CISC 1400: Discrete Structures

Chapter 9
Graphs

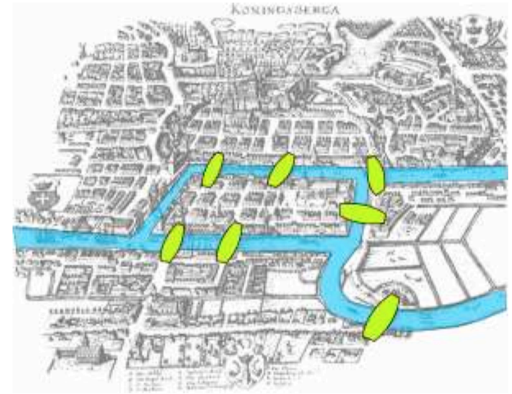
Arthur G. Werschulz

Fordham University Department of Computer and Information Sciences

Summer, 2019

Introduction

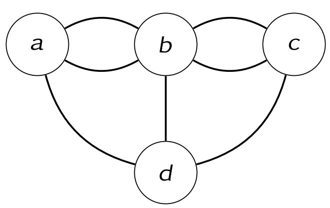
The city of Königsberg:



Can one "einen Spaziergang machen" that crosses each bridge in Königsberg exactly once?

Introduction (cont'd)

Get rid of all the "non-essentials", get a (multi)graph:



Introduction (cont'd)

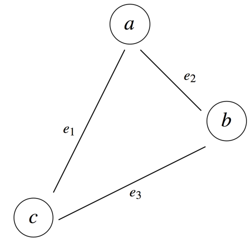
- Situations we can visualize using graphs
  - People in a social network
  - Cities in a country
  - Jobs in a "to-do" list
  - Electrical connections
  - The Internet
- Questions we can ask:
  - Can we visit every vertex and end up where we started?
  - Is there any vertex we cannot reach from other places?
  - What is the shortest distance between two vertices?
  - How to connect all vertices using the fewest number of edges? the minimal cost?

Outline

- Graph notation
- Euler trails and circuits
- Weighted graphs
- Minimum spanning trees
- Matrix notation for graphs

Graph notation: vertices and edges

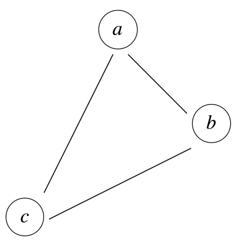
- Graph $G = (V, E)$
  - V: set of *vertices*
  - E: set of *edges*
  - $\{v, w\} \in E$ means that $v, w \in V$ are *connected*
- For the graph



  - $V = \{a, b, c\}$,
  - $E = \{e_1, e_2, e_3\}$.

Graph notation: vertices and edges (cont'd)

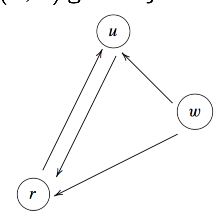
- Sometimes we simply indicate edges by the vertices they connect.
- For the graph



  - $V = \{a, b, c\}$,
  - $E = \{\{a, b\}, \{b, c\}, \{a, c\}\}$.

Graph notation: directed and undirected graphs

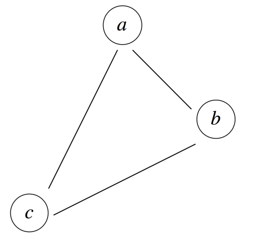
- $G = (V, E)$ is a *directed graph* (or *digraph*) if the edge from v_1 to v_2 can only be traversed in that direction.
- The graphs in previous examples were *undirected*.
- For an undirected graph, edge connecting distinct $v, w \in V$ is $\{v, w\}$.
- For a directed graph, edge connecting $v, w \in V$ is (v, w).
- For the graph $G = (V, E)$ given by



  - $V = \{u, r, w\}$,
  - $E = \{(r, u), (u, r), (w, r), (w, u)\}$.

Graph notation: complete graphs

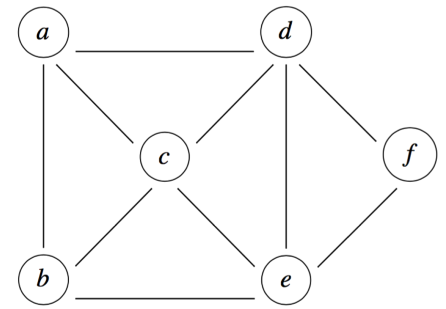
- A graph is *complete* if all possible edges are present.
- An undirected graph $G = (V, E)$ is complete if $\{v, w\} \in E$ for any distinct $v, w \in V$. The graph



  is complete.
  An undirected graph with n vertices has $\frac{1}{2}n(n-1)$ edges.
- A directed graph $G = (V, E)$ is complete if $(v, w) \in E$ for any distinct $v, w \in V$, i.e.if $E = V \times V$.
  A directed graph with n vertices has n^2 edges.

