Network Flows

Madhavan Mukund

https://www.cmi.ac.in/~madhavan

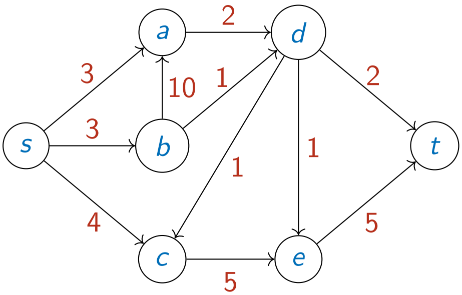
Programming, Data Structures and Algorithms using Python

Week 11

Oil network

- Network of pipelines

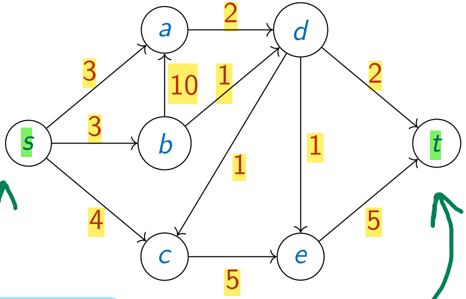
Its a directed graph.
The arrow represents direction in which oil will flow



Oil network

- Network of pipelines
- Ship as much oil as possible from s to t

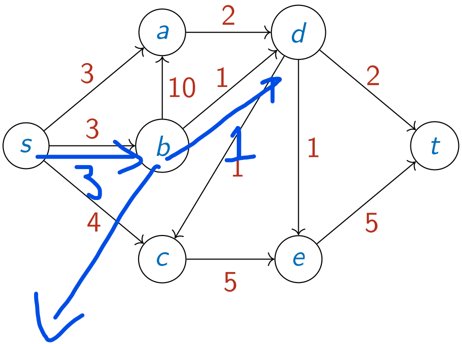
Edges are weighted, weights on edges are the capacity of that pipe line



GOAL: Take as much oil as possible from source to a target

Oil network

- Network of pipelines
- Ship as much oil as possible from s to t
- No storage along the way



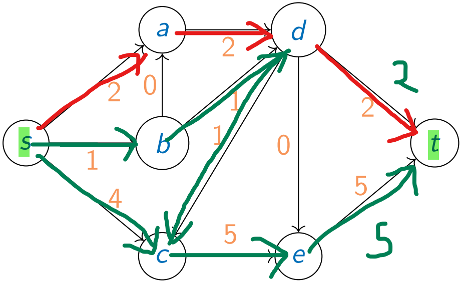
Store 2 here and send 1 ahead

THIS TYPE OF STORING IS NOT ALLOWED

Oil network

- Network of pipelines
- Ship as much oil as possible from s to t
- No storage along the way
- A flow of 7 is possible

Remember s: source and t: target

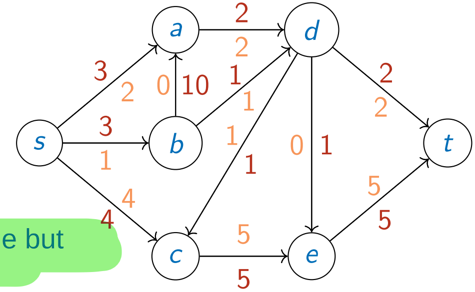


$2 + 5 = 7$

Oil network

- Network of pipelines
- Ship as much oil as possible from s to t
- No storage along the way
- A flow of 7 is possible
- Is this the maximum?

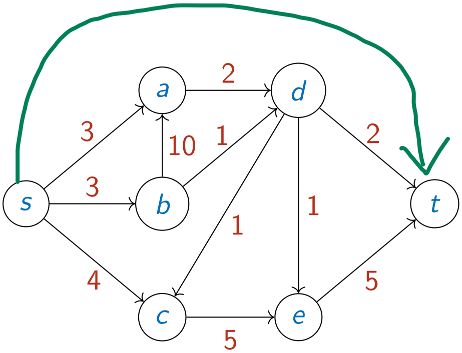
Transferring 7 from source to target node is possible but is this the maximum amount that we can do?



