CS 344: Design and Analysis of Computer Algorithms

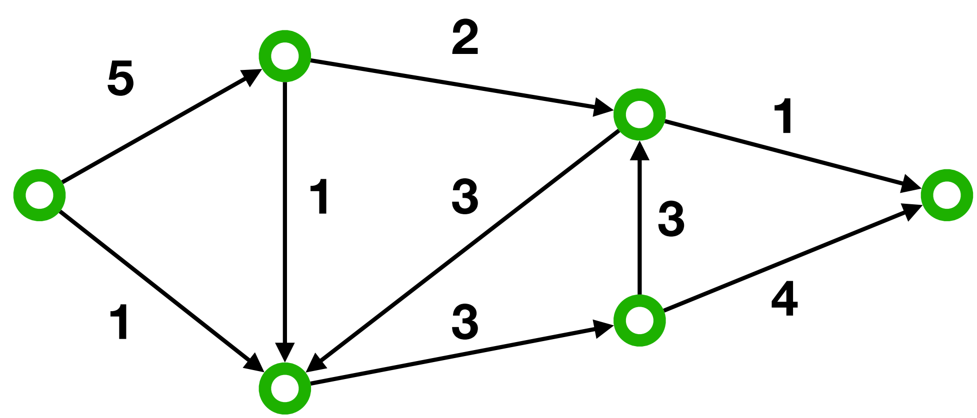
(Spring 2022 — Sections 5,6,7,8)

Lecture 21: Network Flow

The Network Flow Problem

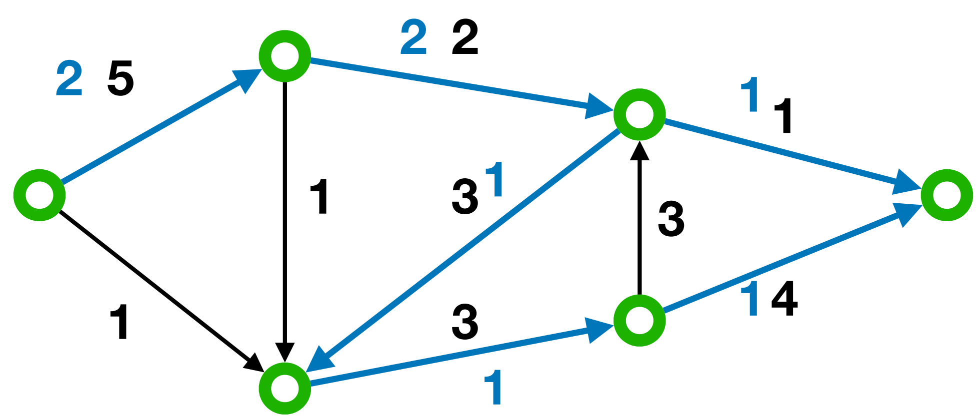
Motivation

- Think of a collection of pipes
- A source s and a sink t — the source produces some material, say, water, and the sink consumes it
- Each pipe e can carry a certain amount of water c_e at any point of time
- Maximize the rate of sending water from source to sink



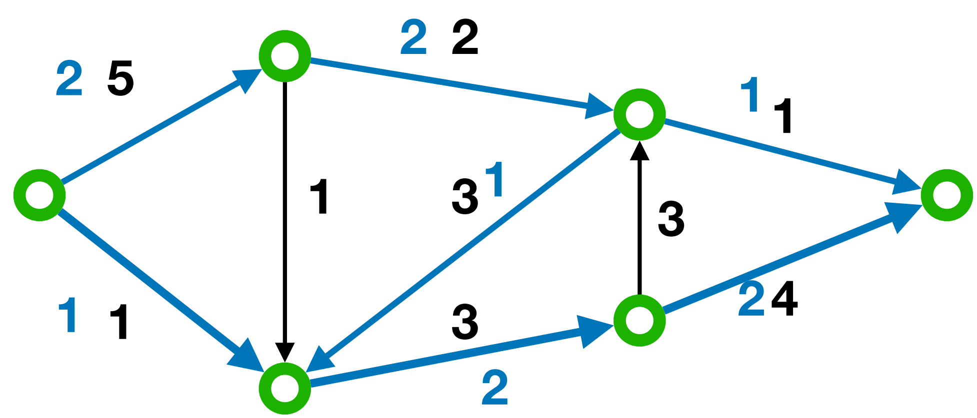
Motivation

- Think of a collection of pipes
- A source s and a sink t — the source produces some material, say, water, and the sink consumes it
- Each pipe e can carry a certain amount of water c_e at any point of time
- Maximize the rate of sending water from source to sink



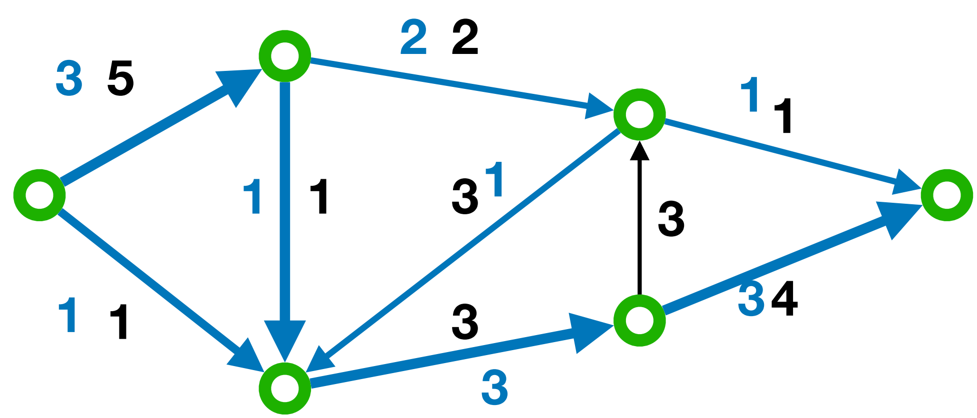
Motivation

- Think of a collection of pipes
- A source s and a sink t — the source produces some material, say, water, and the sink consumes it
- Each pipe e can carry a certain amount of water c_e at any point of time
- Maximize the rate of sending water from source to sink



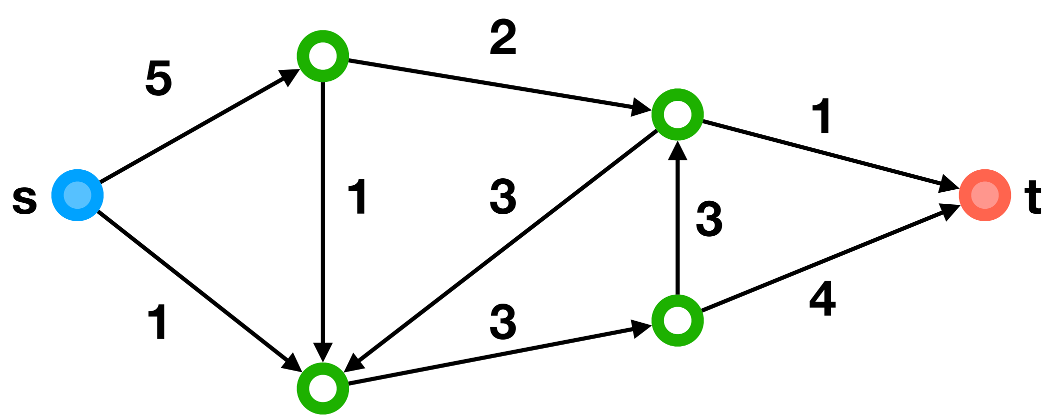
Motivation

- Think of a collection of pipes
- A source s and a sink t — the source produces some material, say, water, and the sink consumes it
- Each pipe e can carry a certain amount of water c_e at any point of time
- Maximize the rate of sending water from source to sink



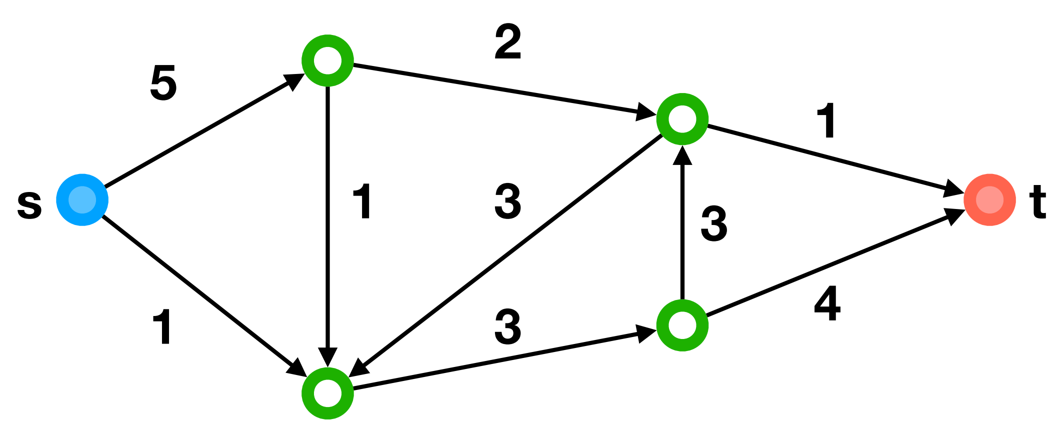
Networks

- Directed graph G=(V,E)
- A source vertex s and a sink vertex t
- Capacity c_e on any edge e



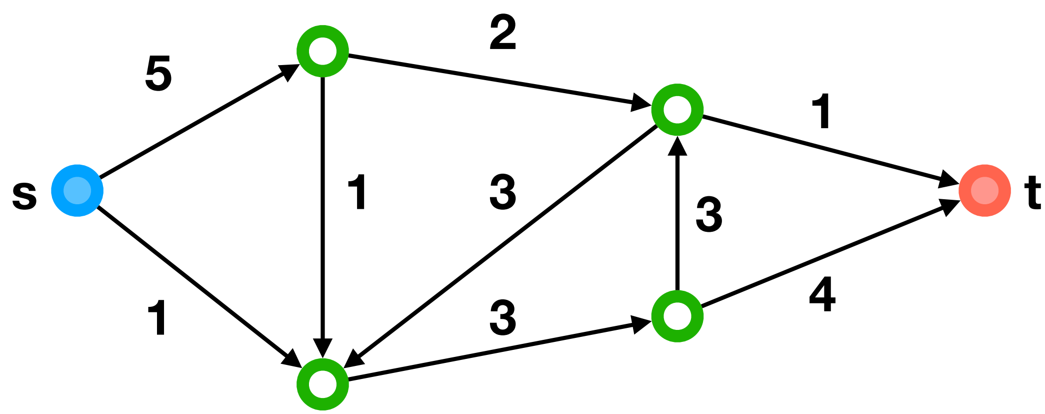
Flow

- A function $f: V \times V \to \mathbb{R}$ with the following properties:
- **Capacity constraint:**
  - for any edge $e = (u, v) : f(u, v) \leq c_e$ — if there is no edge from u to v, then $f(u, v) = 0$



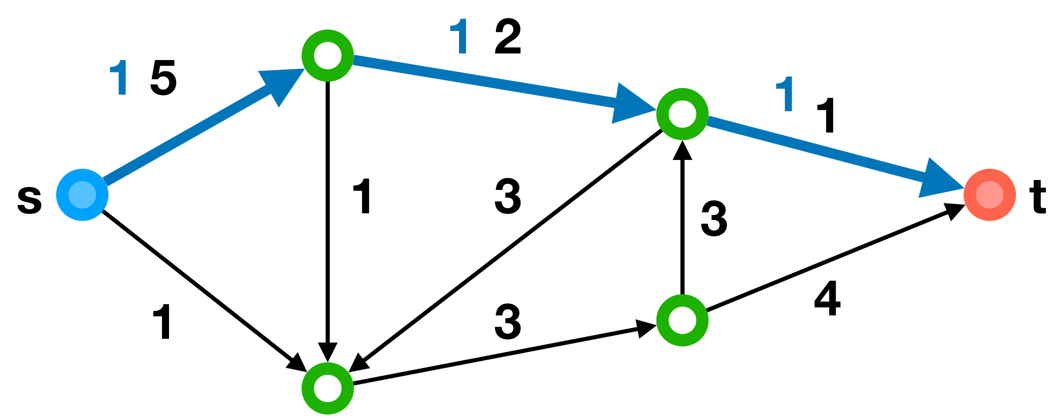
Flow

- A function $f : V \times V \to \mathbb{R}$ with the following properties:
- **Capacity constraint**
- **Preservation of flow:** for any vertex $v \in V - \{s, t\}$:
$$\sum_{w \in V} f(w, v) = \sum_{w \in V} f(v, w)$$



