CS 344: Design and Analysis of Computer Algorithms

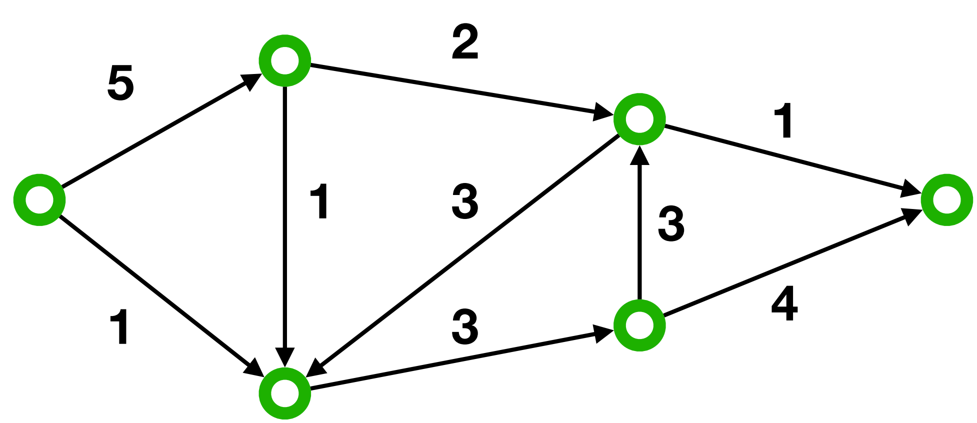
(Spring 2022 — Sections 5,6,7,8)

