Max Flow Problem

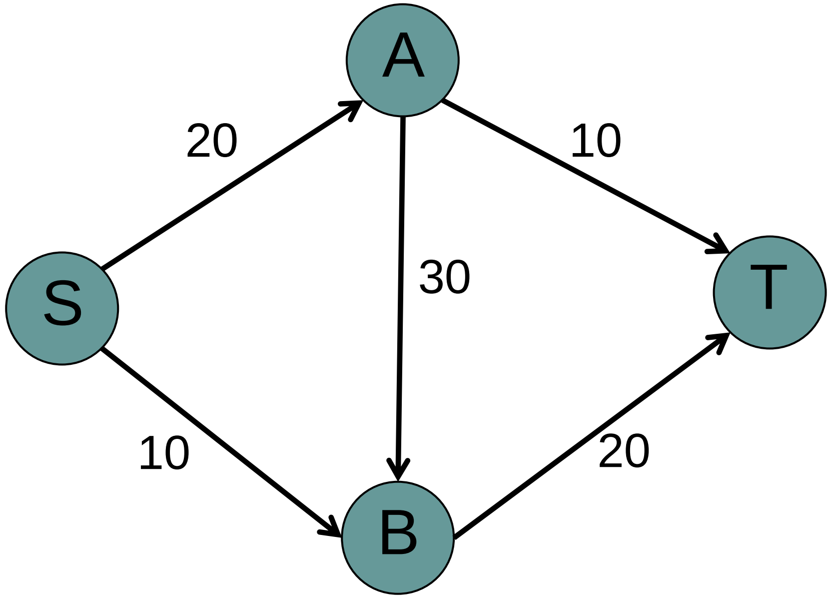
Dr. Navjot Singh

Design and Analysis of Algorithms

Student networking

You decide to create your own computer network:

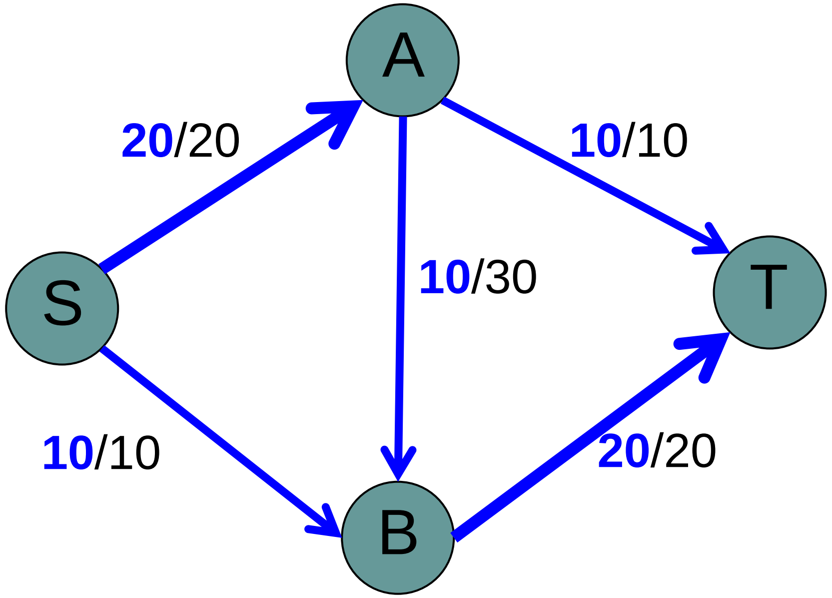
- You get three of your friends and string some network cables
- Because of capacity (due to cable type, distance, computer, etc) you can only send a certain amount of data to each person
- If edges denote capacity, what is the maximum throughput you can send from S to T?



Student networking

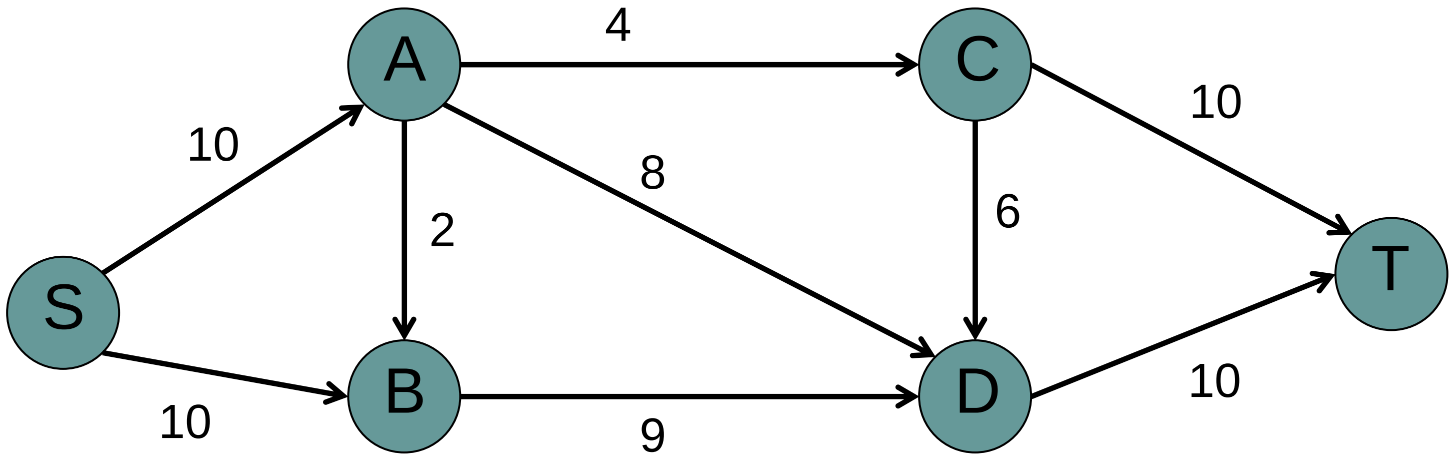
You decide to create your own computer network:

- You get three of your friends and string some network cables
- Because of capacity (due to cable type, distance, computer, etc) you can only send a certain amount of data to each person
- If edges denote capacity, what is the maximum throughput you can send from S to T?



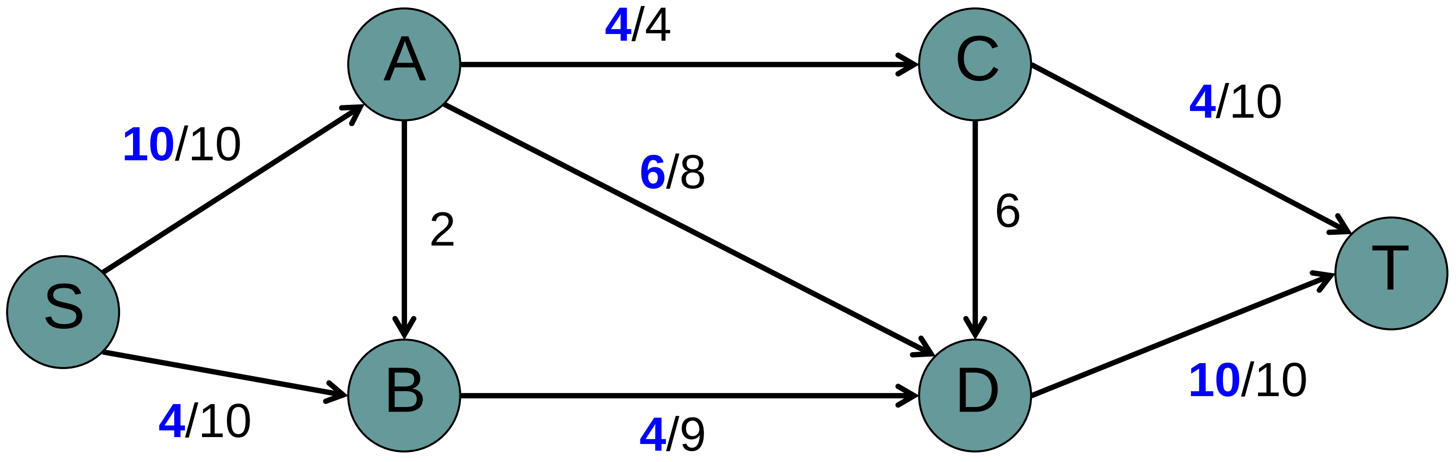
30 units

Another flow problem



How much water flow can we continually send from s to t?

Another flow problem

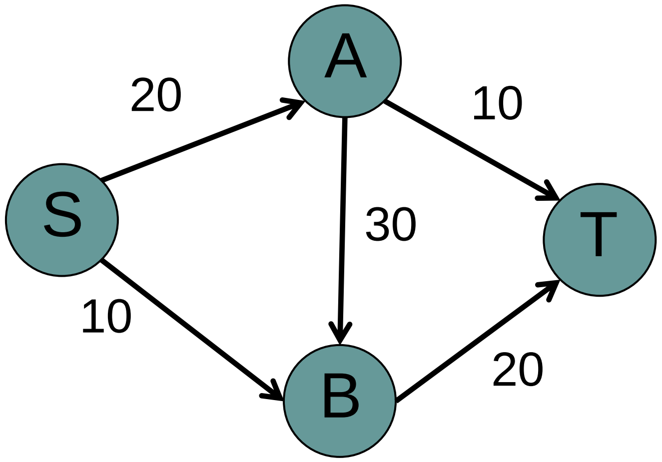


14 units

Flow graph/networks

Flow network

- directed, weighted graph (V, E)
- positive edge weights indicating the "capacity" (generally, assume integers)
- contains a single source $s \in V$ with no incoming edges
- contains a single sink/target $t \in V$ with no outgoing edges
- every vertex is on a path from *s* to *t*

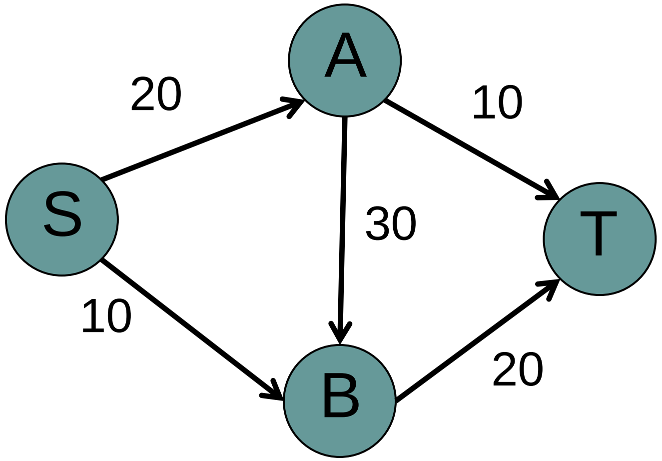


Flow constraints

in-flow = out-flow for every vertex (except s, t)

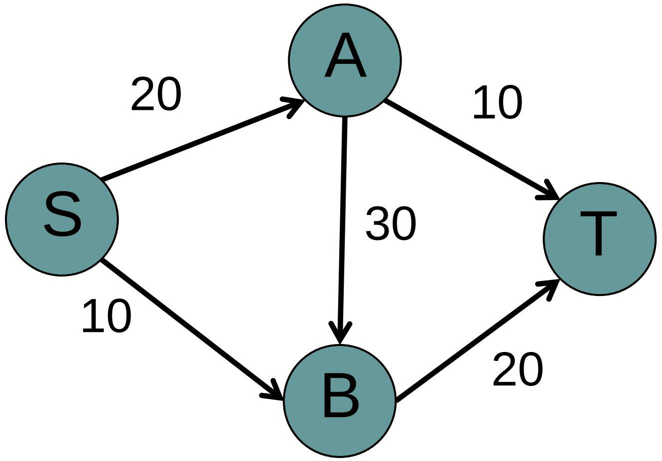
flow along an edge cannot exceed the edge capacity

flows are positive



Max flow problem

Given a flow network: *what is the maximum flow we can send from s to t that meet the flow constraints?*



Network flow properties

If one of these is true then all are true (i.e. each implies the others):

f is a maximum flow

G_f (residual graph) has no paths from s to t

$|f|$ = minimum capacity cut

Applications?

network flow

- water, electricity, sewage, cellular…
- traffic/transportation capacity

bipartite matching

sports elimination

…

Algorithm ideas?

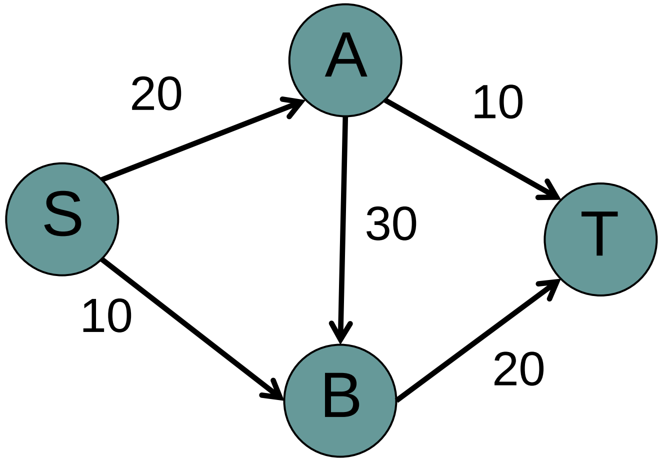
graph algorithm?

- BFS, DFS, shortest paths…
- MST

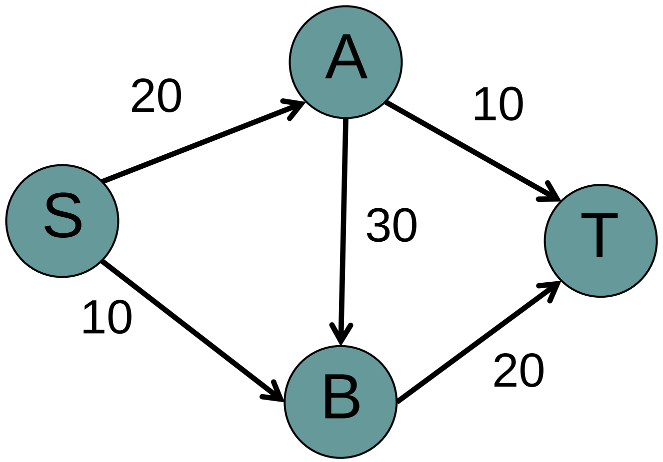
divide and conquer?

greedy?

dynamic programming?

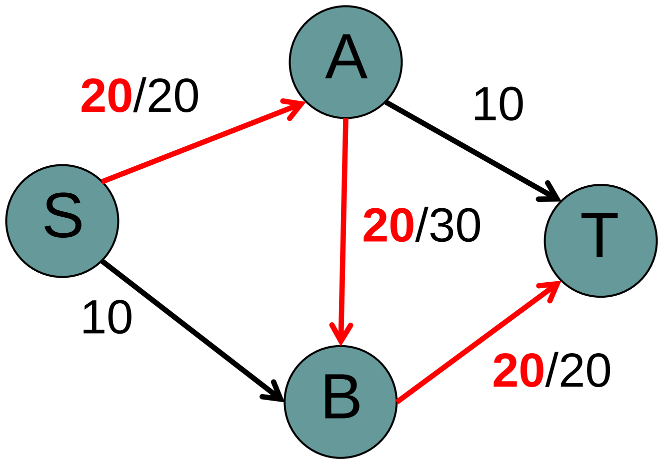


Algorithm idea



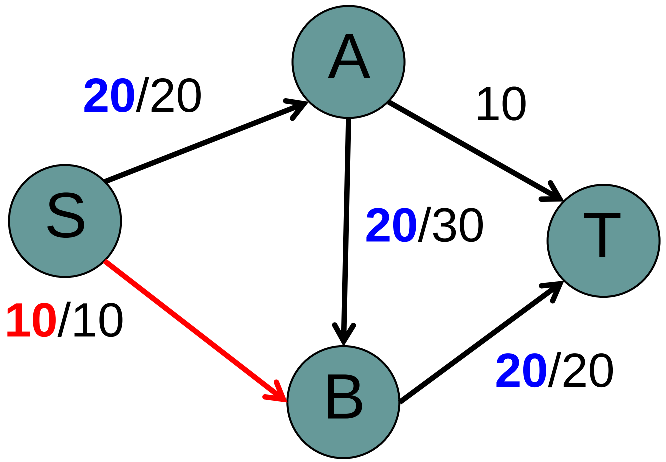
Algorithm idea

send some flow down a path



Algorithm idea

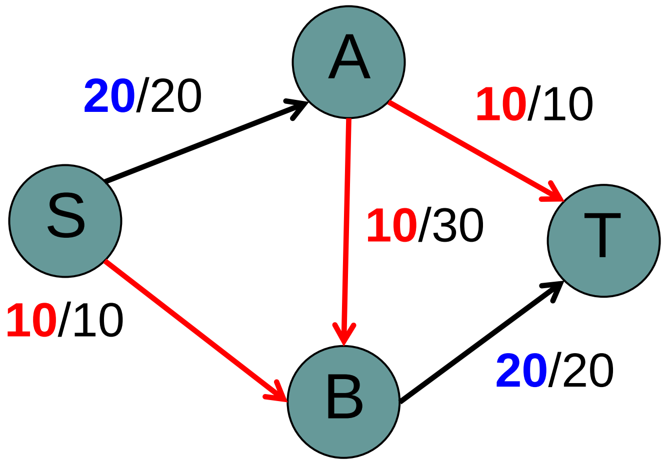
send some flow down a path



Now what?

Algorithm idea

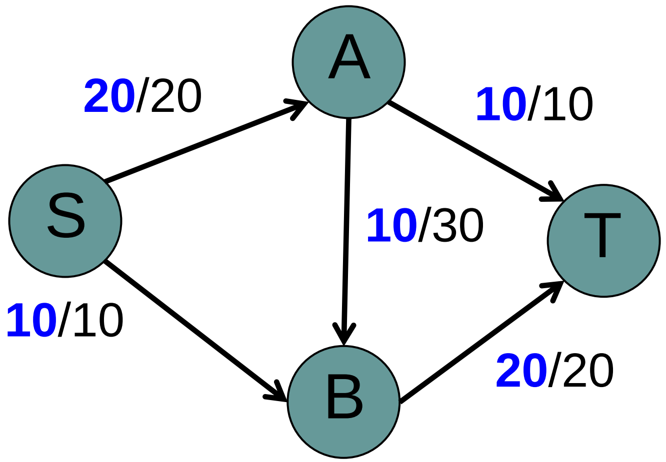
reroute some of the flow



Total flow?