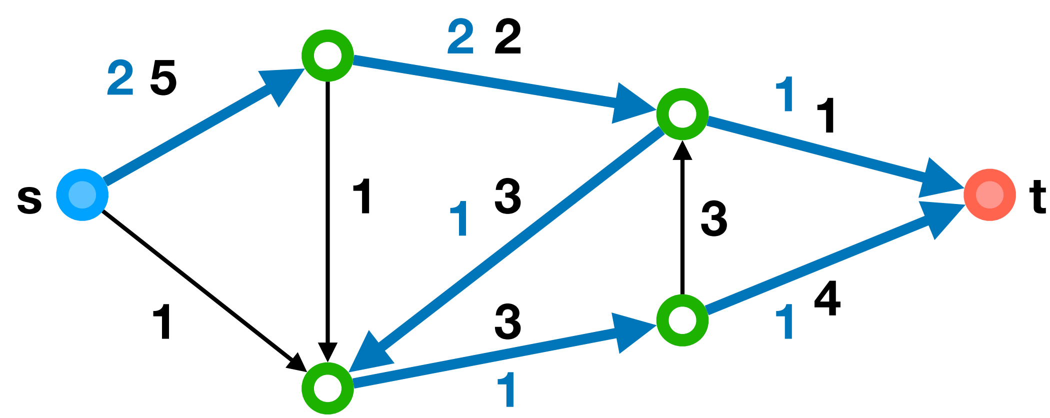
Flow: Example

- A function $f : V \times V \rightarrow \mathbb{R}$ with the following properties:
- **Capacity constraint**
- **Preservation of flow**



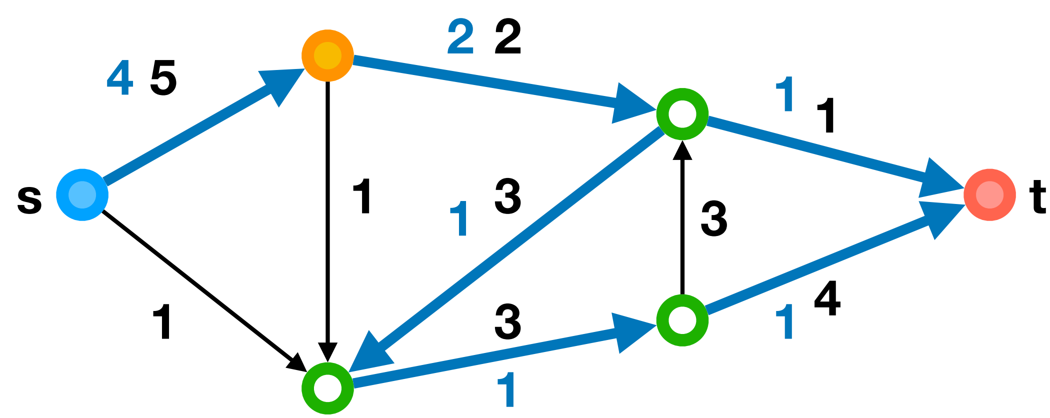
Flow: Example

- A function $f : V \times V \to \mathbb{R}$ with the following properties:
- **Capacity constraint**
- **Preservation of flow**



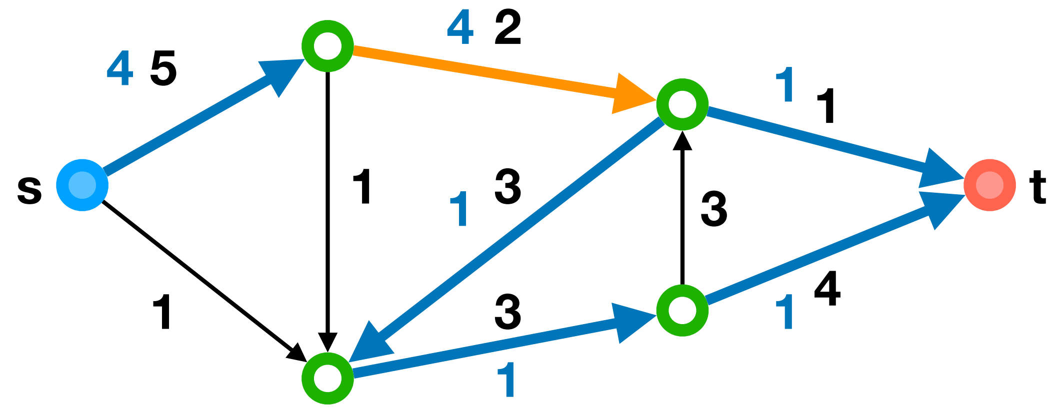
Flow: Example

- A function $f : V \times V \to \mathbb{R}$ with the following properties:
- **Capacity constraint**
- **Preservation of flow**



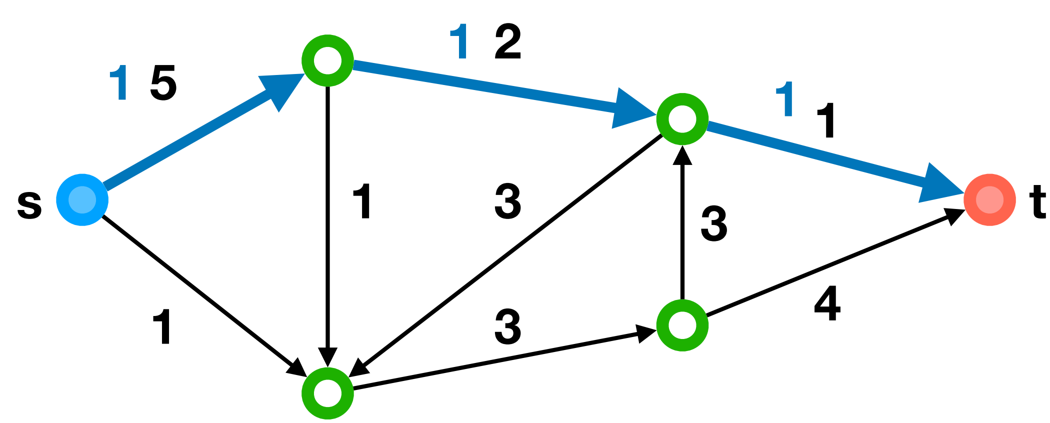
Flow: Example

- A function $f : V \times V \to \mathbb{R}$ with the following properties:
- **Capacity constraint**
- **Preservation of flow**



Flow

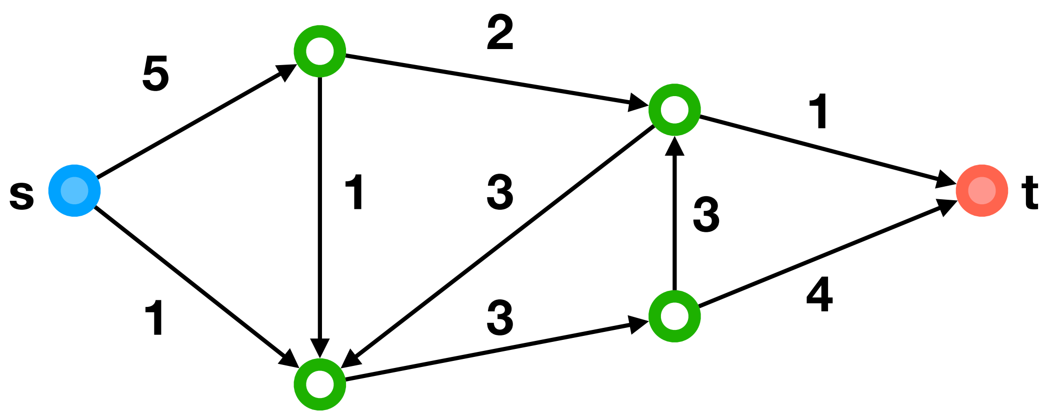
- A function $f : V \times V \to \mathbb{R}$ with the following properties:
- **Capacity constraint**
- **Preservation of flow**
- **Value** of a flow f: amount of flow leaving s = $\sum_{v \in V} f(s, v)$



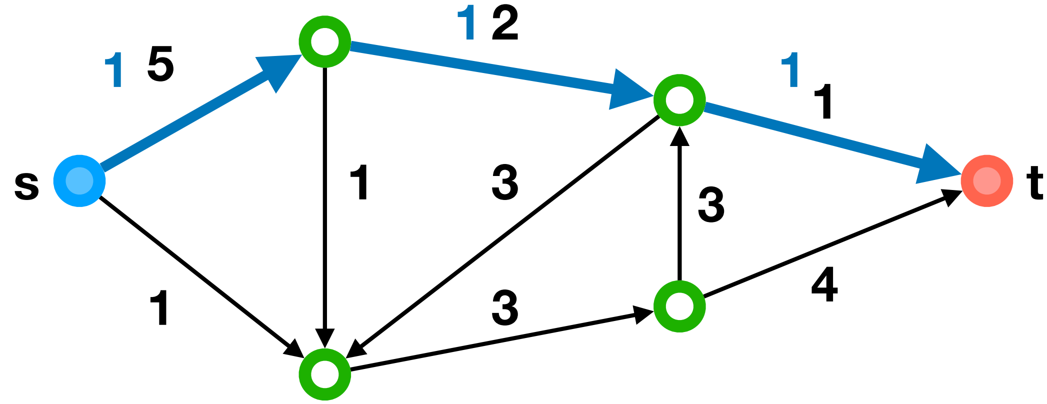
Network Flow Problem

- Network flow (or maximum flow) problem:
- **Input:**
  - A network $G=(V,E)$ with edge-capacities and a source and a sink
- Output:
  - Find a flow with largest value in G