Lecture 22: Applications of Network Flow

The Network Flow Problem

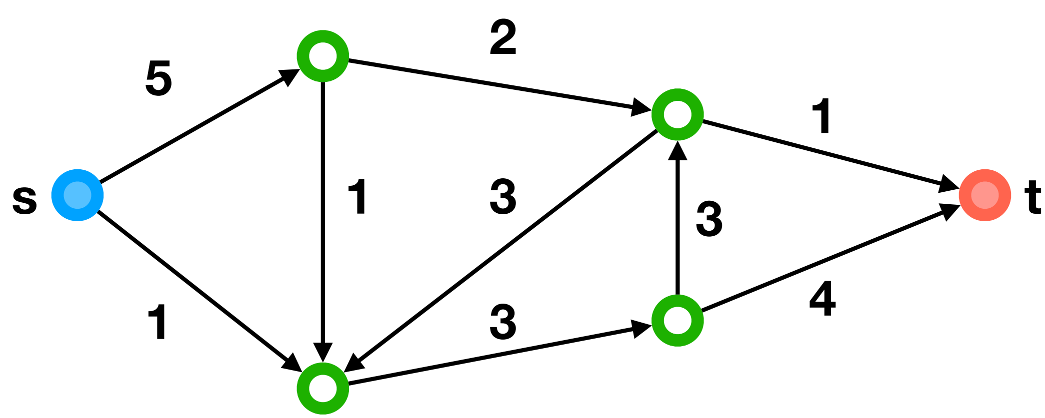
Motivation

- 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



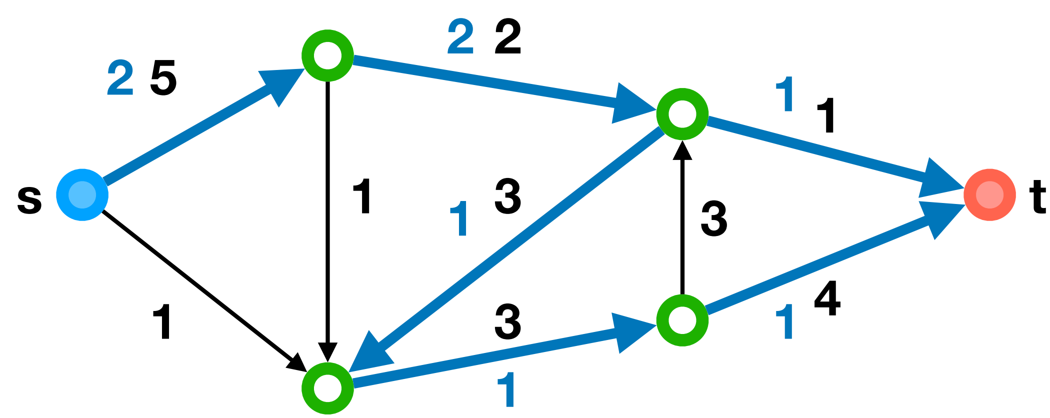
Networks

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



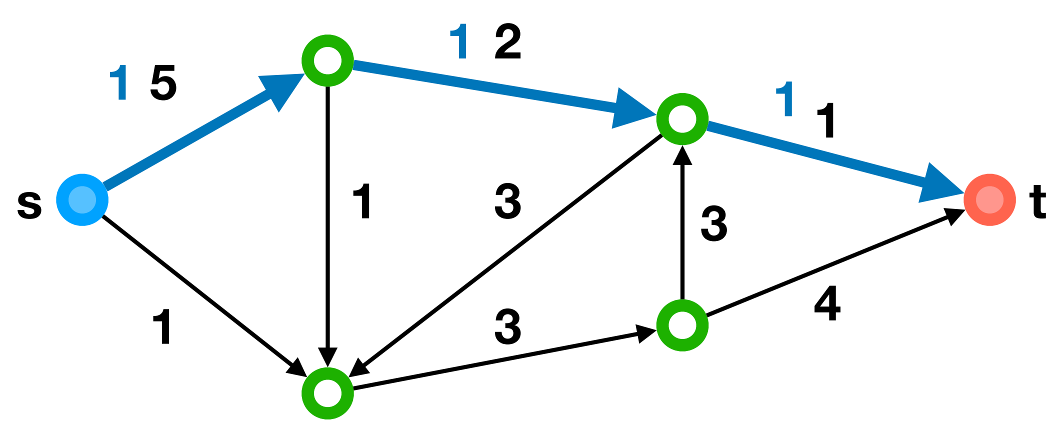
Flow: Example

- 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

Ford-Fulkerson

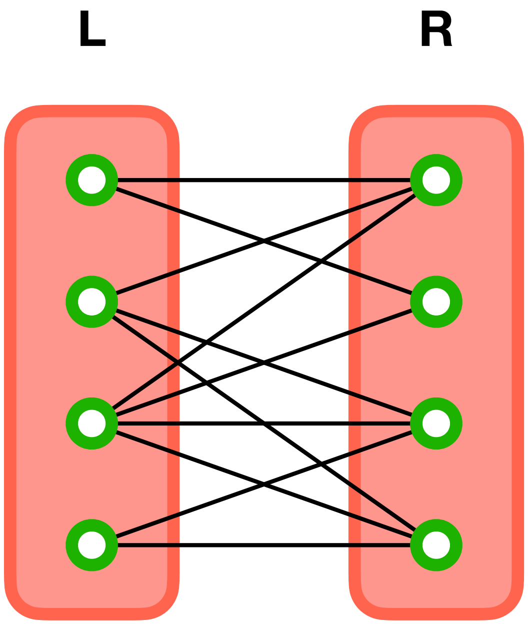
- For concreteness, we stick with the first and simplest network flow algorithm
- **Ford-Fulkerson Algorithm:**
  - Solves maximum flow in $O(m \cdot F)$ time where F is the value of maximum flow from s to t.