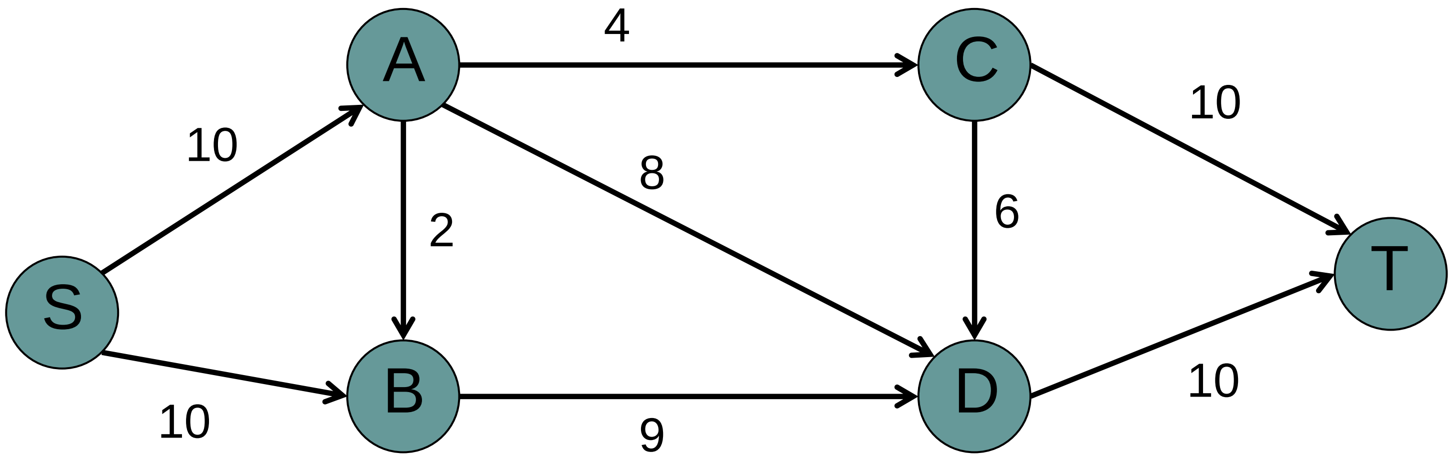
Algorithm idea

reroute some of the flow



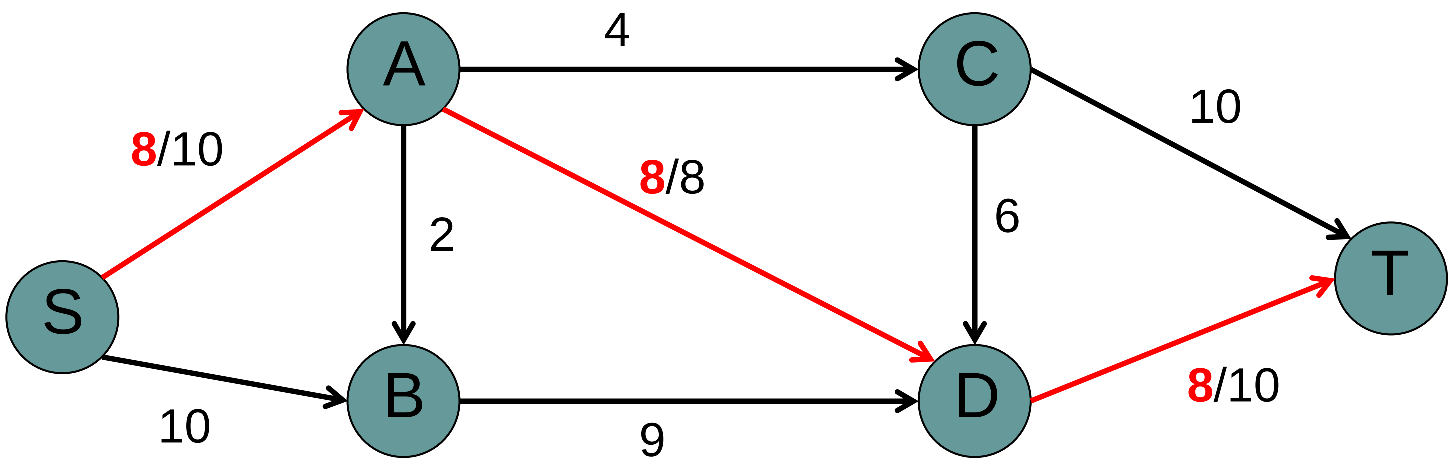
30

Algorithm idea



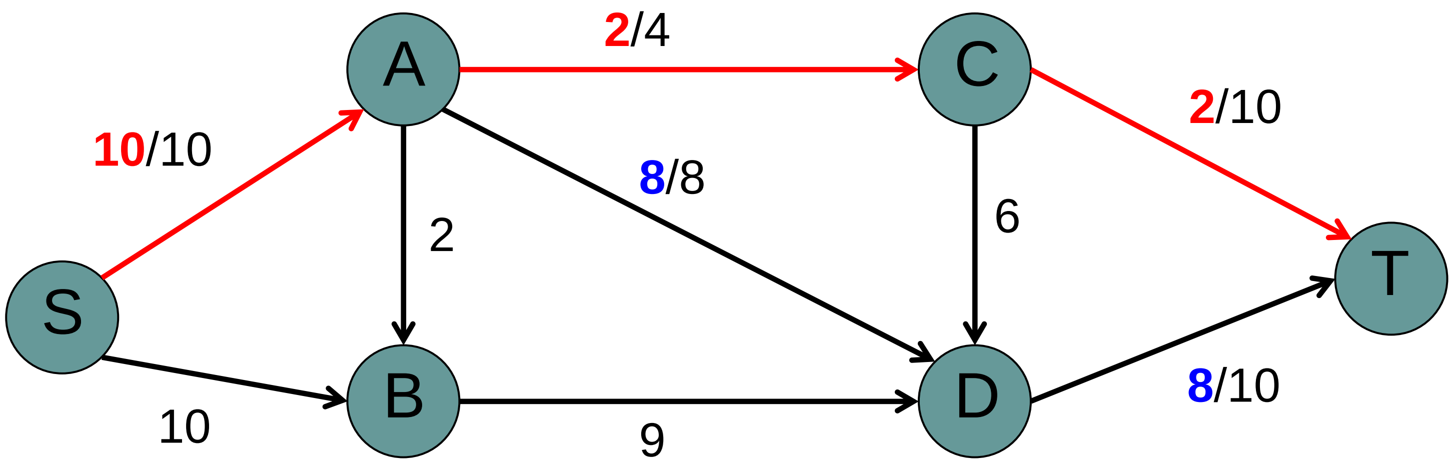
Algorithm idea

send some flow down a path



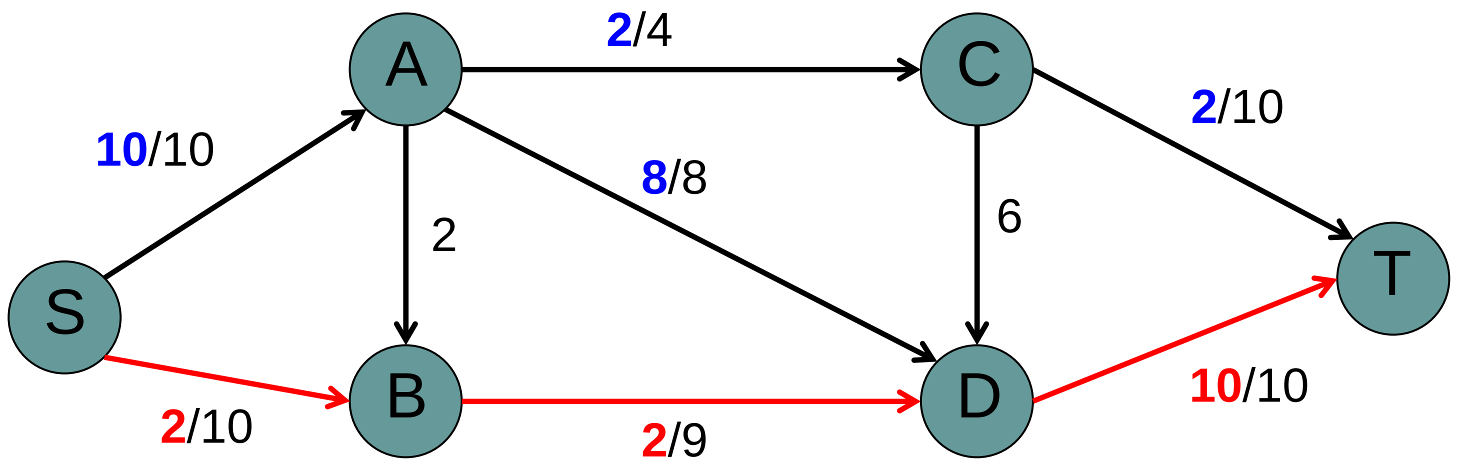
Algorithm idea

send some flow down a path

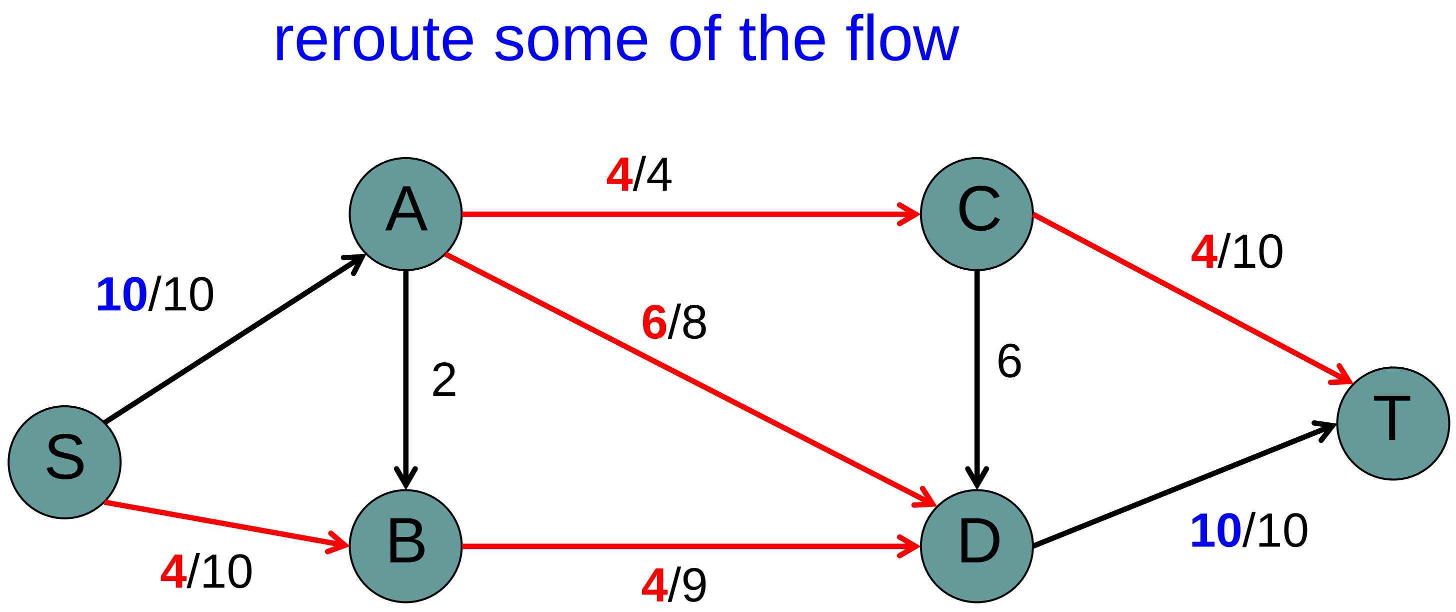


Algorithm idea

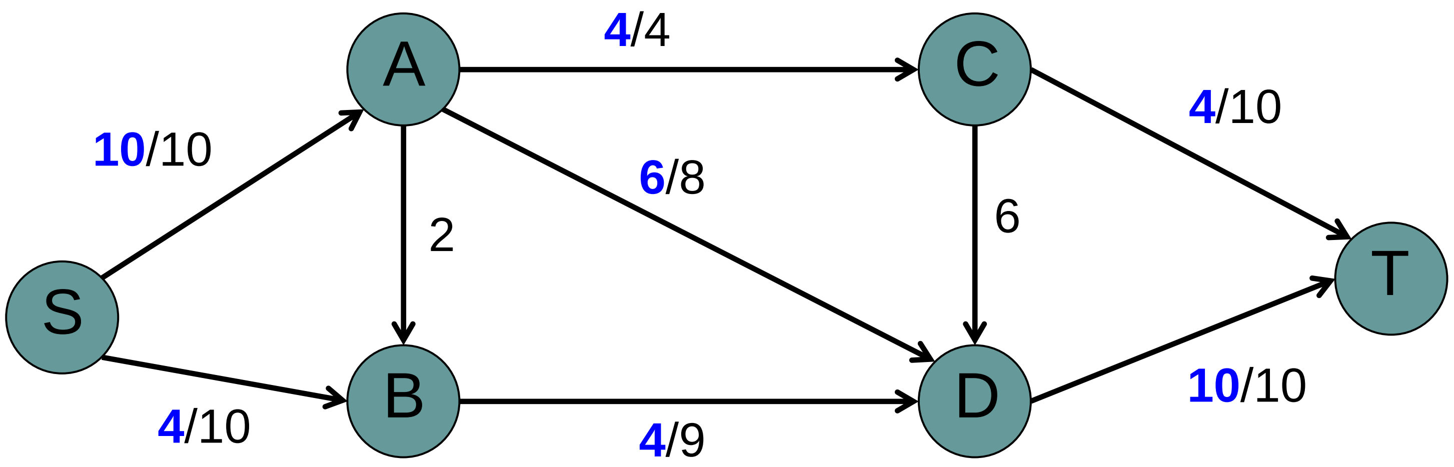
send some flow down a path



Algorithm idea



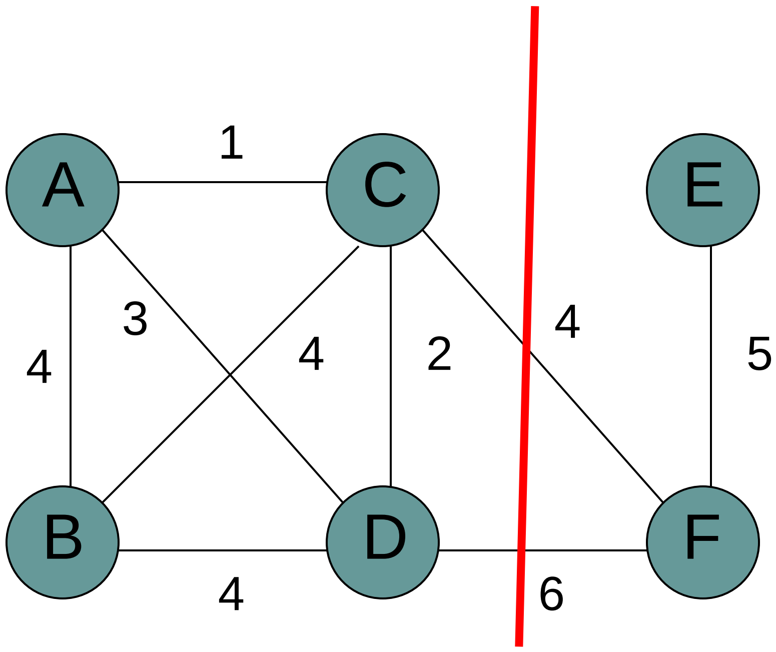
Algorithm idea



Are we done?
Is this the best we can do?

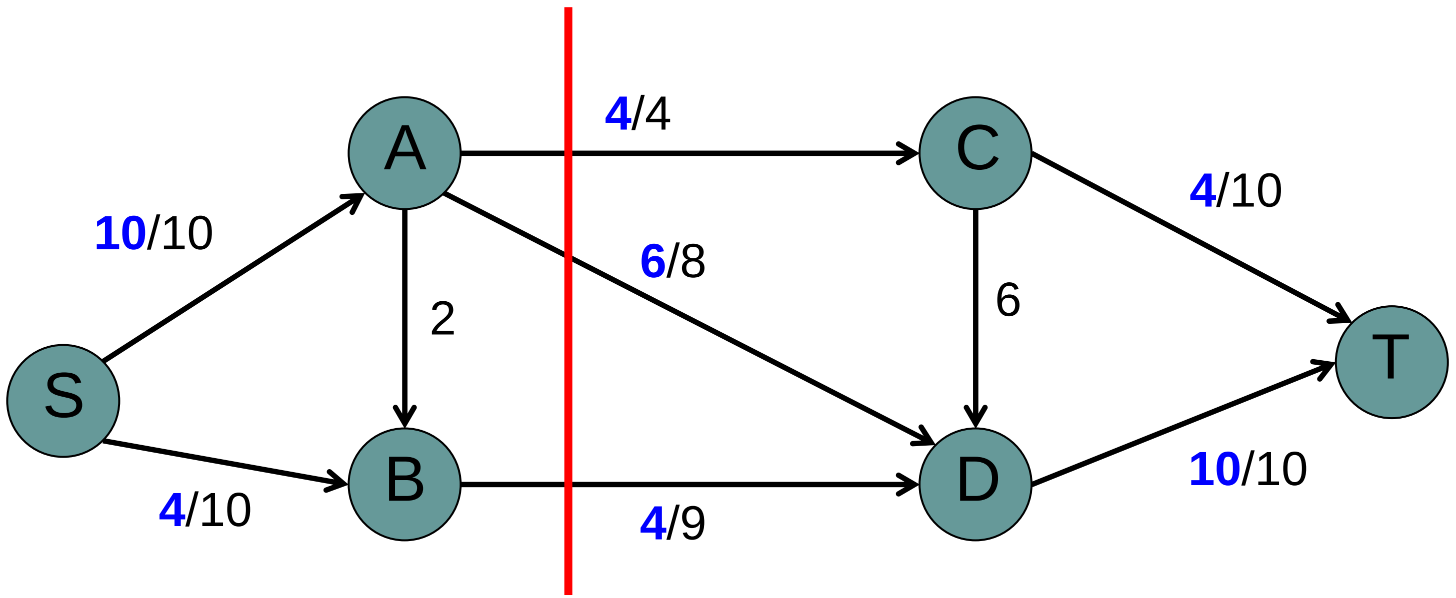
Cuts

A cut is a partitioning of the vertices into two sets S_s and $S_t = V\text{-}S_s$



Flow across cuts

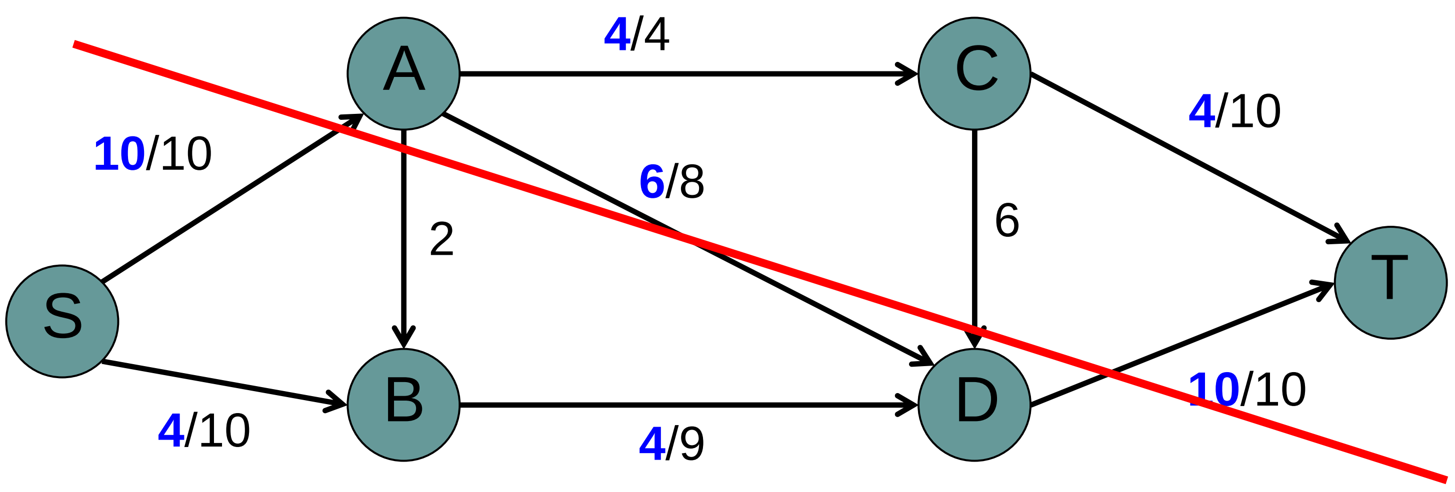
In flow graphs, we're interested in cuts that separate s from t, that is $s \in S_s$ and $t \in S_t$



Flow across cuts

The flow "across" a cut is the total flow from nodes in S_s to nodes in S_t ***minus*** the total from nodes in S_t to S_s

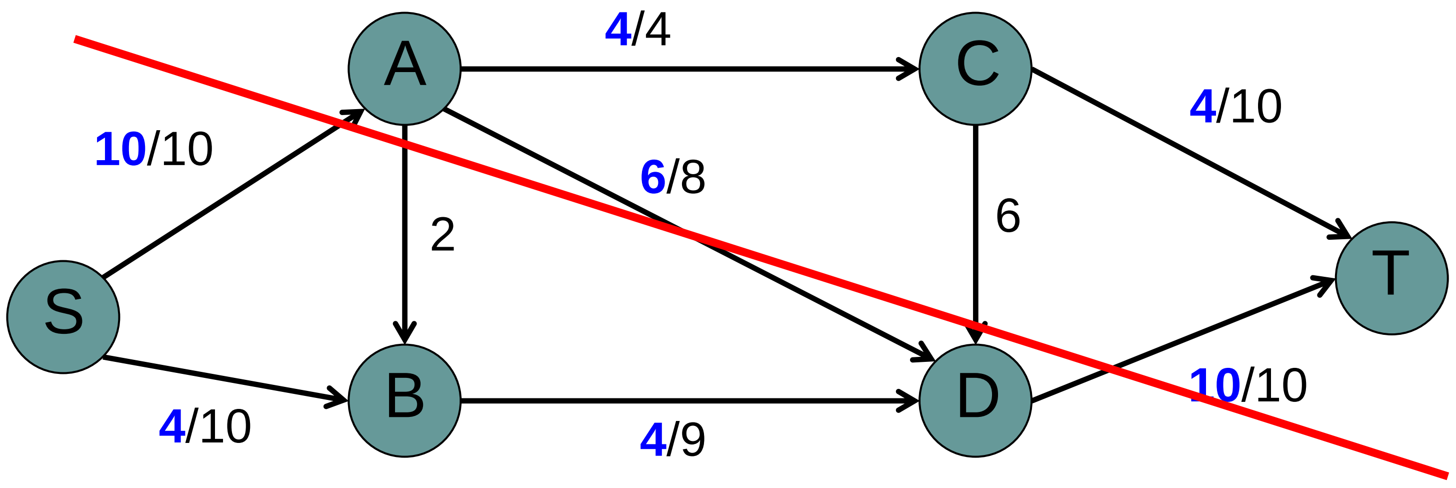
What is the flow across this cut?



Flow across cuts

The flow "across" a cut is the total flow from nodes in S_s to nodes in S_t ***minus*** the total from nodes in S_t to S_s

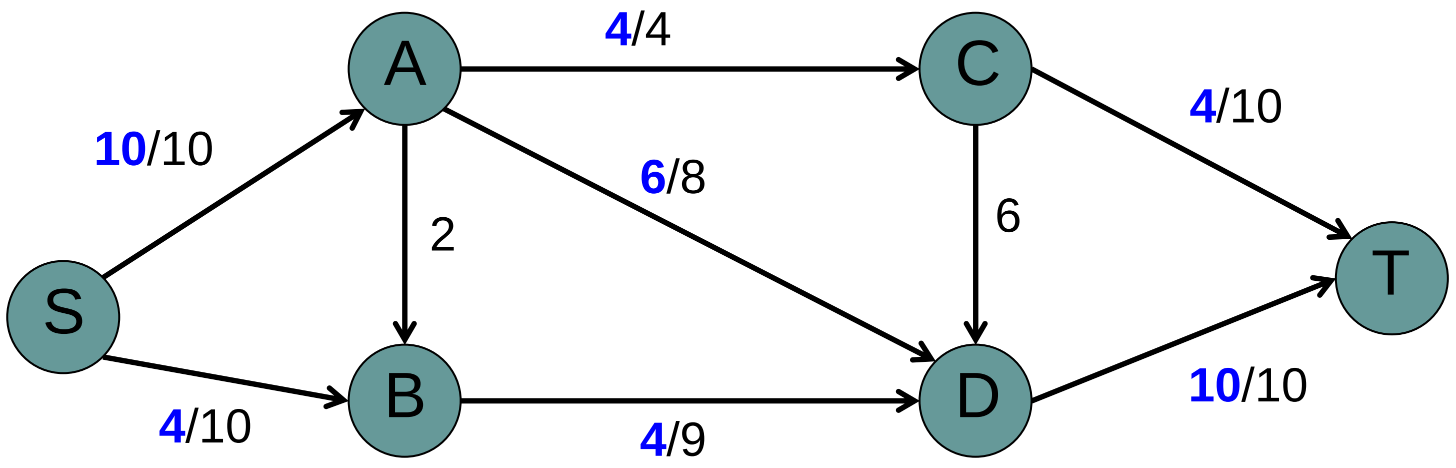
$$10+10-6 = 14$$



Flow across cuts

Consider any cut where $s \in S_s$ and $t \in S_t$, i.e. the cut partitions the source from the sink

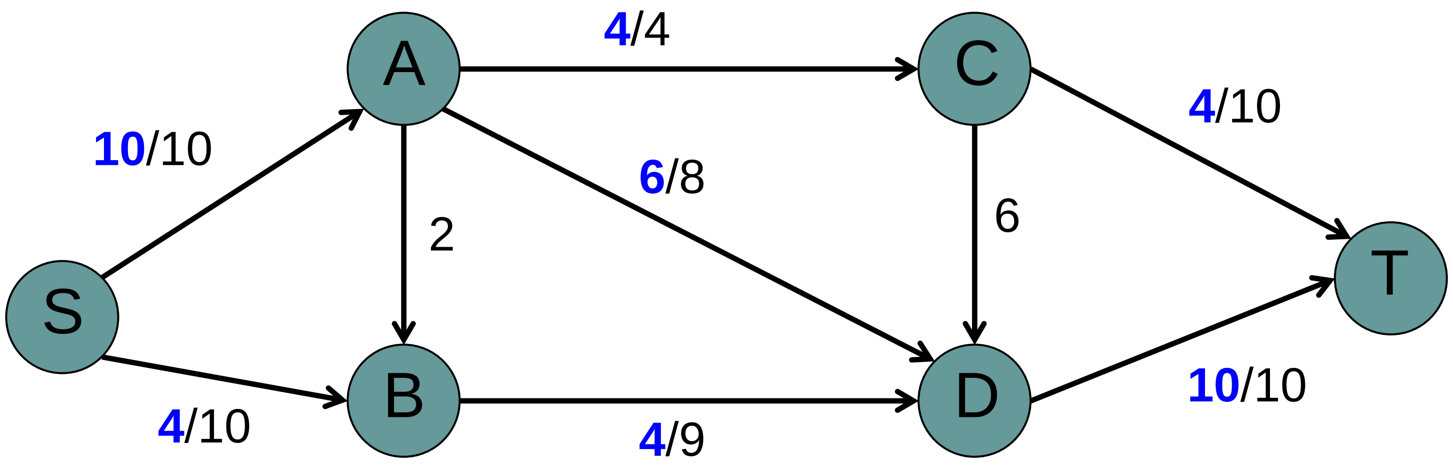
What do we know about the flow across the any such cut?



Flow across cuts

Consider any cut where $s \in S_s$ and $t \in S_t$, i.e. the cut partitions the source from the sink

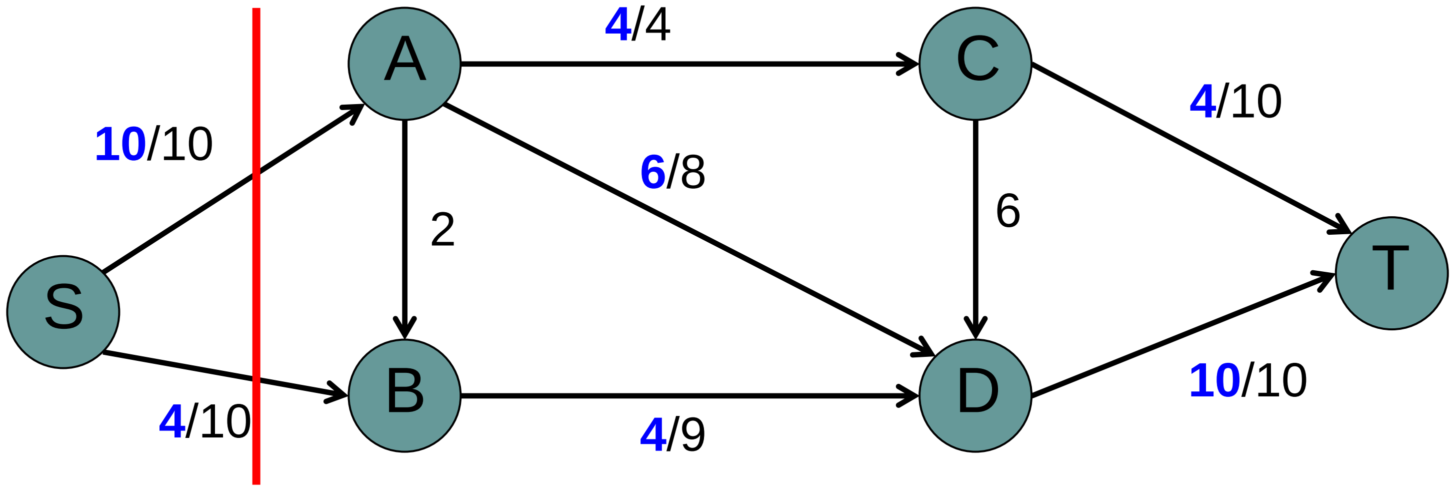
The flow across ANY such cut is the same and is the current flow in the network



Flow across cuts

Consider any cut where s $\in$ S_s and t $\in$ S_t, i.e. the cut partitions the source from the sink

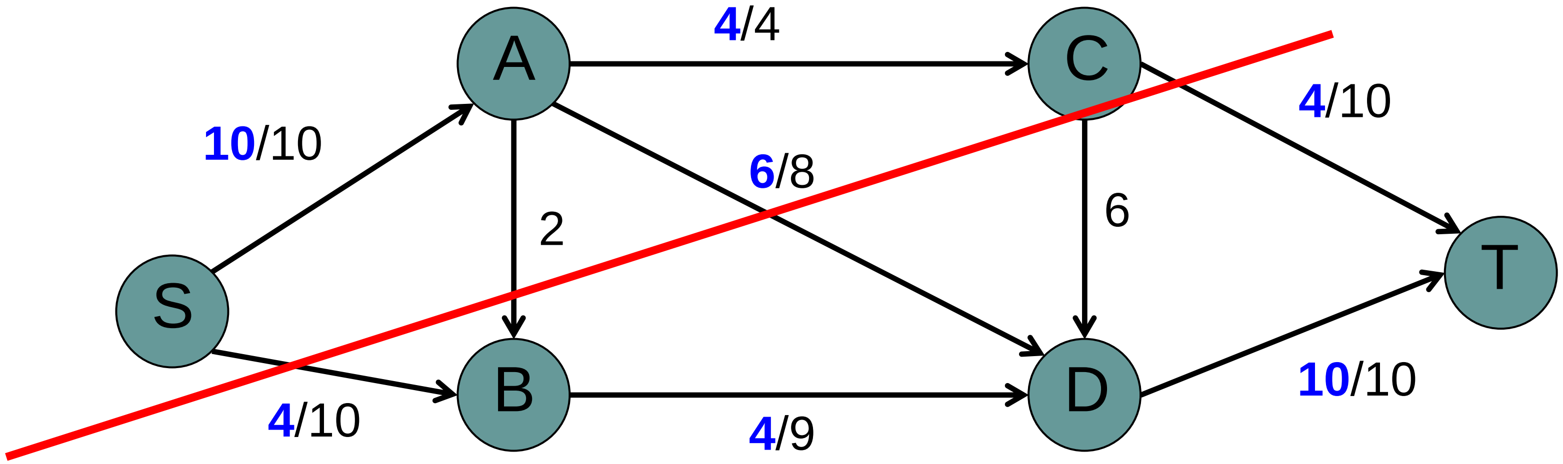
4+10 = 14



Flow across cuts

Consider any cut where $s \in S_s$ and $t \in S_t$, i.e. the cut partitions the source from the sink

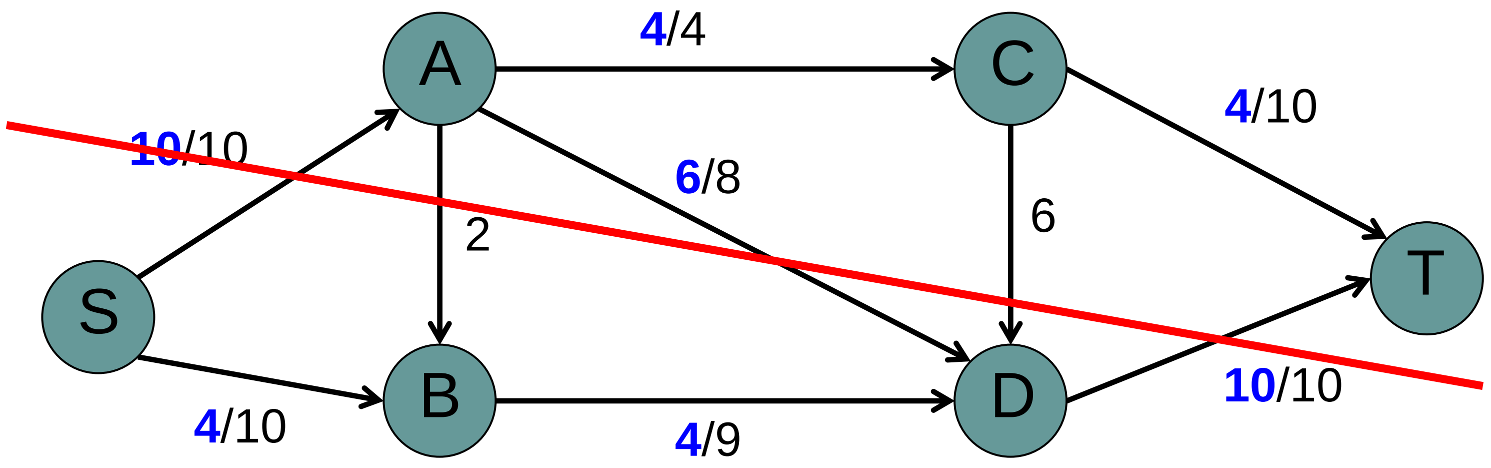
4+6+4 = 14



Flow across cuts

Consider any cut where $s \in S_s$ and $t \in S_t$, i.e. the cut partitions the source from the sink

10+10-6 = 14

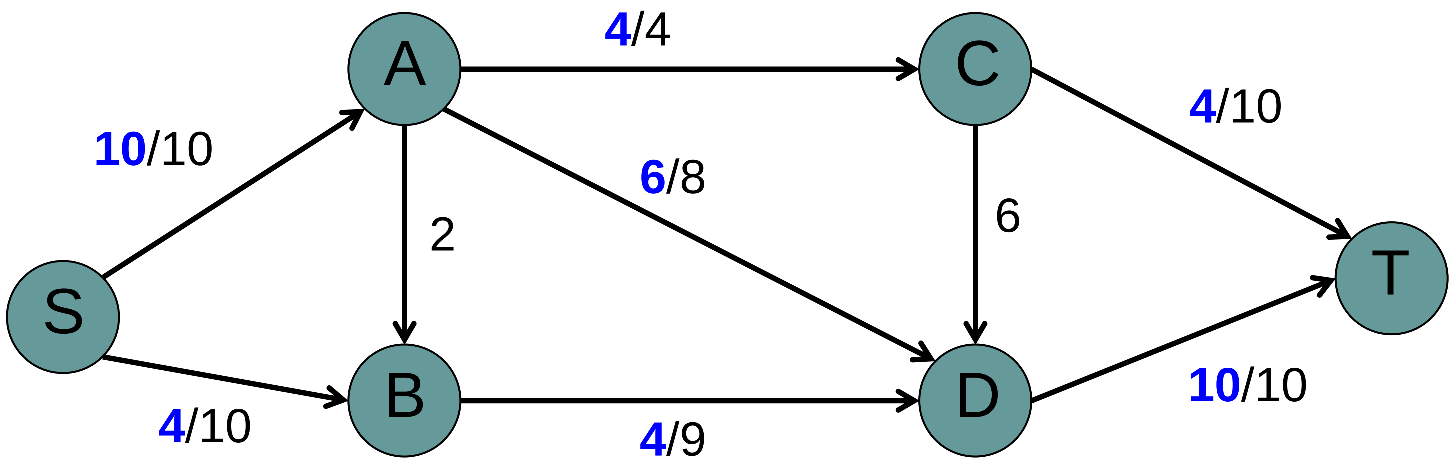


Flow across cuts

Consider any cut where s $\in S_s$ and t $\in S_t$, i.e. the cut partitions the source from the sink

The flow across ANY such cut is the same and is the current flow in the network

Why? Can you prove it?



