Intro to Flow Problems

Victor Gao

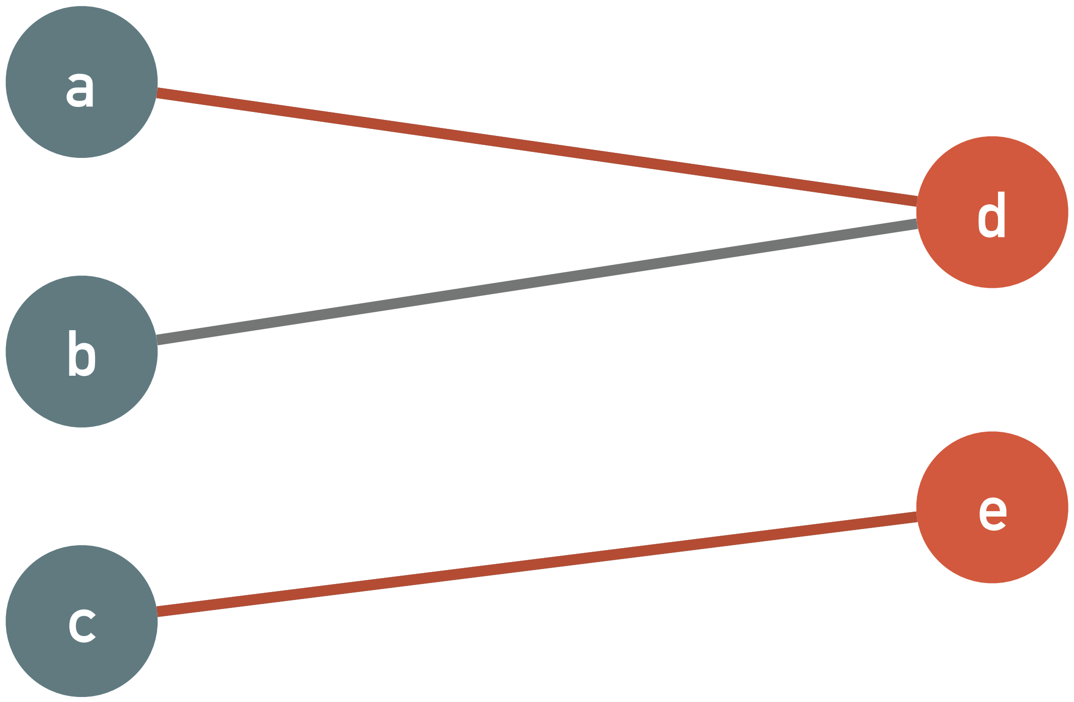
Nov 10, 2017

WHAT YOU SHOULD EXPECT FROM THIS LECTURE

- First of all, know that this is an advanced topic and don’t get frustrated if you can’t understand it immediately.
- Know Bipartite Matching
- Know what a Flow Network is
- Know how to model a problem as a Flow Network
- Understand two basic algorithms: Ford-Fulkerson & Edmonds-Karp
- Know that more advanced algorithms exist

BIPARTITE MATCHING

- Given a bipartite graph, with n nodes on one side and m nodes on the other side
- Find a maximum set of edges such that no two edges in the set share a common node