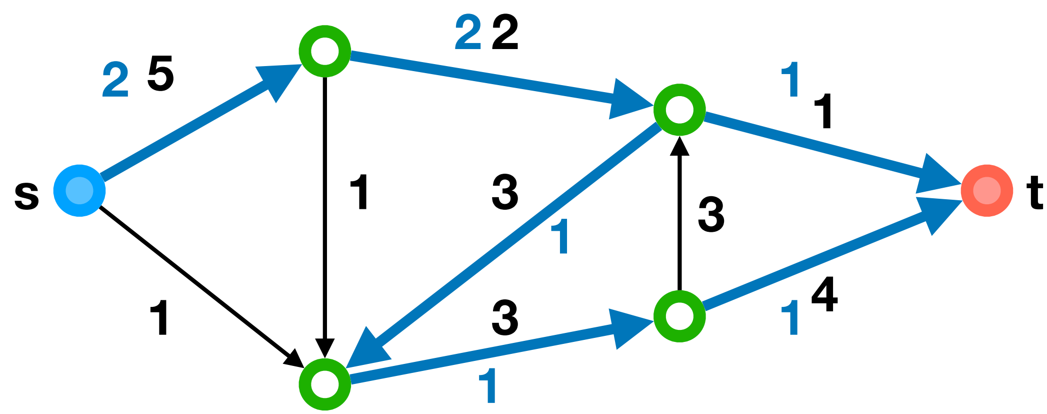
Network Flow Problem: Example



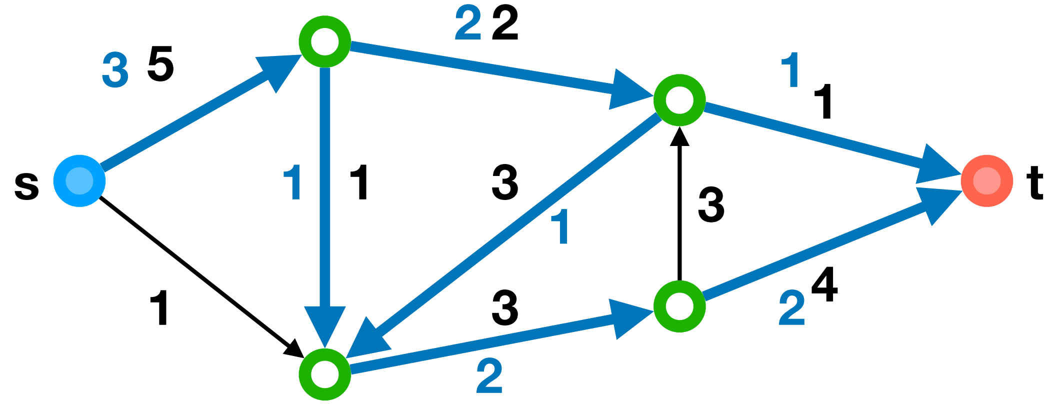
Network Flow Problem: Example



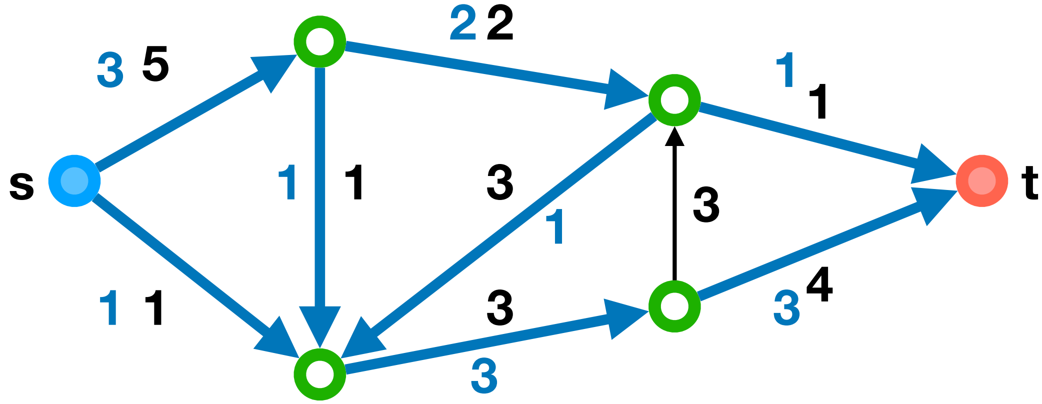
Network Flow Problem: Example



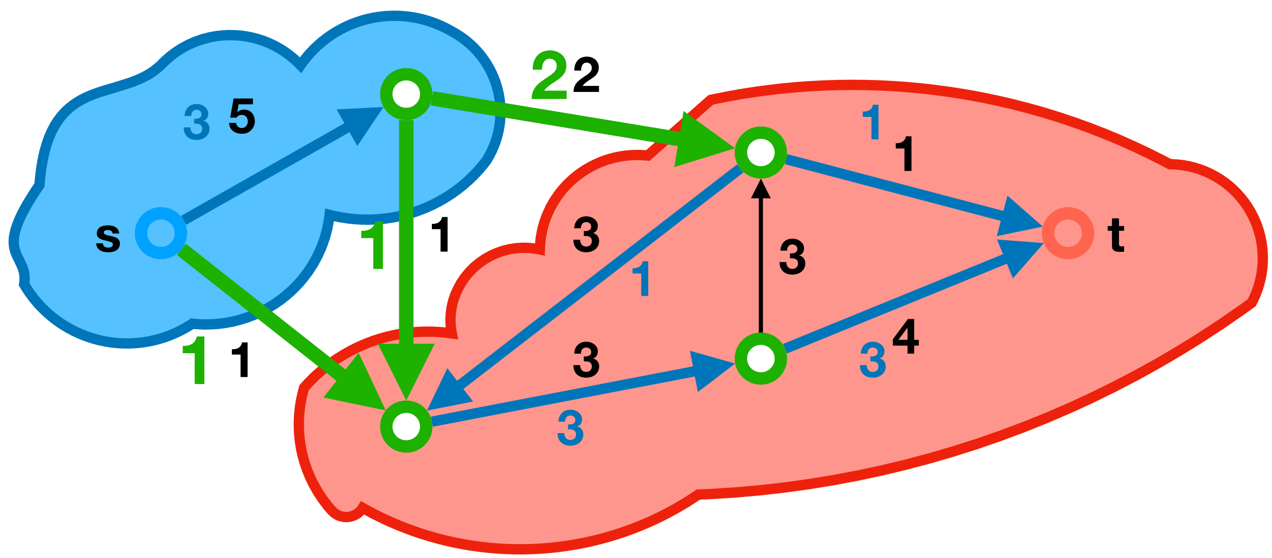
Network Flow Problem: Example



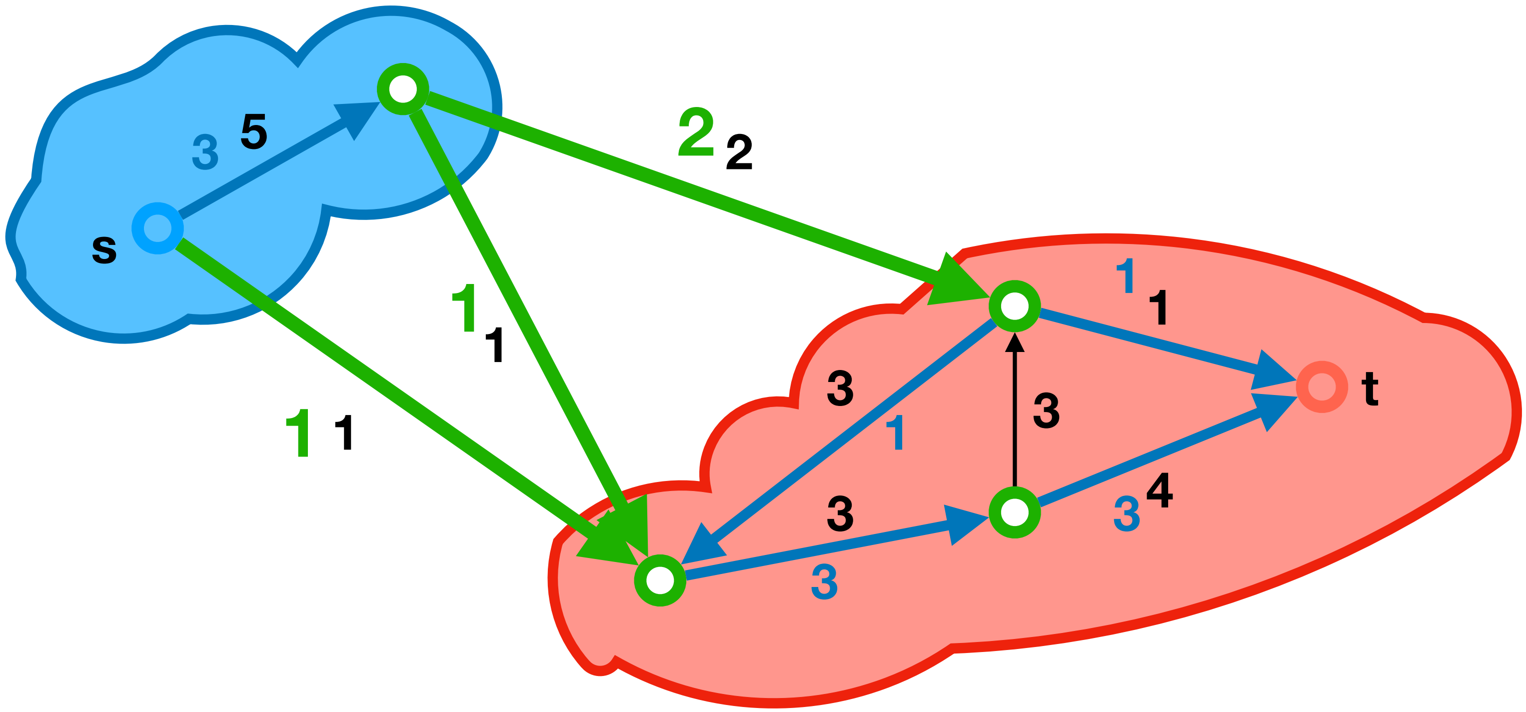
Network Flow Problem: Example



Network Flow Problem: Example



Network Flow Problem: Example



Algorithms for Network Flow

Algorithms

- So how do we solve the network flow problem?
- There are numerous algorithms for the problem:

Algorithms

- So how do we solve the network flow problem?

Ford-Fulkerson   Edmond-Karp   ...   Goldberg-Tarjan   ...   Orlin

...   Madry   ...   Lee-Sidford   ...   Liu-Sidford

Algorithms

- So how do we solve the network flow problem?

Ford-Fulkerson — 1956

Edmond-Karp — 1972

...

Goldberg-Tarjan — 1986

...

Orlin — 2012

...

Madry — 2013

...

Lee-Sidford — 2014

...

Liu-Sidford — 2020

Algorithms

- So how do we solve the network flow problem?

Algorithm	Year
Ford-Fulkerson	1956
Edmond-Karp	1972
...	
Goldberg-Tarjan	1986
...	
Orlin	2012
...	
Madry	2013
...	
Lee-Sidford	2014
...	
Liu-Sidford	2020
Chen-Kyng-Liu-Peng-Gotenberg-Sachdeva	2022

Algorithms

- We will NOT go through any of these algorithms
  - The early ones are not particularly too complicated (the latter ones definitely are!)
  - But they are still considerably more challenging than everything we covered so far
- Instead, we will see many important applications of network flow
- We use graph reduction so we can use ANY algorithm for network flow for solving these problems

Pointers

- If you are interested just to see how complicated these recent algorithms can be:

- Breakthrough result that solves the problem in $O(m^{1+o(1)})$ time

  - https://arxiv.org/pdf/2203.00671.pdf

  - https://www.youtube.com/watch?v=KsMtVthpkzI

Ford-Fulkerson

- For concreteness, we stick with the first and simplest network flow algorithm

Ford-Fulkerson

- For concreteness, we stick with the first and simplest network flow algorithm

- **Ford-Fulkerson Algorithm:**

  - Solves maximum flow in $O(m \cdot F)$ time where F is the value of maximum flow from s to t.

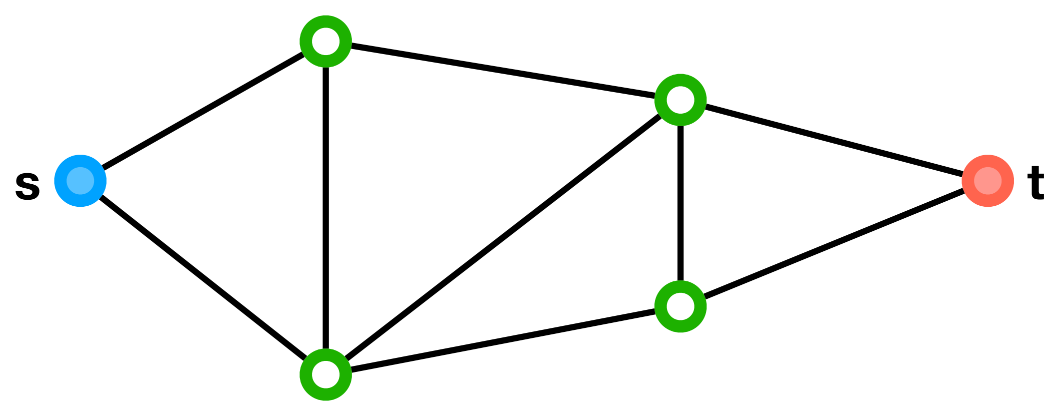
Application I: Edge-Disjoint Paths

Edge-Disjoint Paths Problem

- Input:
  - An undirected graph G=(V,E)
  - Two vertices s and t
- Output:
  - Maximum number of edge-disjoint paths between s and t

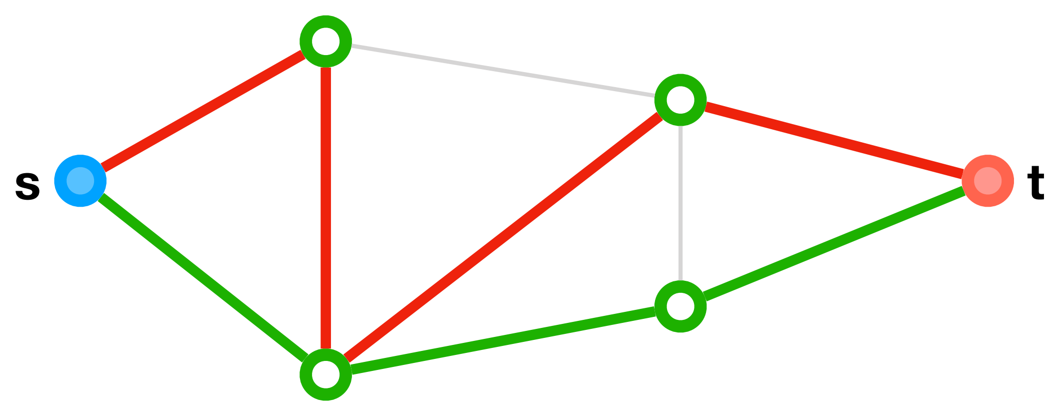
Edge-Disjoint Paths Problem

- Input:
  - An undirected graph G=(V,E)
  - Two vertices s and t
- Output:
  - Maximum number of edge-disjoint paths between s and t