Graph notation

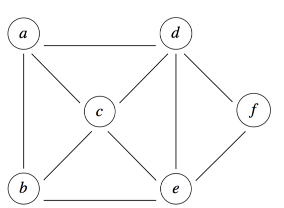
For the graph



- How many vertices are there? Six.
- How many edges? Ten
- Directed or undirected? Undirected
- Is this graph complete? No.

Graph notation

Let $G = (V, E)$. If $V' \subseteq V$ and $E' \subseteq E$, then $G' = (V', E')$ is a *subgraph* of G.
Consider once again the graph



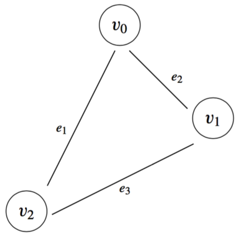
Find the largest n such that this graph has a complete subgraph with n vertices.
The subgraph with vertex set $\{a, b, c\}$ is complete.
There are no 4-element vertex sets yielding a complete subgraph.
So $n = 3$.

Euler trails and circuits

- A *walk* in a graph $G = (V, E)$ is a sequence of vertices $v_0, v_1, \ldots, v_n \in V$ and edges $e_1, e_2, \ldots, e_n \in E$, where each e_i is an edge connecting v_{i-1} and v_i.
- A *trail* is a walk in which no edge is traversed more than once.
- A *path* is a walk in which no vertex is traversed more than once.
- A *circuit* is a trail that begins and ends at the same vertex.
- A *cycle* is a circuit in which the start vertex is the *only* vertex appearing more than once.

Euler trails and circuits (cont'd)

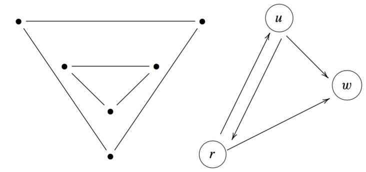
For the graph



- How many trails are there from v_0 to v_2?
  Two: v_0, e_2, v_1, e_3, v_2 and v_0, e_1, v_2.
- How many circuits start at v_0?
  Only one: $v_0, e_2, v_1, e_3, v_2, e_1, v_0$
- How many cycles are there?
  Infinitely many: repeat the circuit above over and over.

Euler trails and circuits (cont'd)

A graph is *connected* if there is a walk from any vertex to any other vertex. Which of these graphs is connected?



Neither of them!

Euler trails and circuits (cont'd)

- An *Euler trail* is a trail that includes every edge in the graph exactly once.
- An *Euler circuit* is a circuit that includes every edge in the graph exactly once.
- When does a graph (or multigraph) have an Euler trail? an Euler circuit?

Euler trails and circuits (cont'd)

(Only covering undirected graphs here; digraphs are a bit more complicated.)

- Every vertex in an Euler circuit must have an entry edge and an exit edge.
- The *degree* of a vertex is the number of edges that it has.
- A vertex is *even* or *odd* if its degree is even or odd, respectively.
- All the vertices in an Euler circuit must be even (or else you'd be stuck at a vertex).
- So if a graph G has any odd vertices, then G cannot contain an Euler circuit.
- The converse is also true (but we won't prove it here).

Euler trails and circuits (cont'd)

- What about Euler trails?
- Suppose $G = (V, E)$ has Euler trail that's not a circuit, which means that there must be some odd vertices.
- If you add up the degrees of all the vertices, you get $2|E|$.
- Hence the number of odd vertices must be an even number.
- Furthermore, an odd vertex must be an endpoint of a non-circuit Euler trail.
- So there must be exactly two odd vertices.
- Add an extra "artificial" edge between them, getting a new graph.
- Now all the vertices in the new graph are even.
- Hence the new graph has an Euler circuit.
- Remove the artificial edge from the Euler circuit.
- You now have an Euler trail, whose terminal vertices are the two odd vertices.

Euler trails and circuits (cont'd)

Summarizing these results:

Let $G = (V, E)$ be a graph.

