



Admin

- Midterm 2
 - Due in exactly 1 week
 - Possible topics:
 - Greedy Algorithms
 - Dynamic Programming
 - Graph Algorithms
 - Earlier materials?
 - Review questions?
 - If you have questions you'd like to see covered Thursday, write them down on today's worksheet!

Network Flow

- The idea: Use a graph to model material that flows through conduits
 - Each edge represents one conduit and has a capacity, which is an upper bound on the flow rate = units/time
 - We want to compute max rate that we can ship material from a designated source to designated sink.



Network Flow (Key Ideas Preview)

- Terminology / Notation:
 - Capacity:
 - Source:
 - Sink:
 - Flow:
 - Capacity constraint:
 - Flow conservation
 - Value of flow:
- Maximum-flow problem
- Cuts:
 - Net flow:
 - Capacity:



Three Theorems about Flow

The Cut Theorem: For any cut S, T and flow f , the flow across the cut is equal to $|f|$. That is:

$$\sum_{x \in S, y \in T} f(x, y) = |f|$$

The Capacity Theorem: For any cut S, T and flow f , the flow is bounded by the capacity of the cut. That is: $|f| \leq \sum_{x \in S, y \in T} c(x, y)$

Max Flow/Min Cut Theorem: For any cut S, T and flow f , if $|f| = \sum_{x \in S, y \in T} c(x, y)$ then f is max flow and S, T is a min cut.

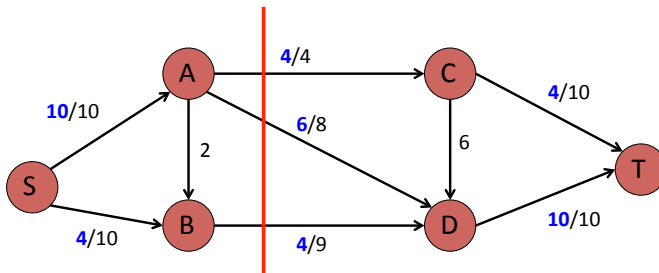
Network flow properties

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

- f is a maximum flow
- G_f (residual graph) has no paths from s to t
- $|f|$ = minimum capacity cut

Flow across cuts

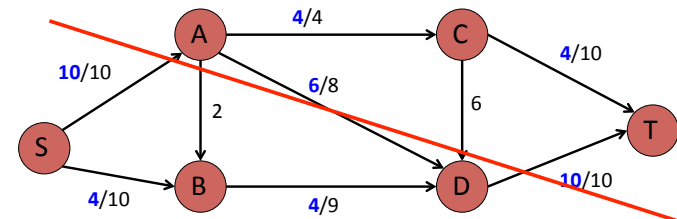
In flow graphs, we're interested in cuts that separate s from t , that is $s \in A$ and $t \in B$



Flow across cuts

The **net flow** “across” a cut is the total flow from nodes in A to nodes in B *minus* the total from from B to A

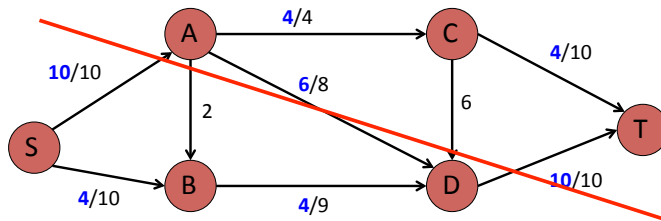
What is the flow across this cut?



Flow across cuts

The net flow “**across**” a cut is the total flow from nodes in A to nodes in B *minus* the total from from B to A

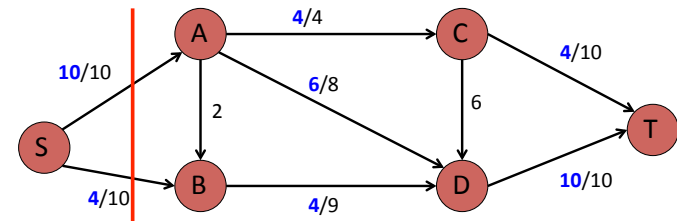
$$10+10-6 = 14$$



Flow across cuts

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

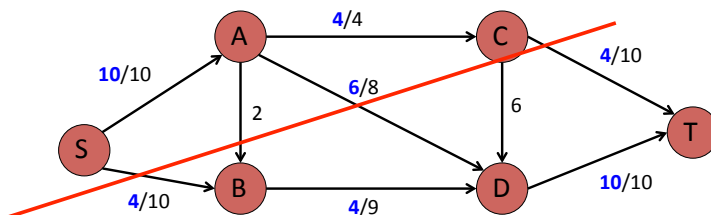
$$4+10 = 14$$



Flow across cuts

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

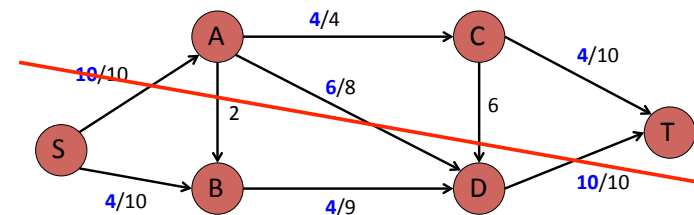
$$4+6+4 = 14$$



Flow across cuts

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

$$10+10-6 = 14$$

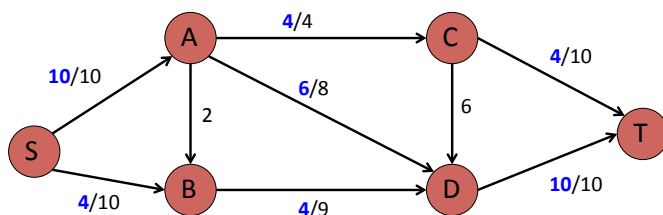


Flow across cuts

Consider any cut where $s \in A$ and $t \in B$, 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

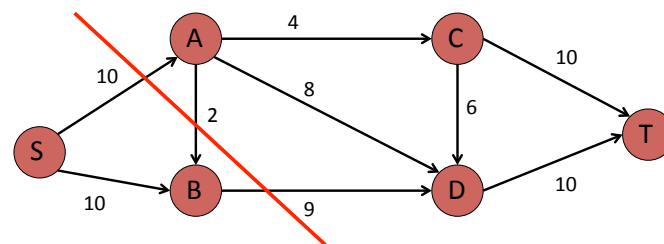
Why? Can you prove it?



