CSE 2320

Week of 07/06/2020

Instructor : Donna French

Rubber Duck Debugging

https://en.wikipedia.org/wiki/Rubber_duck_debugging

In software engineering, rubber duck debugging is a method of debugging code.

The name is a reference to a story in the book "The Pragmatic Programmer" in which a programmer would carry around a rubber duck and debug their code by forcing themselves to explain it, line-by-line, to the duck.

Many other terms exist for this technique, often involving different (usually) inanimate objects (teddy bear) or pets such as a dog or a cat.

Rubber Duck Debugging

Many programmers have had the experience of explaining a problem to someone else, possibly even to someone who knows nothing about programming, and then hitting upon the solution in the process of explaining the problem.

In describing what the code is supposed to do and observing what it actually does, any incongruity between these two becomes apparent.

More generally, teaching a subject forces its evaluation from different perspectives and can provide a deeper understanding.

By using an inanimate object, the programmer can try to accomplish this without having to interrupt anyone else.

Quick Sort

{9,7,6,5}	5 is pivot
	9 $\not<$ 5 => no swap
	7 $\not<$ 5 => no swap
	6 $\not<$ 5 => no swap
	final => swap 9 and 5
{5,7,6,9}	5 was pivot so divide into {} and {7,6,9}
{}{5}{7,6,9}	nothing to the left of 5
{7,6,9}	9 is pivot

```c
int partition (int A[], int low, int high)
{
   int i, j = 0;
   int pivot = A[high];

   i = (low - 1);

   for (j = low; j < high; j++)
   {
      if (A[j] < pivot)
      {
         i++;
         swap(&A[i], &A[j]);
      }
   }
   swap(&A[i + 1], &A[high]);

   return (i + 1);
}
```

0	1	2	3

{5, 7, 6, 9}

{5, 7, 6, 9}

partition(A, 1, 3)

i	j	pivot	low	high

Quick Sort

{9,7,6,5}	5 is pivot
	9 ≮ 5 => no swap
	7 ≮ 5 => no swap
	6 ≮ 5 => no swap
	final => swap 9 and 5
{5,7,6,9}	5 was pivot so divide into {} and {7,6,9}
{}{5}{7,6,9}	nothing to the left of 5
{7,6,9}	9 is pivot
	7 < 9 => swap 7 and 7
{7,6,9}	6 < 9 => swap 6 and 6
{7,6,9}	final => swap 9 and 9
{7,6,9}	9 was pivot so divide into {7,6} and {9} and {}
{7,6}{9}{}	nothing to the right of 9
{7,6}	6 is pivot
{7,6}	7 ≮ 6 => no swap
{5,6,7,9}	final => swap 7 and 6

Quick Sort

`{5,6,7,9}`

`9 is pivot`

```
int partition (int A[], int low, int high)
{
   int i, j = 0;
   int pivot = A[high];

   i = (low - 1);

   for (j = low; j < high; j++)
   {
      if (A[j] < pivot)
      {
         i++;
         swap(&A[i], &A[j]);
      }
   }
   swap(&A[i + 1], &A[high]);

   return (i + 1);
}
```

0	1	2	3
{5,	6,	7,	9}
{5,	6,	7,	9}

`partition(A, 0, 3)`

i	j	pivot	low	high

Quick Sort

{5,6,7,9}	9 is pivot
	5 < 9 => swap 5 and 5
	6 < 9 => swap 6 and 6
	7 < 9 => swap 7 and 7
	final => swap 9 and 9
{5,6,7,9}	9 was pivot so divide into {5,6,7} and {9} and {}
{5,6,7}{9}{}	nothing to the right of 9
{5,6,7}	7 is pivot
	5 < 7 => swap 5 and 5
	6 < 7 => swap 6 and 6
	final => swap 7 and 7
{5,6,7}	7 was pivot so divide into {5,6} and {7} and {}
{5,6}{7}{}	nothing to the right of 7
{5,6}	6 is pivot
	5 < 6 => swap 5 and 5
{5,6,7,9}	final => swap 6 and 6

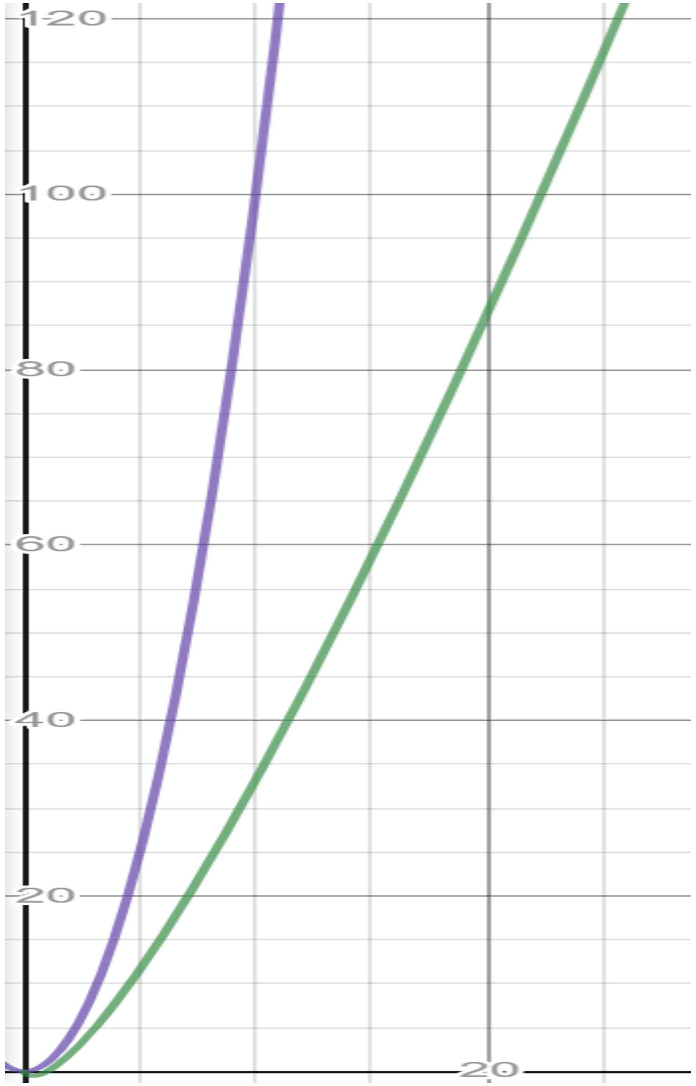
Quick Sort Analysis

Quick Sort's worst case run time is

$\Theta(n^2)$

Quick Sort's best case run time is

$\Theta(n\log_2 n)$



Quick Sort Analysis

So why think about Quick Sort when Merge Sort is at least as good?

Because the constant factor hidden in the big-Θ notation for Quick Sort is quite good.

In practice, Quick Sort outperforms Merge Sort and it significantly outperforms Selection Sort and Insertion Sort.

Quick Sort Analysis

How is it that Quick Sort's worst-case and best-case running times differ?

Let's start by looking at the worst-case running time.

Suppose that we're really unlucky and the partition sizes are really unbalanced.

In particular, suppose that the pivot chosen by the partition function is always either the smallest or the largest element in the n element subarray.

Quick Sort Analysis

In particular, suppose that the pivot chosen by the partition function is always either the smallest or the largest element in the *n* element subarray.

Let's start with the case there the pivot is the largest element

{1,2,4,6,7,8,14,18,19}

Pivot would be 19 so all elements are less than pivot so there would be multiple swaps of numbers with themselves but 19 would still remain on the far right.

{1,2,4,6,7,8,14,18} and {}.
{1,2,4,6,7,8,14} and {}.
{1,2,4,6,7,8} and {}

{1,2,4,6,7} and {}
{1,2,4,6} and {}
{1,2,4} and {}
{1,2} and {}

Quick Sort Analysis

As we can see, one of the partitions will contain no elements and the other partition will contain $n - 1$ (all but the pivot) every time.

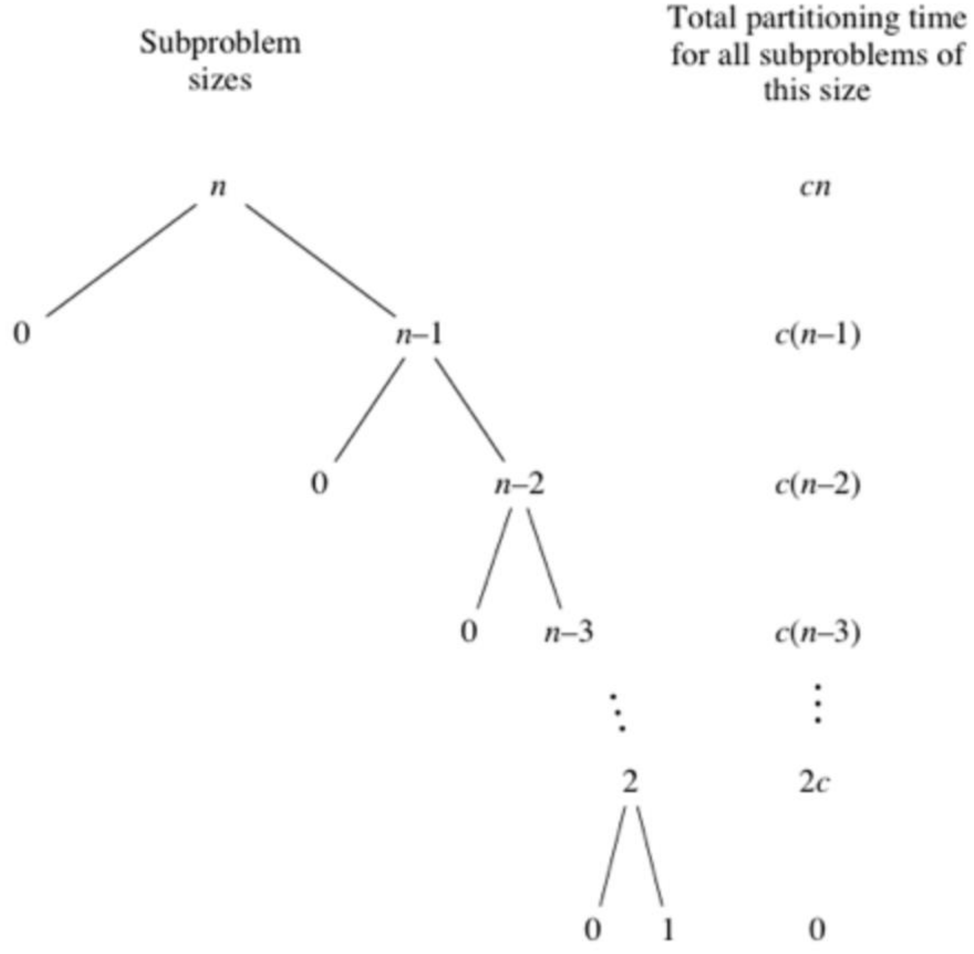
So the recursive calls will be on subarrays of sizes 0 and $n - 1$.

In this situation, divide and conquer with recursion does not help the run time so we don't get the benefit of it either - our runtime suffers.

An array in reverse sorted order would have the same issue.

Quick Sort Analysis

When Quick Sort always has the most unbalanced partitions possible, then the original call takes *cn* time for some constant *c*, the recursive call on $n-1$ elements takes $c(n-1)$, the recursive call on $n-2$ elements takes $c(n-2)$ time and so on … until we get to the final 2 elements.



Quick Sort Analysis

If n is 4

c(4 + (4-1) + (4-2) + (4-3)) =
c(4 + 3 + 2 + 1) = 10

$c(\frac{n(n+1)}{2}) = \frac{4(4+1)}{2} = 10$

We can add up the runtime of those partition steps

$cn + c(n-1) + c(n-2) + \cdots + 2c = c(n + (n-1) + (n-2) + \cdots + 2) =$

This pattern indiciates an arithmetic series that we sum with $\frac{n(n+1)}{2}$

Since our pattern ends with 2 instead of 1, we subtract 1

$c(\frac{n(n+1)}{2} + 2 - 1) = c(\frac{1}{2}n^2 + \frac{1}{2}n + 1) = \Theta(n^2)$

Worst case run time

Quick Sort Analysis

Suppose we implement QuickSort so that ChoosePivot always selects the first element of the array. What is the running time of this algorithm on an input array that is already sorted?

- Not enough information to answer this question
- $\Theta(n)$
- $\Theta(n \log n)$
- $\Theta(n^2)$

Quick Sort Analysis

What does the best case look like?

Quick Sort's best case occurs when the partitions are as evenly balanced as possible

their sizes either are equal or are within 1 of each other.

Quick Sort Analysis

their sizes are either equal or are within 1 of each other.

if the subarray has an odd number of elements and the pivot is right in the middle after partitioning and each partition has $\frac{n-1}{2}$ elements

if the subarray has an even number of elements and one partition has $\frac{n}{2}$ elements with the having $\frac{n}{2}-1$

Quick Sort Analysis

In either of these cases, each partition has at most $\frac{n}{2}$ elements

The tree of subproblem sizes looks a lot like the tree of subproblem sizes for Merge Sort…

this is where $\log_2 n$ was introduce to the runtime

The partitioning times look like the merging times…

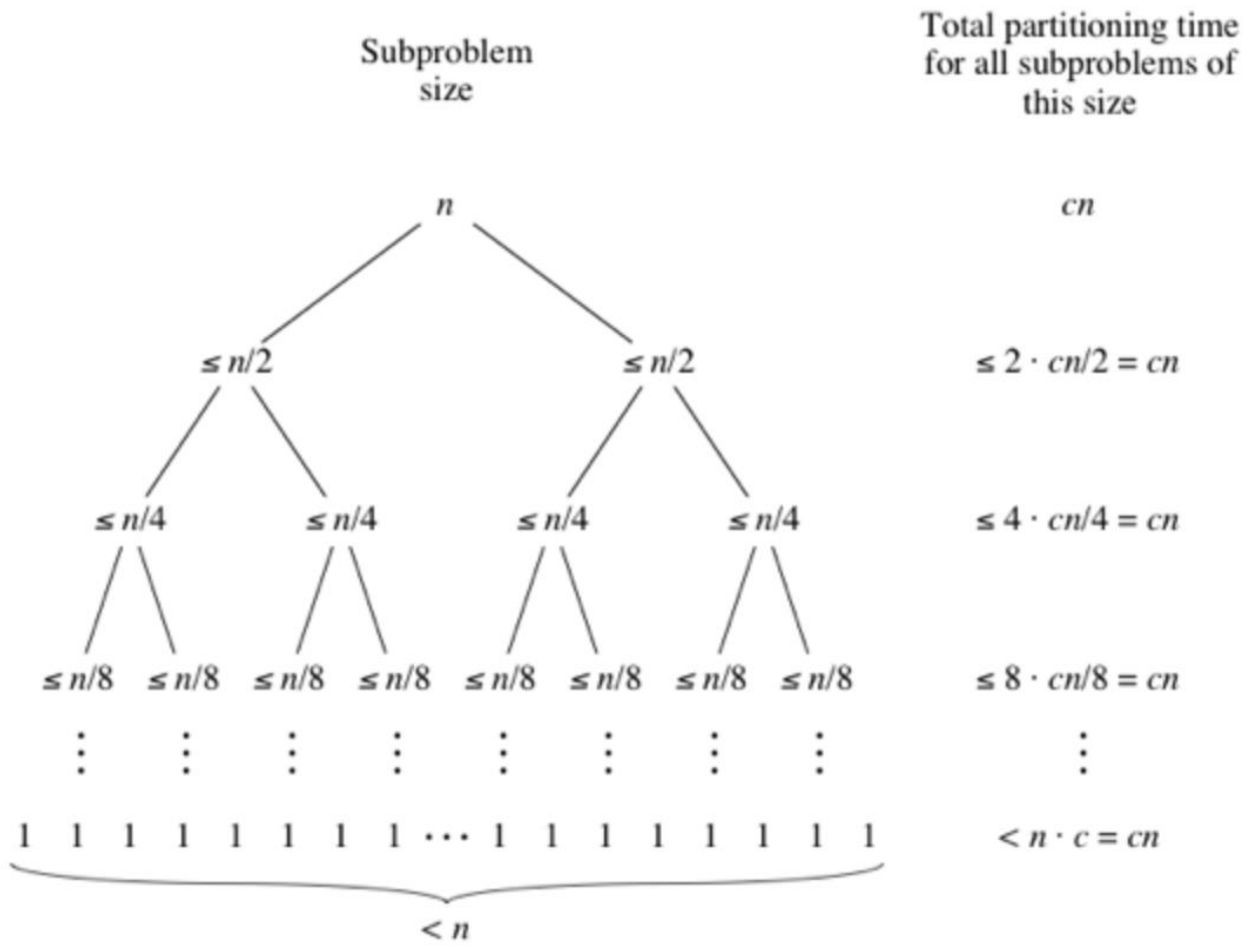
this is where n was introduced to the runtime

Quick Sort Analysis

This pattern tell us the same thing it told us about MergeSort.

Partitioning time (n) * number of partitions to make ($\log_2 n$)

$\Theta(n\log_2 n)$



Suppose we run QuickSort on some input, and, magically, every recursive call chooses the median element of its subarray as its pivot. What's the running time in this case?