Flow across cuts

The flow across ANY such cut is the same and is the current flow in the network

Inductively?

- every vertex is on a path from *s* to *t*
- in-flow = out-flow for every vertex (except s, t)
- flow along an edge cannot exceed the edge capacity
- flows are positive

Flow across cuts

The flow across ANY such cut is the same and is the current flow in the network

Base case: S_s = s

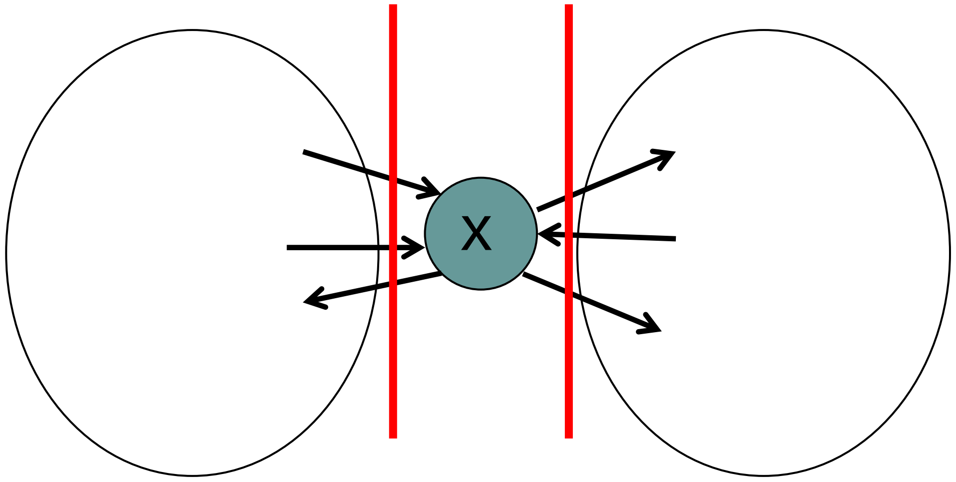
- Flow is total from from s to t: therefore the total flow out of s should be the flow
- All flow from s gets to t
  - every vertex is on a path from s to t
  - in-flow = out-flow

Flow across cuts

The flow across ANY such cut is the same and is the current flow in the network

Inductive case: Consider moving a node *x* from S_t to S_s

Is the flow across the different partitions the same?

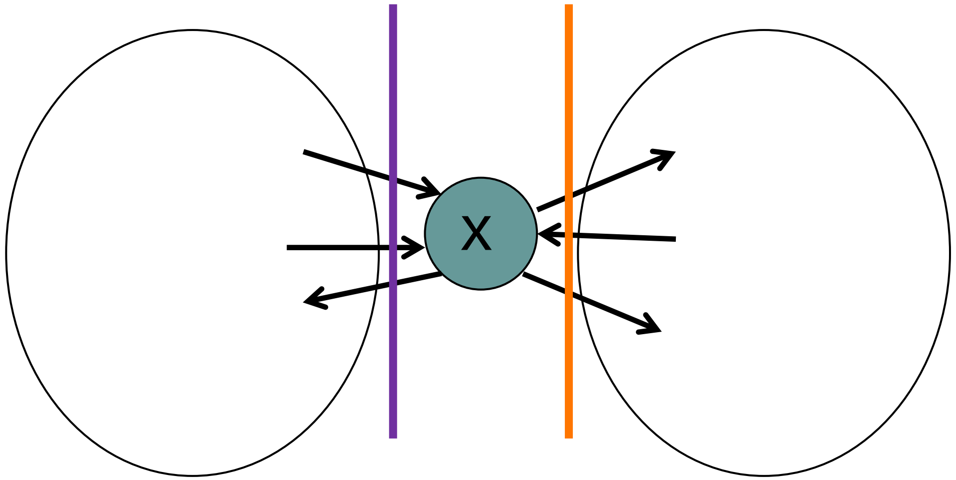


Flow across cuts

Inductive case: Consider moving a node *x* from S_t to S_s

cut = left-inflow(x) – left-outflow(x)

cut = right-outflow(x) – right-inflow(x)



left-inflow(x) + right-inflow(x) = left-outflow(x) + right-outflow(x)

in-flow = out-flow

left-inflow(x) - left-outflow(x) = right-outflow(x) – right-inflow(x)

Flow across cuts

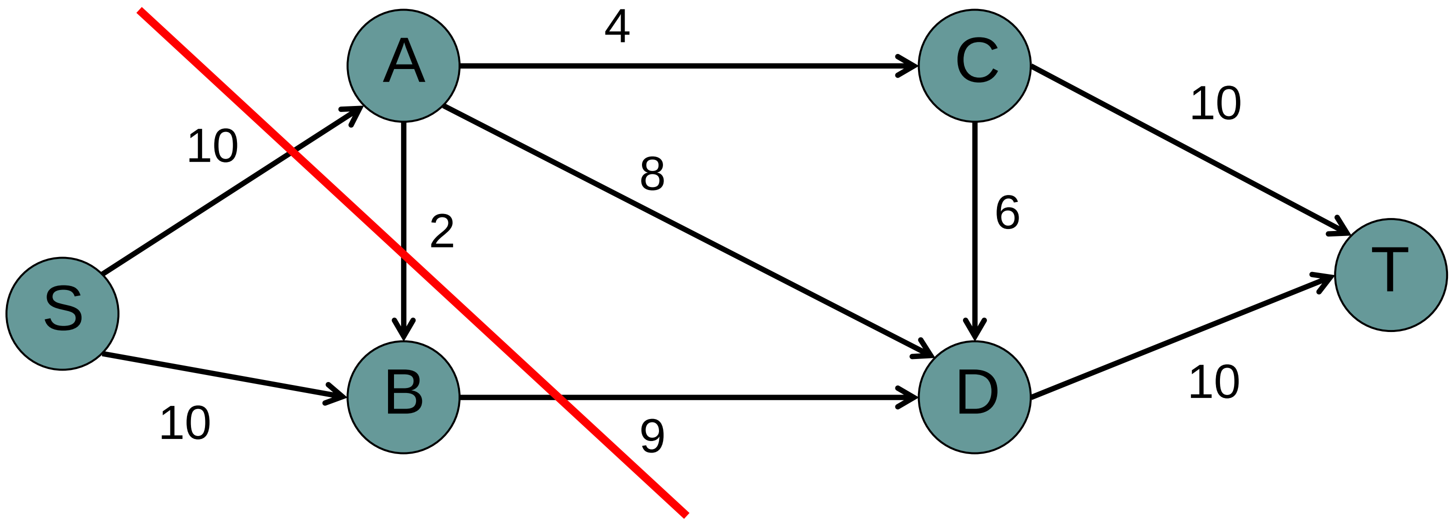
Consider any cut where $s \in S_s$ and $t \in S_t$, i.e. the cut partitions the source from the sink

The flow across ANY such cut is the same and is the current flow in the network

Capacity of a cut

The “**capacity of a cut**” is the maximum flow that we *could* send from nodes in S_s to nodes in S_t (i.e. across the cut)

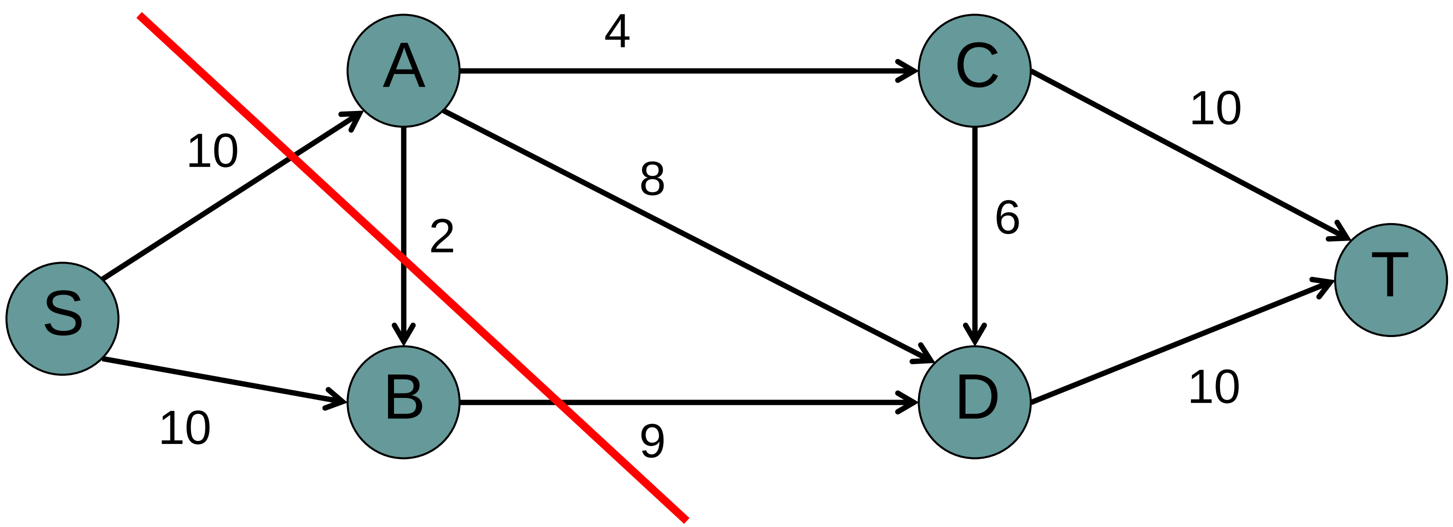
How do we calculate the capacity?



Capacity of a cut

The "**capacity of a cut**" is the maximum flow that we *could* send from nodes in S_s to nodes in S_t (i.e. across the cut)

Capacity is the sum of the edges from S_s to S_t



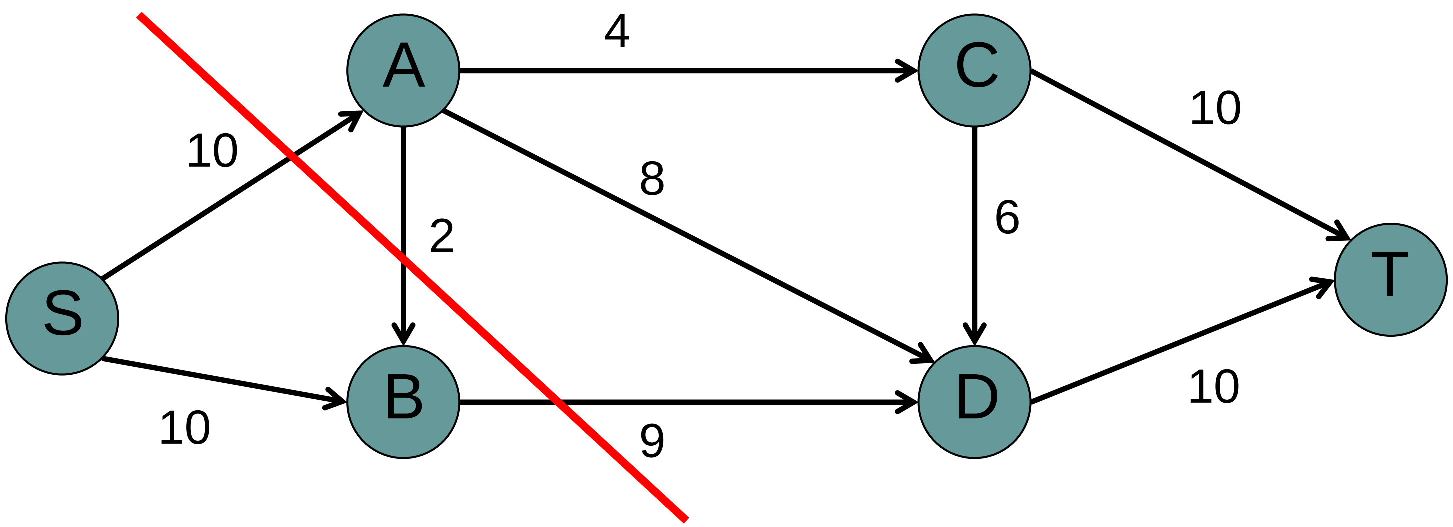
10 + 9 = 19

Capacity of a cut

The “**capacity of a cut**” is the maximum flow that we *could* send from nodes in S_s to nodes in S_t (i.e. across the cut)

Capacity is the sum of the edges from S_s to S_t

Why?



Capacity of a cut

The "**capacity of a cut**" is the maximum flow that we *could* send from nodes in S_s to nodes in S_t (i.e. across the cut)

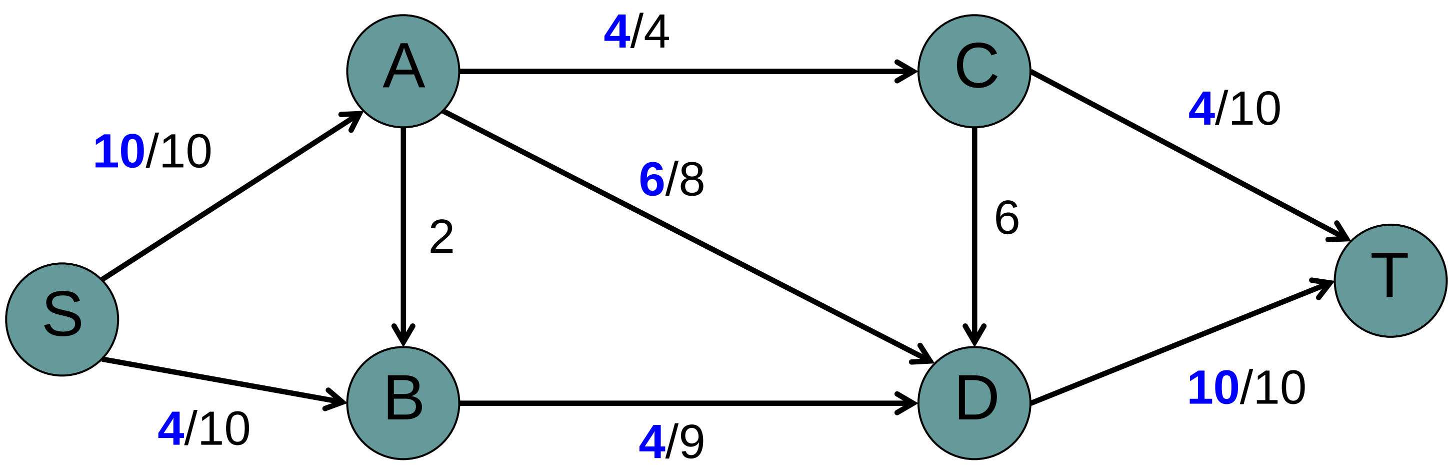
Capacity is the sum of the edges from S_s to S_t

- Any more and we would violate the edge capacity constraint
- Any less and it would not be maximal, since we could simply increase the flow

Maximum flow

For any cut where $s \in S_s$ and $t \in S_t$

- the flow across the cut is the same
- the maximum capacity (i.e. flow) across the cut is the sum of the capacities for edges from S_s to S_t



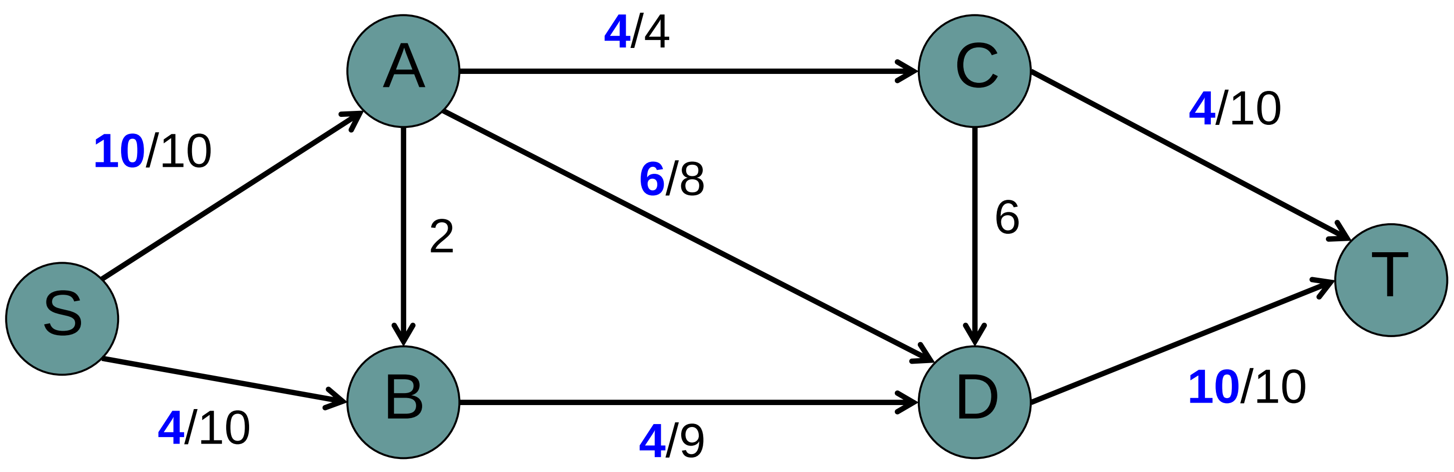
Are we done?

Is this the best we can do?

Maximum flow

For any cut where $s \in S_s$ and $t \in S_t$

- the flow across the cut is the same
- the maximum capacity (i.e. flow) across the cut is the sum of the capacities for edges from S_s to S_t

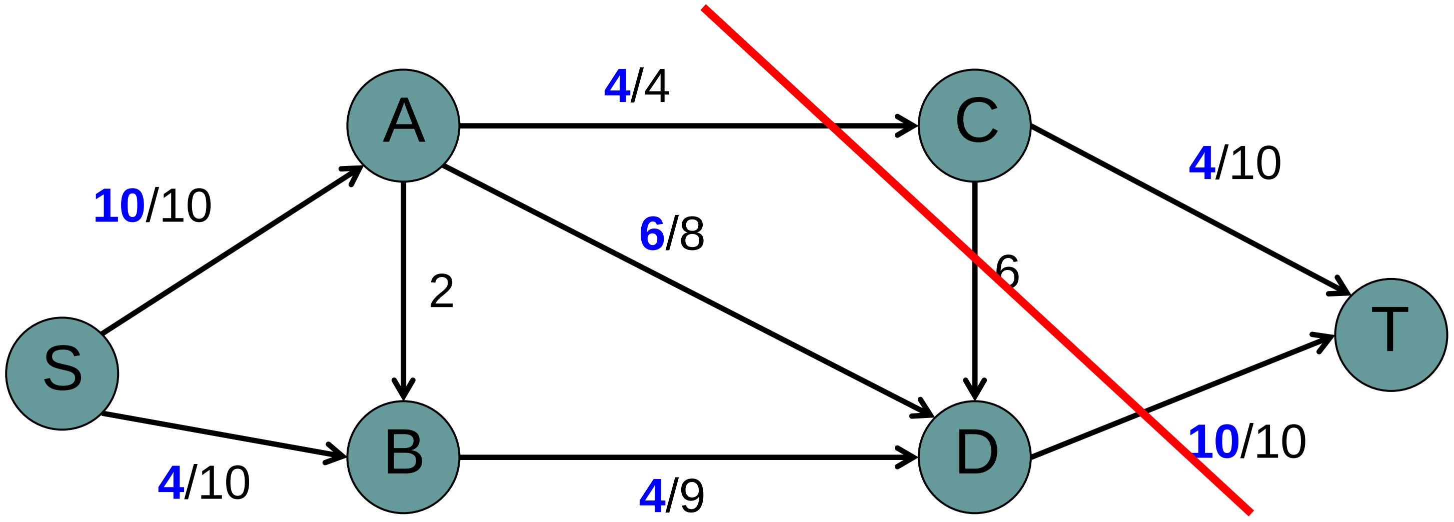


We can do no better than the minimum capacity cut!

Maximum flow

What is the minimum capacity cut for this graph?

Capacity = 10 + 4

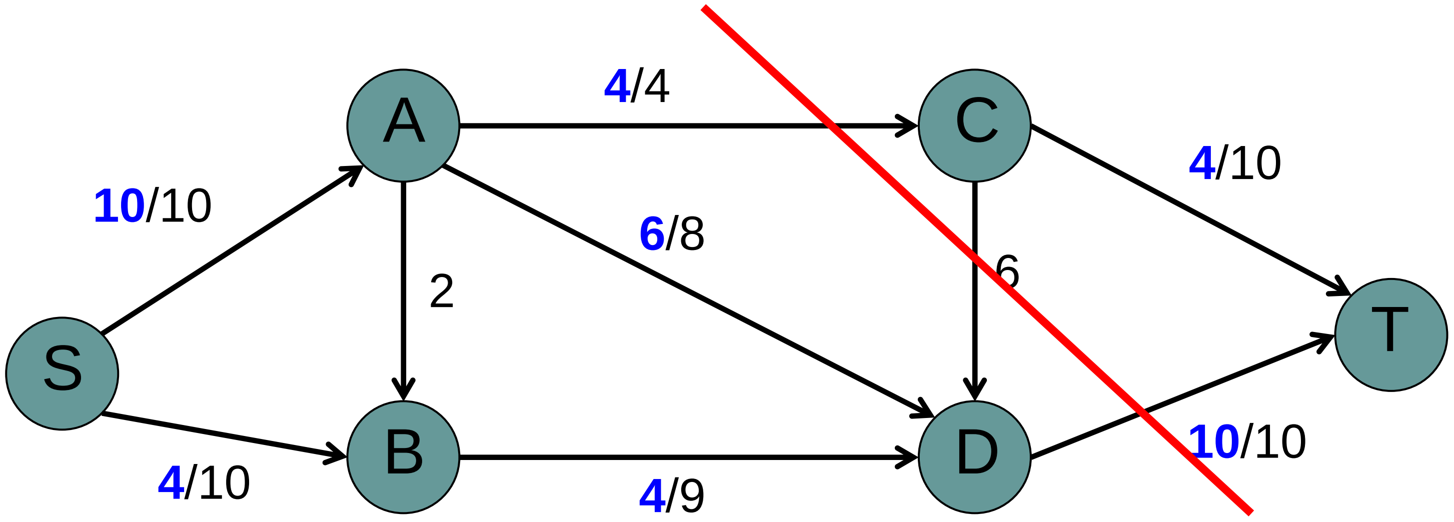


Is this the best we can do?

Maximum flow

What is the minimum capacity cut for this graph?

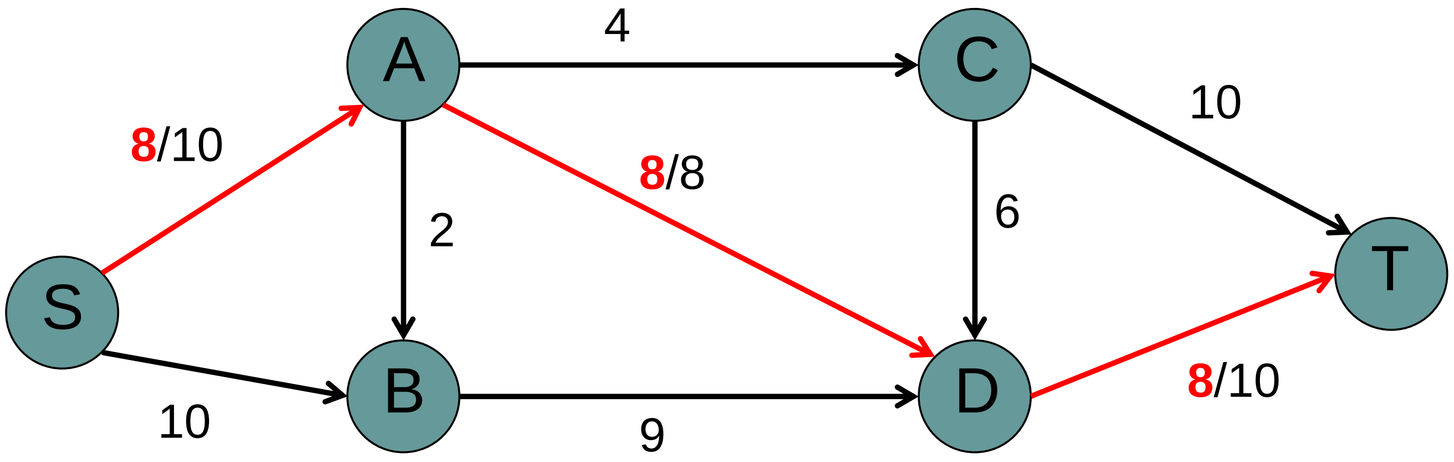
Capacity = 10 + 4



flow = minimum capacity, so we can do no better

Algorithm idea

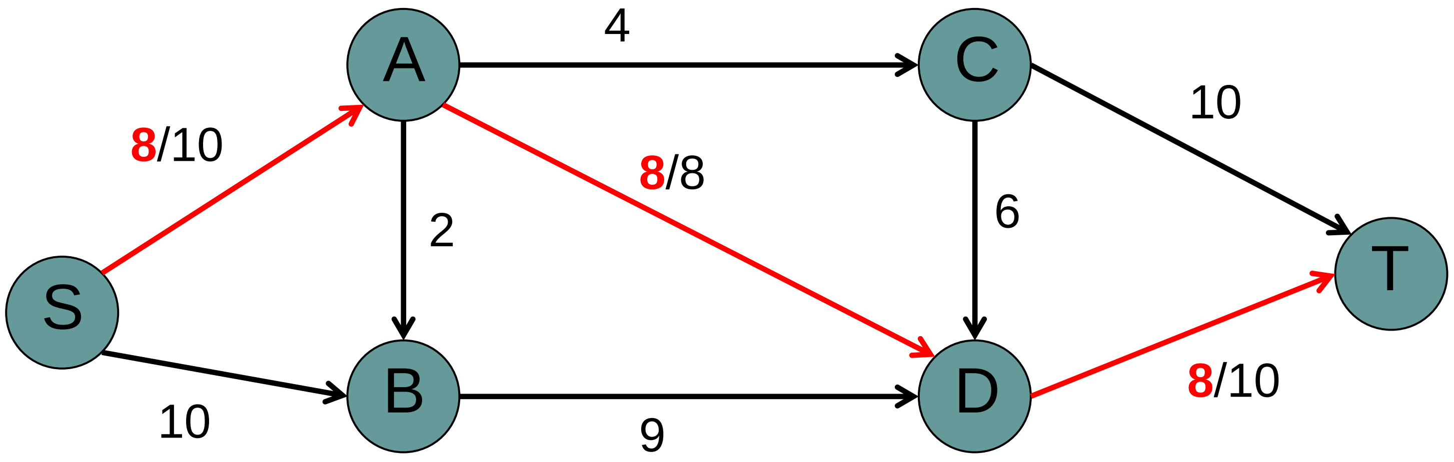
send some flow down a path



How do we determine the path to send flow down?