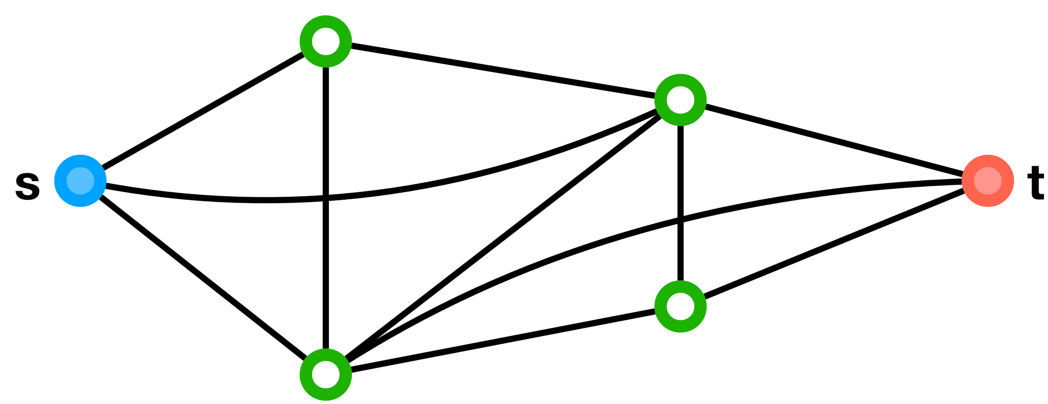
Edge-Disjoint Paths Problem

- Input:
  - An undirected graph G=(V,E)
  - Two vertices s and t
- Output:
  - Maximum number of edge-disjoint paths between s and t



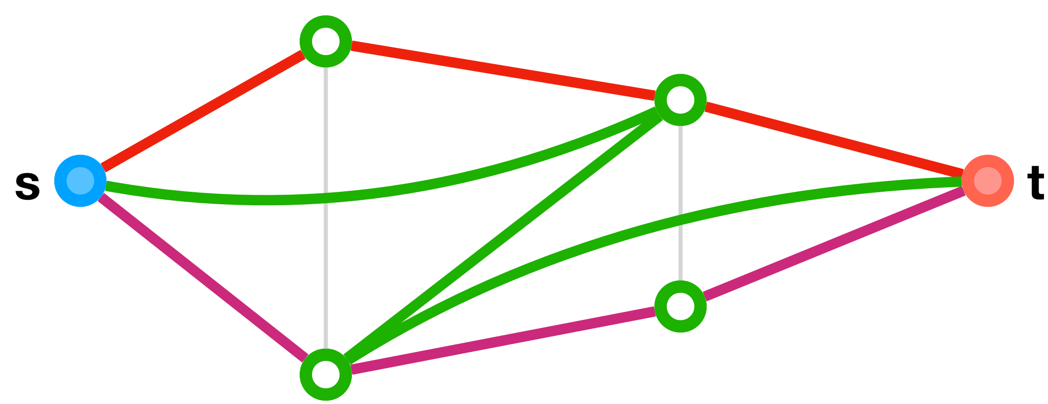
Edge-Disjoint Paths Problem

- Input:
  - An undirected graph G=(V,E)
  - Two vertices s and t
- Output:
  - Maximum number of edge-disjoint paths between s and t



Edge-Disjoint Paths Problem

- Input:
  - An undirected graph G=(V,E)
  - Two vertices s and t
- Output:
  - Maximum number of edge-disjoint paths between s and t



Edge-Disjoint Paths Problem

- Input:
  - An undirected graph $G=(V,E)$
  - Two vertices s and t
- Output:
  - Maximum number of edge-disjoint paths between s and t
- Application: Fault-Tolerance

Reduction to Network Flow

- Create a network $G' = (V', E')$ as follows:
  - Vertices are the same as G
  - For any undirected edge {u,v} in G, add both directed edges (u,v) and (v,u) in G' with capacity 1
  - Let source be s and sink be t

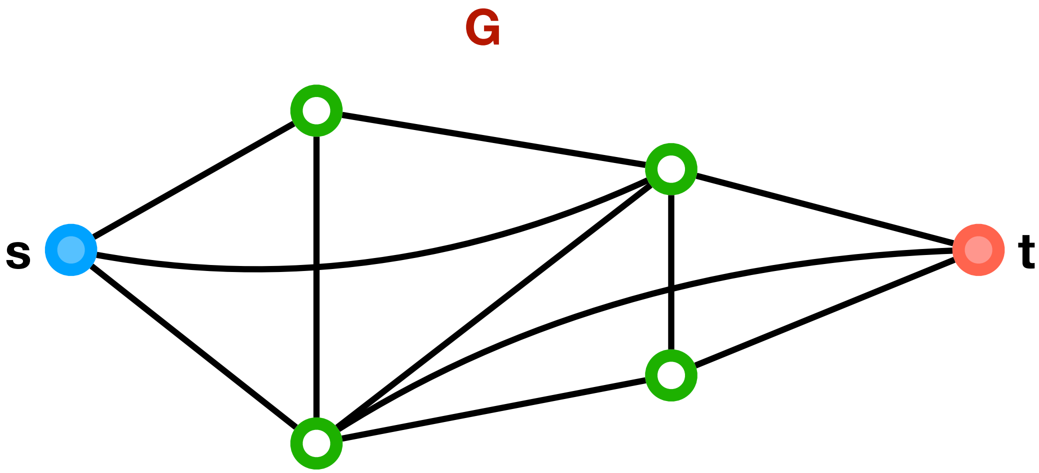
Reduction to Network Flow

- Create a network $G' = (V', E')$ as follows:
  - Vertices are the same as G
  - For any undirected edge {u,v} in G, add both directed edges (u,v) and (v,u) in G' with capacity 1
  - Let source be s and sink be t
- Compute a maximum flow f in G'

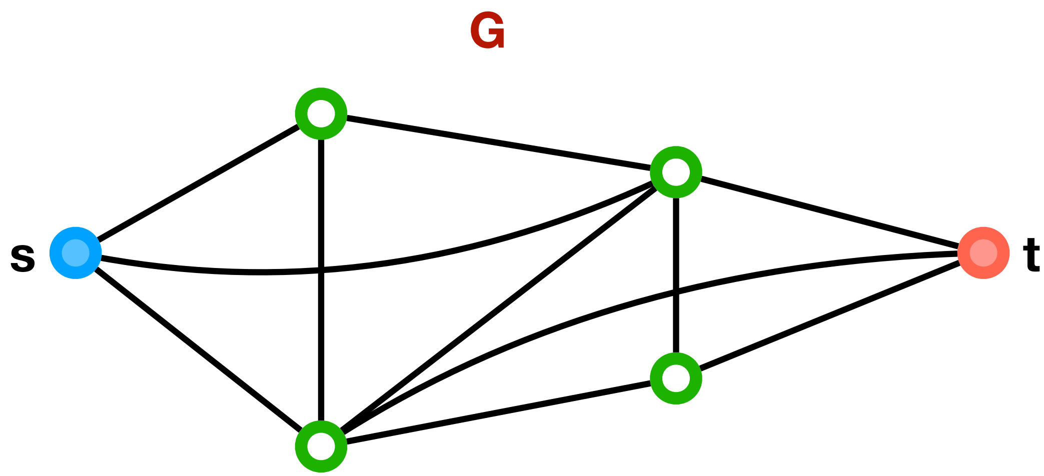
Reduction to Network Flow

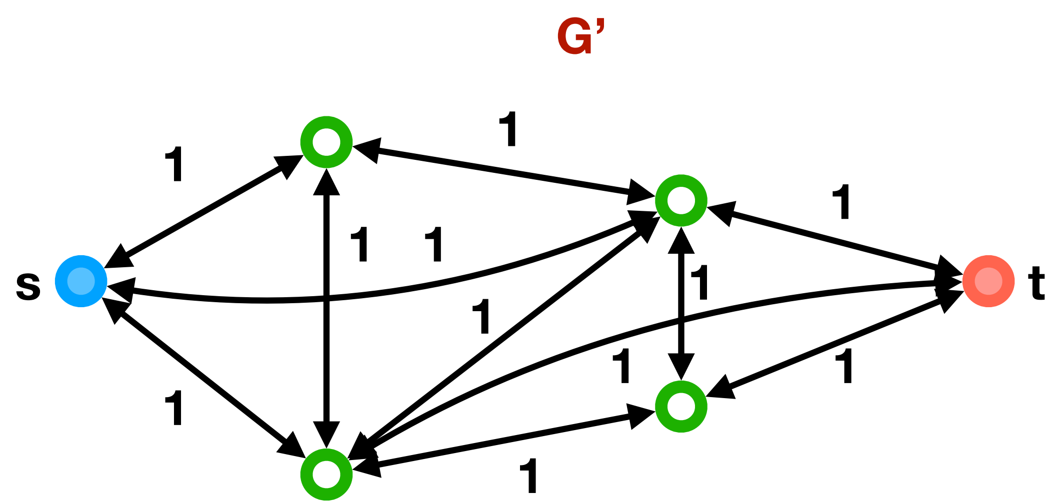
- Create a network $G' = (V', E')$ as follows:
  - Vertices are the same as G
  - For any undirected edge {u,v} in G, add both directed edges (u,v) and (v,u) in G' with capacity 1
  - Let source be s and sink be t
- Compute a maximum flow f in G'
- Return the value of f as the maximum number of edge-disjoint paths possible in G

