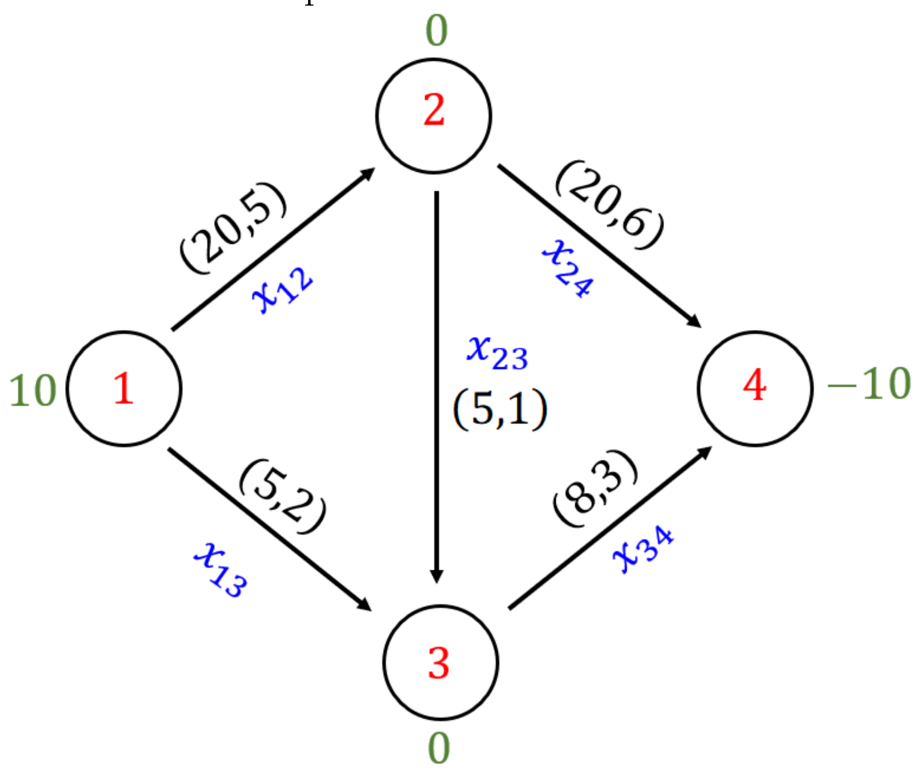
LECTURE 14: NETWORK PROBLEMS (II)

1. Recap: Network Problem

Last time: Coffee Transportation Problem:



$(20, 5)$ means: the max weight is 20 lbs and the cost/toll is \$5

Decision Variables: x_{ij} = coffee transported on the edge (i, j)

Date: Tuesday, October 25, 2022.

LP Problem:

$$\begin{aligned}
\min z &= 5x_{12} + 2x_{13} + 1x_{23} + 6x_{24} + 3x_{34} \\
\text{subject to } & x_{12} + x_{13} = 10 \\
& -x_{12} + x_{23} + x_{24} = 0 \\
& -x_{23} - x_{13} + x_{34} = 0 \\
& -x_{24} - x_{34} = -10 \\
& x_{12} \leq 20 \\
& x_{13} \leq 5 \\
& x_{23} \leq 5 \\
& x_{24} \leq 20 \\
& x_{34} \leq 8 \\
& x_{ij} \geq 0
\end{aligned}$$

First 4 equations come from conservation of flow (coffee in = coffee out), last 5 equations come from the capacity constraint (max weight)

Important Remark: Can write the conservation of flow as $Mx = b$

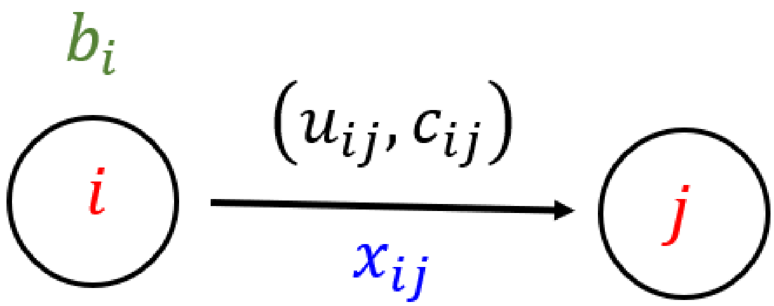
$$x = \begin{bmatrix} x_{12} \\ x_{13} \\ x_{23} \\ x_{24} \\ x_{34} \end{bmatrix} \quad M = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 1 & 0 \\ 0 & -1 & -1 & 0 & 1 \\ 0 & 0 & 0 & -1 & -1 \end{bmatrix} \quad b = \begin{bmatrix} 10 \\ 0 \\ 0 \\ -10 \end{bmatrix}$$

