Announcements

Announcements

- Sign up for exam time if you haven’t yet done so!
- Exam questions have been posted- please check

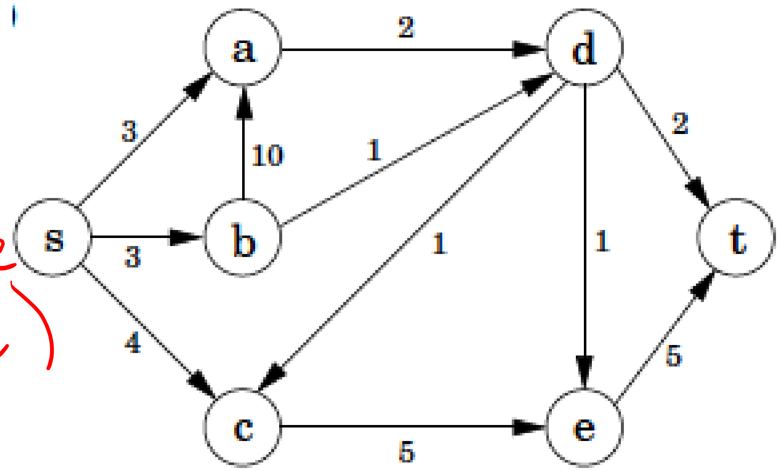
Network Flows

Network Flows

$a - b = a \rightleftarrows b$

- You are given a network with capacities on the edges

source (lake)



sink (city)

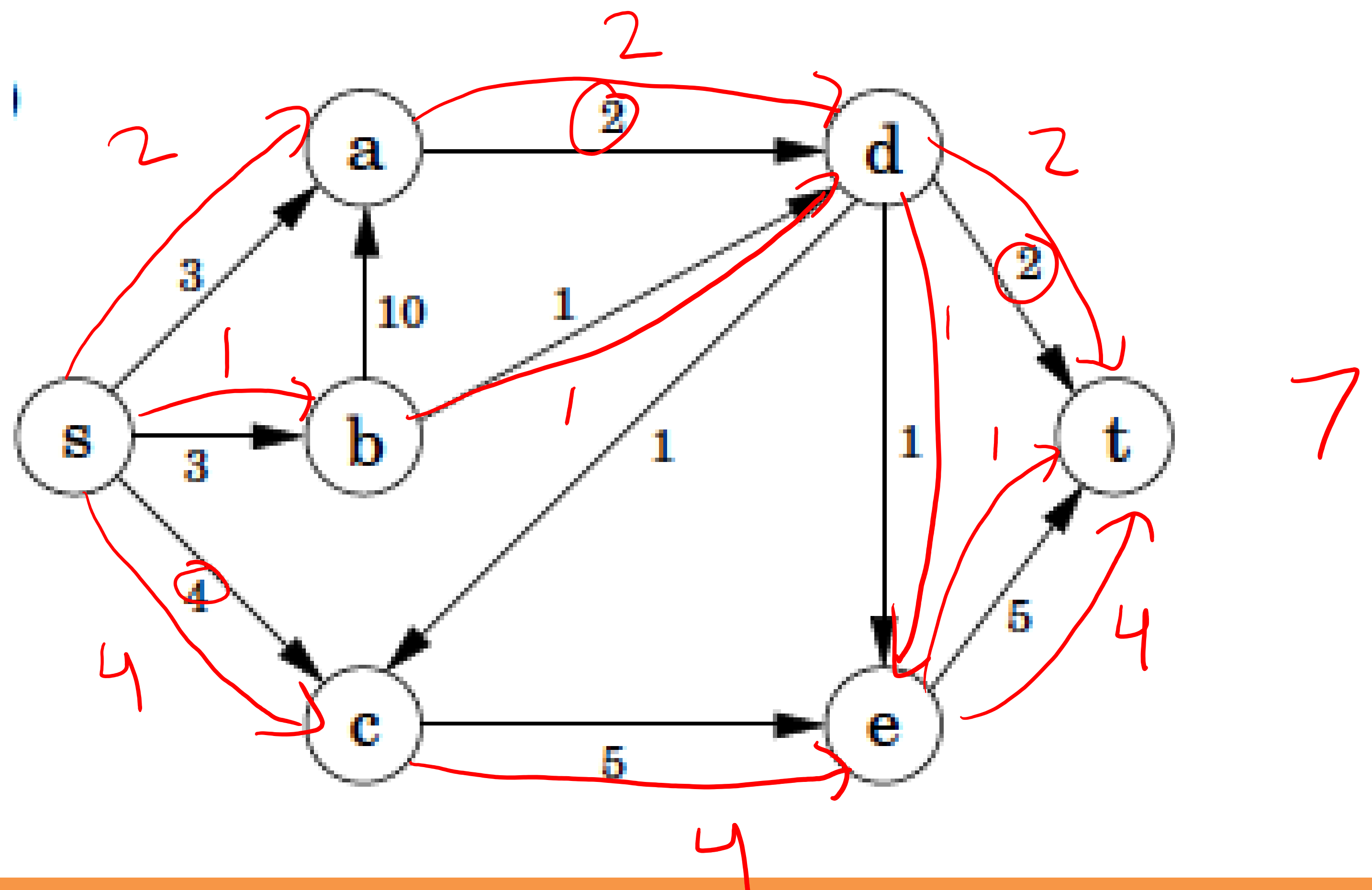
- Goal is to decide how much flow to send along each edge, to maximize total amount from s to t