Example

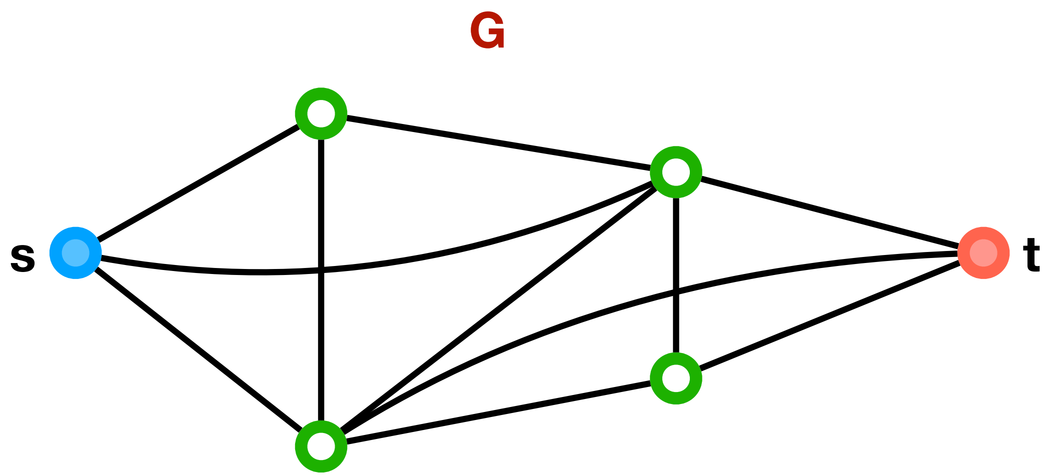


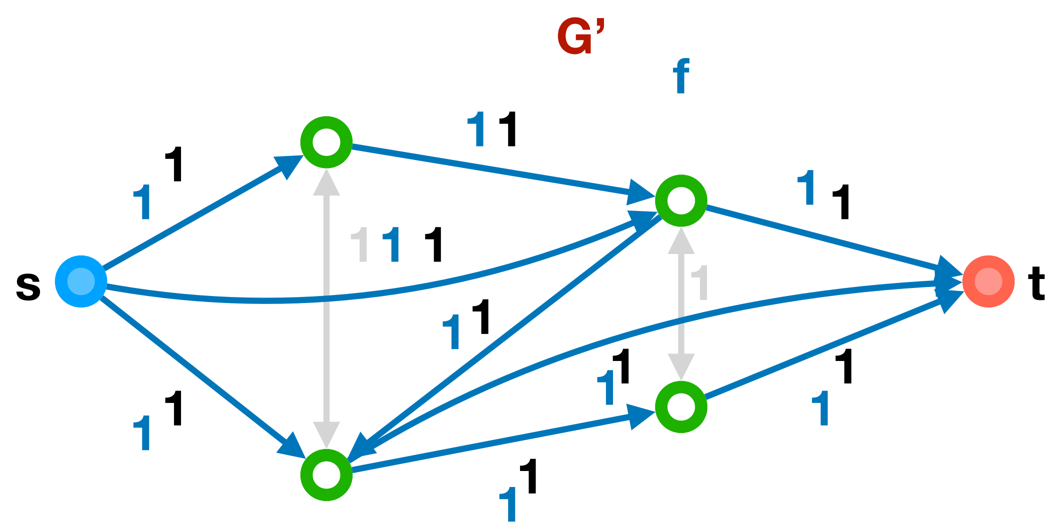
Example



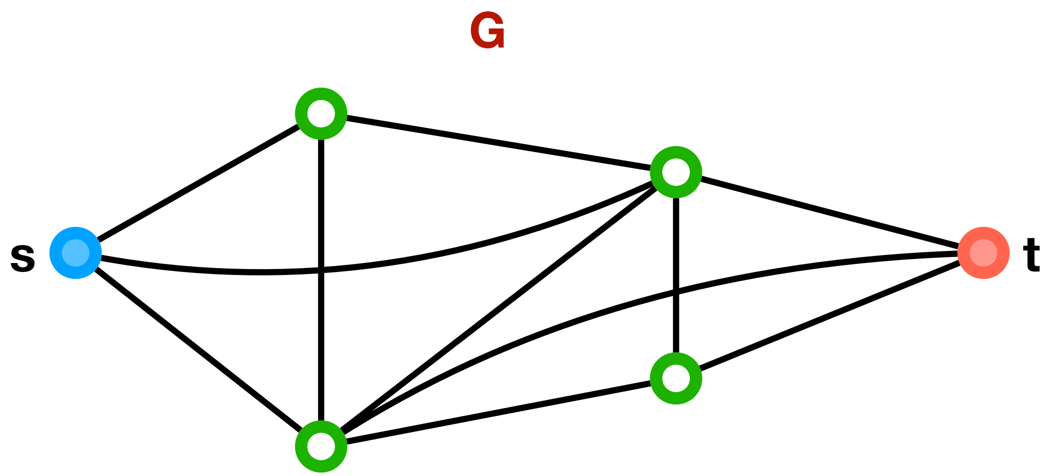


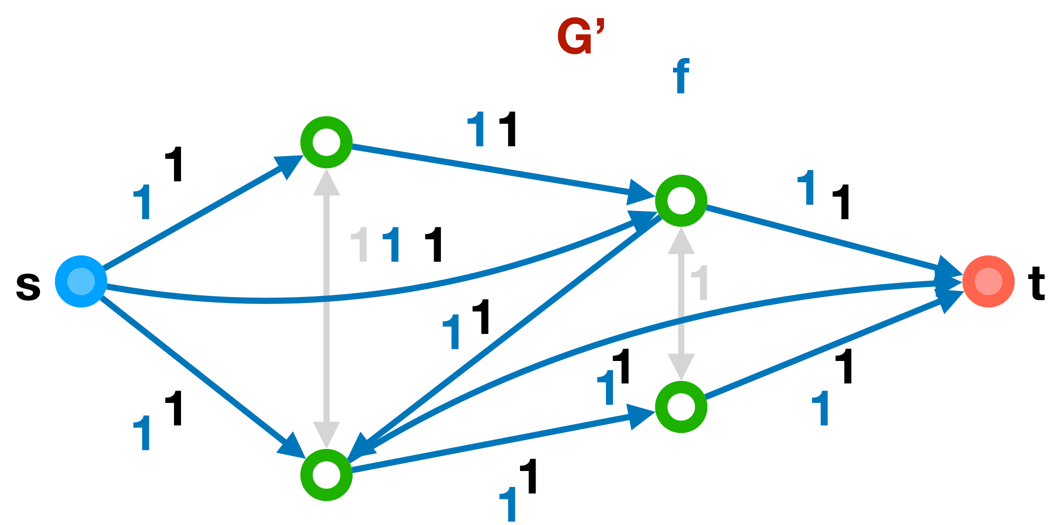
Example





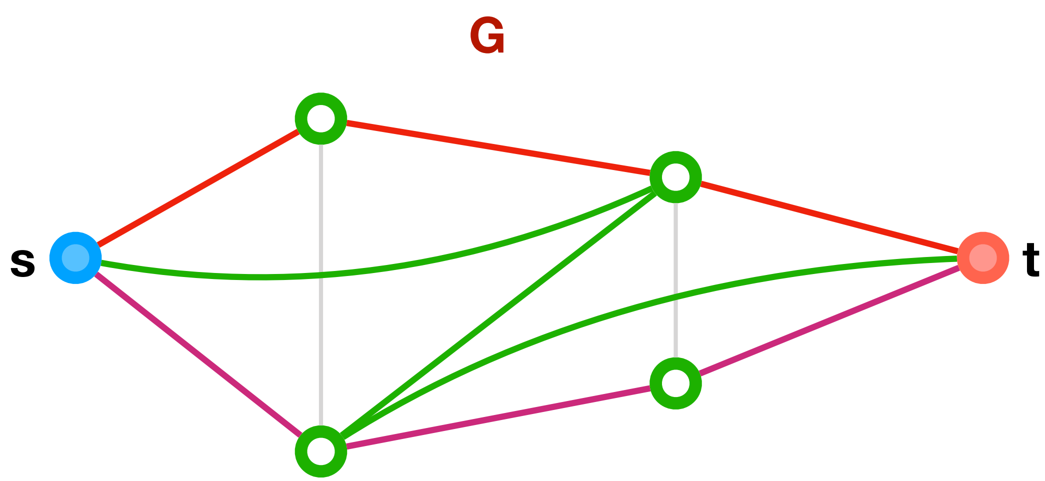
Example

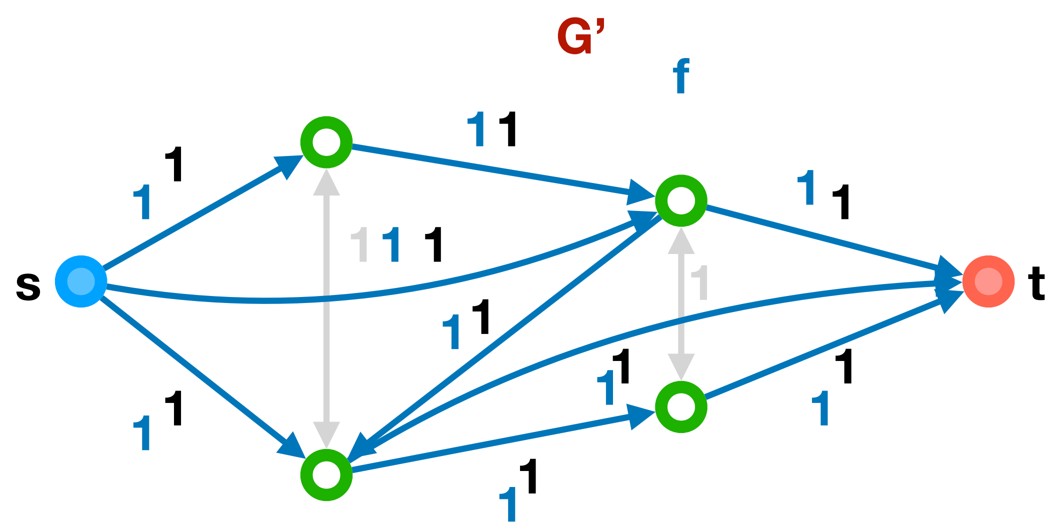




Value of f = 3

Example





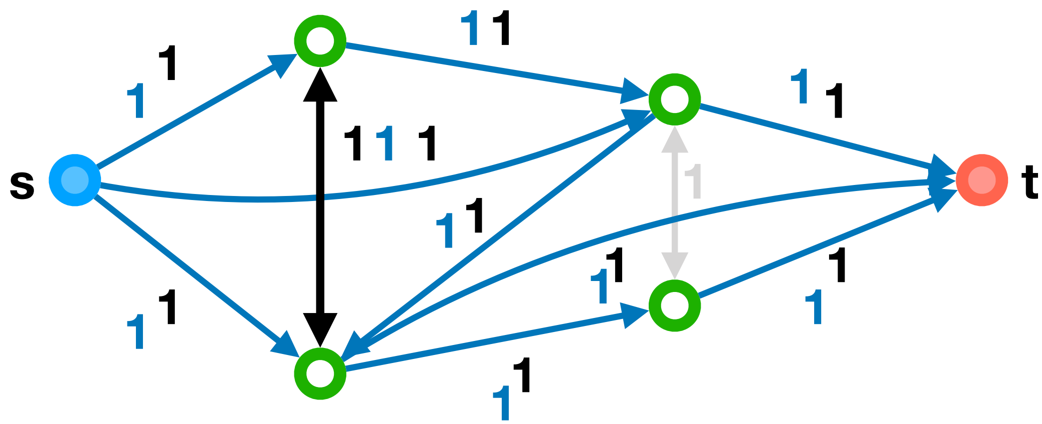
Value of f = 3

Detour

- Can Ford-Fulkerson pick both direction of an undirected edge?

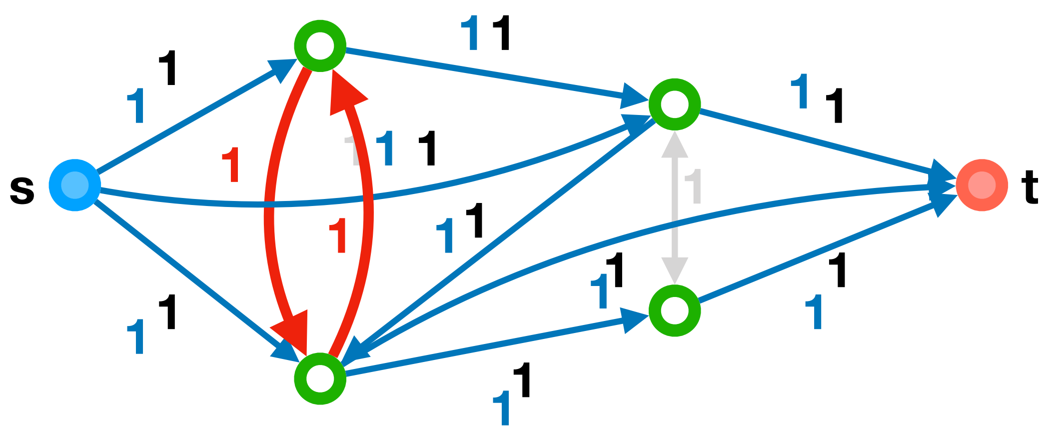
Detour

- Can Ford-Fulkerson pick both direction of an undirected edge?



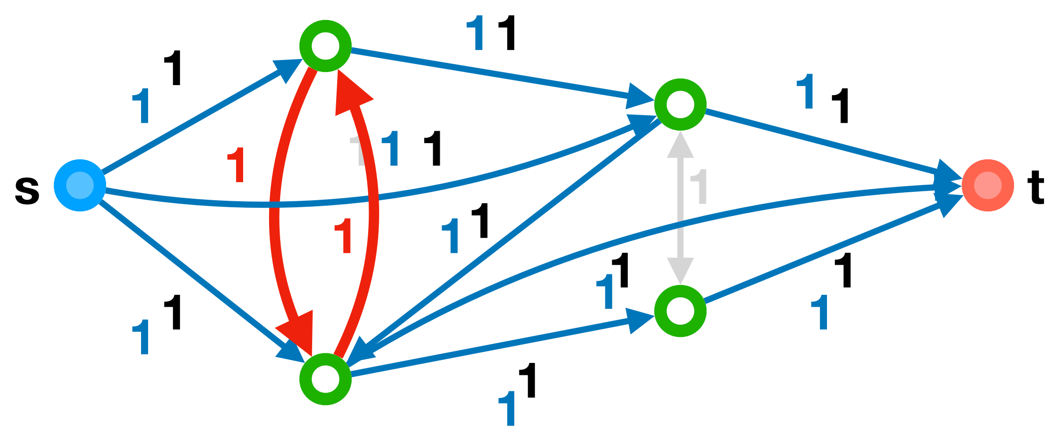
Detour

- Can Ford-Fulkerson pick both direction of an undirected edge?



