Lecture 16: Floyd-Fulkerson

Tim LaRock

larock.t@northeastern.edu

bit.ly/cs3000syllabus

Business

HW 3+4 due last night

Homework 5 released this morning, due next Tuesday the 9th at 11:59PM Boston time

Midterm 2 next Wednesday night through Friday night

Question on grades

Last time: Weak MaxFlow-MinCut Duality

- For any s-t flow f and any s-t cut (A, B) $val(f) \leq cap(A, B)$

$$val(f) = \sum_{e \text{ } from \text{ } A \to B} f(e) - \sum_{e \text{ } from \text{ } B \to A} f(e)$$

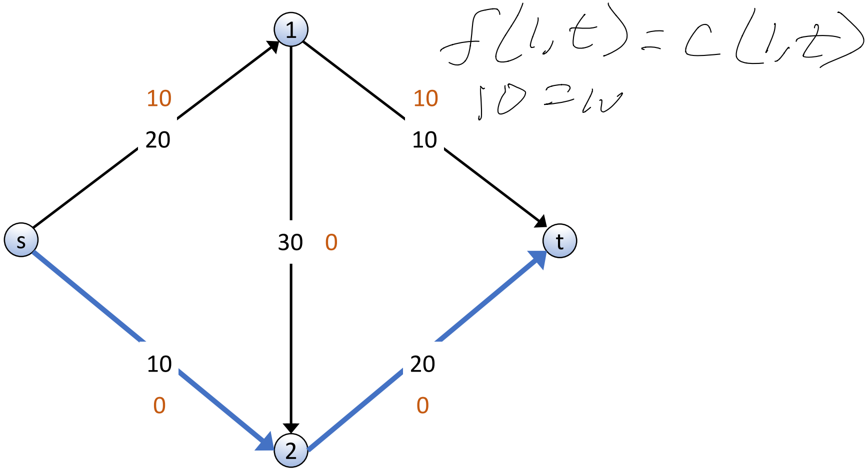
$$\leq \sum_{e \text{ } from \text{ } A \to B} f(e) \qquad \text{(by non-negativity)}$$

$$\leq \sum_{e \text{ } from \text{ } A \to B} c(e) = cap(A, B) \qquad \text{(definition of capacity)}$$

- If f is a flow, (A, B) is a cut, and $val(f) = cap(A, B)$, then f is a max flow and (A, B) is a min cut

Augmenting Paths

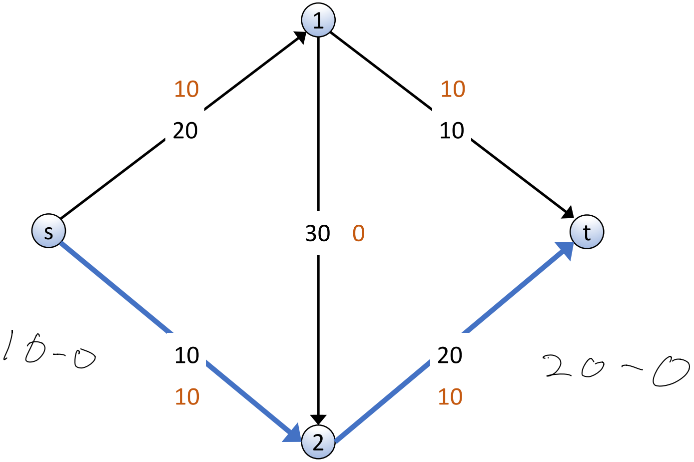
- Given a network $G = (V, E, s, t, \{c(e)\})$ and a flow f, an **augmenting path** P is an $s \to t$ path such that $f(e) < c(e)$ for every edge $e \in P$



Augmenting Paths

- Given a network $G = (V, E, s, t, \{c(e)\})$ and a flow f, an **augmenting path** P is an $s \rightarrow t$ path such that $f(e) < c(e)$ for every edge $e \in P$

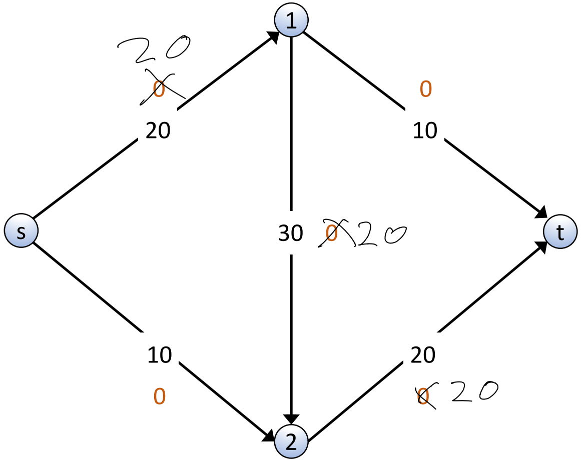
val(f) < val(f')



Adding uniform flow on an augmenting path results in a new valid s-t flow!