- If every vertex in V is even, then G has an Euler circuit.
- If exactly two vertices $v, w \in V$ are odd, then G has an Euler trail, starting at v and ending at w. Moreover, G does not have an Euler circuit.
- Otherwise, G has neither an Euler trail nor an Euler circuit.

Now suppose we change "edge" to "vertex", i.e., we look for a path that visits each *vertex* exactly once, a *Hamiltonian* (perhaps more properly, a *Rudrata*) circuit or trail.

Note the following:

- We can solve Euler circuit/trail problem in polynomial time.
- However:
  - No polynomial-time algorithm exists to solve the Hamiltonian circuit/trail problem in polynomial time, but
  - Nobody has shown that this problem cannot be solved in polynomial time.
  - This problem is NP-complete.

Weighted graphs

We sometimes label edges of a graph by a number:

- Mileage between cities.
- Cost of sending a message between cell towers.
- Time to send an internet packet between two hosts.

We call this number a *weight*.

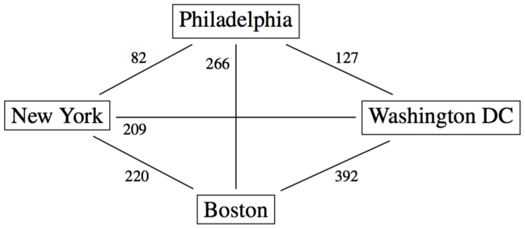
A graph, all of whose edges have weights, is called a *weighted graph*.

Let $G = (V, E)$ be a weighted graph, and let w_e denote the weight associated with the edge $e \in E$. Then the total *weight* w_G of G is given by

$$w_G = \sum_{e \in E} w_e.$$

Weighted graphs (cont'd)

For example, let G be the weighted graph



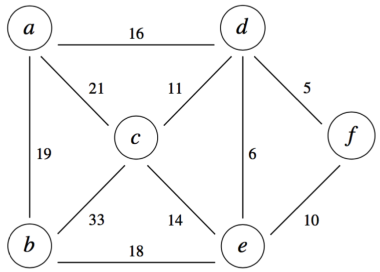
Then $w_G = 82 + 127 + 209 + 220 + 266 + 392 = 1296$. The minimal weight of a trip from New York to Washington, DC is

$$\min\{82 + 127, 209, 220 + 392\} = \min\{209, 209, 612\} = 209$$

and is given either by a direct trip, or a trip through Philadelphia.

Minimum spanning trees

Suppose a building has rooms, and the cost of connecting them by fiber-optic cable is given by the following graph:

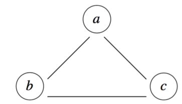


What's the minimal-cost netwrok that connects all the rooms?

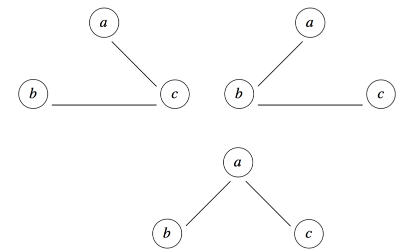
Minimum spanning trees (cont'd)

Let $G = (V, E)$ be a connected graph. A subgraph $T = (V, E')$ is a *spanning tree* for G is T is connected and acyclic.

Example: What are the spanning trees for the graph



Solution: There are three spanning trees, given by

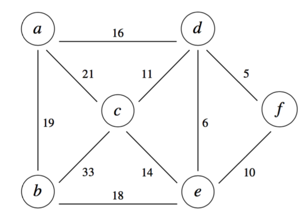


Minimum spanning tree (cont'd)

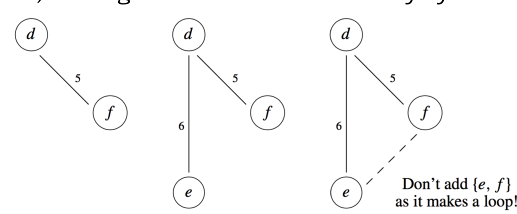
- *Minimal spanning tree* (MST) of a graph is a spanning tree having minimal weight among all spanning trees.
- Exhaustive listing: takes too long!!
- *Prim's algorithm* is far more efficient:
  1. Sort the edges by increasing weight.
  2. Working from smallest to largest, add each edge to the MST iff it doesn't make a cycle.
- How efficient?
  - Step 1 can be done with cost $O(|E| \log |E|)$.
  - Step 2 only requires $\min\{|V| - 1, |E|\}$ iterations.

