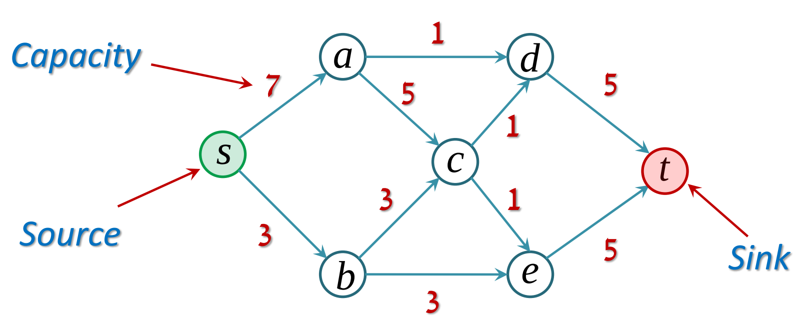
Ma/CS 6a

Class 15: Flows and Bipartite Graphs

By Adam Sheffer

Reminder: Flow Networks

- A ***flow network*** is a digraph $G = (V, E)$, together with a ***source*** vertex $s \in V$, a ***sink*** vertex $t \in V$, and a ***capacity function*** $c: E \to \mathbb{N}$.



Reminder: Flow in a Network

- Given a flow network $G = (V, E, s, t, c)$, a ***flow*** in G is a function $f: E \to \mathbb{N}$ that satisfies
  - Every $e \in E$ satisfies $f(e) \leq c(e)$.
  - Every $v \in V \setminus \{s, t\}$ satisfies

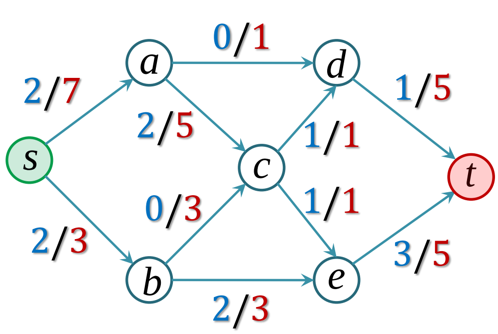
$$\sum_{(u,v)\in E} f(u,v) = \sum_{(v,w)\in E} f(v,w)$$

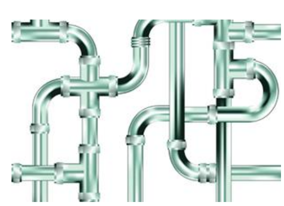
Total flow entering v.

Total flow exiting v.

Example: Flow

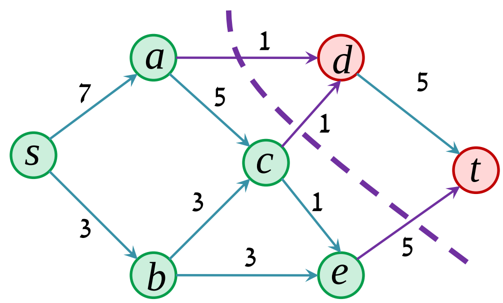
- The capacities are in red.
- The flow is in blue.





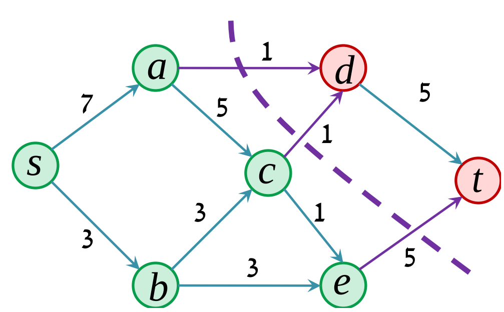
Reminder: Cuts

- A ***cut*** is a partitioning of the vertices of the flow network into two sets S, T such that $s \in S$ and $t \in T$.
- The ***size of a cut*** is the sum of the capacities of the edges from S to T.



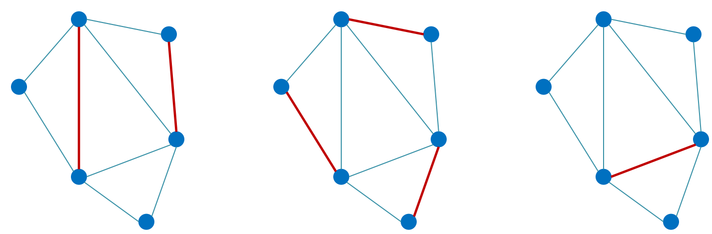
Reminder: Max Flow – Min Cut

- **Max flow – min cut theorem.** In every flow network, the size of the minimum cut ***is equal*** to the size of the maximum flow.



