If last fired laser at time i

And you fire at $j > i$

Max matter destroyed is d_{j-i}

Actually destroyed:

$\min(X_j, d_{j-i})$

— Prob #2

Ex: $opt(j) = \max(opt(j-1), a_j + opt(j - s(j)))$

CS3000: Algorithms & Data
Paul Hand

Lecture 12:

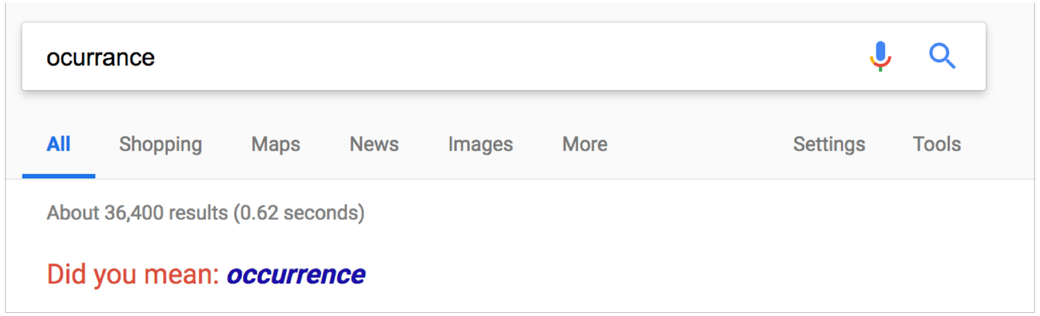
- Dynamic Programming – Sequence Alignment
- Introduction to Graphs

Feb 25, 2019

Sequence Alignments and Edit Distance

Distance Between Strings

- Autocorrect works by finding similar strings



If similarity is # characters that are different; these 2 words are not similar

- **ocurrance** and **occurrence** seem similar, but only if we define similarity carefully

ocurrance
occurrence

7 changes

oc urrance
occurrence

2 changes

insertion
deletion
swapping

Edit Distance / Alignments

Σ is set of letters "alphabet"

x has n characters, each from Σ

- Given two strings $x \in \Sigma^n, y \in \Sigma^m$, the **edit distance** is the number of insertions, deletions, and swaps required to turn x into y.

minimum

- Given an alignment, the cost is the number of positions where the two strings don't agree

o	c		u	r	r	a	n	c	e
o	c	c	u	r	r	e	n	c	e

insertion (with respect to first string)

swap

Ask the Audience

Can use
- insertion
- deletion
- swap one char. for another

- What is the minimum cost alignment of the strings **smitten** and **sitting**

s	m	i	t	t	e	n	
s		i	t	t	i	n	g

deletion

swap

insertion

edit distance of 3

Activity

- Find two strings where two different alignments (insertions, deletions, replacements) realize the edit distance between them.

a	b				a	b
x	a			x	a	

swap

insertion

deletion

Edit Distance / Alignments

- **Input:** Two strings $x \in \Sigma^n, y \in \Sigma^m$
- **Output:** The minimum cost alignment of x and y
  - **Edit Distance** = cost of the minimum cost alignment

output
is not
necessarily
unique

Dynamic Programming

- Consider the **optimal** alignment of x, y
- Three choices for the final column
  - **Case I:** only use x ($x_n, -$) deletion
  - **Case II:** only use y ($-, y_m$) insertion
  - **Case III:** use one symbol from each (x_n, y_m) swap

Case I

optimal align of $x_1 \cdots x_{n-1}$ w/ $y_1 \cdots y_m$ | x_n

Case II

optimal align. of $x_1 \cdots x_n$ w/ $y_1 \cdots y_{m-1}$ | y_m

Case III

opt. align. $x_1 \cdots x_{n-1}$ $y_1 \cdots y_{m-1}$ | x_n y_m

Dynamic Programming

Pay attn to cost

- Consider the **optimal** alignment of x, y
- **Case I:** only use x ($x_n, -$)
  - deletion + optimal alignment of $x_{1:n-1}, y_{1:m}$

deletion cost 1

- **Case II:** only use y ($-, y_m$)
  - insertion + optimal alignment of $x_{1:n}, y_{1:m-1}$

insertion cost 1

- **Case III:** use one symbol from each (x_n, y_m)
  - If $x_n = y_m$: optimal alignment of $x_{1:n-1}, y_{1:m-1}$
  - If $x_n \neq y_m$: mismatch + opt. alignment of $x_{1:n-1}, y_{1:m-1}$