Here M is precisely the **oriented incidence matrix** of the graph

This makes sense because for the conservation of flow, we're basically asking ourselves "Which edges are going in/out of a given vertex?" which is precisely the definition of the oriented incidence matrix.

2. General Min Flow Problem

We can generalize this by using variables



Decision Variables: x_{ij} = amount transported on edge (i, j)

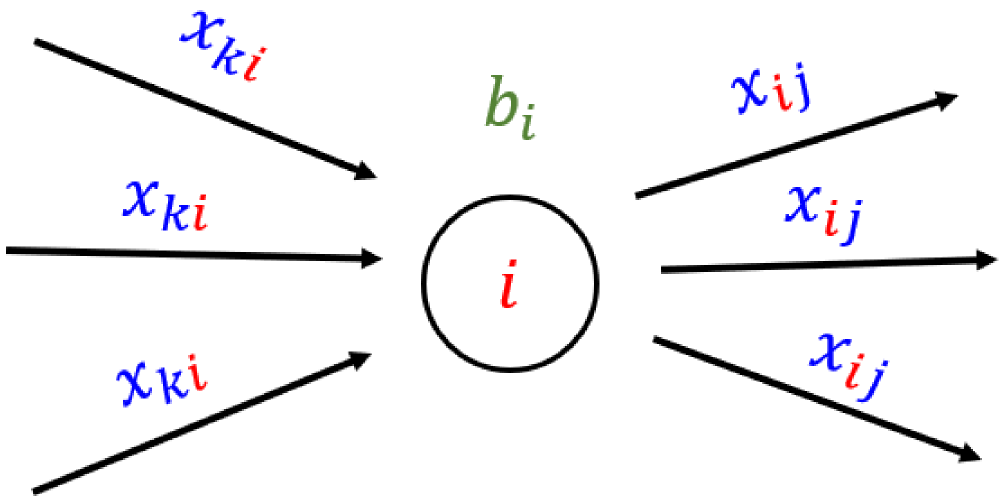
Objective Function: The cost on edge (i, j) is c_{ij} so

$$z = \sum_{i,j} c_{ij} x_{ij}$$

Here the sum runs over all edges (i, j)

Conservation of Flow:

In our example, the edges going out of 2 are x_{23} and x_{24}, whereas the edges going in of 2 are x_{12}



More generally, the edges going out of (i) are x_{ij} whereas the edges going out are x_{ki} so the conservation of flow just becomes

$$\underbrace{\sum_j x_{ij}}_{\text{Out}} - \underbrace{\sum_k x_{ki}}_{\text{In}} = b_i$$

Capacity Constraint: The max weight is u_{ij} and the min weight is l_{ij} (not depicted)