Algorithm idea

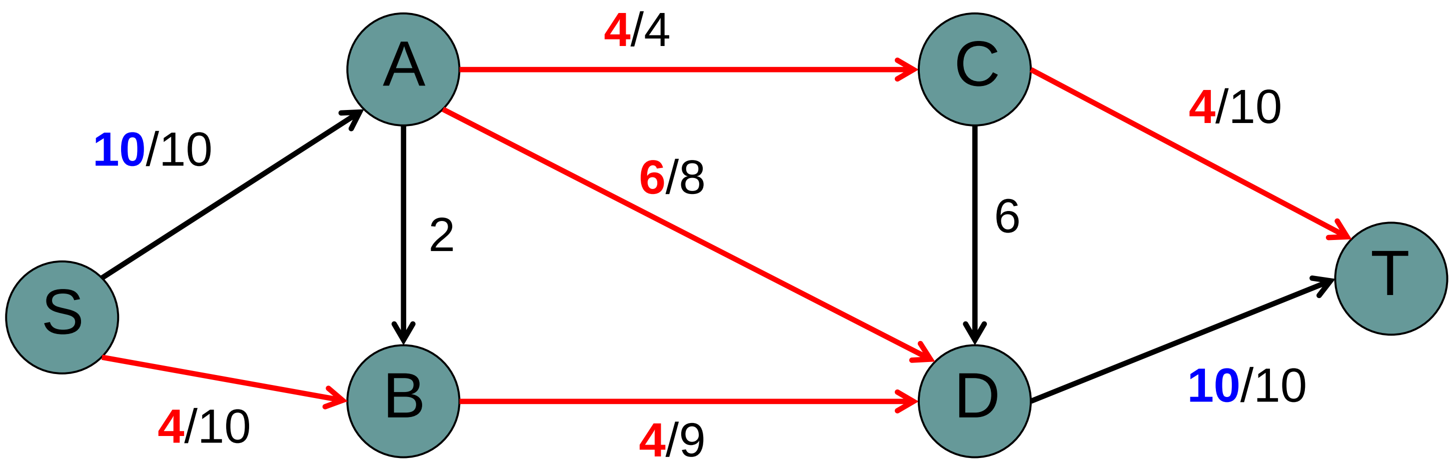
send some flow down a path



Search for a path with remaining capacity from s to t

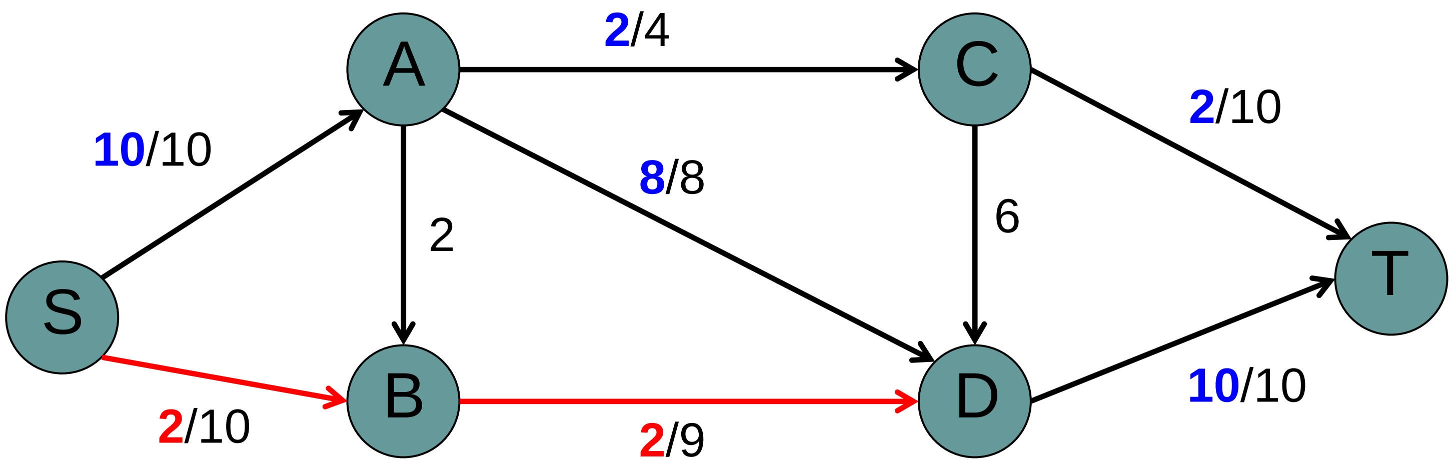
Algorithm idea

reroute some of the flow



How do we handle “rerouting” flow?

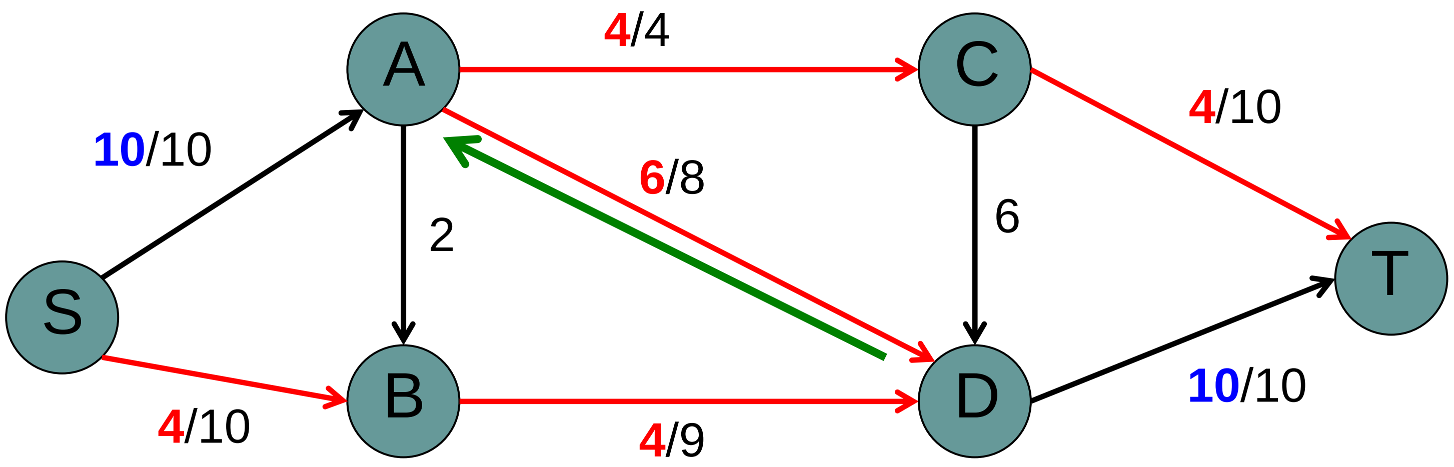
Algorithm idea



During the search, if an edge has some flow, we consider “reversing” some of that flow

Algorithm idea

reroute some of the flow



During the search, if an edge has some flow, we consider "reversing" some of that flow

The residual graph

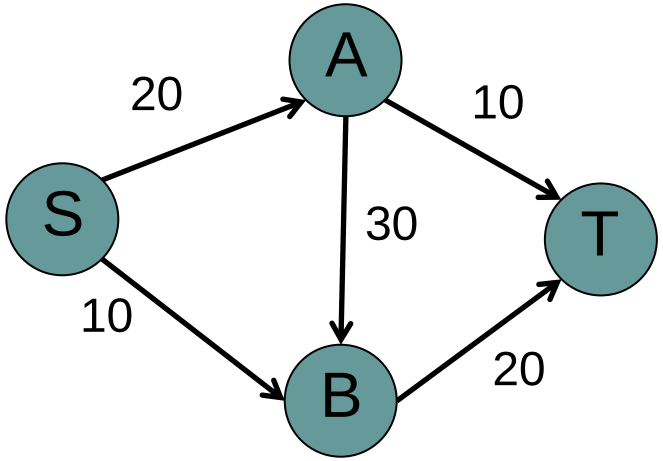
The *residual graph* G_f is constructed from G

For each edge *e* in the original graph (G):

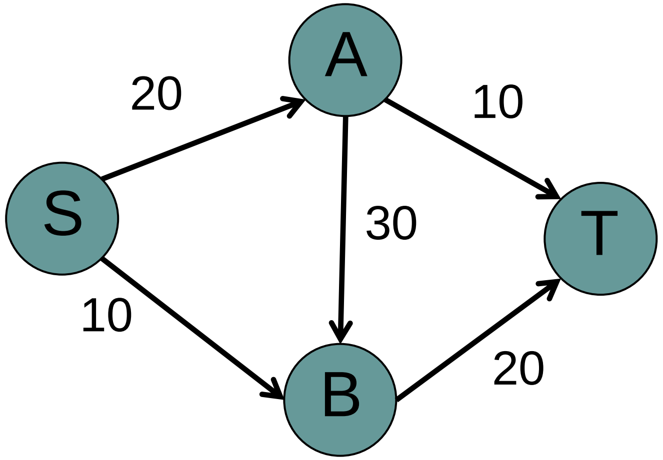
- if *flow(e)* < capacity(e)
  - introduce an edge in G_f with capacity = capacity(e)-flow(e)
  - this represents the remaining flow we can still push
- if flow(e) > 0
  - introduce an edge in G_f in the *opposite direction* with capacity = flow(e)
  - this represents the flow that we can reroute/reverse

Algorithm idea

G



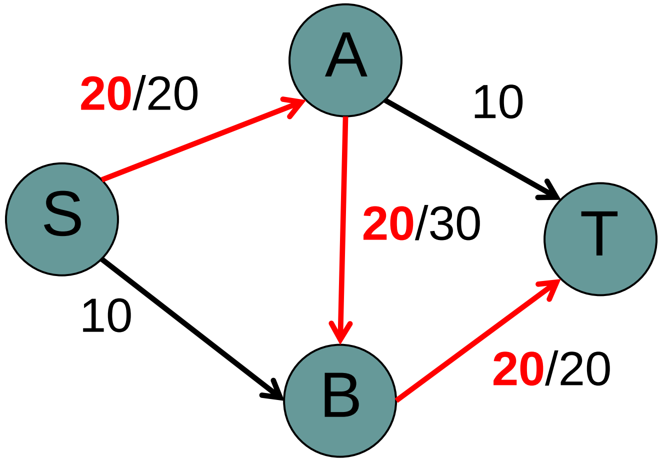
G_f



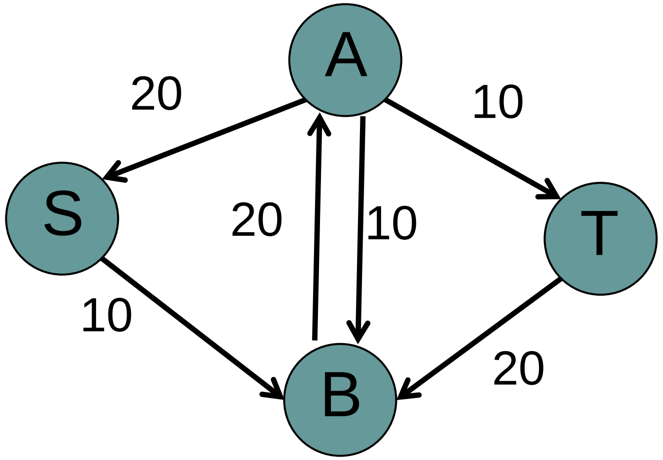
Find a path from s to t in G_f

Algorithm idea

G



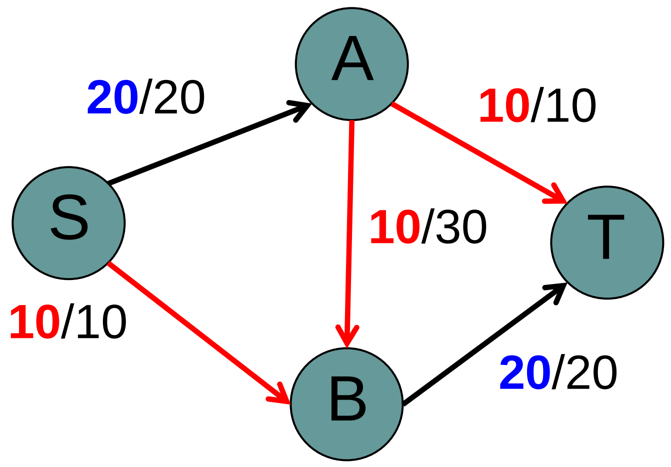
G_f



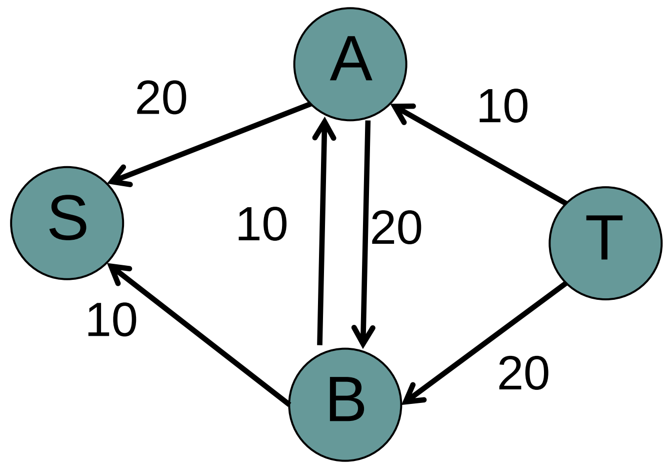
Find a path from s to t in G_f

Algorithm idea

G



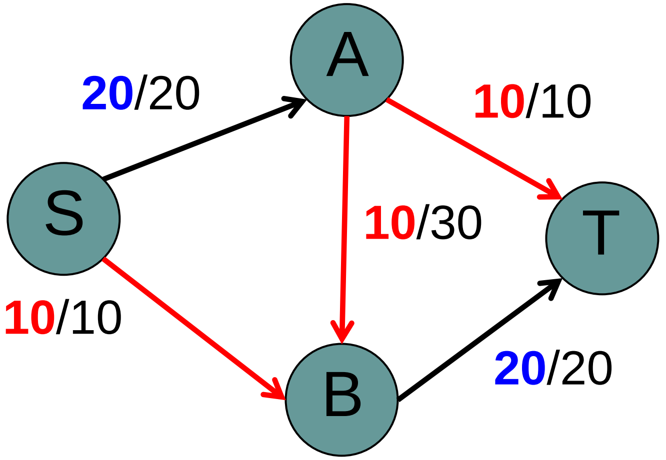
G_f



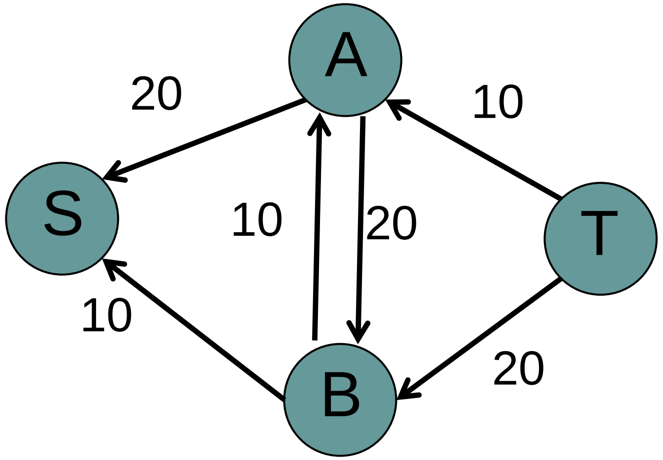
Find a path from s to t in G_f

Algorithm idea

G

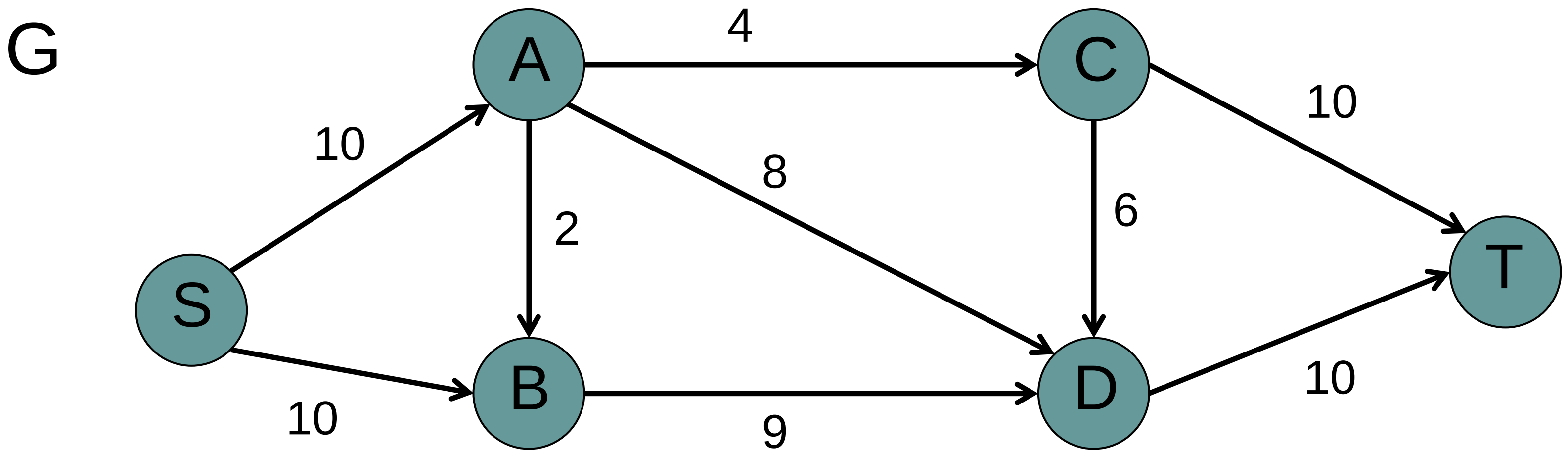


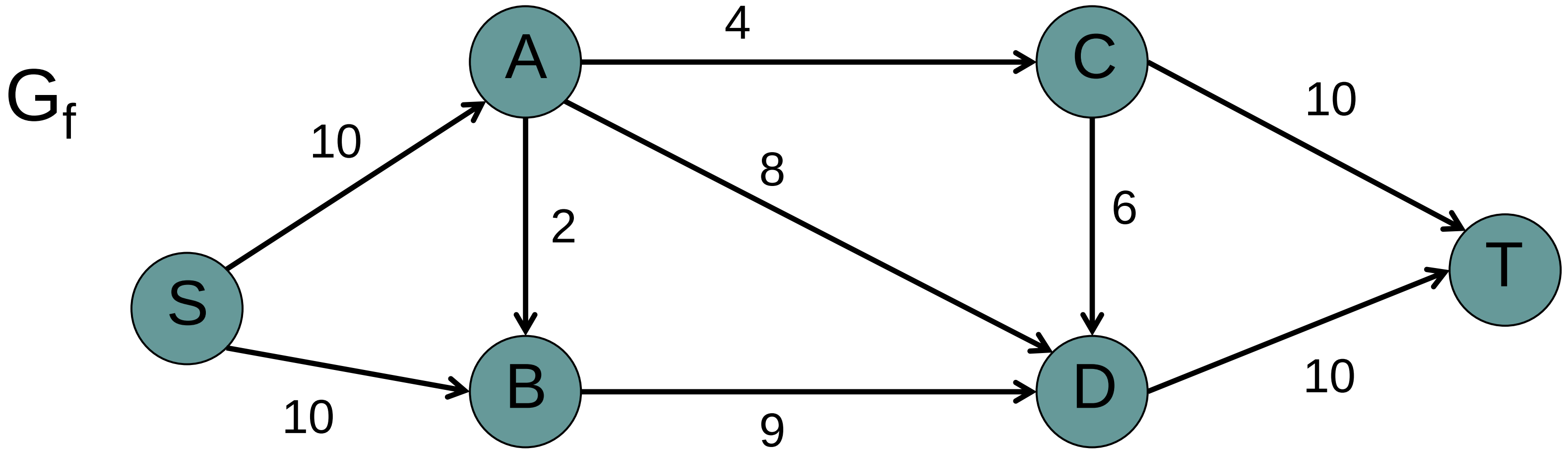
G_f



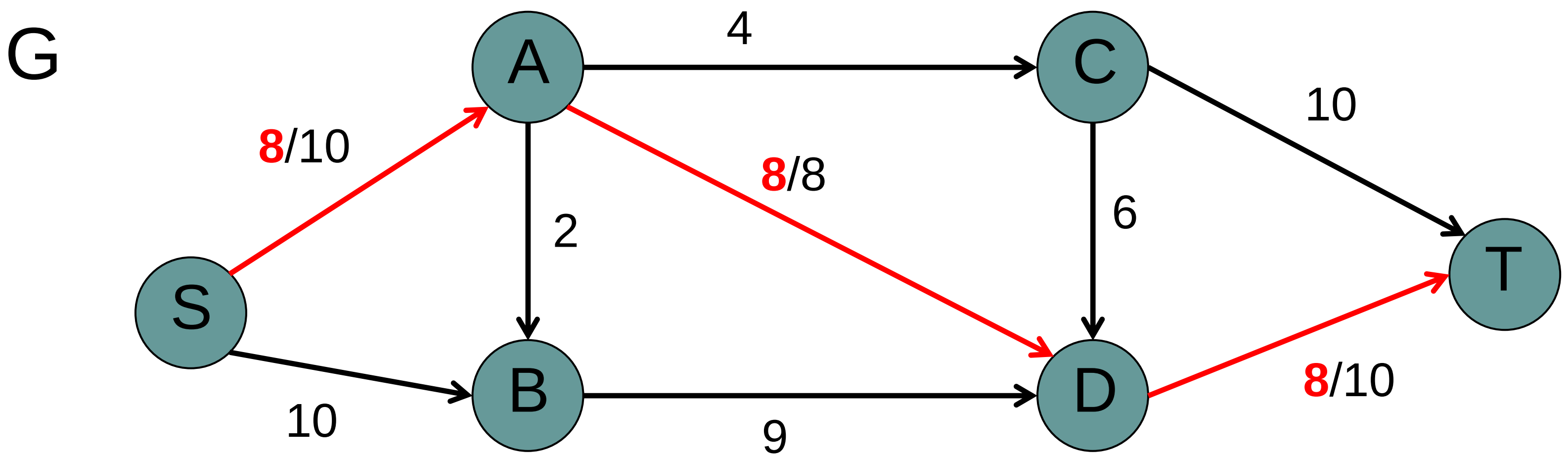
None exist… done!