no swap: cost 0

swap: cost 1

Dynamic Programming

two variables

A lot of work was done to just write this

- $\mathbf{OPT}(\boldsymbol{i}, \boldsymbol{j})$ = cost of opt. alignment of $x_{1:i}$ and $y_{1:j}$
- **Case I:** only use x ($x_i, -$)
- **Case II:** only use y ($-, y_j$)
- **Case III:** use one symbol from each (x_i, y_j)

only need to solve all subproblems w/ beginning string position 1

Dynamic Programming

- $\mathbf{OPT}(\boldsymbol{i}, \boldsymbol{j})$ = cost of opt. alignment of $x_{1:i}$ and $y_{1:j}$
- **Case I:** only use x ($x_i, -$)
- **Case II:** only use y ($-, y_j$)
- **Case III:** use one symbol from each (x_i, y_j)

Recurrence:

$$\mathrm{OPT}(i,j) = \begin{cases} \min\{1 + \mathrm{OPT}(i-1,j), 1 + \mathrm{OPT}(i,j-1), \mathrm{OPT}(i-1,j-1)\} & x_i = y_j \\ \min\{\underbrace{1 + \mathrm{OPT}(i-1,j)}_{\text{deletion}}, \underbrace{1 + \mathrm{OPT}(i,j-1)}_{\text{insertion}}, \underbrace{1 + \mathrm{OPT}(i-1,j-1)}_{\text{swap}}\} & x_i \neq y_j \end{cases}$$

which corresponds to insertion, deletion, swap?

Base Cases:

$$\mathrm{OPT}(i,0) = \underbrace{i}_{\text{deletions}}, \quad \mathrm{OPT}(0,j) = \underbrace{j}_{\text{insertions}}$$

9

Start w/ base cases.

Example

x = pert

y = beast

	- ($j=0$)	b ($j=1$)	e ($j=2$)	a	s	t
- ($i=0$)	0	1	2	3	4	5
p ($i=1$)	1	1	$\boxed{2}$			
e	2	$\boxed{2}$	$\boxed{1}$			
r	3					
t	4					

Fill in from top left to bottom right

Values in table are opt(i, j).

represents edit distance of "pe" & "be"

$$\mathrm{OPT}(i,j) = \begin{cases} \min\{1 + \mathrm{OPT}(i-1,j), 1 + \mathrm{OPT}(i,j-1), \quad \mathrm{OPT}(i-1,j-1)\} & x_i = y_j \\ \min\{1 + \mathrm{OPT}(i-1,j), 1 + \mathrm{OPT}(i,j-1), 1 + \mathrm{OPT}(i-1,j-1)\} & x_i \neq y_j \end{cases}$$

Finding the Alignment

- $\mathbf{OPT}(\boldsymbol{i}, \boldsymbol{j})$ = cost of opt. alignment of $x_{1:i}$ and $y_{1:j}$
- **Case I:** only use x ($x_i, -$)
- **Case II:** only use y ($-, y_j$)
- **Case III:** use one symbol from each (x_i, y_j)

Edit Distance ("Bottom-Up")

```
// All inputs are global vars
FindOPT(n,m):
  M[0,j] ← j, M[i,0] ← i

  for (i= 1,…,n):                compute cells in this order
    for (j = 1,…,m):
      if (x_i = y_j):
        M[i,j] = min{1+M[i-1,j],1+M[i,j-1],M[i-1,j-1]
      elseif (x_i != y_j):
        M[i,j] = 1+min{M[i-1,j],M[i,j-1],M[i-1,j-1]}

  return M[n,m]
```

