(v,t)$	That is, the flow leaving from s must be equal to the flow arriving at t.

This means that the flow through the network is a *legal flow* after each round in the algorithm. We define the **residual network** $G_f(V,E_f)$ to be the network with capacity $c_f(u,v) = c(u,v) - f(u,v)$ and no flow. Notice that it can happen that a flow from v to u is allowed in the residual network, though disallowed in the original network: if $f(u,v) > 0$ and $c(v,u) = 0$ then $c_f(v,u) = c(v,u) - f(v,u) = f(u,v) > 0$.

Algorithm Ford–Fulkerson

Inputs Given a Network $G = (V,E)$ with flow capacity c, a source node s, and a sink node t

Output Compute a flow f from s to t of maximum value

1. $f(u,v) \leftarrow 0$ for all edges (u,v)
2. While there is a path p from s to t in G_f, such that $c_f(u,v) > 0$ for all edges $(u,v) \in p$:
    1. Find $c_f(p) = \min\{c_f(u,v) : (u,v) \in p\}$
    2. For each edge $(u,v) \in p$
        1. $f(u,v) \leftarrow f(u,v) + c_f(p)$ (*Send flow along the path*)
        2. $f(v,u) \leftarrow f(v,u) - c_f(p)$ (*The flow might be "returned" later*)

The path in step 2 can be found with for example a breadth-first search or a depth-first search in $G_f(V,E_f)$. If you use the former, the algorithm is called Edmonds–Karp.

FORD-FULKERSON

- **Bleh. Garbage. Who has the time?**

FORD-FULKERSON

- Bleh. Garbage. Who has the time?
- Can we estimate the min-cut?

FORD-FULKERSON

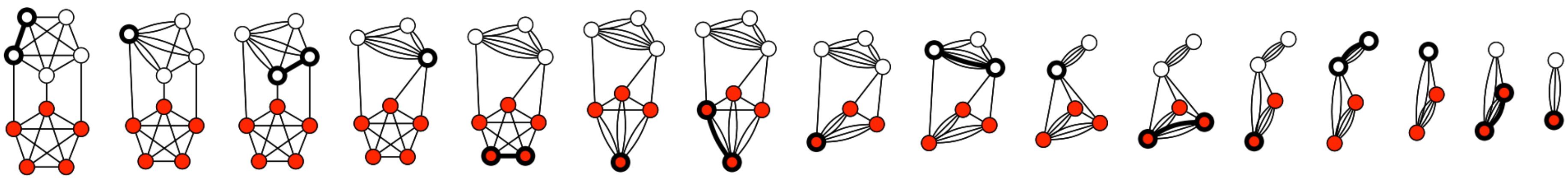
- **Bleh. Garbage. Who has the time?**
- **Can we estimate the min-cut?**
  - What might be an easy estimator?

FORD-FULKERSON

- **Bleh. Garbage. Who has the time?**
- **Can we estimate the min-cut?**
  - What might be an easy estimator?

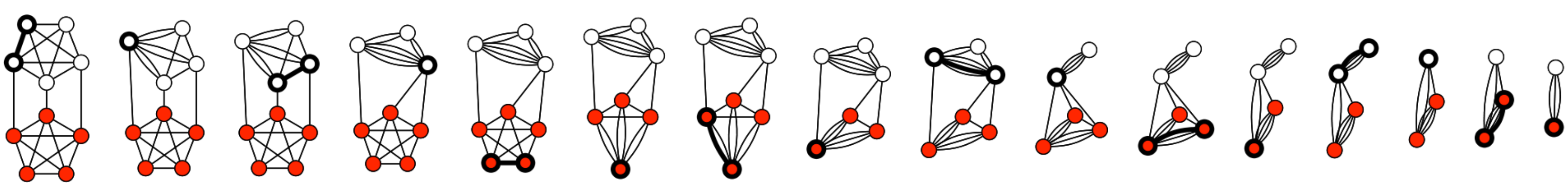
KARGER'S ALGORITHM

- **Bleh. Garbage. Who has the time?**
- **Can we estimate the min-cut?**
  - What might be an easy estimator?
- **Contract edges at random!**
  - How many edges will you contract to get two subgraphs?



KARGER'S ALGORITHM

- **Bleh. Garbage. Who has the time?**
- **Can we estimate the min-cut?**
  - What might be an easy estimator?
- **Contract edges at random!**
  - How many edges will you contract to get two subgraphs?
  - Only |V|-2



KARGER'S ALGORITHM

- **Does this work?**

KARGER'S ALGORITHM

- **Does this work?**
  - Success probability of 2/|E|

KARGER'S ALGORITHM

- **Does this work?**
  - Success probability of 2/|E|
  - Run it O(E) times, and you have a bounded success rate!
  - O(|V||E|)

REDUCTIONS

- **Anytime you can use one algorithm to solve another, this is called a reduction**

REDUCTIONS

- **Anytime you can use one algorithm to solve another, this is called a reduction**
  - Suppose we have an unweighted graph, how might we find the max-cut?

REDUCTIONS

- **Anytime you can use one algorithm to solve another, this is called a reduction**
  - Suppose we have an unweighted graph, how might we find the max-cut?
  - Swap all the edges in the graph and solve the min-cut!

REDUCTIONS

- **Anytime you can use one algorithm to solve another, this is called a reduction**
  - What if we wanted to find the graph of maximum flow that also has minimum weight?

REDUCTIONS

- **Anytime you can use one algorithm to solve another, this is called a reduction**
  - What if we wanted to find the graph of maximum flow that also has minimum weight?
  - This problem is so difficult, no one has found a way to solve it efficiently

TAKE AWAYS

- **We'll talk about approximation and algorithm design more next week**
  - Graph problems can get very difficult very quickly
  - Many problems are related
  - Proving that solving one problem gives a solution to another is called a *reduction*