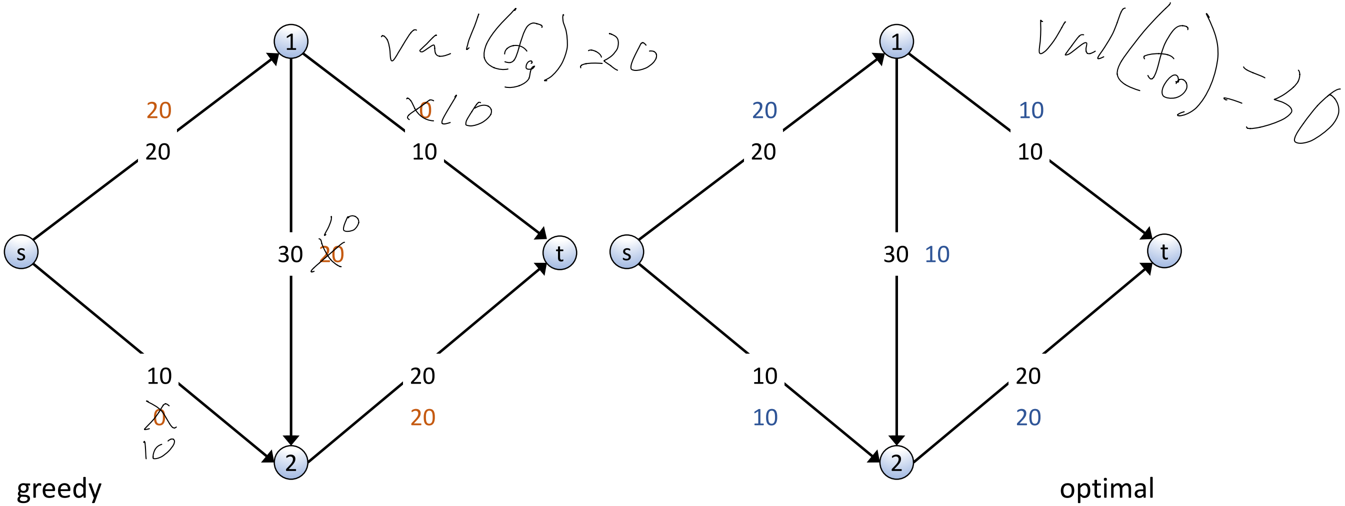
Greedy Max Flow

- Start with $f(e) = 0$ for all edges $e \in E$
- Find an **augmenting path** P
- Repeat until you get stuck



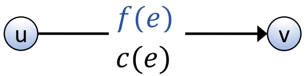
Does Greedy Work?

- Greedy gets stuck before finding a max flow
- How can we get from our solution to the max flow?

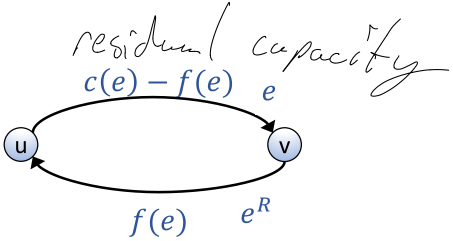


Residual Graphs

- Original edge: $e = (u, v) \in E$.
  - Flow $f(e)$, capacity $c(e)$



- Residual edge
  - Allows "undoing" flow
  - $e = (u, v)$ and $e^R = (v, u)$.
  - Residual capacity



- Residual graph $G_f = (V, E_f)$
  - Edges with positive residual capacity.
  - $E_f = \{e : f(e) < c(e)\} \cup \{e^R : c(e) > 0\}$.

Augmenting Paths in Residual Graphs

- Let G_f be a **residual graph**
- Let P be an augmenting path in the **residual graph**
- **Fact:** $f' = \text{Augment}(G_f, P)$ is a valid flow

```
Augment(G_f, P)
    b ← the minimum capacity of an edge in P
    for e ∈ P
        if e ∈ E:    f(e) ← f(e) + b
        else:        f(e) ← f(e) - b
    return f
```

f is current flow

f' is new augmented flow

Augmenting Paths in Residual Graphs

- Let G_f be a **residual graph**
- Let P be an augmenting path in the **residual graph**
- **Fact:** $f' = \text{Augment}(G_f, P)$ is a valid flow

```
Augment(G_f, P)
    b ← the minimum capacity of an edge in P
    for e ∈ P
        if e ∈ E:    f(e) ← f(e) + b
        else:        f(e) ← f(e) - b
    return f
```

Note: This is the same process as the recurrence in Erickson 10.3!

$$f'(u \to v) = \begin{cases} f(u \to v) + F & \text{if } u \to v \in P \\ f(u \to v) - F & \text{if } v \to u \in P \\ f(u \to v) & \text{otherwise} \end{cases}$$

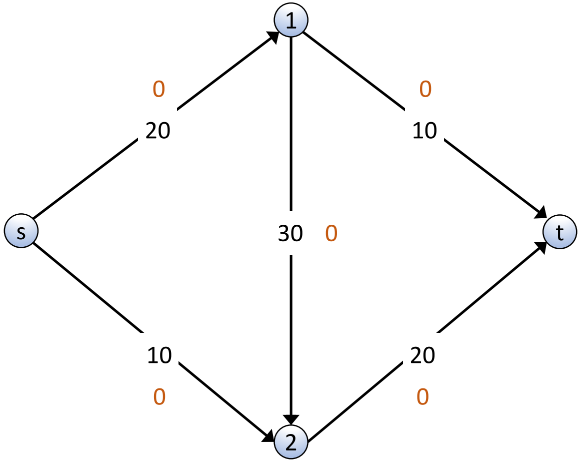
Ford-Fulkerson Algorithm

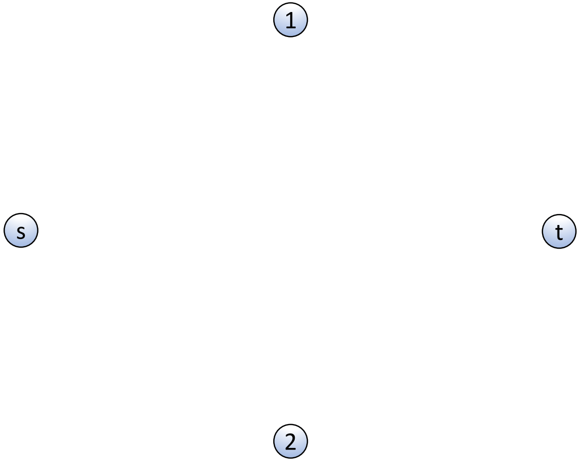
```
FordFulkerson(G,s,t,{c(e)})
    for e ∈ E: f(e) ← 0
    G_f is the residual graph

    while (there is an s-t path P in G_f)
        f ← Augment(G_f,P)
        update G_f

    return f
```

```
Augment(G_f, P)
    b ← the minimum capacity of an edge in P
    for e ∈ P
        if e ∈ E:   f(e) ← f(e) + b
        else:       f(e) ← f(e) - b
    return f
```