Application I: Edge-Disjoint Paths

Application II:
Bipartite Matching

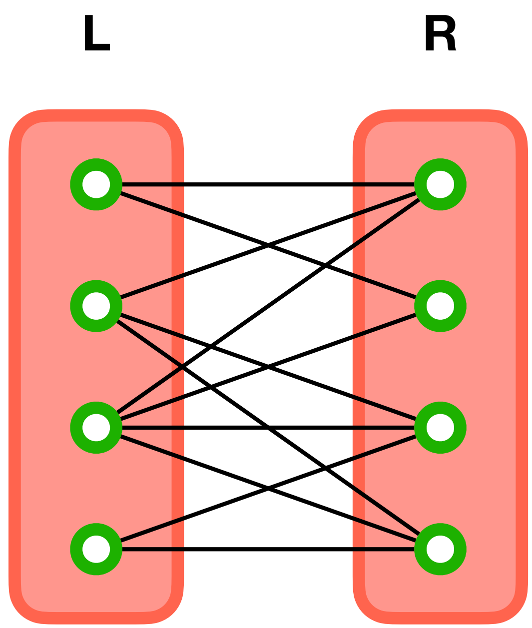
Bipartite Matching

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



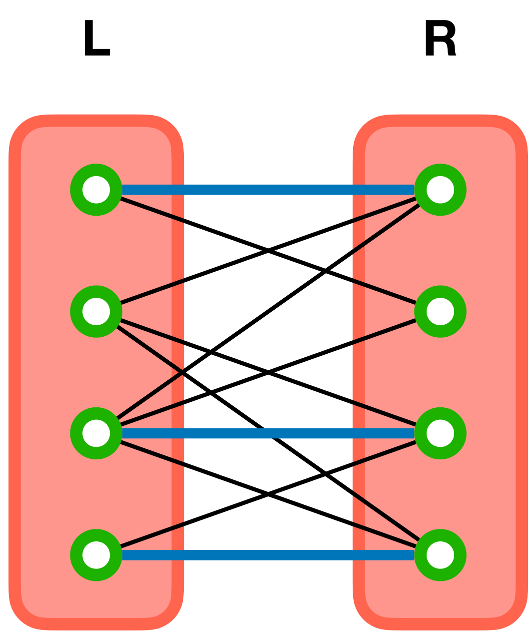
Bipartite Matching

- 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



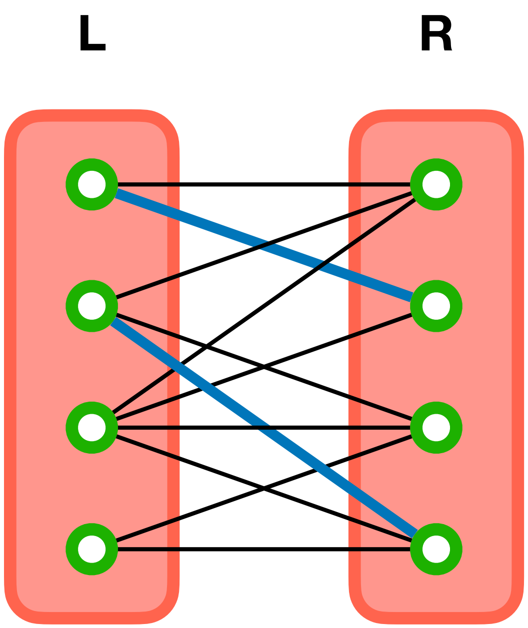
Bipartite Matching

- 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



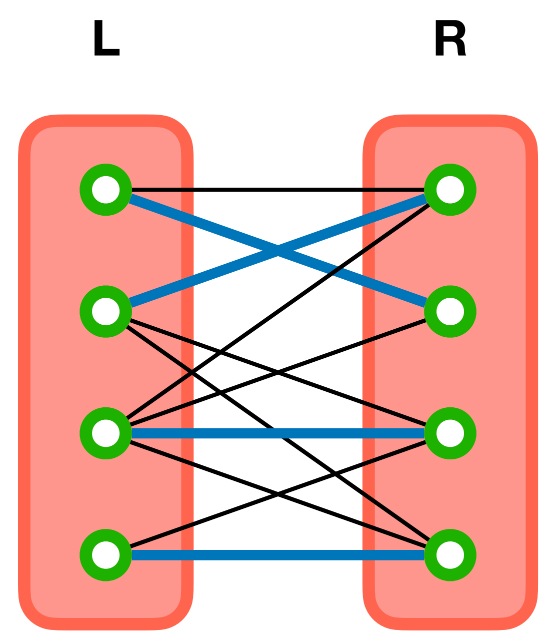
Bipartite Matching

- 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



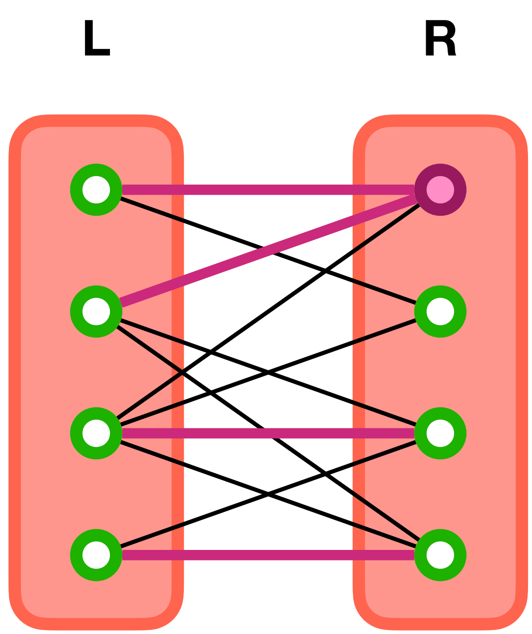
Bipartite Matching

- 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

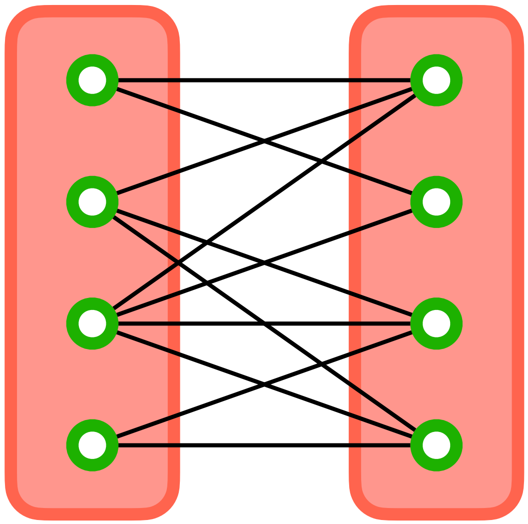


Bipartite Matching Problem

- **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

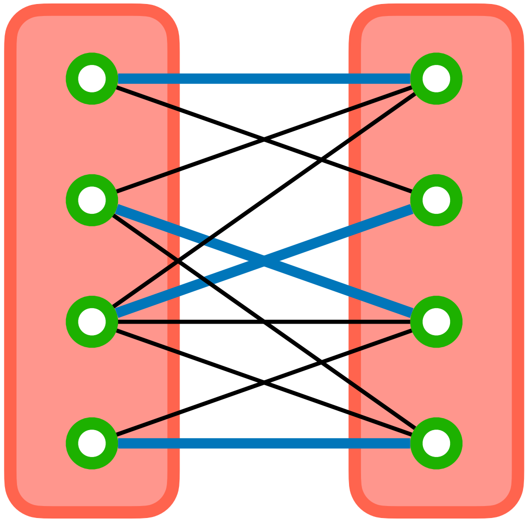
Bipartite Matching Problem

- **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

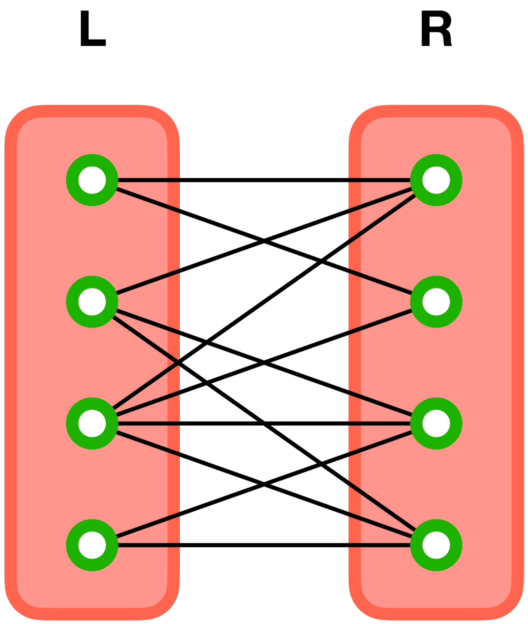
- **Input:**
  - a bipartite graph G=(V,E) with bipartition L and R
- **Output:**
  - Output a maximum matching in G, i.e., a matching with the largest number of edges