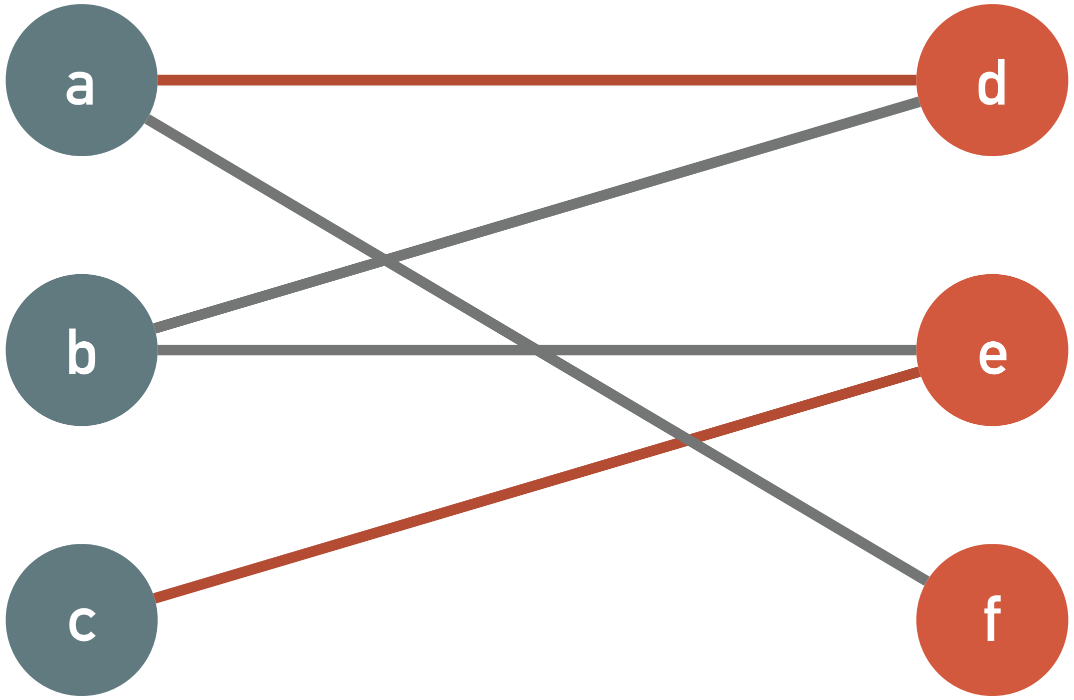


BIPARTITE MATCHING - AUGMENTING PATH

- In order to compute bipartite matching algorithmically, we need to use the idea of "Augmenting Path"
- Let's say we have a set S which contains the edges we have currently selected (initially empty)
- An augmenting path is a path such that
  - The length is odd
  - The 1st, 3rd, 5th, … edges are currently NOT in S
  - The 2nd, 4th, 6th, … edges are currently in S

BIPARTITE MATCHING - AUGMENTING PATH

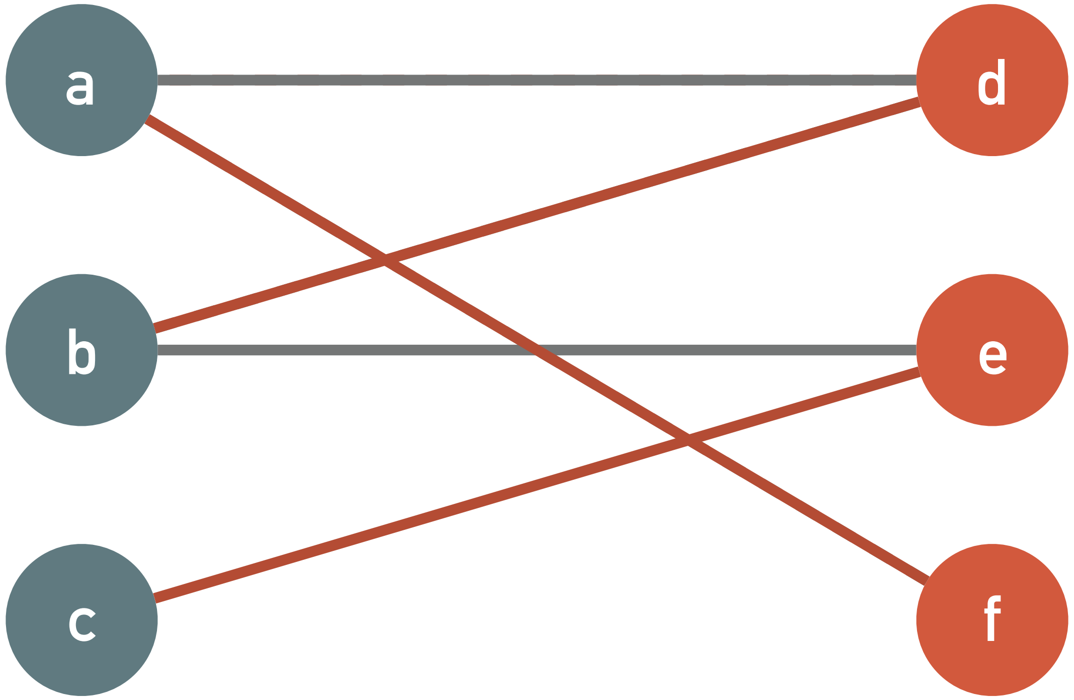
- Here the red edges have been selected



- Can you find an augmenting path?

BIPARTITE MATCHING - AUGMENTING PATH

- If we *invert* the augmenting path we just found



- We increase the size of our matching by 1

BIPARTITE MATCHING – HUNGARIAN ALGORITHM

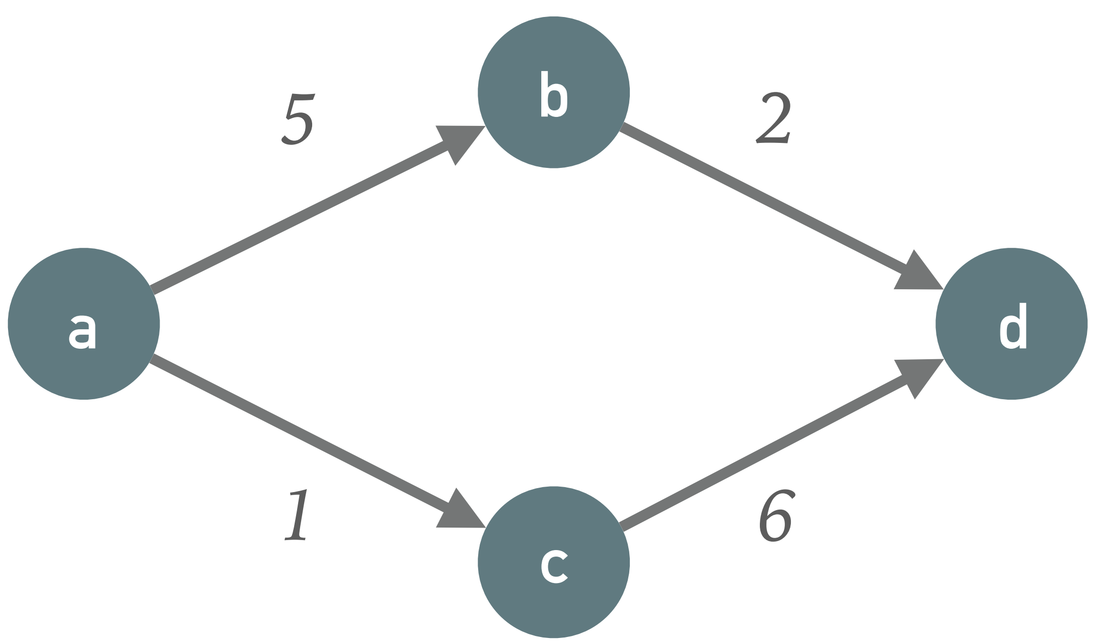
- Berge's Lemma: a matching in a graph is maximum if and only there is no augmenting path
- In other words, we can just keep finding augmenting paths (using bfs/dfs) and inverting them, and eventually we will get a maximum matching
- This idea is formalized as the Hungarian algorithm

```
for each node v on the left:
    find an augmenting path starting with v
    if found:
        invert the path

    // if not found, do not terminate the process
    // move on to the next node
    // this is necessary since the graph may be disconnected
```

