Lecture 15: Intro to Graphs

CSE 373: Data Structures and Algorithms

Slido Event #2085756
https://app.sli.do/event/3YyoRiYobDRxkEvADqWofr

Announcements

- EX 2 regrade requests due 5/7
  - Note these are re*grades* not re*submissions*
- EX 4 due today
- EX 5 out later today
- P3 due next Wednesday 5/10
- Midterm grades will be out by Wed 5/3
  - Resubmission will open on Wednesday
  - You will receive numerical scores per question
  - Resubmit only the questions you want to re-do
    - those you skip we will just carry your initial score over
  - final midterm score will be the average of your two submissions

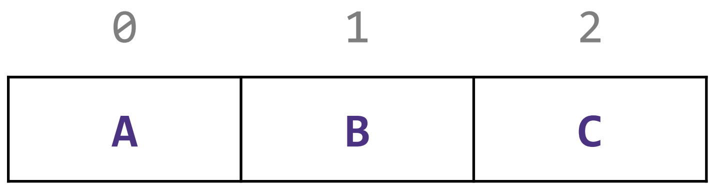
Introduction to Graphs

Graph Modelling

Inter-data Relationships

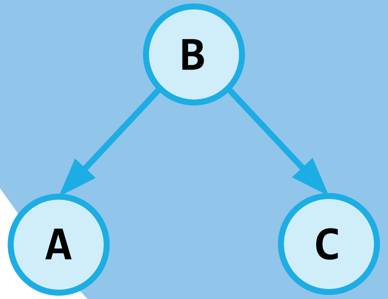
Arrays

- Elements only store pure data, no connection info
- Only relationship between data is order



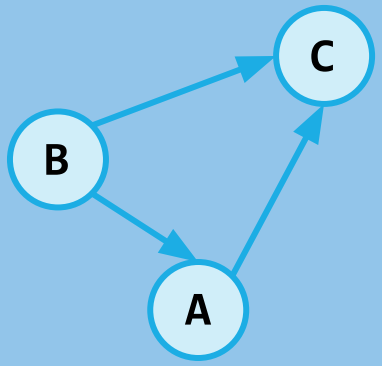
Trees

- Elements store data and connection info
- Directional relationships between nodes; limited connections



Graphs

- Elements AND connections can store data
- Relationships dictate structure; huge freedom with connections



Graphs

- Everything is graphs.
- Most things we've studied this quarter can be represented by graphs.
  - BSTs are graphs
  - Linked lists? Graphs.
  - Heaps? Also can be represented as graphs.
  - Those trees we drew in the tree method? Graphs.
- But it's not just data structures that we've discussed…
  - Google Maps database? Graph.
  - Facebook? They have a "graph search" team. Because it's a graph
  - Gitlab's history of a repository? Graph.
  - Those pictures of prerequisites in your program? Graphs.
  - Family tree? That's a graph

Applications

Physical Maps

- Airline maps
  - Vertices are airports, edges are flight paths
- Traffic
  - Vertices are addresses, edges are streets