Bipartite Matching Problem

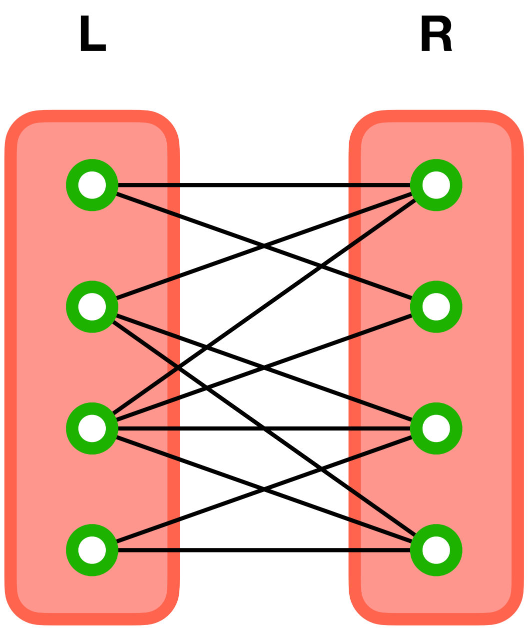
- Applications:
  - Online advertising
  - Auctions and markets
  - Students-Dorms Assignments
  - Kidney exchange program
  - …

Reduction to Network Flow

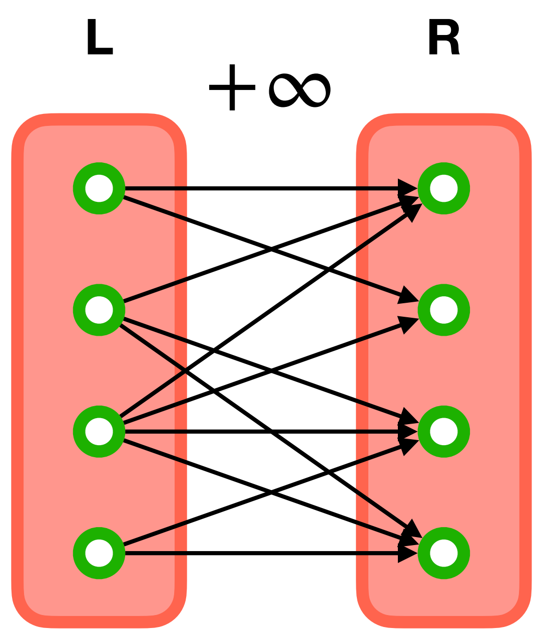


G

Reduction to Network Flow

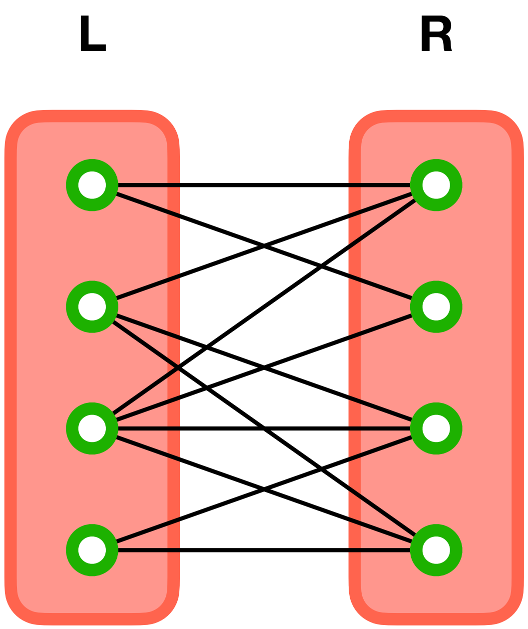


G

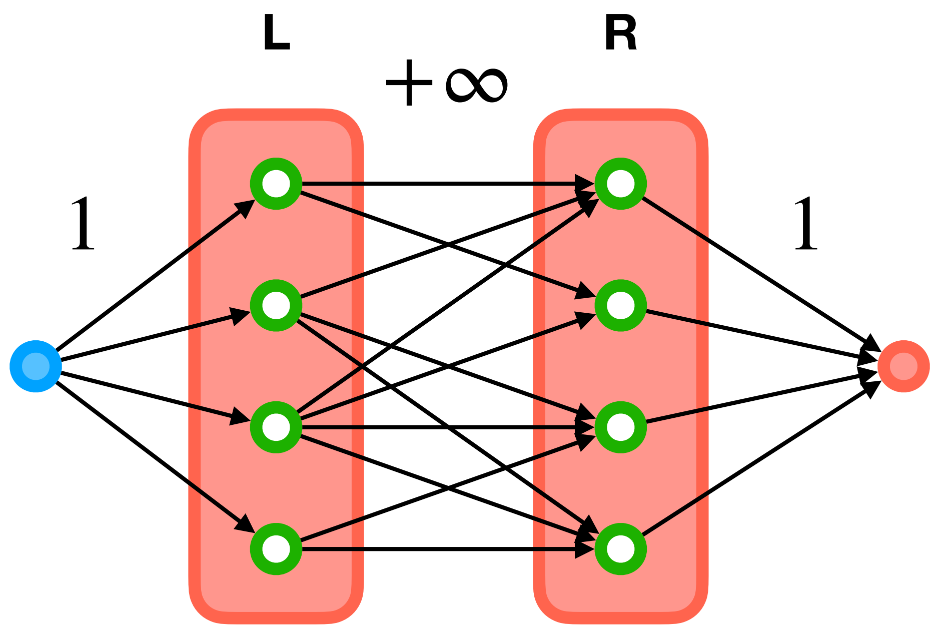


G’