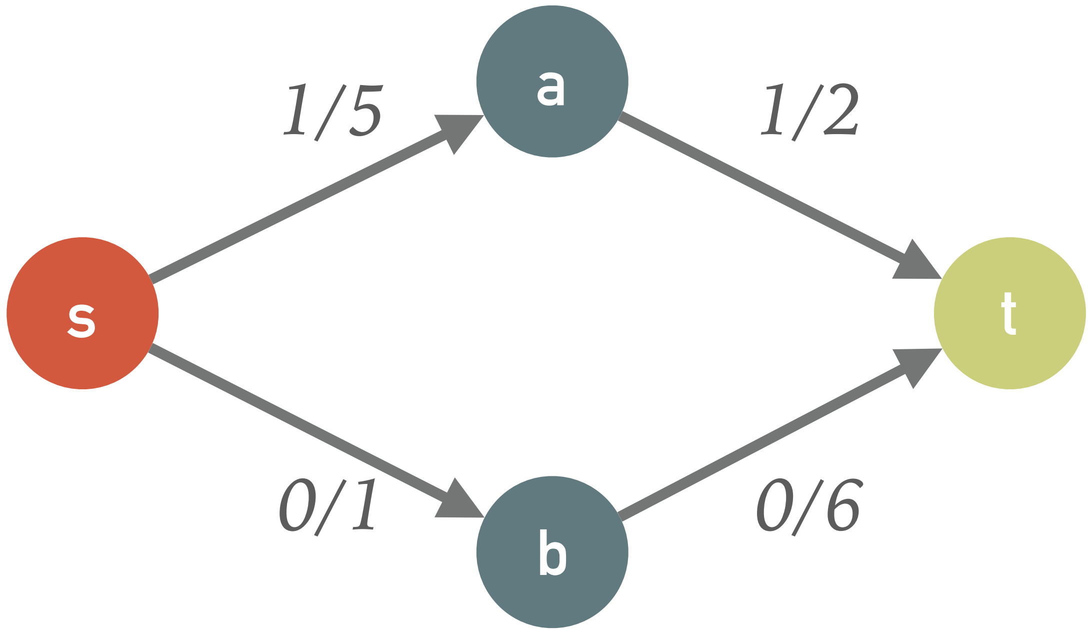
FLOW NETWORK - NON-RIGOROUS DEFINITION

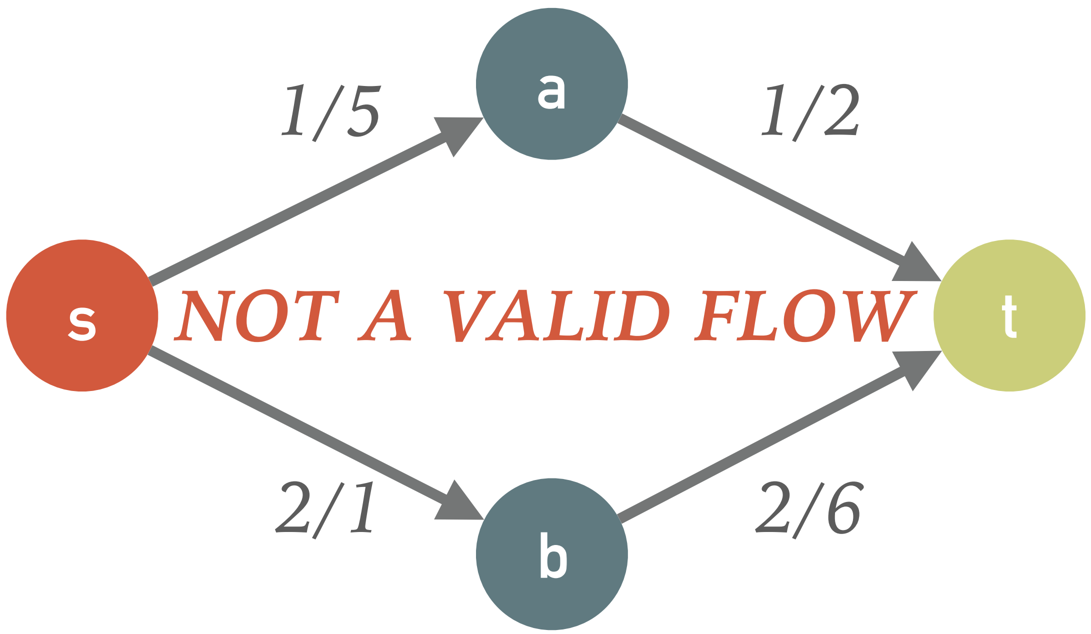
- A **Flow Network** is essentially a directed graph with a **capacity** assigned to each edge
- Think of it as the roads in a city, or a system of interconnected water pipes