Rules of Network Flows

- The flow sent along an edge cannot exceed the total capacity of that edge (respect direction of edge)
- The total amount of flow entering a node must equal the amount of flow leaving a node (except source, sink)
- This is a linear programming problem!

In-Class Exercise

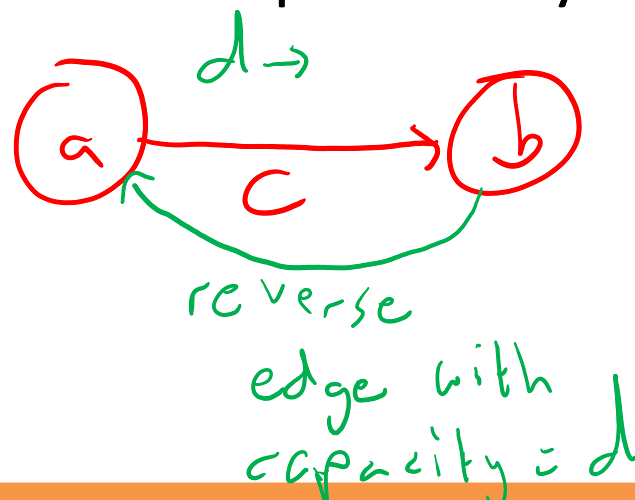
Find the maximum amount of flow from s to t



The Ford-Fulkerson Network Flow Algorithm

(Bellman-Ford)

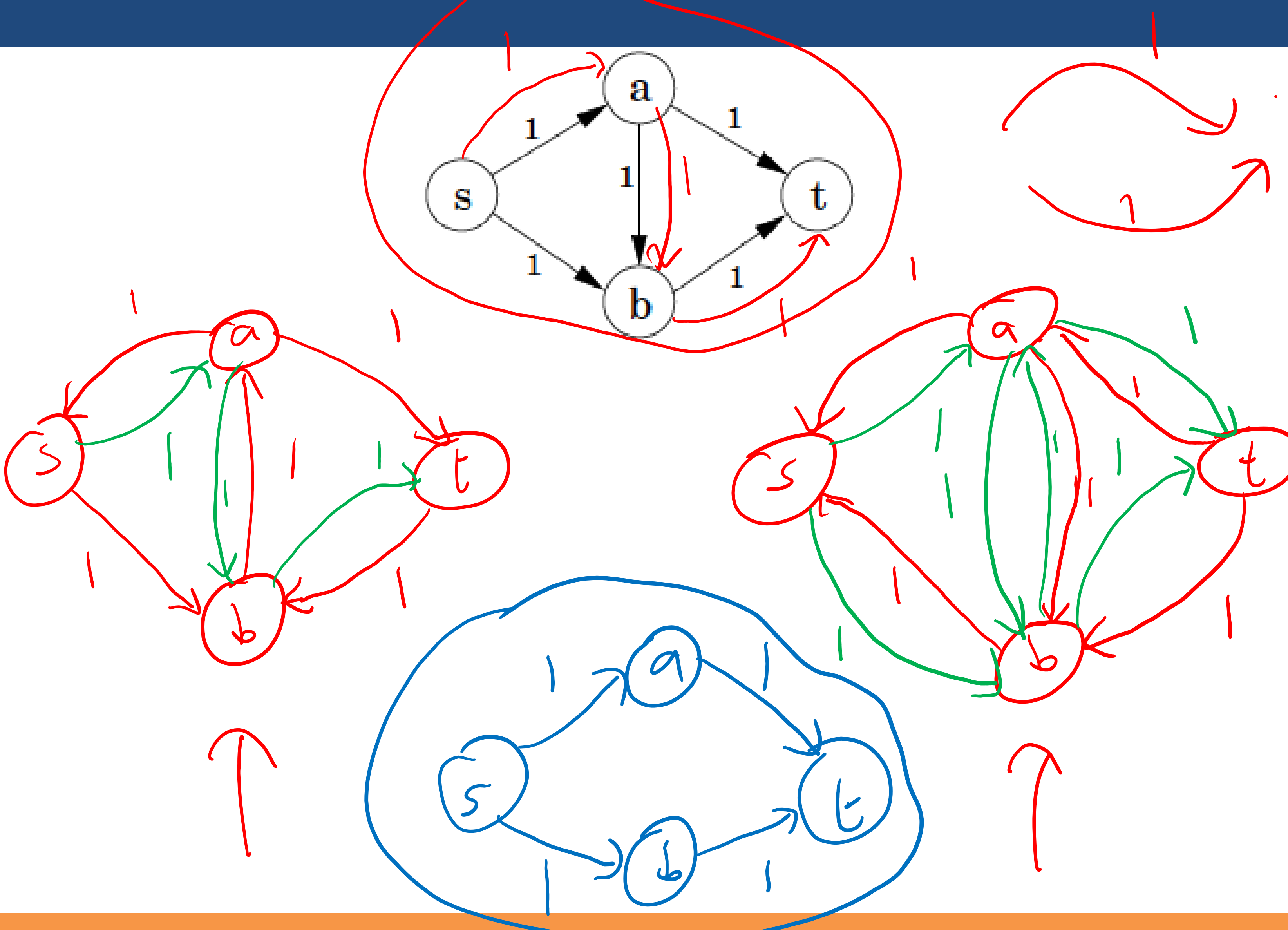
- Choose a path from S to T (if no path exists, terminate)
- Send as much flow as possible along that path (minimum remaining capacity of any edge along that path)
- Rules of path selection:
  - You can use a *forward* edge if there is capacity left (pre-existing)
  - You can *reverse* flow previously sent along an edge