Oil network

- Network: graph $G = (V, E)$
- Special nodes: s (source), t (sink)

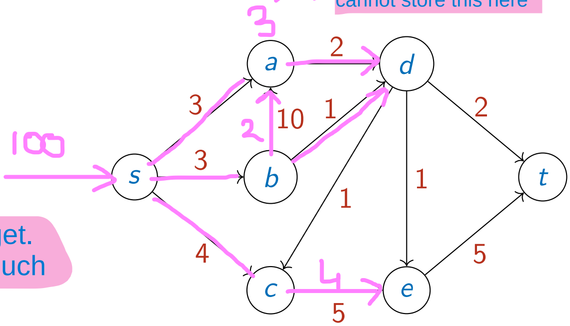
Our goal is to find maximum possible liquid we can transfer from (s) source node to (t) target node through this pipes



Oil network

- Network: graph $G = (V, E)$
- Special nodes: s (source), t (sink)
- Each edge e has capacity c_e

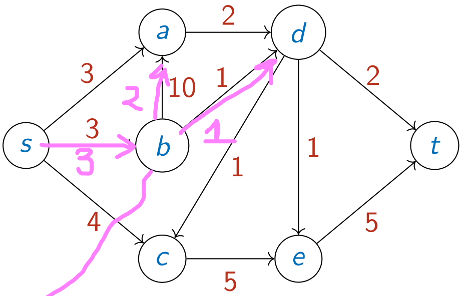
1 left, and constraint is you cannot store this here



You simply cannot, say 100 from source to target.
As there are no pipes which can transfer this much

Oil network

- Network: graph $G = (V, E)$
- Special nodes: s (source), t (sink)
- Each edge e has capacity c_e
- Flow: f_e for each edge e
  - $f_e \leq c_e$ Flow cannot be more than capacity
  - At each node, except s and t, sum of incoming flows equal sum of outgoing flows

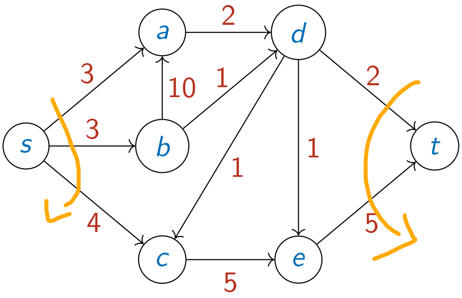


For "b" if input flow is 3 then total output flow must be 3 as well

Oil network

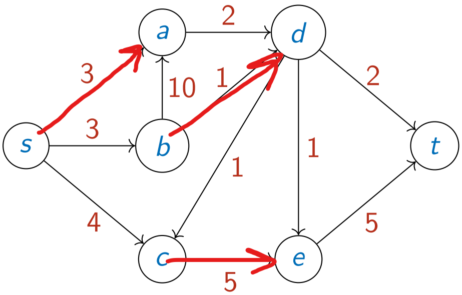
- Network: graph $G = (V, E)$
- Special nodes: s (source), t (sink)
- Each edge e has capacity c_e
- Flow: f_e for each edge e
  - $f_e \leq c_e$
  - At each node, except s and t, sum of incoming flows equal sum of outgoing flows
- Total volume of flow is sum of outgoing flow from s

Alternatively sum of incoming flow at target node



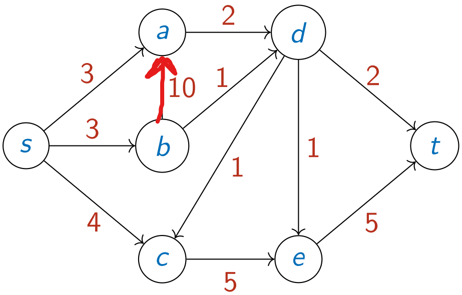
LP formulation

- Variable f_e for each edge e
  - f_{sa}, f_{bd}, f_{ce}, ...