Relationships

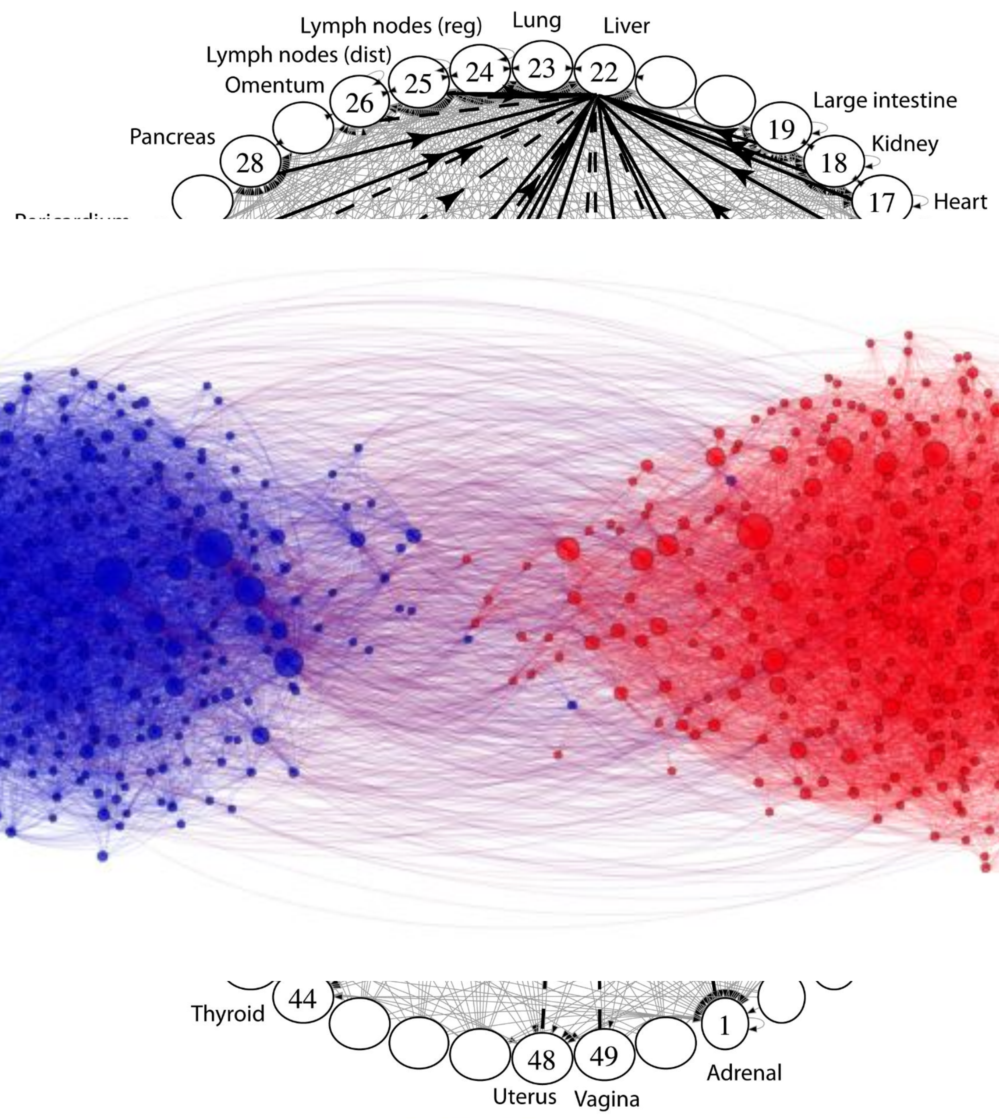
- Social media graphs
  - Vertices are accounts, edges are follower relationships
- Traffic
  - Vertices are classes, edges are usage