FLOW NETWORK - NON-RIGOROUS DEFINITION

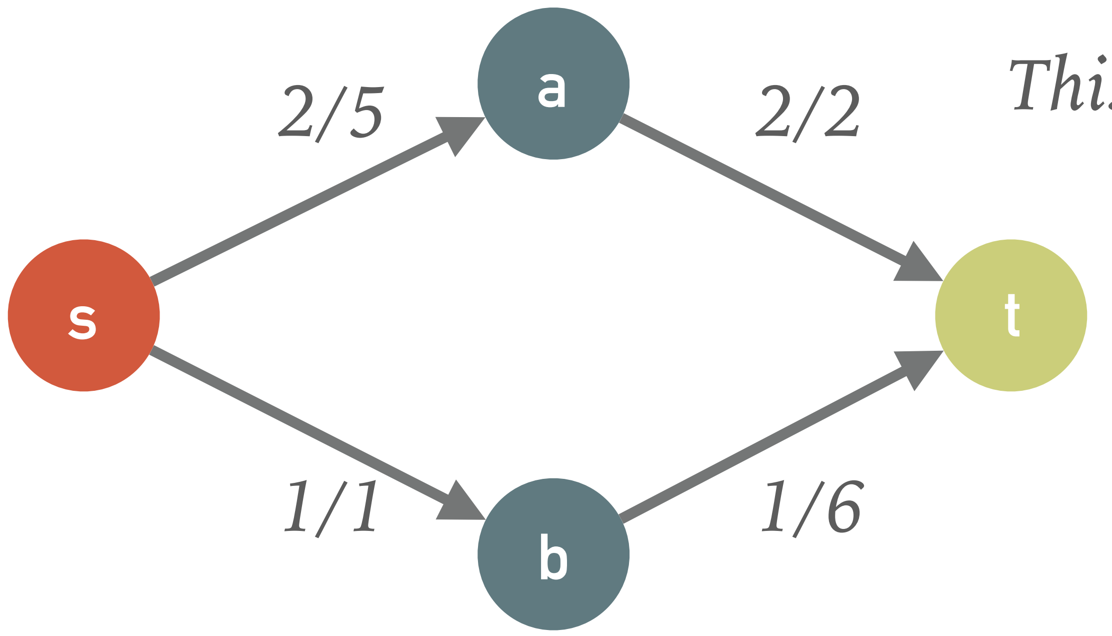
- A (feasible) **Flow** in a network, is a way to assign a non-negative **value** to each edge such that
  - The **value** is less then or equal to the **capacity** for each edge
  - Except two nodes that are specified as **source** and **sink**, the amount of flow going to and leaving each node should be equal (in other words, the net flow should be 0)





FLOW NETWORK – MAXIMUM FLOW

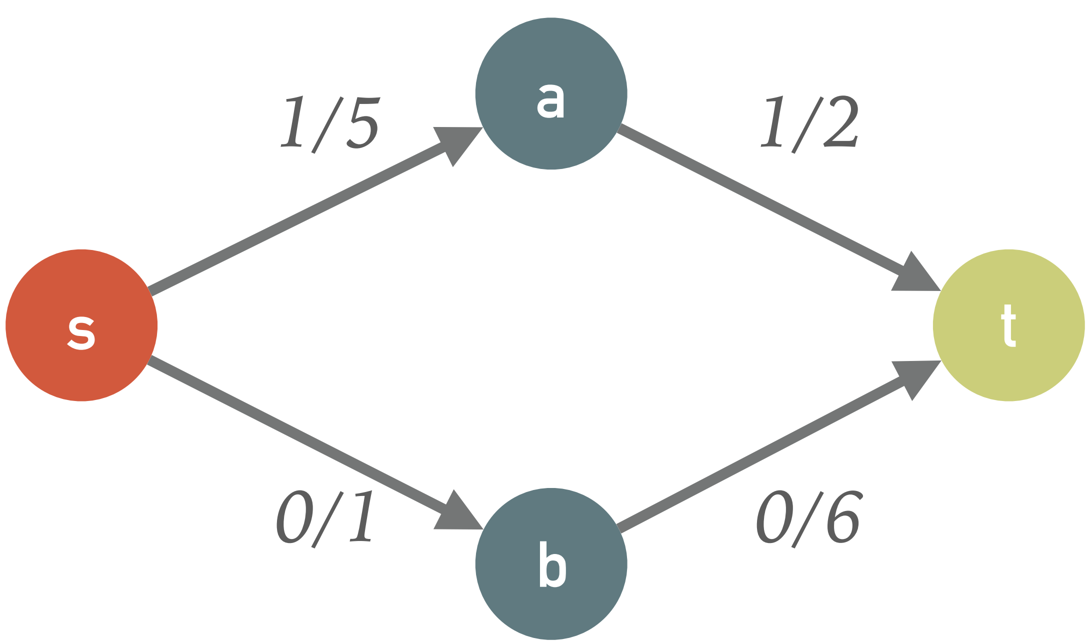
- The maximum flow problem asks for the maximum possible amount of **flow** going from the **source** to the **sink** in a given **flow network**
- A real example: in average, how many cars can go from one location to another per minute, knowing the capacity of each road in the city? (bandwidth)