- Not enough information to answer question
- $\Theta(n)$
- $\Theta(n \log n)$
- $\Theta(n^2)$

Fix two elements of the input array. How many times can these two elements be compared during the execution of QuickSort ?

- 0
- 0 or 1
- 0 or 1 or 2
- Any number in between 0 and n-1

{9, 7, 15, 16, 12}	12 is pivot
{9, 7, 15, 16, 12}	9 < 12 swap 9 with 9
{9, 7, 15, 16, 12}	7 < 12 swap 7 and 7
	15 $\not<$ 12 so no swap
	16 $\not<$ 12 so no swap
	final swap of 15 and 12
	recursive call (left and right)
{9, 7} 12, {16, 15}	7 is pivot in left/15 is pivot in right
	9 $\not<$ 7 so no swap
	final swap of 9 and 7
{7,9} 12 {16,15}	16 $\not<$ 15 so no swap
	final swap of 16 and 15
{7,9} 12 {15,16}	

Quick Sort Analysis

Worst case is $\Theta(n^2)$ and best case is $\Theta(n\log_2 n)$

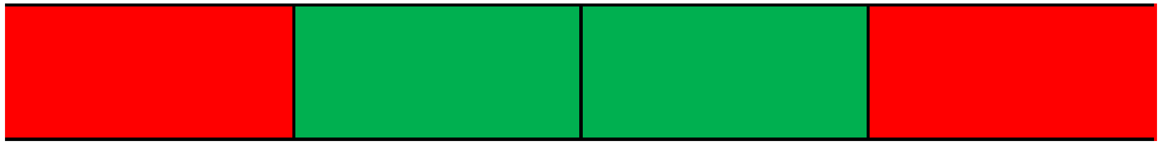
What is an average case?

In the average case, all elements are equally likely to be chosen as the pivot.

Quick Sort Analysis

Worst case is $\Theta(n^2)$ and best case is $\Theta(n\log_2 n)$

After partitioning, we would expect half the time the pivot to end up in the middle two quarters and half the time for it to end up in the outer two quarters.



It can be proven mathematically that this will result in a runtime of $\Theta(n\log_2 n)$ for the average case.

OLQ5

- The Quick Sort code from lecture will be provided. Be sure to click on the arrow to get the PDF. You can try this out with LockDown Browser and the Practice Quiz.

Click here ⬇ for PDF.

- You will be provided with a copy of this table for reference.

i	j	pivot	low	high

OLQ5

You will be given an array to work with (for example)

{9, 7, 5, 12, 11}

Assume that a function to print the entire array is included the line after each call to function `swap()` inside `parition()`.

This call will be in the code provided with the quiz itself and will be in the OLQ5 review.

OLQ5

```
int partition (int A[], int low, int high)
{
   int i, j = 0;
   int pivot = A[high];

   i = (low - 1);

   for (j = low; j < high; j++)
   {
      if (A[j] < pivot)
      {
         i++;
         swap(&A[i], &A[j]);
         printArray(A);
      }
   }
   swap(&A[i + 1], &A[high]);
   printArray(A);

   return (i + 1);
}
```

OLQ5

For your given array, you will need to write the output of every call to this function - what does the array look like after each call to function `swap()`?

For example, if your given array is

`{9,7,5,12,11}`

then your answer will be

{9,7,5,12,11}
{9,7,5,12,11}
{9,7,5,12,11}
{9,7,5,11,12}
{5,7,9,11,12}
{5,7,9,11,12}
{5,7,9,11,12}

OLQ5

{9, 7, 5, 12, 11}

```
i = low -1

for (j = low -> j < high)
{
  if A[j] < pivot
     move i
     swap A[i] A[j]
}
swap A[i+1] with A[high]
```

loop swap 9 9	{9,7,5,12,11}
loop swap 7 7	{9,7,5,12,11}
loop swap 5 5	{9,7,5,12,11}
final swap 12 11	{9,7,5,11,12}
final swap 9 5	{5,7,9,11,12}
loop swap 7 7	{5,7,9,11,12}
final swap 9 9	{5,7,9,11,12}

OLQ5

{7,12,5,9,11}

loop swap 7 7	{7,12,5,9,11}
loop swap 12 5	{7,5,12,9,11}
loop swap 12 9	{7,5,9,12,11}
final swap 12 11	{7,5,9,11,12}
loop swap 7 7	{7,5,9,11,12}
loop swap 5 5	{7,5,9,11,12}
final swap 9 9	{7,5,9,11,12}
final swap 7 5	{5,7,9,11,12}

```
i = low -1

for (j = low -> j < high)
{
  if A[j] < pivot
    move i
    swap A[i] A[j]
}
swap A[i+1] with A[high]
```

OLQ5

Coding Assignment 2 will be using Quick Sort.

I HIGHLY recommend that you go ahead and start on the Coding Assignment and get the Quick Sort code work on a small hardcoded array. Add the print statements as shown here in the slides.

Practice by giving yourself an array and figuring out the prints. Then, run your program and check your answer.

Coding Assignment 2

Make a copy of Coding Assignment 1. Remove the Merge Sort and Insertion Sort code.

Add a Quick Sort function to your code.

You will run your code on the same files from Coding Assignment 1.

There will be other things to do but get this working to practice for OLQ5.

Graphs

So you might wonder…

What do graphs have to do with algorithms?

A lot of problems can be converted into graph problems.

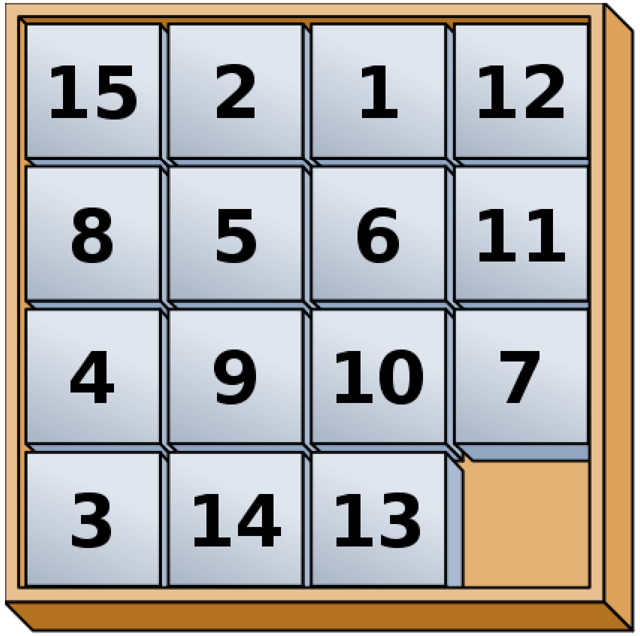
If we have algorithms for solving graph problems, then we can also solve the problems that we can convert into graph problems.

Graphs

For example, have you seen one of these?

It has many names

15-puzzle
Gem Puzzle
Boss Puzzle
Game of Fifteen
Mystic Square



This game can be converted into a graph problem and be solved with a graph traversal algorithm.

Graphs

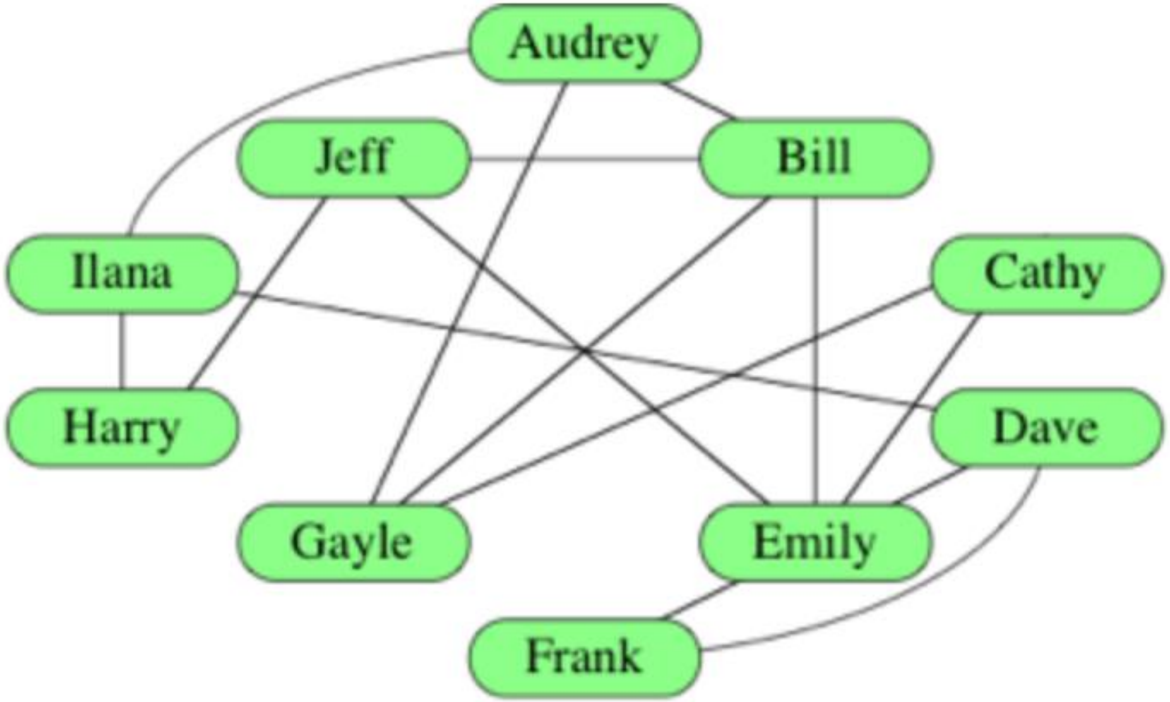
This represents a graph of a social network.

A line between 2 names indicates that people know each other.

Cathy knows Gayle and Gayle knows Bill but Cathy does not know Bill.

Each name is a **vertex** of the graph.

Each line is an **edge** – an edge connects 2 vertices (plural of vertex).



Graphs

How many vertices does this graph have?

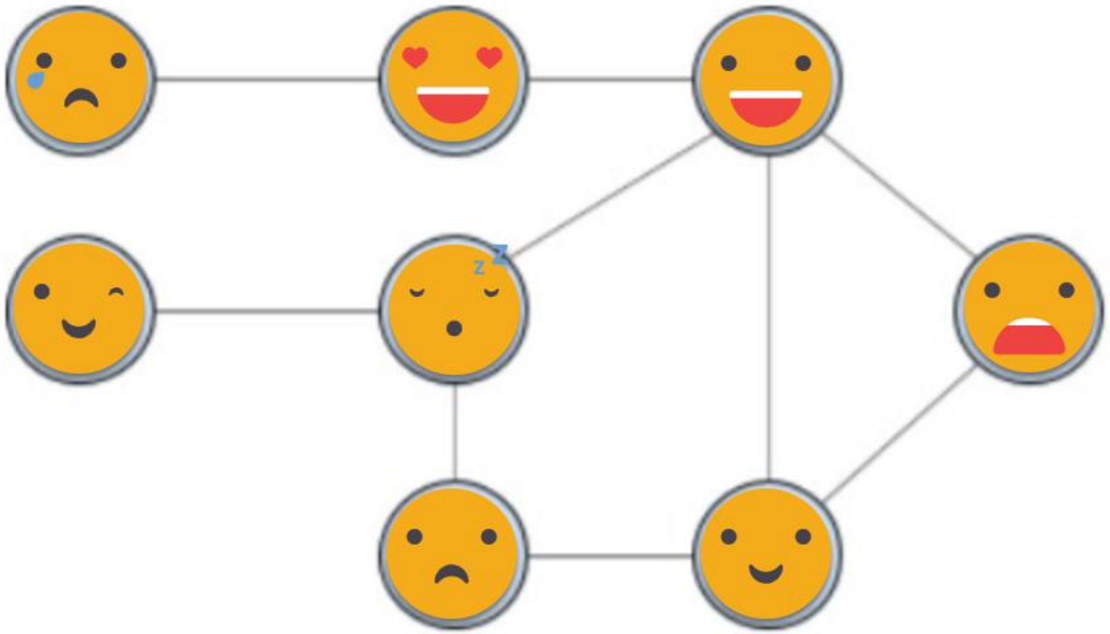
8

How many edges?

9

How many edges could be eliminated and still be able to connect all vertices without backtracking?

2 and we would be left with 7 edges.



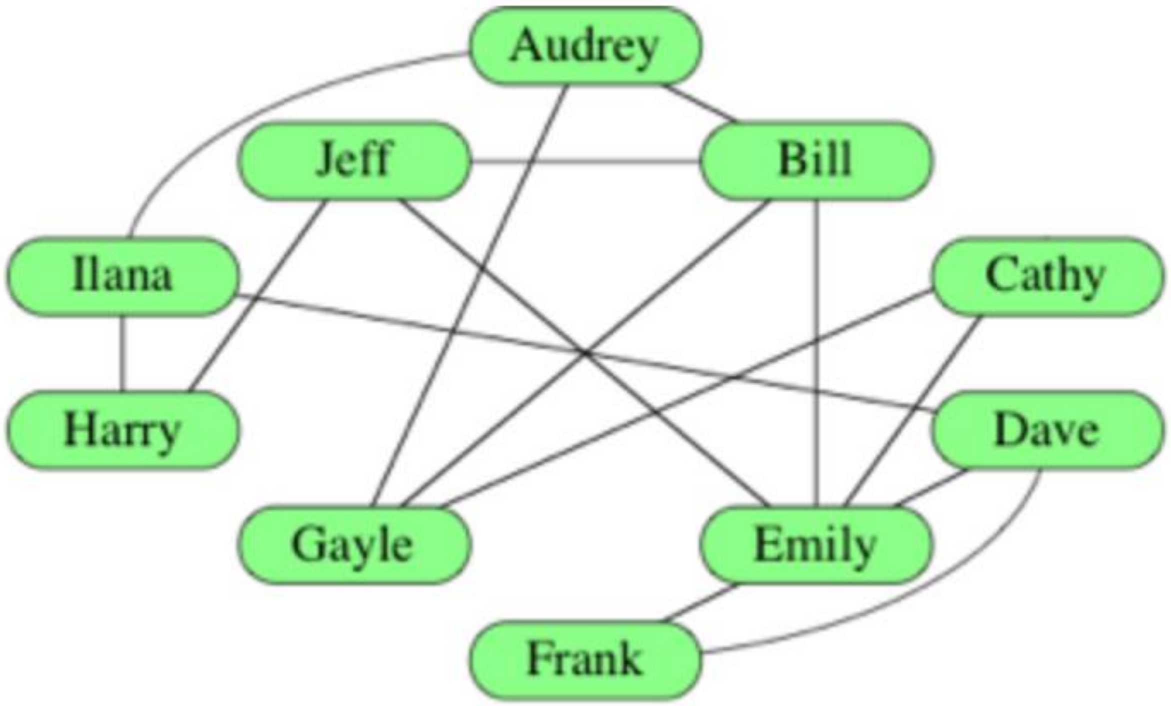
Graphs

We can depict that two vertices are connected via an edge

(Audrey, Bill) or (A, B)

What is the edge between Jeff and Emily?

(J,E)



Is there a difference between
(A,B) and (B,A) or (J,E) and (E,J)?

No. The "knows each other" relationship indicates a bidirectional relationship.

All of this graph's edges are undirected.

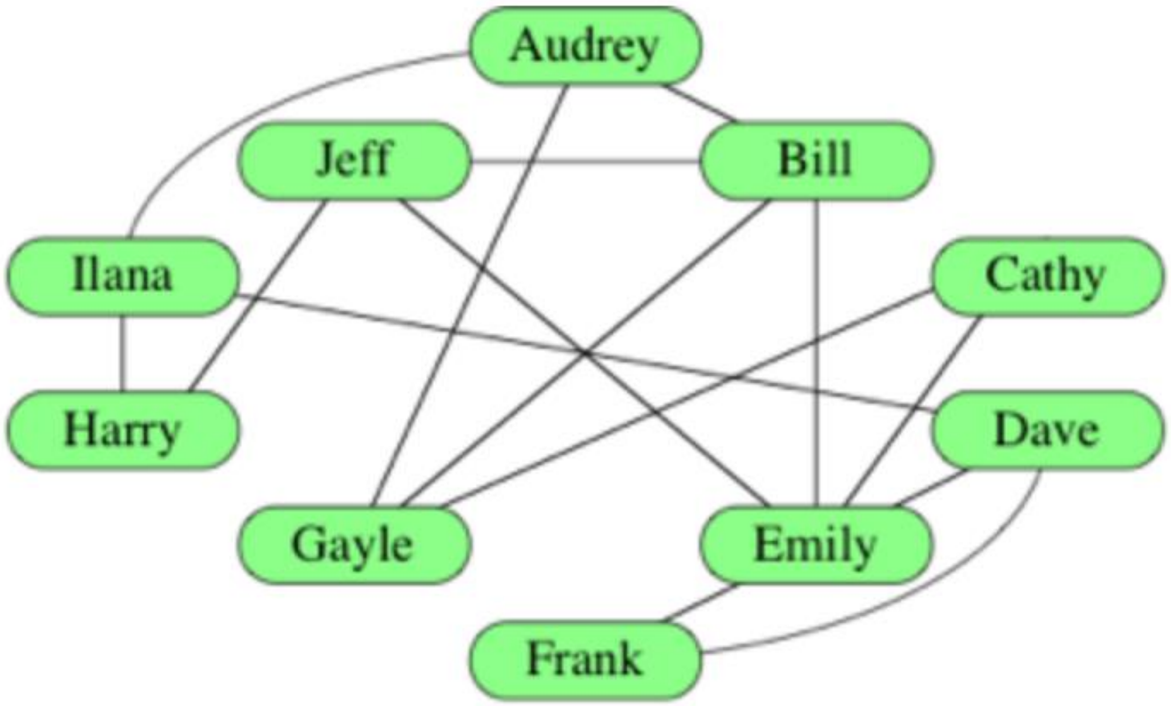
Graphs

All of this graph's edges are undirected.