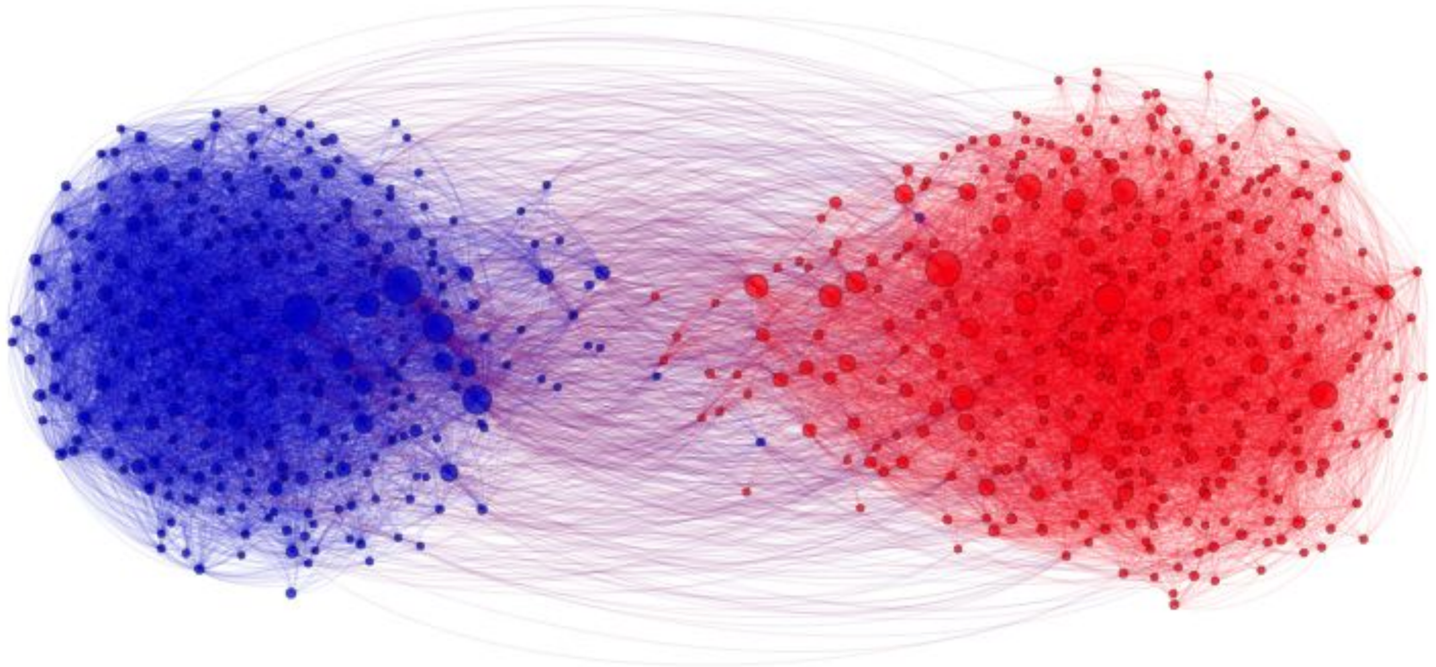
Influence

- Biology
  - Vertices are cancer cell desinations, edges are migration paths

Related topics

- Web Page Ranking
  - Vertices are web pages, edges are hyperlinks
- Wikipedia
  - Vertices are articles, edges are links





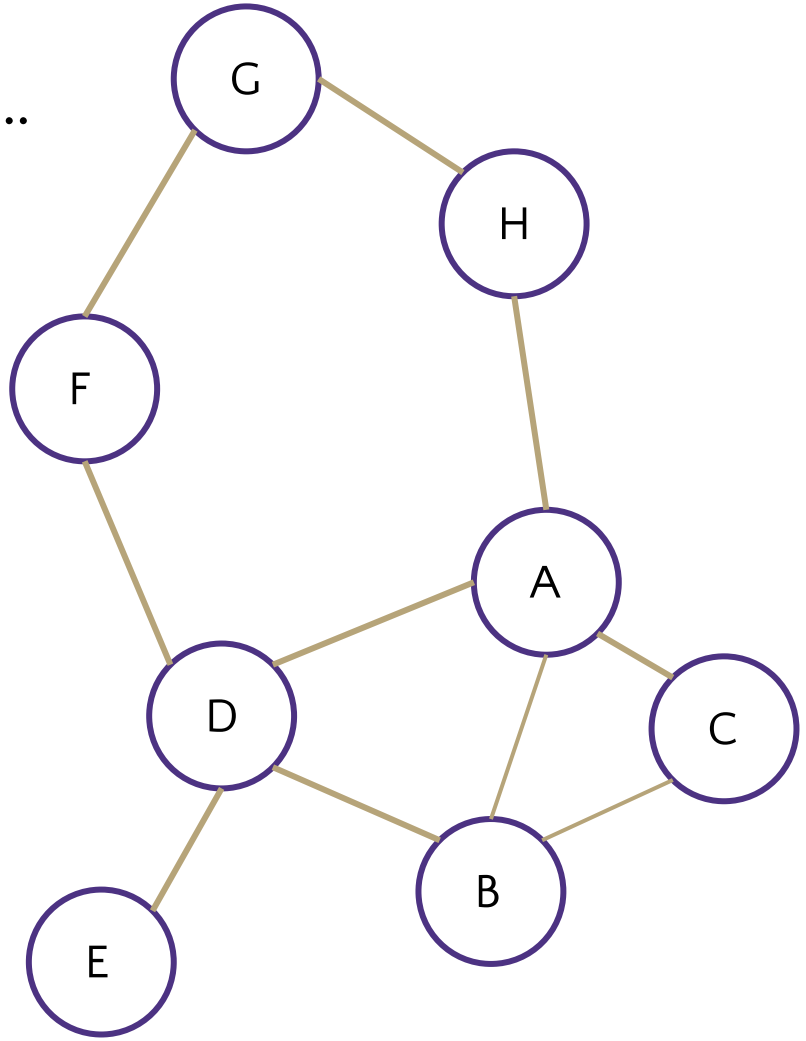
And so many more!!

www.allthingsgraphed.com

Graph: Formal Definition

A **graph** is defined by a pair of sets G = (V, E) where…

- V is a set of **vertices**
  - A vertex or "node" is a data entity
  - V = { A, B, C, D, E, F, G, H }
- E is a set of **edges**
  - An edge is a connection between two vertices
  - E = { (A, B), (A, C), (A, D), (A, H), (C, B), (B, D), (D, E), (D, F), (F, G), (G, H)}