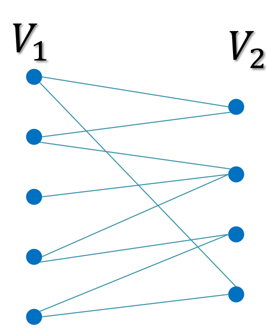
Reminder: Matchings

- A ***matching*** of an undirected graph G is a set of vertex-disjoint edges of G.
- The ***size*** of a matching is the number of edges in it.
- A ***maximum matching*** of G is a matching of maximum size.



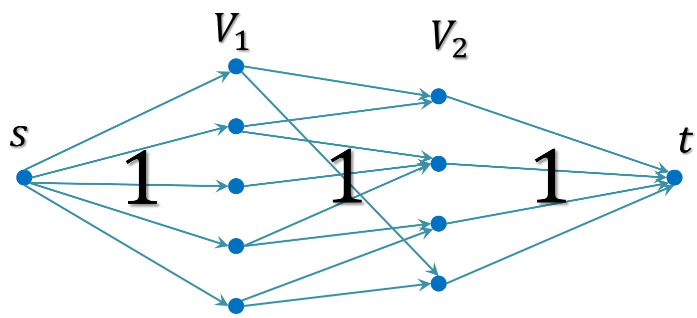
Flows and Bipartite Matchings

- **Problem.** given a bipartite graph $G = (V_1 \cup V_2, E)$, use a flow algorithm to find a maximum matching of G.



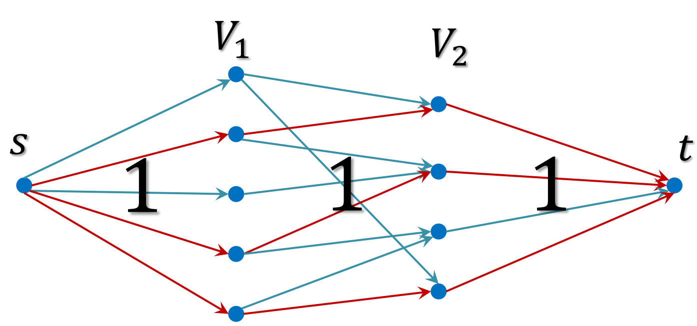
Flows and Bipartite Matchings (2)

- Direct the edges from V_1 to V_2.
- Add a source s and edges from it to every vertex of V_1. Similarly, add a sink t.
- Direct edges to the right and set capacities to 1.



Flows and Bipartite Matchings (3)

- There is a bijection between the perfect matchings of G and the flows of the network.
- A max matching corresponds to a max flow.

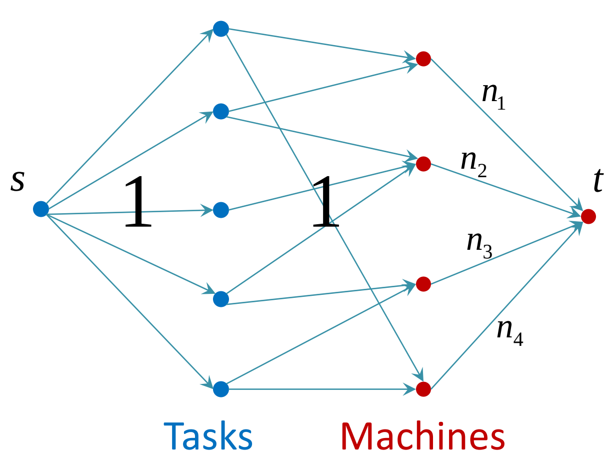


Machines and Jobs

- **Problem.** We are given n machines and m tasks that the machines need to do.
  - The i'th machine performs at most n_i tasks.
  - Every machine has a subset of the tasks that it is able to do.
  - Describe an algorithm for assigning the tasks (or stating that no valid assignment exists).



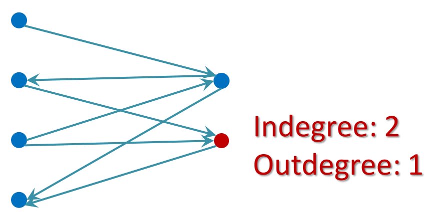
Solution



We wish to find a flow of size m.

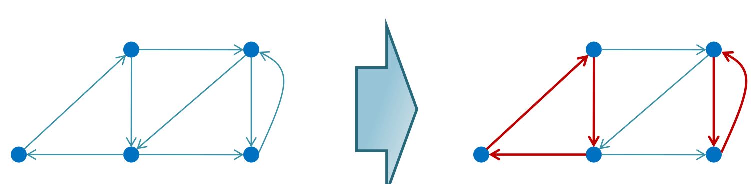
Indegree and Outdegree

- Consider a digraph $G = (V, E)$.
  - The ***indegree*** of a vertex $v \in V$ is the number of edges of E are directed into it.
  - The ***outdegree*** of a vertex $v \in V$ is the number of edges of E are directed out of it.