Capacity of a cut

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

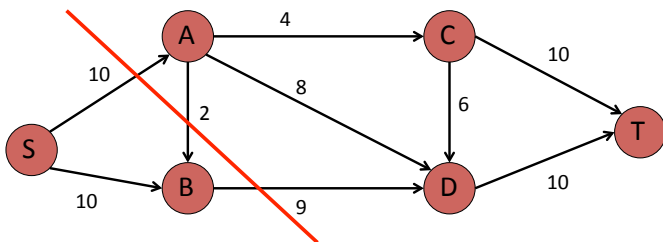
How do we calculate the capacity?



Capacity of a cut

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

Capacity is the sum of the edges from A to B

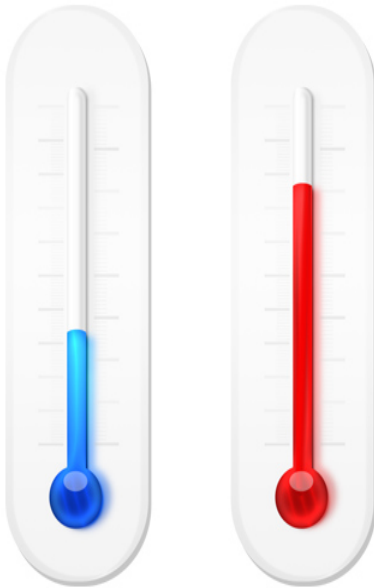


Quick recap

A cut is a partitioning of the vertices into two sets A and $B = V - A$

For any cut where $s \in A$ and $t \in B$, i.e. the cut partitions the source from the sink

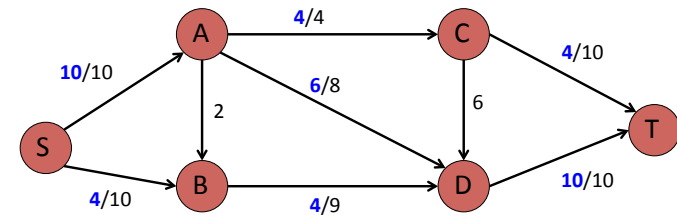
- the flow across any such cut is the same
- the maximum capacity (i.e. flow) across the cut is the sum of the capacities for edges from A to B



Maximum flow

For any cut where $s \in A$ and $t \in B$

- 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 A to B

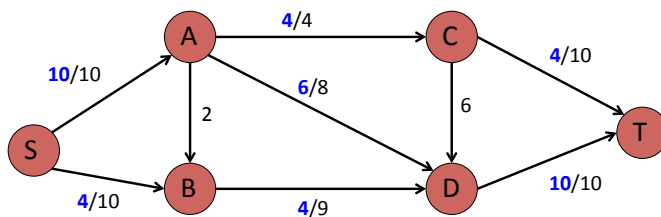


Are we done?
Is this the best we can do?

Maximum flow

For any cut where $s \in A$ and $t \in B$

- 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 A to B

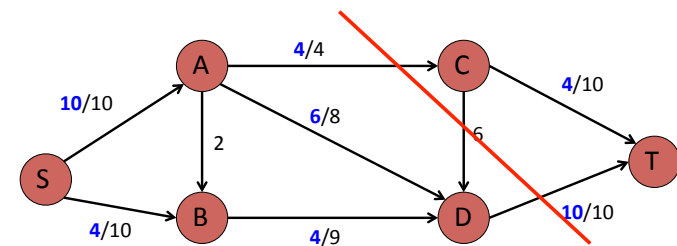


We can do no better than the minimum capacity cut!

Maximum flow

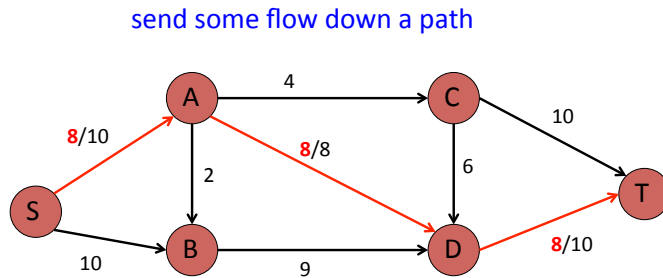
What is the minimum capacity cut for this graph?

Capacity = 10 + 4



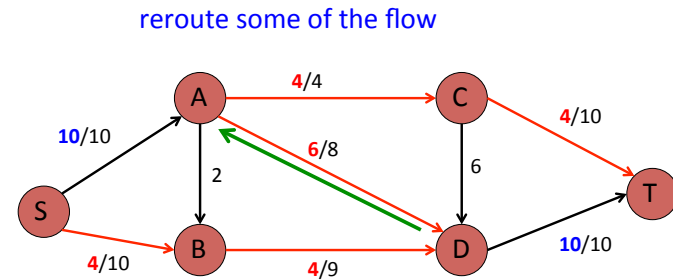
flow = minimum capacity, so we can do no better

Algorithm idea



How do we determine the path to send flow down?