This is a maximum flow

FLOW NETWORK – AUGMENTING PATH

- In order to compute the maximum flow, again we need to use the idea of **augmenting path** (modified) and **residual network**
- Definition: an **augmenting path** is an s-t path with positive remaining capacity on each edge



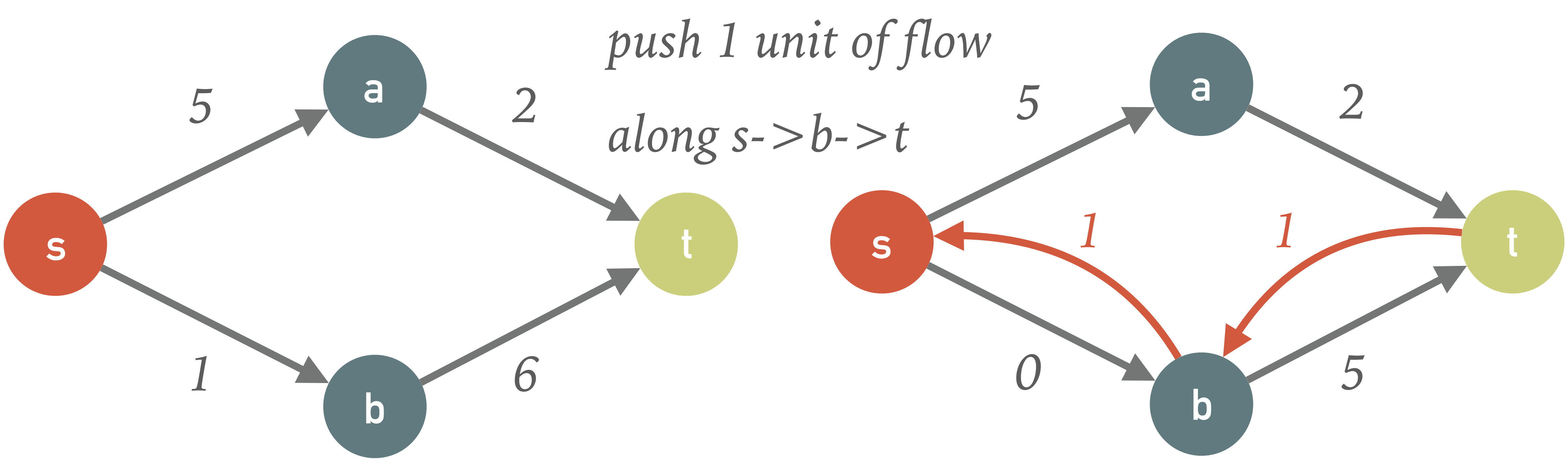
Can you find the augmenting paths?

s -> a -> t

s -> b -> t

FLOW NETWORK – RESIDUAL NETWORK

- Let's say we have found an augmenting path in a network G, and we push f amount of flow along the path
- The **residual network** is the network obtained by
  - Decreasing the capacity of each edge along the path by f
  - Increasing the capacity of each backward edge along the path by f, allowing the flow to come back