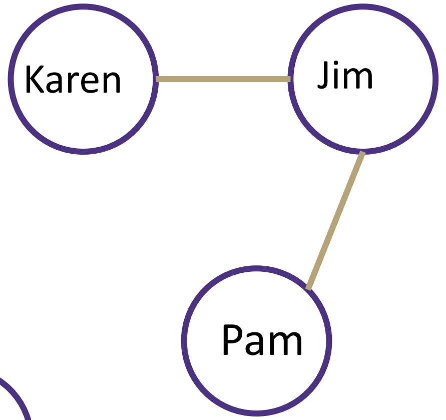


Graph Vocabulary

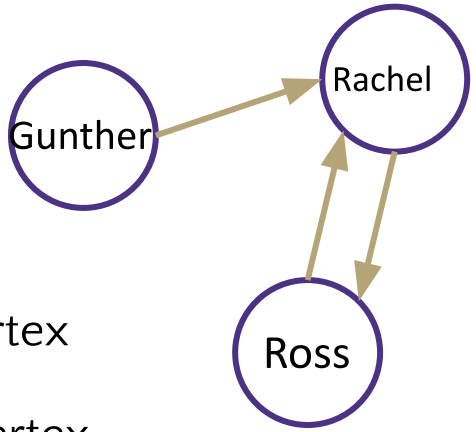
Graph Direction

- **Undirected graph** – edges have no direction and are two-way
  - V = { Karen, Jim, Pam }
  - E = { (Jim, Pam), (Jim, Karen) } *inferred (Karen, Jim) and (Pam, Jim)*
- **Directed graphs** – edges have direction and are thus one-way
  - V = { Gunther, Rachel, Ross }
  - E = { (Gunther, Rachel), (Rachel, Ross), (Ross, Rachel) }

Undirected Graph:



Directed Graph:



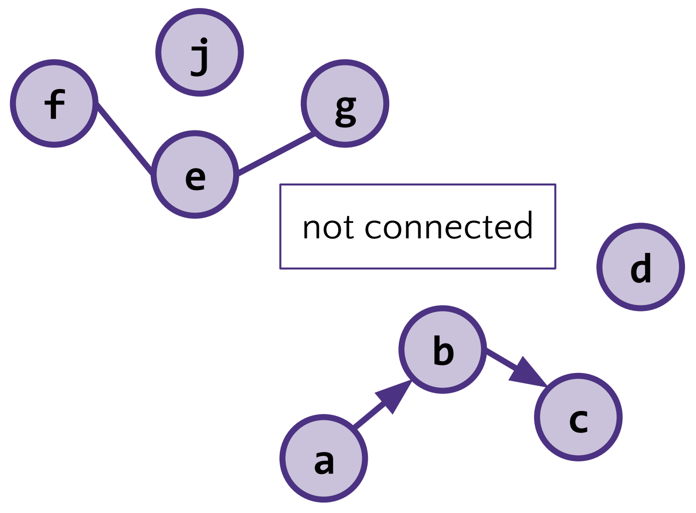
Degree of a Vertex

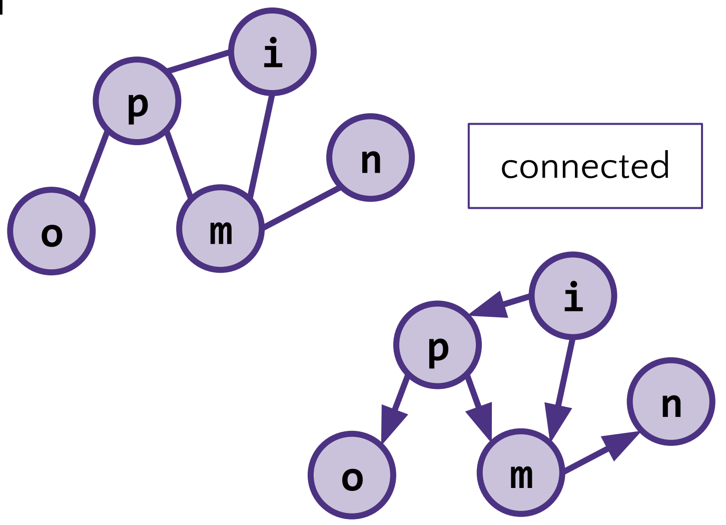
- **Degree** – the number of edges connected to that vertex
  - Karen : 1, Jim : 1, Pam : 1
- **In-degree** – the number of directed edges that point to a vertex
  - Gunther : 0, Rachel : 2, Ross : 1
- **Out-degree** – the number of directed edges that start at a vertex
  - Gunther : 1, Rachel : 1, Ross : 1