Ford-Fulkerson Algorithm

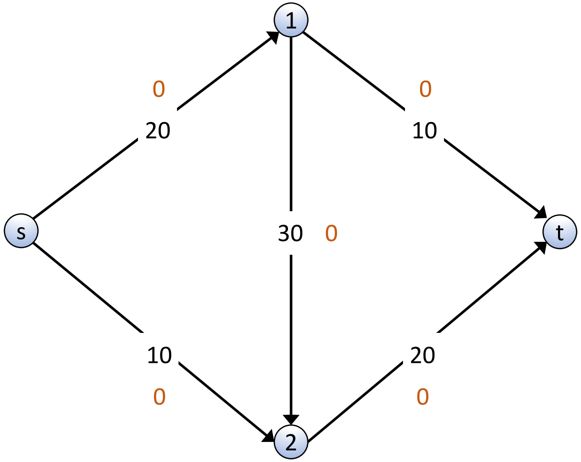
- Start with $f(e) = 0$ for all edges $e \in E$
- Find an **augmenting path** P in the **residual graph**
- Repeat until you get stuck

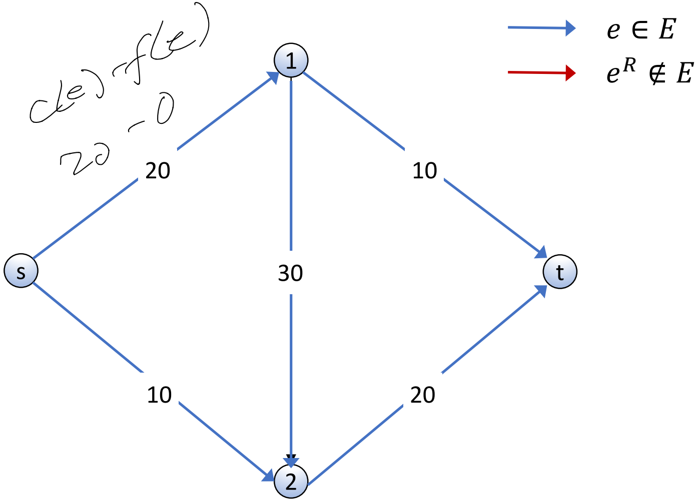




Ford-Fulkerson Algorithm

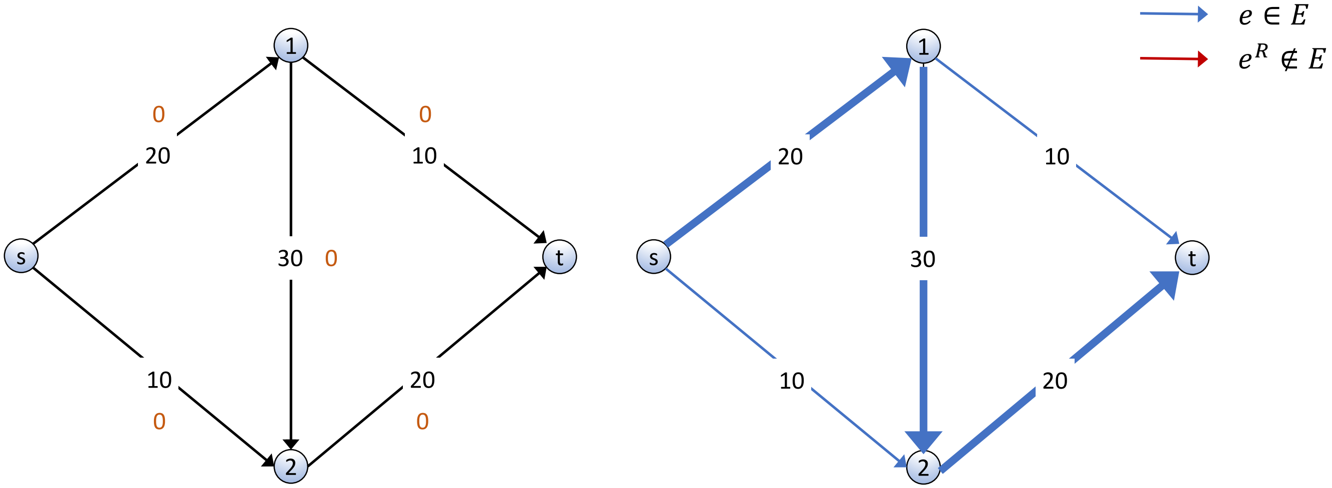
- Start with $f(e) = 0$ for all edges $e \in E$
- Find an **augmenting path** P in the **residual graph**
- Repeat until you get stuck





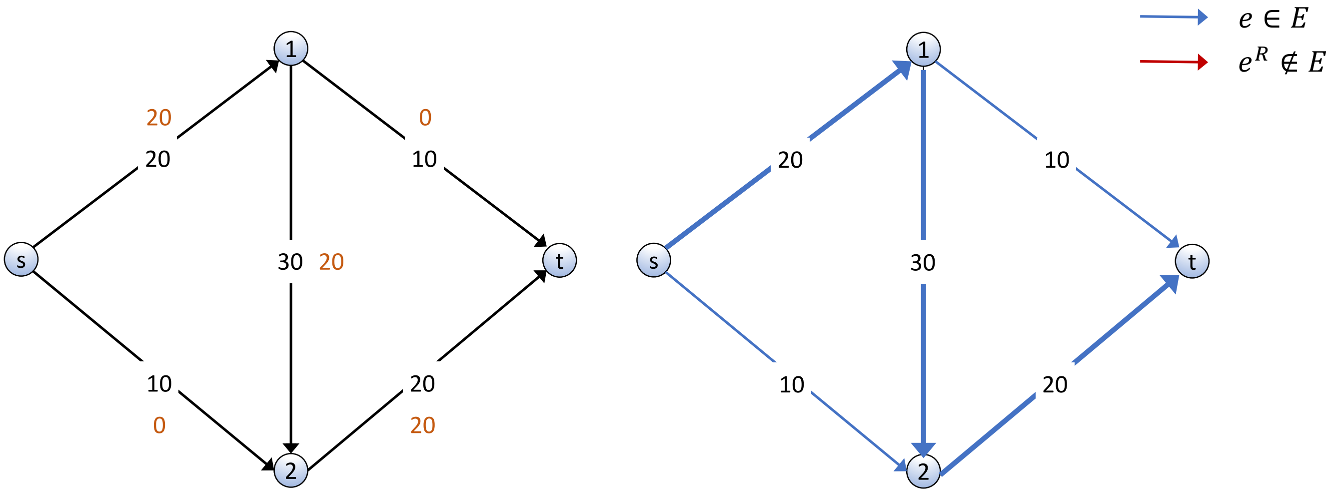
Ford-Fulkerson Algorithm

- Start with $f(e) = 0$ for all edges $e \in E$
- Find an **augmenting path** P in the **residual graph**
- Repeat until you get stuck