An undirected edge of (r,s) is the same as edge (s,r)

In an undirected graph, the edge between two vertices is incident on the two vertices.

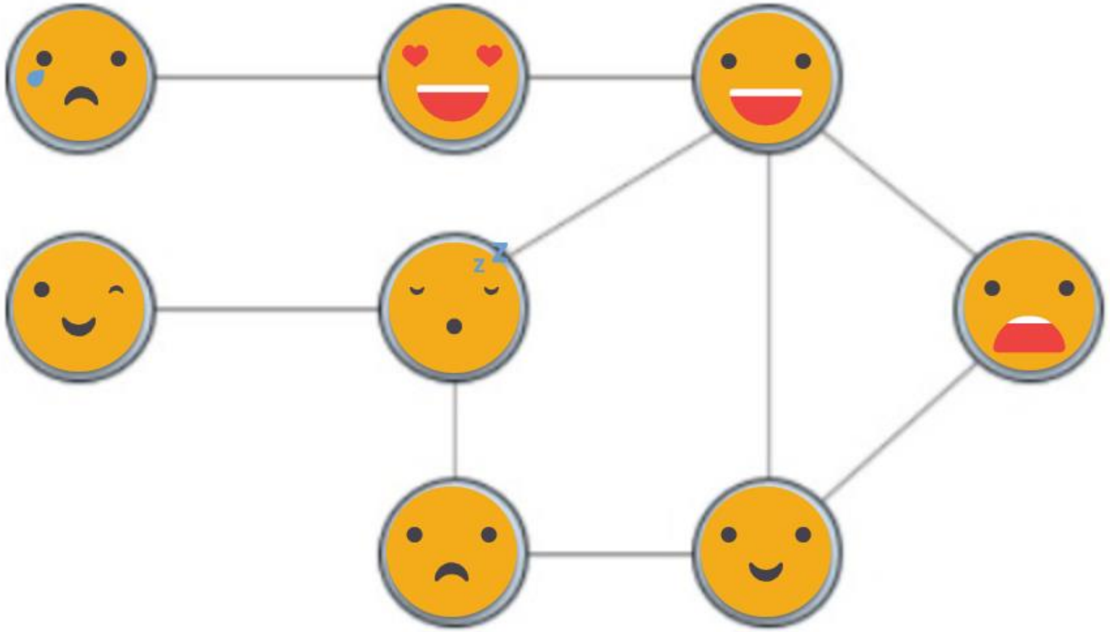
The vertices connected by an edge are adjacent or neighbors.



The number of edges incident on a vertex is the degree of the vertex.

Graphs

Two vertices are called adjacent if they are connected by an edge.

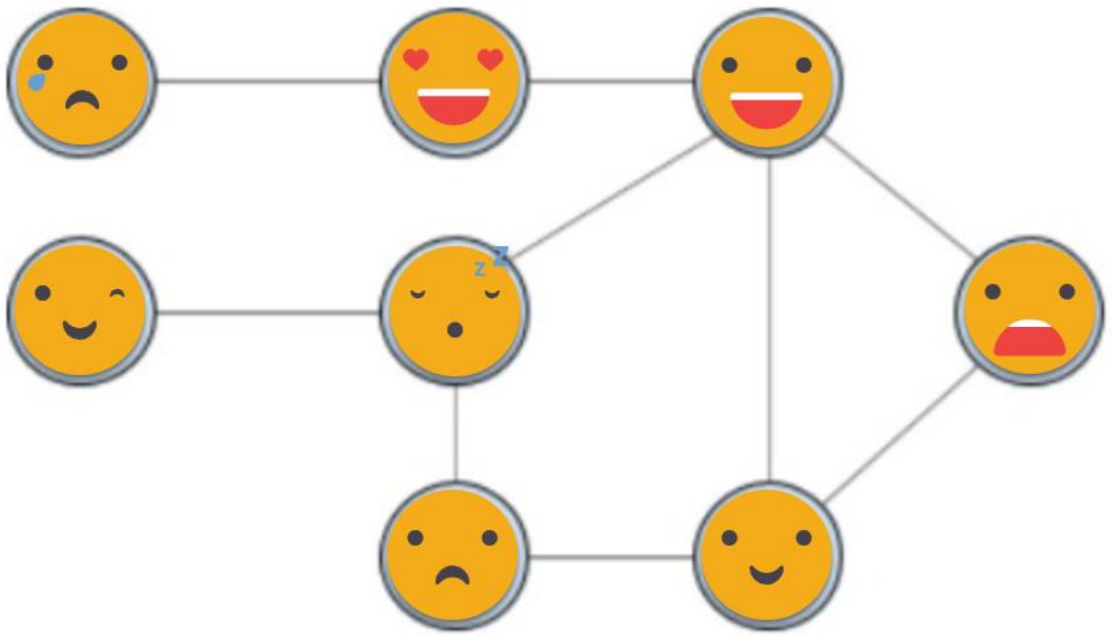


Is  adjacent to  ?

Is  adjancent to  ?

Graphs

Two edges are called incident, if they share a vertex.



Also, a vertex and an edge are called incident, if the vertex is one of the two vertices the edge connects.

Graphs

What if Gayle wants to meet Harry?

Gayle could ask Cathy because Cathy knows Emily who know Jeff who knows Harry.

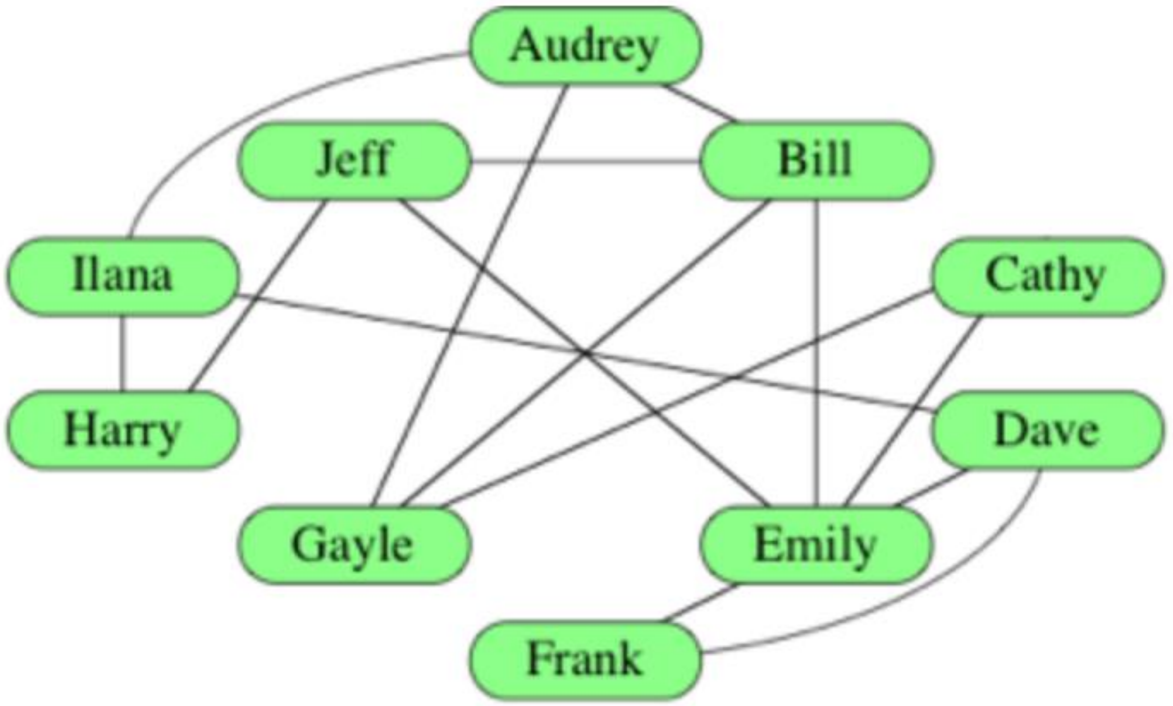
This is called the path between Gayle and Harry.

Is there a shorter route?

Yes – Gayle -> Audrey -> Ilana -> Harry

Is there another?

Yes – Gayle -> Bill ->Jeff -> Harry



Is there a path shorter than these two?

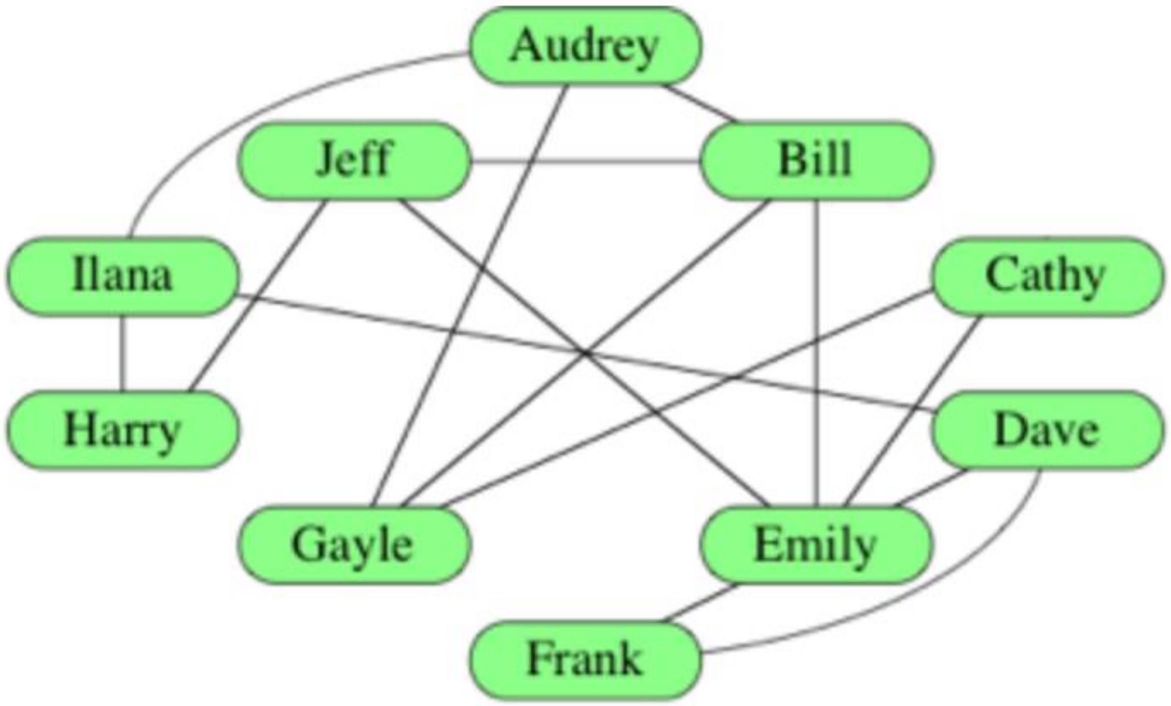
Graphs

When a path goes from a particular vertex back to itself, that path is also called a

cycle

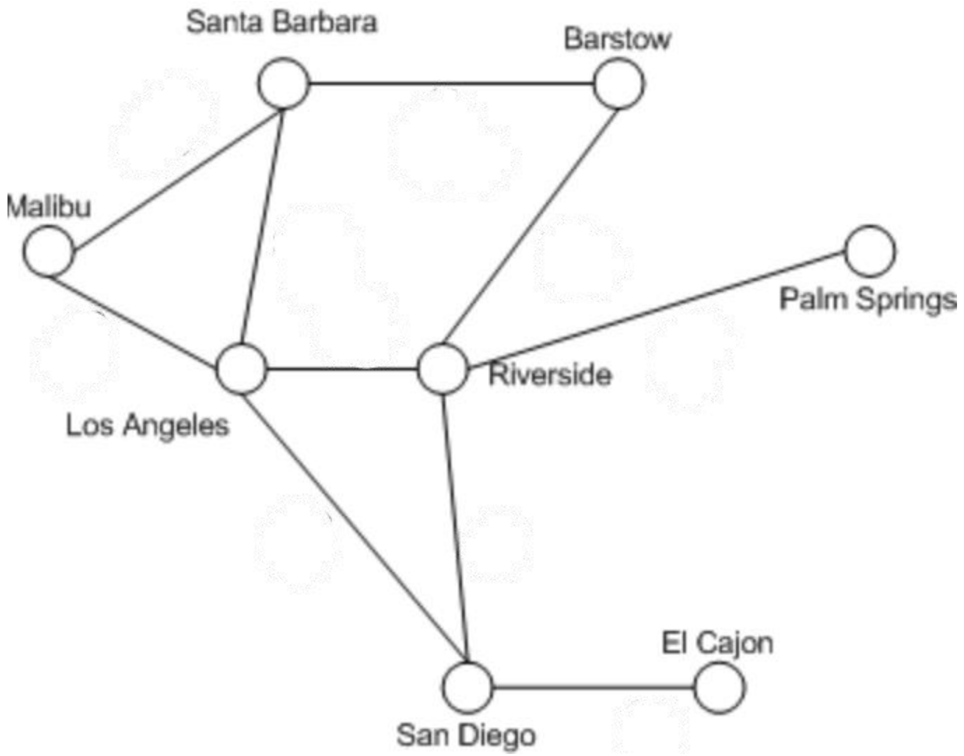
Are there any cycles in this graph?

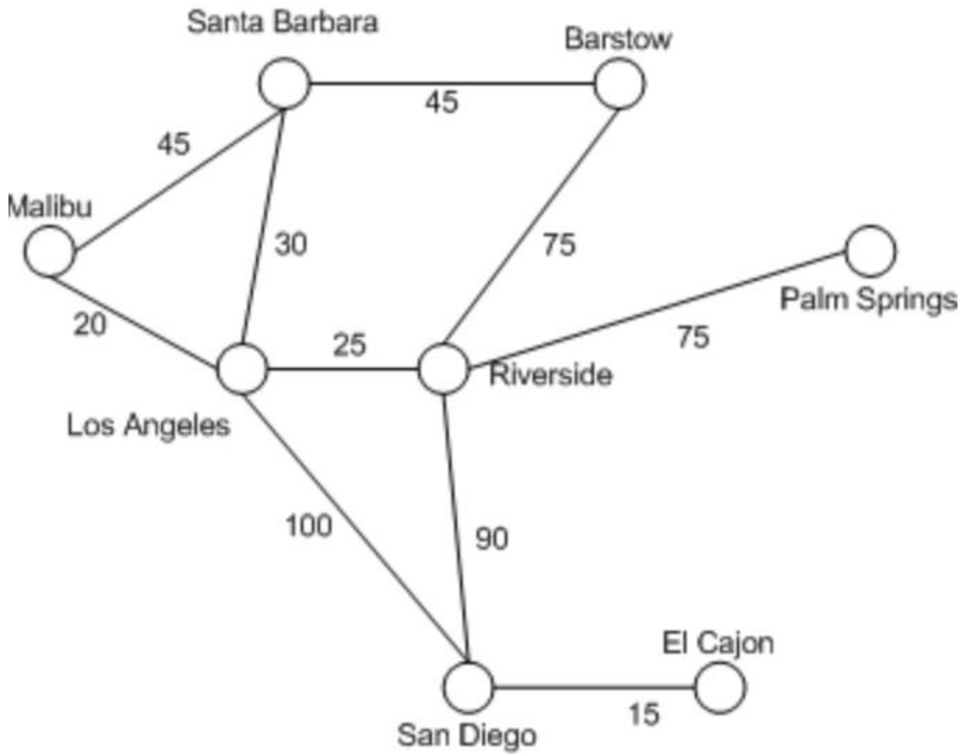
Is there a path from every vertex to every other vertex?



Graphs

We can add numeric values to edges to indicate relationships between vertices.