More More Graph Terminology

Two vertices are **connected** if there is a path between them

- If all the vertices are connected, we say the graph is **connected**
  - A directed graph is **weakly connected** if replacing every directed edge with an undirected edge results in a connected graph
  - A directed graph is **strongly connected** if a directed path exists between every pair of vertices
- The number of edges leaving a vertex is its **degree**





A **path** is a sequence of vertices connected by edges

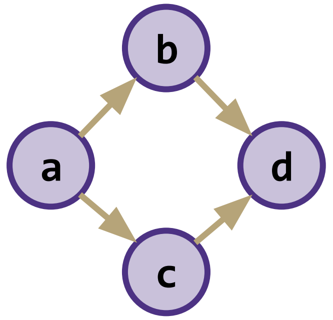
- A **simple path** is a path without repeated vertices
- A **cycle** is a path whose first and last vertices are the same
  - A graph with a cycle is **cyclic**

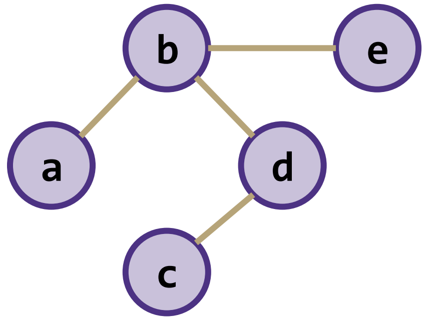
Directed vs Undirected; Acyclic vs Cyclic

Directed:

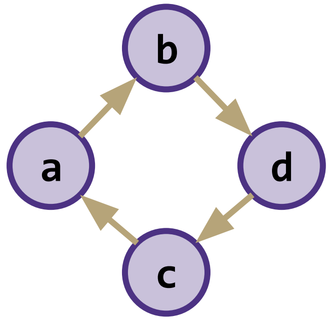
Undirected:

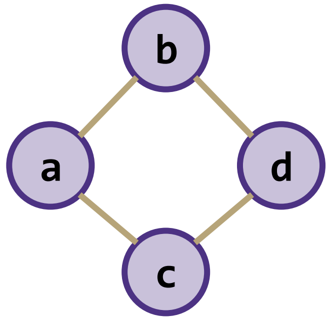
Acyclic:





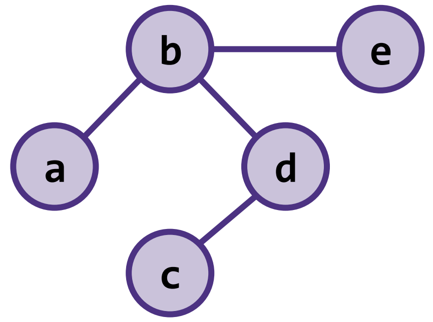
Cyclic:



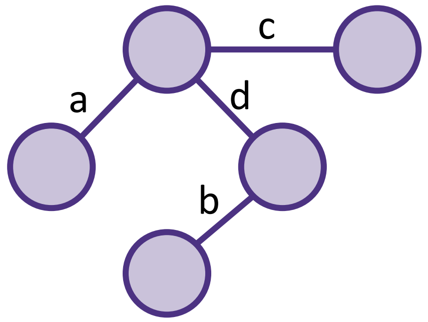


Labeled and Weighted Graphs

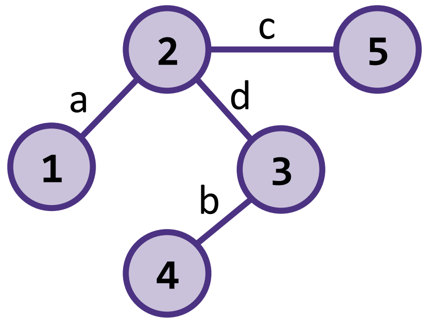
Vertex Labels



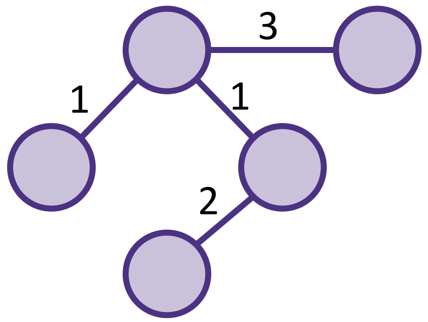
Edge Labels



Vertex & Edge Labels



Numeric Edge Labels
(Edge Weights)