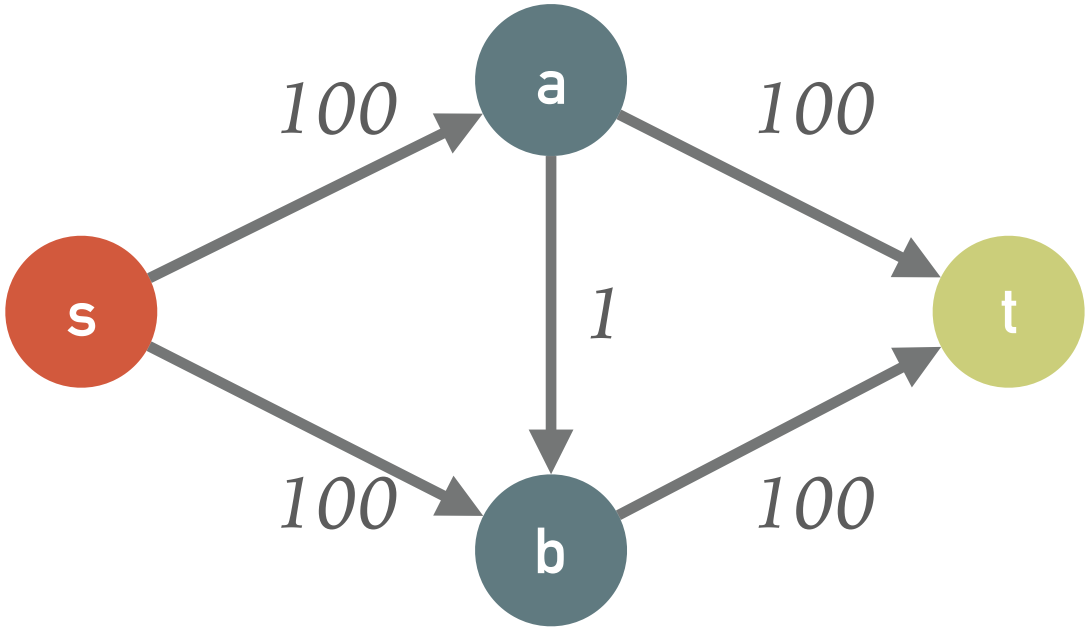


FLOW NETWORK - FORD-FULKERSON

- A flow we obtained is maximum if and only if we can not find any augmenting path in the network
- Similar to bipartite matching
  - Find an arbitrary augmenting path if there is one
  - Push as much flow as you can along the path
  - Repeat the process in the **residual network**
- This is called the **Ford-Fulkerson** Algorithm

FLOW NETWORK - FORD-FULKERSON

- Fork-Fulkerson has a time complexity of O(Ef), where f is the value of the maximum flow
- It is very inefficient in a case like this:

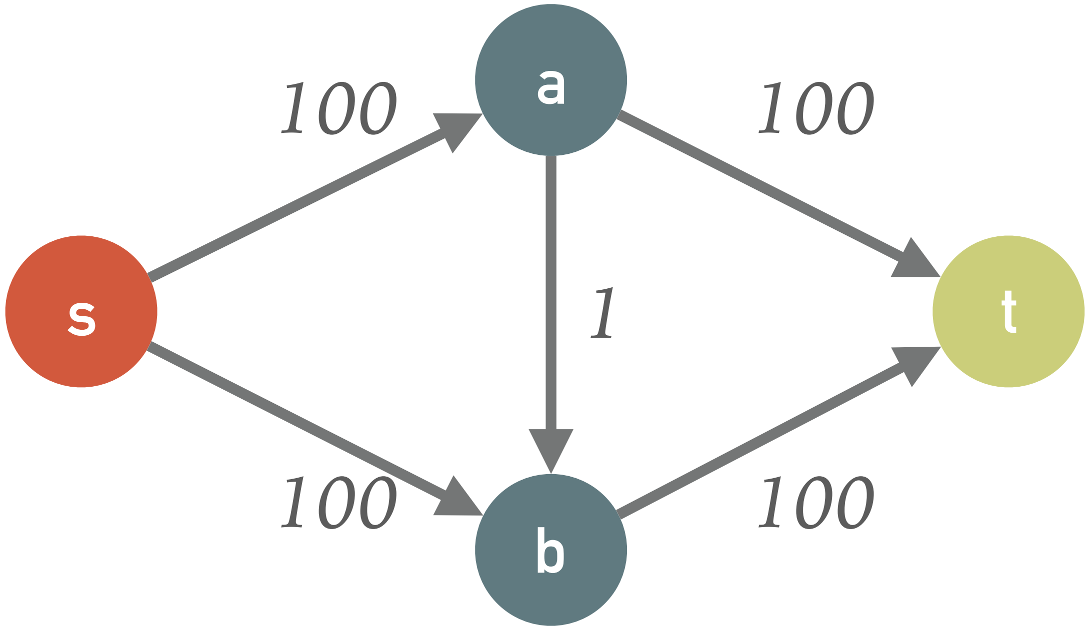


try to push flow along s->a->b->t

then try to push flow along s->b->a->t

FLOW NETWORK - EDMONDS-KARP

- **Edmonds-Karp** is essentially the same as Ford-Fulkerson except that instead of choose an arbitrary augmenting path at a time, it always picks the shortest path
- Note "shortest" means the least number of edges



In this case, Edmonds-Karp will pick

s->a->t or s->b->t

- Time Complexity: O(VE ^ 2) (no longer depends on f!)