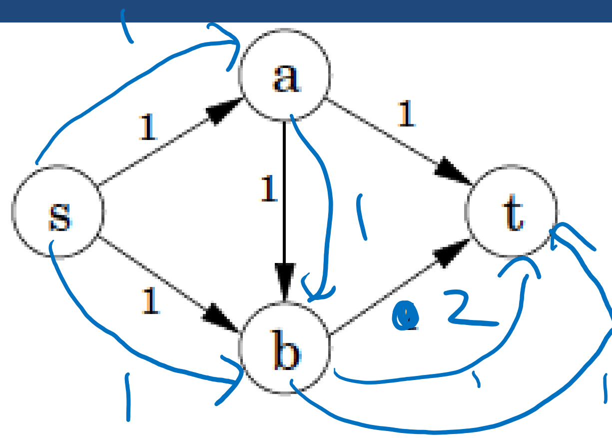
$d \leq c$

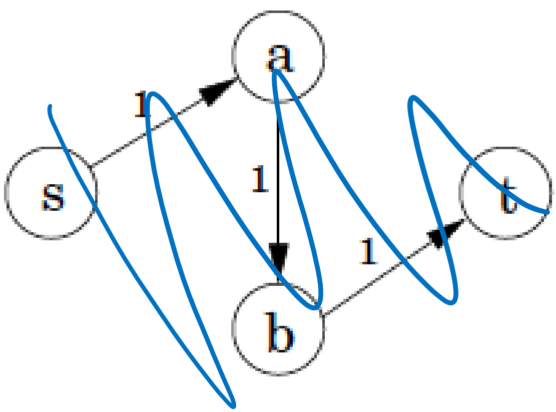
an

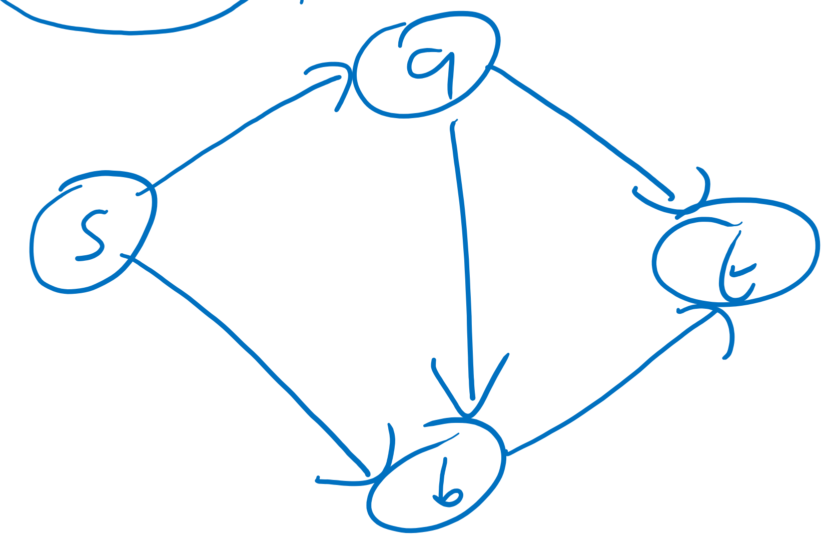
Sketch of Network Flow Algorithm



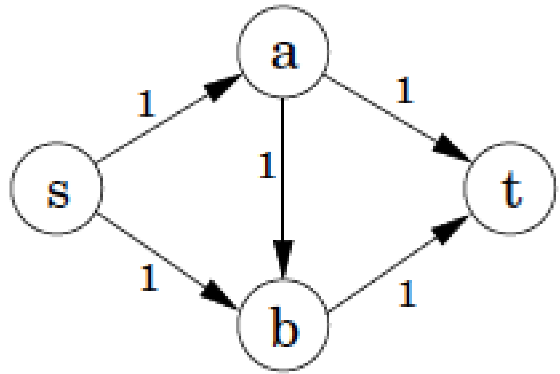
Sketch of Network Flow Algorithm

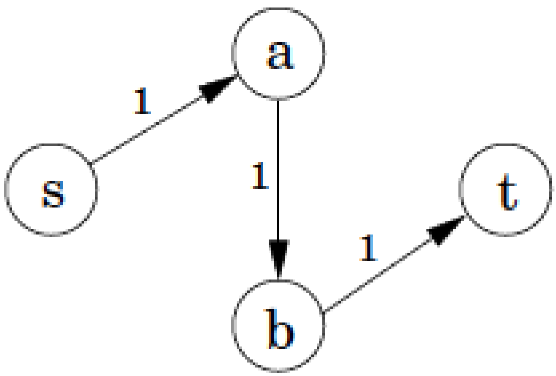


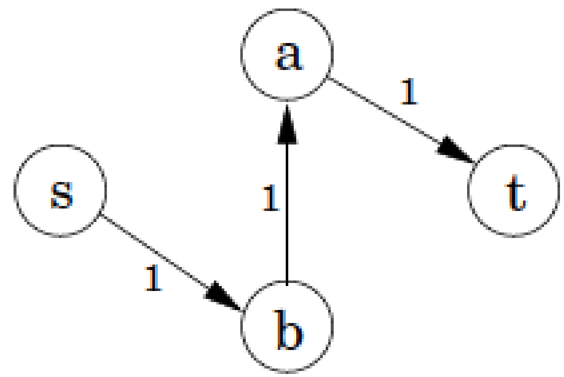




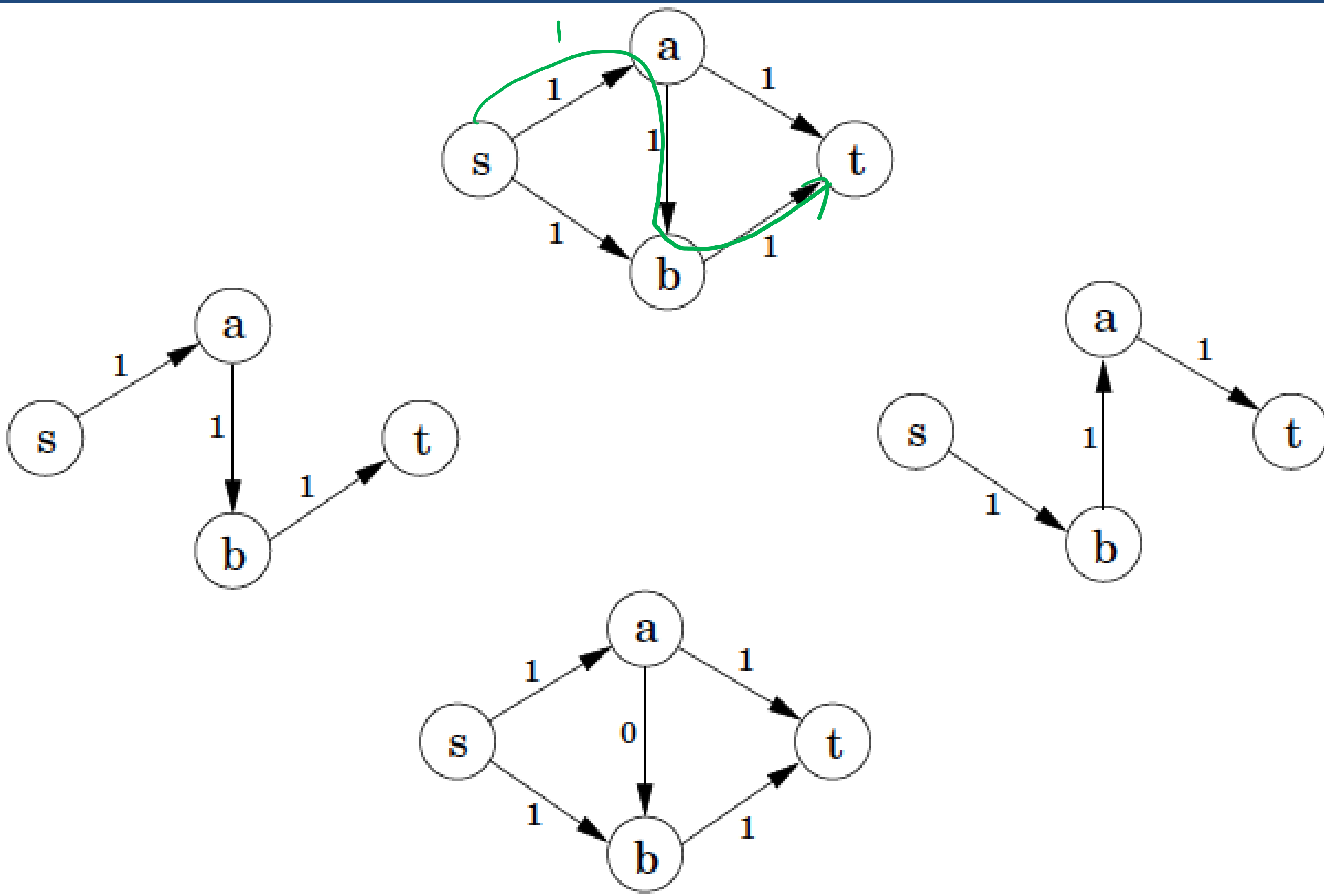
Sketch of Network Flow Algorithm







Sketch of Network Flow Algorithm



Running time of Flow Algorithm