Multi-Variable Analysis

- So far, we thought of everything as being in terms of some single argument “n” (sometimes its own parameter, other times a size)
  - But there’s no reason we can’t do reasoning in terms of multiple inputs!
- Why multi-variable?
  - Remember, algorithmic analysis is just a tool to help us understand code. Sometimes, it helps our understanding more to build a Oh/Omega/Theta bound for multiple factors, rather than handling those factors in case analysis.
- With graphs, we usually do our reasoning in terms of:
  - n (or |V|): total number of vertices (sometimes just call it V)
  - m (or |E|): total number of edges (sometimes just call it E)
  - deg(u): degree of node u (how many outgoing edges it has)

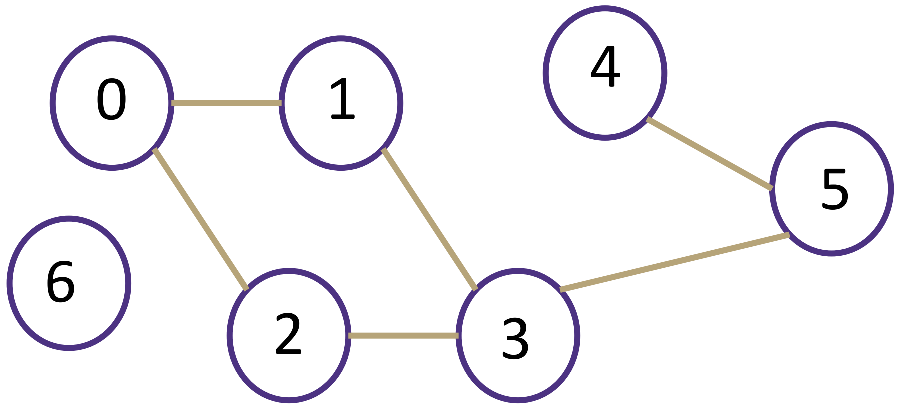
Adjacency Matrix

In an adjacency matrix a[u][v] is 1 if there is an edge (u,v), and 0 otherwise.

Worst-case Time Complexity ($|V| = n$, $|E| = m$):

- Add Edge: $\Theta(1)$
- Remove Edge: $\Theta(1)$
- Check edge exists from (u,v): $\Theta(1)$
- Get outneighbors of u: $\Theta(n)$
- Get inneighbors of u: $\Theta(n)$

Space Complexity: $\Theta(n * n)$



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

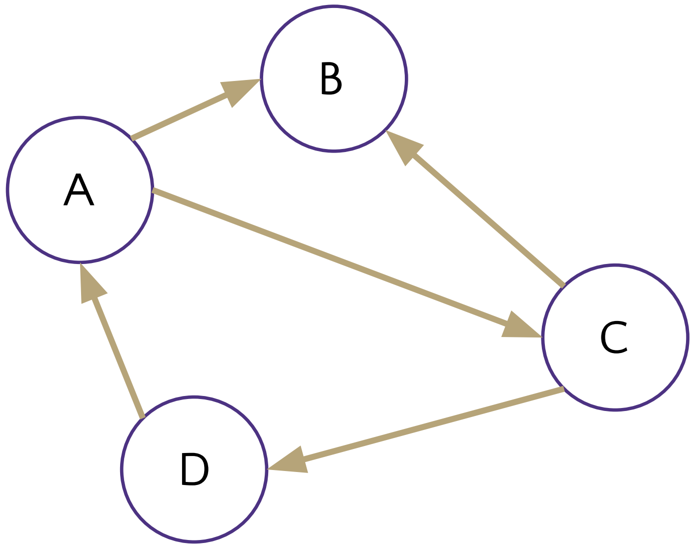
Adjacency List

In an adjacency matrix a[u][v] is 1 if there is an edge (u,v), and 0 otherwise.