Algorithm idea



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

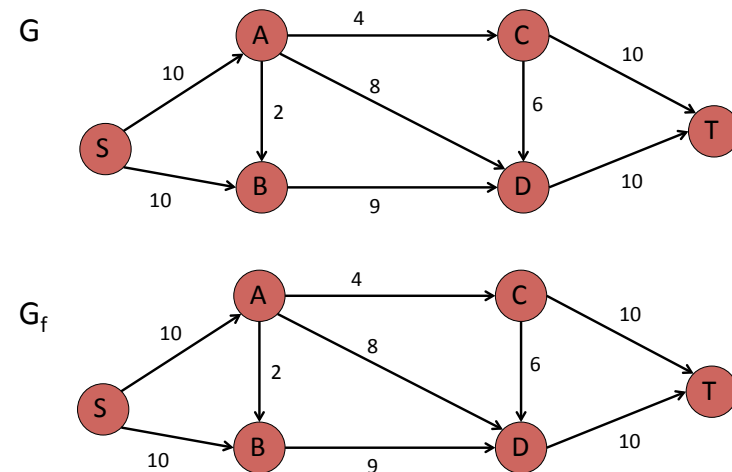
The residual graph

The **residual graph** G_f is constructed from G

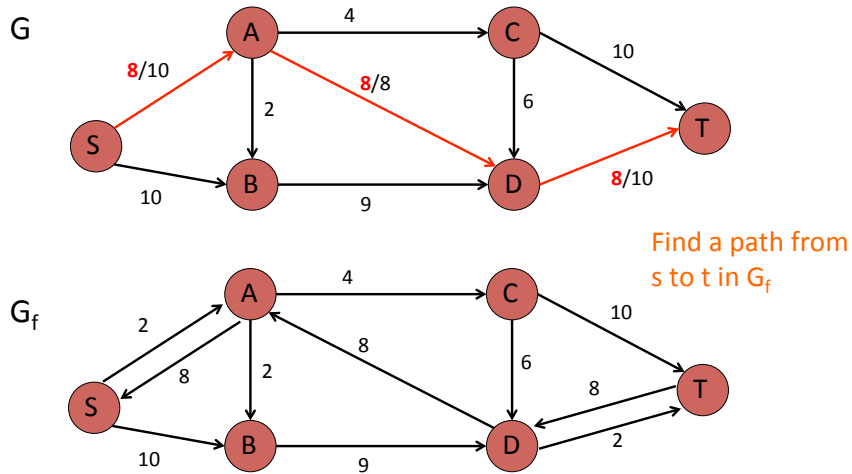
For each edge e in the original graph (G):