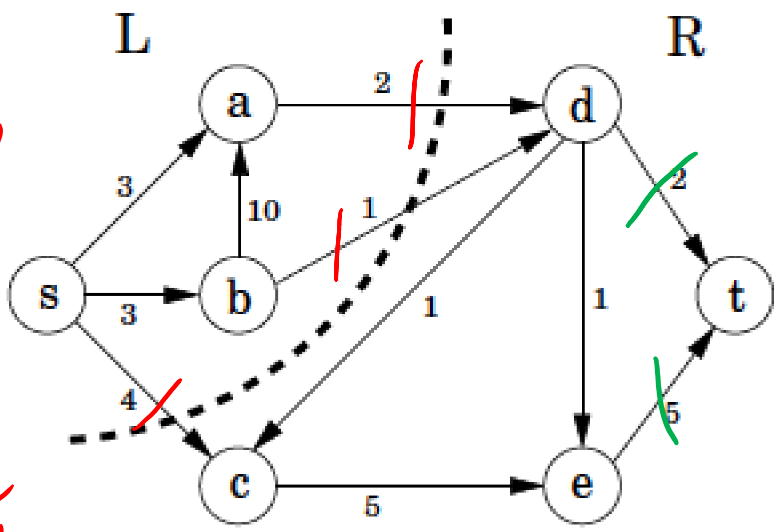
- Suppose all weights are integers
- If you choose paths unwisely, or send flow unwisely it can take a very long time, if the edge capacities are large!
  - E.g., only push one unit of flow at a time
- But if you pick paths sensibly, using BFS to look for shortest remaining path, you get $O(|V| * |E|^2)$

and send max flow poss. along each path

Min-Cut vs Max Flow