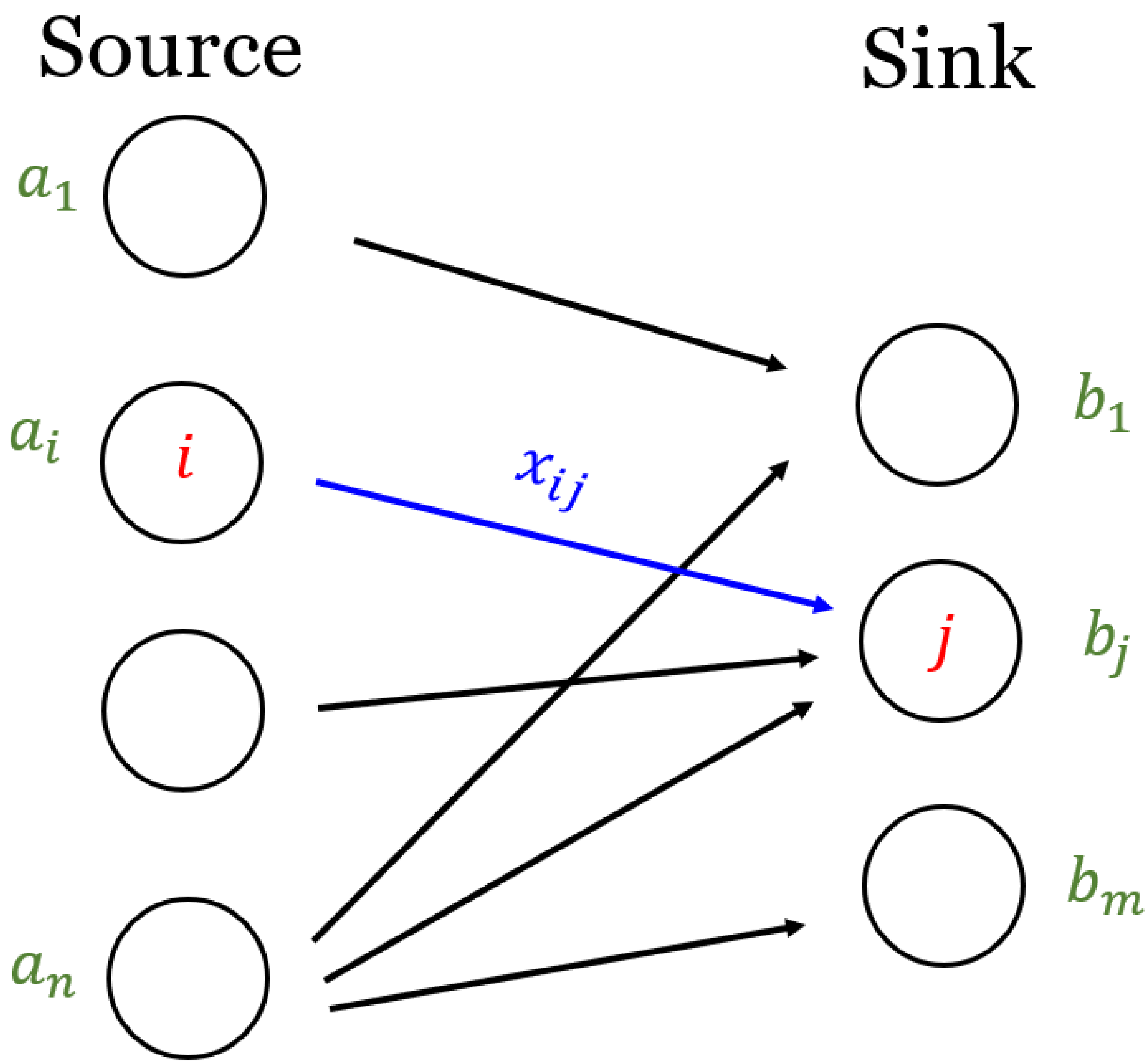
$$l_{ij} \leq x_{ij} \leq u_{ij}$$

LP Problem:

$$\begin{aligned} \min z = \sum_{i,j} c_{ij} x_{ij} & \\ \text{subject to } \sum_j x_{ij} - \sum_k x_{ki} = b_i & \quad \text{Conservation of flow} \\ l_{ij} \leq x_{ij} \leq u_{ij} & \quad \text{Capacity Constraint} \end{aligned}$$

3. Variation 1: Transportation Problem

Assume only two types of vertices: Sources (departures) and Sinks (arrivals), no intermediate cities



This is an example of a **bipartite graph** (bi = two, partite = classes)

Assume no max/min weight and also

$$\sum_{i=1}^{n} a_i = \sum_{j=1}^{m} b_j \qquad \text{(Supply = Demand)}$$

LP Problem:

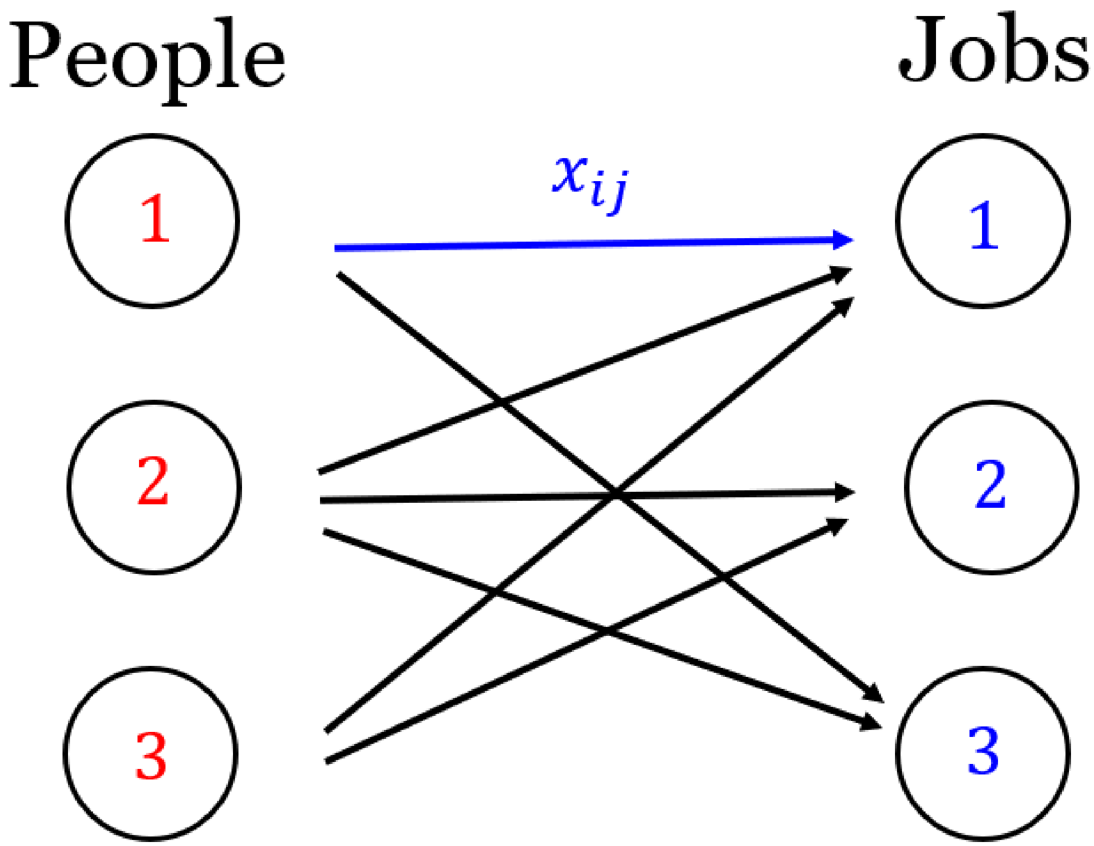
$$\begin{aligned} \min z &= \sum_{i,j} c_{ij} x_{ij} \\ \text{subject to } &\sum_{j=1}^{m} x_{ij} = a_i \\ &\sum_{i=1}^{n} -x_{ij} = -b_j \\ &x_{ij} \geq 0 \end{aligned}$$

4. Variation 2: Assignment Problem

As a special case of the above, suppose the left represents people and the right represents jobs, and your task is to assign people to jobs

Assume number of people = number of jobs, so $n = m$

$$x_{ij} = \begin{cases} 1 & \text{if person } i \text{ gets assigned to job } j \\ 0 & \text{otherwise} \end{cases}$$



LP Problem:

$$
\begin{aligned}
\min z &= \sum_{i,j} c_{ij} x_{ij} \\
\text{subject to} \quad \sum_{j=1}^{m} x_{ij} &= 1 \qquad \text{Every person gets assigned 1 job} \\
\sum_{i=1}^{m} x_{ij} &= 1 \qquad \text{Every job gets filled} \\
x_{ij} &= 0 \text{ or } 1
\end{aligned}
$$

Here c_{ij} is the cost of assigning worker i to cost j (think cost of training)

This is an example of an **integer programming problem**, where we require that x_{ij} be an integer. Those problems are usually much harder to solve than LP problems, but surprisingly here it's not that bad!