Activity

- Suppose **inserting/deleting costs $\delta > 0$** and **swapping $a \leftrightarrow b$ costs $c_{a,b} > 0$**
- Write a recurrence for the min-cost alignment

$$\mathrm{OPT}(i,j) = \begin{cases} \min\{\overset{\delta}{\cancel{1}} + \mathrm{OPT}(i-1,j), \overset{\delta}{\cancel{1}} + \mathrm{OPT}(i,j-1), \quad \mathrm{OPT}(i-1,j-1)\} & x_i = y_j \\ \min\{\underset{\delta}{\cancel{1}} + OPT(i-1,j), \underset{\delta}{\cancel{1}} + \mathrm{OPT}(i,j-1), \underset{c_{x[i],y[j]}}{\cancel{1}} + \mathrm{OPT}(i-1,j-1)\} & x_i \neq y_j \end{cases}$$

was Edit distance

now cost to transform $X_{1:i}$ w/ $y_{1:j}$

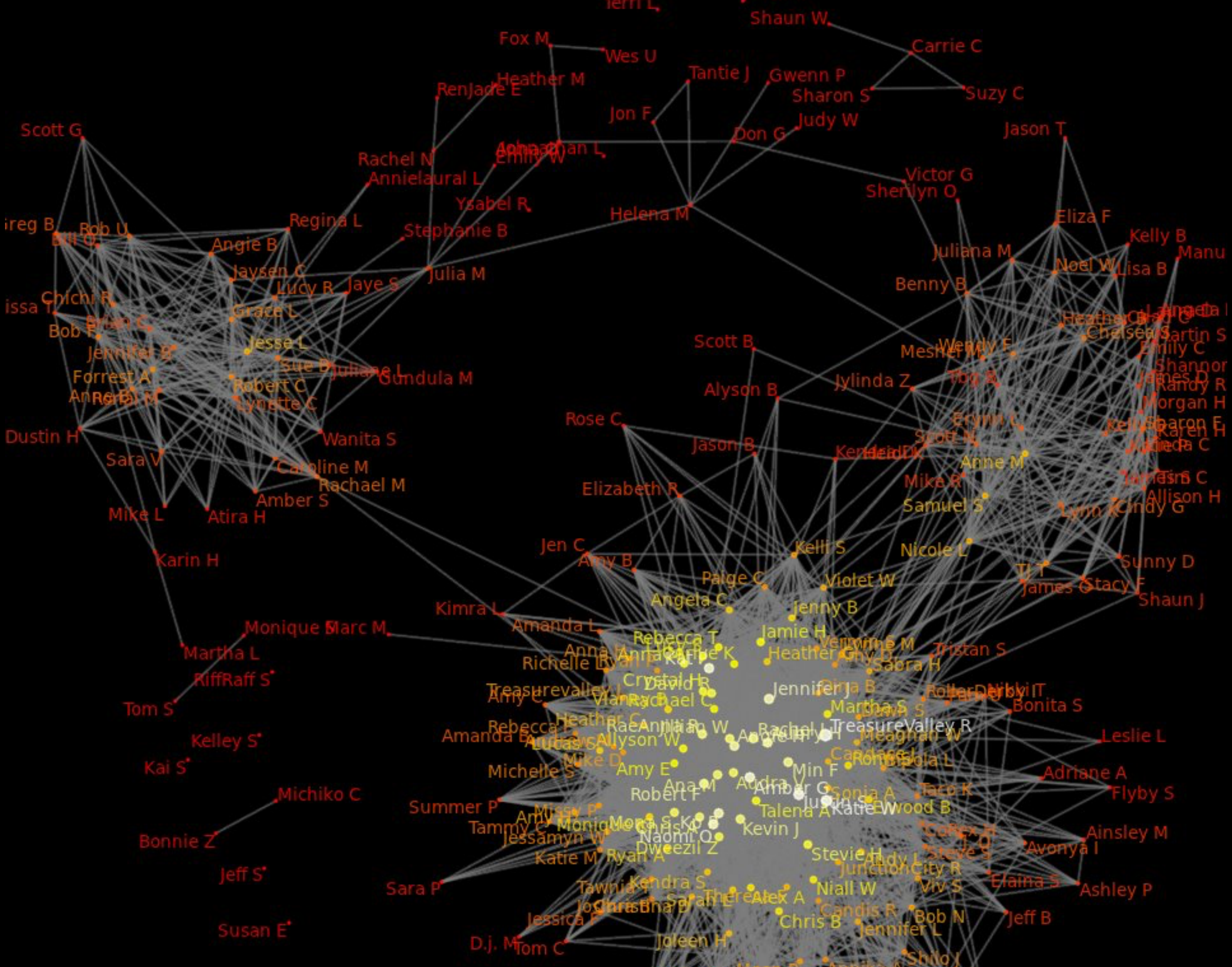
Discussion

- Dynamic Programming is a time-space tradeoff. Comment on the tradeoff in the case of edit distance.