Algorithm idea

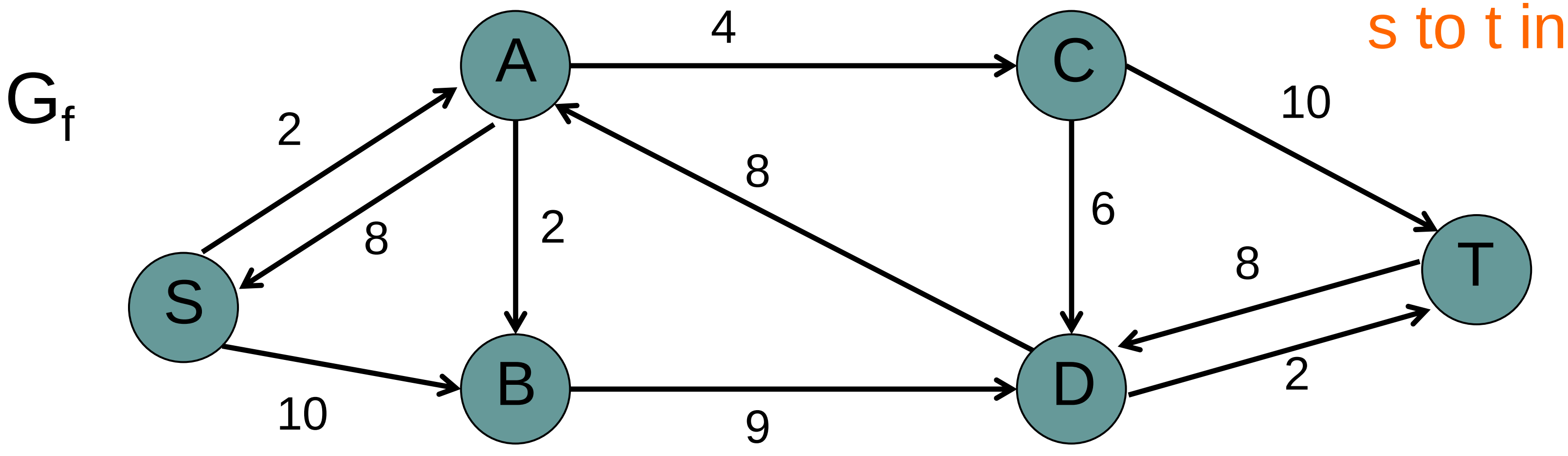




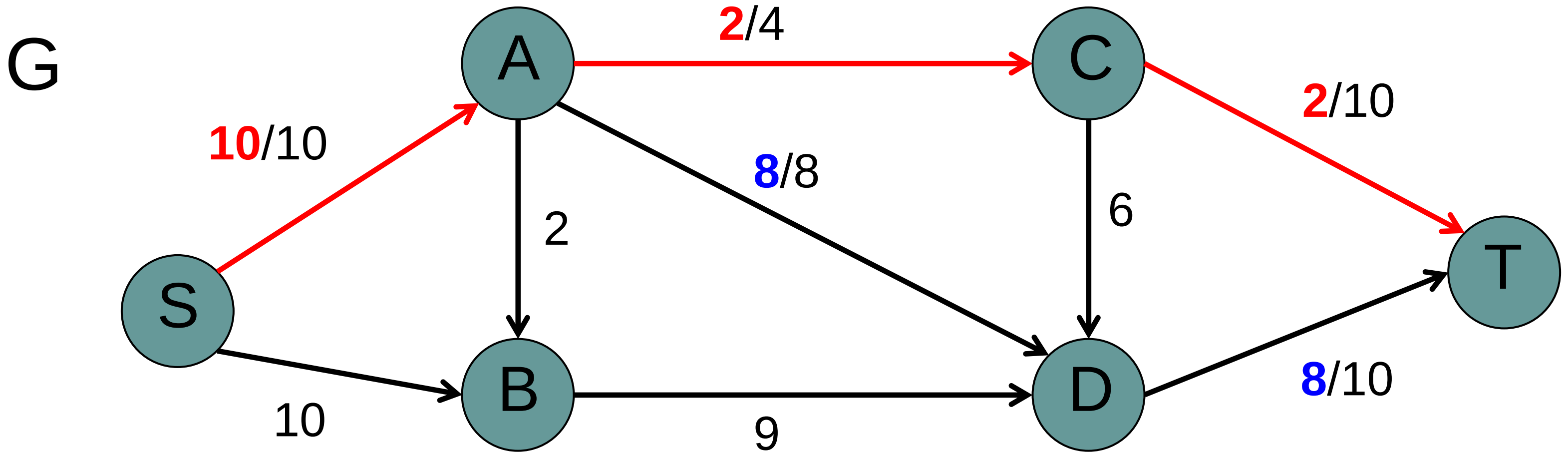
Algorithm idea

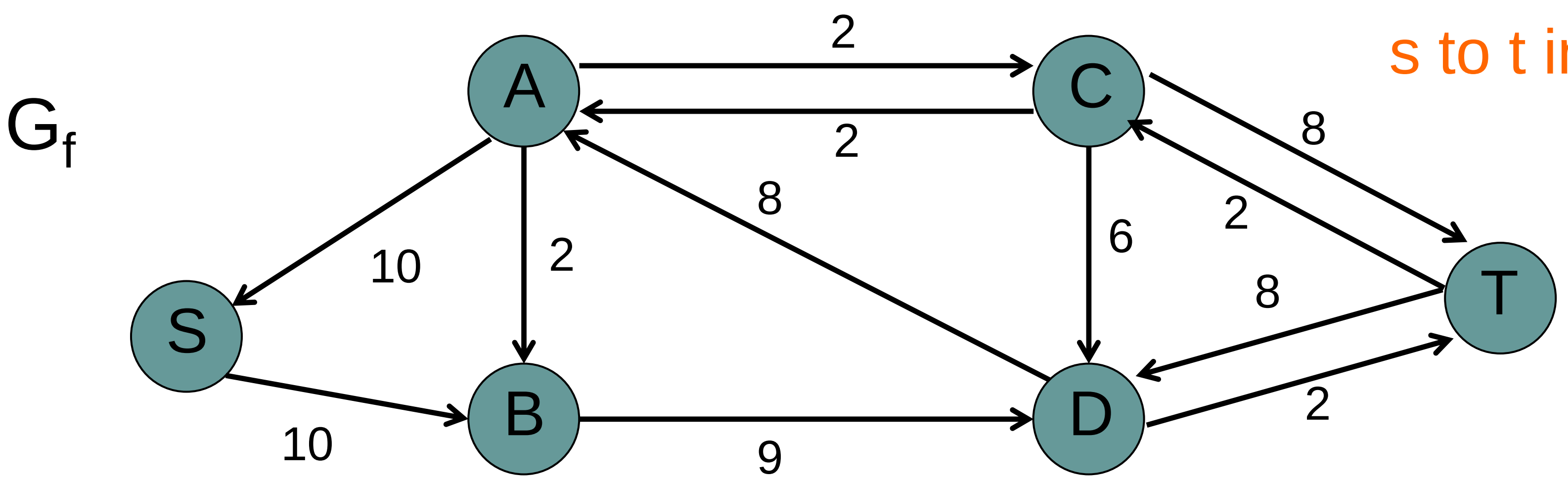


Find a path from s to t in G_f



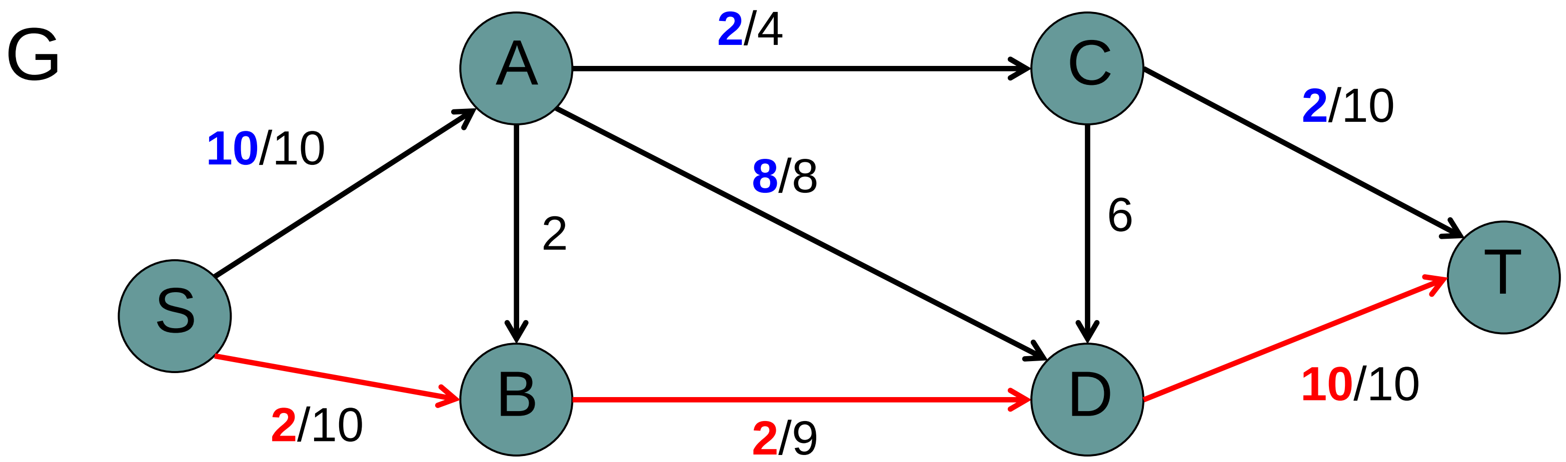
Algorithm idea



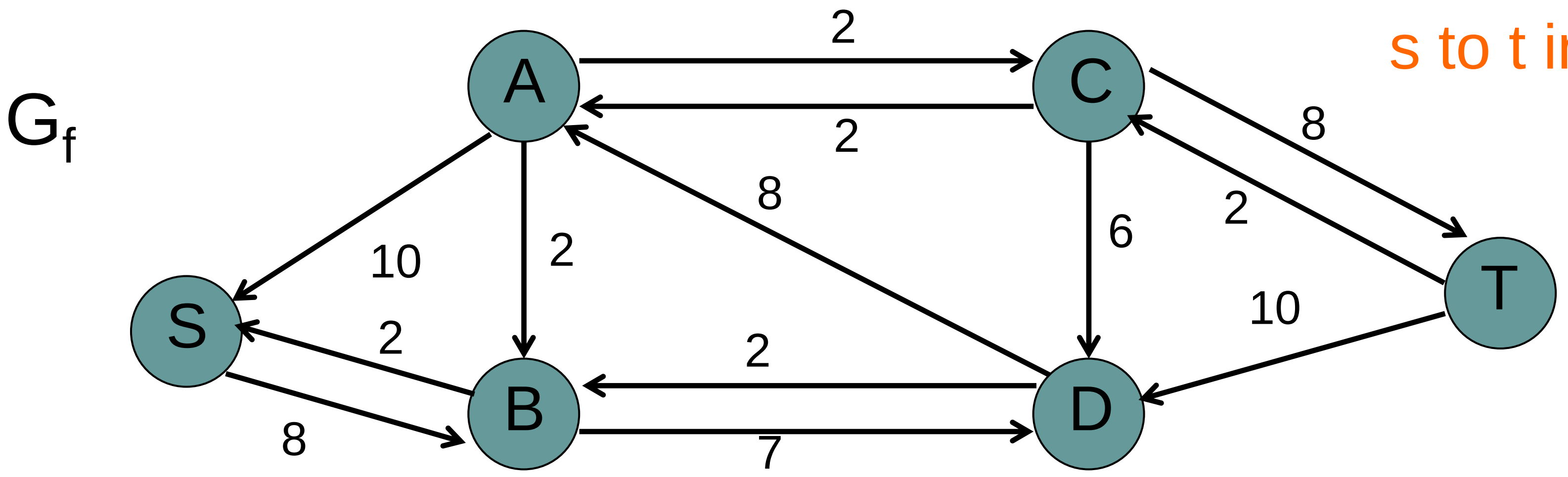


Find a path from s to t in G_f

Algorithm idea

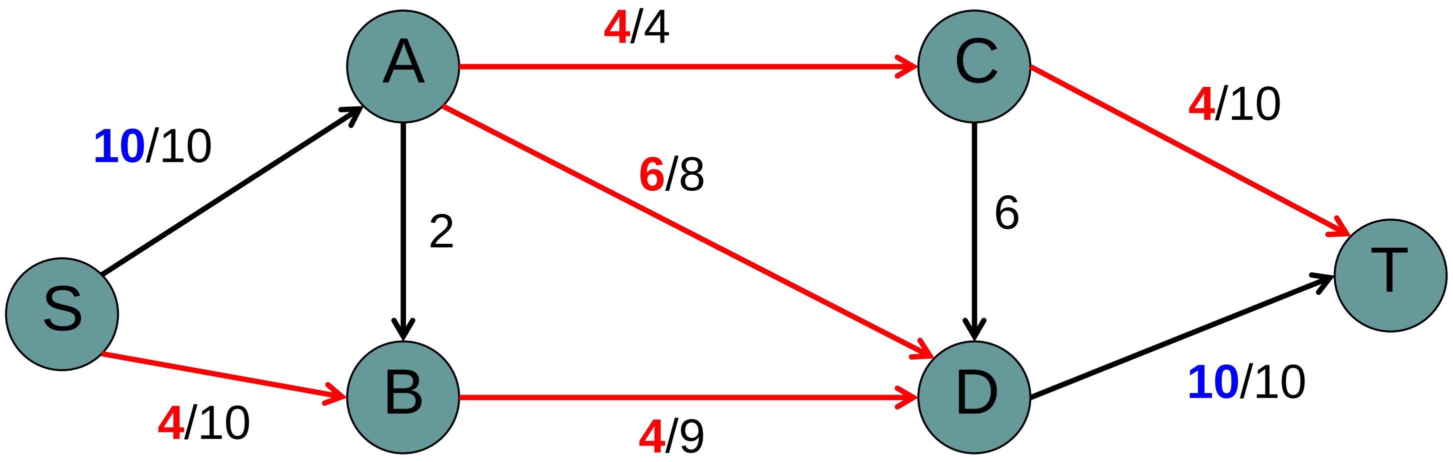


Find a path from s to t in G_f



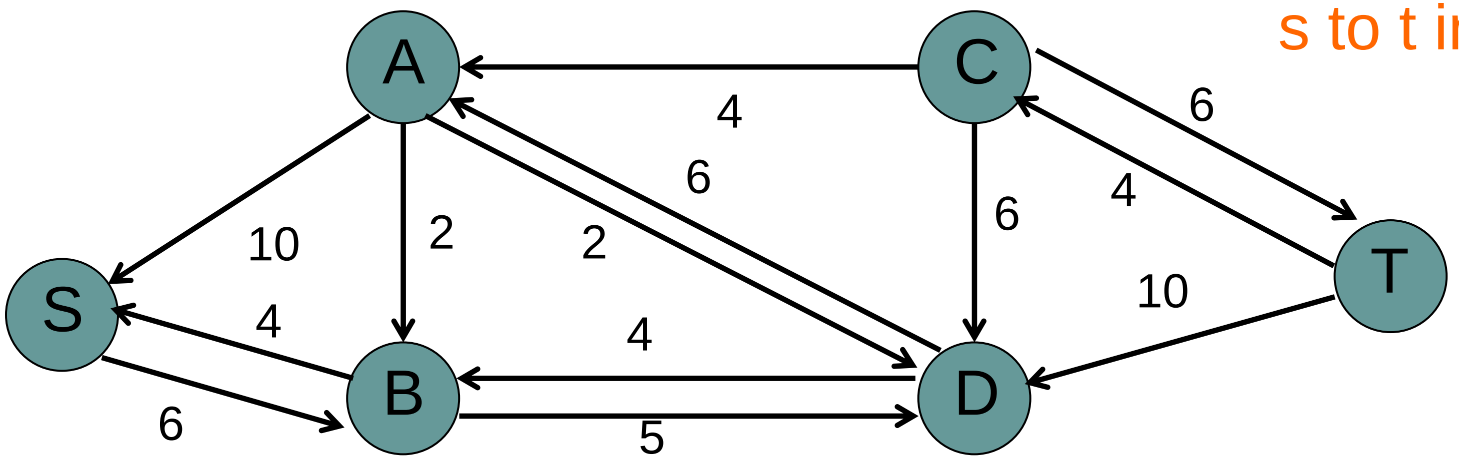
Algorithm idea

G

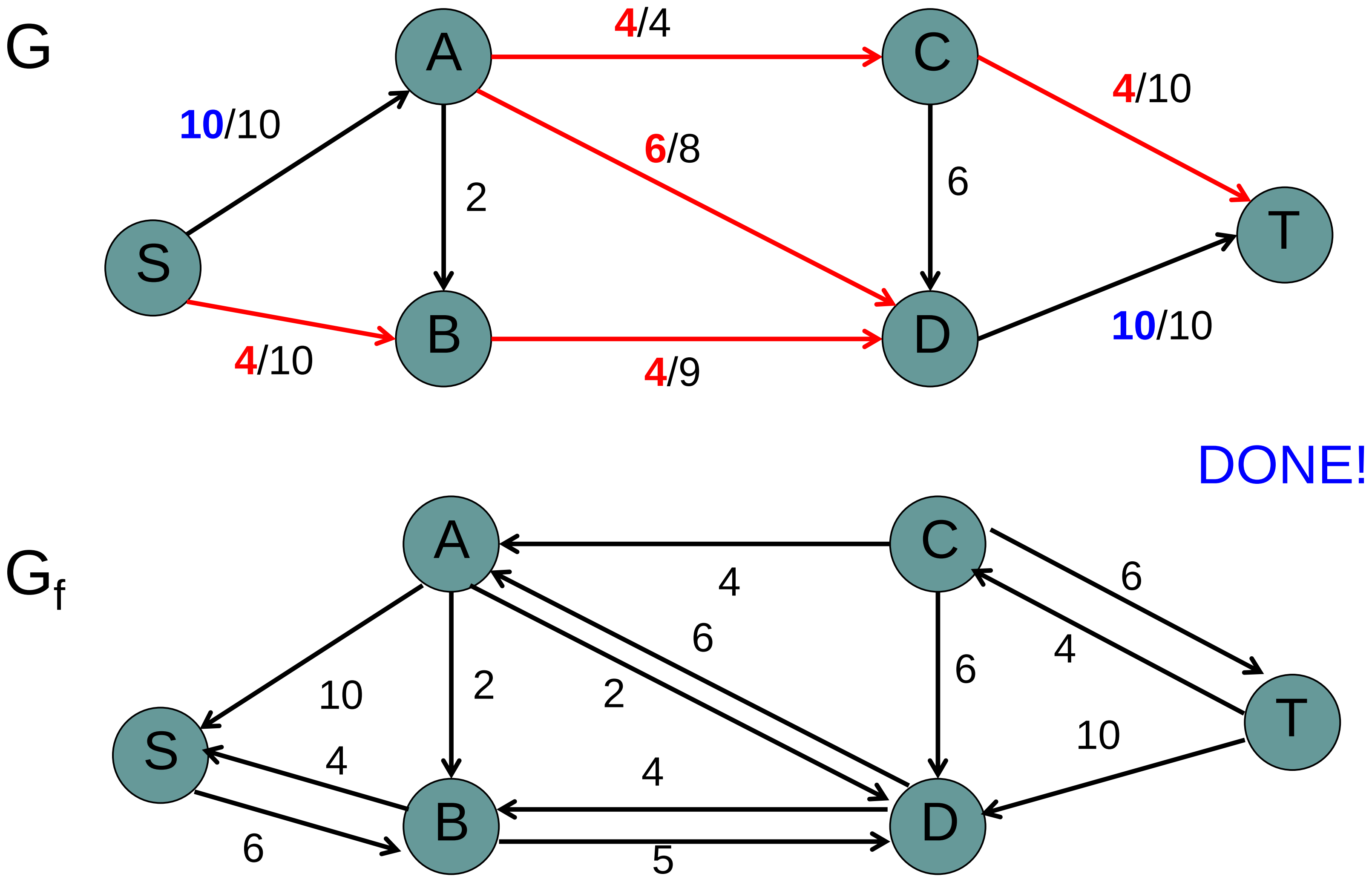


Find a path from s to t in G_f

G_f



Algorithm idea



Ford-Fulkerson

Ford-Fulkerson(G, s, t)

flow = 0 for all edges

G_f = residualGraph(G)

while a simple path exists from s to t in G_f

send as much flow along the path as possible

G_f = residualGraph(G)

return flow

a simple path contains no repeated vertices

Ford-Fulkerson: is it correct?

Does the function terminate?

Every iteration increases the flow from s to t

- Every path must start with s
- The path has positive flow (or it wouldn't exist)
- The path is a simple path (so it cannot revisit s)
- conservation of flow

```
Ford-Fulkerson(G, s, t)
  flow = 0 for all edges
  G_f = residualGraph(G)
  while a simple path exists from s to t in G_f
    send as much flow along path as possible
    G_f = residualGraph(G)
  return flow
```

Ford-Fulkerson: is it correct?

Does the function terminate?

- Every iteration increases the flow from s to t
- the flow is bounded by the min-cut

```
Ford-Fulkerson(G, s, t)
   flow = 0 for all edges
   G_f = residualGraph(G)
   while a simple path exists from s to t in G_f
      send as much flow along path as possible
      G_f = residualGraph(G)
   return flow
```

Ford-Fulkerson: is it correct?

When it terminates is it the maximum flow?

Ford-Fulkerson(G, s, t)
  flow = 0 for all edges
  G_f = residualGraph(G)
  while a simple path exists from s to t in G_f
    send as much flow along path as possible
    G_f = residualGraph(G)
  return flow

Ford-Fulkerson: is it correct?

When it terminates is it the maximum flow?

Assume it didn't

- We know then that the flow < min-cut
- therefore, the flow < capacity across EVERY cut
- therefore, across each cut there must be a forward edge in G_f
- thus, there must exist a path from s to t in G_f
  - start at s (and A = s)
  - repeat until t is found
    - pick one node across the cut with a forward edge
    - add this to the path
    - add the node to A (for argument sake)
- However, the algorithm would not have terminated… a contradiction

Ford-Fulkerson: runtime?

```
Ford-Fulkerson(G, s, t)
  flow = 0 for all edges
  Gf = residualGraph(G)
  while a simple path exists from s to t in Gf
    send as much flow along path as possible
    Gf = residualGraph(G)
  return flow
```

Ford-Fulkerson: runtime?

```
Ford-Fulkerson(G, s, t)
   flow = 0 for all edges
   Gf = residualGraph(G)
   while a simple path exists from s to t in Gf
      send as much flow along path as possible
      Gf = residualGraph(G)
   return flow
```

- traverse the graph
- at most add 2 edges for original edge
- $\boxed{\theta(V + E)}$

Can we simplify this expression?

Ford-Fulkerson: runtime?

```
Ford-Fulkerson(G, s, t)
   flow = 0 for all edges
   Gf = residualGraph(G)
   while a simple path exists from s to t in Gf
      send as much flow along path as possible
      Gf = residualGraph(G)
   return flow
```

- traverse the graph
- at most add 2 edges for original edge
- $\theta(V + E) = \theta(E)$
- (all nodes exists on paths from s to t)

Ford-Fulkerson: runtime?

Ford-Fulkerson(G, s, t)
    flow = 0 for all edges
    G_f = residualGraph(G)
    $\boxed{\text{while}}$ a simple path exists from s to t in G_f
        send as much flow along path as possible
        G_f = residualGraph(G)
    return flow

- BFS or DFS
- $O(V + E) = O(E)$

Ford-Fulkerson: runtime?

Ford-Fulkerson(G, s, t)
  flow = 0 for all edges
  G_f = residualGraph(G)
  while a simple path exists from s to t in G_f
    send as much flow along path as possible
    G_f = residualGraph(G)
  return flow

- max-flow!
- increases ever iteration
- integer capacities, so integer increases

Can we bound the number of times the loop will execute?

Ford-Fulkerson: runtime?

Ford-Fulkerson(G, s, t)
  flow = 0 for all edges
  G_f = residualGraph(G)
  while a simple path exists from s to t in G_f
    send as much flow along path as possible
    G_f = residualGraph(G)
  return flow

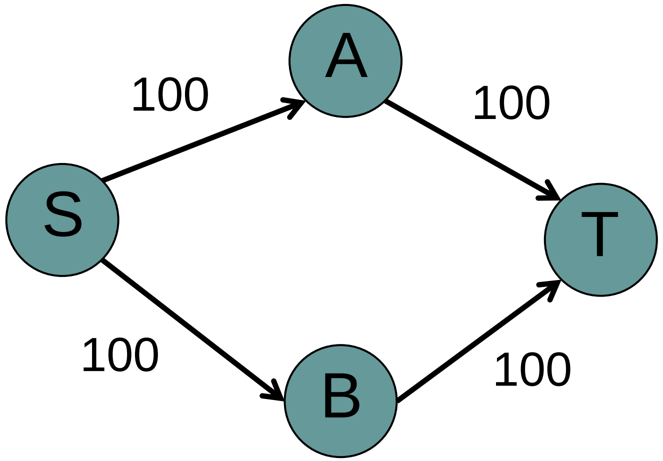
- max-flow!
- increases ever iteration
- integer capacities, so integer increases

Overall runtime?   O(max-flow * E)

O(max-flow * E)

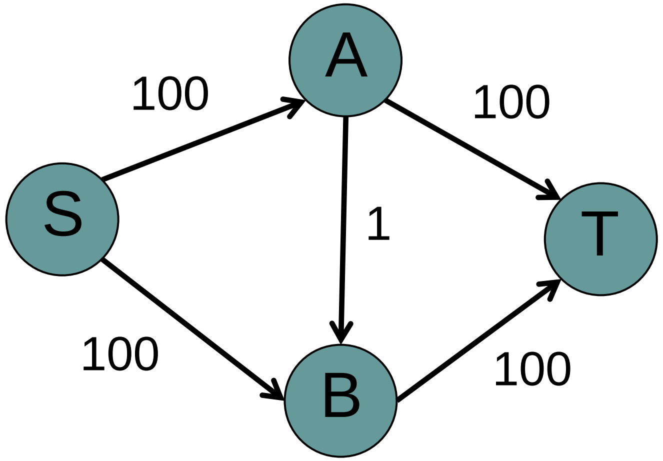
Can you construct a graph that *could get* this running time?

Hint:



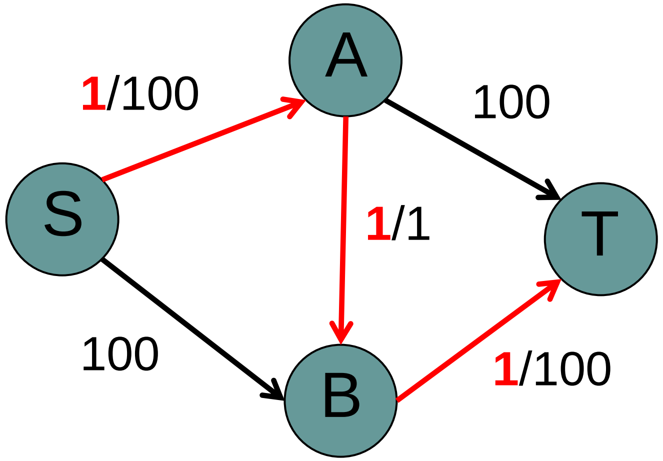
O(max-flow * E)

Can you construct a graph that *could get* this running time?



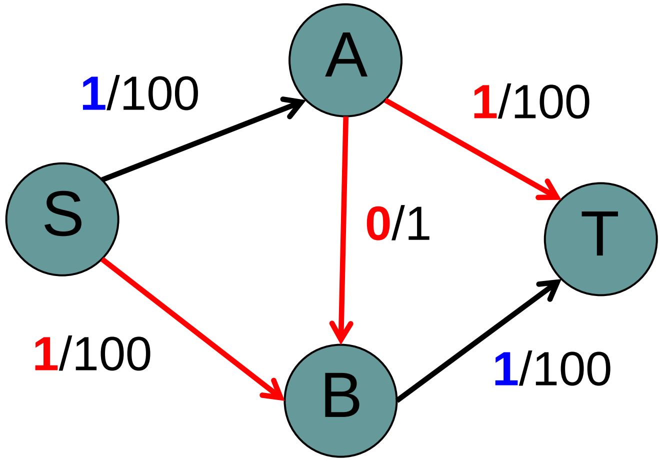
O(max-flow * E)

Can you construct a graph that *could get* this running time?



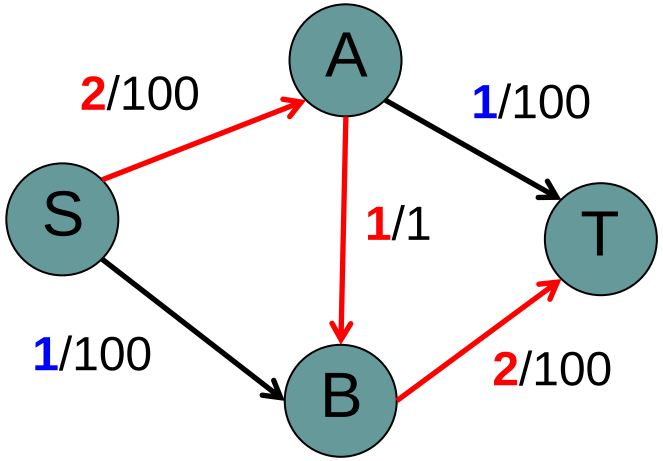
O(max-flow * E)

Can you construct a graph that *could get* this running time?



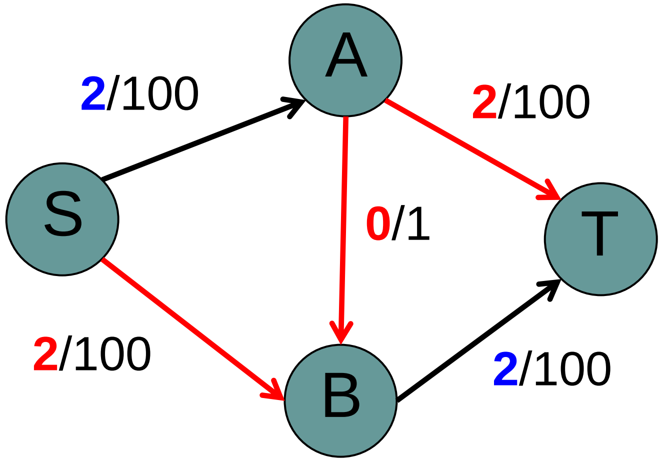
O(max-flow * E)

Can you construct a graph that *could get* this running time?



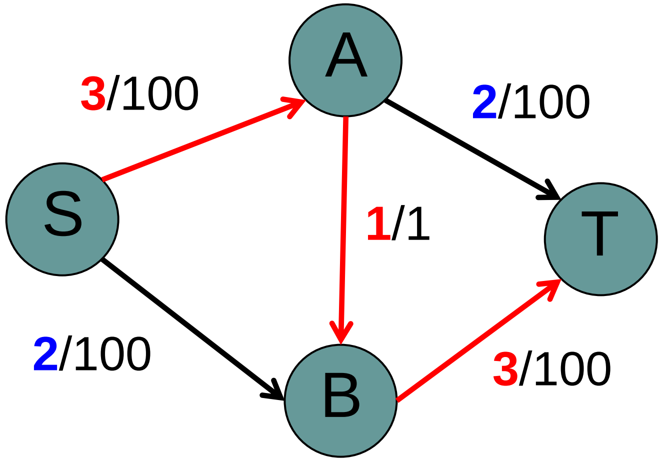
O(max-flow * E)

Can you construct a graph that *could get* this running time?



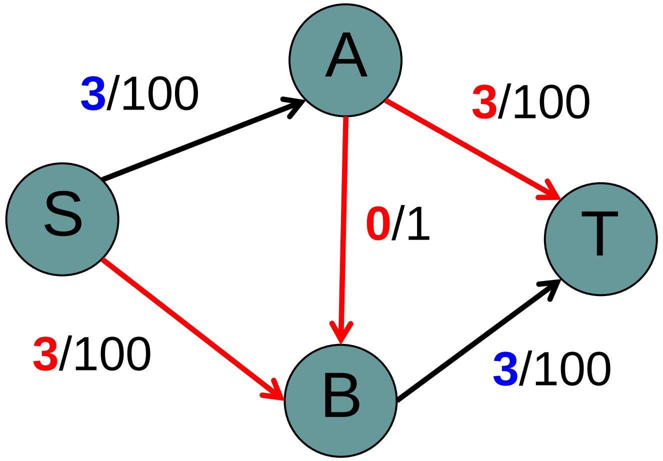
O(max-flow * E)

Can you construct a graph that *could get* this running time?



O(max-flow * E)

Can you construct a graph that *could get* this running time?



What is the problem here?
Could we do better?