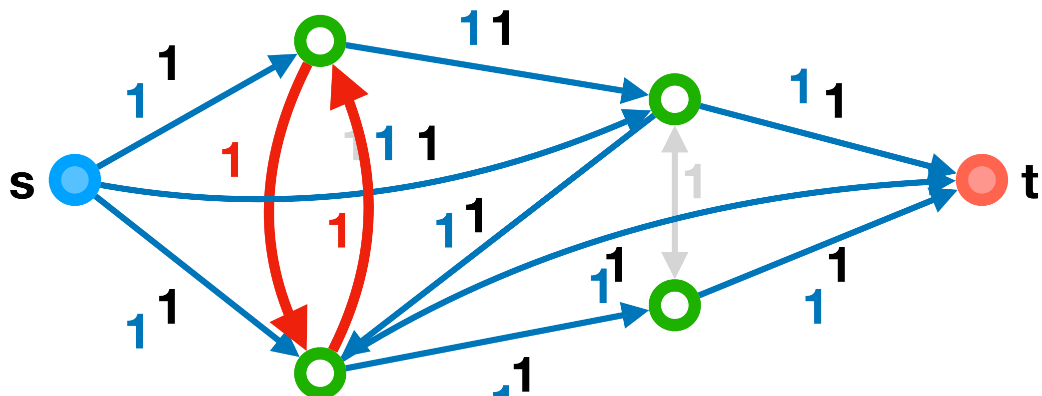
Detour

- Can Ford-Fulkerson pick both direction of an undirected edge?
- This is a form of a flow-cycle



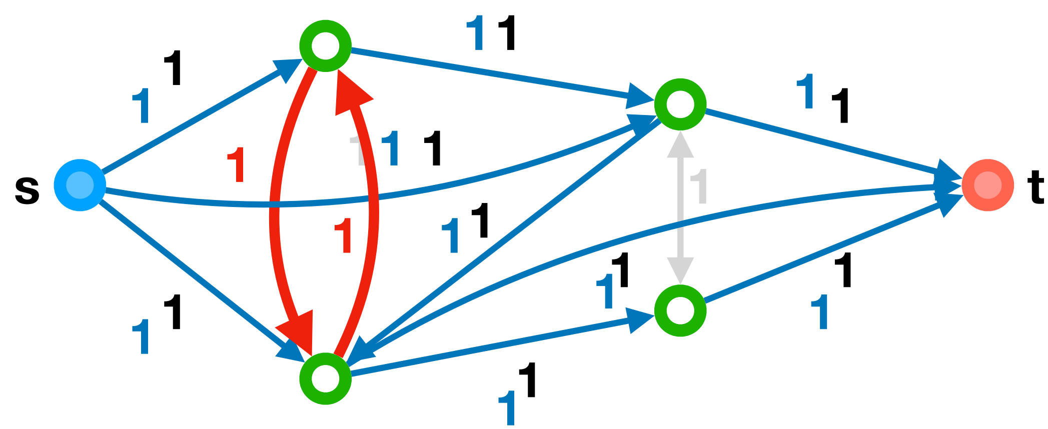
Detour

- Can Ford-Fulkerson pick both direction of an undirected edge?
- This is a form of a flow-cycle
- Flow-cycles are valid by the definition of flow



Detour

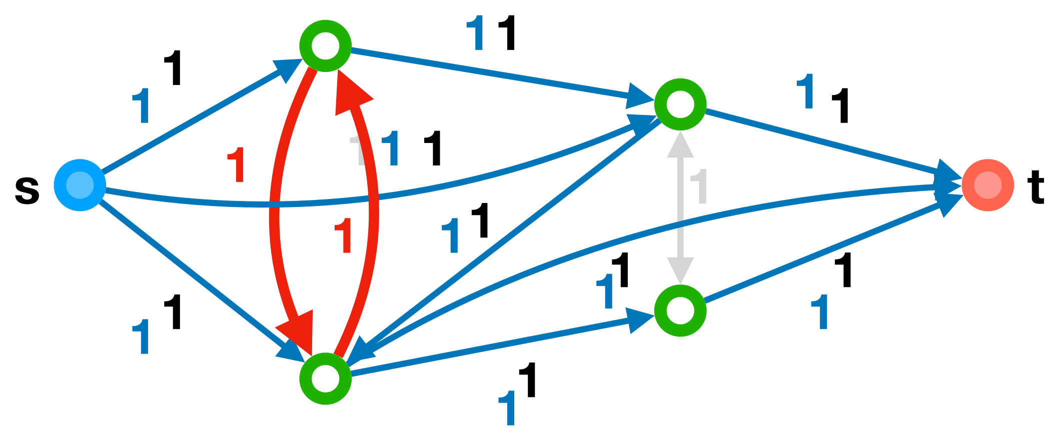
- Can Ford-Fulkerson pick both direction of an undirected edge?
- This is a form of a flow-cycle
- Flow-cycles are valid by the definition of flow



- Removing them will NOT change the value of flow (or its feasibility)

Detour

- Can Ford-Fulkerson pick both direction of an undirected edge?
- This is a form of a flow-cycle
- Flow-cycles are valid by the definition of flow



- Removing them will NOT change the value of flow (or its feasibility)
- Ford-Fulkerson never outputs a flow with flow-cycles

Proof of Correctness

- Create a network $G' = (V', E')$:
  - Vertices are the same as G
  - For any edge {u,v} in G, add both directed edges (u,v) and (v,u) in G' with capacity 1
  - Let source be s and sink be t
- Compute a maximum flow f in G'
- Return the value of f as the maximum number of edge-disjoint paths possible in G

- Part one: Any flow of value k in G' can be turned into k edge-disjoint paths in G

Proof of Correctness

- Create a network $G' = (V', E')$:
  - Vertices are the same as G
  - For any edge {u,v} in G, add both directed edges (u,v) and (v,u) in G' with capacity 1
  - Let source be s and sink be t
- Compute a maximum flow f in G'
- Return the value of f as the maximum number of edge-disjoint paths possible in G

- Part two: Any collection of k edge-disjoint paths in G can be turned into a flow of value k in G'

Proof of Correctness

- Create a network $G' = (V', E')$:
  - Vertices are the same as G
  - For any edge {u,v} in G, add both directed edges (u,v) and (v,u) in G' with capacity 1
  - Let source be s and sink be t
- Compute a maximum flow f in G'
- Return the value of f as the maximum number of edge-disjoint paths possible in G

- So maximum flow f in the reduction gives maximum number of edge-disjoint paths

Runtime Analysis

- Create a network $G' = (V', E')$:
  - Vertices are the same as G
  - For any edge {u,v} in G, add both directed edges (u,v) and (v,u) in G' with capacity 1
  - Let source be s and sink be t
- Compute a maximum flow f in G'
- Return the value of f as the maximum number of edge-disjoint paths possible in G

- It takes O(n+m) time to create the network G'
- G' has 2m directed edges
- So Ford-Fulkerson takes O(mF) time

Runtime Analysis

- Create a network $G' = (V', E')$:
  - Vertices are the same as G
  - For any edge {u,v} in G, add both directed edges (u,v) and (v,u) in G’ with capacity 1
  - Let source be s and sink be t
- Compute a maximum flow f in G’
- Return the value of f as the maximum number of edge-disjoint paths possible in G

- Are we done?

Runtime Analysis

- Create a network $G' = (V', E')$:
  - Vertices are the same as G
  - For any edge {u,v} in G, add both directed edges (u,v) and (v,u) in G’ with capacity 1
  - Let source be s and sink be t
- Compute a maximum flow f in G’
- Return the value of f as the maximum number of edge-disjoint paths possible in G

- Are we done? NO!

Runtime Analysis

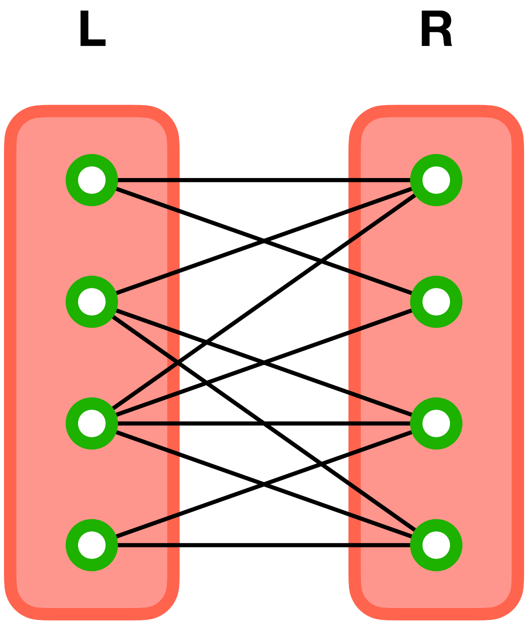
- Create a network $G' = (V', E')$:
  - Vertices are the same as G
  - For any edge {u,v} in G, add both directed edges (u,v) and (v,u) in G’ with capacity 1
  - Let source be s and sink be t
- Compute a maximum flow f in G’
- Return the value of f as the maximum number of edge-disjoint paths possible in G

- Are we done? NO!
- We cannot leave F here in the runtime as it is not a parameter of the original problem but only the solution.
- But we know F < n as s only has n-1 outgoing edges
- So the runtime is O(mn) at most

Application II: Bipartite Matching

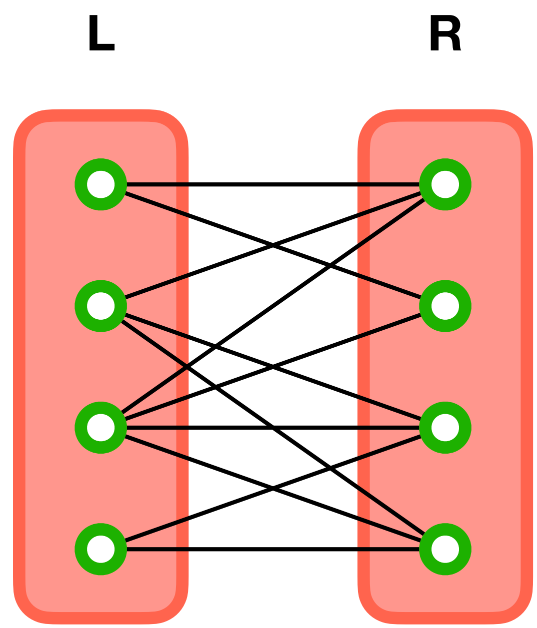
Bipartite Matching

- Bipartite graph G=(V,E):
  - There are two sets of vertices L and R
  - All edges are only between L and R



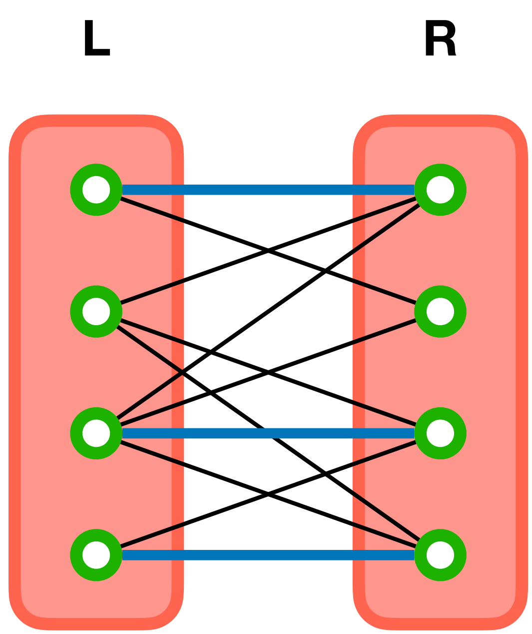
Bipartite Matching

- Bipartite graph G=(V,E):
  - There are two sets of vertices L and R
  - All edges are only between L and R
- Matching M in G:
  - A subset of edges in E
  - No vertex used more than once



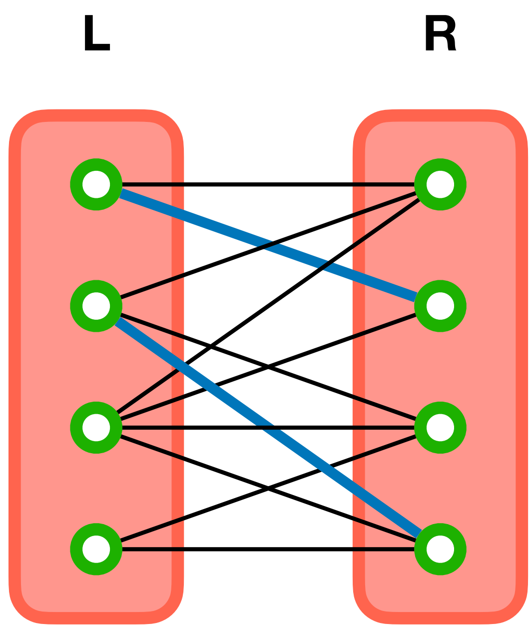
Bipartite Matching

- Bipartite graph G=(V,E):
  - There are two sets of vertices L and R
  - All edges are only between L and R
- Matching M in G:
  - A subset of edges in E
  - No vertex used more than once



