MATH 566: Lecture 4 (08/29/2024)

Today: * More definitions
* Network representations

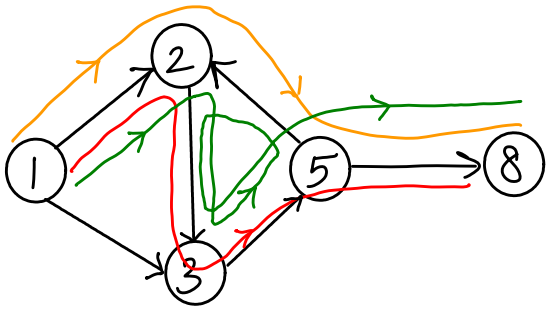
Definitions (contd..)

A **walk** in $G=(N,A)$ is a subgraph of G made of a sequence of nodes and arcs $i_1-a_1-i_2-a_2-\cdots a_{r-1}-i_r$ with $a_k=(i_k,i_{k+1})$ or $a_k=(i_{k+1},i_k)$ for $1\le k\le r-1$.

Thus, a walk need not necessarily be directed.

A **directed walk** is the oriented version of a walk with $a_k=(i_k,i_{k+1})$.

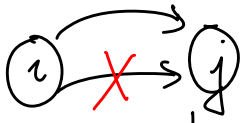
Here are some examples.



1-2-5-8 is a walk
1-2-3-5-8 is a directed walk

But 1-2-3-5-2-3-5-8 may not be considered a (directed) walk under this definition since arcs are repeated.

But some other book allows repeated arcs in walks. Also, look up "trail" and "circuit".

We assume there are no duplicate arcs: 

Notice that nodes could be repeated in a walk.

A **path** is a walk without repetition of nodes; and a **directed path** is a directed walk without repetition of nodes.

e.g., 1-2-5-8 is a path, 1-2-3-5-8 is a directed path.

An arc (i,j) in a path is a **forward arc** if i is visited before j and is a **backward arc** if j is visited before i.

For instance, in 1-2-5-8, $(1,2)$ and $(5,8)$ are forward arcs, while $(5,2)$ is a backward arc.

A directed path has no backward arcs.

Convention: A path with backward arcs called a **non**-directed path; an **un**directed path will be a path in an undirected graph.

If (i,j) is in a directed path, i is the **predecessor** of j, and we denote it by $\text{pred}(j) = i$.

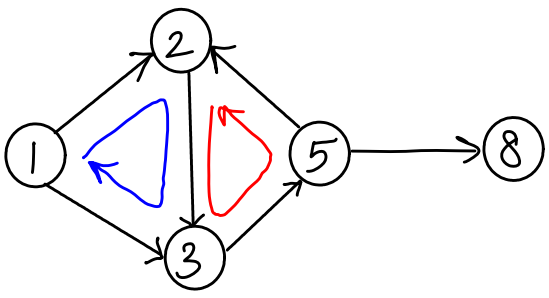
For the directed path 1-2-3-5-8, we set

$\text{pred}(8) = 5$
$\text{pred}(5) = 3$
$\text{pred}(3) = 2$
$\text{pred}(2) = 1$
$\text{pred}(1) = 0$ → marks the start of the directed path

We will use pred(·) indices as the standard data structure to represent paths – we will use a pred vector of length n.

A **cycle** is a path $i_1 - i_2 - \cdots - i_r$ with arc (i_r, i_1) or (i_1, i_r), denoted as $i_1 - i_2 - \cdots - i_r - i_1$.

A **directed cycle** is a directed path $i_1 - i_2 - \cdots - i_r$ with arc (i_r, i_1)



1-2-3-1 is a cycle
2-3-5-2 is a directed cycle.

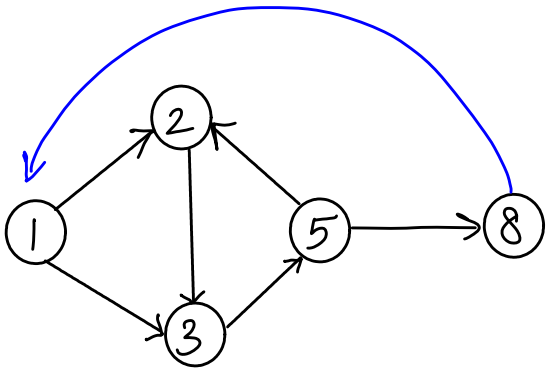
Many problems are more complicated when the network has directed cycles.