Faster variants

Edmunds-Karp

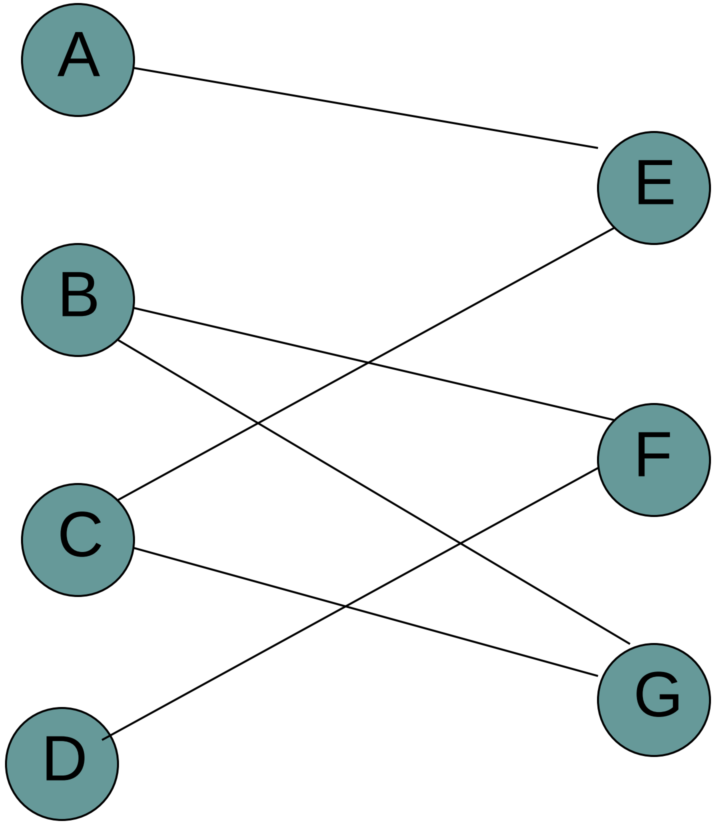
- Select the *shortest path* (in number of edges) from s to t in G_f
  - How can we do this?
    - use BFS for search
- Running time: $O(V E^2)$
  - avoids issues like the one we just saw

preflow-push (aka push-relabel) algorithms

- $O(V^3)$

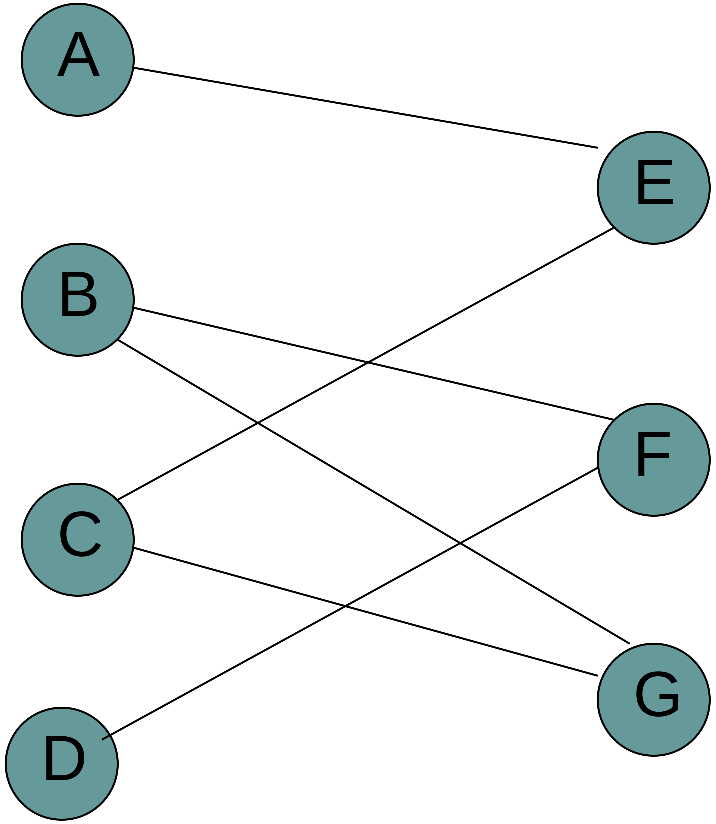
Application: bipartite graph matching

Bipartite graph – a graph where every vertex can be partitioned into two sets X and Y such that all edges connect a vertex $u \in X$ and a vertex $v \in Y$



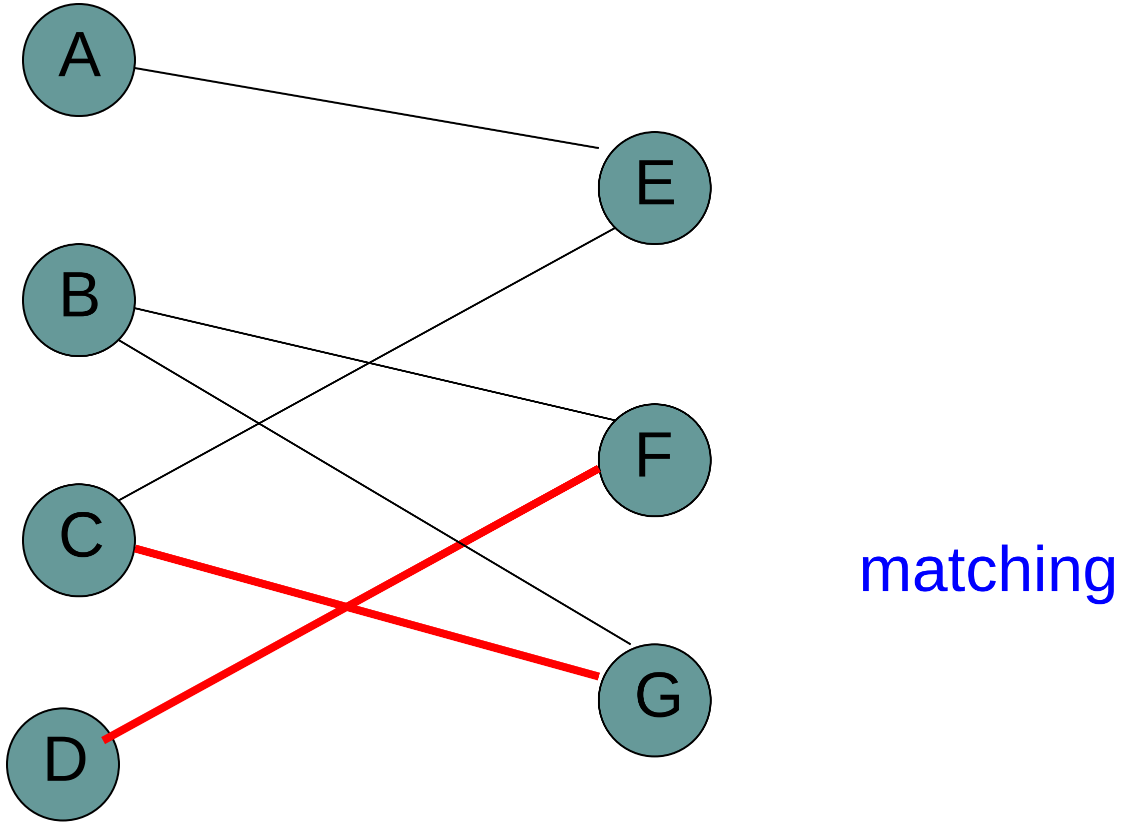
Application: bipartite graph matching

A *matching* M is a subset of edges such that each node occurs **at most once** in M



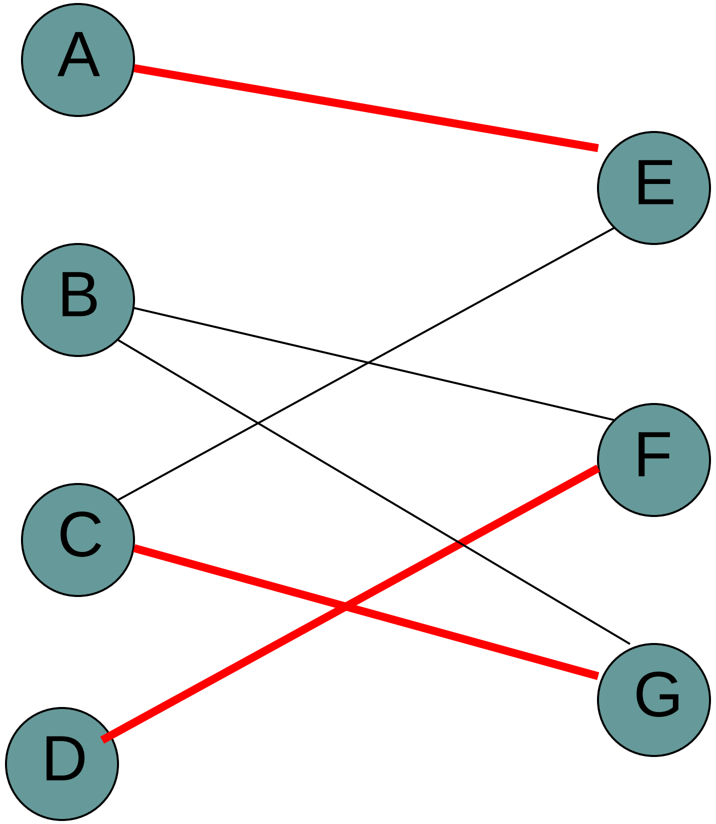
Application: bipartite graph matching

A *matching* M is a subset of edges such that each node occurs **at most once** in M



Application: bipartite graph matching

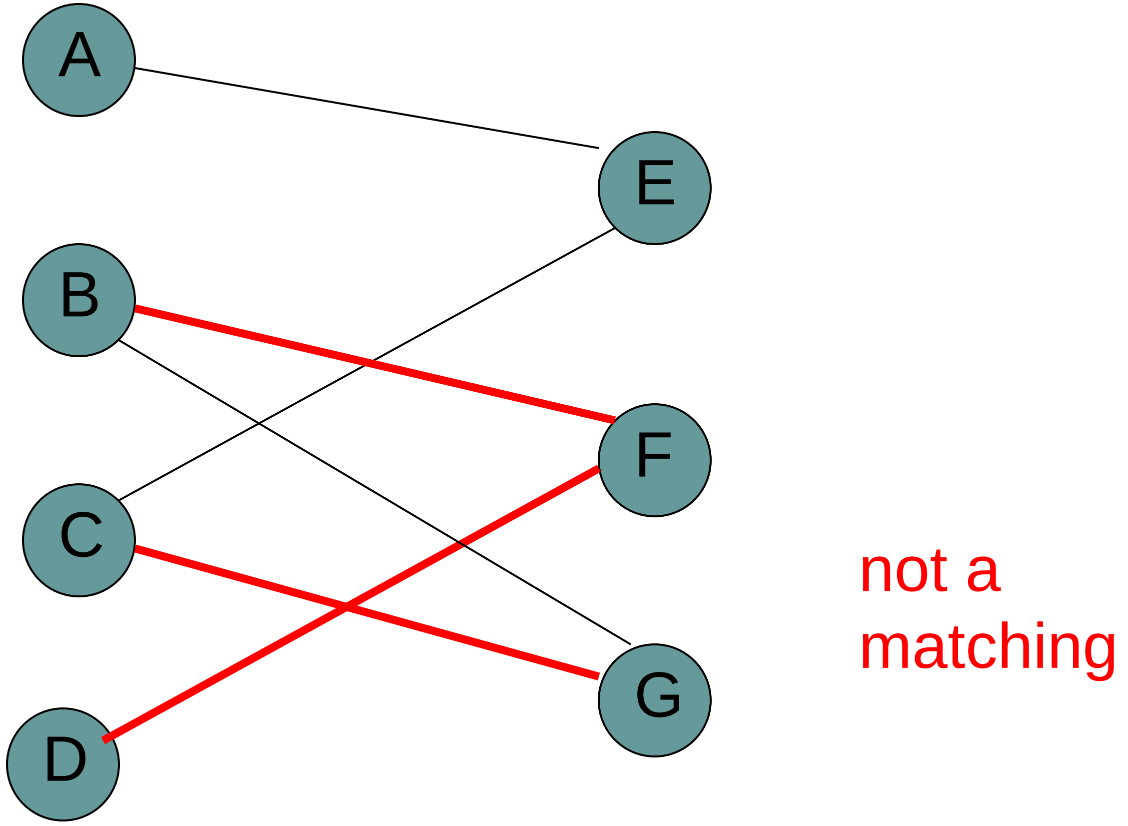
A *matching* M is a subset of edges such that each node occurs **at most once** in M



matching

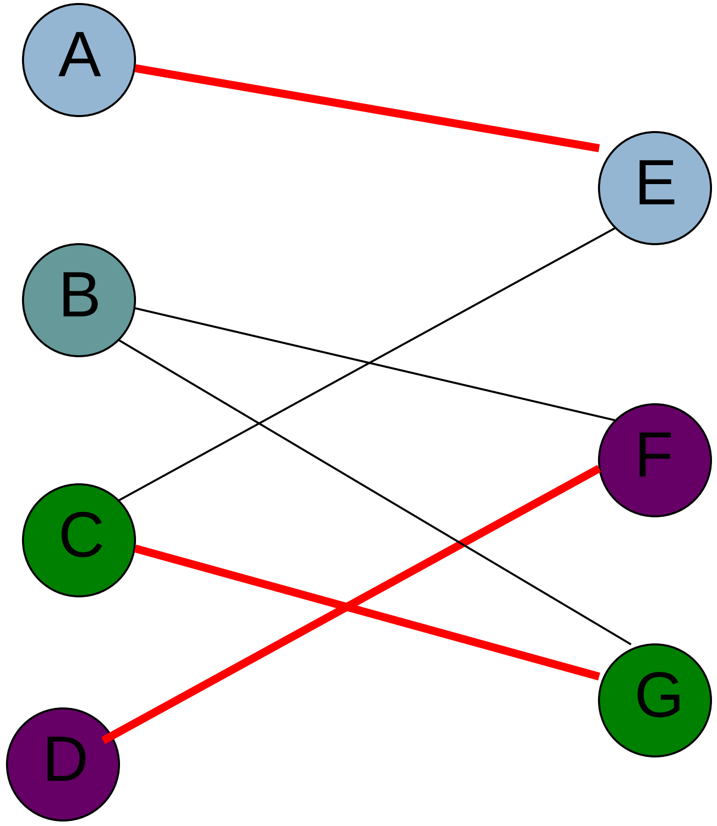
Application: bipartite graph matching

A *matching* M is a subset of edges such that each node occurs **at most once** in M



Application: bipartite graph matching

A *matching* can be thought of as pairing the vertices

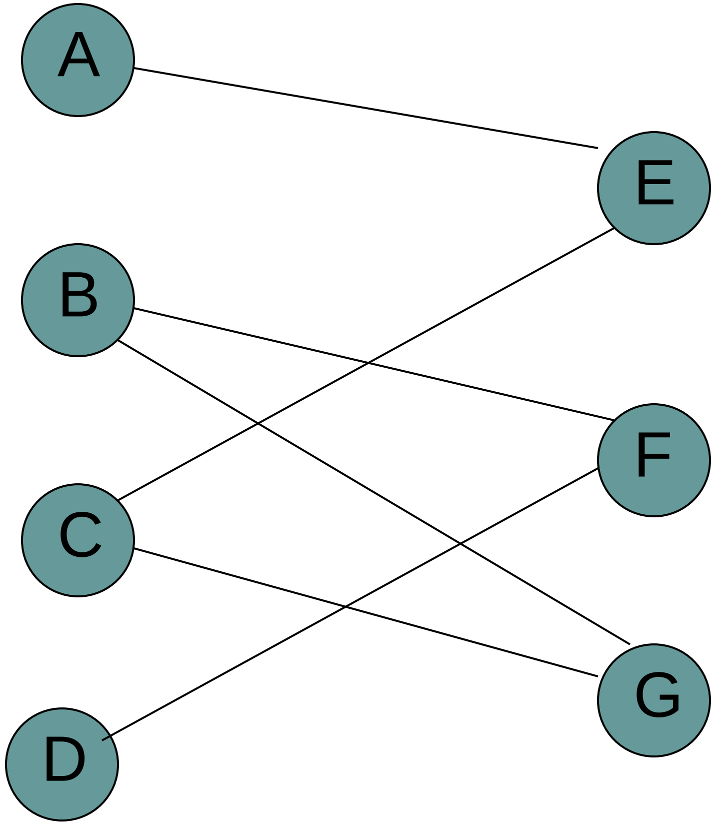


Application: bipartite graph matching

Bipartite matching problem: find the *largest* matching in a bipartite graph

Where might this problem come up?

- IT department has n courses and m faculty
- Every instructor can teach *some* of the courses
- What course should each person teach?
- Anytime we want to match n things with m, but not all things can match

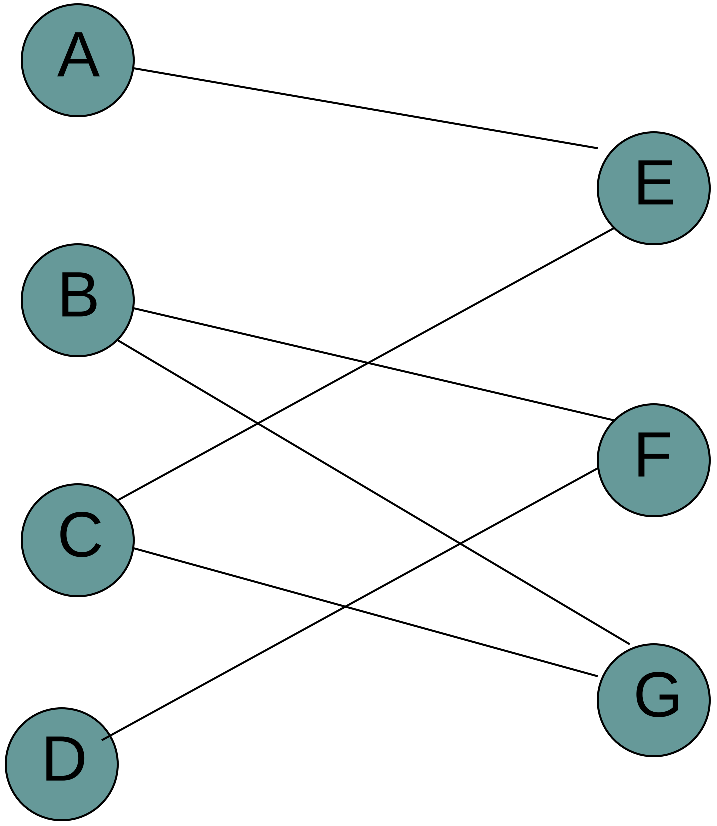


Application: bipartite graph matching

Bipartite matching problem: find the *largest* matching in a bipartite graph

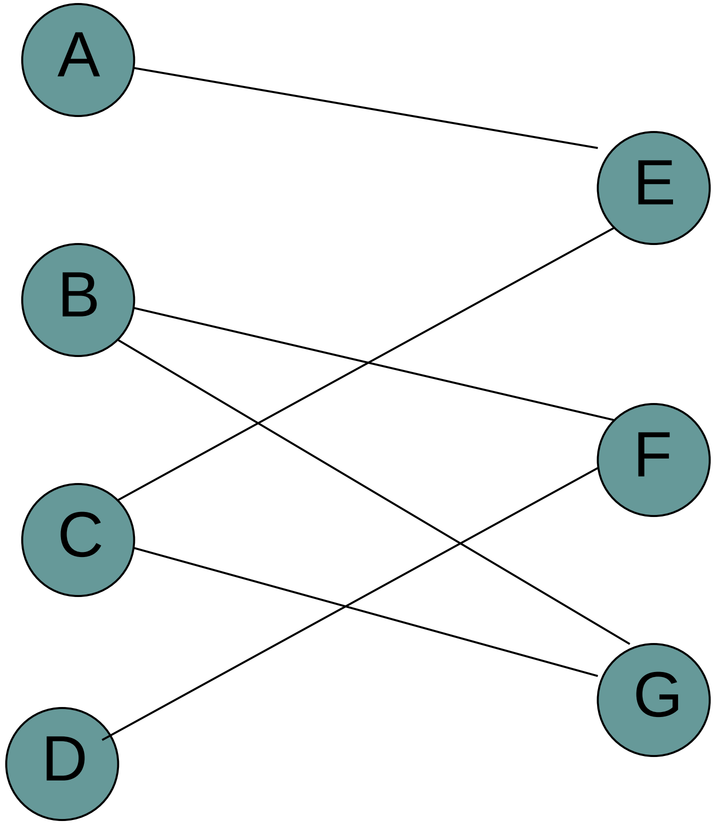
ideas?

- greedy?
- dynamic programming?



Application: bipartite graph matching

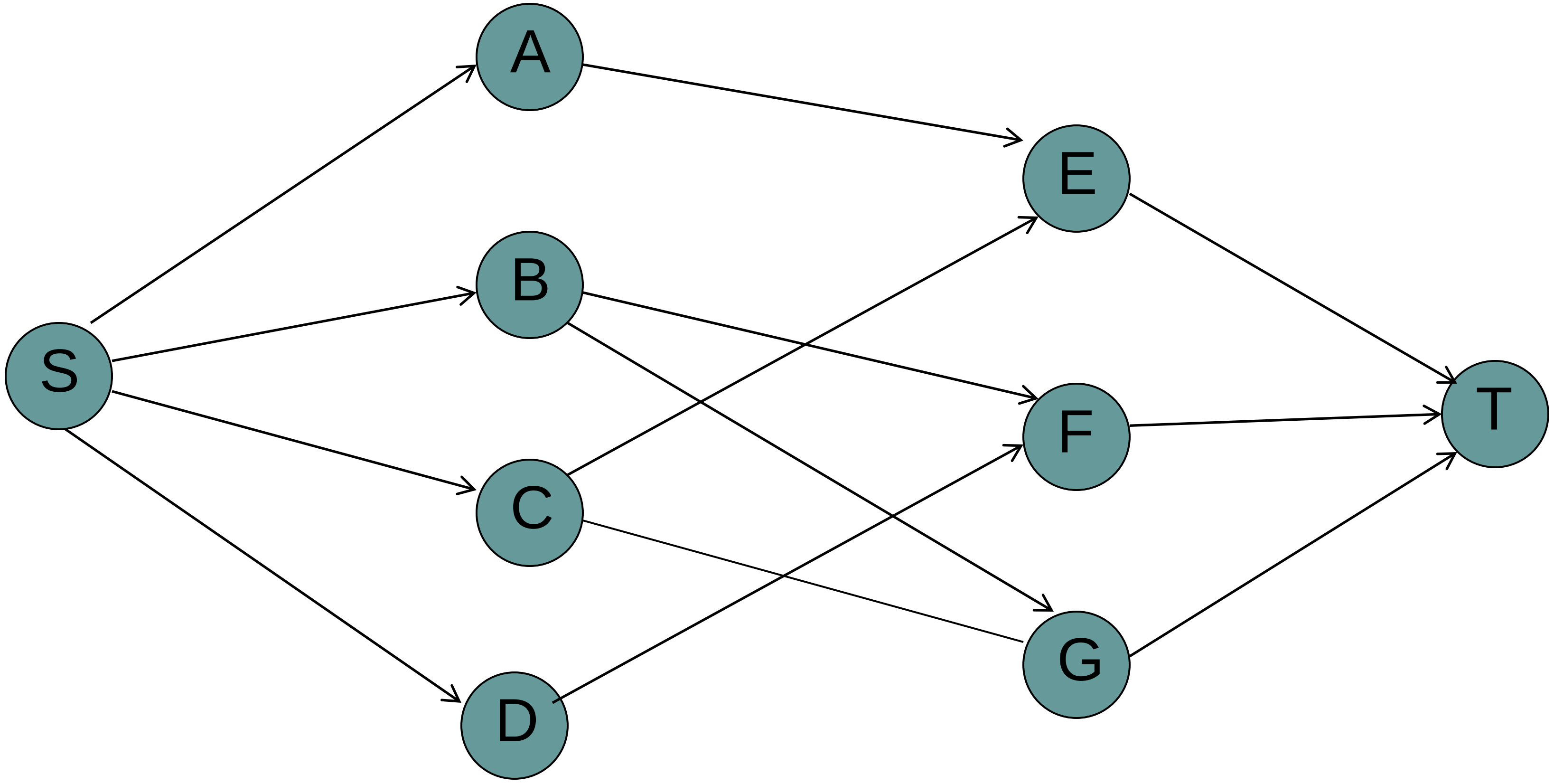
Setup as a flow problem:



Application: bipartite graph matching

Setup as a flow problem:

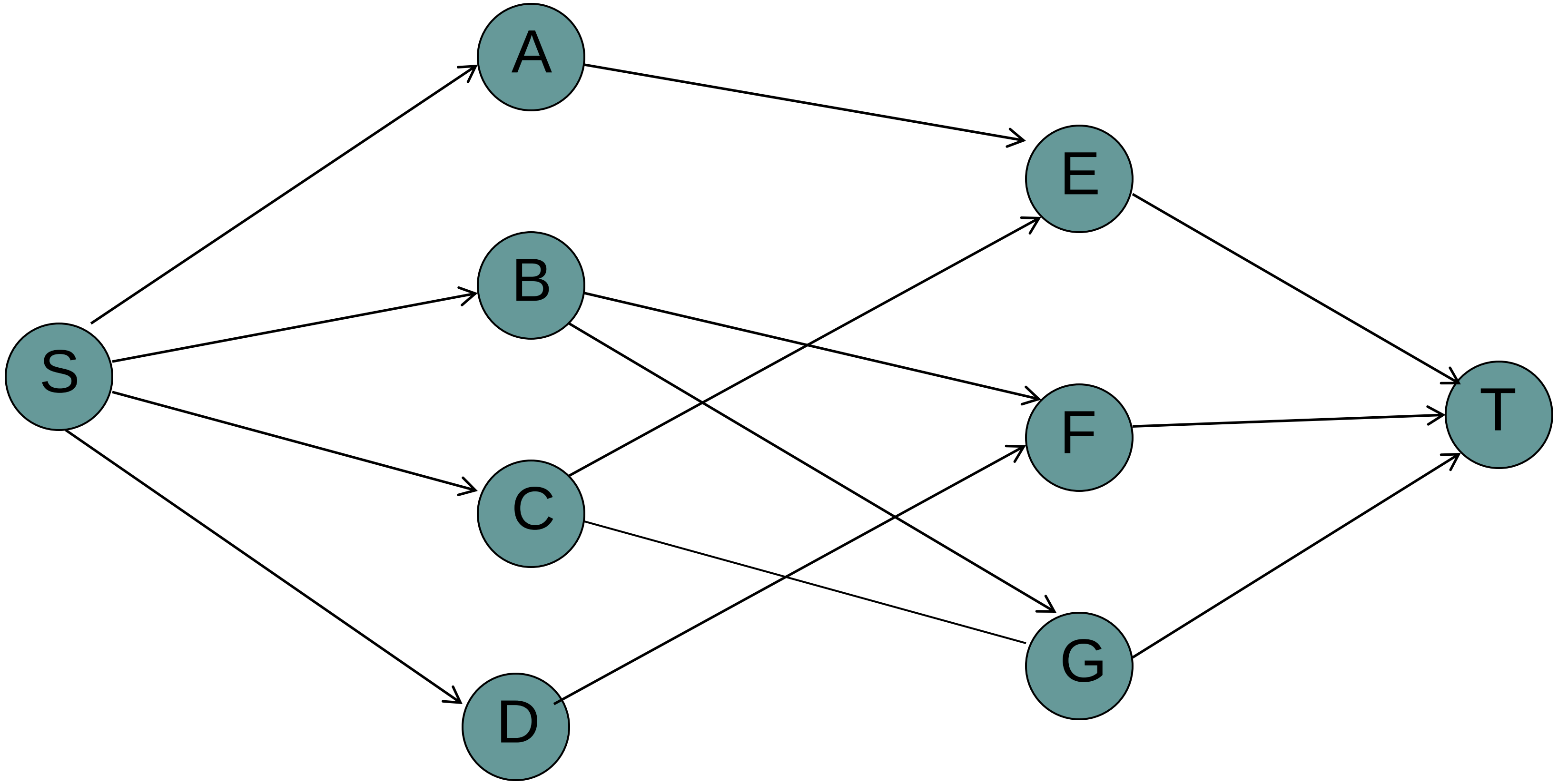
edge weights?



