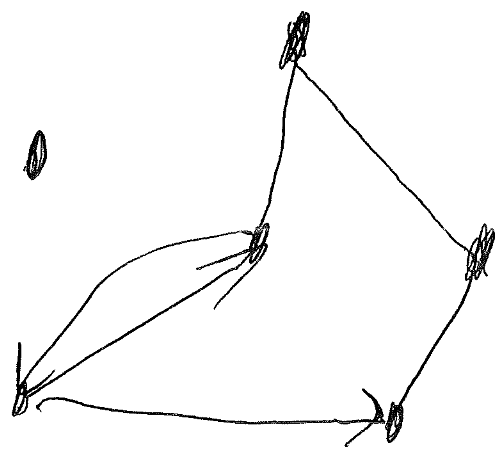
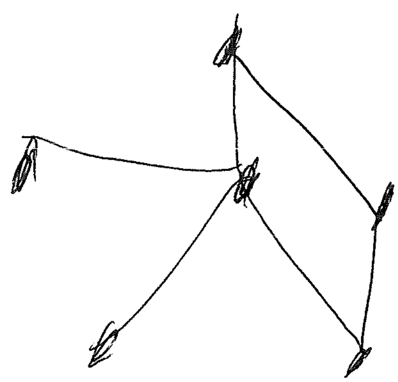
Lecture 3

More With Graphs

Recall



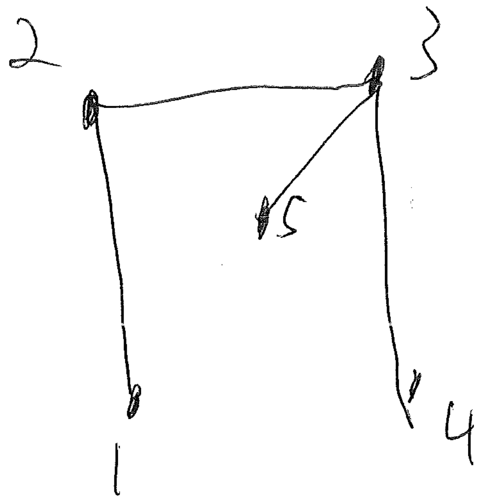
This is a digraph



Adjacency Matrix

For a graph G, we define the adjacency matrix $A_{ij} = \begin{cases} 1 & \text{if } \{i,j\} \in E \\ 0 & \text{o.w.} \end{cases}$

Ex.



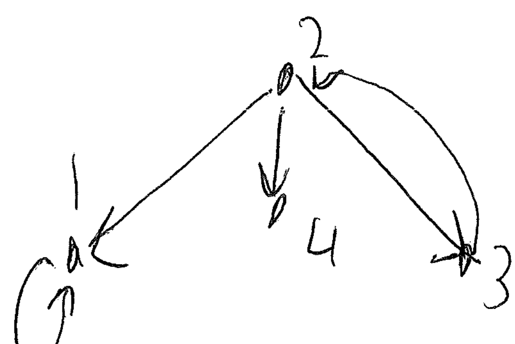
$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

For a digraph G, there are two definitions for A_{ij}.

1. $A_{ij} = \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{o.w.} \end{cases}$

2. $A_{ij} = \begin{cases} 1 & \text{if } (j,i) \in E \\ 0 & \text{o.w.} \end{cases}$

Ex.



1st method

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

2nd method

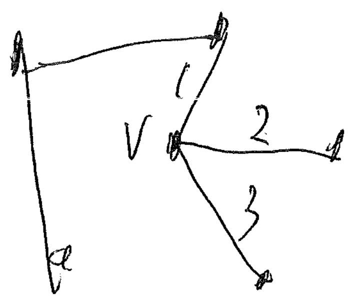
$$A = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

Properties of adjacency matrices.

Degree

The degree of a vertex v in graph G is the number of edges containing v.

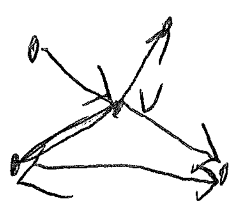
Ex.



$\deg(v) = 3$

The in (out) degree of a vertex v in a digraph G is the number of edges entering (exiting) v.

Ex.