Reduction to Network Flow

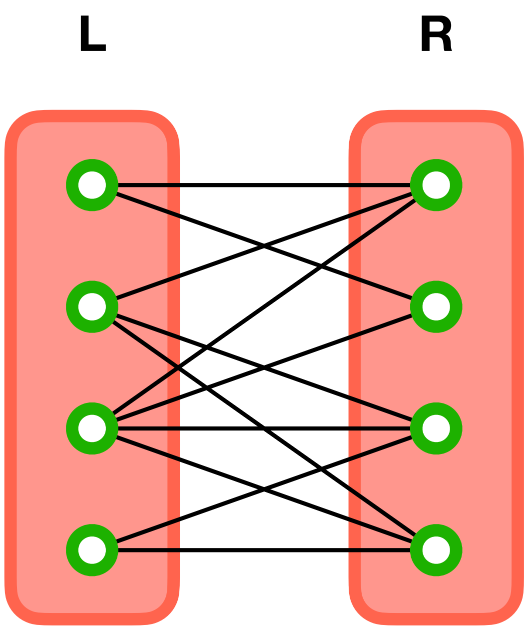


G

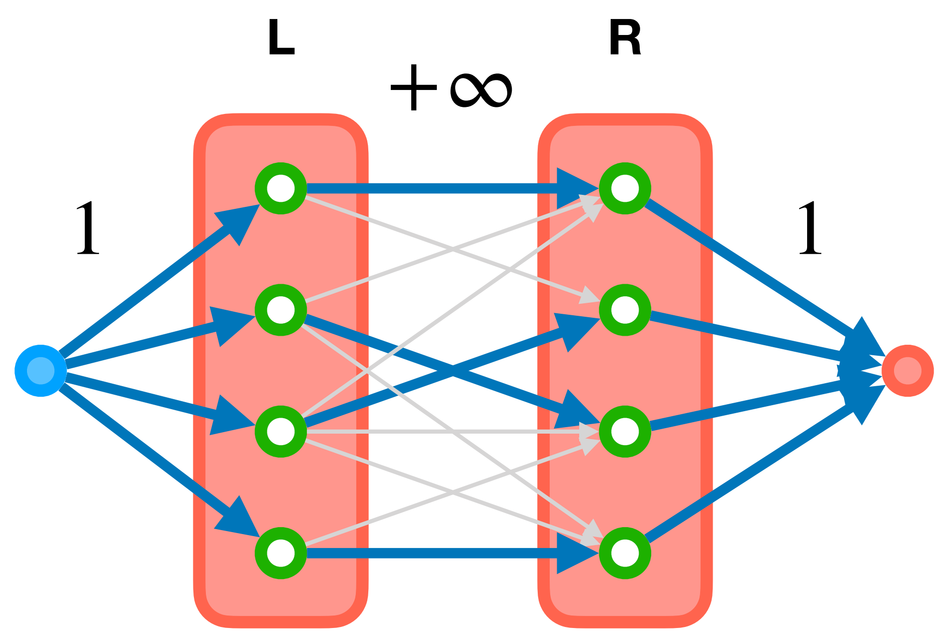


G’