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

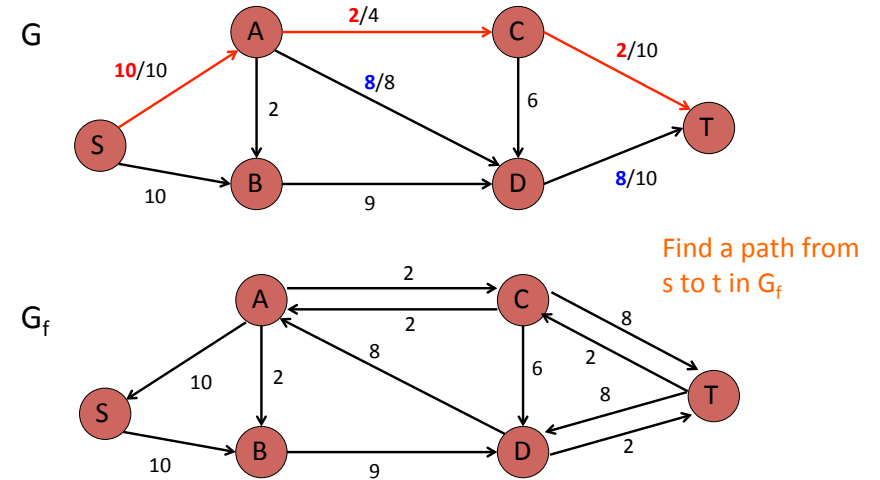
Algorithm idea



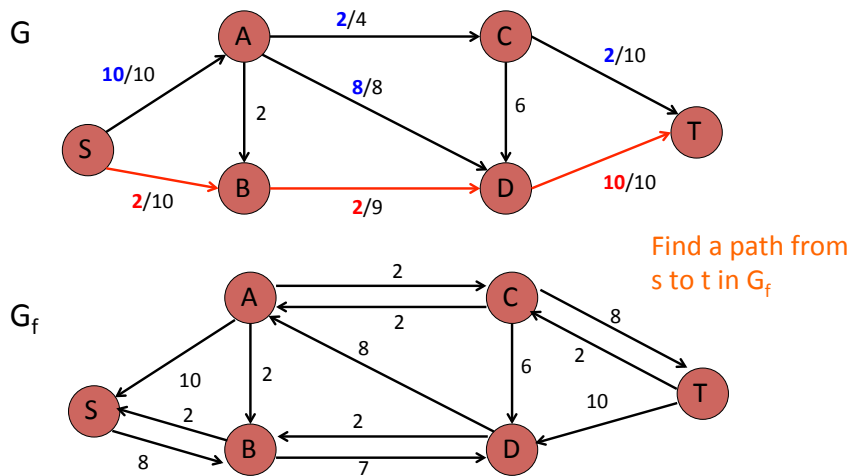
Algorithm idea



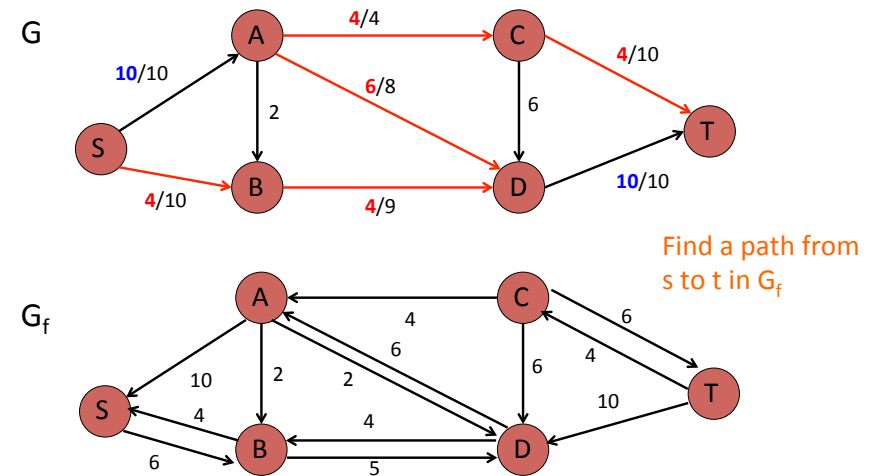
Algorithm idea



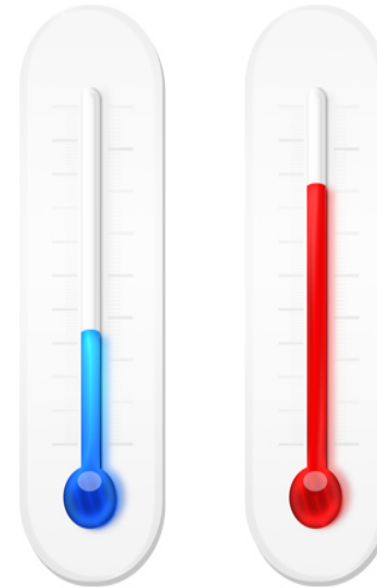
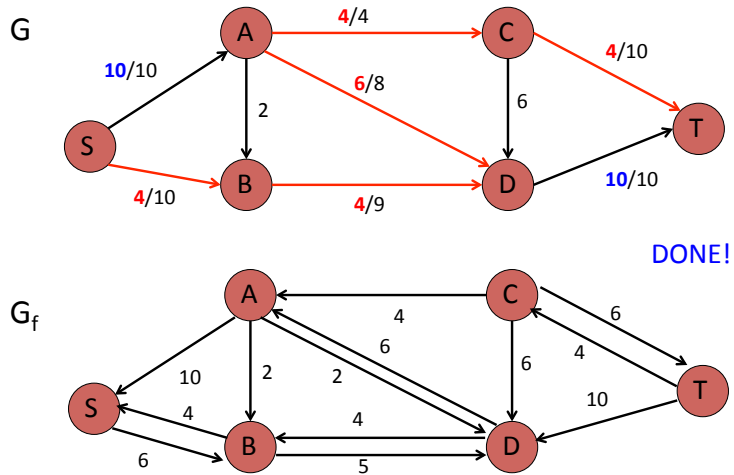
Algorithm idea



Algorithm idea



Algorithm idea



Ford-Fulkerson

Ford-Fulkerson(G, s, t)

flow = 0 for all edges

a simple path contains no repeated vertices

$G_f = \text{residualGraph}(G)$

while a simple path exists from s to t in G_f

send as much flow along the path as possible

$G_f = \text{residualGraph}(G)$

return flow

Ford-Fulkerson: runtime?

Ford-Fulkerson(G, s, t)

flow = 0 for all edges

$G_f = \text{residualGraph}(G)$

while a simple path exists from s to t in G_f

send as much flow along the path as possible

$G_f = \text{residualGraph}(G)$

return flow

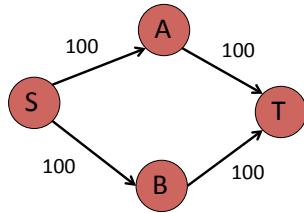
- max-flow!
- increases ever iteration
- integer capacities, so integer increases

Overall runtime? $O(\text{max-flow} * E)$

$O(\text{max-flow} * E)$

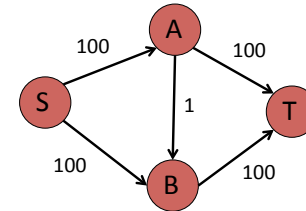
Can you construct a graph that *could get* this running time?

Hint:



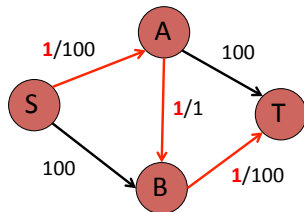
$O(\text{max-flow} * E)$

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



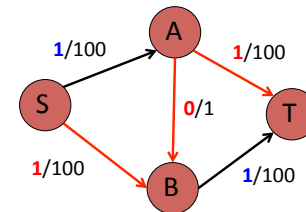
$O(\text{max-flow} * E)$

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



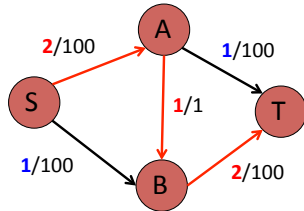
$O(\text{max-flow} * E)$

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



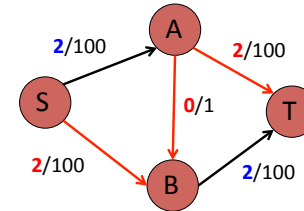
$O(\text{max-flow} * E)$

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



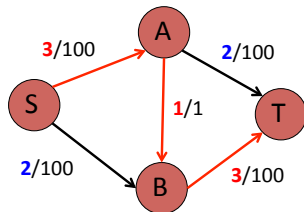
$O(\text{max-flow} * E)$

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



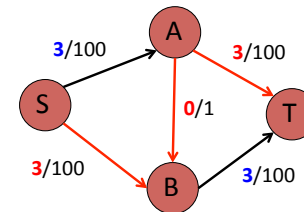
$O(\text{max-flow} * E)$

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



$O(\text{max-flow} * E)$

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



What is the problem here?
Could we do better?

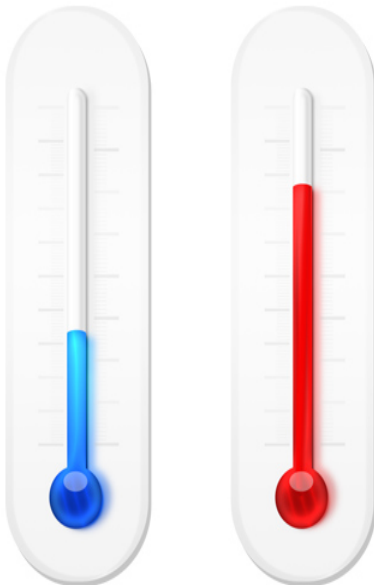
Faster variants

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
 - see the book for the proof
 - or <http://www.cs.cornell.edu/courses/CS4820/2011sp/handouts/edmondskarp.pdf>

preflow-push (aka push-relabel) algorithms

- $O(V^3)$



Other variations...