Minimum spanning trees (cont'd)

Example: Find an MST for

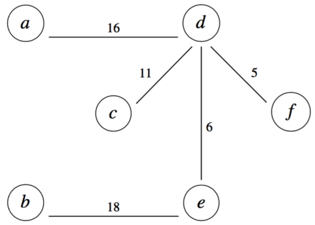


Solution: Order the edges by increasing weight. Now add them one by one, making sure to not introduce any cycles:

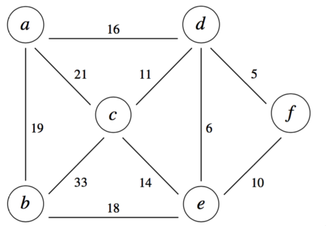


Minimum spanning trees (cont'd)

Continuing on, we find that



is an MST for our graph



Matrix notation for graphs

- How have we represented graphs so far?
  - By a picture.
  - By an explicit list of vertices and edges.
- Why might this not be good enough?
  - Too tedious if graph is big.How big is "big"?

    As of April 2019, Facebook has around 2.4 billion users.
  - How to input into a computer, even if size isn't a problem?
- One idea: *adjacency matrix* (also called *incidence matrix*)

Matrix notation for graphs (cont'd)

- You may have seen matrices such as

$$\begin{bmatrix} 2 & 3 & -1 & 5 \\ 7.01 & -4.2 & 13.77 & 14.92 \\ -6 & 42 & 1.7 & 0.5 \\ 0 & 19 & -3.14 & 2 \end{bmatrix}$$

  in which all the entries are numbers, as well as operations such as matrix addition and matrix multiplication.
- Here, we'll study *Boolean matrices* (or *bit matrices*) such as

$$\begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix}$$

  as well as operations such as *Boolean matrix addition* and *Boolean matrix multiplication*.

Matrix notation for graphs (cont'd)

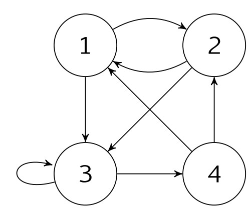
Main ideas:

- Number of rows = number of columns = number of nodes
- Use 0 or 1 in row i and column j to indicate absence or presence of an edge from node i to node j.

So the matrix

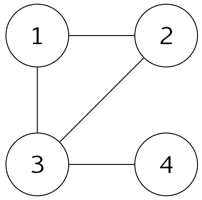
$$\begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix}$$

represents the digraph



Matrix notation for graphs (cont'd)

What about undirected graphs, such as represents the graph



This would be represented as

$$\begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad \text{or, better yet:} \quad \begin{bmatrix} & 1 & 1 & 0 \\ & & 1 & 0 \\ & & & 1 \\ & & & \end{bmatrix}$$

Don't save redundant information!

Matrix notation for graphs (cont'd)

Definition

Let $G = (V, E)$ be a graph, where $V = \{v_1, v_2, \dots, v_n\}$ is an ordering of V. The *adjacency matrix* of G is the $n \times n$ array of zeros and ones, whose entry in row i and column j is given by

- 1 if there is an edge from v_i to v_j, and
- 0 if there is not an edge from v_i to v_j.

If we let M denote the adjacency matrix of G, then

$$M = \begin{bmatrix} m_{1,1} & m_{1,2} & \dots & m_{1,n} \\ m_{2,1} & m_{2,2} & \dots & m_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ m_{n,1} & m_{n,2} & \dots & m_{n,n} \end{bmatrix},$$

where for each $i, j \in \{1, \dots, m\}$, we have

$$m_{i,j} = \begin{cases} 1 & \text{if there is an edge from } v_i \text{ to } v_j, \\ 0 & \text{otherwise.} \end{cases}$$

Matrix notation for graphs (cont'd)

- The adjacency matrix of a n-vertex digraph can be stored using n^2 bits.
- The adjacency matrix of an n-vertex undirected graph can be stored using $\frac{1}{2}n(n-1)$ bits. (Why?)
- This is optimal if no a priori connectivity knowledge.
- In practice, large graphs are *sparse*, i.e., $|E| \ll |V|$ when $|V|$ is large.
- Consider the Facebook graph, with $|V| \doteq 2.4 \times 10^9$. The average Facebook user has around 338 friends. So the Facebook graph has around 8.1×10^{11} edges. But the adjacency matrix would use up around 2.9×10^{18} bits, which is roughly 3 million times bigger than the number of edges!!
- You'll learn about more efficient ways (e.g., adjacency lists) to store sparse graphs in future courses.