Reduction to Network Flow

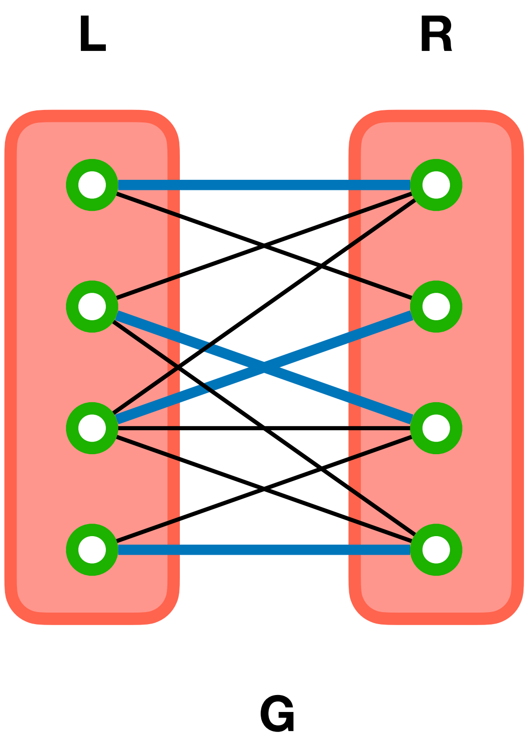


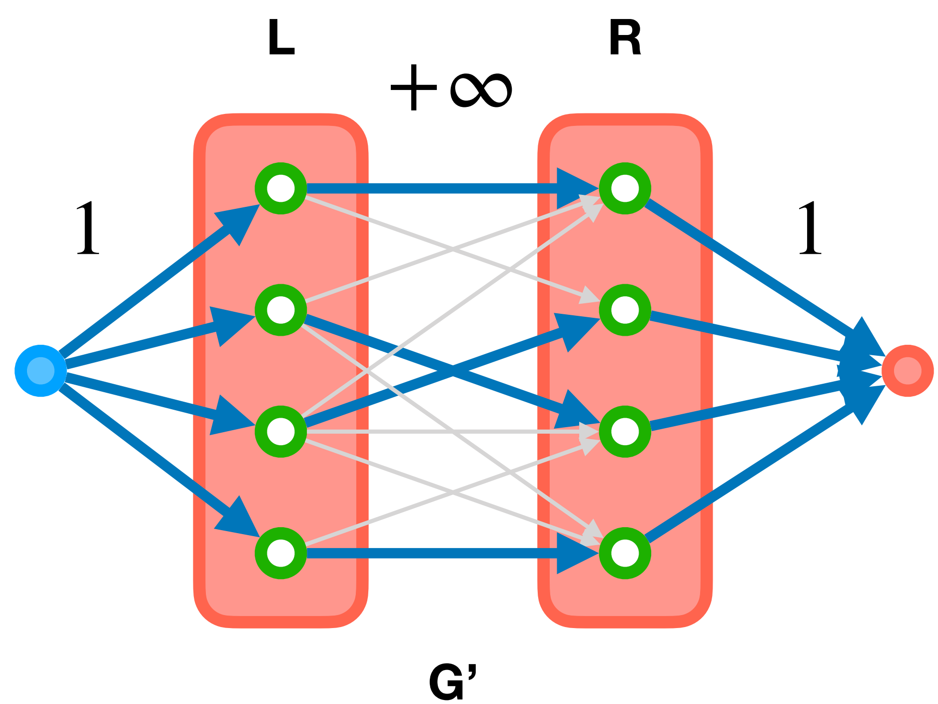
G



G’

Reduction to Network Flow





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 $f(u,v) = 1$

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 $O(m*F)$ 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 $O(m*F)$ time
- $F <= n/2$ as F is equal to the maximum matching size and that is $<= n/2$
- So it takes $O(mn)$ time

Application III: Exam Scheduling

Exam Scheduling Problem

- **Input:**
  - c courses with course i having $E[i]$ enrolled students
  - r rooms with room j having $S[j]$ available seats
  - t available time-slots for exams denoted by $\{1,2,\ldots,t\}$
  - p proctors with proctor k being available at times $T[k] \subseteq \{1,\ldots,t\}$

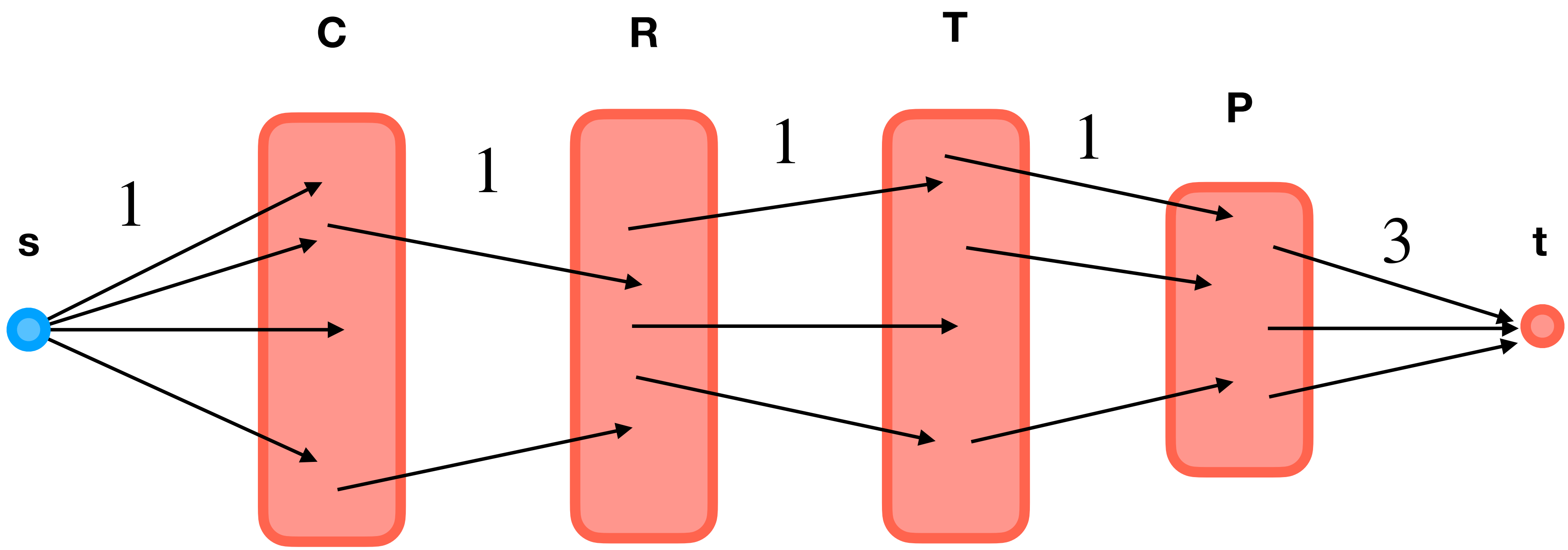
Exam Scheduling Problem

- **Input:**
  - c courses with course i having $E[i]$ enrolled students
  - r rooms with room j having $S[j]$ available seats
  - t available time-slots for exams denoted by $\{1,2,\ldots,t\}$
  - p proctors with proctor k being available at times $T[k] \subseteq \{1,\ldots,t\}$
- **Constraints:**
  - Room j can only be assigned to course i if $E[i] \leq S[j]$
  - Any room can be assigned to only one exam in a given time-slot
  - A proctor k can only attend exams at time-slots in $T[k]$
  - No proctor can work for more than 3 exams and each exam needs exactly 1 proctor