Method	Complexity
Linear programming	
Ford-Fulkerson algorithm	$O(E \max f)$
Edmonds-Karp algorithm	$O(VE^2)$
Dinitz blocking flow algorithm	$O(V^2E)$
General push-relabel maximum flow algorithm	$O(V^2E)$
Push-relabel algorithm with FIFO vertex selection rule	$O(V^3)$
Dinitz blocking flow algorithm with dynamic trees	$O(VE \log(V))$
Push-relabel algorithm with dynamic trees	$O(VE \log(V^2/E))$
Binary blocking flow algorithm [1]	$O(E \min(V^{2/3}, \sqrt{E}) \log(V^2/E) \log(U))$
MPM (Malhotra, Pramodh-Kumar and Maheshwari) algorithm	$O(V^3)$

TABLE I. POLYNOMIAL-TIME ALGORITHMS FOR THE MAXIMUM FLOW PROBLEM^a

Algorithm no.	Date	Discoverer	Running time	References
1	1969	Edmonds and Karp	$O(nm^2)$	[5]
2	1970	Dinic	$O(n^2m)$	[4]
3	1974	Karzanov	$O(n^2)$	[18]
4	1977	Cherkasky	$O(n^2m^{1/2})$	[3]
5	1978	Malhotra, Pramodh Kumar, and Maheshwari	$O(n^3)$	[21]
6	1978	Galil	$O(n^{3/2}m^{2/3})$	[11]
7	1978	Galil and Naamad; Shiloach	$O(nm(\log n)^2)$	[12, 25]
8	1980	Sleator and Tarjan	$O(nm \log n)$	[27, 28]
9	1982	Shiloach and Vishkin	$O(n^3)$	[26]
10	1983	Gabow	$O(nm \log U)$	[10]
11	1984	Tarjan	$O(n^3)$	[31]
12	1985	Goldberg	$O(n^3)$	[14]
13	1986	Goldberg and Tarjan	$O(nm \log(n^2/m))$	[16, 15]
14	1986	Aluja and Orlin	$O(nm + n^2 \log U)$	[1]

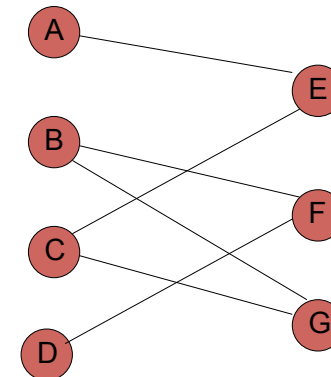
^a Algorithm 13 is presented in this paper.

http://akira.ruc.dk/~keld/teaching/algoritmedesign_f03/Artikler/08/Goldberg88.pdf

http://en.wikipedia.org/wiki/Maximum_flow

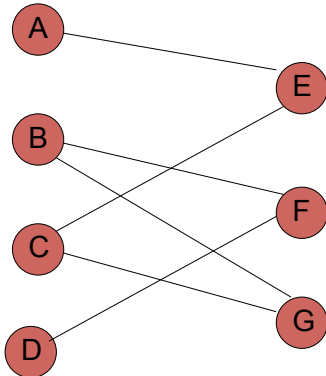
Application: bipartite graph matching

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 $v \in Y$



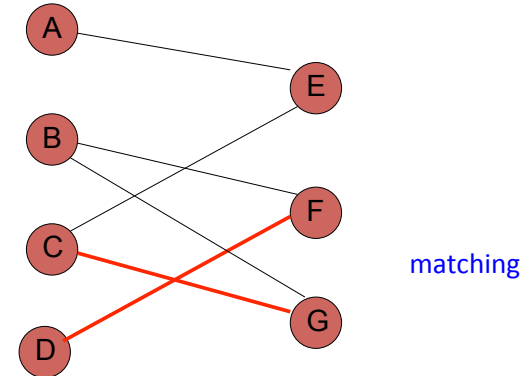
Application: bipartite graph matching

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



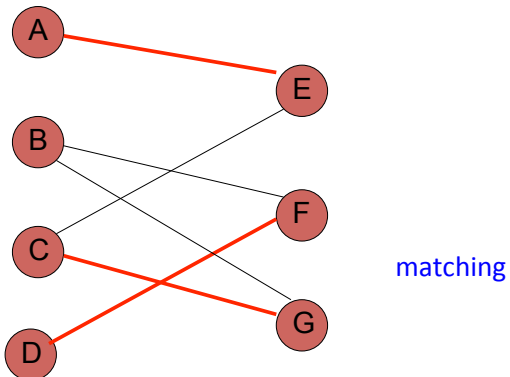
Application: bipartite graph matching

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



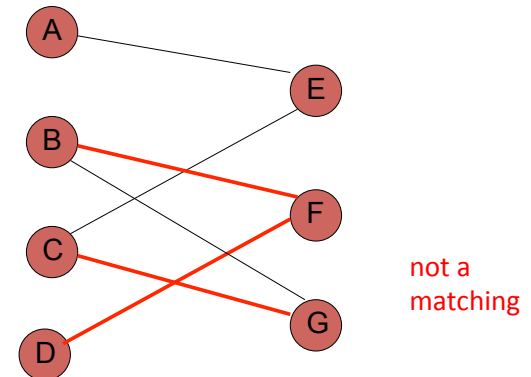
Application: bipartite graph matching

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



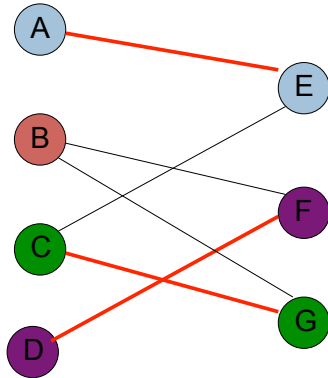
Application: bipartite graph matching

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

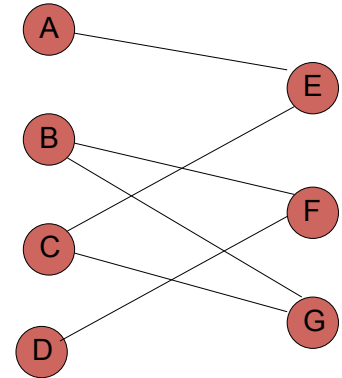


Application: bipartite graph matching

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

Where might this problem come up?