FLOW NETWORK – ADVANCED ALGORITHMS

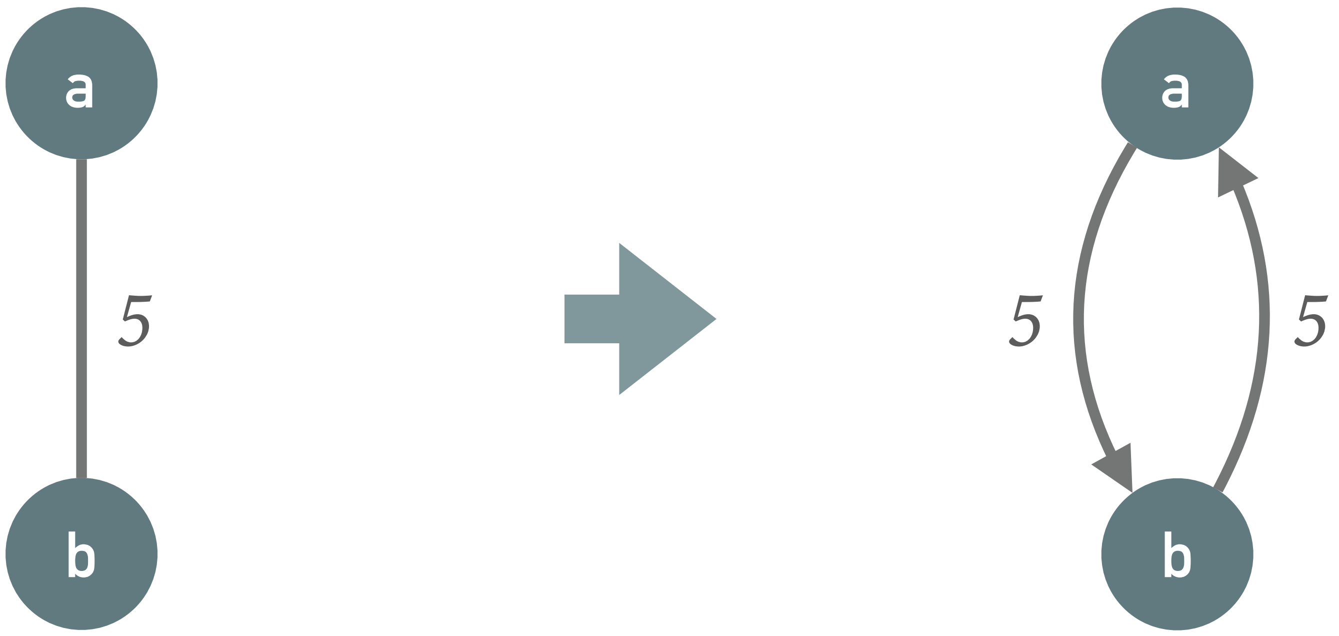
- There are some faster (and more complicated) algorithms to compute the maximum flow
- Here is a list from Prof. Jeff Erickson's notes:

Technique	Direct	With dynamic trees	Sources
Blocking flow	$O(V^2E)$	$O(VE\log V)$	[Dinits; 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)	$O(V^3)$	$O(V^2\log(V^2/E))$	[Goldberg and Tarjan]
Push-relabel (highest label)	$O(V^2\sqrt{E})$	—	[Cheriyan and Maheshwari; Tunçel]
Pseudoflow	$O(V^2E)$	$O(VE\log V)$	[Hochbaum]
Compact abundance graphs		$O(VE)$	[Orlin 2012]

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

FLOW NETWORK – A NOTE ON UNDIRECTED GRAPHS

- If you are given a undirected graph, convert it into a directed one by replacing each undirected edge with two directed edges



FLOW NETWORK - MAX-FLOW MIN-CUT THEOREM (OPTIONAL)

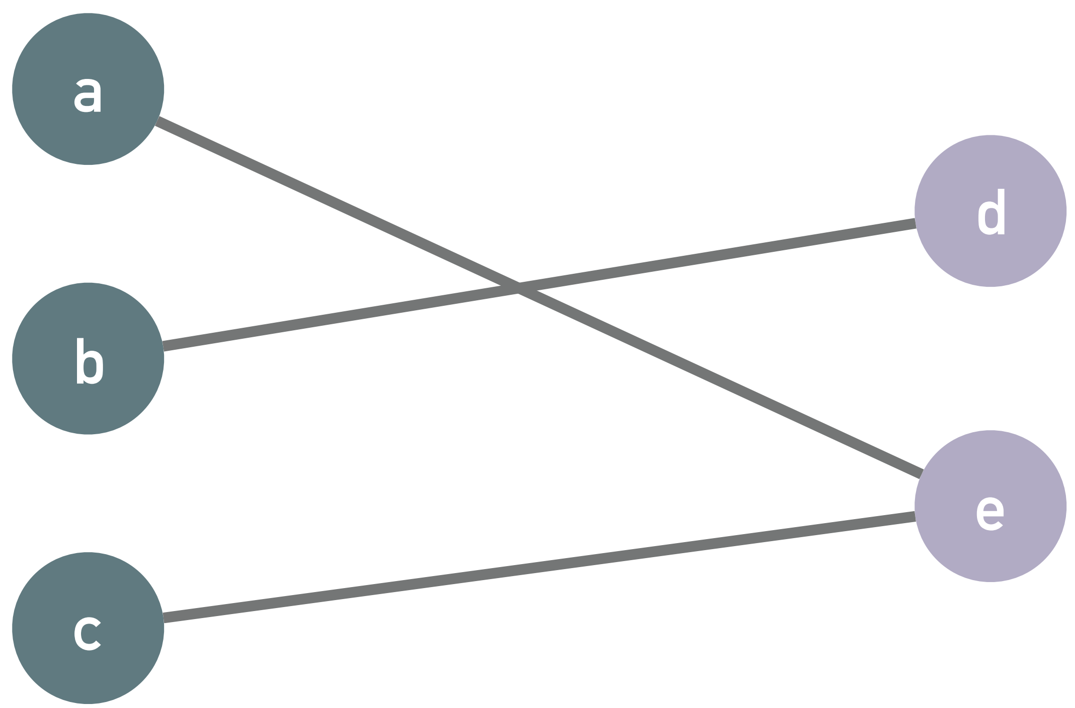
- What is a **Cut**?
  - A cut is a partition of the nodes of a graph into two disjoint subsets S, T
  - A minimum cut is a cut such that the sum of the weights of the edges going from S to T is minimized
- **Max-Flow Min-Cut Theorem**:
  - The maximum amount of flow passing from the source to the sink is equal to the total weight of the edges in the minimum cut.