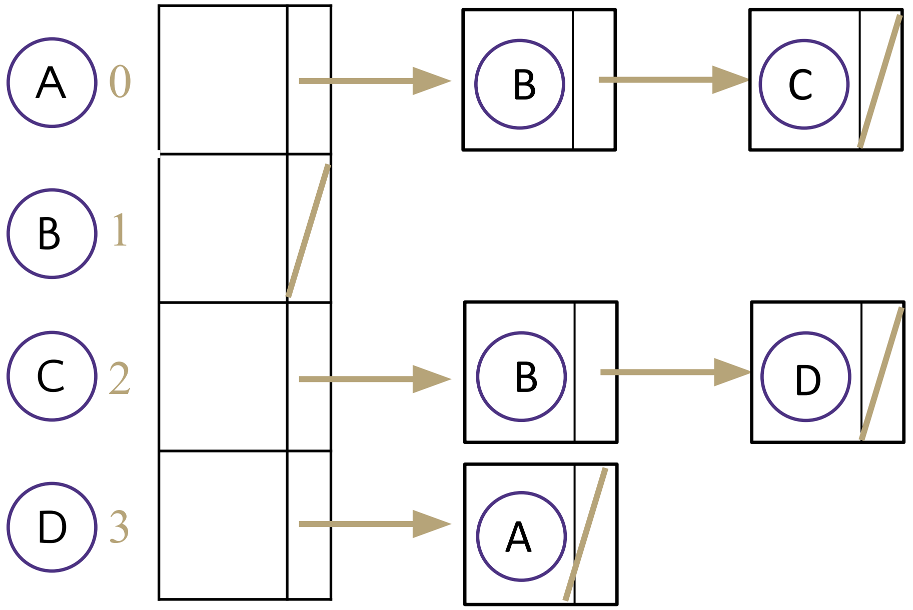
Worst-case Time Complexity (|V| = n, |E| = m):

- Add Edge: $\Theta(1)$
- Remove Edge: $\Theta(\deg(u))$
- Check edge exists from (u,v): $\Theta(\deg(u))$
- Get outneighbors of u: $\Theta(\deg(u))$
- Get inneighbors of u: $\Theta(n + m)$

Space Complexity: $\Theta(n + m)$



Linked Lists



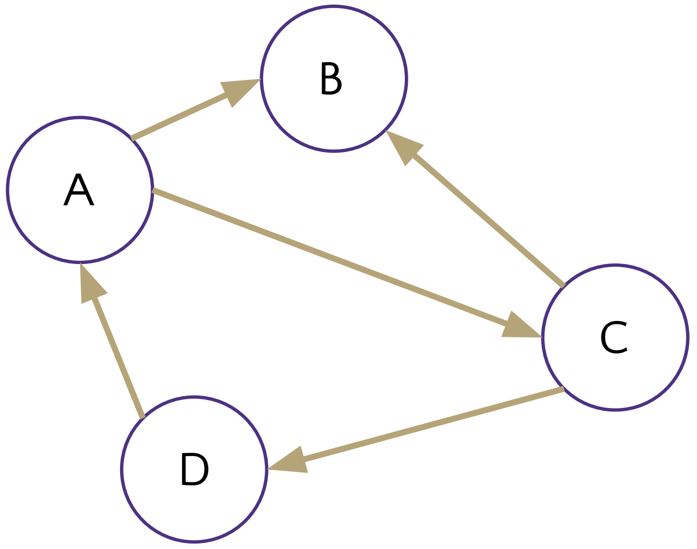
Adjacency List

In an adjacency matrix a[u][v] is 1 if there is an edge (u,v), and 0 otherwise.

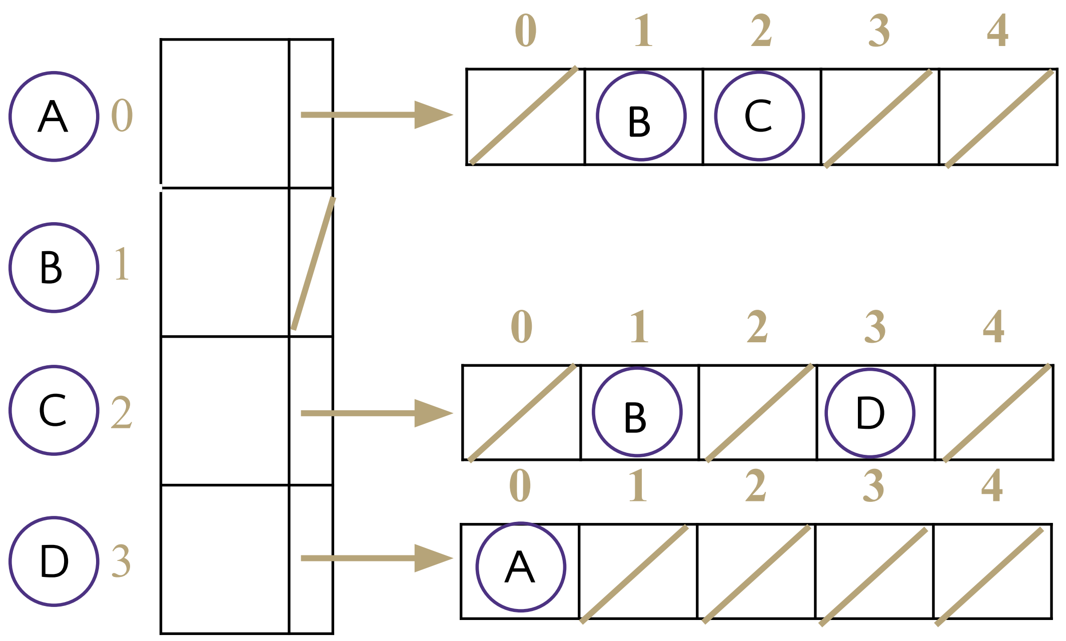
Worst-case Time Complexity (|V| = n, |E| = m):