LP formulation

- Variable f_e for each edge e
  - f_{sa}, f_{bd}, f_{ce}, ...
- Capacity constraints per edge
  - $f_{ba} \leq 10, \ldots$

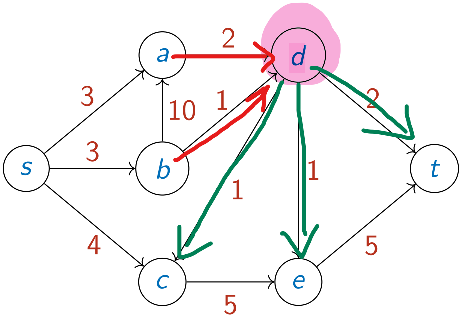


LP formulation

- Variable f_e for each edge e
  - f_{sa}, f_{bd}, f_{ce}, ...
- Capacity constraints per edge
  - $f_{ba} \leq 10$, ...
- Conservation of flow at each internal node
  - $f_{ad} + f_{bd} = f_{dc} + f_{de} + f_{dt}$, ...

Sum of input flow and outflow must be conserved

Consider the node "d"

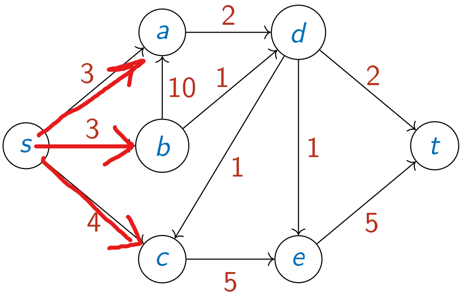


output

input

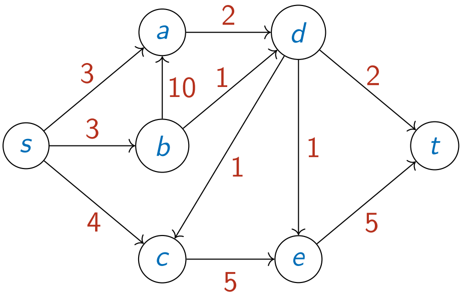
LP formulation

- Variable f_e for each edge e
  - $f_{sa}, f_{bd}, f_{ce}, \ldots$
- Capacity constraints per edge
  - $f_{ba} \leq 10, \ldots$
- Conservation of flow at each internal node
  - $f_{ad} + f_{bd} = f_{dc} + f_{de} + f_{dt}, \ldots$
- Objective: maximize flow volume
  - Maximize $f_{sa} + f_{sb} + f_{sc}$



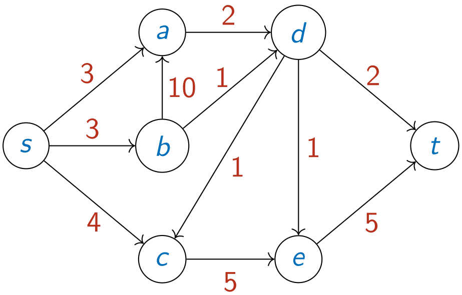
LP formulation

- Variable f_e for each edge e
  - $f_{sa}, f_{bd}, f_{ce}, \ldots$
- Capacity constraints per edge
  - $f_{ba} \leq 10, \ldots$
- Conservation of flow at each internal node
  - $f_{ad} + f_{bd} = f_{dc} + f_{de} + f_{dt}, \ldots$
- Objective: maximize flow volume
  - Maximize $f_{sa} + f_{sb} + f_{sc}$
- Simplex explores vertices of feasible region to solve LP, find maximum flow



LP formulation

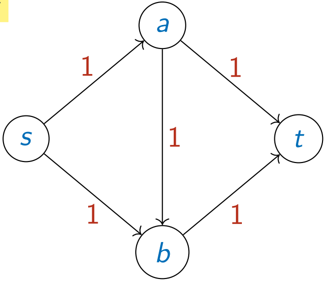
- Variable f_e for each edge e
  - $f_{sa}, f_{bd}, f_{ce}, \ldots$
- Capacity constraints per edge
  - $f_{ba} \leq 10, \ldots$
- Conservation of flow at each internal node
  - $f_{ad} + f_{bd} = f_{dc} + f_{de} + f_{dt}, \ldots$
- Objective: maximize flow volume
  - Maximize $f_{sa} + f_{sb} + f_{sc}$
- Simplex explores vertices of feasible region to solve LP, find maximum flow
- Moving from vertex to vertex gives a more direct algorithm for maximum flow