FLOW NETWORK – A NOTE ON MIN COST MAX FLOW

- A variation of the maximum flow problem is the minimum cost maximum flow problem
- Each edge not only has a capacity, but also a unit cost for each unit of flow to pass by
- You need to not only find a maximum flow but also one with minimum cost
- Basic idea:
  - Use Bellman-Ford instead of BFS
  - When reversing an edge, assign (-cost) to the backward edge

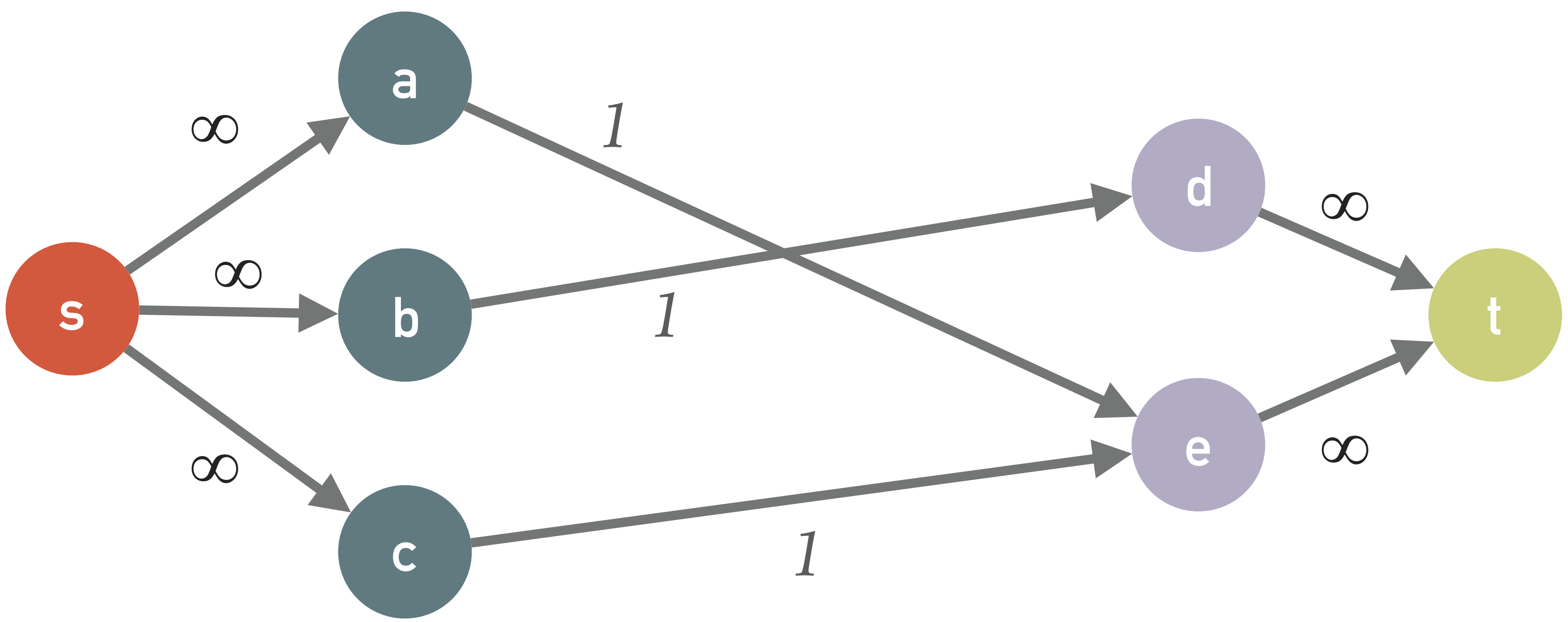
REDUCING BIPARTITE MATCHING TO MAXIMUM FLOW

- Bipartite matching problem can be reduced to maximum flow problem by adding a super source and a super sink



REDUCING BIPARTITE MATCHING TO MAXIMUM FLOW

- Each edge in the original bipartite graph is replaced by a directed one with capacity 1
- Add edges with infinite capacity from the source and to the sink



EXAMPLE PROBLEM – DRAINAGE DITCHES

http://poj.org/problem?id=1273

EXAMPLE PROBLEM - MY T-SHIRT SUITS ME

search “uva 11045” using your search engine

QUESTIONS?