- A cut in a network between S and T is a set of edges E_{cut}, such that if you remove all edges in E_{cut}, you can no longer reach T from S

min-cut: the cut b/w S & T with the smallest total weight

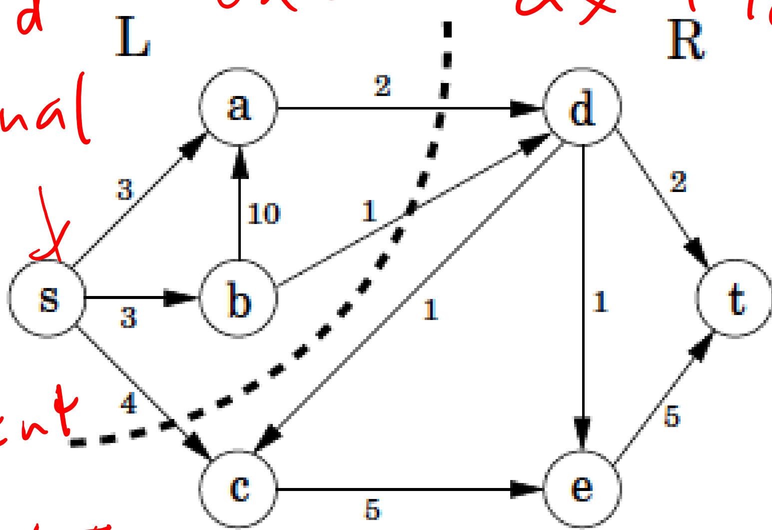


min-cut has weight 7

Min-Cut vs Max Flow

- Very useful property: the weight of the minimum weight cut is equal to the maximum flow!

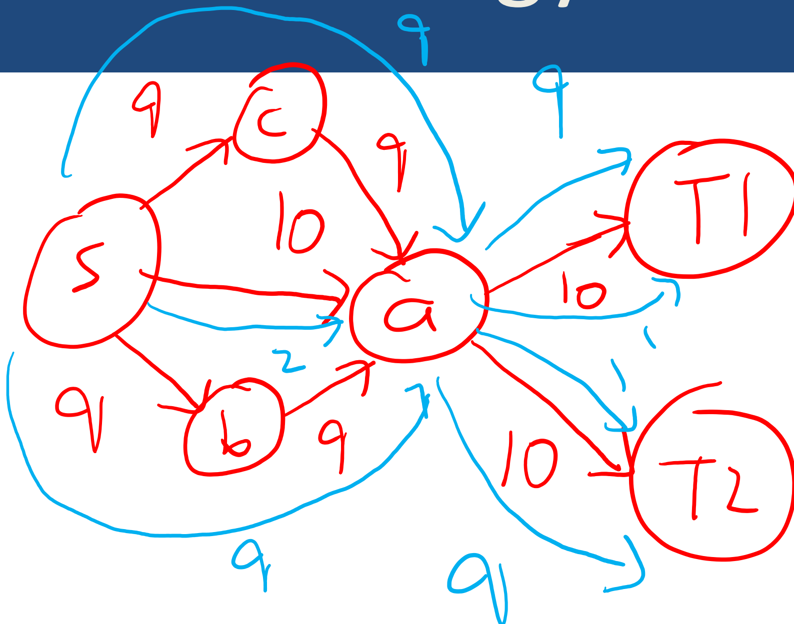
- given a directed, weighted graph, if you find the max flow from s→t, that is equal to weight of the min-cut between S/T