	Naive Approach	This Dynamic Programming Approach
time	exponential	nm
Space	constant	nm

$$\mathrm{OPT}(i,j) = \begin{cases} \min\{1+\mathrm{OPT}(i-1,j), 1+\mathrm{OPT}(i,j-1), \quad \mathrm{OPT}(i-1,j-1)\} & x_i = y_j \\ \min\{1+OPT(i-1,j), 1+\mathrm{OPT}(i,j-1), 1+\mathrm{OPT}(i-1,j-1)\} & x_i \neq y_j \end{cases}$$

Graphs



Shaun W
Fox M
Wes U
Carrie C
Heather M
Tantie J
Gwenn P
RenJade E
Sharon S
Suzy C
Scott G
Jon F
Judy W
Jason T
Don G
Rachel N
Victor G
Annielaural L
Sherilyn O
Ysabel R
Helena M
Eliza F
Regina L
Stephanie B
Kelly B
Manu
Angie B
Juliana M
Noel W
Lisa B
Jaysen C
Julia M
Lucy R
Jaye S
Benny B
Chichi R
Grace L
Bob F
Jesse L
Scott B
Martin S
Jennifer B
Emily C
Sue D
Gundula M
Forrest A
Robert C
Jylinda Z
Alyson B
Randy R
Lynette C
Morgan H
Rose C
Dustin H
Wanita S
Jason B
Anne M
Sara V
Caroline M
Rachael M
Amber S
Elizabeth R
Mike R
Allison H
Samuel S
Cindy G
Mike L
Atira H
Karin H
Kelli S
Nicole L
Sunny D
Jen C
Amy B
Paige C
Violet W
Shaun J
Angela C
Jenny B
Kimra L
Amanda L
Monique S
Marc M
Jamie H
Martha L
Tristan S
Richelle L
RiffRaff S
Jennifer J
Bonita S
Tom S
TreasureValley R
Kelley S
Leslie L
Meaghan W
Alyson W
Kai S
Amy E
Min F
Adriane A
Michelle S
Michiko C
Robert F
Amber G
Flyby S
Summer P
Talena A
Katie W
Elwood B
Kevin J
Ainsley M
Naomi Q
Bonnie Z
Dweezil Z
Avonya I
Katie M
Ryan A
Stevie H
Jeff S
Sara P
Kendra S
Elaina S
Ashley P
Niall W
Alex A
Candis R
Jessica F
Bob N
Susan E
Chris B
Jeff B
Jennifer L
D.j. M
Tom C
Joleen H

Graphs Are Everywhere

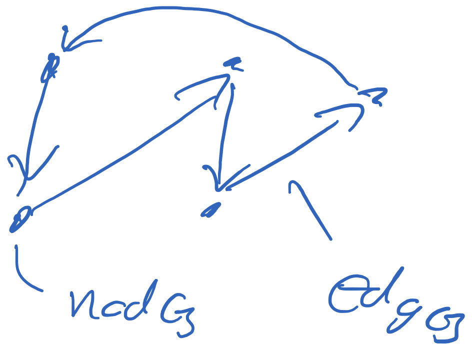
- Transportation networks
- Communication networks
- WWW
- Biological networks
- Citation networks
- Social networks
- …

What’s Next

- **Graph Algorithms:**
  - **Graphs:** Key Definitions, Properties, Representations
  - **Exploring Graphs:** Breadth/Depth First Search
    - Applications: Connectivity, Bipartiteness, Topological Sorting
  - **Shortest Paths:**
    - Dijkstra
    - Bellman-Ford (Dynamic Programming)
  - **Minimum Spanning Trees:**
    - Borůvka, Prim, Kruskal
  - **Network Flow:**
    - Algorithms
    - Reductions to Network Flow