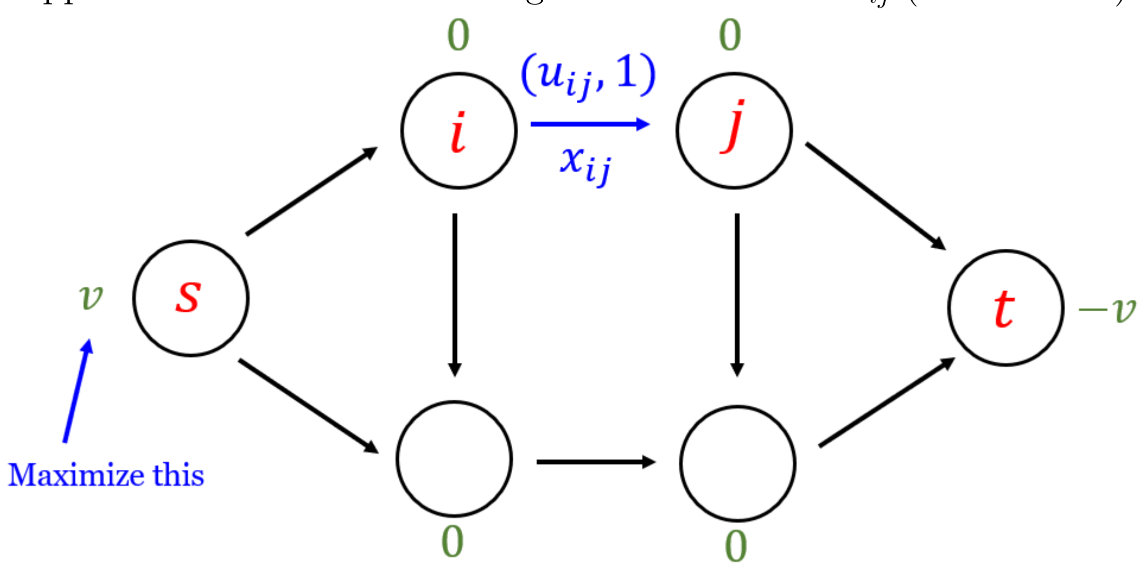
Note: This is usually written as a max problem, where c_{ij} is the salary or revenue generated by a worker.

5. Variation 3: Maximal Flow

This time assume only one source and one sink, and ask: What is the biggest supply we can provide *at the source*?

Think for example a country delivering as much food supply to another one, while not really caring about the cost of production.

Suppose the cost is 1 at each edge and max load is u_{ij} (no min load)



Here for the conservation of flow, we need to distinguish the cases $i = s$ (at the source), $i = t$ (at the sink), and otherwise.

LP Problem:

$$\max z = v$$

$$\text{subject to } \underbrace{\sum_j x_{ij}}_{\text{Out}} - \underbrace{\sum_k x_{ki}}_{\text{In}} = \begin{cases} v & \text{if } i = s \\ -v & \text{if } i = t \\ 0 & \text{otherwise} \end{cases}$$

$$0 \leq x_{ij} \leq u_{ij}$$

Note: Here the v is defined in terms of the x_{ij} via the conservation of flow, so we are indeed maximizing a function of x_{ij}

How annoying is this constraint?!? Luckily there's an insane way of getting around that!

Trick: Introduce another edge x_{ts} which goes *directly* from t to s (see picture below)

Then v (the max amount transported) is precisely x_{ts}, and the demand/supply at every vertex becomes 0.