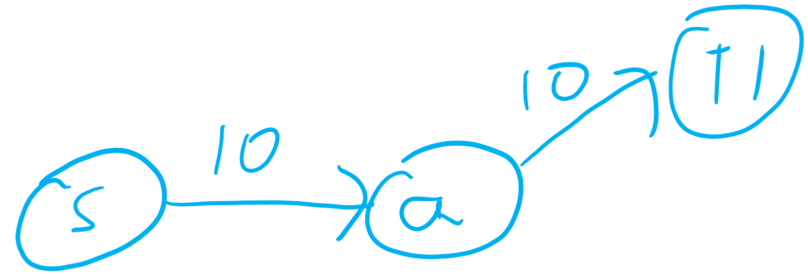


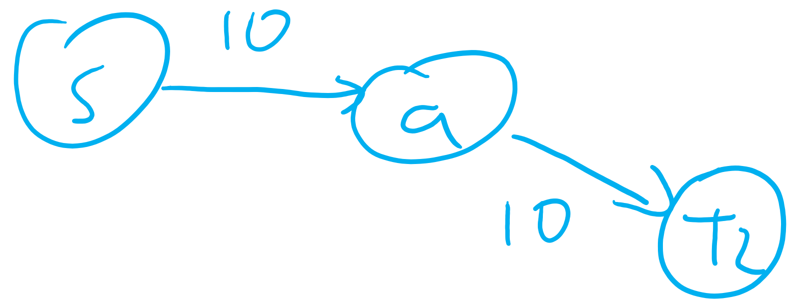
Useful Strategy: Modifying the Input Graph

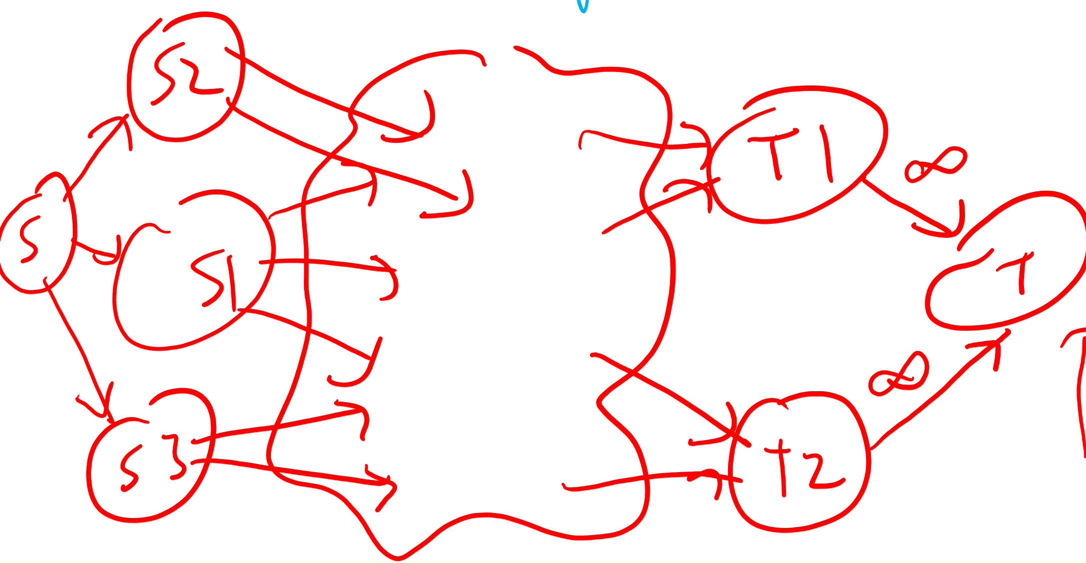
- Many problems with network flow require you to modify the input graph
- Example: Suppose you are given a graph with positive integer capacities. You are given a source node S, and two target nodes T_1, T_2. You want to maximize the total amount of flow sent to T_1 and T_2 combined.
- How can you use the existing flow algorithm on a modified graph to answer this question?