Graphs: Key Definitions

Edges are like arrows



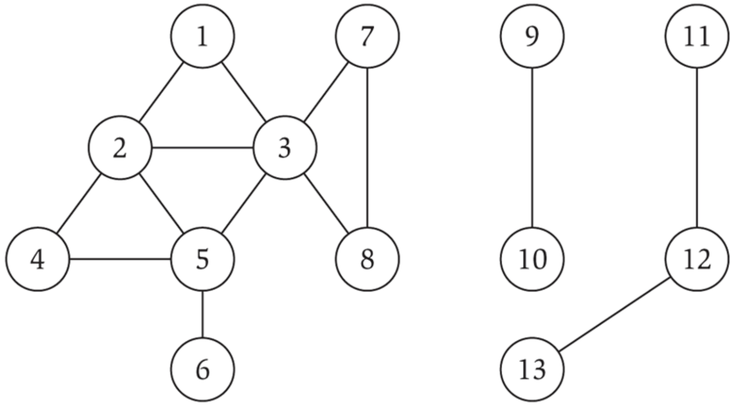
- **Definition:** A directed graph $G = (V, E)$
  - V is the set of nodes/vertices
  - $E \subseteq V \times V$ is the set of edges
  - An edge is an ordered $e = (u, v)$ "from u to v"

set of pairs of vertices

If $(u,v) \in E$, it is an edge of the graph

- **Definition:** An undirected graph $G = (V, E)$
  - Edges are unordered $e = (u, v)$ "between u and v"

- **Simple Graph:**
  - No duplicate edges
  - No self-loops $e = (u, u)$

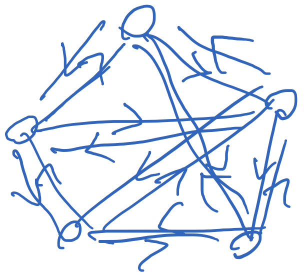


all one graph

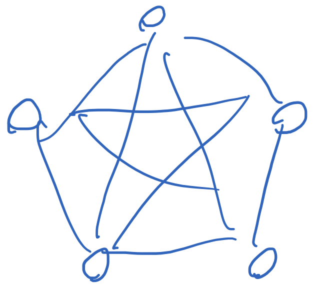
Activity

- How many edges can there be in a **simple** directed/undirected graph?

Directed



Undirected



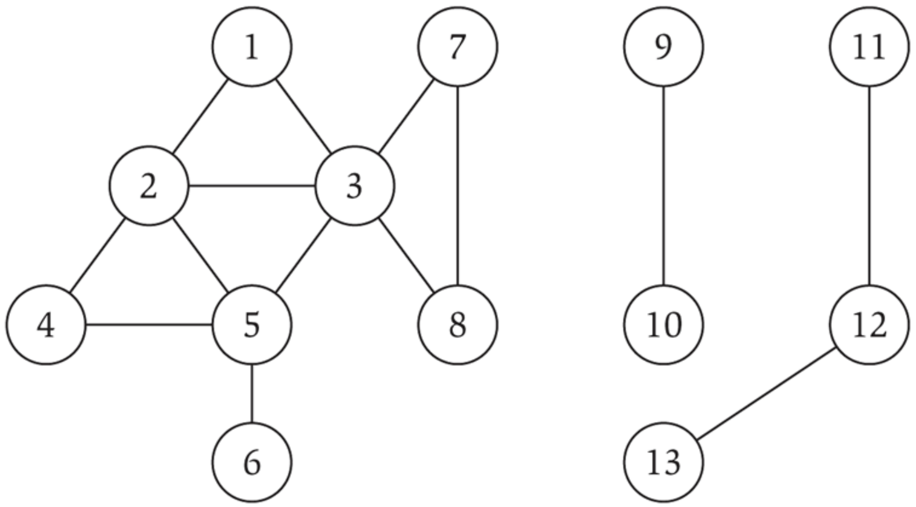
Paths/Connectivity

- A path is a sequence of consecutive edges in E
  - $P = \{(u, w_1), (w_1, w_2), (w_2, w_3), \dots, (w_{k-1}, v)\}$
  - $P = u - w_1 - w_2 - w_3 - \cdots - w_{k-1} - v$
  - The length of the path is the # of edges

- An undirected graph is connected if for every two vertices $u, v \in V$, there is a path from u to v
- A directed graph is strongly connected if for every two vertices $u, v \in V$, there are paths from u to v and from v to u

Cycles

- A cycle is a path $v_1 - v_2 - \cdots - v_k - v_1$ where $k \geq 3$ and $v_1, \ldots, v_k$ are distinct



Activity: how many cycles are there in this graph?

Activity

- Suppose an undirected graph G is connected
  - True/False? G has at least $n - 1$ edges

Activity

- Suppose an undirected graph G has $n - 1$ edges
    - True/False? G is connected