- CS 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

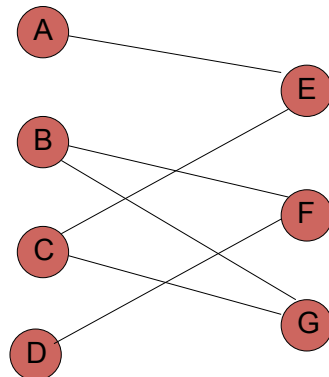


Application: bipartite graph matching

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

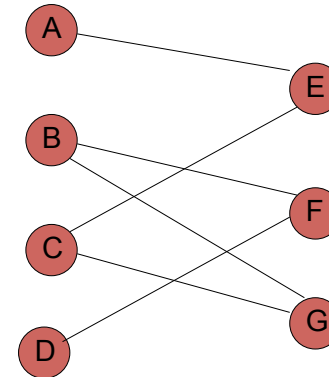
ideas?

- greedy?
- dynamic programming?



Application: bipartite graph matching

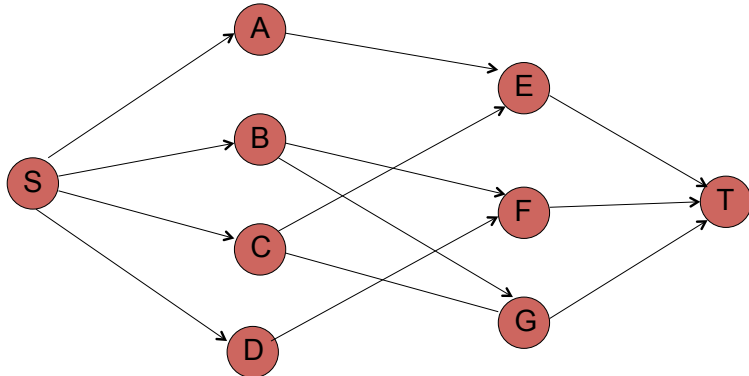
Setup as a flow problem:



Application: bipartite graph matching

Setup as a flow problem:

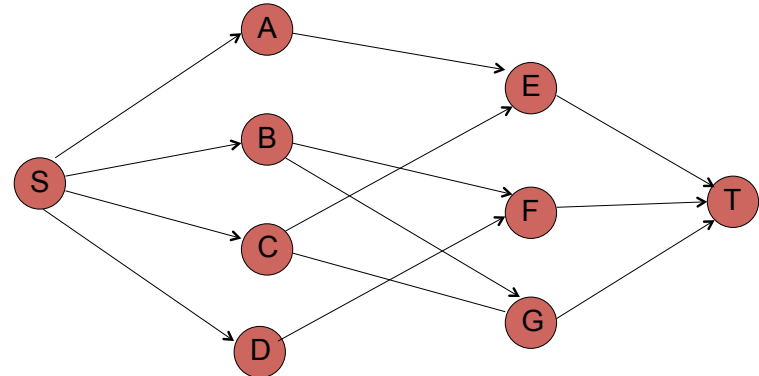
edge weights?



Application: bipartite graph matching

Setup as a flow problem:

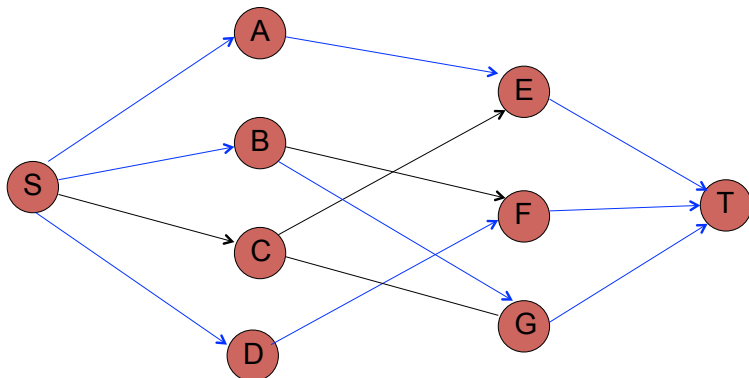
all edge weights are 1



Application: bipartite graph matching

Setup as a flow problem:

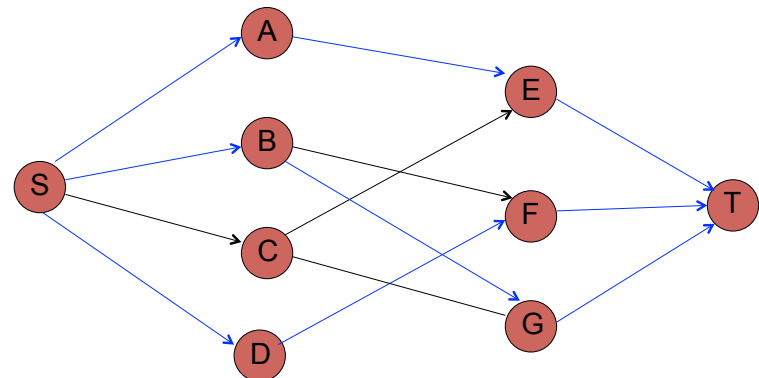
after we find the flow, how do we find the matching?



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: $O(\text{max-flow} * E)$
- Edmonds-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?