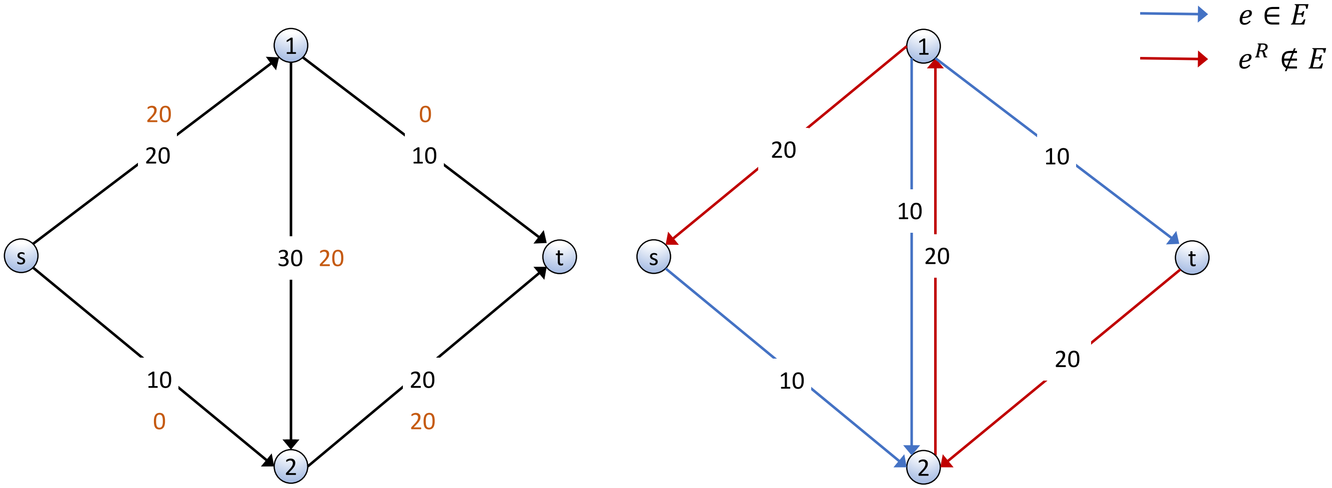
Ford-Fulkerson Algorithm

- Start with $f(e) = 0$ for all edges $e \in E$
- Find an **augmenting path** P in the **residual graph**
- Repeat until you get stuck



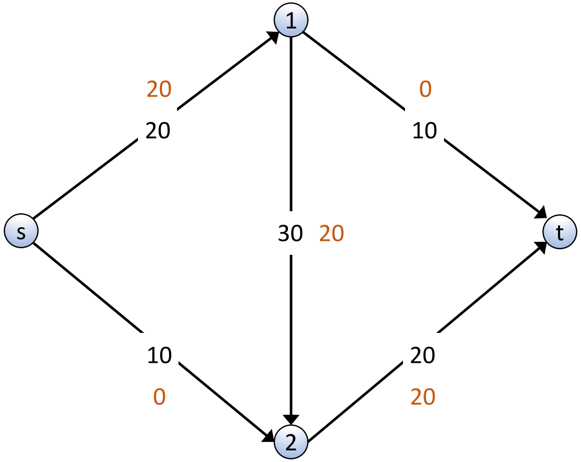
Ford-Fulkerson Algorithm

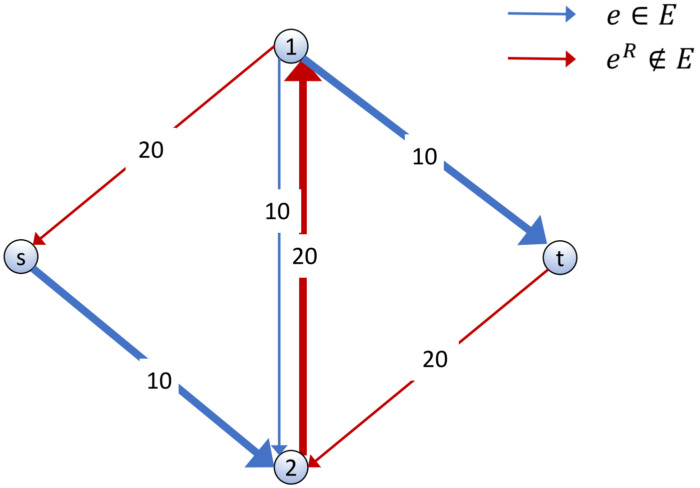
- Start with $f(e) = 0$ for all edges $e \in E$
- Find an **augmenting path** P in the **residual graph**
- Repeat until you get stuck