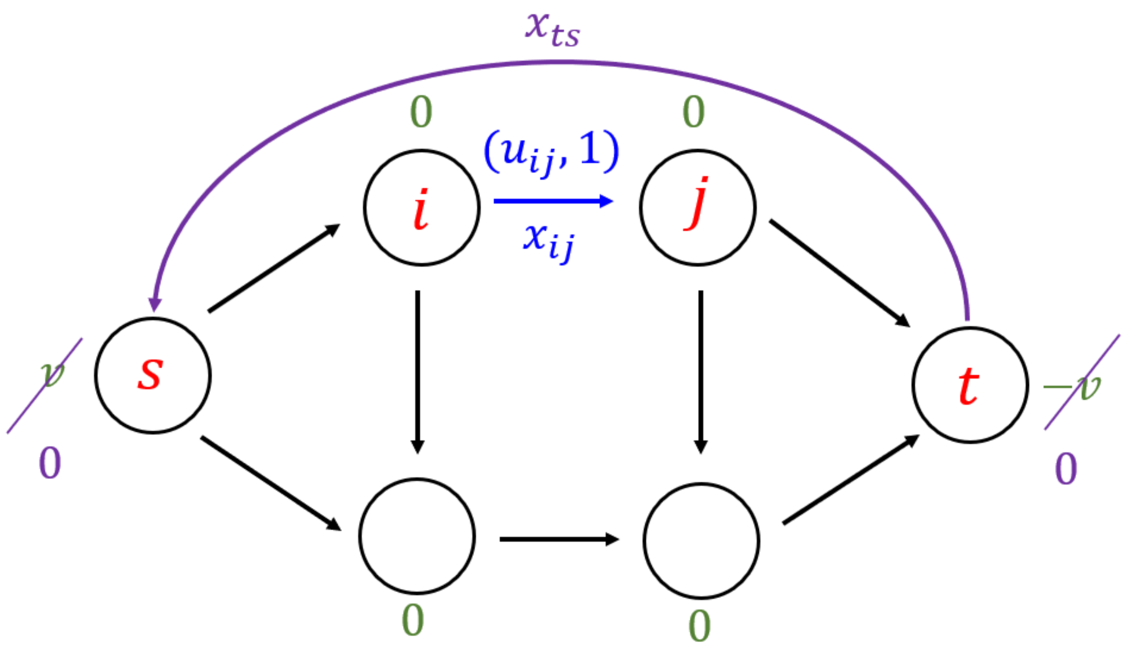
New LP Problem:

$$\max z = x_{ts}$$

$$\text{subject to } \sum_j x_{ij} - \sum_k x_{ki} = 0$$

$$0 \leq x_{ij} \leq u_{ij}$$

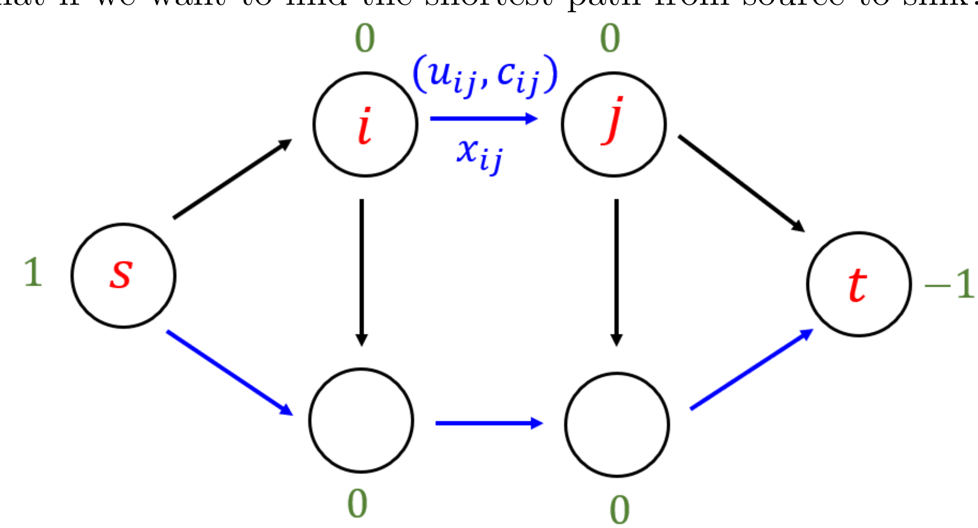
$$x_{ts} \geq 0$$



Note: The dual to the max flow problem is called **min cut**

6. Variation 4: Shortest Path

What if we want to find the shortest path from source to sink?



This is actually just the same problem as usual, except our supply/demand is 1. In other words, how fast does it take to ship *one* product from source to sink?