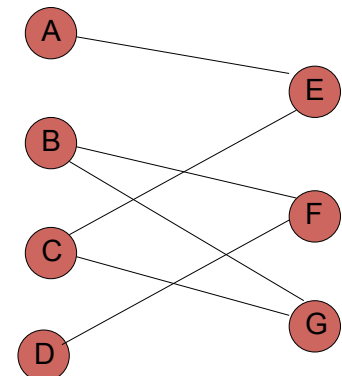
- Basic Ford-Fulkerson: $O(\text{max-flow} * E)$
 - max-flow = $O(V)$
 - $O(V E)$

Application: bipartite graph matching

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

- CS 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

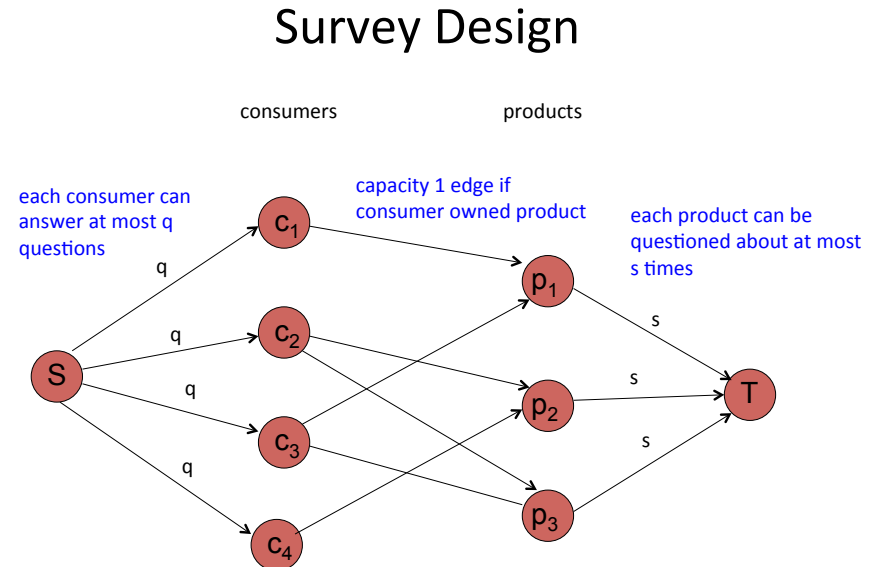


Survey Design

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

Is it correct?

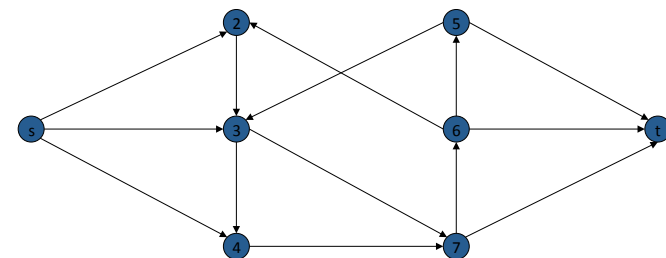
- Each of the comments above the flow graph match the problem constraints
- max-flow finds the maximum matching, given the problem constraints

What is the run-time?

- Basic Ford-Fulkerson: $O(\text{max-flow} * E)$
- Edmonds-Karp: $O(V E^2)$
- Preflow-push: $O(V^3)$

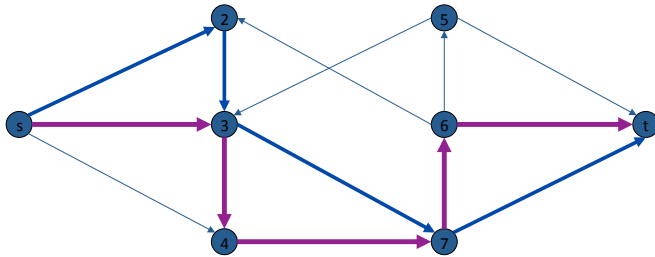
Edge Disjoint Paths

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



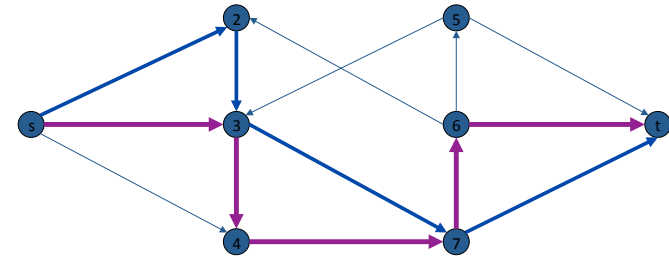
Edge Disjoint Paths

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?

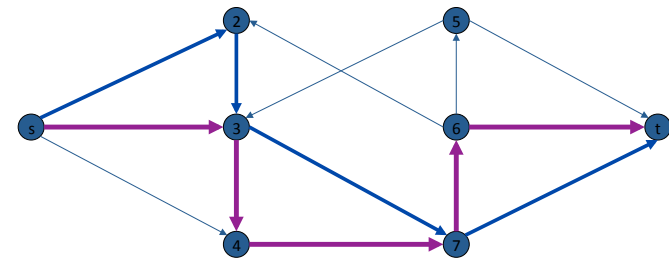
Edge Disjoint Paths Problem

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?

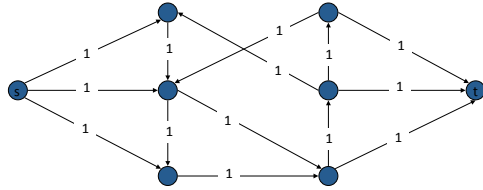
- edges are unique resources (e.g. communications, transportation, etc.)
- how many *concurrent (non-conflicting)* paths do we have from s to t

Algorithm ideas?



Edge Disjoint Paths

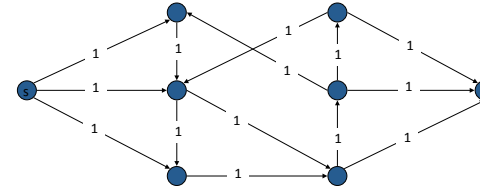
Max flow formulation: assign unit capacity to every edge



What does the max flow represent?
Why?