Ford-Fulkerson algorithm

- Start with zero flow

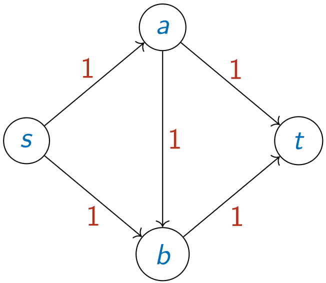
Currently no edge has flow, every edge (pipe) is empty



Ford-Fulkerson algorithm

- Start with zero flow
- Choose a path from s to t that is not saturated and augment the flow as much as possible

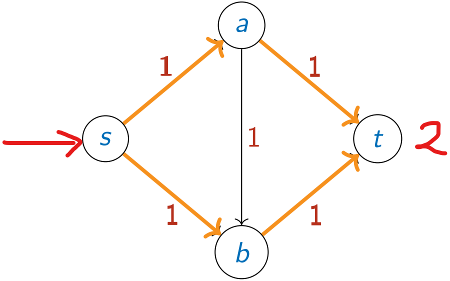
In other words add some flow to edges which have capacity



Ford-Fulkerson algorithm

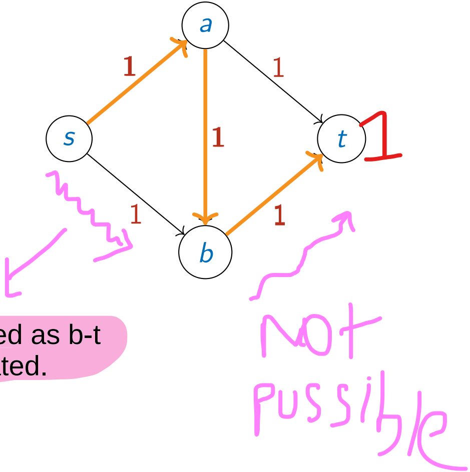
- Start with zero flow
- Choose a path from s to t that is not saturated and augment the flow as much as possible
- Network on the right has max flow 2

This network has max flow



Ford-Fulkerson algorithm

- Start with zero flow
- Choose a path from s to t that is not saturated and augment the flow as much as possible
- Network on the right has max flow 2
- What if one chooses a bad flow to begin with?

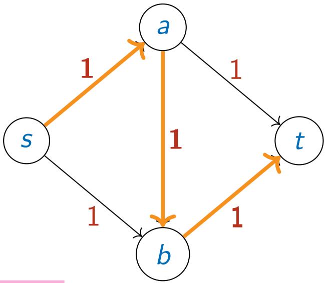


This cannot be used as b-t has already saturated.

Ford-Fulkerson algorithm

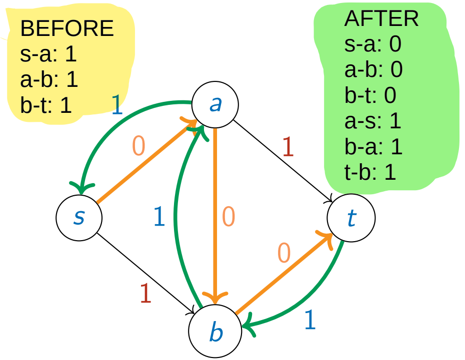
- Start with zero flow
- Choose a path from s to t that is not saturated and augment the flow as much as possible
- Network on the right has max flow 2
- What if one chooses a bad flow to begin with?
- Add reverse edges to undo flow from previous steps

In the last step we made a mistake in choosing the flow. Now what can we do?



Ford-Fulkerson algorithm

- Start with zero flow
- Choose a path from s to t that is not saturated and augment the flow as much as possible
- Network on the right has max flow 2
- What if one chooses a bad flow to begin with?
- Add reverse edges to undo flow from previous steps
- Residual graph: for each edge e with capacity c_e and current flow f_e
  - Reduce capacity to $c_e - f_e$
  - Add reverse edge with capacity f_e



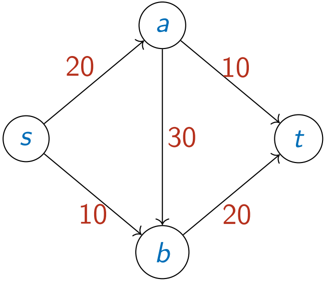
1. c_e: Capacity of the edge e originally