LP Problem:

$$\min z = \sum_{i,j} c_{ij} x_{ij}$$

$$\text{subject to } \sum_j x_{ij} - \sum_k x_{ki} = \begin{cases} 1 & \text{if } i = s \\ -1 & \text{if } i = t \\ 0 & \text{otherwise} \end{cases}$$

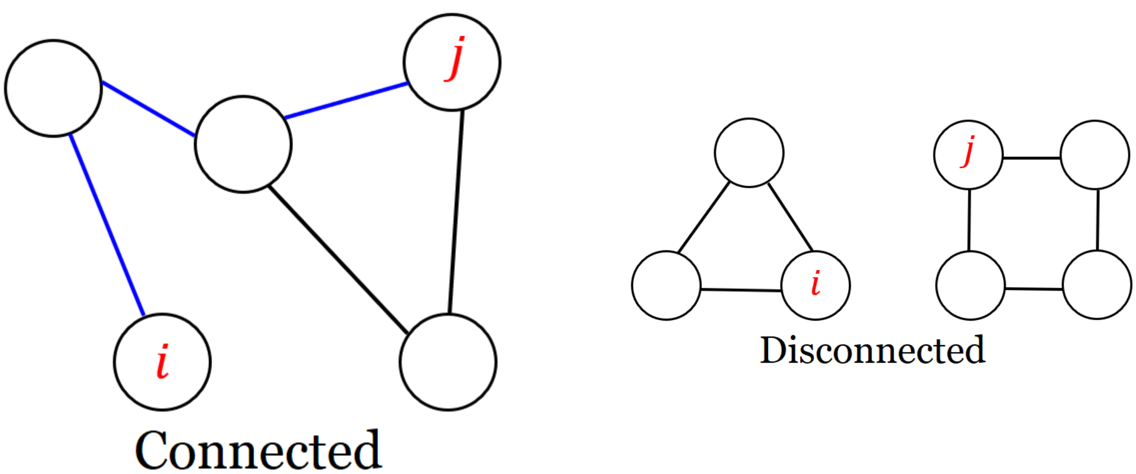
$$l_{ij} \leq x_{ij} \leq u_{ij}$$

7. Trees

Let's now focus on special types of graphs called **trees**

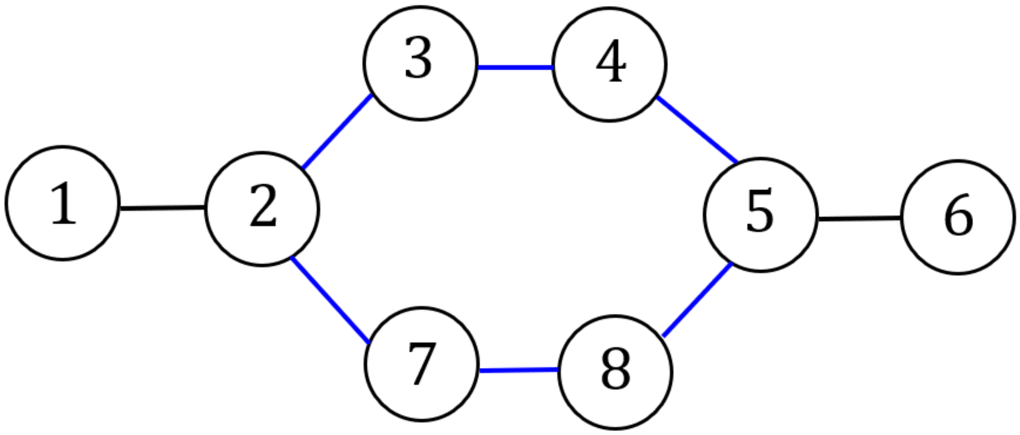
Definition:

A graph is **connected** if for any two vertices i and j, there is a path going from i to j. Else it is **disconnected**.