An **acyclic graph** is a graph with no directed cycles. So, non-directed cycles are permitted here. We get a tree if we avoid those cycles as well, assuming the graph is connected.

Connectivity

Nodes i and j are **connected** if there is a path from i to j. The (entire) graph is connected if every pair of vertices is connected. Else it is **disconnected**.

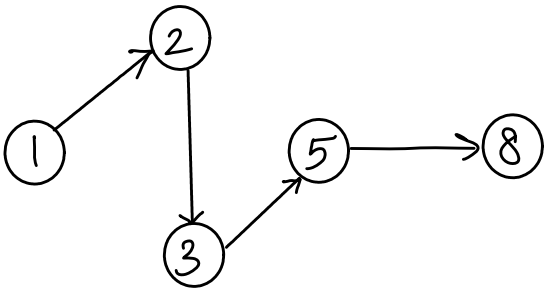
A graph is **strongly connected** if there is a directed path from every node i to every node j.



G is connected but not strongly connected. Adding $(8,1)$ makes it strongly connected.

A **tree** is a connected graph with no cycles. So, we avoid both directed and non-directed cycles in a tree.

e.g., T

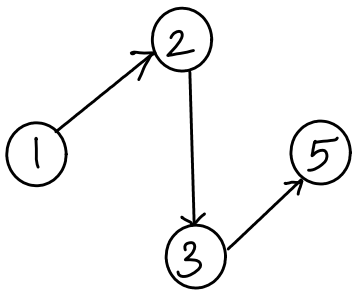


is a tree of the graph G introduced previously.

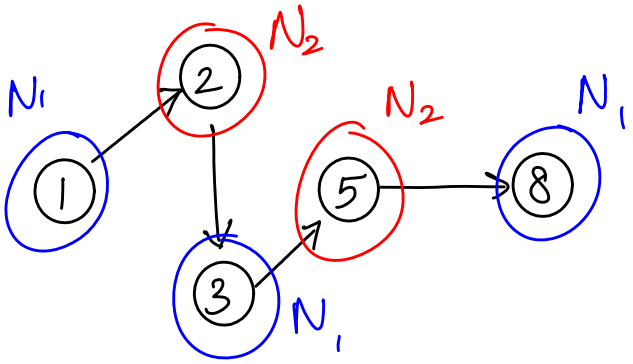
A tree T is a **spanning tree** of graph G if T is a spanning subgraph of G.

For instance, T' is not a spanning tree of G.

T'



Bipartite graph $G=(N,A)$ is **bipartite** if we can partition the node set N into subsets N_1 and N_2 such that $N = N_1 \cup N_2$, and each $(i,j) \in A$ has either $i \in N_1, j \in N_2$ or $i \in N_2, j \in N_1$.

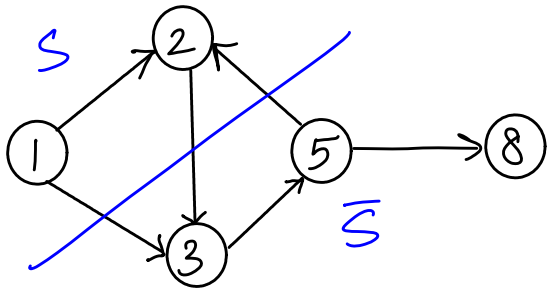


Note: A bipartite graph need not always be "drawn" in an obvious bipartite fashion, e.g., in the case of transportation problems.

In fact, one could prove that every tree is a bipartite graph — think about it!

A **cut** is a partition of the node set N into two parts S and $\bar{S}$.
"S-bar" or "S-complement".