2. Reduce capacity of edge e from c_e to c_e - f_e

3. f_e is the current flow of edge e

Ford-Fulkerson algorithm

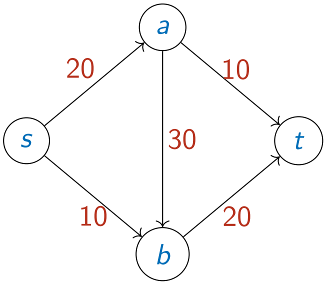
- Start with zero flow



Ford-Fulkerson algorithm

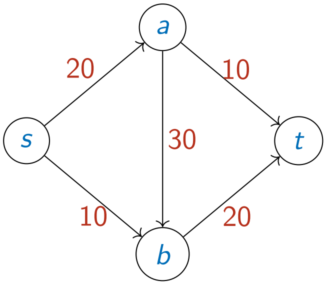
- Start with zero flow
- Choose a path from s to t that is not saturated and augment the flow as much as possible

1. You can choose "some" path from s to t
2. The path should not necessarily be the optimal one



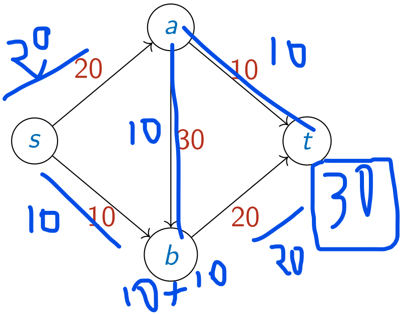
Ford-Fulkerson algorithm

- Start with zero flow
- Choose a path from s to t that is not saturated and augment the flow as much as possible
- Build residual graph



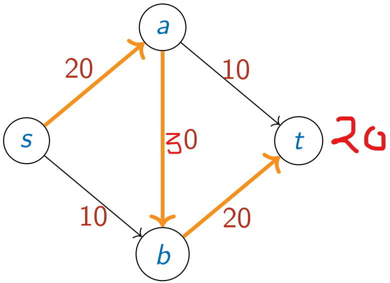
Ford-Fulkerson algorithm

- Start with zero flow
- Choose a path from s to t that is not saturated and augment the flow as much as possible
- Build residual graph
- Repeat the previous two steps till there is no feasible flow from s to t



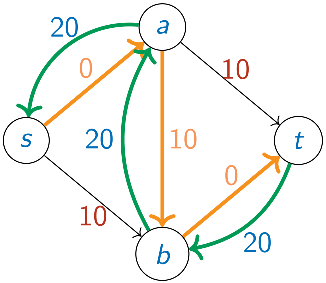
Ford-Fulkerson algorithm

- Start with zero flow
- Choose a path from s to t that is not saturated and augment the flow as much as possible
- Build residual graph
- Repeat the previous two steps till there is no feasible flow from s to t
- Flow 20, $s - a - b - t$,



Ford-Fulkerson algorithm

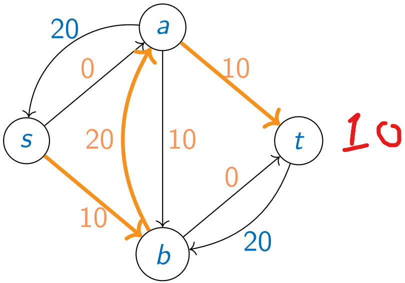
- Start with zero flow
- Choose a path from s to t that is not saturated and augment the flow as much as possible
- Build residual graph
- Repeat the previous two steps till there is no feasible flow from s to t
- Flow 20, $s - a - b - t$, build residual graph



1. Now you work on the residual.
2. You never go back to the original graph instead you keep working on the new residual graphs
3. The goal remains the same which is maximizing flow from source to target

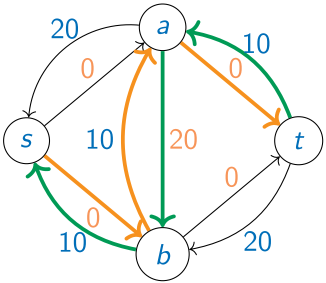
Ford-Fulkerson algorithm

- Start with zero flow
- Choose a path from s to t that is not saturated and augment the flow as much as possible
- Build residual graph
- Repeat the previous two steps till there is no feasible flow from s to t
- Flow 20, $s - a - b - t$, build residual graph
- Add flow 10, $s - b - a - t$,