Matrix notation for graphs (cont'd)

- Social networks often suggest new connections to members. How do these suggestions arise?
- Friend of a friend (FOAF), friend of a FOAF (FOAFOAF), FOAFOAFOAF, etc.
- Can use adjacency matrix to help compute same.
- But first, a slight detour ...

Matrix notation for graphs (cont'd)

Definition

Let $A = [a_{i,j}] 1 \leq i,j \leq n$ and $B = [b_{i,j}] 1 \leq i,j \leq n$ be Boolean matrices. The *Boolean sum* $A \vee B$ of A and B is the $n \times n$ Boolean matrix $C = [c_{i,j}]_{1 \leq i,j \leq n}$, with

$$c_{i,j} = a_{i,j} \vee b_{i,j} \qquad (1 \leq i,j \leq n)$$

If A and B are the matrices of graphs $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ and $C = A \vee B$, then

$$c_{i,j} = 1 \iff (v_i, v_j) \in E_1 \text{ or } (v_i, v_j) \in E_2 \iff (v_i, v_j) \in E_1 \cup E_2.$$

Matrix notation for graphs (cont'd)

Definition

Let $A = [a_{i,j}] 1 \leq i,j \leq n$ and $B = [b_{i,j}] 1 \leq i,j \leq n$ be Boolean matrices. The *Boolean product* $A * B$ of A and B is the $n \times n$ Boolean matrix $C = [c_{i,j}]_{1 \leq i,j \leq n}$, with

$$\begin{aligned} c_{i,j} &= \bigvee_{1 \leq k \leq n} a_{i,k} \wedge b_{k,j} \\ &= (a_{i,1} \wedge b_{1,j}) \vee (a_{i,2} \wedge b_{2,j}) \vee \cdots \vee (a_{i,n} \wedge b_{n,j}) \end{aligned}$$

for $1 \leq i,j \leq n$.

If A and B are the matrices of graphs $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ and $C = A * B$, then

$$c_{i,j} = 1 \iff \exists\, v_k \in V \text{ such that} (v_i, v_k) \in E_1 \text{ and } (v_k, v_j) \in E_2.$$

Matrix notation for graphs (cont'd)

Example

Let

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \qquad \text{and} \qquad B = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

Let's calculate ...

$$A \vee B = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \qquad A * B = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \end{bmatrix} \qquad B * A = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Matrix notation for graphs (cont'd)

Some properties of Boolean matrix operations:

- $\vee$ and $*$ are *associative*:

  $$(A \vee B) \vee C = A \vee (B \vee C) \qquad \text{and} \qquad (A * B) * C = A * (B * C)$$

  The former is easy to prove, the latter is messy.
- $\vee$ is *commutative*:

  $$A \vee B = B \vee A$$

  But $*$ is not commutative

  $$A * B \neq B * A \qquad \text{in general}$$

- Other laws (e.g., distributive) as well.

Matrix notation for graphs (cont'd)

If M is an $n \times n$ Boolean matrix, we write

$$M^{[2]} = M * M$$

Let $G = (V, E)$ be the graph determined by M. Then

$$\begin{aligned} m_{i,j}^{[2]} = 1 &\iff \exists\, v_k \in V \text{ such that} (v_i, v_k) \in E \text{ and } (v_k, v_j) \in E \\ &\iff \text{there is a path of length 2 from } v_i \text{ to } v_j \end{aligned}$$

We write

$$M^{[3]} = M * M * M = M * M^{[2]}$$

Then

$$\begin{aligned} m_{i,j}^{[3]} = 1 &\iff \exists\, v_k \in V \text{ such that } (v_i, v_k) \in E \text{ and} \\ &\quad\ \exists \text{ path of length 2 from } v_i \text{ to } v_k \\ &\iff \text{there is a path of length 3 from } v_i \text{ to } v_j \end{aligned}$$

Matrix notation for graphs (cont'd)

Recall that M is an $n \times n$ Boolean matrix. Continuing on, we write

$$M^{[4]} = M * M * M * MM = M * M^{[3]}$$

Then

$$\begin{aligned} m_{i,j}^{[4]} = 1 &\iff \exists\, v_k \in V \text{ such that } (v_i, v_k) \in E \text{ and} \\ &\quad\ \exists \text{ path of length 3 from } v_i \text{ to } v_k \\ &\iff \text{there is a path of length 4 from } v_i \text{ to } v_j \end{aligned}$$