Ford-Fulkerson Algorithm

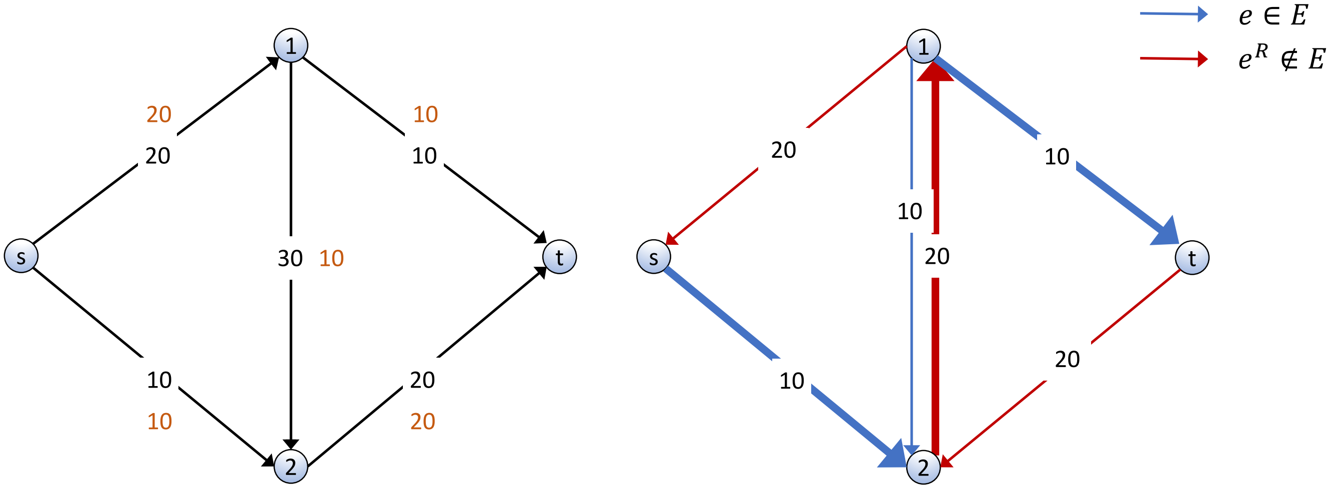
- Start with $f(e) = 0$ for all edges $e \in E$
- Find an **augmenting path** P in the **residual graph**
- Repeat until you get stuck





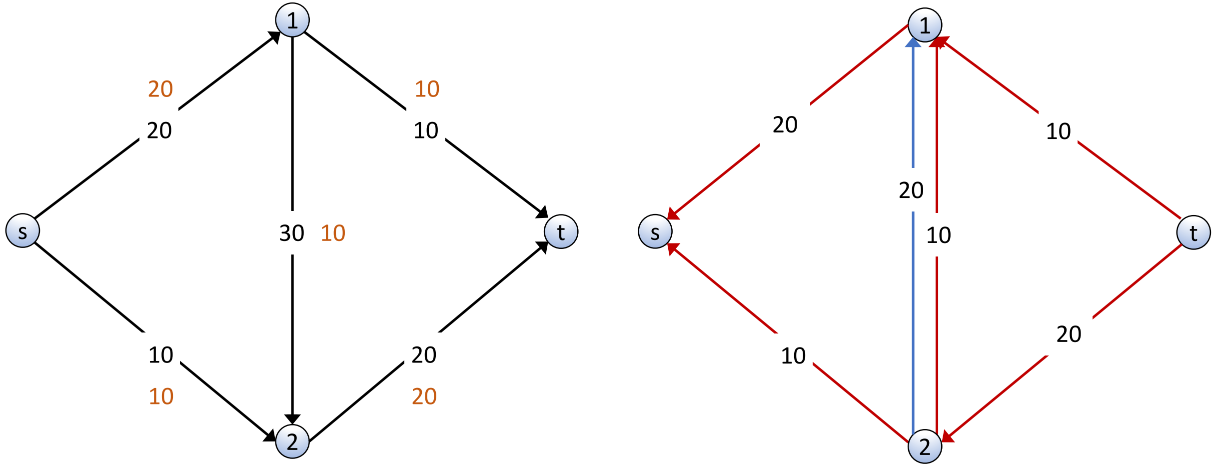
Ford-Fulkerson Algorithm

- Start with $f(e) = 0$ for all edges $e \in E$
- Find an **augmenting path** P in the **residual graph**
- Repeat until you get stuck



Ford-Fulkerson Algorithm

- Start with $f(e) = 0$ for all edges $e \in E$
- Find an **augmenting path** P in the **residual graph**
- Repeat until you get stuck



Running Time of Ford-Fulkerson

- For **integer capacities**, $\leq val(f^*)$ augmentation steps

- Can perform each augmentation step in $O(m)$ time
  - find augmenting path in $O(m)$
  - augment the flow along path in $O(n)$
  - update the residual graph along the path in $O(n)$

- For integer capacities, FF runs in $O\big(m \cdot val(f^*)\big)$ time
  - $O(mn)$ time if all capacities are $c_e = 1$
  - $O(mnC_{\max})$ time for any integer capacities $\leq C_{\max}$

- We can speed FF up by choosing smarter augmenting paths
  - Fattest path: Choose the augmenting path with max capacity
    - Use modified BFS/MST or similar to find max capacity path
    - $\leq m \ln v^*$ augmenting paths
    - $O(m^2 \ln n \ln v^*)$ total running time
  - Shortest augmenting paths (“shortest augmenting path”)
    - $O(m^2 n)$ time

f^* is optimal flow

Correctness of Ford-Fulkerson

- **Theorem:** f is a maximum s-t flow if and only if there is no augmenting s-t path in G_f

- **Strong MaxFlow-MinCut Duality:** The value of the max s-t flow equals the capacity of the min s-t cut

- We'll prove that the following are equivalent for all f
  1. There exists a cut (A, B) such that $val(f) = cap(A, B)$
  2. Flow f is a maximum flow
  3. There is no augmenting path in G_f

Optimality of Ford-Fulkerson

- **Theorem:** the following are equivalent for all f
  1. There exists a cut (A, B) such that $val(f) = cap(A, B)$
  2. Flow f is a maximum flow
  3. There is no augmenting path in G_f