Subgraphs with Bounded Degrees

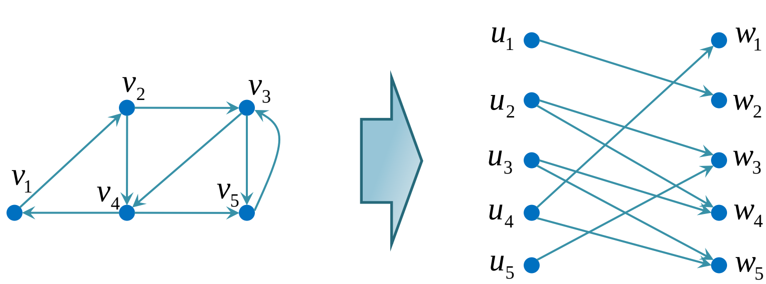
- **Problem.** Given a directed graph $G = (V, E)$, describe an algorithm for finding a subset of the edges $E' \subset E$ so that every vertex of V has an indegree and an outdegree of ***exactly*** 1 in (V, E') (or announce that such a set does not exist).



Solution

- Build a bipartite graph $G^* = (U \cup W, E^*)$ such that $V = U = W$.
  - Write $V = \{v_1, \dots, v_n\}$, $U = \{u_1, \dots, u_n\}$, and $W = \{w_1, \dots, w_n\}$.
  - For every edge $(v_i, v_j) \in E$, we add $(u_i, w_j) \in E^*$.
- Check whether there exists a perfect matching in G^*.
  - If so, return the edges of the matching.
  - Otherwise, no valid subset exists.

Example: Building G^*

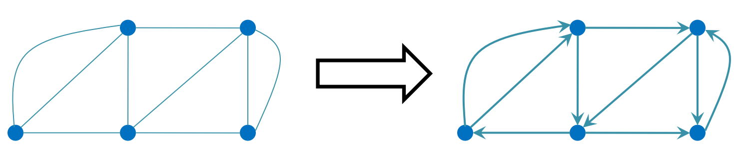


Correctness

- An edge $(v_i, v_j) \in E$ contributes 1 to the outdegree of v_i and to the indegree of v_j.
- In the bipartite graph, it corresponds to (u_i, w_j), contributing 1 to the degrees of u_i, w_j.
- Consider a subset of the edges $E' \subset E$ and the degrees that the degrees that E' induces.
  - A vertex v_i has indegree and outdegree 1 iff both u_i and w_i have a degree of 1.
  - Every degree in the bipartite subgraph is 1 iff every indegree and outdegree is 1 in the induced subgraph.

Directing a Graph with Degree Constrains

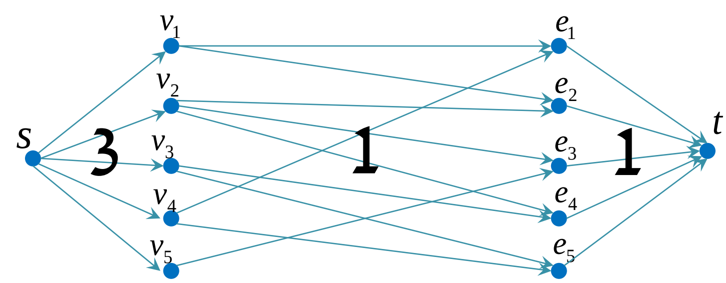
- **Problem.** Given an undirected graph $G = (V, E)$, which might not be simple, describe an algorithm that directs each of the edges of G, such that no outdegree is larger than 3.



$2 \geq d_{out}$

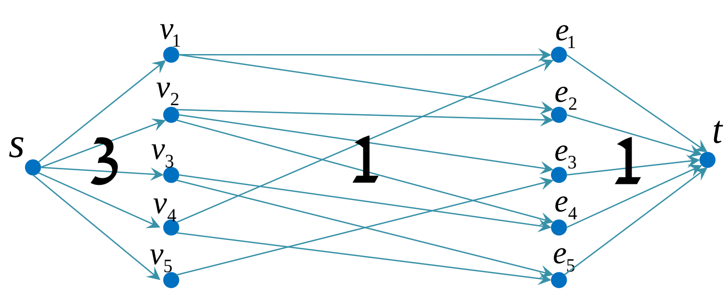
Solution

- We build a flow network:



A vertex for every element of V.	An edge for every vertex-edge pair that are adjacent in G.	A vertex for every element of E.