Ford-Fulkerson algorithm

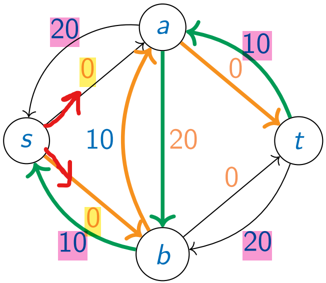
- Start with zero flow
- Choose a path from s to t that is not saturated and augment the flow as much as possible
- Build residual graph
- Repeat the previous two steps till there is no feasible flow from s to t
- Flow 20, $s - a - b - t$, build residual graph
- Add flow 10, $s - b - a - t$, build residual graph



Now again you work on the new residual graph, the goal remains the same which is maximizing flow from source to target

Ford-Fulkerson algorithm

- Start with zero flow
- Choose a path from s to t that is not saturated and augment the flow as much as possible
- Build residual graph
- Repeat the previous two steps till there is no feasible flow from s to t
- Flow 20, $s - a - b - t$, build residual graph
- Add flow 10, $s - b - a - t$, build residual graph
- No more feasible paths from s to t



So maximum possible flow from source to node is 10 + 20 = 30

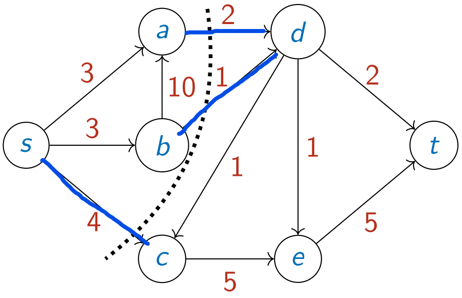
Certificate of optimality

- Edges $\{ad, bd, sc\}$ disconnect s and t
  - (s, t)-cut

On disconnecting edges {ad, bd,sc} cut all the paths from s to t

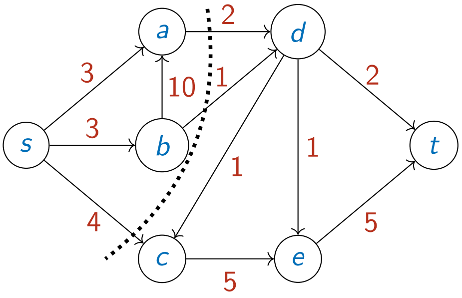
In other words, on disconnecting these 3 edges you cannot reach s to t

Such a set of edges which disconnects s and t is called a (s, t)-cut



Certificate of optimality

- Edges $\{ad, bd, sc\}$ disconnect s and t
  - (s, t)-cut
- Flow from s to t must go through this cut



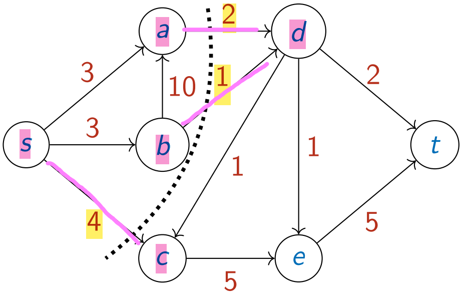
Certificate of optimality

- Edges $\{ad, bd, sc\}$ disconnect s and t
  - (s, t)-cut
- Flow from s to t must go through this cut
- Cannot exceed cut capacity, 7

This is because maximum flow that can pass through cut is 7.