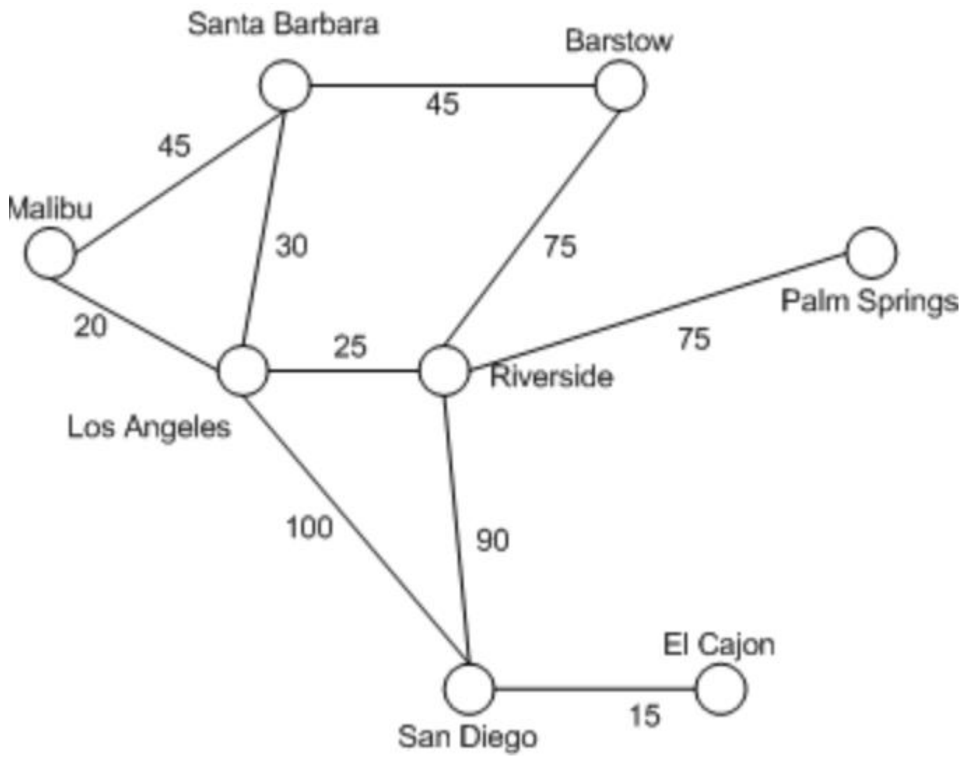


Graphs

In this graph, the number on the edge indicates the distance between each vertex.

Malibu is 45 units from Santa Barbara.

The unit is not specified here so we should not assume miles – could be minutes or kilometers or something else.



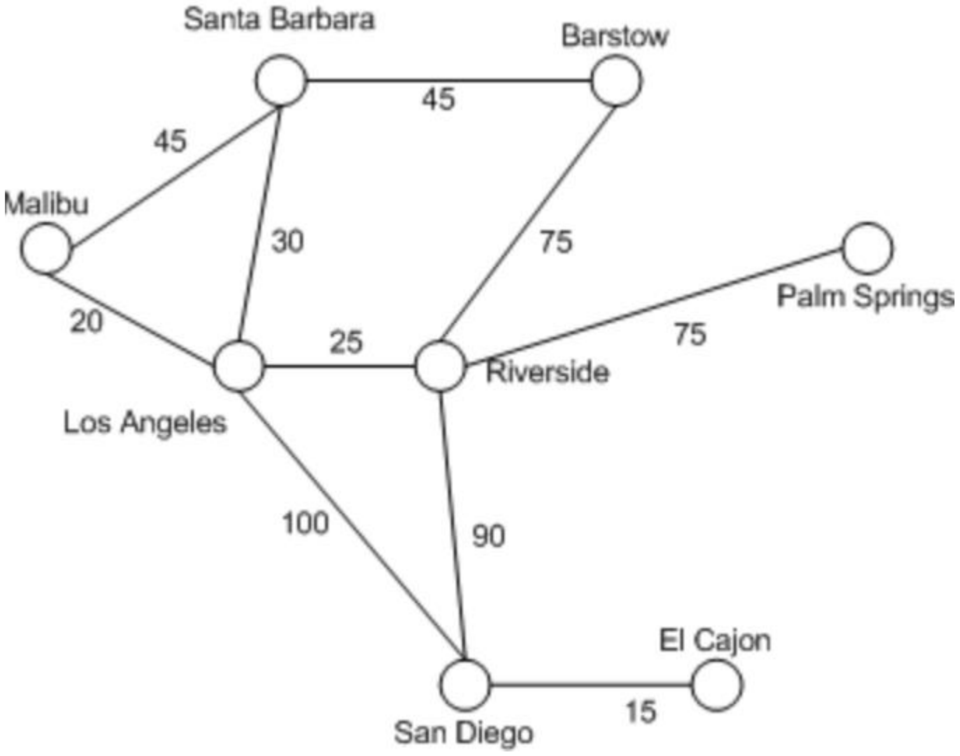
Graphs

When a number is put on an edge, that edge is said to have a **weight**.

A graph whose edges have weights is called a

weighted graph

The shortest path in a weighted graph is the path with the minimum sum of edge weights over all paths between two vertices.



Graphs

What is the shortest path between Palm Springs and Malibu?

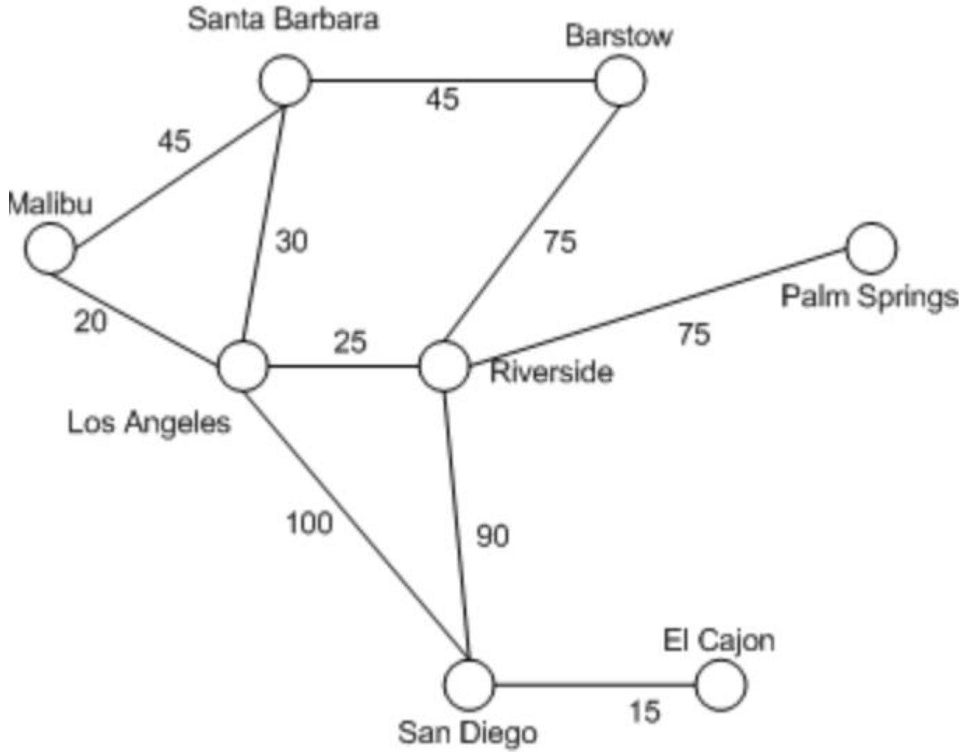
Palm Springs -> Riverside	75

Three choices from Riverside

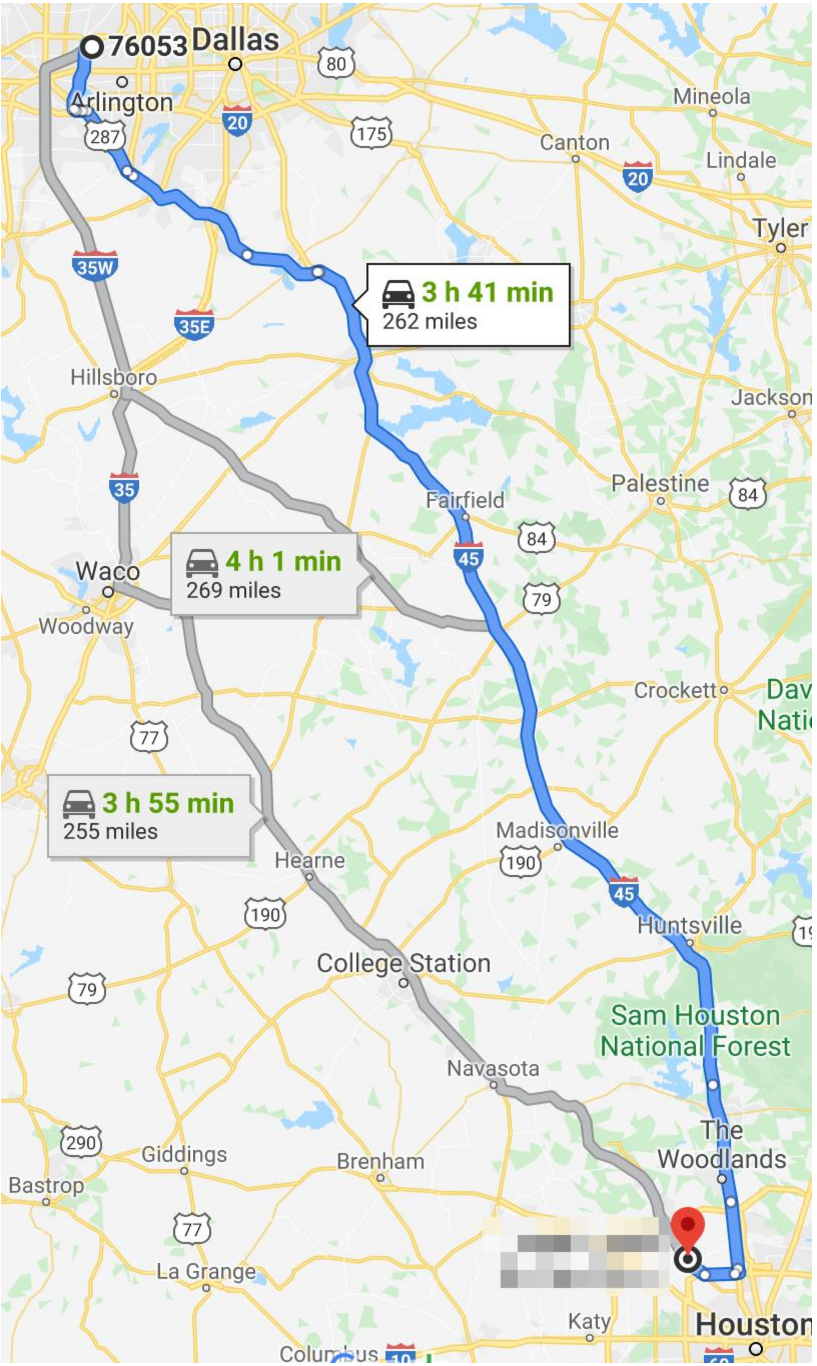
Barstow	75
Los Angeles	25
San Diego	90

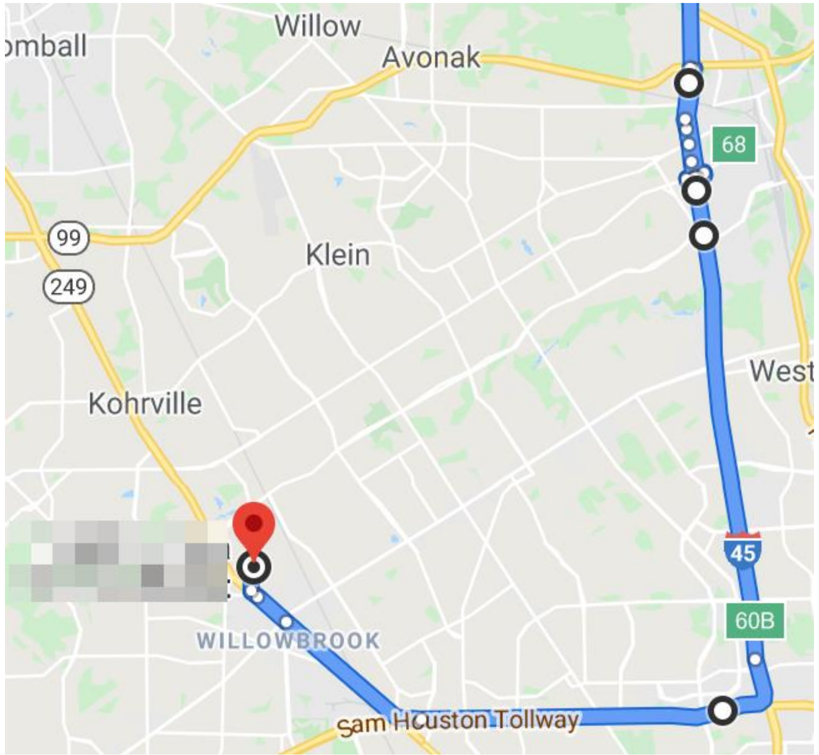
Which route is "best"?

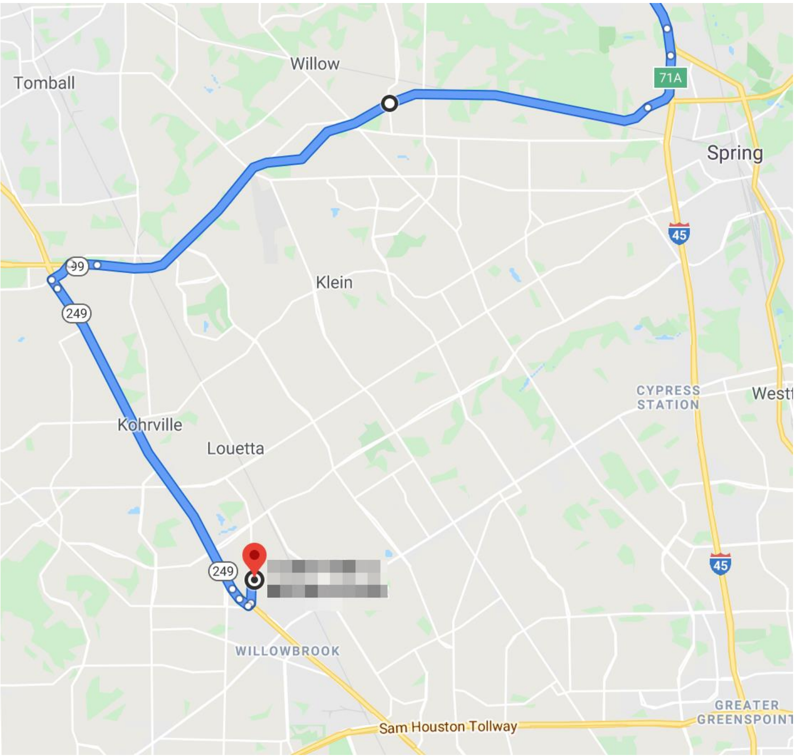
What happens when you factor in traffic, tollways, construction…?



Graphs

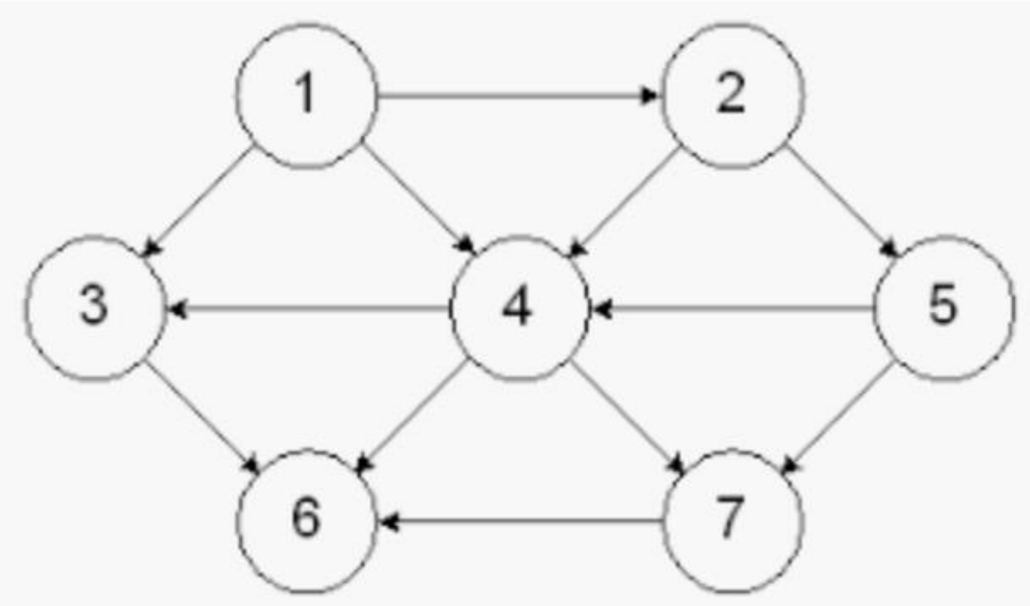

76053
Dallas
Arlington
Mineola
Canton
Lindale
Tyler
3 h 41 min
262 miles
Hillsboro
Jackson
Palestine
Fairfield
Waco
4 h 1 min
269 miles
Woodway
Crockett
3 h 55 min
255 miles
Madisonville
Hearne
Huntsville
College Station
Sam Houston
National Forest
Navasota
The
Woodlands
Giddings
Brenham
Bastrop
La Grange
Katy
Houston


Willow
Avonak
Klein
Kohrville
West
WILLOWBROOK
Sam Houston Tollway
68
60B


Tomball
Willow
Spring
Klein
Kohrville
Louetta
CYPRESS
STATION
WILLOWBROOK
Sam Houston Tollway
GREATER
GREENSPOINT

Graphs

The relationship between vertices does not always go both ways. Notice that the edges now have arrows? This is called **directed graph**.