Edge Disjoint Paths

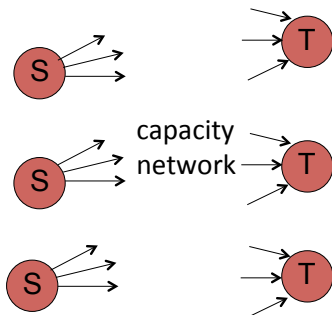
Max flow formulation: assign unit capacity to every edge



- 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

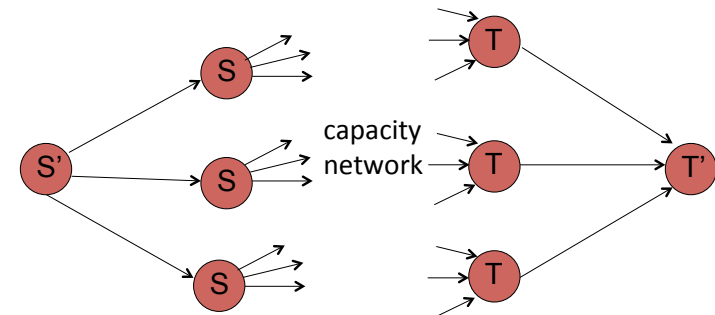
Max-flow variations

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



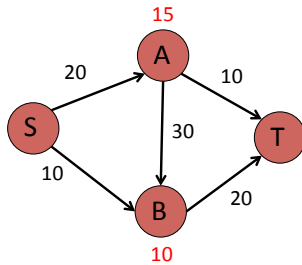
Max-flow variations

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



Max-flow variations

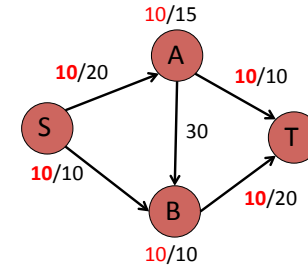
Vertex capacities: in addition to having edge capacities we can also restrict the amount of flow through each vertex



What is the max-flow

Max-flow variations

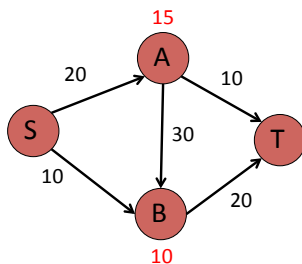
Vertex capacities: in addition to having edge capacities we can also restrict the amount of flow through each vertex



20 units

Max-flow variations

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

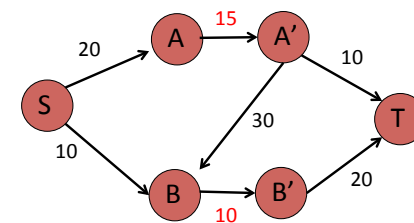


How can we solve this problem?

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?

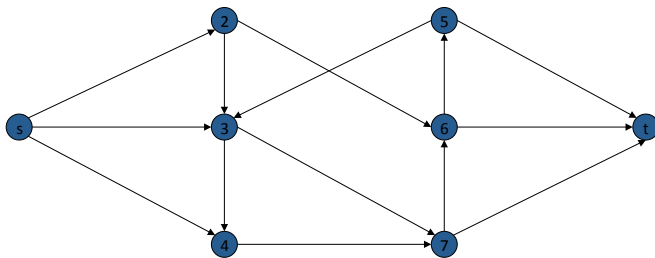
Max-flow variations

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

More problems:
maximum independent path

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



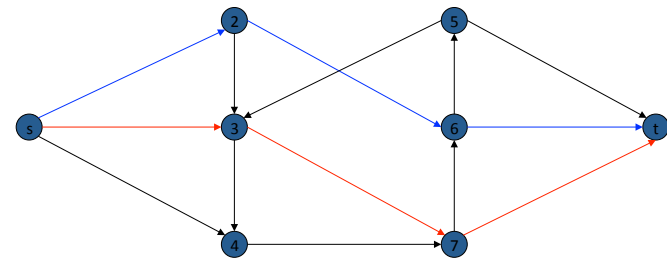
Max-flow variations

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

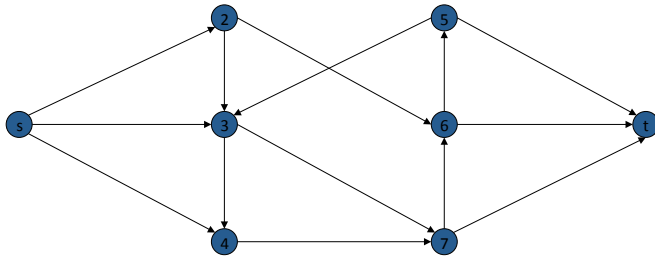
Two paths are *independent* if they have no *vertices* in common



More problems: maximum independent path

Find the maximum number of independent paths

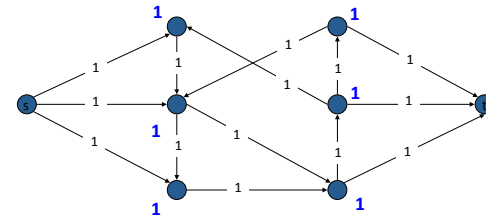
Ideas?



maximum independent path

Max flow formulation:

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



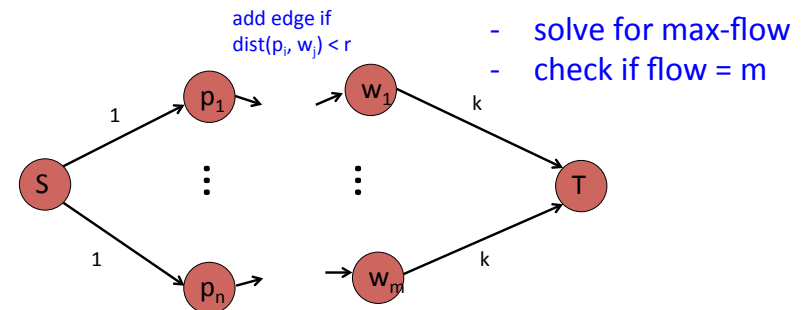
Same idea as the maximum edge-disjoint paths, but now we also constrain the vertices

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 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 $\text{dist}(p_i, w_j) < r$ then add an edge from p_i to w_j with weight 1 (where dist is euclidean distance)
- add edges $s \rightarrow p_i$ with weight 1
- add edges $w_j \rightarrow t$ with weight k



Correctness

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

if $\text{dist}(p_i, w_j) < r$ then add an edge from p_i to w_j with weight 1 (where dist is euclidean distance)

- only people able to connect to node could have flow

add edges $s \rightarrow p_i$ with weight 1

- each person can only connect to one wireless node

add edges $w_j \rightarrow t$ 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
- $O(VE^2) = O((m+n)m^2n^2)$: Edmonds-Karp
- $O(V^3) = O((m+n)^3)$: preflow-push variant