Application: bipartite graph matching

Setup as a flow problem:

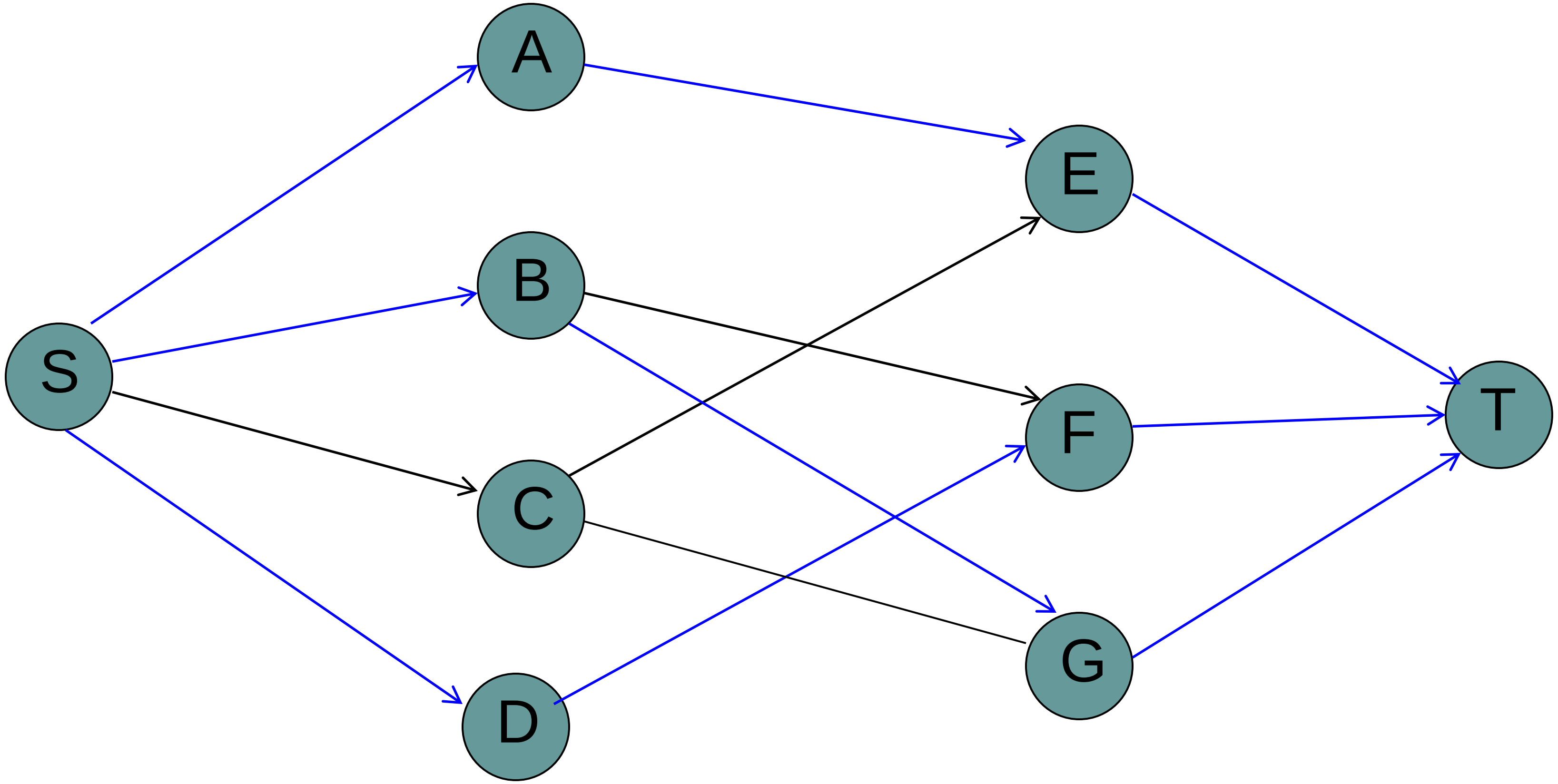
all edge weights are 1



Application: bipartite graph matching

Setup as a flow problem:

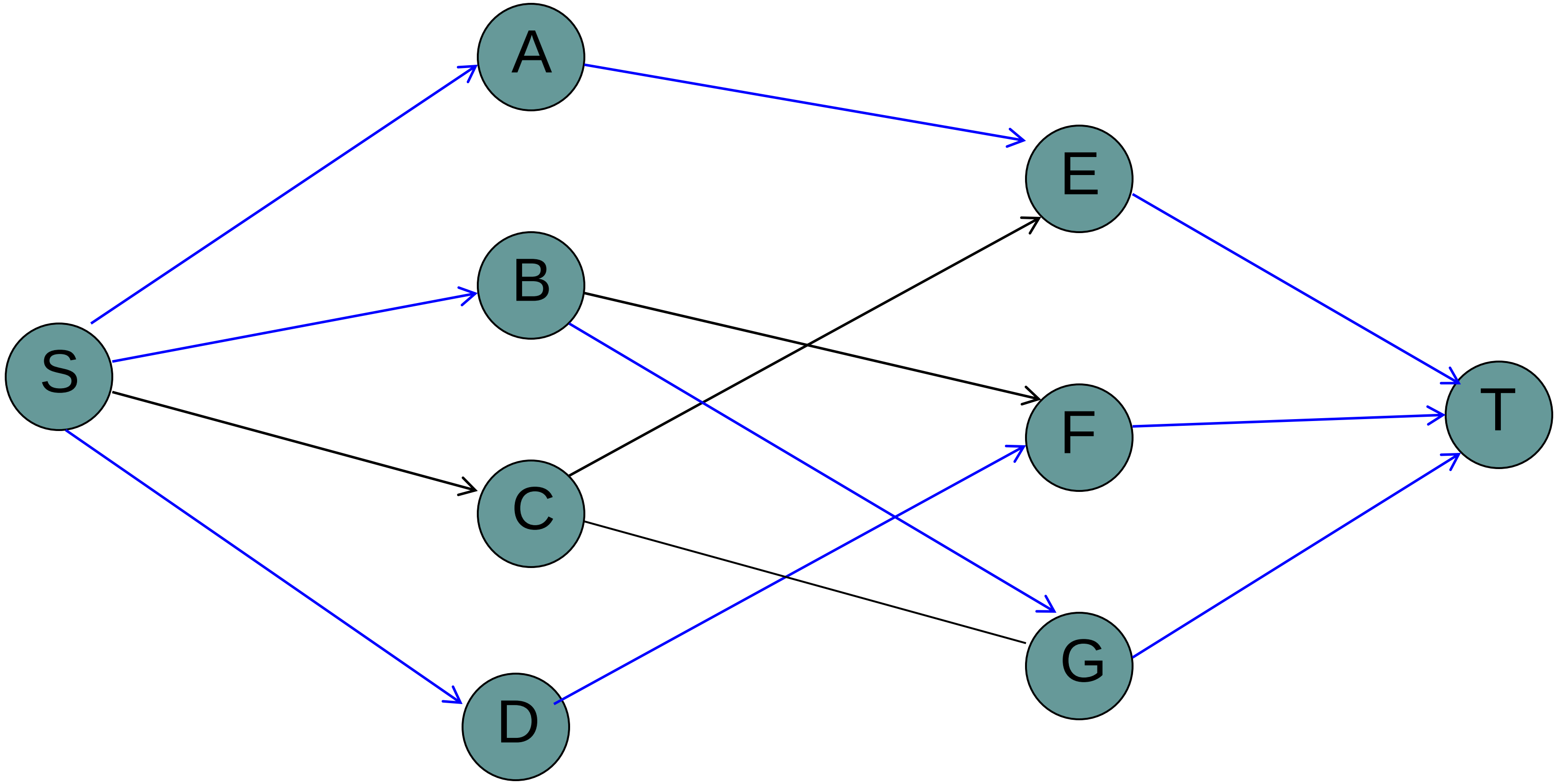
after we find the flow, how do we find the matching?



Application: bipartite graph matching

Setup as a flow problem:

match those nodes with flow between them



Application: bipartite graph matching

Is it correct?

Assume it's not

- there is a better matching
- because of how we setup the graph flow = # of matches
- therefore, the better matching would have a higher flow
- contradiction (max-flow algorithm finds maximal!)

Application: bipartite graph matching

Run-time?

Cost to build the flow?

- O(E)
  - each existing edge gets a capacity of 1
  - introduce V new edges (to and from s and t)
  - V is O(E) (for non-degenerate bipartite matching problems)

Max-flow calculation?

- Basic Ford-Fulkerson: O(max-flow * E)
- Edmunds-Karp: $O(V E^2)$
- Preflow-push: $O(V^3)$

Application: bipartite graph matching

Run-time?

Cost to build the flow?

- O(E)
  - each existing edge gets a capacity of 1
  - introduce V new edges (to and from s and t)
  - V is O(E) (for non-degenerate bipartite matching problems)

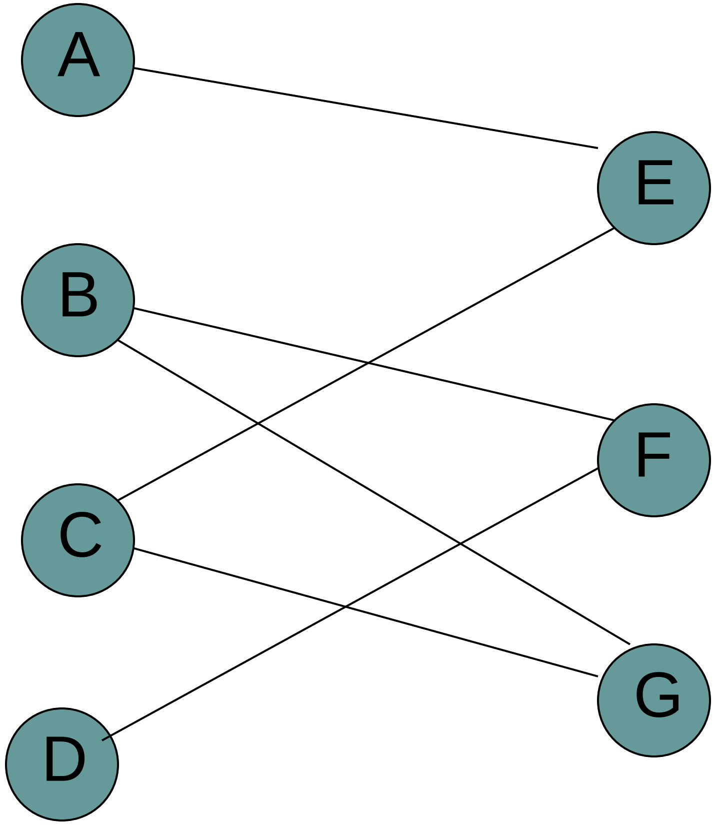
Max-flow calculation?

- Basic Ford-Fulkerson: O(max-flow * E)
  - max-flow = O(V)
  - O(V E)

Application: bipartite graph matching

Bipartite matching problem: find the *largest* matching in a bipartite graph

- IT department has n courses and m faculty
- Every instructor can teach *some* of the courses
- What course should each person teach?
- Each faculty can teach at most 3 courses a semester?



Change the s edge weights (representing faculty) to 3

Survey Design

Design a survey with the following requirements:

- Design survey asking *n* consumers about *m* products
- Can only survey consumer about a product if they own it
- Question consumers about at most *q* products
- Each product should be surveyed at most *s* times
- Maximize the number of surveys/questions asked

How can we do this?

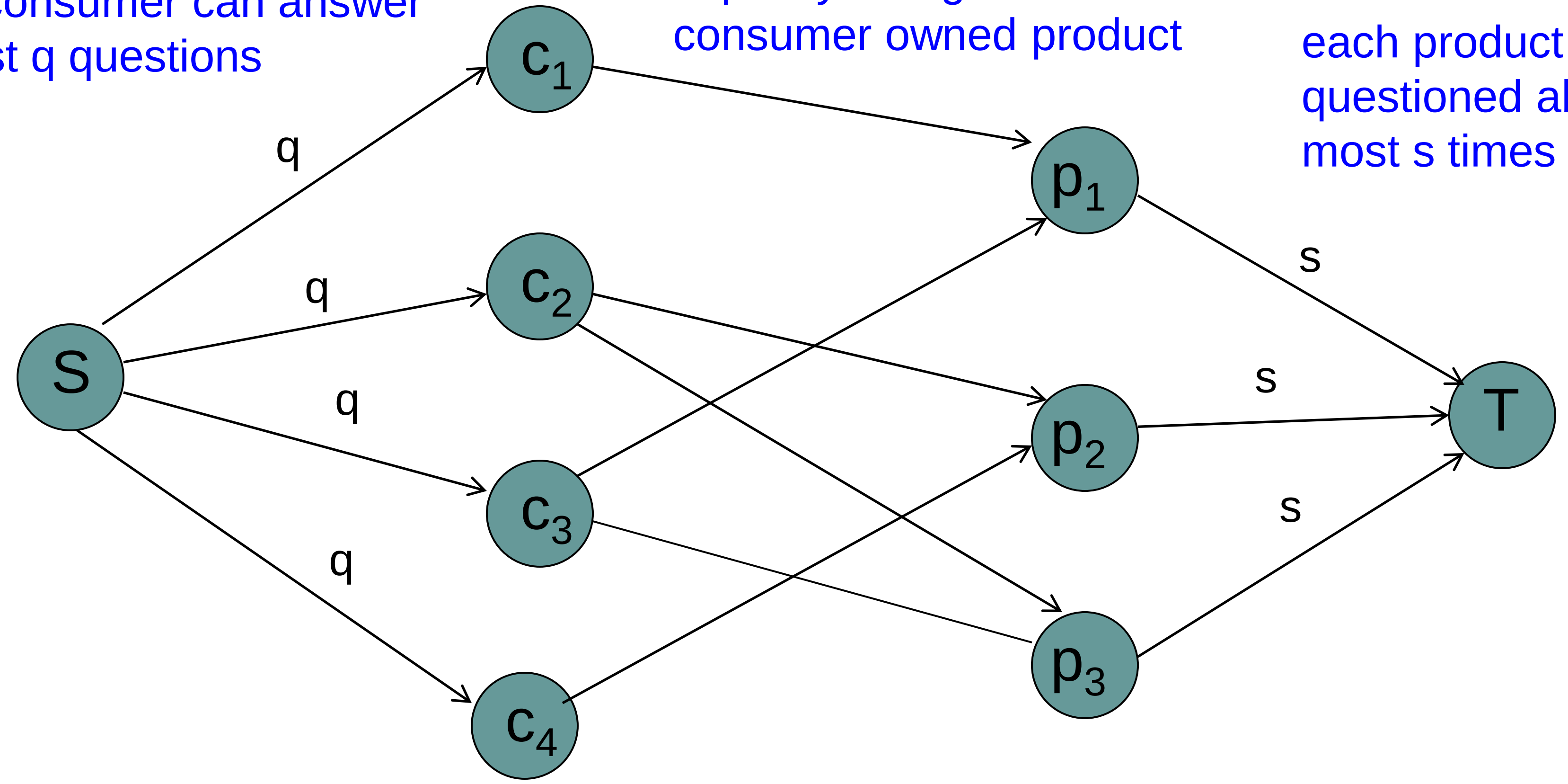
Survey Design

consumers products

each consumer can answer at most q questions

capacity 1 edge if consumer owned product

each product can be questioned about at most s times



Survey design

Is it correct?

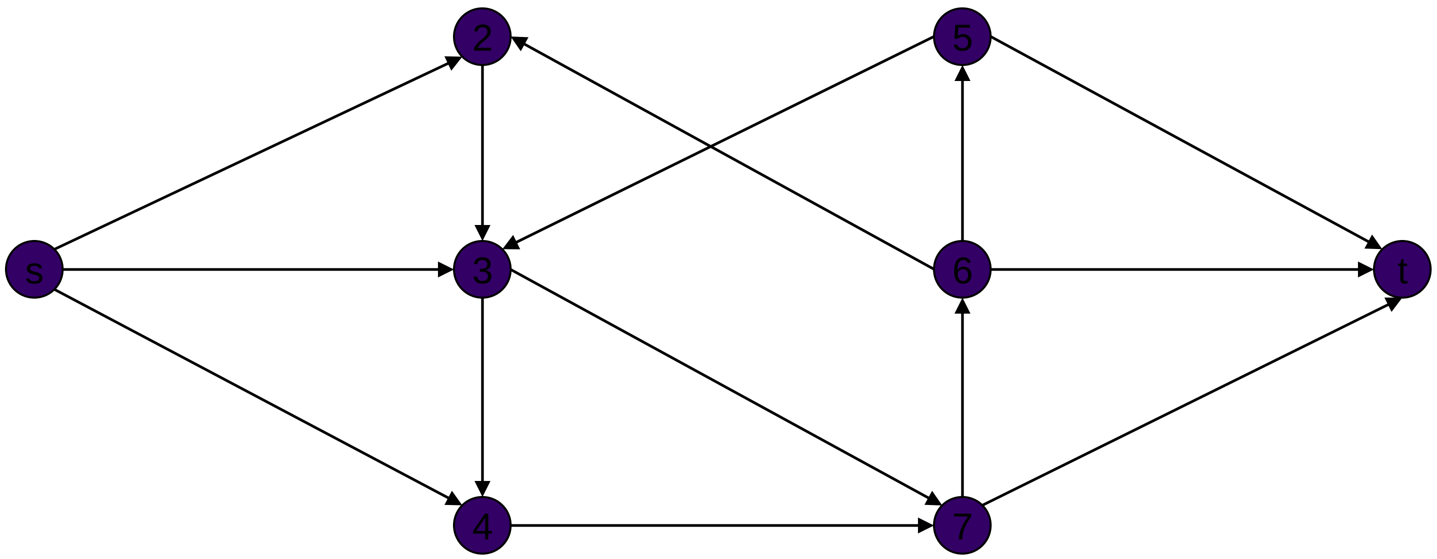
- Each of the comments above the flow graph match the problem constraints
- max-flow finds the maximum matching, given the problem constraints

What is the run-time?

- Basic Ford-Fulkerson: O(max-flow * E)
- Edmunds-Karp: $O(V E^2)$
- Preflow-push: $O(V^3)$

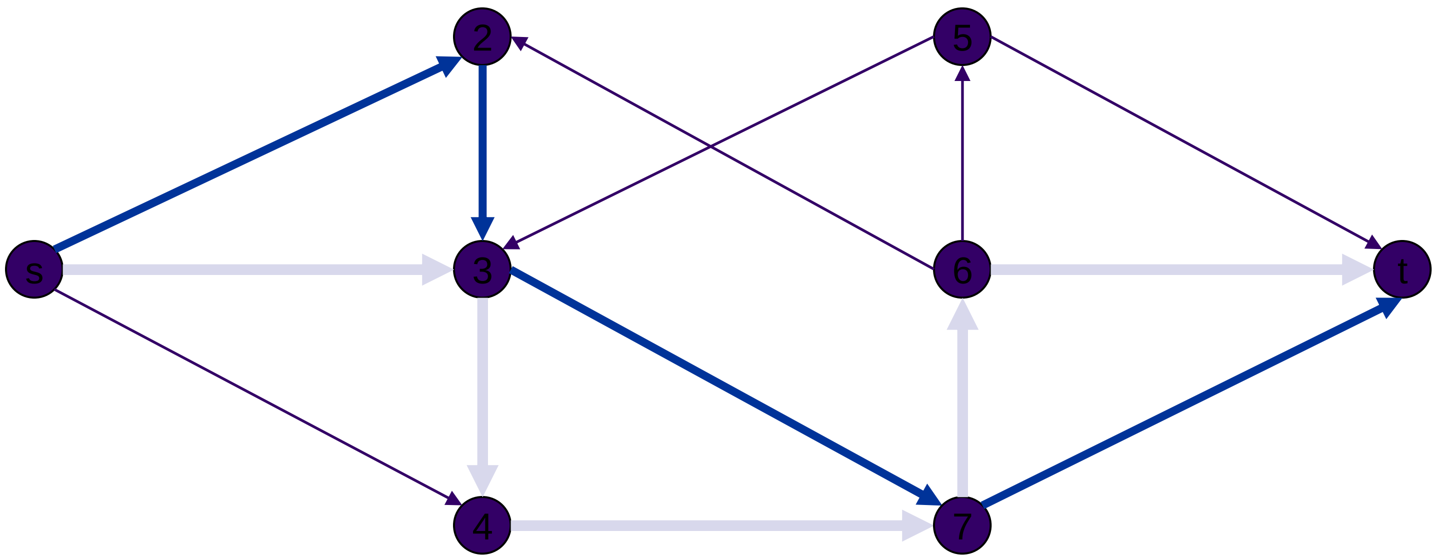
Edge Disjoint Paths

Two paths are edge-disjoint if they have no edge in common



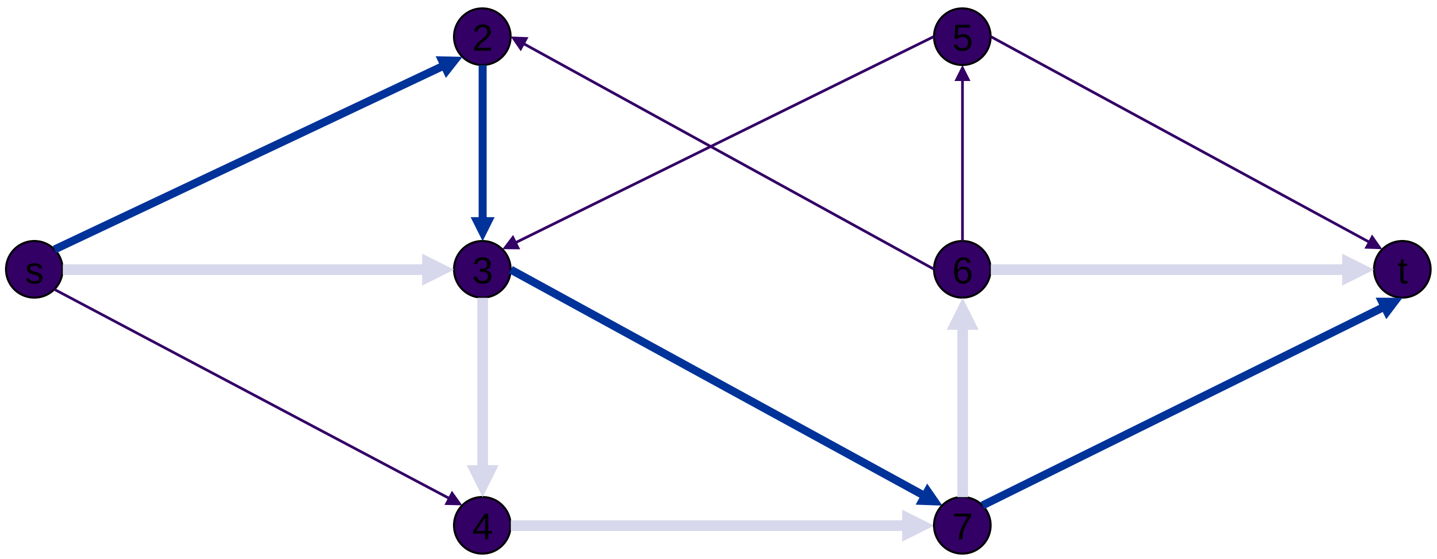
Edge Disjoint Paths

Two paths are edge-disjoint if they have no edge in common



Edge Disjoint Paths Problem

Given a directed graph G = (V, E) and two nodes s and t, find the max number of edge-disjoint paths from s to t



Why might this be useful?

Edge Disjoint Paths Problem

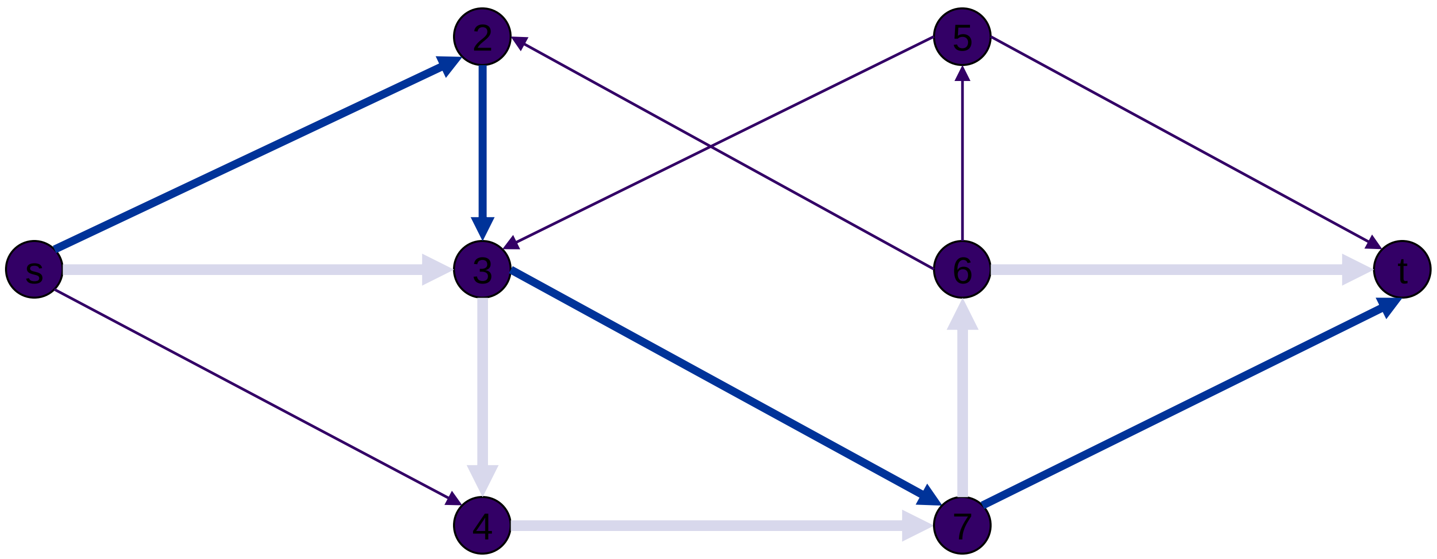
Given a directed graph G = (V, E) and two nodes s and t, find the max number of edge-disjoint paths from s to t

Why might this be useful?

- edges are unique resources (e.g. communications, transportation, etc.)
- how many *concurrent (non-conflicting)* paths do we have from s to t

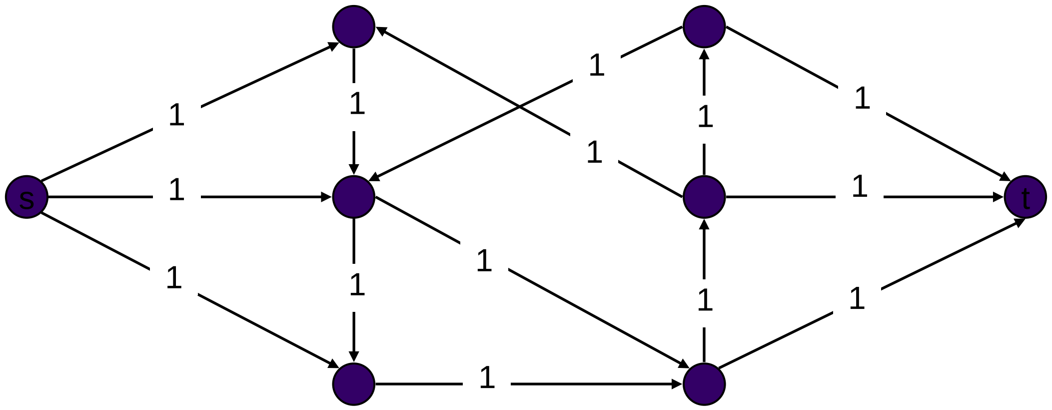
Edge Disjoint Paths

Algorithm ideas?