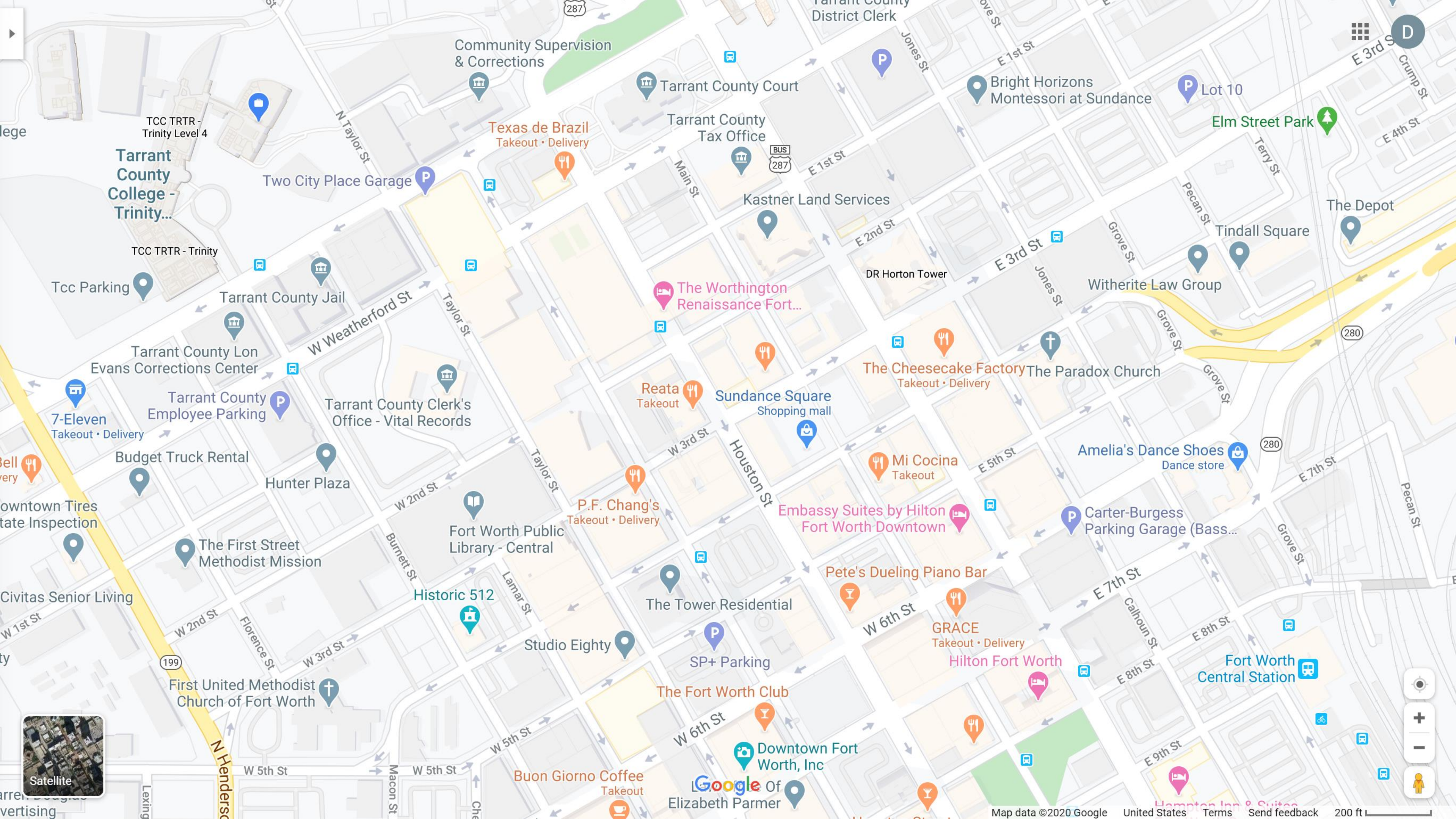


Does this directed graph have any cycles?

No.

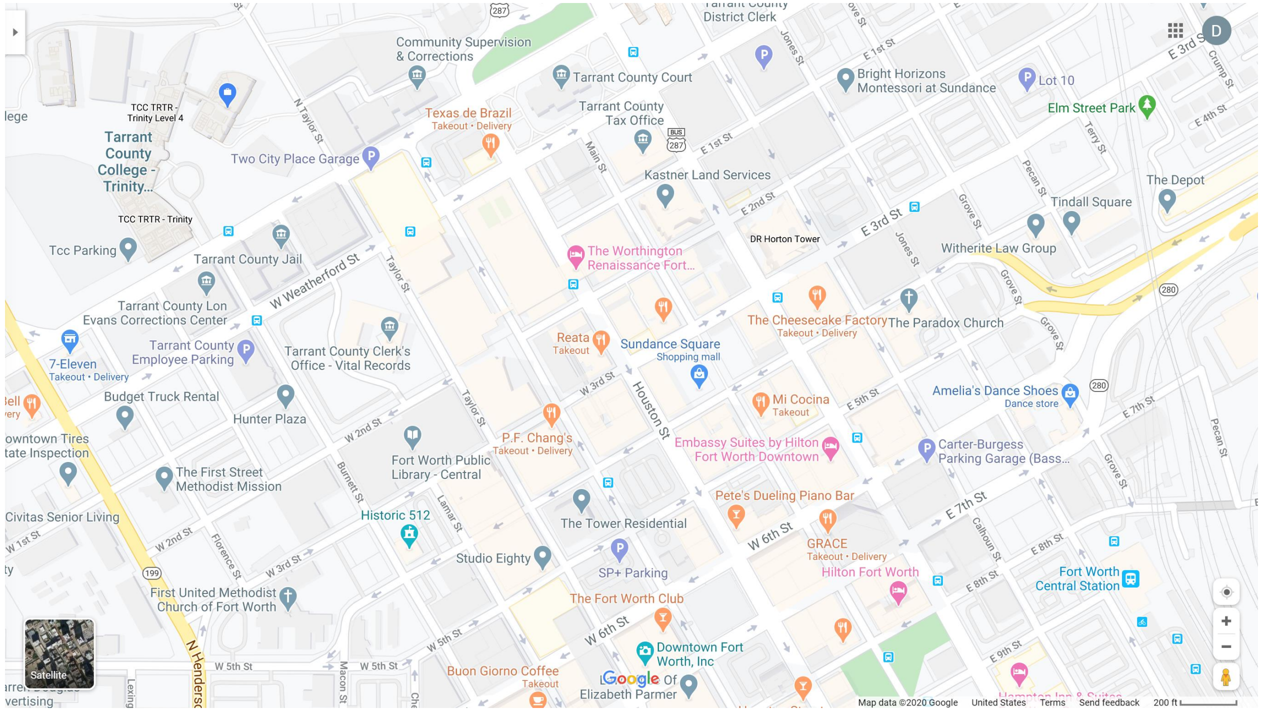
It is a directed acyclic graph (dag).

In a road map, for example, there could be one-way streets.


District Clerk
Community Supervision & Corrections
Tarrant County Court
Bright Horizons Montessori at Sundance
Lot 10
Elm Street Park
E 3rd St
E 1st St
E 4th St
Crump St
Jones St
TCC TRTR - Trinity Level 4
Tarrant County College - Trinity...
TCC TRTR - Trinity
Texas de Brazil
Takeout • Delivery
Tarrant County Tax Office
Two City Place Garage
N Taylor St
Main St
Kastner Land Services
E 2nd St
Terry St
Pecan St
The Depot
Tindall Square
Grove St
DR Horton Tower
Witherite Law Group
Tcc Parking
Tarrant County Jail
W Weatherford St
Taylor St
The Worthington Renaissance Fort...
Tarrant County Lon Evans Corrections Center
The Cheesecake Factory
Takeout • Delivery
The Paradox Church
7-Eleven
Takeout • Delivery
Tarrant County Employee Parking
Tarrant County Clerk's Office - Vital Records
Reata
Takeout
Sundance Square
Shopping mall
Budget Truck Rental
Hunter Plaza
W 3rd St
Houston St
Mi Cocina
Takeout
E 5th St
Amelia's Dance Shoes
Dance store
E 7th St
W 2nd St
Fort Worth Public Library - Central
P.F. Chang's
Takeout • Delivery
Embassy Suites by Hilton Fort Worth Downtown
Carter-Burgess Parking Garage (Bass...
The First Street Methodist Mission
Burnett St
Pete's Dueling Piano Bar
Civitas Senior Living
W 1st St
Historic 512
Lamar St
The Tower Residential
W 6th St
GRACE
Takeout • Delivery
Calhoun St
E 8th St
Studio Eighty
SP+ Parking
Hilton Fort Worth
Fort Worth Central Station
Florence St
First United Methodist Church of Fort Worth
The Fort Worth Club
W 5th St
Downtown Fort Worth, Inc
E 9th St
N Henderso
Buon Giorno Coffee
Takeout
Macon St
Elizabeth Parmer
Satellite
Google
Map data ©2020 Google
United States
Terms
Send feedback
200 ft

Graphs

If distances were added to this map, then it would be a weighted directed graph.



Graphs

Directed edges require some more detailed vocabulary for describing them.

A directed edge leaves one vertex and enters another.

When a directed edge leaves vertex A and enters vertex B, then we denote that with (A,B) and the order of the vertices matters.

The number of edges leaving a vertex is its out-degree.

The number of edges entering a vertex is its in-degree.

Graphs

How would you show the relationship between vertex 1 and vertex 3?

(1,3) or ~~(3,1)~~

How would you show the relationship between vertex 4 and vertex 7?

(4,7) or ~~(7,4)~~

What is the out-degree of vertex 4?

number of edges leaving a vertex

3

What is the in-degree of vertex 4?

number of edges entering a vertex

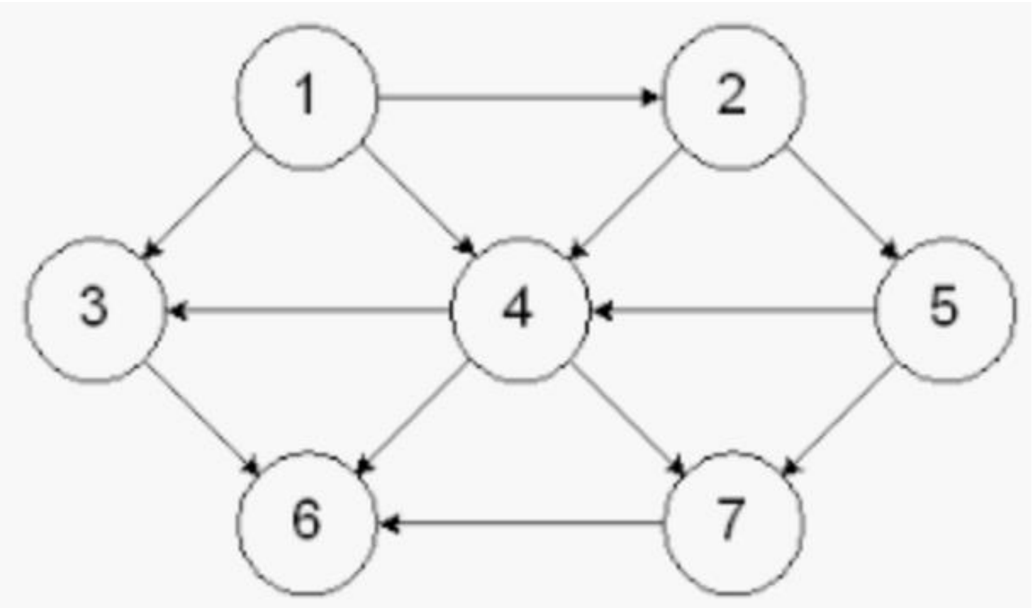
3

Out-degree of 3?

In-degree of 5?

Out-degree of 2?

In-degree of 6?



Graphs

When working with graphs, we need to refer to the set of vertices and the set of edges.

Common notation is V for a vertex set and E for an edge set.

When representing a graph or running an algorithm on a graph, we often want to use the sizes of the vertex and edge sets in asymptotic notation.

Graphs

If we used strict set notation, then it would be

$\Theta(|V|)$ or $\Theta(\log_2|E|)$

But, when using asymptotic notation, you will typically see the || (set notation) dropped.

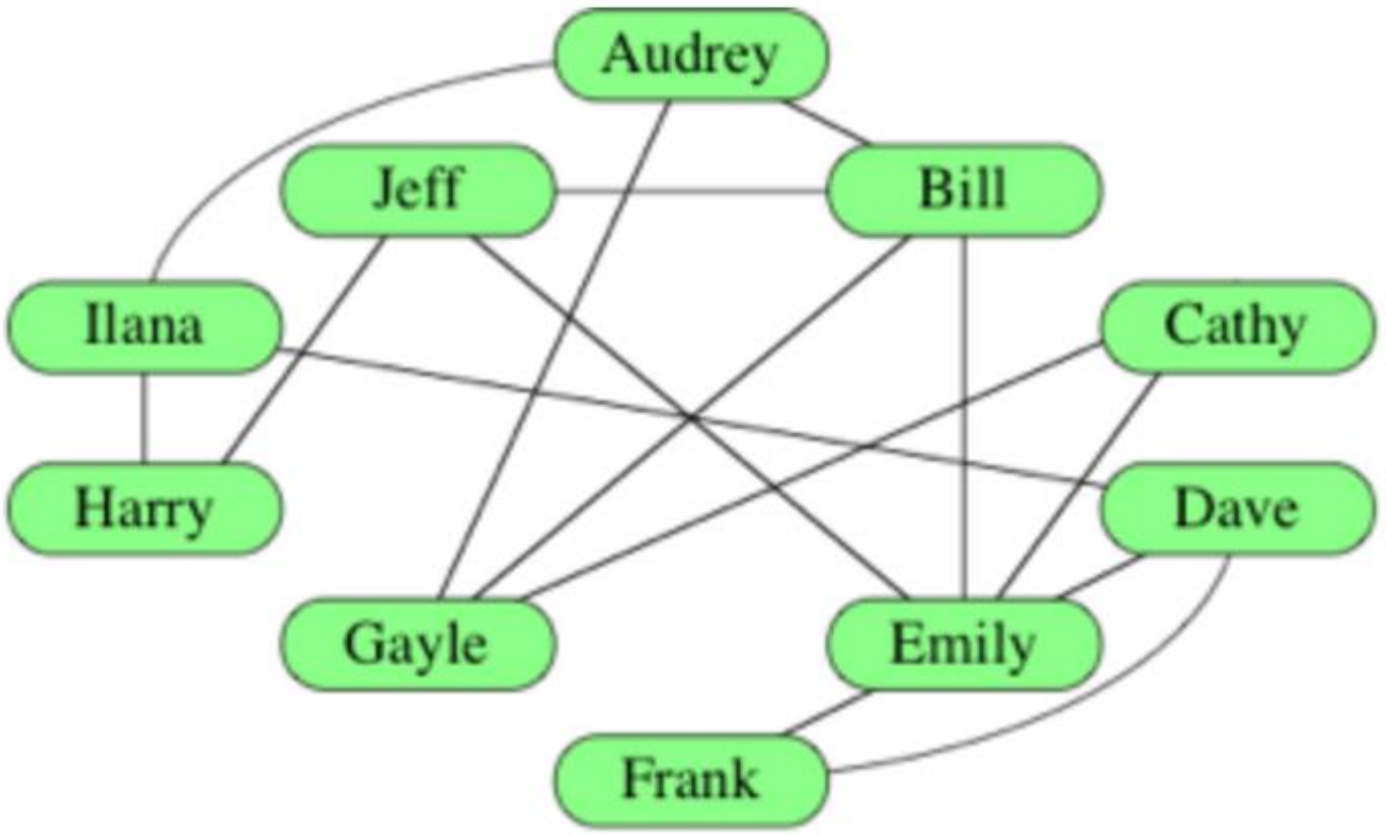
So you will see

$\Theta(V)$ or $\Theta(\log_2 E)$

Graphs

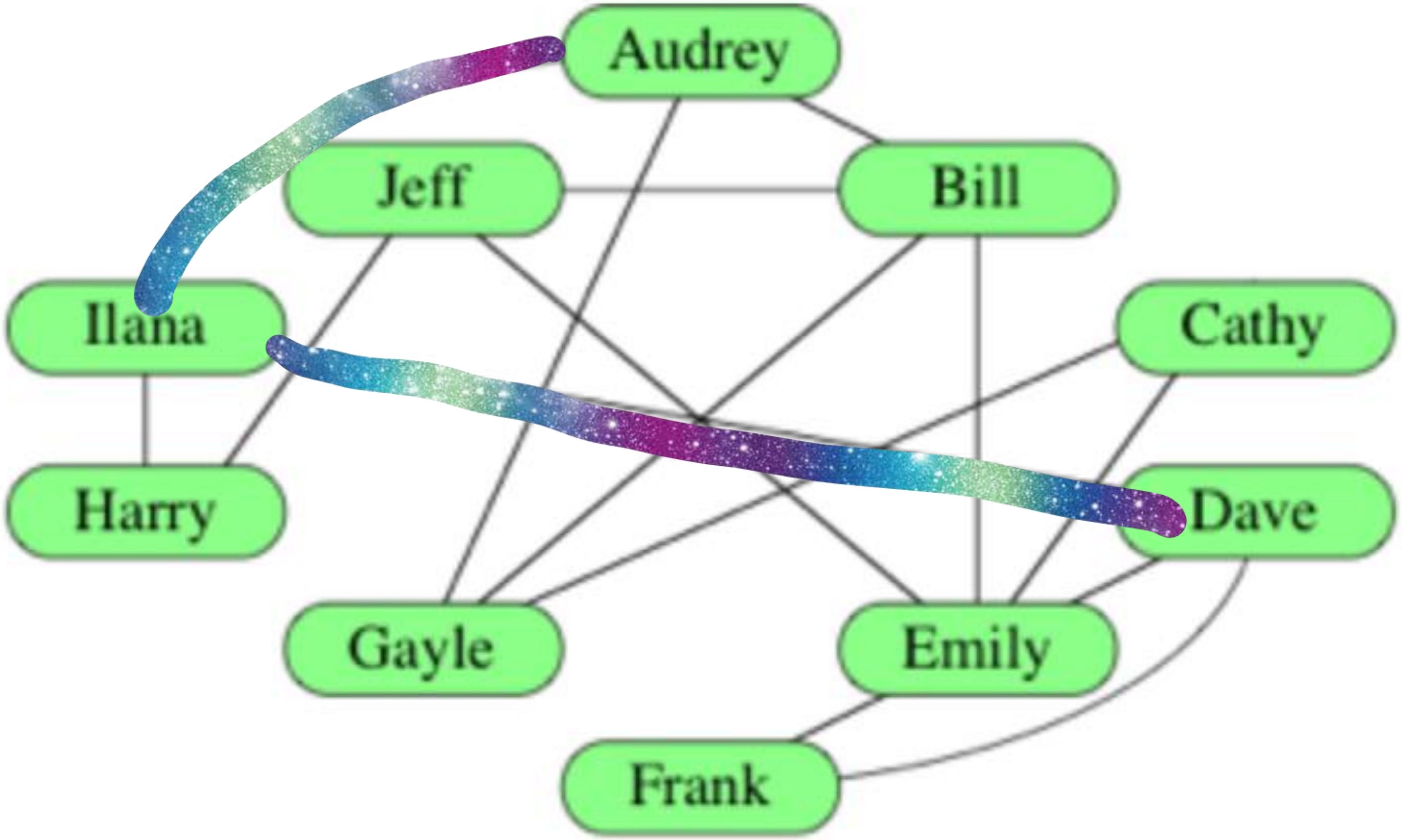
How many vertices are in the graph below?