Useful Strategy: Modifying the Input Graph





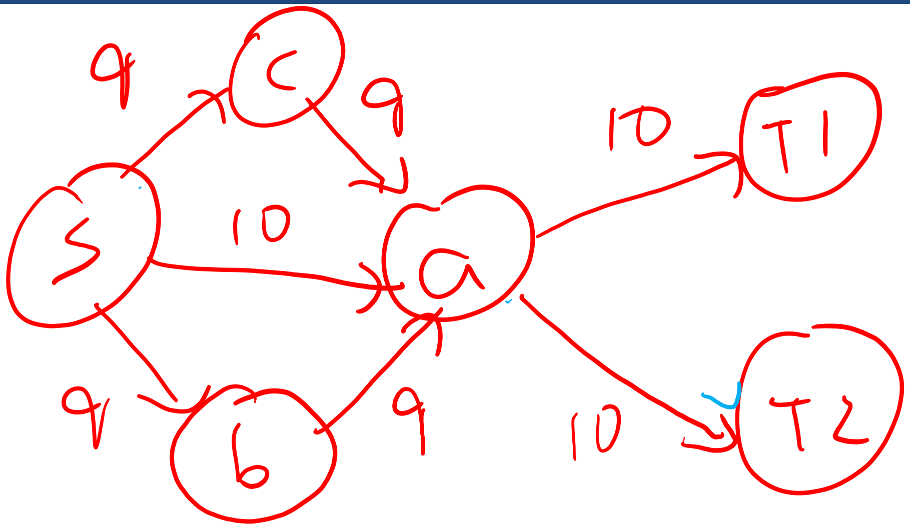




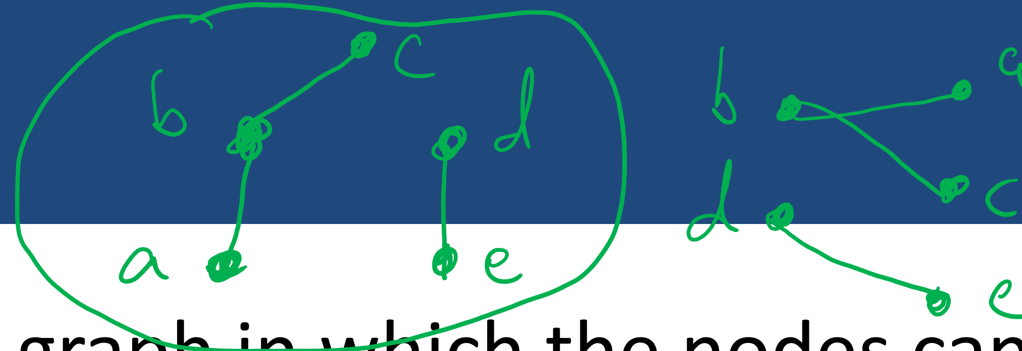
amount of flow entering T = flow @ T1 + flow @ T2

↑ target

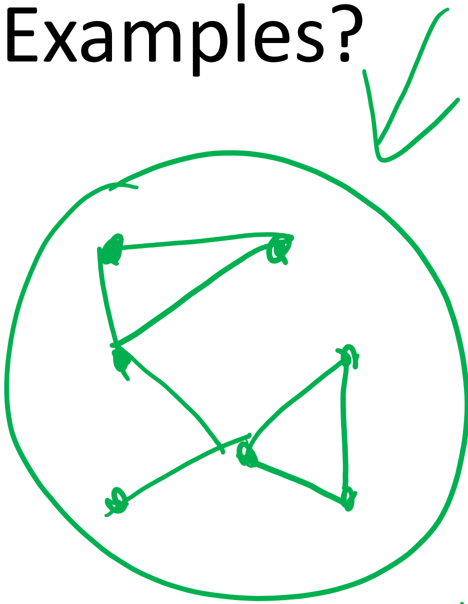
Useful Strategy: Modifying the Input Graph