More generally, for any $n, \ell \in \mathbb{Z}^+$ and any $n \times n$ Boolean matrix M, let

$$M^{[\ell]} = \overbrace{M * M * \cdots * M}^{\ell \text{ times}} = \begin{cases} M * M^{[\ell-1]} & \text{if } \ell \geq 2, \\ M & \text{if } \ell = 1. \end{cases}$$

Now denote $M^{[\ell]} = [m_{i,j}^{[\ell]}]_{1 \leq i,j \leq n}$. Then $m_{i,j}^{[\ell]} = 1$ if and only if there is path of length ℓ connecting vertex v_i with vertex v_j.

Matrix notation for graphs (cont'd)

Example

Suppose that M is the adjacency matrix for the Facebook friendship (undirected) graph. Then

- $M^{[2]}$ is the adjacency matrix for FOAF.
- $M^{[3]}$ is the adjacency matrix for FOAFOAF $= (\text{FOA})^2\text{F}$.
- $M^{[4]}$ is the adjacency matrix for FOAFOAFOAF $= (\text{FOA})^3\text{F}$.
- In general, $M^{[\ell]}$ is the adjacency matrix for $(\text{FOA})^{\ell-1}\text{F}$, i.e., "friendship chains" of length ℓ.

How far can this process extend? Let n denote the number of Facebook users. Then the length of a friendship chains is at most $n - 1$.

Matrix notation for graphs (cont'd)

Example (cont'd)

Recall that M is the adjacency matrix for the Facebook friendship (undirected) graph, so that $M^{[\ell]}$ is the adjacency matrix for "friendship chains" of length ℓ.
How far can this process extend? Let n denote the number of Facebook users. Then the length of a friendship chains is at most $n - 1$.
Define the *reachability matrix* M^* of the undirected graph determined by M to be

$$M^* = M \vee M^{[2]} \vee M^{[3]} \vee \cdots \vee M^{[n-1]}$$

Then two Facebook members (labeled i and j) are connected by a friendship chain iff $m_{i,j}^* = 1$.

Matrix notation for graphs (cont'd)

Example (cont'd)

Suppose we had a social network allowing one-way links, whose adjacency matrix is

$$M = M^{[1]} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

We find that

$$M^{[2]} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad M^{[3]} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Note that $M^{[4]} = M^{[5]} = \cdots = \mathbf{0}$. Why?

Matrix notation for graphs (cont'd)

Example (cont'd)

So

$$\begin{aligned} M^* &= M \vee M^{[2]} \vee M^{[3]} \\ &= \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \vee \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \vee \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \\ &= \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \end{aligned}$$

Cost of Boolean matrix operations

Count the number of Boolean operations ($\wedge$ and $\vee$) needed to compute Boolean sum and product, as well as reachability matrix.

Let A and B be $n \times n$ bit matrices.

Cost of $A \vee B$? Simply n^2 $\vee$-operations, so cost is $O(n^2)$.

Cost of $A * B$? THere are n^2 entries, each having the form

$$\bigvee_{1 \le k \le n} a_{i,k} \wedge b_{k,j}$$

To calculate just one of these, need n $\wedge$-operations and $n-1$ $\vee$-operations. So total cost is n^3 $\wedge$ operations and $n(n-1)$ $\vee$ operations, which is $O(n^3)$.

Cost of Boolean matrix operations (cont'd)

Cost of A^*? Recall that we can compute the Boolean powers of A via

$$A^{[\ell]} = \begin{cases} A * A^{[\ell-1]} & \text{if } \ell \ge 2, \\ A & \text{if } \ell = 1. \end{cases}$$

We can compute $A^{[2]}, A^{[3]}, \ldots, A^{[n-1]}$ as $n-1$ Boolean matrix products, each of which has cost $O(n^3)$. So their total cost is $O(n^4)$.

Having computed $A^{[2]}, A^{[3]}, \ldots, A^{[n-1]}$, we then compute

$$A^* = A^{[1]} \vee A^{[2]} \vee A^{[3]} \vee \cdots \vee A^{[n-1]}$$

as $n-2$ Boolean matrix sums, each having cost $O(n^2)$.

So this final phase has cost $O(n^3)$.

Hence total cost of computing A^* is $O(N^4)$… expensive!

But can compute reachability matrix (or *transitive closure*) with cost $O(n^3)$ via *Floyd-Warshall algorithm*.