Each cut defines a set of arcs $(i,j) \in A$ with $i \in S, j \in \bar{S}$ or $i \in \bar{S}, j \in S$.
forward arcs; backward arcs



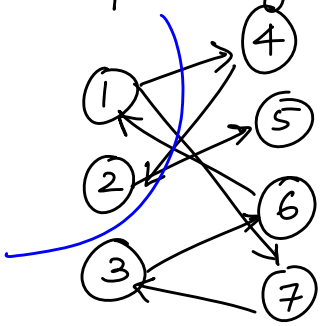
$S = \{1,2\}$, $\bar{S} = \{3,5,8\}$

$(1,3), (2,3)$: forward arcs

$(5,2)$: backward arc

Notice that we could talk about cuts for a bipartite graph just the same as in a general graph.

We will talk more about cuts when we introduce max flow in detail.

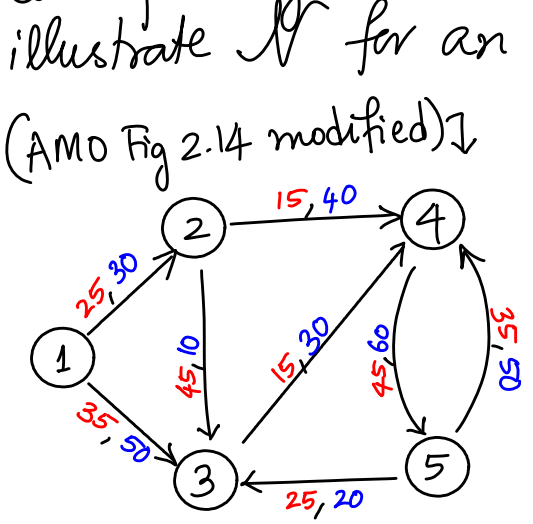


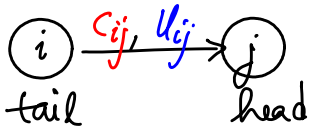
Network Representations

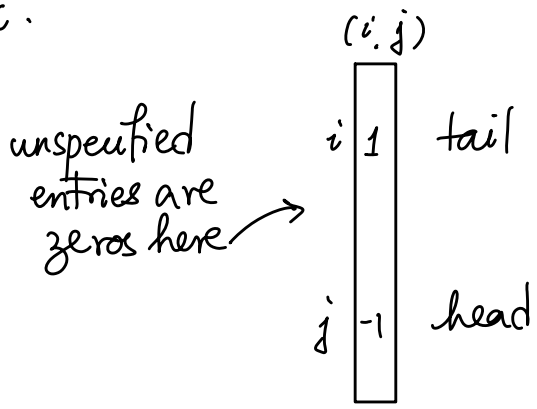
We need to represent $G = (N, A)$, along with data, i.e., $b(i)$, l_{ij}, u_{ij}, c_{ij}, etc. The ultimate goal is to be able to access and use all this info efficiently in algorithms. We consider several options for representing networks now.

Node-arc incidence matrix N

N is an $n \times m$ matrix with one row for each node, and one column for each arc, with entries in $\{-1, 0, 1\}$. The column of N corresponding to arc (i, j) is shown to the right. We also illustrate N for an example network.







$$N = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ (1,2) & (1,3) & (2,3) & (2,4) & (3,4) & (4,5) & (5,3) & (5,4) \end{array} \\ \begin{bmatrix} 1 & 1 & & & & & & \\ -1 & & & 1 & 1 & & & \\ & -1 & -1 & & & & -1 & \\ & & & & -1 & -1 & 1 & & -1 \\ & & & & & & -1 & 1 & 1 \end{bmatrix} \end{array}$$

↑ unspecified entries are zeros.

Lexicographic or dictionary ordering of arcs. This will be the default way in which we order arcs

AMO uses c, u, etc to denote these vectors

$\bar{c} = [25 \quad 35 \quad 45 \quad 15 \quad 15 \quad 45 \quad 25 \quad 35]$

$\bar{u} = [30 \quad 50 \quad 10 \quad 40 \quad 30 \quad 60 \quad 20 \quad 50]$

$\bar{b}$ → a 5-vector of supply/demand values.