Edge Disjoint Paths

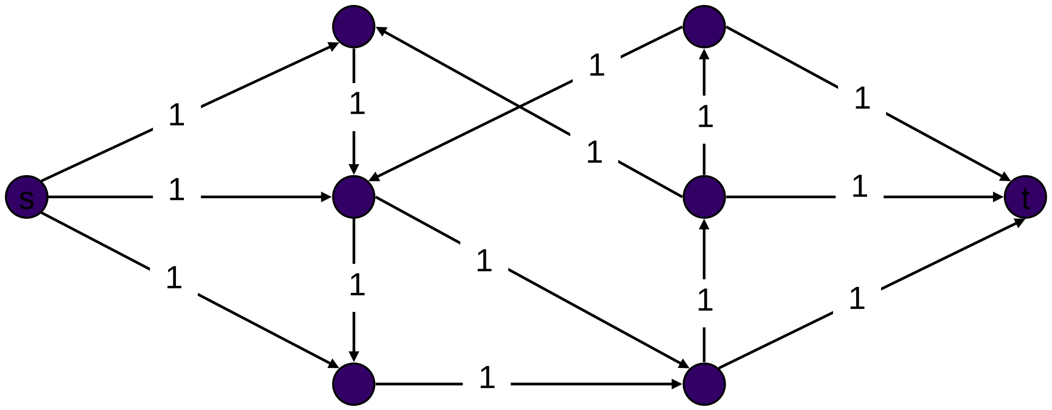
Max flow formulation: assign unit capacity to every edge



What does the max flow represent?
Why?

Edge Disjoint Paths

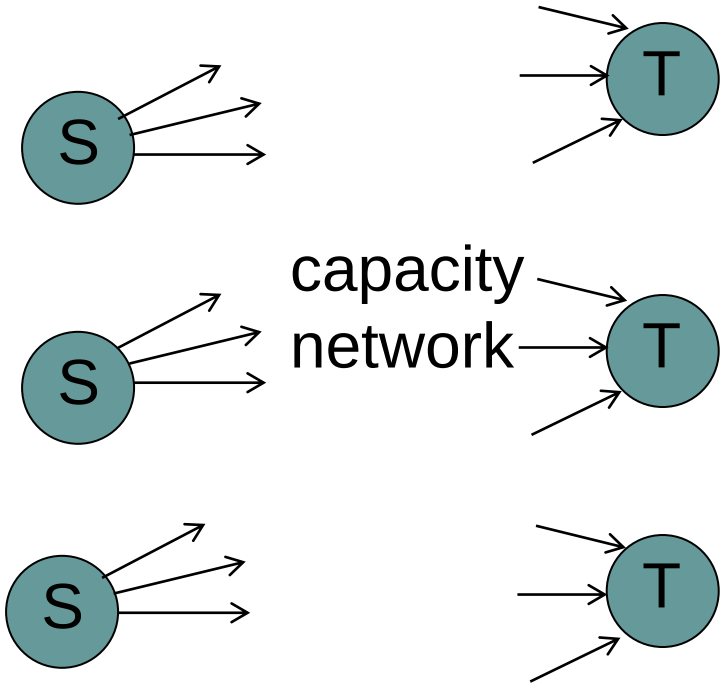
Max flow formulation: assign unit capacity to every edge



- max-flow = maximum number of disjoint paths
- correctness:
  - each edge can have at most flow = 1, so can only be traversed once
  - therefore, each unit out of s represents a separate path to t

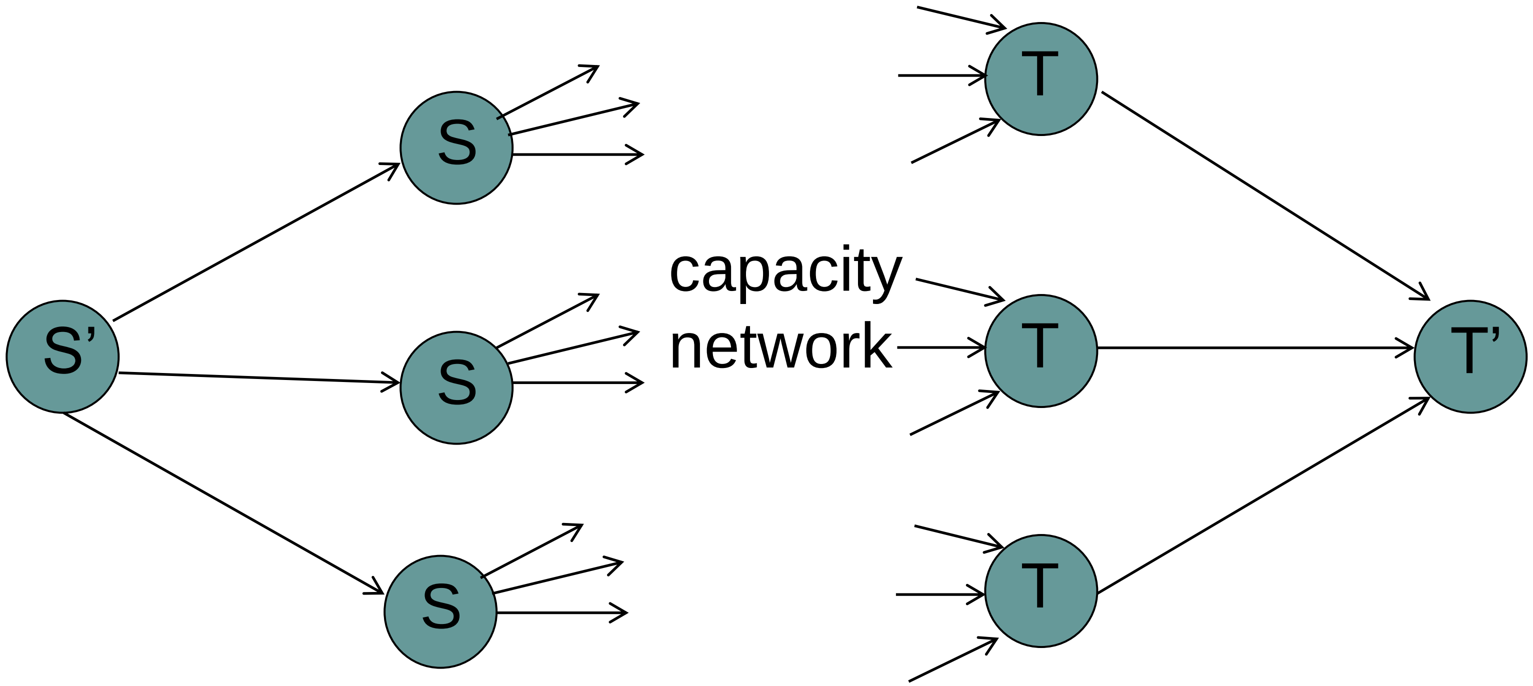
Max-flow variations

What if we have multiple sources and multiple sinks (e.g. the Russian train problem has multiple sinks)?



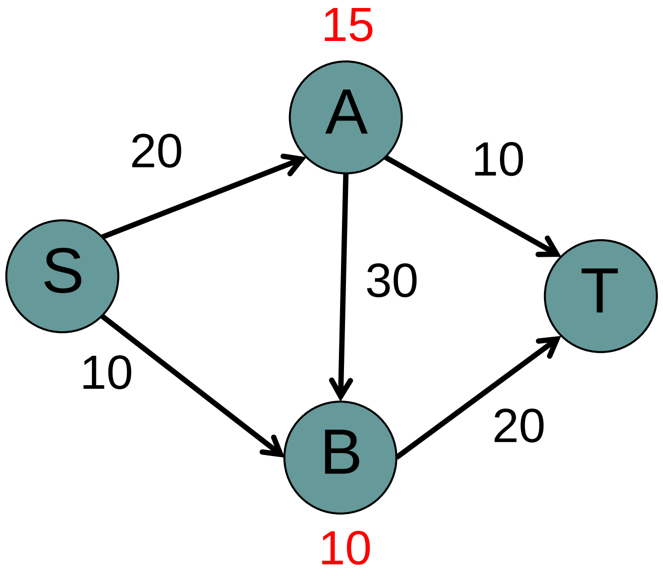
Max-flow variations

Create a new source and sink and connect up with infinite capacities…



Max-flow variations

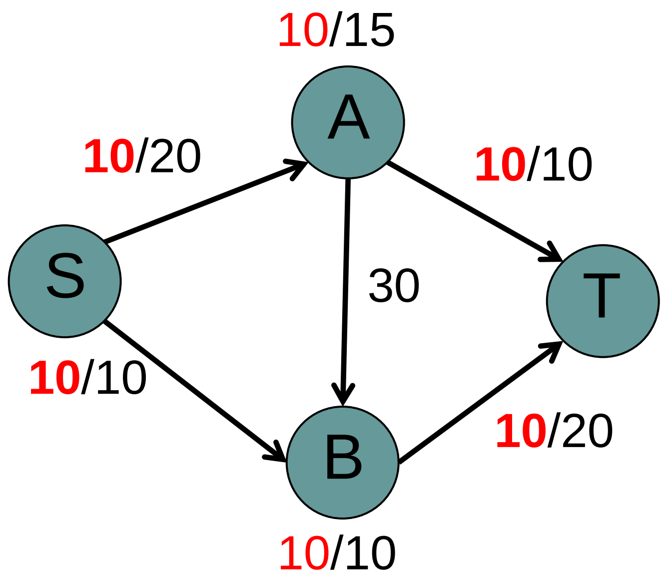
Vertex capacities: in addition to having edge capacities we can also restrict the amount of flow through each vertex



What is the max-flow now?

Max-flow variations

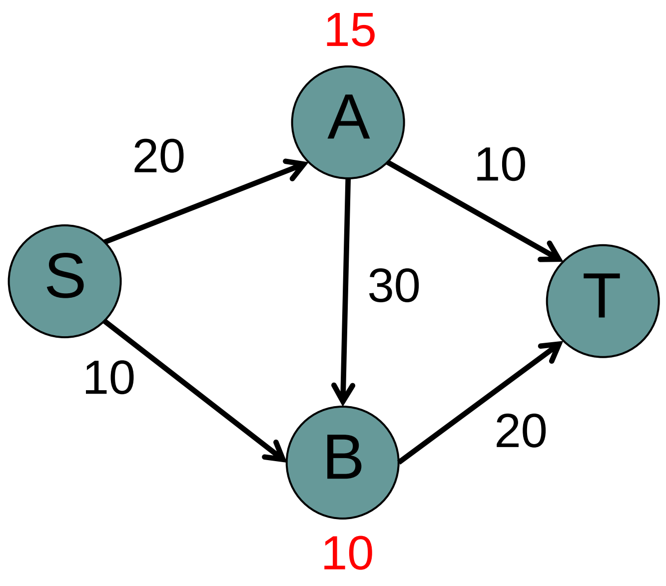
Vertex capacities: in addition to having edge capacities we can also restrict the amount of flow through each vertex



20 units

Max-flow variations

Vertex capacities: in addition to having edge capacities we can also restrict the amount of flow through each vertex

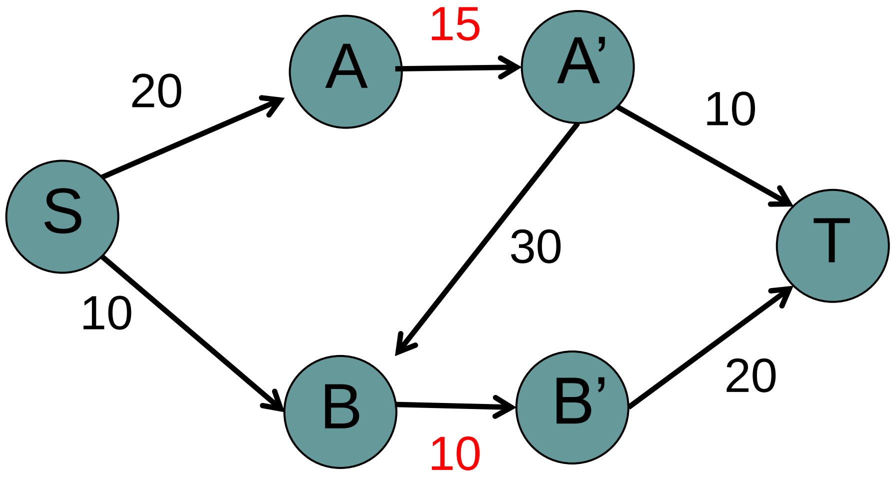


How can we solve this problem?

Max-flow variations

For each vertex v

- create a new node v'
- create an edge with the vertex capacity from v to v'
- move all outgoing edges from v to v'



Can you now prove it's correct?

Max-flow variations

Proof:

1. show that if a solution exists in the original graph, then a solution exists in the modified graph
2. show that if a solution exists in the modified graph, then a solution exists in the original graph

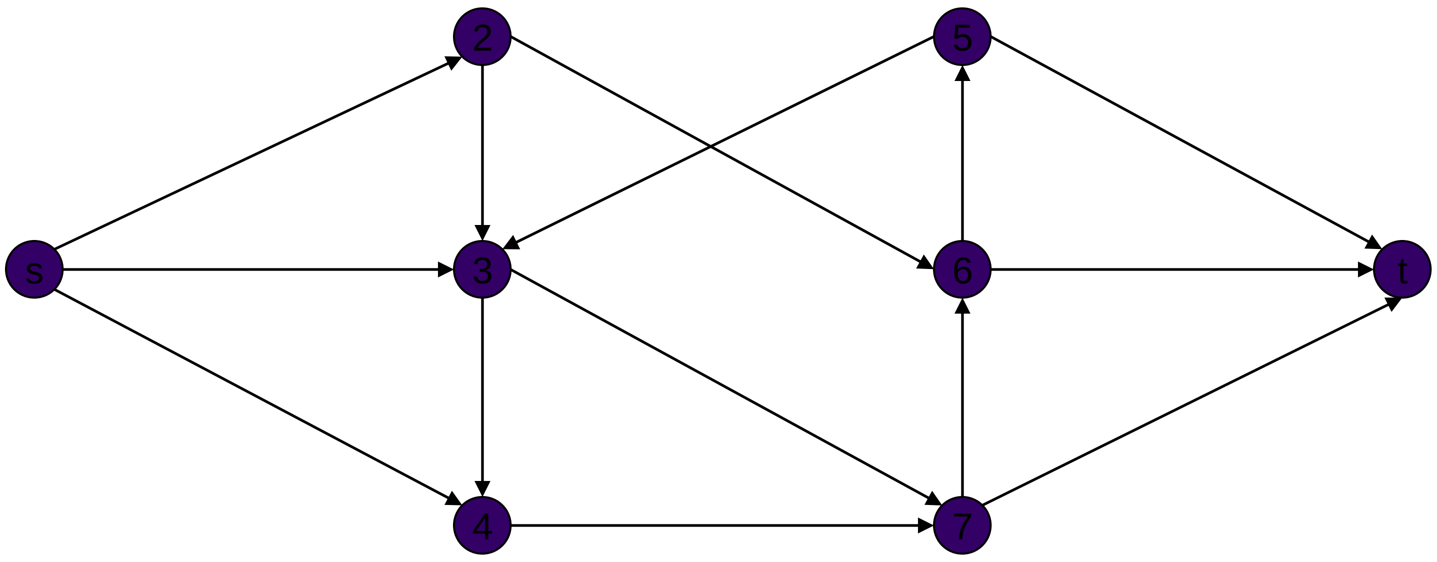
Max-flow variations

Proof:

- we know that the vertex constraints are satisfied
  - no incoming flow can exceed the vertex capacity since we have a single edge with that capacity from v to v’
- we can obtain the solution, by collapsing each v and v’ back to the original v node
  - in-flow = out-flow since there is only a single edge from v to v’
  - because there is only a single edge from v to v’ and all the in edges go in to v and out to v’, they can be viewed as a single node in the original graph

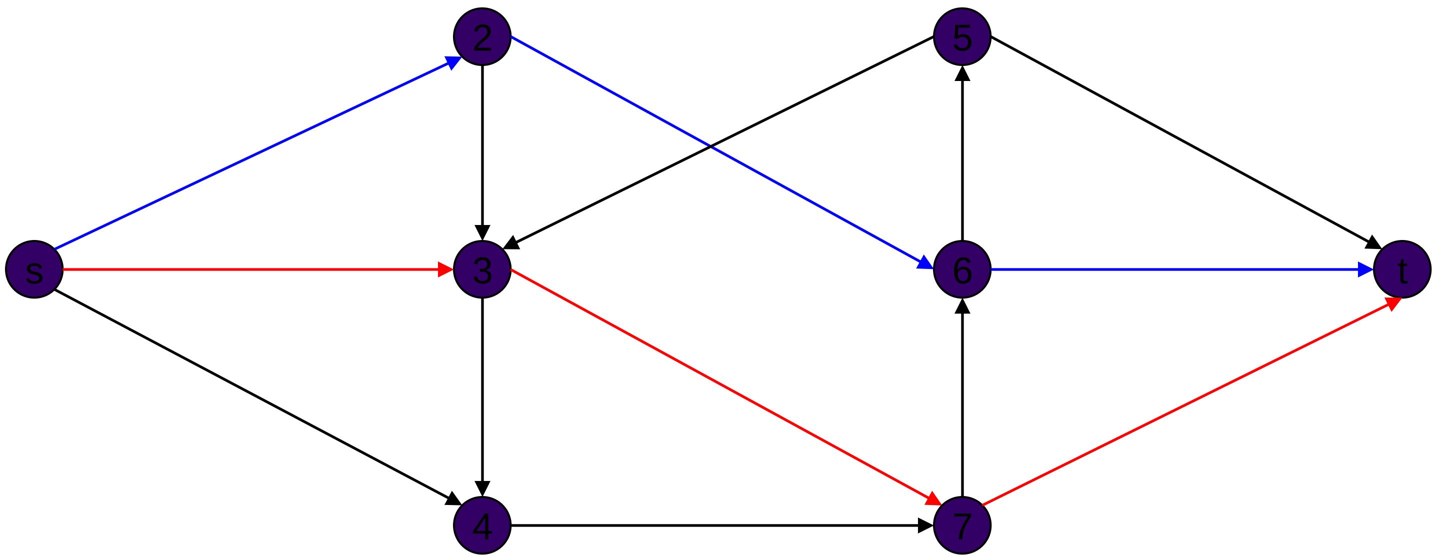
More problems: Maximum independent path

Two paths are independent if they have no *vertices* in common



More problems:
Maximum independent path

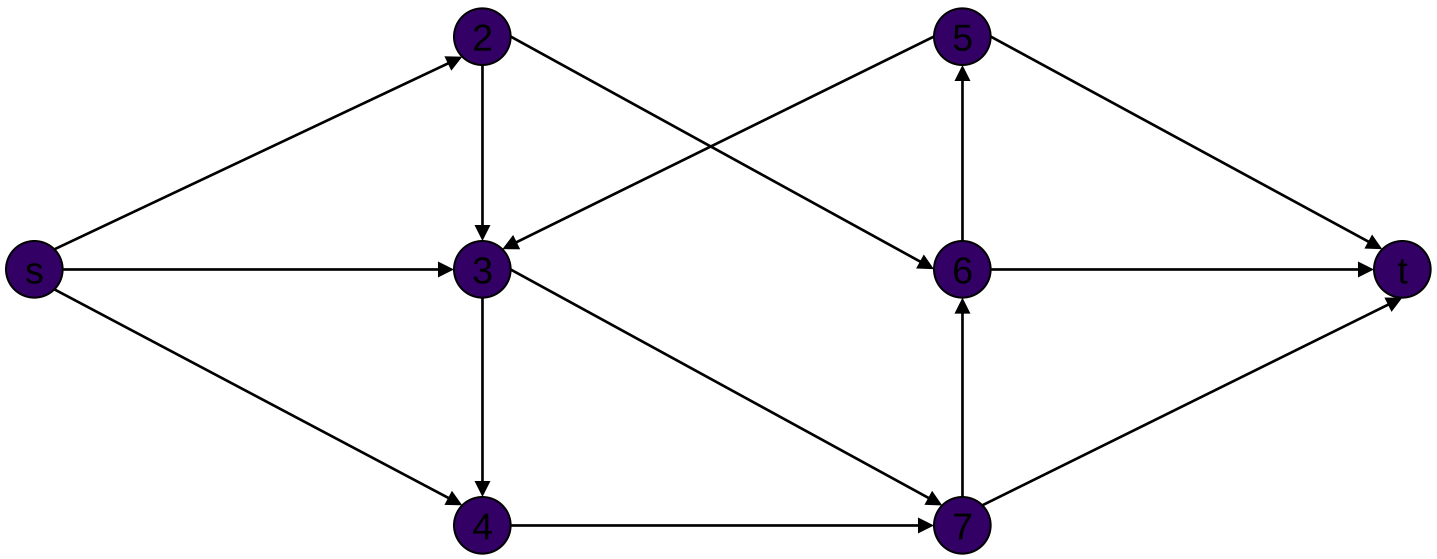
Two paths are independent if they have no *vertices* in common



More problems:
Maximum independent path

Find the maximum number of independent paths

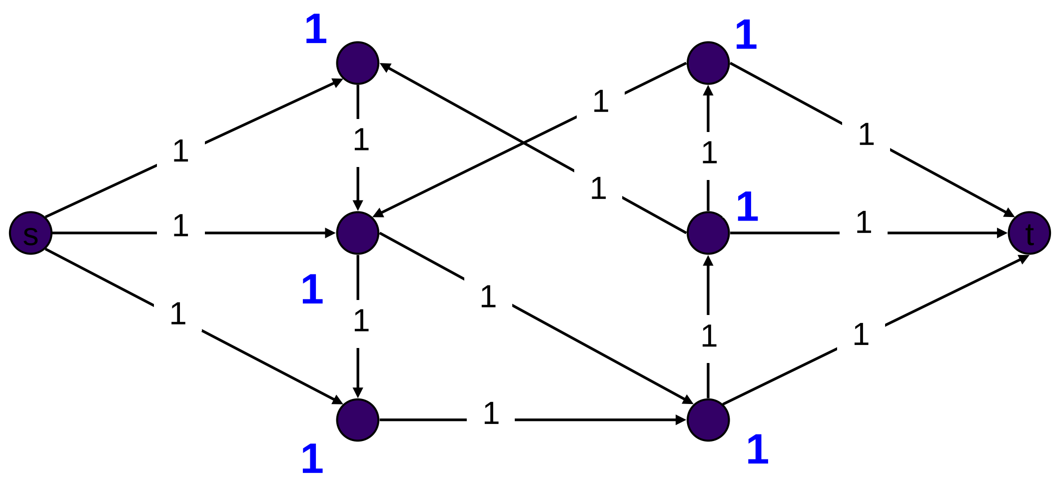
Ideas?



Maximum independent path

Max flow formulation:

- assign unit capacity to every edge (though any value would work)
- assign unit capacity to every vertex



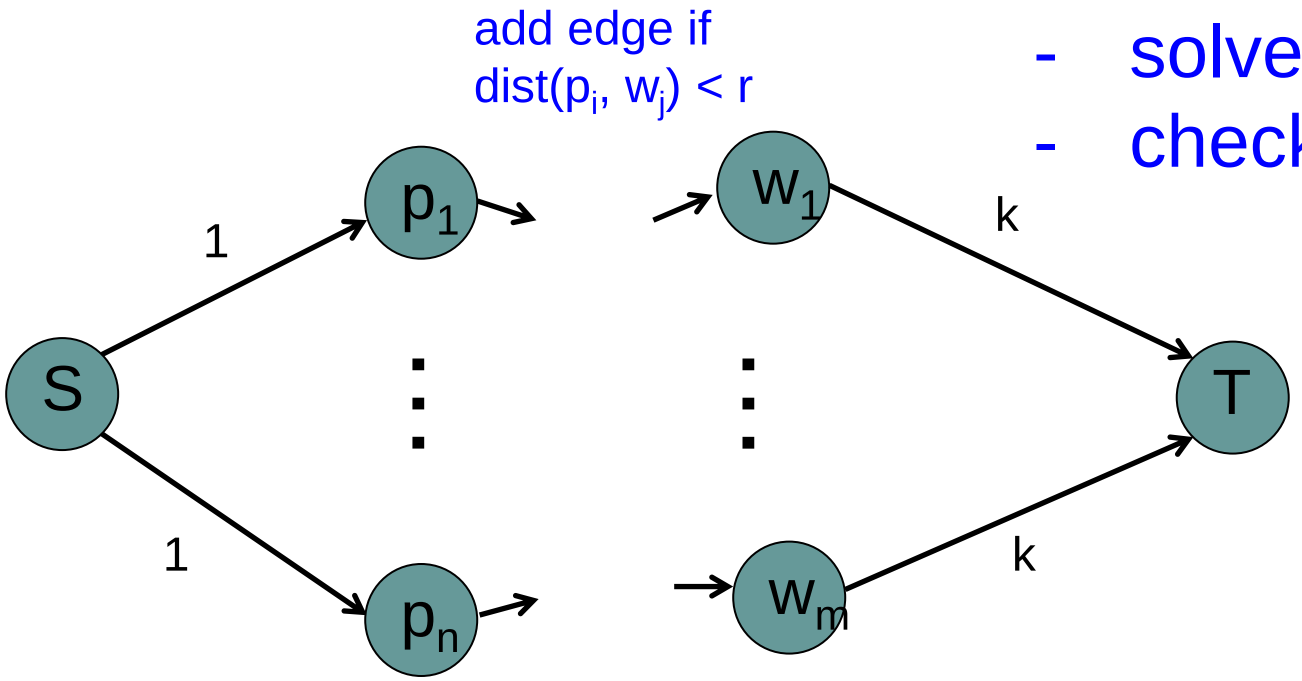
Same idea as the maximum edge-disjoint paths, but now we also constrain the vertices

More problems: wireless network

- The campus has hired you to setup the wireless network
- There are currently *m* wireless stations positioned at various (x,y) coordinates on campus
- The range of each of these stations is *r* (i.e. the signal goes at most distance *r*)
- Any particular wireless station can only host *k* people connected
- You've to calculate the *n* most popular locations on campus and have their (x,y) coordinates
- Could the current network support n different people trying to connect at each of the *n* most popular locations (i.e. one person per location)?
- Prove correctness and state run-time

Another matching problem

- n people nodes and m station nodes
- if dist(p_i,w_j) < r then add an edge from pi to wj with weight 1 (where dist is euclidean distance)
- add edges s -> p_i with weight 1
- add edges w_j -> t with weight k



- solve for max-flow
- check if flow = m

Correctness

If there is flow from a person node to a wireless node then that person is attached to that wireless node

if dist(pi,wj) < r then add an edge from pi to wj with weigth 1 (where dist is euclidean distance)

- only people able to connect to node could have flow

add edges s -> pi with weight 1

- each person can only connect to one wireless node

add edges wj -> t with weight L

- at most L people can connect to a wireless node

If flow = m, then every person is connected to a node

Runtime

$E = O(mn)$: every person is within range of every node

$V = m + n + 2$

max-flow = $O(m)$, s has at most m out-flow

- $O(\text{max-flow} * E) = O(m^2n)$: Ford-Fulkerson
- $O(VE^2) = O((m+n)m^2n^2)$: Edmunds-Karp
- $O(V^3) = O((m+n)^3)$: preflow-push variant

Acknowledgements

- Cormen, T.H., Leiserson, C.E., Rivest, R.L. and Stein, C., Introduction to algorithms. MIT press, 2009
- Dr. David Kauchak, Pomona College
- Prof. David Plaisted, The University of North Carolina at Chapel Hill