10

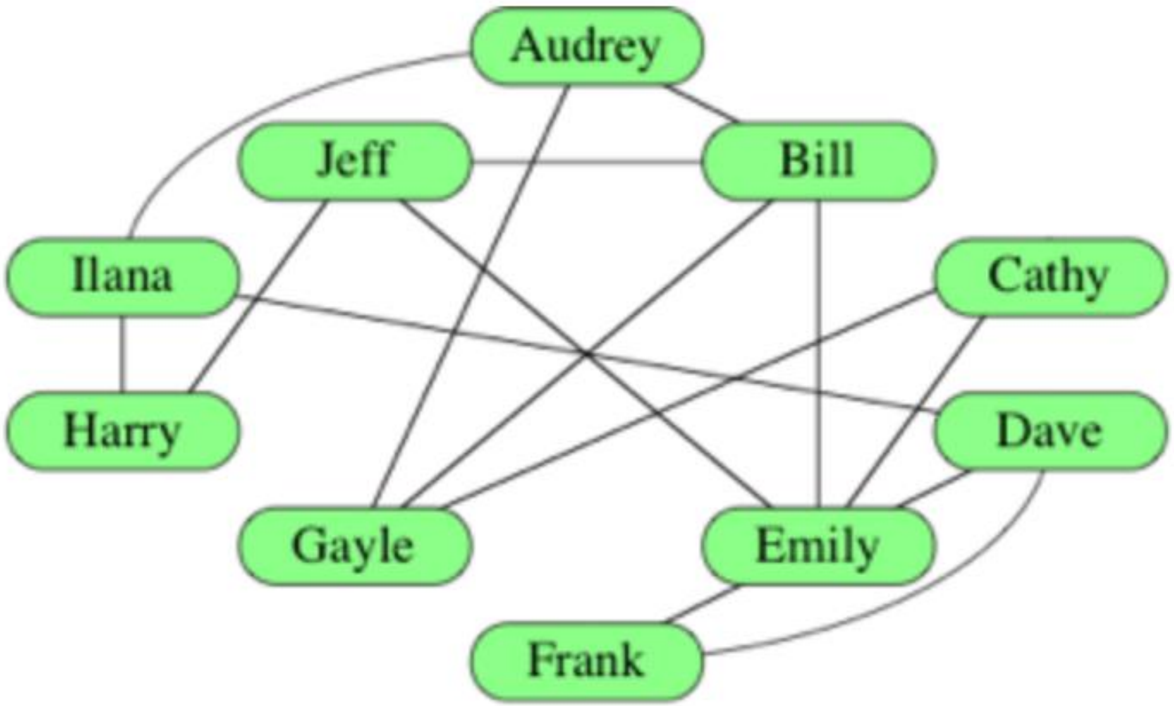


Graphs

What is the shortest path between Audrey and Dave in this graph?



Given this graph, identify cycles in it that contain Harry.



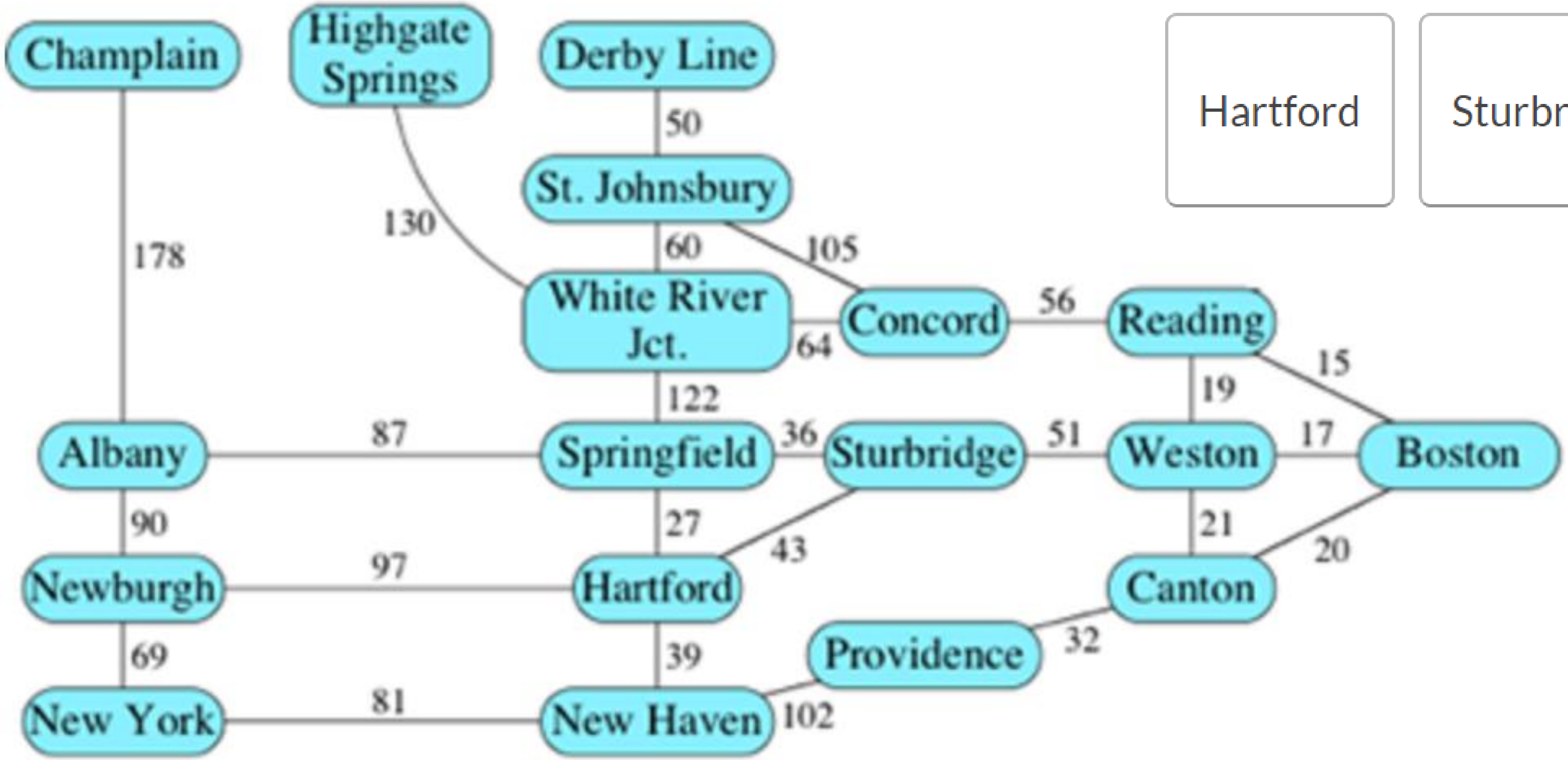
Remember that a cycle cannot contain repeated edges or vertices other than the starting one (Harry).

Choose all answers that apply:

- (A) Harry → Ilana → Audrey → Gayle → Bill → Jeff → Harry
- (B) Harry → Ilana → Audrey → Bill → Emily → Jeff → Harry
- (C) Harry → Jeff → Gayle → Harry
- (D) Harry → Jeff → Bill → Emily → Jeff → Harry
- (E) Harry → Jeff → Bill → Audrey → Ilana → Harry
- (F) Harry → Jeff → Bill → Gayle → Audrey → Ilana → Harry
- (G) Harry → Ilana → Audrey → Ilana → Harry

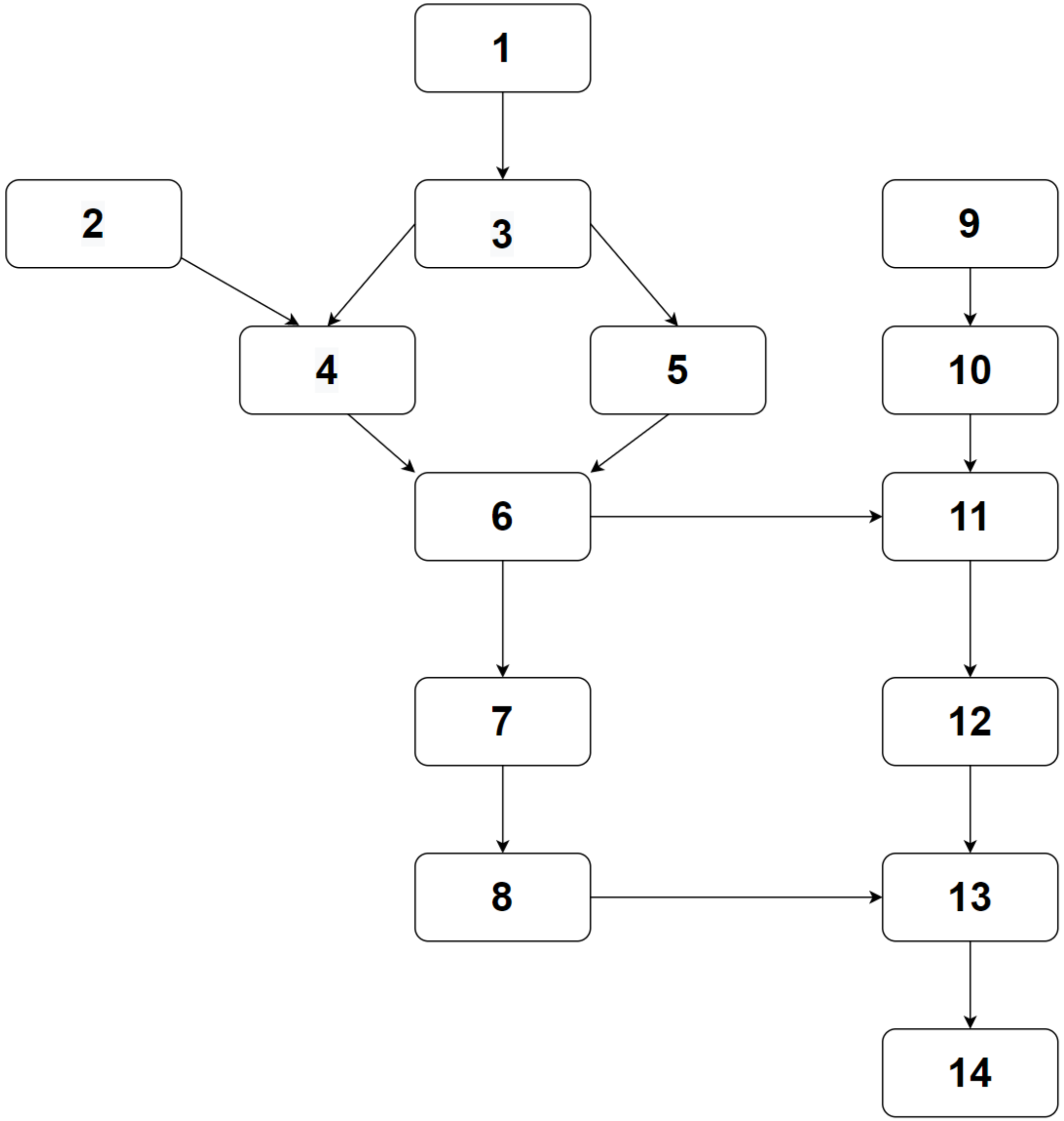
Graphs

What is the shortest path between Hartford and Canton?



Hartford | Sturbridge | Weston | Canton

Graphs



Vertex	In-degree	Out-degree
1		
3		
2		
4		
5		
6		

Graphs

There are several ways to represent graphs, each with its advantages and disadvantages.

Some situations, or algorithms that we want to run with graphs as input, call for one representation and others call for a different representation.

We will learn three ways to represent graphs.

Graphs

There is 3 criteria we will use when choosing how to represent a graph

1. How much memory space is needed for a representation

space

2. How long it takes to determine whether a given edge is in the graph
3. How long it takes to find the neighbors of a given vertex.

time

Graphs

In order to determine how much memory, or space, we need in each representation, we will use asymptotic notation.

We can use asymptotic notation for purposes other than expressing running times.

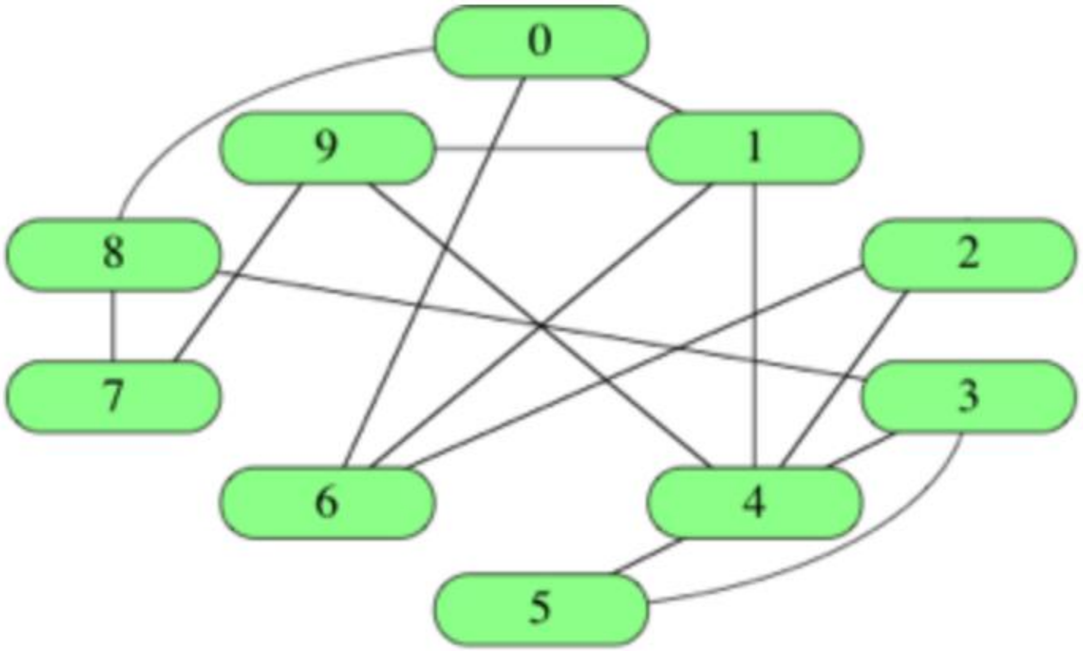
It's really a way to characterize *functions* and a function can describe a running time, an amount of space required, or some other resource.

Graphs

Let's start by identifying vertices by numbers rather than names.

We typically number the $|V|$ vertices from 0 to $|V|-1$

Remember that $|V|$ means the set of vertices.



Graphs

One simple way to represent a graph is an

edge list

which is just a list of $|E|$ edges.