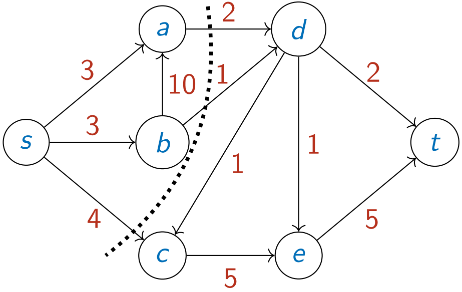
And as the only way to reach s to t is through cut

This implies that maximum flow cannot exceed 7 which is the cut capacity



Certificate of optimality

- Edges $\{ad, bd, sc\}$ disconnect s and t
  - (s, t)-cut
- Flow from s to t must go through this cut
- Cannot exceed cut capacity, 7
- Max flow cannot exceed capacity of min cut

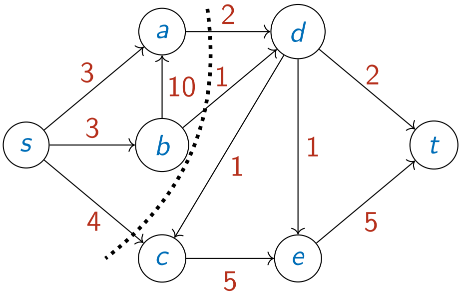


min cut: Minimum number of edges to disconnect s and t

Example: Edges {ad, bd,sc, sb} is a cut but it is not a min cut

Certificate of optimality

- Edges $\{ad, bd, sc\}$ disconnect s and t
  - (s,t)-cut
- Flow from s to t must go through this cut
- Cannot exceed cut capacity, 7
- Max flow cannot exceed capacity of min cut



Max flow-min cut theorem

- In fact, max flow is always equal to min cut

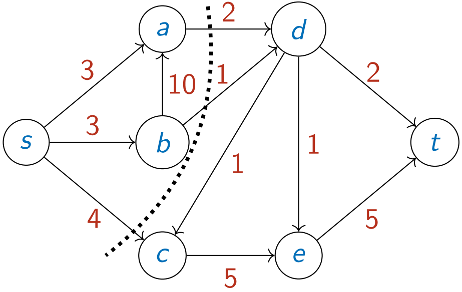
We know that max flow cannot exceed the cut capacity

But it is a fact that max flow is always equal to min cut

This implies that for the above graph which has min cut 7 also has max flow of 7

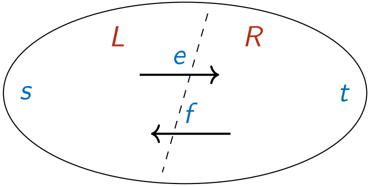
Certificate of optimality

- Edges $\{ad, bd, sc\}$ disconnect s and t
  - (s,t)-cut
- Flow from s to t must go through this cut
- Cannot exceed cut capacity, 7
- Max flow cannot exceed capacity of min cut



Max flow-min cut theorem

- In fact, max flow is always equal to min cut
- At max flow, no path from s to t in residual graph
  - s can reach L, R can reach t
  - Any edge from L to R must be at full capacity
  - Any edge from R to L must be at zero capacity



Ford-Fulkerson algorithm

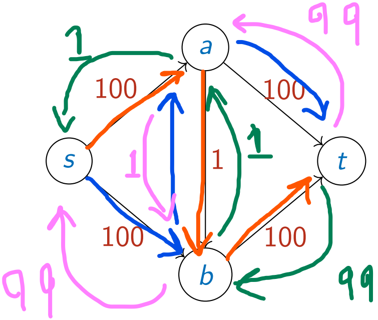
- Choose augmenting paths wisely

— 1

— 2 residual

— 3

— 4 residual

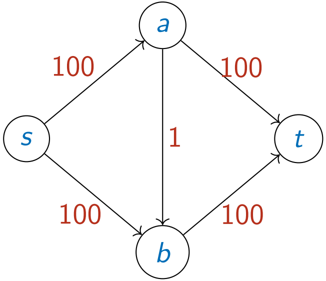


So in 2 iterations I managed to reduce from 100 to 99
It will take 200 iterations for reaching the final solution

Ford-Fulkerson algorithm

- Choose augmenting paths wisely
- If we keep going through the middle edge, 200 iterations to find the max flow
  - Ford-Fulkerson can take time proportional to max capacity

Here max capacity is 100



Ford-Fulkerson algorithm

- Choose augmenting paths wisely
- If we keep going through the middle edge, 200 iterations to find the max flow
  - Ford-Fulkerson can take time proportional to max capacity
- Use BFS to find augmenting path with fewest edges
- Iterations bounded by $|V| \times |E|$, regardless of capacities

No. of vertices * No. of edges

This is definitely better than taking time proportional to max capacity