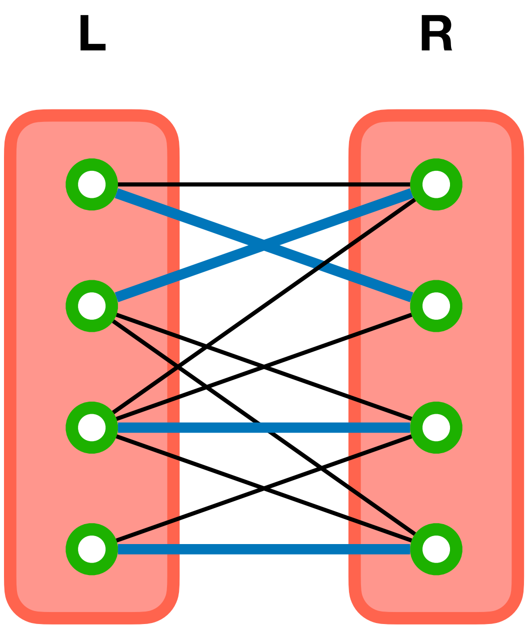
Bipartite Matching

- Bipartite graph G=(V,E):
  - There are two sets of vertices L and R
  - All edges are only between L and R
- Matching M in G:
  - A subset of edges in E
  - No vertex used more than once



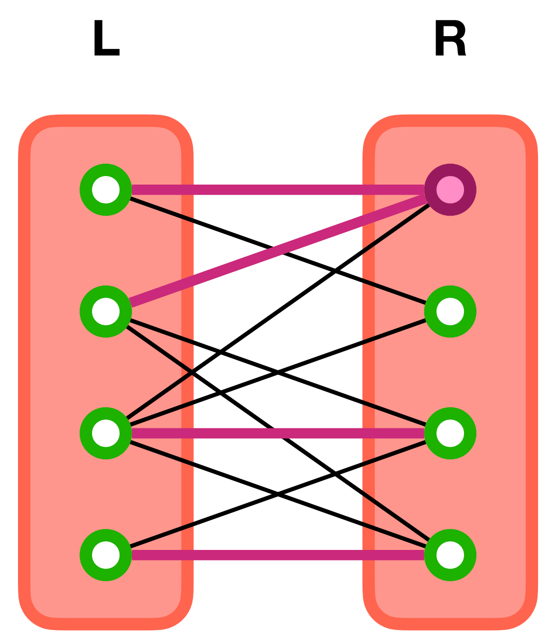
Bipartite Matching

- Bipartite graph G=(V,E):
  - There are two sets of vertices L and R
  - All edges are only between L and R
- Matching M in G:
  - A subset of edges in E
  - No vertex used more than once



Bipartite Matching

- Bipartite graph G=(V,E):
  - There are two sets of vertices L and R
  - All edges are only between L and R
- Matching M in G:
  - A subset of edges in E
  - No vertex used more than once

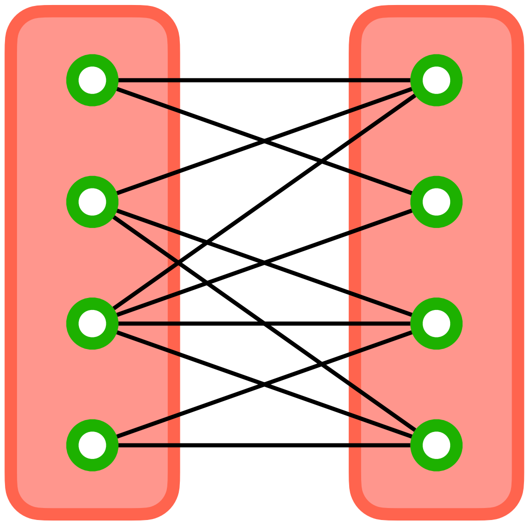


Bipartite Matching Problem

- **Input:**
  - a bipartite graph G=(V,E) with bipartition L and R
- **Output:**
  - Output a maximum matching in G, i.e., a matching with the largest number of edges

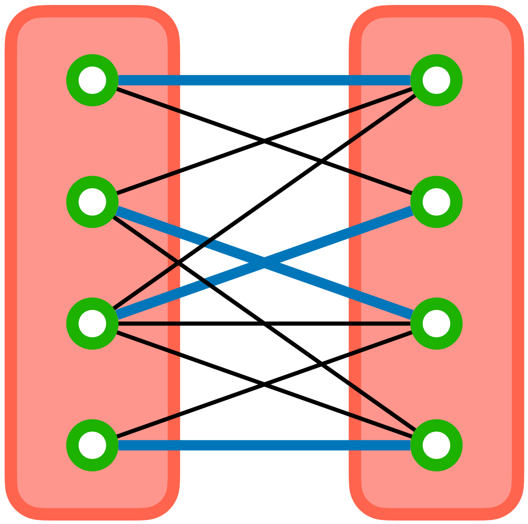
Bipartite Matching Problem

- **Input:**
  - a bipartite graph G=(V,E) with bipartition L and R
- **Output:**
  - Output a maximum matching in G, i.e., a matching with the largest number of edges



Bipartite Matching Problem

- **Input:**
  - a bipartite graph G=(V,E) with bipartition L and R
- **Output:**
  - Output a maximum matching in G, i.e., a matching with the largest number of edges



Bipartite Matching Problem

- Applications:
  - Online advertising
  - Auctions and markets
  - Students-Dorms Assignments
  - Kidney exchange program
  - …

Reduction to Network Flow

- Create a network G'=(V',E') as follows:
  - Copy the vertices in L and R of G in G' also
  - For any edge (u,v) in G with u in L and v in R, add a directed edge from u to v in G'. Set the capacities to $+\infty$
  - Add new vertices s and t, which will be source and sink
  - Connect s to every vertex in L with capacity 1
  - Connect every vertex in R to t with capacity 1

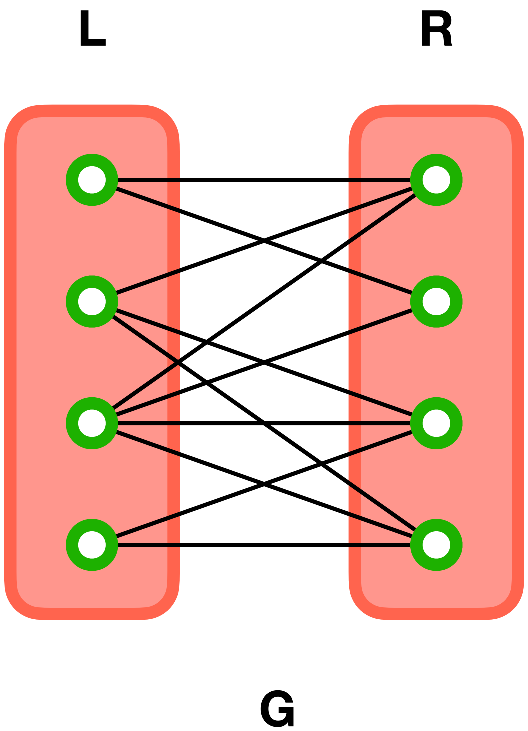
Reduction to Network Flow

- Create a network G'=(V',E') as follows:
  - Copy the vertices in L and R of G in G' also
  - For any edge (u,v) in G with u in L and v in R, add a directed edge from u to v in G'. Set the capacities to $+\infty$
  - Add new vertices s and t, which will be source and sink
  - Connect s to every vertex in L with capacity 1
  - Connect every vertex in R to t with capacity 1
- Compute a maximum flow f in G'

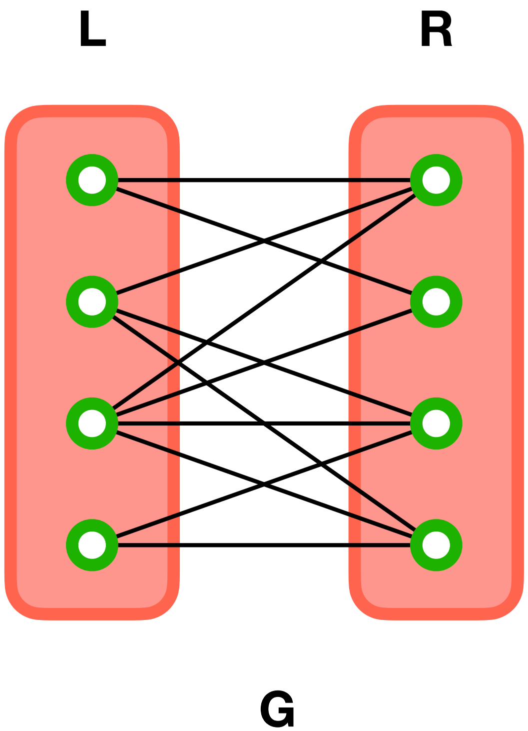
Reduction to Network Flow

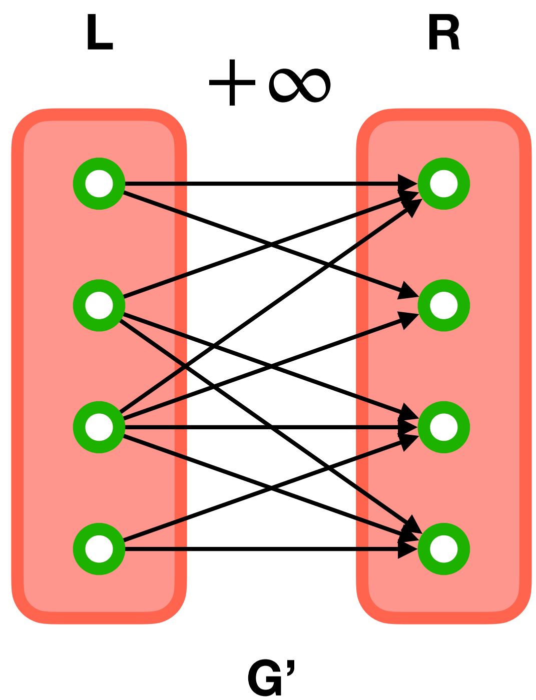
- Create a network G'=(V',E') as follows:
  - Copy the vertices in L and R of G in G' also
  - For any edge (u,v) in G with u in L and v in R, add a directed edge from u to v in G'. Set the capacities to $+\infty$
  - Add new vertices s and t, which will be source and sink
  - Connect s to every vertex in L with capacity 1
  - Connect every vertex in R to t with capacity 1
- Compute a maximum flow f in G'
- Return edges (u,v) in G if u in L and v in R and f(u,v) = 1