Optimality of Ford-Fulkerson

- **(3 → 1)** If there is no augmenting path in G_f, then there is a cut (A, B) such that $val(f) = cap(A, B)$

f is a max flow

Optimality of Ford-Fulkerson

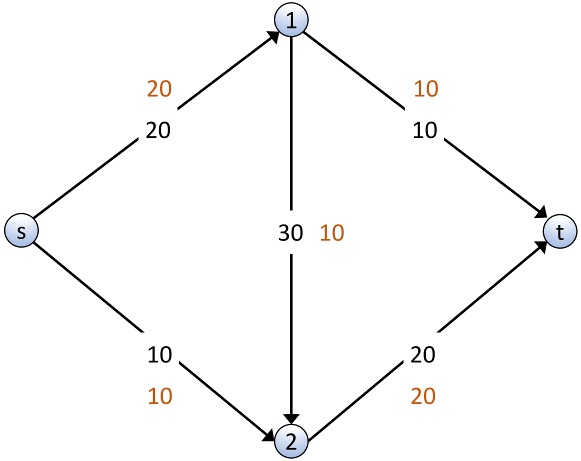
val(f) = 30

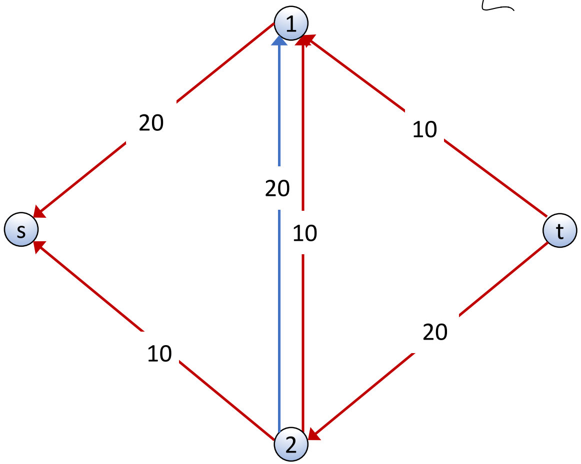
- **(3 → 1)** If there is no augmenting path in G_f, then there is a cut (A, B) such that $val(f) = cap(A, B)$

A = {s}

B = {1, 2, t}

- Sanity check: Is there such a cut in our example?



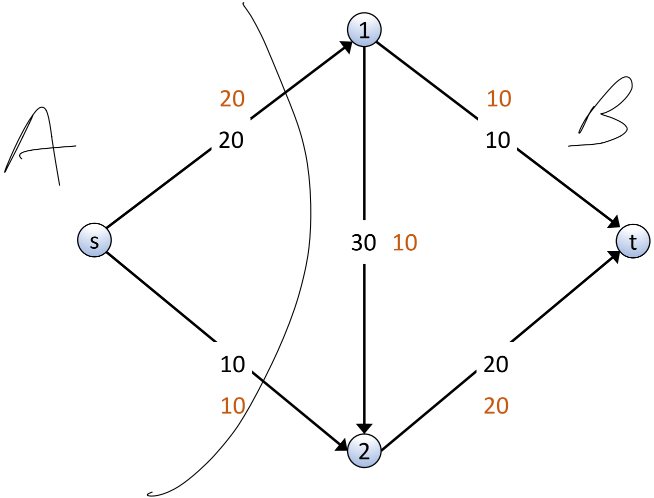


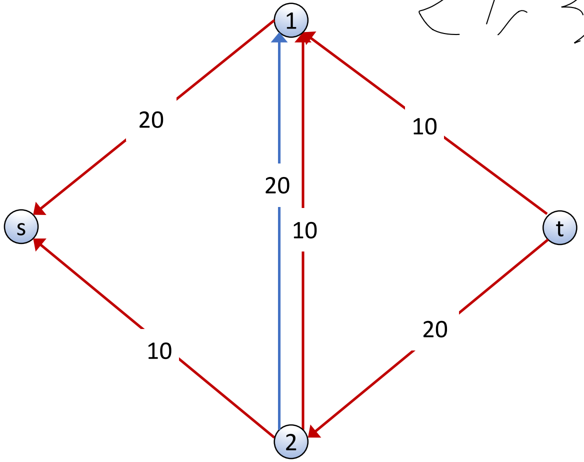
Optimality of Ford-Fulkerson

- **(3 → 1)** If there is no augmenting path in G_f, then there is a cut (A, B) such that $val(f) = cap(A, B)$
  - Let A be the set of nodes reachable from s in G_f
  - Let B be all other nodes
  - **Key observation:** no edges in G_f go from A to B

$A = \{s\}$

$B = \{1, 2, t\}$

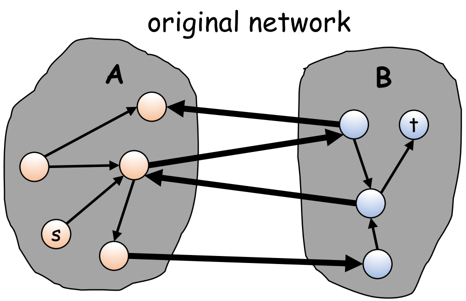




Optimality of Ford-Fulkerson

- **(3 → 1)** If there is no augmenting path in G_f, then there is a cut (A, B) such that $val(f) = cap(A, B)$
  - Let A be the set of nodes reachable from s in G_f
  - Let B be all other nodes
  - **Key observation:** no edges in G_f go from A to B

- If e is $A \to B$, then $f(e) = c(e)$
- If e is $B \to A$, then $f(e) = 0$



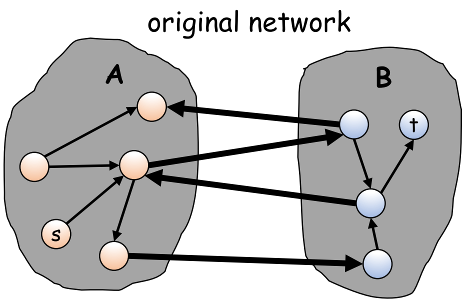
Optimality of Ford-Fulkerson

- **(3 → 1)** If there is no augmenting path in G_f, then there is a cut (A, B) such that $val(f) = cap(A, B)$
  - Let A be the set of nodes reachable from s in G_f
  - Let B be all other nodes
  - **Key observation:** no edges in G_f go from A to B

in the original graph

- If e is $A \to B$, then $f(e) = c(e)$
- If e is $B \to A$, then $f(e) = 0$

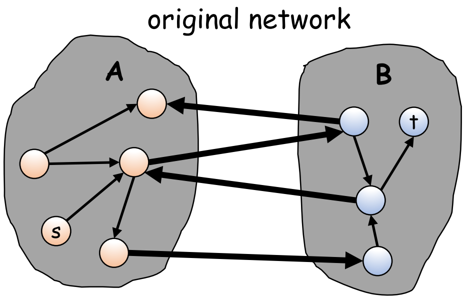
$$val(f) = \sum_{e:A \to B} f(e) - \sum_{e:B \to A} f(e)$$