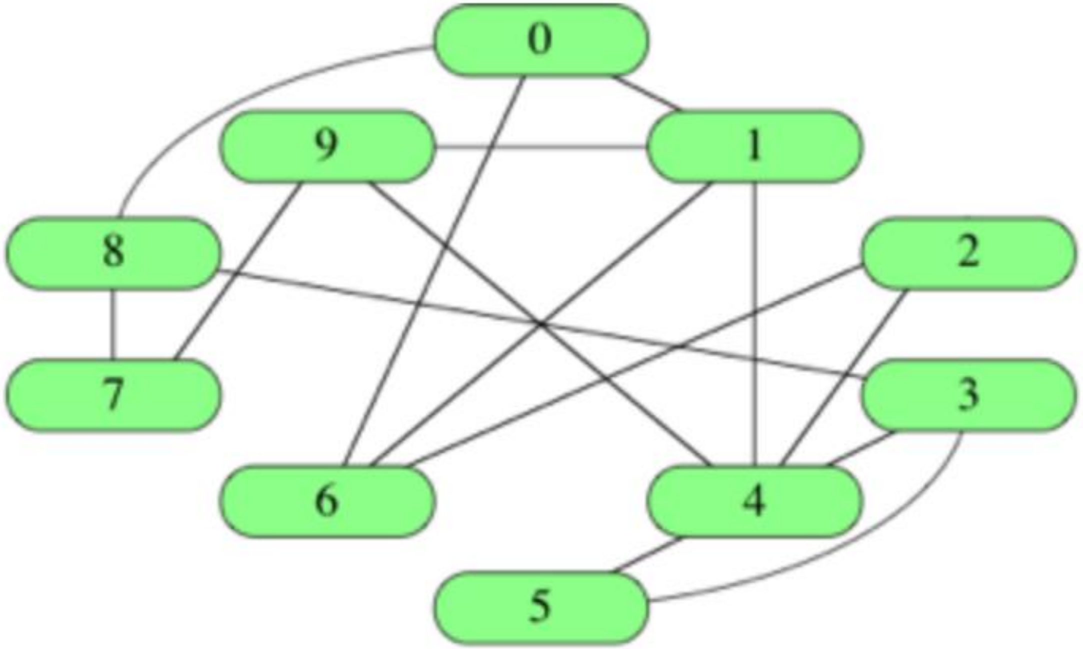
To represent an edge, create an array of two vertex numbers. In an OOP language, an array of objects could be used. An array of structs could also be used to create an edge list. A 2D array could be used also.

The total space needed by an edge list is $\Theta(E)$.

Graphs

This graph could be represented with an edge list like this…

{{0,1}, {4,5}, {2,4}, {0,6}, {1,4}, {3,8}, {1,6}, {2,6}, {3,4}, {0,8}, {3,5}, {4,9}, {7,8}, {1,9}, {7,9} }



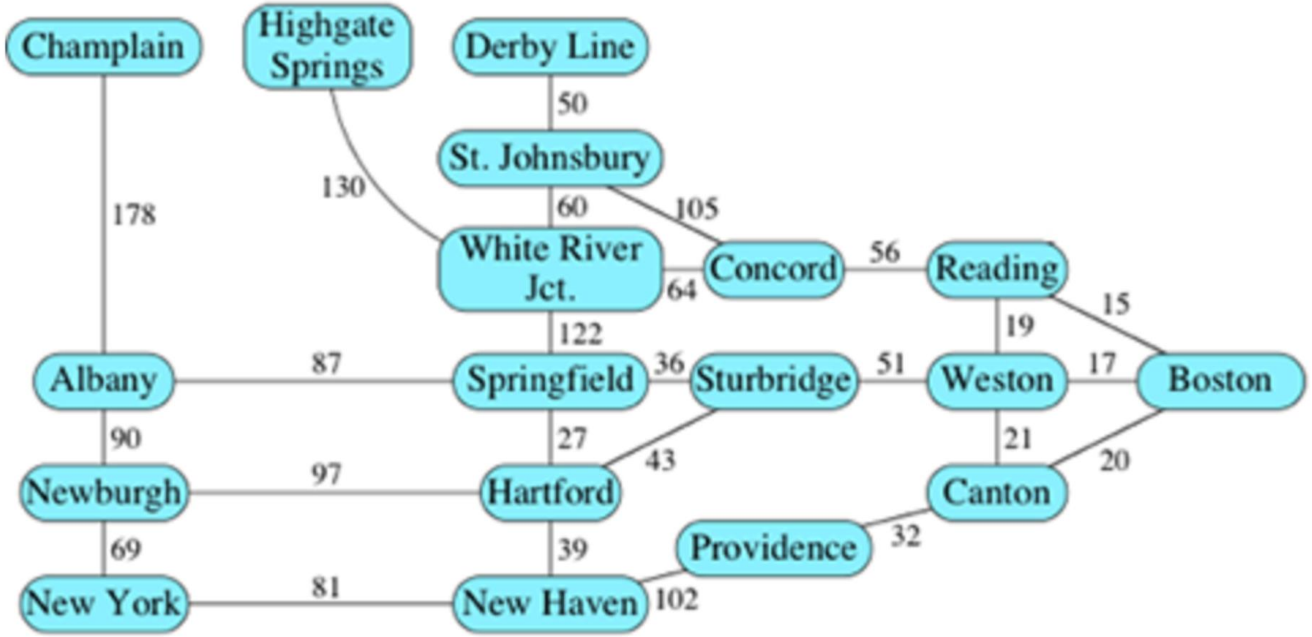
Graphs

What if the edges had weight?

If we are using objects in OOP, then we could add an attribute to the class.

If we are using arrays, we could add another dimension to the array.

If we are using structs, then we could add another member to the struct.



Graphs

Edge lists are simple but…

What if we wanted to find whether the graph contains a particular edge?

How would we find it?

Are the edges in any particular order?

{{0,1}, {4,5}, {2,4}, {0,6}, {1,4}, {3,8}, {1,6}, {2,6}, {3,4}, {0,8}, {3,5}, {4,9}, {7,8}, {1,9}, {7,9} }

A linear search would require $O(E)$ run time.

Graphs

{{0,1}, {4,5}, {2,4}, {0,6}, {1,4}, {3,8}, {1,6}, {2,6}, {3,4}, {0,8}, {3,5}, {4,9}, {7,8}, {1,9}, {7,9} }

A linear search would require $O(E)$ run time.

What could we change about the list to make the runtime $O(\log_2 E)$?

Binary search has a runtime of $O(\log_2 n)$…

Could we organize/sort our edge list so that a binary search could be used?

Graphs

Could we organize/sort our edge list so that a binary search could be used?

We could sort the list by the 1st vertex and then the 2nd vertex.

{{0,1}, {0,6}, {0,8}, {1,4}, {1,6}, {1,9}, {2,4}, {2,6}, {3,4}, {3,5}, {3,8}, {4,5}, {4,9}, {7,8}, {7,9} }

How would the binary search work?

It would still pick the middle element – in this example, array element {2,6} would be in the middle. To decide whether to chose the right half of the array or the left requires a little more work.

Graphs

{{0,1}, {0,6}, {0,8}, {1,4}, {1,6}, {1,9}, {2,4}, {2,6}, {3,4}, {3,5}, {3,8}, {4,5}, {4,9}, {7,8}, {7,9} }

If {2,6} is the middle element, how do we decide whether to go left or right?

Edge (V1, V2) is less than another edge (V3,V4)

if (V1 < V3)

OR

(V1 == V3) AND (V2 < V4)

Edge (V1, V2) is less than another edge (V3,V4)

Graphs

if (V1 < V3)
OR
(V1 == V3) AND (V2 < V4)

{{0,1}, {0,6}, {0,8}, {1,4}, {1,6}, {1,9}, {2,4}, {2,6}, {3,4}, {3,5}, {3,8}, {4,5}, {4,9}, {7,8}, {7,9} }

If we are searching for edge {0,8}, then

V1 = 2 and V2 = 6 and V3 = 0 and V4 = 8

if (2 < 0)   FALSE
OR
(2 == 0) AND (6 < 8) F AND T = FALSE
FALSE OR FALSE is FALSE so edge {0,8} is to the left of {2,6}

If we are searching for edge {2,4} then

V1 = 2 and V2 = 6 and V3 = 2 and V4 = 4

if (2 < 2)   FALSE
OR
(2 == 2) AND (6 < 4)   FALSE

FALSE OR FALSE is FALSE so edge {2,4} is to the left of {2,6}

Graphs

Adjacency Matrices

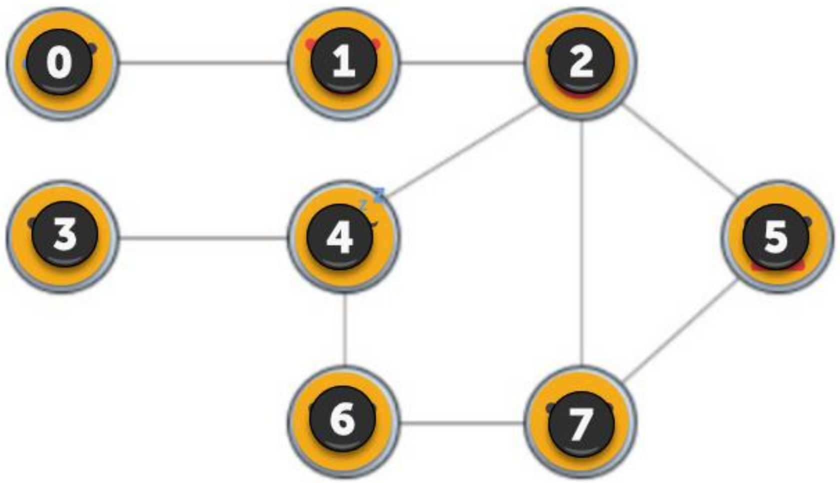
For a graph with $|V|$ vertices, an **adjacency matrix** is a $|V|\times|V|$ matrix of zeroes and ones.

The entry in row i and column j is 1 **if and only if** the edge (i,j) is in the graph. A value of 0 indicates there is no edge between i and j.

To indicate an edge weight, put the weight in the row i, column j entry and reserve a special value (-1 for example) to indicate an absent edge.

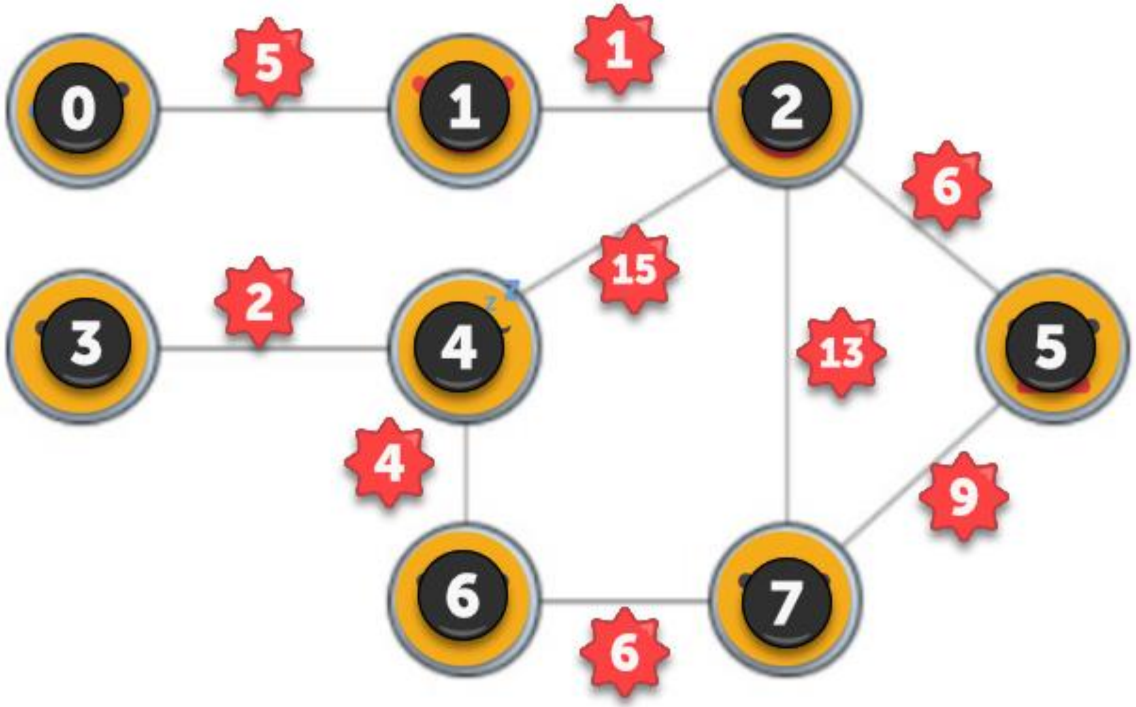
Graphs

	0	1	2	3	4	5	6	7
0	0	1	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0
2	0	1	0	0	1	1	0	1
3	0	0	0	0	1	0	0	0
4	0	0	1	1	0	0	1	0
5	0	0	1	0	0	0	0	1
6	0	0	0	0	1	0	0	1
7	0	0	1	0	0	1	1	0



Graphs

	0	1	2	3	4	5	6	7
0	-1	-1	-1	-1	-1	-1	-1	-1
1	-1	-1	-1	-1	-1	-1	-1	-1
2	-1	-1	-1	-1	-1	-1	-1	-1
3	-1	-1	-1	-1	-1	-1	-1	-1
4	-1	-1	-1	-1	-1	-1	-1	-1
5	-1	-1	-1	-1	-1	-1	-1	-1
6	-1	-1	-1	-1	-1	-1	-1	-1
7	-1	-1	-1	-1	-1	-1	-1	-1



Graphs

We can determine if an edge is present in the adjacency matrix just by looking up the vertices of the edge – an edge's *i*,*j* value.

If we create a 2D array named `AM`, then we could just look up `AM[i][j]`

This look up takes a constant amount of time.

Sounds pretty good?

	0	1	2	3	4	5	6	7
0	0	1	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0
2	0	1	0	0	1	1	0	1
3	0	0	0	0	1	0	0	0
4	0	0	1	1	0	0	1	0
5	0	0	1	0	0	0	0	1
6	0	0	0	0	1	0	0	1
7	0	0	1	0	0	1	1	0

Graphs

Disadvantages of adjacency matrix

- Adjacency matrix requires $\Theta(V^2)$ space to store a graph
    - Our array was mostly zeroes – we filled in 18 of 64 cells
    - Our graph is **sparse** – relatively few edges

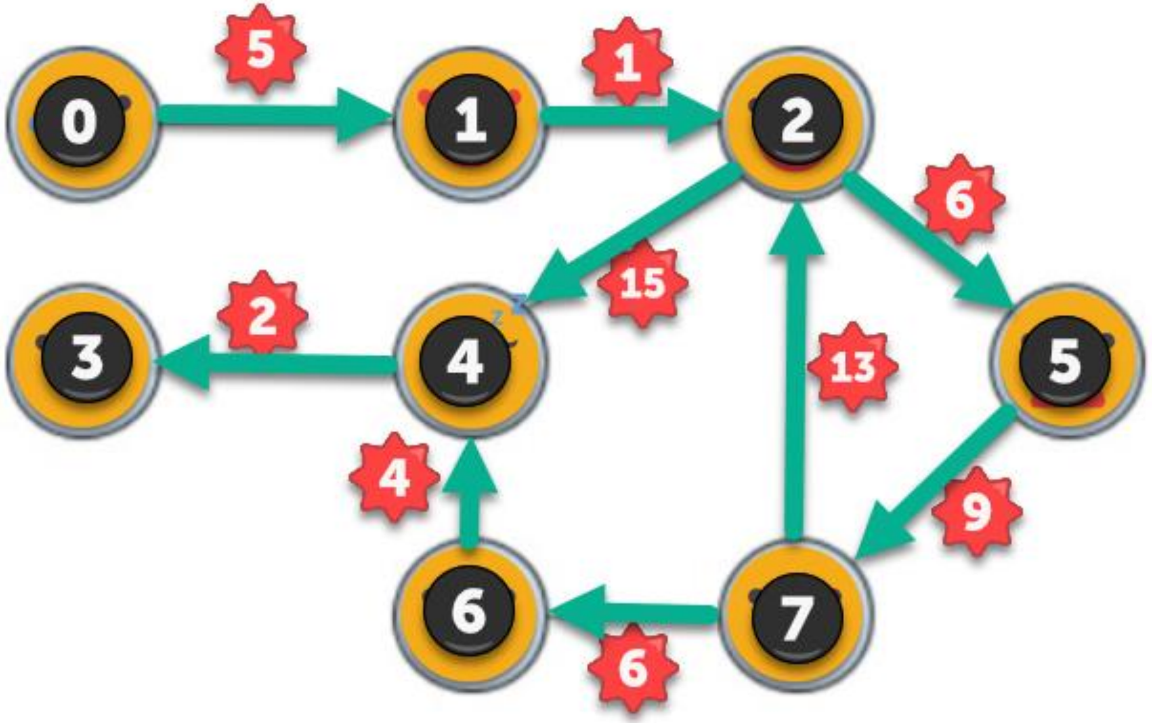
- To find which vertices are adjacent to each other, we have to look at all $|V|$ entries in row i, even if only a few vertices are adjacent to vertex i.