Reduction to Network Flow

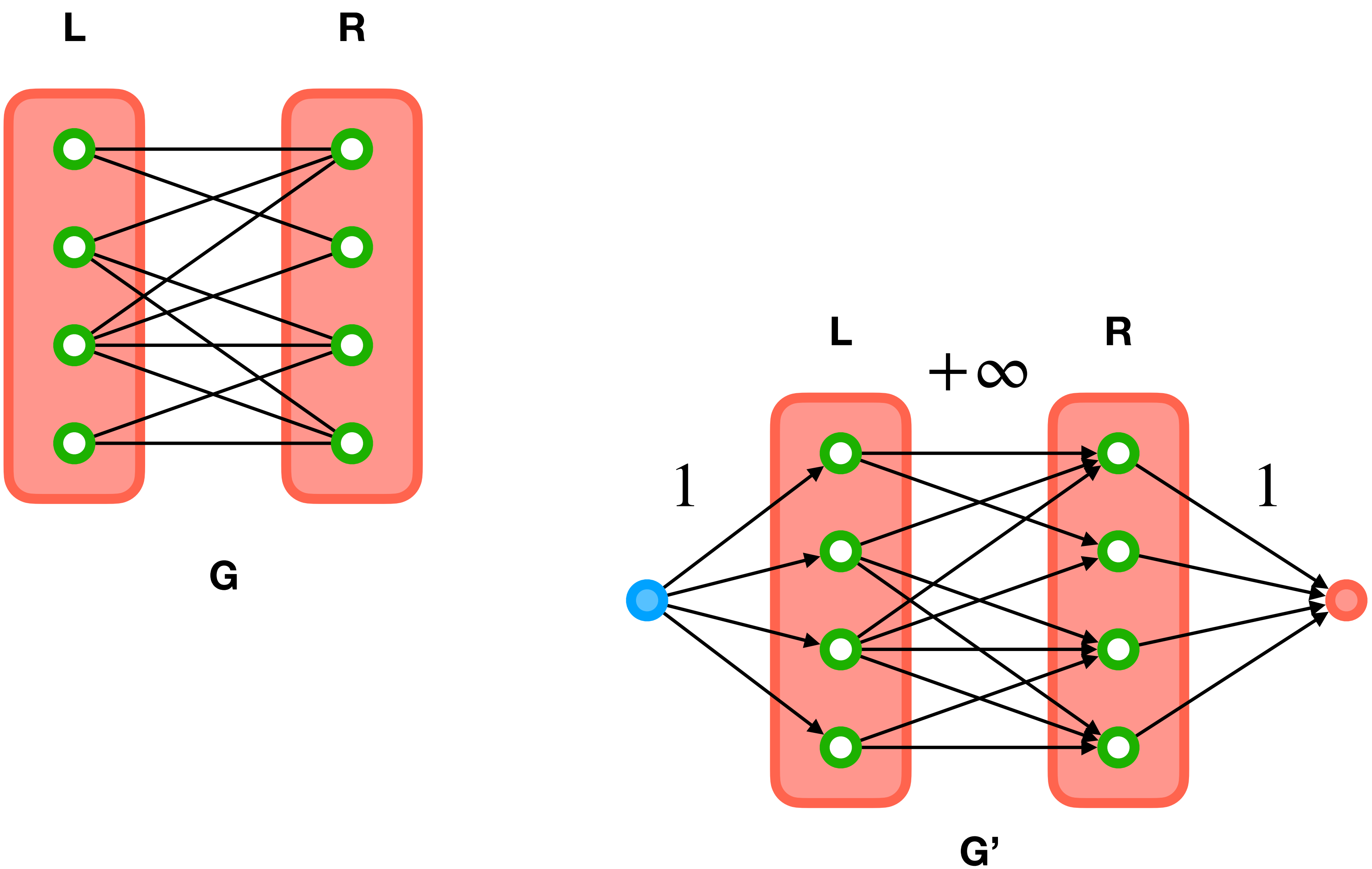


Reduction to Network Flow

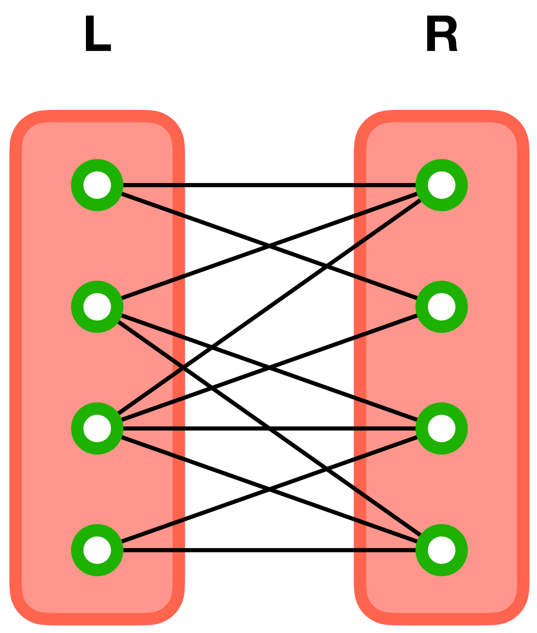




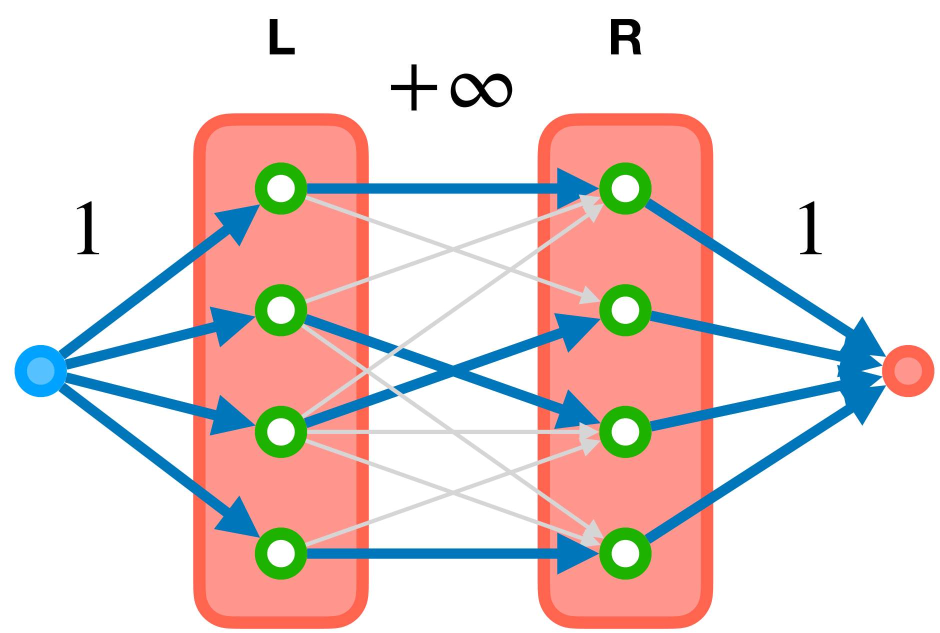
Reduction to Network Flow



Reduction to Network Flow

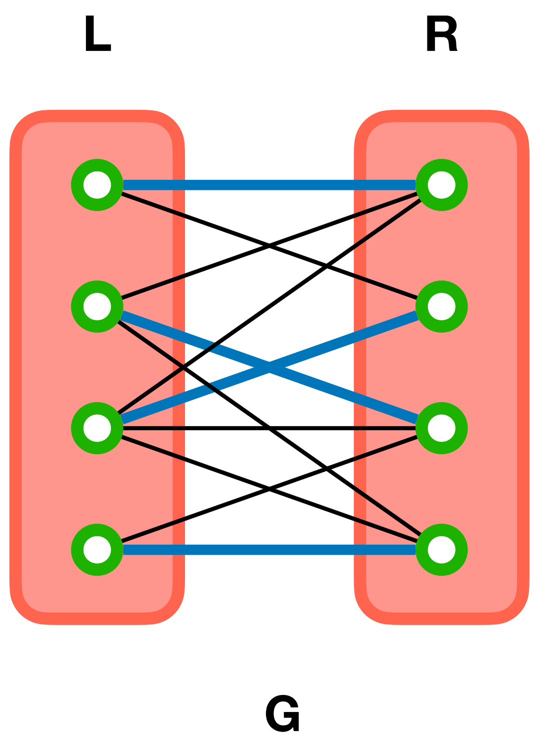


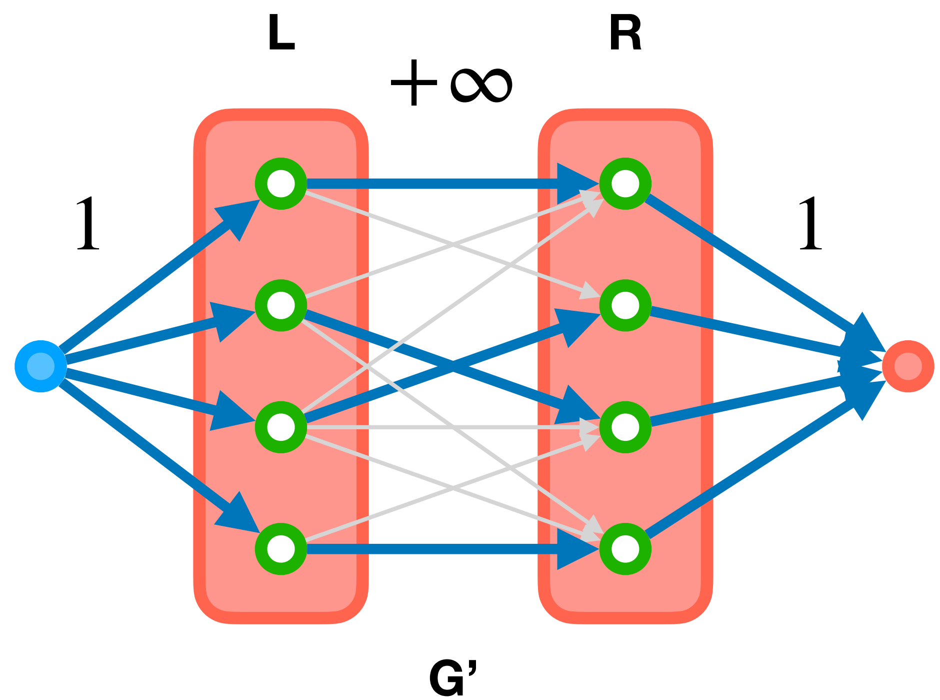
G



G’

Reduction to Network Flow





Proof of Correctness

- Create a network $G'=(V',E')$ as follows:
- Copy the vertices in L and R of G in G'
- For any edge (u,v) in G with u in L and v in R, add a directed edge from u to v in G'. Set the capacities to $+\infty$
- Add a source s and sink t
- Connect s to vertices in L with capacity 1
- Connect vertices in R to t with capacity 1
- Compute a maximum flow f in G'
- Return edges (u,v) in G if u in L and v in R and $f(u,v) = 1$

- Part one: A flow f of value k gives a matching M of size k

Proof of Correctness

- Create a network G'=(V',E') as follows:
- Copy the vertices in L and R of G in G'
- For any edge (u,v) in G with u in L and v in R, add a directed edge from u to v in G'. Set the capacities to $+\infty$
- Add a source s and sink t
- Connect s to vertices in L with capacity 1
- Connect vertices in R to t with capacity 1
- Compute a maximum flow f in G'
- Return edges (u,v) in G if u in L and v in R and f(u,v) = 1

- Part two: A matching M of size k gives a flow f of value k

Proof of Correctness

- Create a network G'=(V',E') as follows:
  - Copy the vertices in L and R of G in G'
  - For any edge (u,v) in G with u in L and v in R, add a directed edge from u to v in G'. Set the capacities to $+\infty$
  - Add a source s and sink t
  - Connect s to vertices in L with capacity 1
  - Connect vertices in R to t with capacity 1
- Compute a maximum flow f in G'
- Return edges (u,v) in G if u in L and v in R and f(u,v) = 1

- So the maximum flow f gives a maximum matching M

Runtime Analysis

- Create a network G'=(V',E') as follows:
- Copy the vertices in L and R of G in G'
- For any edge (u,v) in G with u in L and v in R, add a directed edge from u to v in G'. Set the capacities to $+\infty$
- Add a source s and sink t
- Connect s to vertices in L with capacity 1
- Connect vertices in R to t with capacity 1
- Compute a maximum flow f in G'
- Return edges (u,v) in G if u in L and v in R and f(u,v) = 1

- Creating G' takes O(n+m) time
- G' has n+2 vertices and m+2n edges
- So Ford-Fulkerson takes O((m+n)*F) time

Runtime Analysis

- Create a network G'=(V',E') as follows:
  - Copy the vertices in L and R of G in G'
  - For any edge (u,v) in G with u in L and v in R, add a directed edge from u to v in G'. Set the capacities to $+\infty$
  - Add a source s and sink t
  - Connect s to vertices in L with capacity 1
  - Connect vertices in R to t with capacity 1
- Compute a maximum flow f in G'
- Return edges (u,v) in G if u in L and v in R and f(u,v) = 1

- Creating G' takes O(n+m) time
- G' has n+2 vertices and m+2n edges
- So Ford-Fulkerson takes O((m+n)*F) time
- F <= n/2 as F is equal to the maximum matching size and that is <= n/2
- So it takes O((n+m)n) time

Runtime Analysis

- Create a network G'=(V',E') as follows:
- Copy the vertices in L and R of G in G'
- For any edge (u,v) in G with u in L and v in R, add a directed edge from u to v in G'. Set the capacities to $+\infty$
- Add a source s and sink t
- Connect s to vertices in L with capacity 1
- Connect vertices in R to t with capacity 1
- Compute a maximum flow f in G'
- Return edges (u,v) in G if u in L and v in R and f(u,v) = 1

- We can make it O(mn) only by removing all vertices in G that have no edges so n = O(m)