Optimality of Ford-Fulkerson

- **(3 → 1)** If there is no augmenting path in G_f, then there is a cut (A, B) such that $val(f) = cap(A, B)$
  - Let A be the set of nodes reachable from s in G_f
  - Let B be all other nodes
  - **Key observation:** no edges in G_f go from A to B

- If e is $A \to B$, then $f(e) = c(e)$
- If e is $B \to A$, then $f(e) = 0$



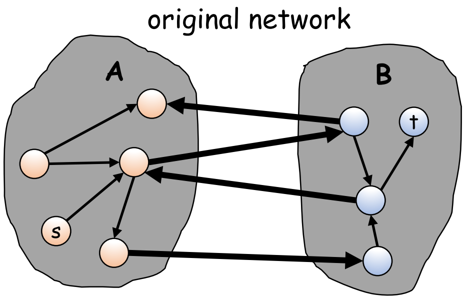
$$val(f) = \sum_{e:A\to B} f(e) - \sum_{e:B\to A} f(e)$$

$$= \sum_{e:A\to B} f(e)$$

Optimality of Ford-Fulkerson

- **(3 → 1)** If there is no augmenting path in G_f, then there is a cut (A, B) such that $val(f) = cap(A, B)$
  - Let A be the set of nodes reachable from s in G_f
  - Let B be all other nodes
  - **Key observation:** no edges in G_f go from A to B

- If e is $A \to B$, then $f(e) = c(e)$
- If e is $B \to A$, then $f(e) = 0$



$$val(f) = \sum_{e:A\to B} f(e) - \sum_{e:B\to A} f(e)$$

$$= \sum_{e:A\to B} f(e)$$

$$= \sum_{e:A\to B} c(e) = cap(A, B)$$

No augmenting path in G_f implies that we have a maximum cut!

Summary

- **The Ford-Fulkerson Algorithm solves maximum s-t flow**
  - Running time $O\big(m \cdot val(f^*)\big)$ in networks with integer capacities
- **Strong MaxFlow-MinCut Duality: max flow = min cut**
  - The value of the maximum s-t flow equals the capacity of the minimum s-t cut
  - If f^* is a maximum s-t flow, then the set of nodes reachable from s in G_{f^*} gives a minimum cut
  - Given a max-flow, can find a min-cut in time $O(n+m)$
- **Every graph with integer capacities has an integer maximum flow**
  - Ford-Fulkerson will return an integer maximum flow

Applications of Network Flow

Applications of Network Flow

- Algorithms for maximum flow can be used to solve:
  - Bipartite Matching
  - Disjoint Paths
  - Survey Design
  - Matrix Rounding
  - Auction Design
  - Fair Division
  - Project Selection
  - Baseball Elimination
  - Airline Scheduling
  - …

Applications of Network Flow

- Algorithms for maximum flow can be used to solve:
  - Bipartite Matching
  - Disjoint Paths
  - Survey Design
  - Matrix Rounding
  - Auction Design
  - Fair Division
  - Project Selection
  - Baseball Elimination
  - Airline Scheduling
  - …

In general: If a problem can be solved in polynomial time, maximum flow can be used to solve it!

Reduction

- **Definition:** a **reduction** is an efficient algorithm that solves problem A using calls to function that solves problem B.

Mechanics of Reductions

- What exactly is a **problem**?
  - A set of legal inputs $\boldsymbol{X}$
    - Ex: An array of numbers $A[1..n]$
  - A set $\boldsymbol{A}(\boldsymbol{x})$ of legal outputs for each $\boldsymbol{x} \in \boldsymbol{X}$
    - Ex: The array A in sorted order

- **Example:** integer maximum flow

  - Input: $G = (V, E, s, t, \{c_e\})$ where c_e is an integer for every $e \in E$

  - Output: A maximum flow $\{f(e)\}$ for G where $f(e)$ is an integer for every $e \in E$ such that $0 \leq f(e) \leq c_e$

Mechanics of Reductions

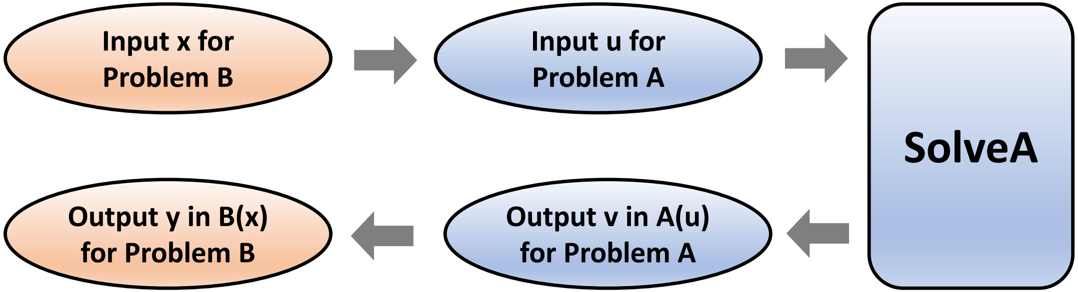
- What exactly is a **problem**?
  - A set of legal inputs $\boldsymbol{X}$
    - Ex: An array of numbers $A[1..n]$
  - A set $\boldsymbol{A}(\boldsymbol{x})$ of legal outputs for each $\boldsymbol{x} \in \boldsymbol{X}$
    - Ex: The array A in sorted order

Sorting

Input: Array of integers A[1...n]