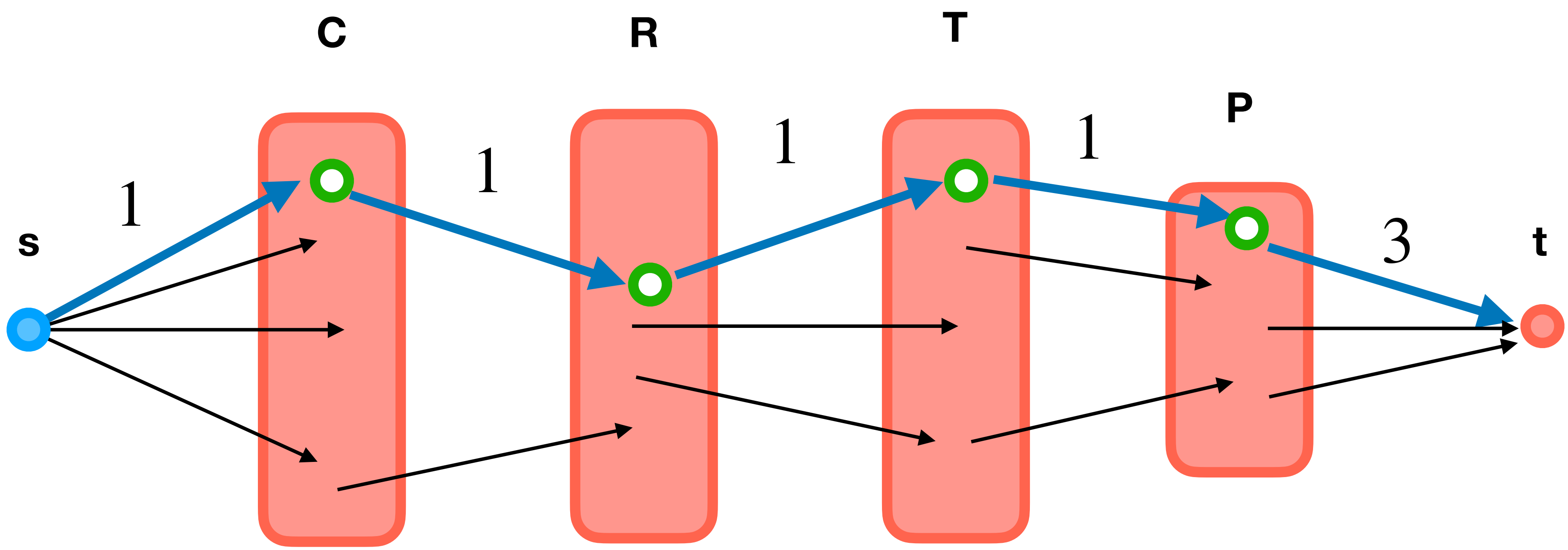
Exam Scheduling Problem

- **Input:**
  - c courses with course i having E[i] enrolled students
  - r rooms with room j having S[j] available seats
  - t available time-slots for exams denoted by {1,2,…,t}
  - p proctors with proctor k being available at times T[k] $\subseteq$ {1,…,t}
- **Constraints:**
  - Room j can only be assigned to course i if E[i] $\leq$ S[j]
  - Any room can be assigned to only one exam in a given time-slot
  - A proctor k can only attend exams at time-slots in T[k]
  - No proctor can work for more than 3 exams and each exam needs exactly 1 proctor
- **Output:**
  - Output c tuples (course, room, time-slot, proctor) satisfying the constraints or say it is not possible

Reduction to Network Flow



Reduction to Network Flow



Reduction to Network Flow

- Create G = (V,E) with 4 layers of vertices C, R, T, and P and one source s and one sink t
  - c vertices corresponding to courses in C
  - r vertices corresponding to rooms in R
  - t vertices corresponding to time-slots in T
  - p vertices corresponding to proctors in P