$In(v) = 2$

$out(v) = 3$

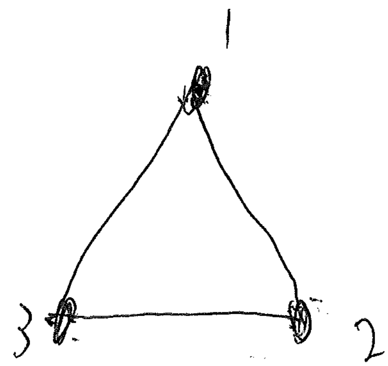
Thm The degree of vertex v is the sum of row (or column) v in the adjacency matrix.

* For digraphs, the in/out degree can be found with either the row or column sum depending on the representation.

Thm The number of distinct walks of length n connecting vertex i to vertex j is

$$(A^n)_{ij}$$

Ex



$$A = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}$$

$$A^2 = \begin{pmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{pmatrix}$$

$$A^3 = \begin{pmatrix} 2 & 3 & 3 \\ 3 & 2 & 3 \\ 3 & 3 & 2 \end{pmatrix}$$

Thm

The number of 3 cycles in undirected G

$$= \frac{\sum_{i=1}^{n} (A^3)_{ii}}{2 \cdot 3}$$

Exercise 1

Find a formula for the number of 4 cycles in undirected G (given A).

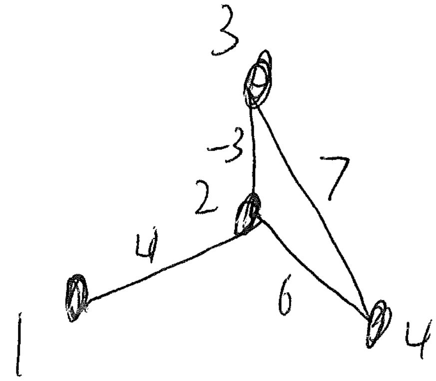
$$\frac{\sum_{i=1}^{n} (A^4)_{ii} - \left(\sum_{i=1}^{n} (A^2)_{ii}\right)^2}{2 \cdot 4}$$

Challenge: Find a general formula for cycles of length N.

Distance matrix

The distance $Dist_{ij} = \begin{cases} weight_{ij} & \text{if } \{i,j\} \in E \\ 0 & \text{o.w.} \end{cases}$

Ex.

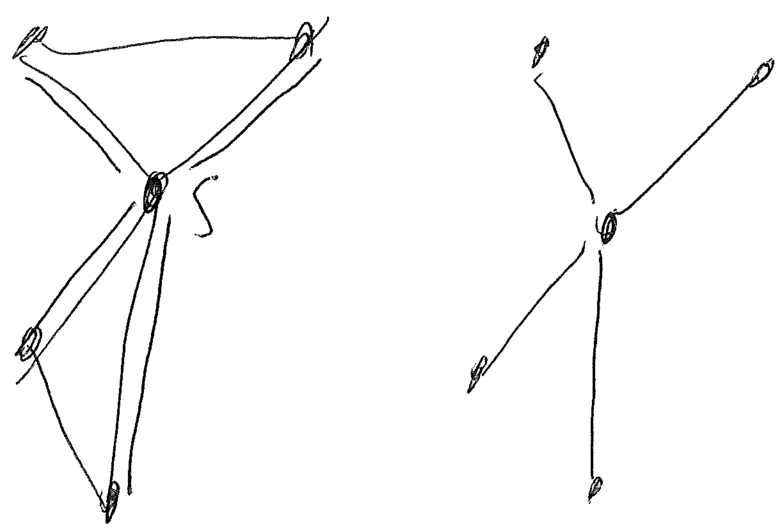


$$Dist = \begin{pmatrix} 0 & 4 & 0 & 0 \\ 4 & 0 & -3 & 6 \\ 0 & -3 & 0 & 7 \\ 0 & 6 & 7 & 0 \end{pmatrix}$$