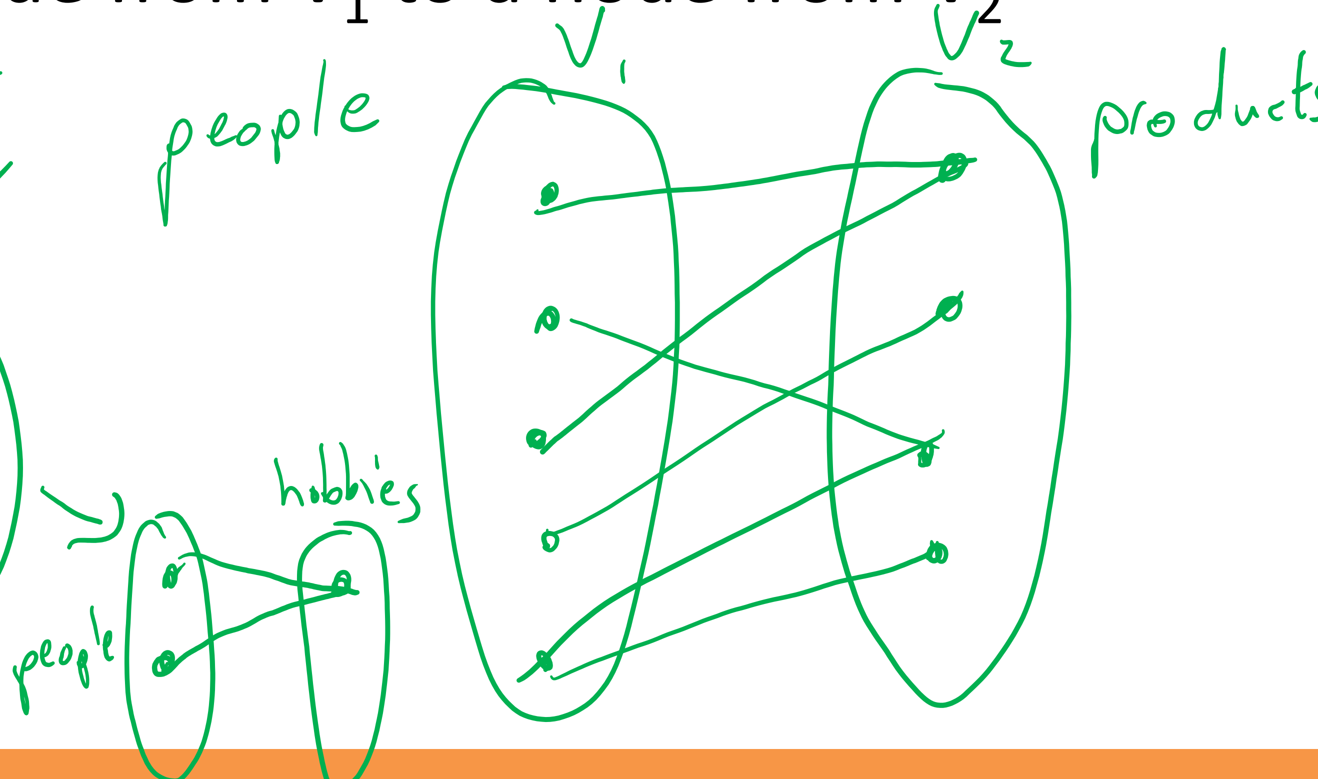


Bipartite Graphs



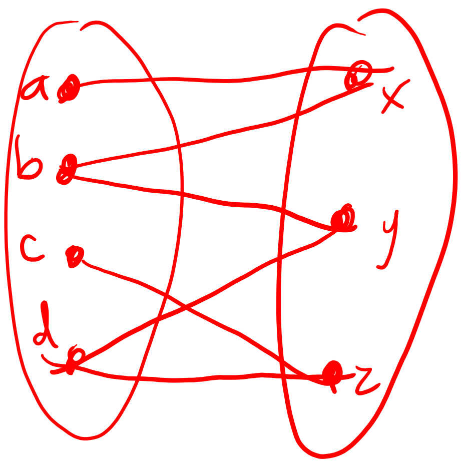
- A bipartite graph is a graph in which the nodes can be divided into two sets V_1, V_2, and all edges connect a node from V_1 to a node from V_2
- Examples?





Bipartite Graphs

- How do we tell if a graph is bipartite?



	a	b	c	d	x	y	z
b	2	0	4	2	1	1	3

even odd

1. Run BFS
2. Split into odd/even sets
3. Check that no edges are in same set

a b c d

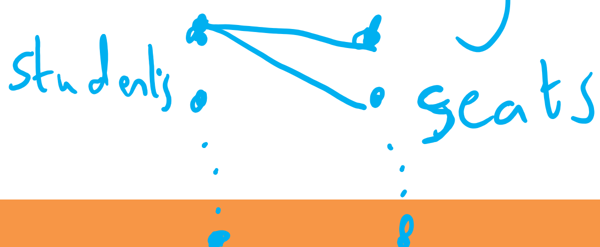
x y z

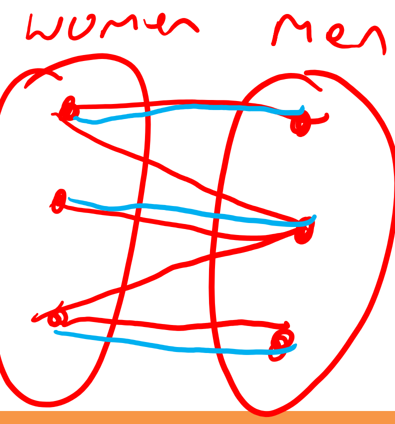
$S1 \rightarrow S2 \rightarrow S1 \rightarrow S2 \rightarrow \ldots$

Bipartite Matching

- A bipartite graph is a graph in which the nodes can be divided into two sets V_1, V_2, and all edges connect a node from V_1 to a node from V_2
- A complete matching on a bipartite graph is a set of edges such that every node is adjacent to exactly one edge (both $|V_1| = |V_2|$)
- Examples?

~~start~~ dating networks





Bipartite Matching