Solution (cont.)



- There is a valid orientation of the edges if and only if the size of the maximum flow is $|E|$.

Correctness (Sketch)

- There is a flow of size $|E|$ in the network.

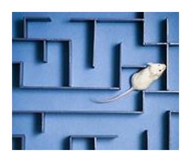
If and only if

- There is a flow where every vertex on the right side of the network receives a flow of 1 from one of its two neighbors.

If and only if

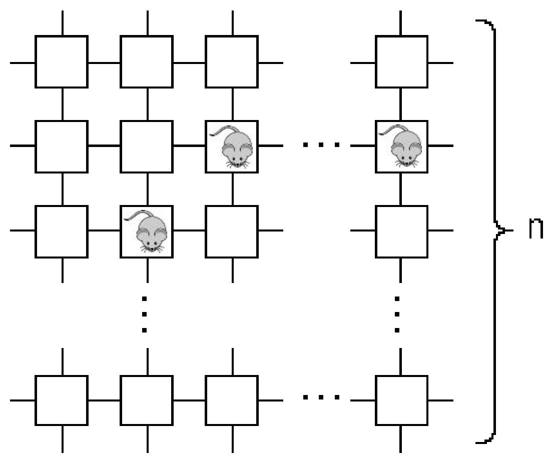
- There is an orientation of the edges of G such that every outdegree is at most 3.

A Mice Maze



- **Problem.** A maze consists of n^2 rooms in the form of an $n \times n$ matrix:
  - Between every two adjacent rooms there is a tunnel containing cheese.
  - Every room on the border of the maze contains a tunnel out, also with cheese.
  - m mice are placed in distinct rooms.
  - A mouse only enters tunnels with cheese in them, and then eats this cheese.
- Describe an algorithm for finding whether all m mice can escape the maze.

An Illustration



Solution

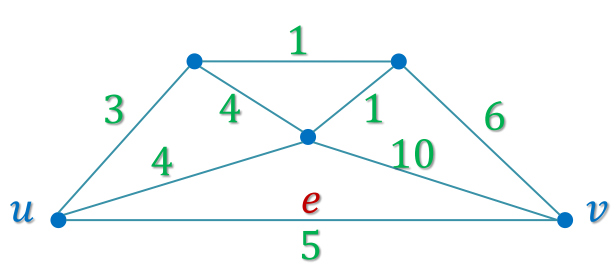
- We turn the maze into a flow network:
  - Every room is a vertex.
  - Every tunnel is a pair of anti-parallel edges.
  - The capacities are all 1.
  - The source is an additional vertex with an edge to every cell that contains a mouse.
  - The sink is an additional vertex with an edge from every exit tunnel.
  - All the mice can escape if and only if the maximum flow is of size m.

Back to MSTs

- **Problem.** Consider a connected undirected graph $G = (V, E)$, a weight function $w: E \to \mathbb{N}$, an edge $e \in E$, and an integer $k > 0$. Describe an efficient algorithm for checking whether we can remove at most k edges from G so that e is in every MST of the resulting graph.
- **Restrictions.**
  - G has to remain connected after the removal.
  - Since k is large, we are not allowed to try all $\sim n^k$ sets of at most k edges.

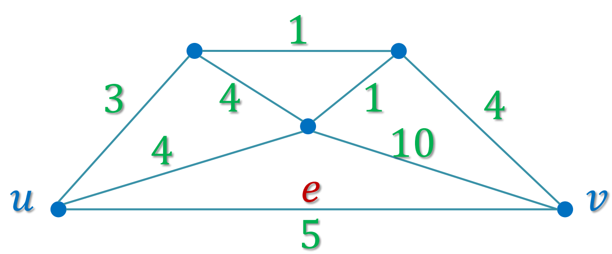
Warm-up

- Set $e = (u, v)$ and $d = w(e)$.
- When is e in every MST of G?
  - If in every other path between u and v there is an edge with a weight larger than d.



Warm-up #2

- Set $e = (u, v)$ and $d = w(e)$.
- When is e in every MST of G after removing at most one edge of E?
    - If after removing at most one edge every other path between u and v contains an edge with a weight larger than d.



Solution

- Set $e = (u, v)$ and $d = w(e)$.
- We wish to remove up to k edges, so that every other path between u and v contains an edge of weight larger than d.
- How can we do that?
    - Remove from G every $e' \in E$ with $w(e') > d$ (These are ignored, and not part of the chosen subset).
    - We get a problem that was solved in the previous lecture: Finding the minimum subset of edges such that its removal disconnects s from t.

The End: An Exciting New Discovery!