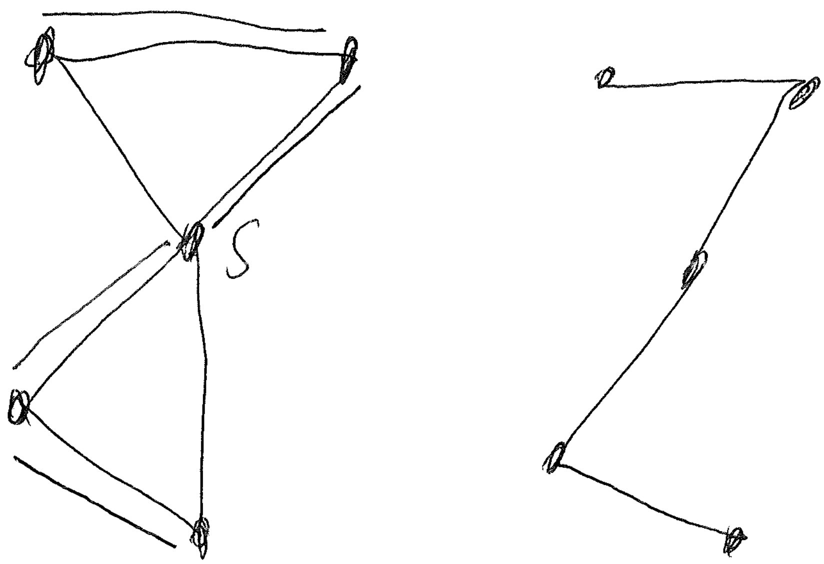
BFS and DFS

Breadth First Search (BFS) and Depth First Search (DFS) are two algorithms for finding a <u>spanning tree</u> of a Graph.

BFS (explore ~~all~~ paths from the starting node first)



DFS (Explore along a single path as far as possible first)



BFS pseudocode

List of Nodes (initialize with 1 entry per node)

List of pred~~ecessors~~ (initialize with n entries each as a very large #)

List Next (initialize this as empty)

pick a starting node S.

For each node adjacent to S. Add the nodes to next.

Take Pred(S) = 0

For each node v adjacent to S, set pred(v) = S while Next nonempty

~~take(Next)~~ take the first entry in Next (call it u)

remove this element from next

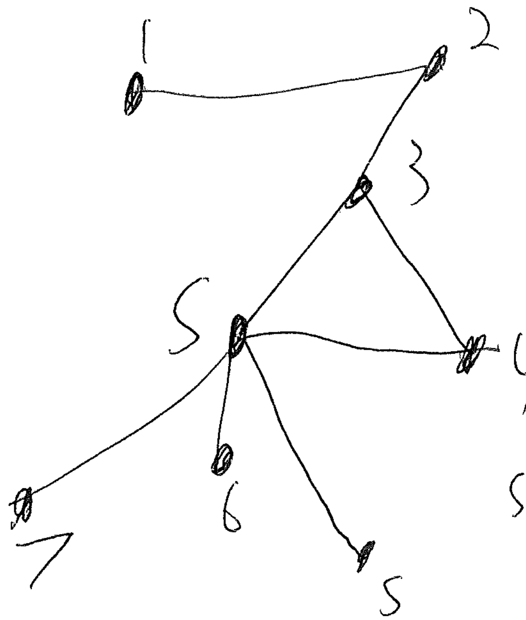
for each node V adjacent to ~~this element~~ u

set ~~Next~~ pred(v) = u if pred(v) is still a large #

if pred(v) = u

add v to Next

Ex.



Nodes = [S, 1, 2, 3, 4, 5, 6, 7]

pred = [10, 10, ..., 10]

Next = [ ]

step 0.

Next = [3, 4, 5, 6, 7]

pred = [0, 10, 10, 10, 10, 10, 10, 10]

pred = [0, 10, 10, S, S, S, S, S]

Next = [4, 5, 6, 7]

pred(2) = 3

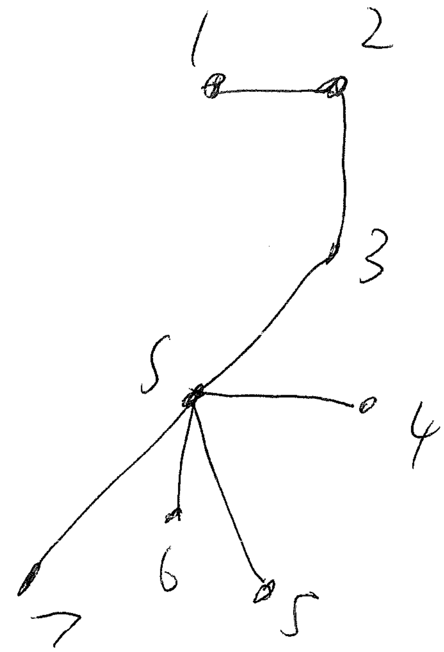
pred = [0, 10, 3, 5, 5, 5, 5, 5]

Next = [4, 5, 6, 7, 2]

4 has pred(4) = 5

5 has pred(5) = 0

Next = [5, 6, 7, 2]



Next = [6, 7, 2]

Next = [7, 2]

Next = [2]

pred(1) = 2

pred = [0, 2, 3, 5, 5, 5, 5, 5]

Next = [1]

Next = [ ] ✓