Graphs

Undirected graph's adjacency matrix is symmetric

Directed graph's adjacency matrix need not be symmetric

	0	1	2	3	4	5	6	7
0	-1	-1	-1	-1	-1	-1	-1	-1
1	-1	-1	-1	-1	-1	-1	-1	-1
2	-1	-1	-1	-1	-1	-1	-1	-1
3	-1	-1	-1	-1	-1	-1	-1	-1
4	-1	-1	-1	-1	-1	-1	-1	-1
5	-1	-1	-1	-1	-1	-1	-1	-1
6	-1	-1	-1	-1	-1	-1	-1	-1
7	-1	-1	-1	-1	-1	-1	-1	-1



Graphs

Adjacency Lists

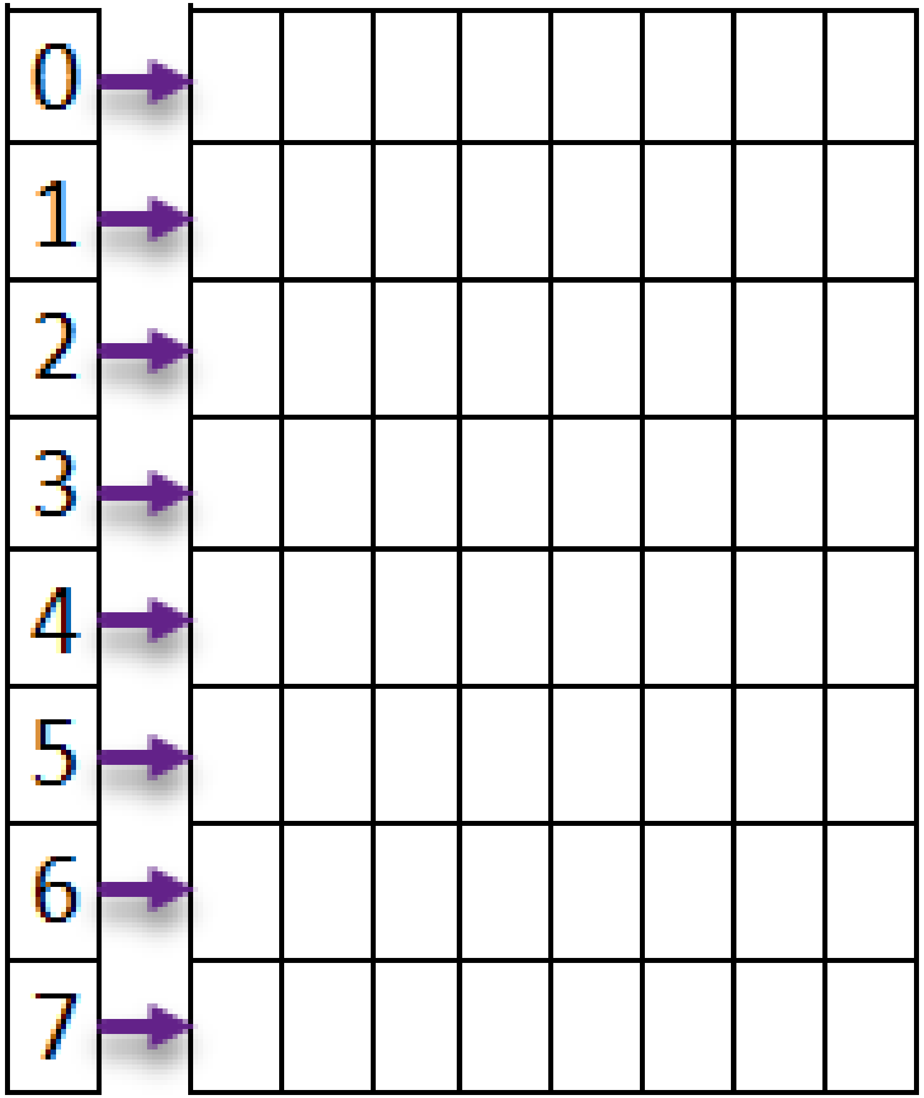
We can combine adjacency matrices with edge lists.

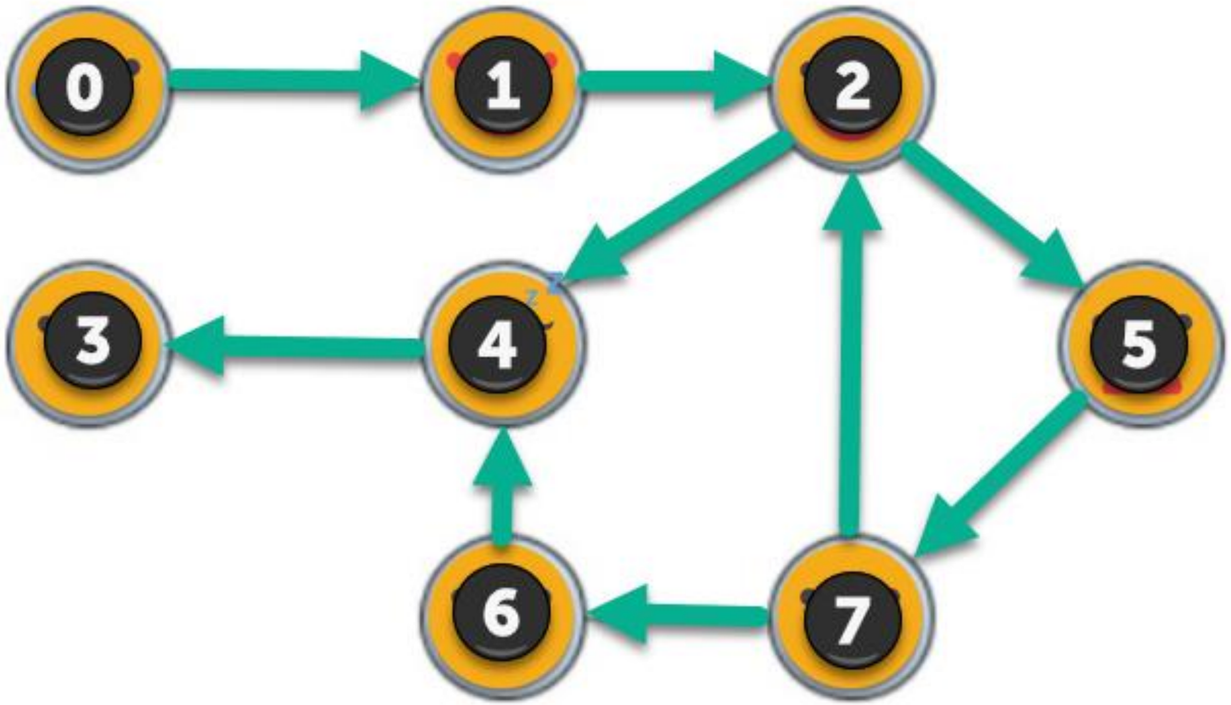
For each vertex i, store an array of the vertices adjacent to it.

We will need an array of $|V|$ adjacency lists where there is one adjacency list per vertex.

Graphs

Adjacency Lists





Graphs

Adjacency Lists

Vertex numbers in an adjacency list are not required to appear in any particular order.

There is a benefit to listing them in increasing order despite the cost of doing the ordering at the time of creating the adjacency list.

Graphs

Adjacency Lists

How much time does it take to find a vertex's adjacency list?

A constant amount of time since we are just going to an array index.

To find if an edge(x,y) is in a graph, we go to x's adjacency list in constant time and then look for y in x's adjacency list.

Graphs

Adjacency Lists

So how long does the worst case search take for edge (x,y)?

How many possible entries could any one vertex have?

Let's recall this definition

The number of edges incident on a vertex is the degree of the vertex.

Use d to represent the degree of a vertex

Graphs

Adjacency Lists

So how long does the worst case take?

$\Theta(d)$

where d is the degree of vertex x because that's how long vertex x's adjacency list is.

What is the highest value (degree) possible for any vertex in $|V|$?

Graphs

Adjacency Lists

What is the highest value (degree) possible for any vertex in $|V|$?

Any vertex in a graph could be adjacent (share an edge with) to every other vertex.

so the degree of any vertex could be as high as $|V|$ - 1 and as low as 0*

*a vertex can be isolated with no incident edges

Graphs

Adjacency Lists

How much space does an adjacency list use?

Every vertex ($|V|$) can have a list of adjacent vertices ($|V|$-1)

For an undirected graph, the adjacency list will contain $2|E|$ elements.

For a directed graph, the adjacency list will contain |E| edges – one element per directed edge.

Graphs

Adjacency Lists

How much space does an adjacency list use?

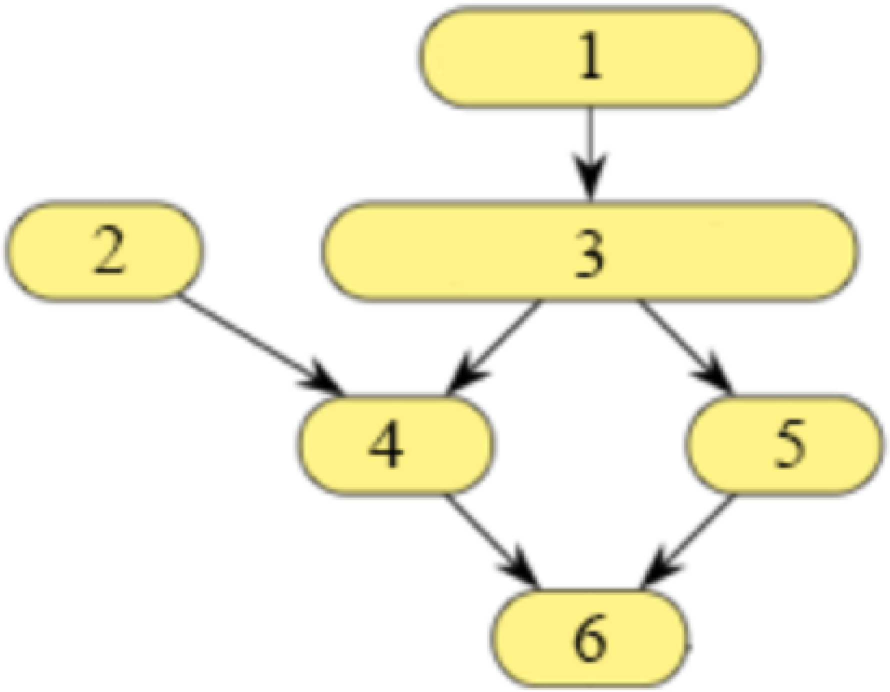
An adjacency list is a list of lists.

Each list corresponds to a vertex u and contains a list of edges (x, y) that originate from x.

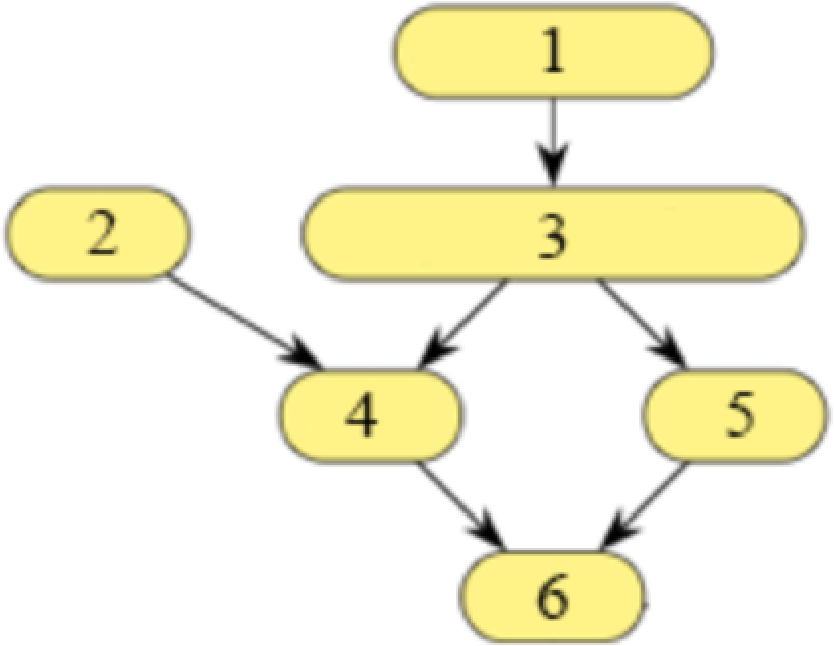
An adjacency list takes up $\Theta(V + E)$ space.

Worst case is when the graph is dense and $E = \Theta(V^2)$

Given the following directed graph, how would you represent it with an edge list?

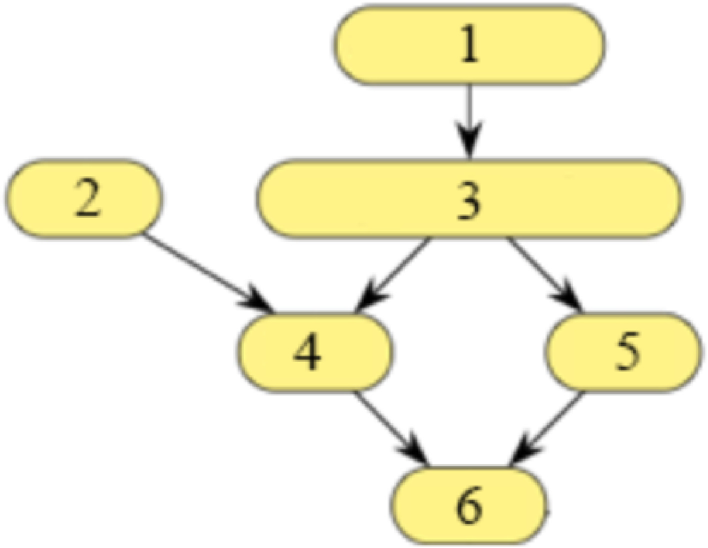


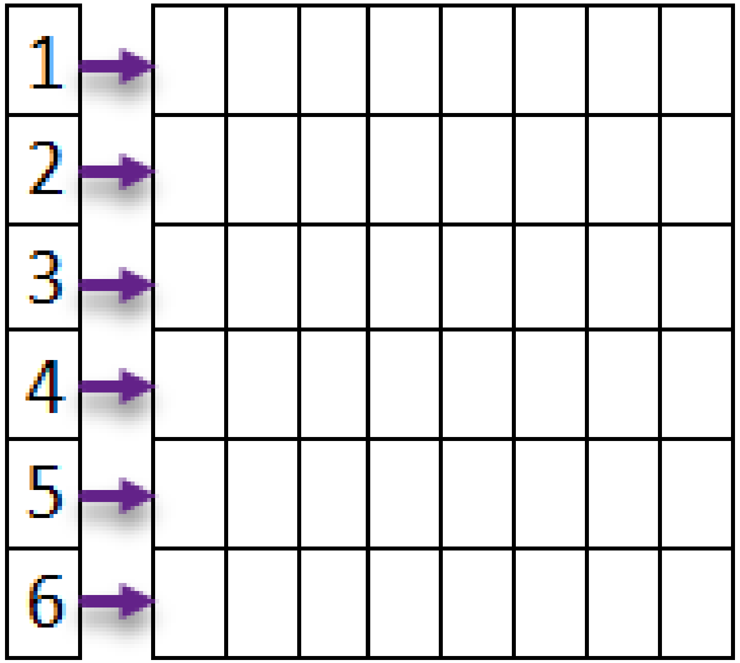
Given the following directed graph, how would you represent it with an adjacency matrix?



	1	2	3	4	5	6
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0
6	0	0	0	0	0	0

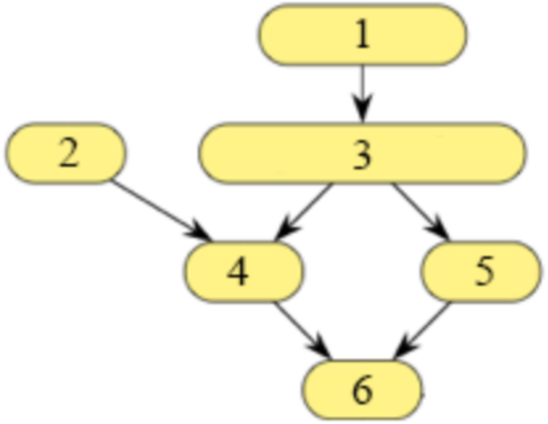
Given the following directed graph, how would you represent it with an adjacency list?





For each vertex i, write the vertices that are adjacent to it, one in each cell. Leave cells empty that you don't need.

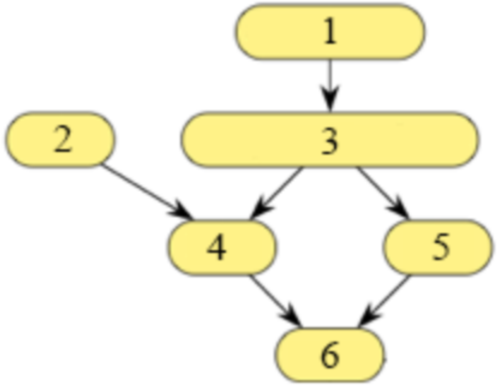
We've seen three ways to store graphs - edge lists, adjacency matrices, and adjacency lists. For an directed graph like the one shown below, how much space do we need for each type of storage?



Assuming E is the number of edges and V is the number of vertices, categorize the space below:

	$\Theta(E)$	$\Theta(V + E)$	$\Theta(V^2)$
edge list	✓		
adjacency matrix			✓
adjacency list		✓	

We've seen three ways to store graphs - edge lists, adjacency matrices, and adjacency lists. For an directed graph like the one shown below, how much time would it take to search for a particular edge through each way of storing the graph?



Assuming E is the number of edges, V is the number of vertices, and d is the degree of each vertex, categorize the time taken below:

	$O(1)$	$O(d)$	$O(E)$
edge list (unordered)			✓
adjacency matrix	✓		
adjacency list		✓	