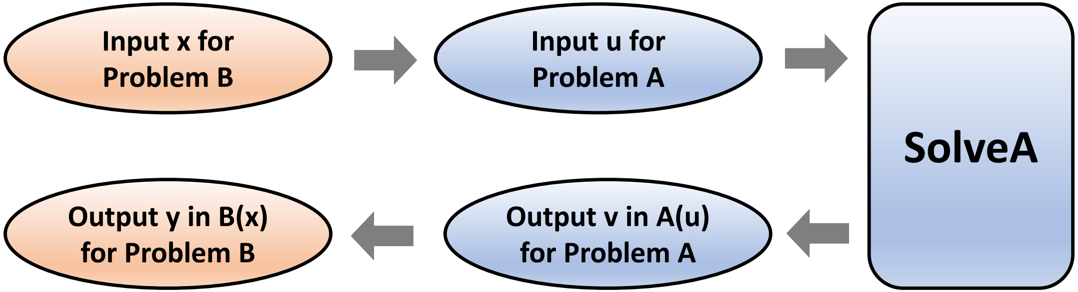
Output: Array A in sorted order

Mechanics of Reductions



In the simplest case, we just call SolveA a single time. In fact we may use SolveA as a subroutine to a more complex reduction.

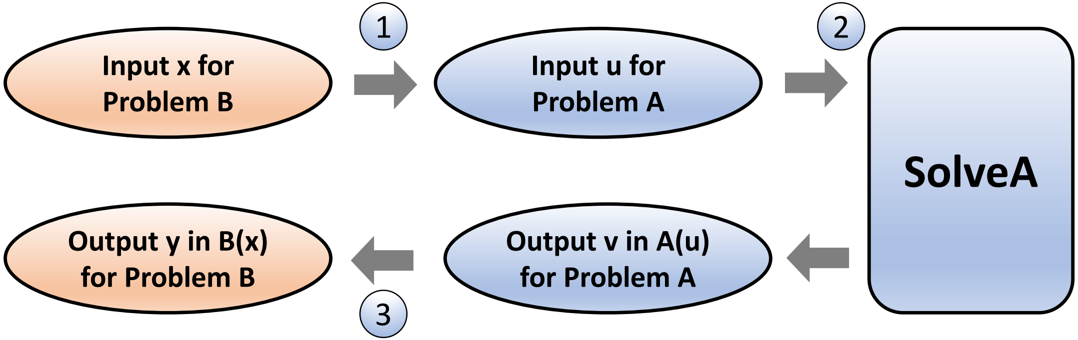
When is a Reduction Correct?



Assume that for valid input u, SolveA returns a valid output v in $A(u)$.

Then for every valid input x, if v is a valid output in $A(u)$, then y is a valid output in $B(x)$.

What is the Running Time?

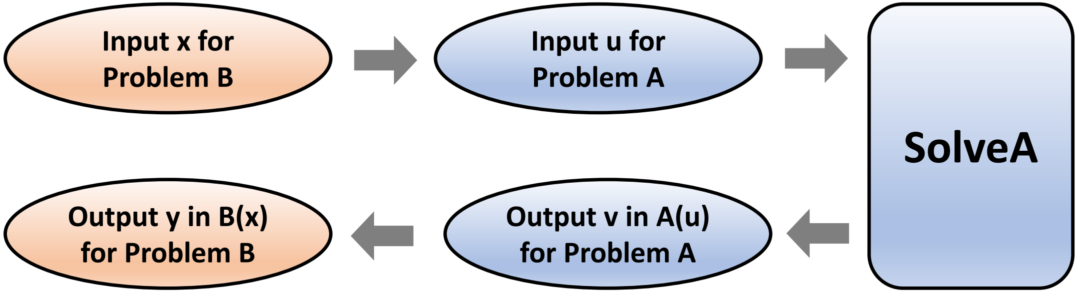


Running time: 

Example: Minimum Cut

A = MaxFlow
B = MinCut

We say "minimum cut from maximum flow"



Input x for B: $G = (V, E, s, t, \{c_e\})$

↓ $O(1)$

Input u for A: $G = (V, E, s, t, \{c_e\})$

↓ $O(m \, val(f^*))$

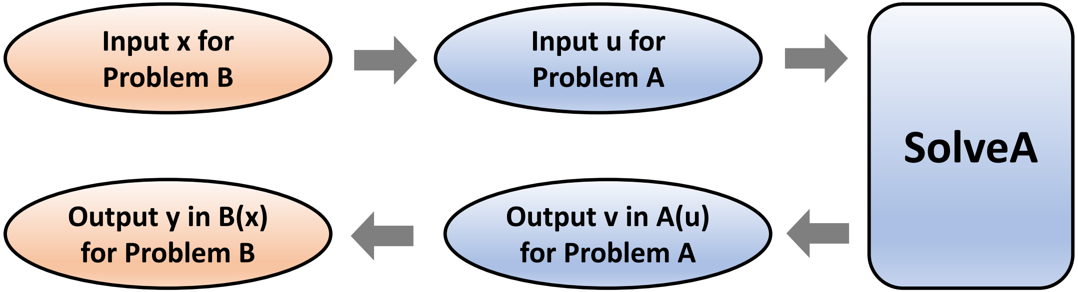
Output $v \in A(u)$: $G = (V, E, s, t, \{c_e\})$

Output $y \in B(x)$: $G = (V, E, s, t, \{c_e\})$

1. Take f, compute the residual graph G_f
2. Find the nodes reachable from s in G_f
3. Output these nodes $O(m+n)$

Example: Median

A = MergeSort
B = Median



Input x for B: Array of length n, $A[1..n]$

↓ $O(1)$

Input u for A: Same array

↓ $O(n \log n)$

Output $v \in A(u)$: Sorted version of $A[1..n]$

↓ $O(1)$

Output y $\in B(x)$: $A[\left\lfloor \frac{n}{2} \right\rfloor]$

Wrap-up

Next time we will see examples of using reductions to solve problems

Suggested Reading:

- Reductions on Wikipedia: https://en.wikipedia.org/wiki/Reduction_(complexity)
- (very optional for now) Erickson Chapter 12
  - He talks about reductions starting in 12.5
  - The first 4 sections will be more relevant for the last week of classes

Work on homework 5!