My notation: lower case letters with a bar are used to denote vectors, e.g., $\bar{x}$, $\bar{c}$, $\bar{u}$, $\bar{b}$, etc.

Properties of $\mathcal{N}$

* each column has one +1 and one -1, in the rows corresponding to the tail and head node, respectively.
* # +1's in row i = outdegree(i) } # nonzeros in row i = degree(i).
  # -1's in row i = indegree(i) }
* the total # of nonzeros is $2m$, i.e., $\mathcal{N}$ is a very sparse matrix.

At the same time, $\mathcal{N}$ is a convenient representation of the network structure especially for proving various properties related to the network. For instance, the linear optimization model for min-cost flow can be written in matrix-vector form using this matrix.

Matrix version of the min-cost flow problem

Recall the linear optimization model for the min-cost flow problem:

$$\min \sum_{(i,j)\in A} c_{ij} x_{ij}$$

$$\text{s.t.} \quad \sum_{(i,j)} x_{ij} - \sum_{(j,i)} x_{ji} = b(i) \quad \forall i \in N$$

$$l_{ij} \le x_{ij} \le u_{ij} \quad \forall (i,j) \in A$$

Notation: $\bar{x}, \bar{c}, \bar{l}, \bar{u}$, are vectors representing $x_{ij}, c_{ij}, l_{ij}, u_{ij}$.

$$\bar{x} = \begin{bmatrix} \vdots \\ x_{ij} \\ \vdots \end{bmatrix}$$

$$\min \quad \bar{c}^T \bar{x} \qquad (T \rightarrow \text{transpose})$$

$$\text{s.t.} \quad \mathcal{N}\bar{x} = \bar{b} \qquad (\bar{b} \rightarrow \text{vector of } b(i)\text{'s})$$

$$\bar{l} \le \bar{x} \le \bar{u}$$

vectors are columns by default, hence the transpose here.

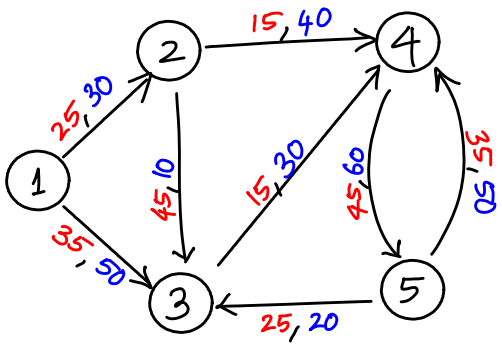
We now introduce another matrix representing the network.

Node-Node adjacency matrix $\mathcal{H}$

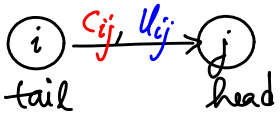
This is an $n \times n$ matrix $\mathcal{H} = [h_{ij}]$, with

$$h_{ij} = \begin{cases} 1 & \text{if } (i,j) \in A \\ 0 & \text{otherwise} \end{cases}$$

→ row of the tail node i and column of head node j.



$$\mathcal{H} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix} \end{array}$$



Properties of the node-node incidence matrix $\mathcal{H}$

* each arc is represented by a 1 in $\mathcal{H}$.
* the # 1's in row i = outdegree (i)
  the # 1's in column j = indegree (j)

  } scan row i to get outarcs of node i; scan column j to get inarcs of node j.

* $\mathcal{H}$ has n^2 elements, m of which are nonzero.
  So, $\mathcal{H}$ is an efficient data structure when $m \gg n$. ("much greater than")

We store l_{ij}, u_{ij}, and c_{ij} as separate $n \times n$ matrices L, U, and C.

Unless the network is really dense, i.e, has lots of arcs connecting the given nodes, $\mathcal{H}$ could have a lots of zeros (and so will L, U, and C). Hence we look for even more efficient data structures.

It must be mentioned that there are standard ways of handling sparse matrices, e.g., the sparse package in Octave/Matlab. But we would ideally like to avoid the overhead associated with such operations by using more efficient representations to start with.

We will talk about efficient ways of representing such lists in general, and (out)arc lists in particular, in the next lecture.