Definition:

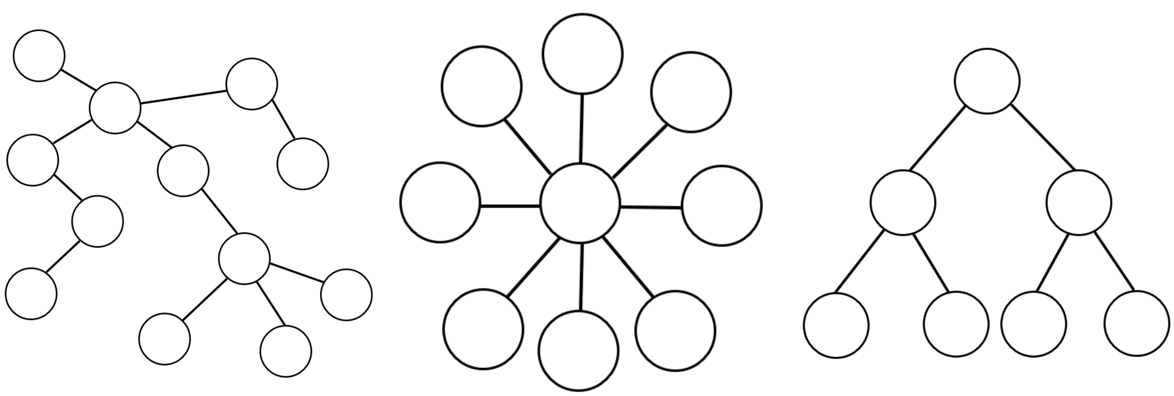
A **cycle** is a (nontrivial) path that starts at a vertex and ends at a same vertex.



Here a cycle is $2 \to 3 \to 4 \to 5 \to 8 \to 7 \to 2$

Definition:

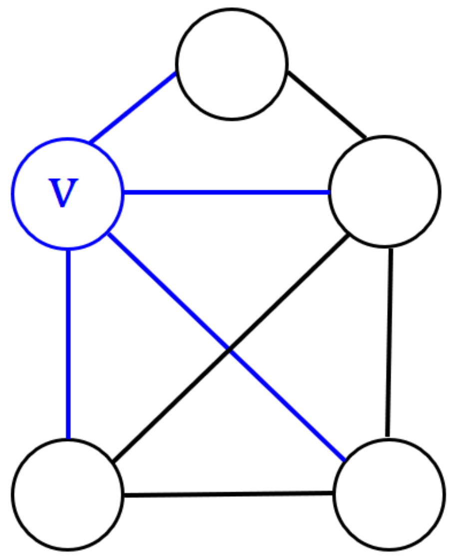
A **tree** is a connected graph that has no cycles.



Definition:

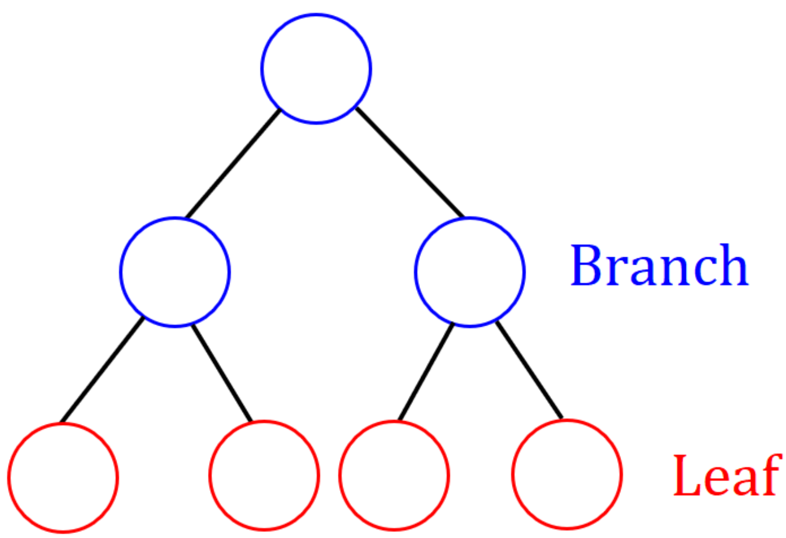
The **degree** of a vertex v is the number of edges connected to v

In the picture below, $\deg(v) = 4$



Definition:

A **leaf** of a tree is a vertex of degree 1, else it is a **branch**



And a collection of trees is a **forest** (lol)