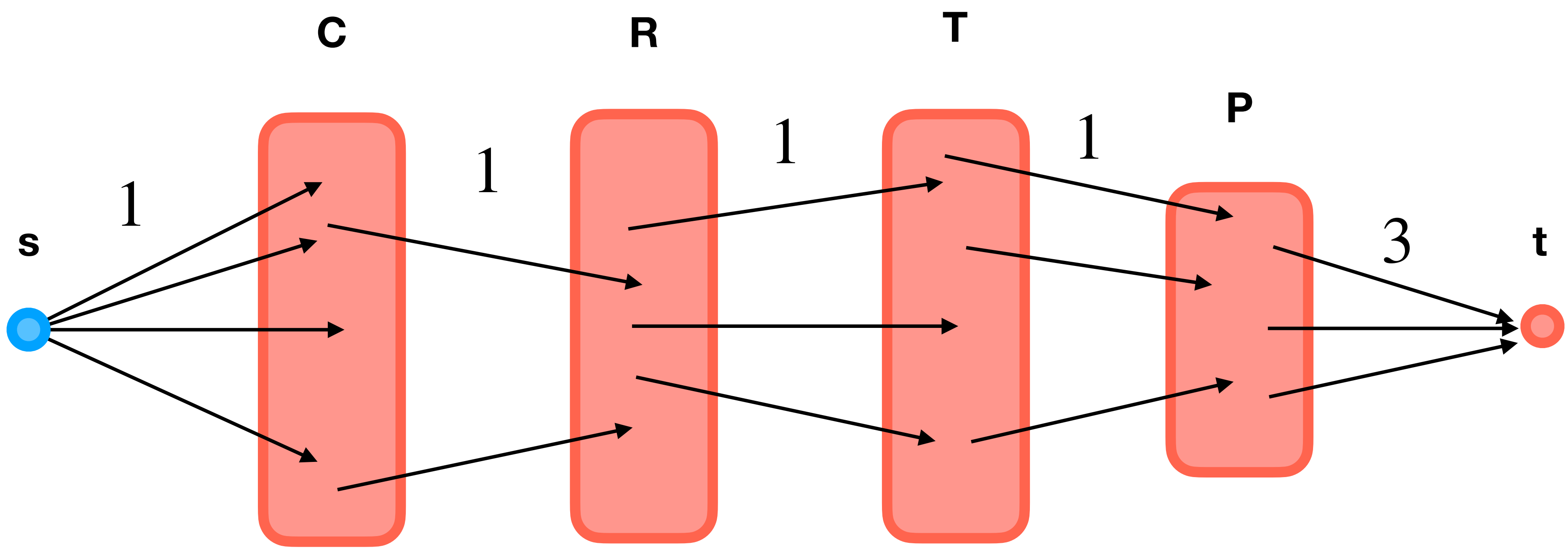
Reduction to Network Flow

- Create G = (V,E) with 4 layers of vertices C, R, T, and P and one source s and one sink t
- Add the following edges:
  - Source s to all vertices in C with capacity 1
  - Vertex i in C to vertex j in R if E[i] <= S[j] (we can hold exam of course i in room j) with capacity 1
  - All vertices in R to all vertices in T with capacity 1
  - Vertex t in T to vertex k in P if t in T[k] with capacity 1 (proctor k can work on an exam at time t)
  - Vertex k in P to t with capacity 3

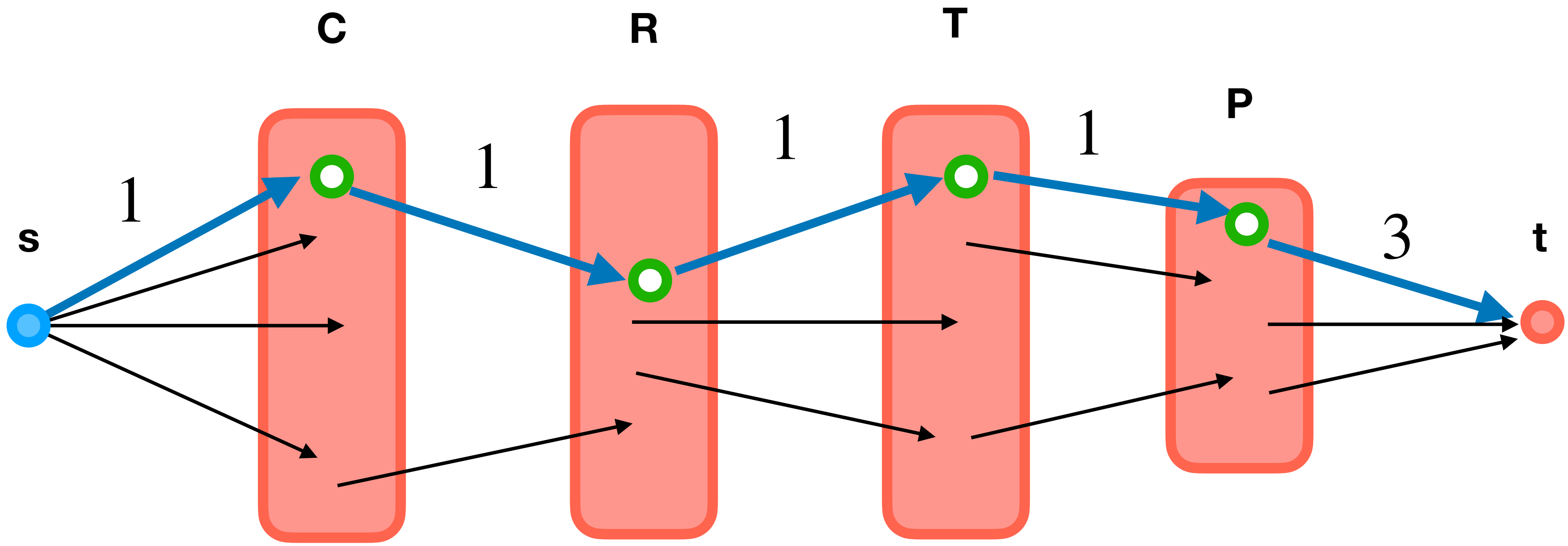
Reduction to Network Flow

- Create G = (V,E) with 4 layers of vertices C, R, T, and P and one source s and one sink t
- Add the following edges
- Find a maximum flow f in the network G
- For any flow-path (s, i in C, j in R, t in T, p in P, t), we return the tuple (i,j,t,p)
- If the number of tuples is less than c, we say scheduling is not possible

Reduction to Network Flow



Reduction to Network Flow



Proof of Correctness

- Part one: any flow f of value ℓ gives a valid schedule of ℓ courses:
- So we output ℓ valid tuples in the schedule

Proof of Correctness

- Part two: any valid schedule of ℓ courses gives a flow of value ℓ
- We thus have maximum flow f will give largest valid schedule

Runtime Analysis

- Network G has n = c + r + t + p + 2 vertices
- So it has at most $m = O(n^2)$ edges
- It takes $O(n+m) = O(n^2)$ times to create the network
- Moreover, largest possible flow has value F $\leq c \leq n$
- So Ford-Fulkerson takes O(mF) = $O(n^3)$ time