- Add Edge: $\Theta(1)$
- Remove Edge: $\Theta(1)$
- Check edge exists from (u,v): $\Theta(1)$
- Get outneighbors of u: $\Theta(\deg(u))$
- Get inneighbors of u: $\Theta(n)$

Space Complexity: $\Theta(n + m)$



Hash Tables



Tradeoffs

Adjacency Matrices take more space, why would you use them?

- For **dense** graphs (where *m* is close to n^2), the running times will be close
- And the constant factors can be much better for matrices than for lists
- Sometimes the matrix itself is useful ("spectral graph theory")

What's the tradeoff between using linked lists and hash tables for the list of neighbors?

- A hash table still *might* hit a worst-case
- And the linked list might not
  - Graph algorithms often just need to iterate over all the neighbors, so you might get a better guarantee with the linked list

373: Graph Implementations

For this class, unless we say otherwise, we'll assume the hash tables operations **on graphs** are all O(1)

- Because you can probably control the keys

Unless we say otherwise, assume we're using the hash table approach

Questions?

relevant ideas for today

- vertices, edges, definitions
- graphs model relationships between real data (you can choose your vertices and edges to
- different graph implementations exist

Introduction to Graphs

Graph Modelling

Some examples

- For each of the following think about what you should choose for vertices and edges.
- **The internet**

  Vertices: webpages. **Edges** from a to b if a has a hyperlink to b.

  Directed, since hyperlinks go in one direction
- **Family tree**

  Vertices: webpages. **Edges** from a to b if a has a hyperlink to b.

  Directed, since hyperlinks go in one direction
- **Input data for the "6 Degrees of Kevin Bacon" game**

  Vertices: actors. **Edges**: movies

  Undirected, a both actor would need to be in the movie for the edge to be added
- **Course Prerequisites**

  Vertices: courses. **Edge**: from a to b if a is a prereq for b.

  Directed, since one course comes before the other
- **Ways to walk between UW buildings**

  Vertices: buildings. **Edges**: A street name or walkway that connects 2 buildings

  Undirected, since each route can be walked both ways

Graph Modeling

Often need to refine original model as you work through details of algorithm

SCENARIO
&
QUESTION TO ANSWER

Many ways to model any scenario with a graph, but question motivates which data is important

MODEL AS A GRAPH

- Choose vertices
- Choose edges
- Directed/Undirected
- Weighted/Unweighted
- Cyclic/Acyclic

…

RUN ALGORITHM

- Just visit every node?
    - BFS or DFS
- s-t Connectivity?
    - BFS or DFS
- Unweighted shortest path?
    - BFS
- Weighted shortest path?
    - Dijkstra's
- Minimum Spanning Tree?
    - Prim's or Kruskal's

ANSWER!

Graph Modeling Activity

Note Passing – Part I

Imagine you are an American High School student. You have a very important note to pass to your crush, but the two of you do not share a class so you need to rely on a chain of friends to pass the note along for you. A note can only be passed from one student to another when they share a class, meaning when two students have the same teacher during the same class period.

Unfortunately, the school administration is not as romantic as you, and passing notes is against the rules. If a teacher sees a note, they will take it and destroy it. Figure out if there is a sequence of handoffs to enable you to get your note to your crush.

How could you model this situation as a graph?

	Period 1	Period 2	Period 3	Period 4
You	Smith	Patel	Lee	Brown
Anika	Smith	Lee	Martinez	Brown
Bao	Brown	Patel	Martinez	Smith
Carla	Martinez	Jones	Brown	Smith
Dan	Lee	Lee	Brown	Patel
Crush	Martinez	Brown	Smith	Patel

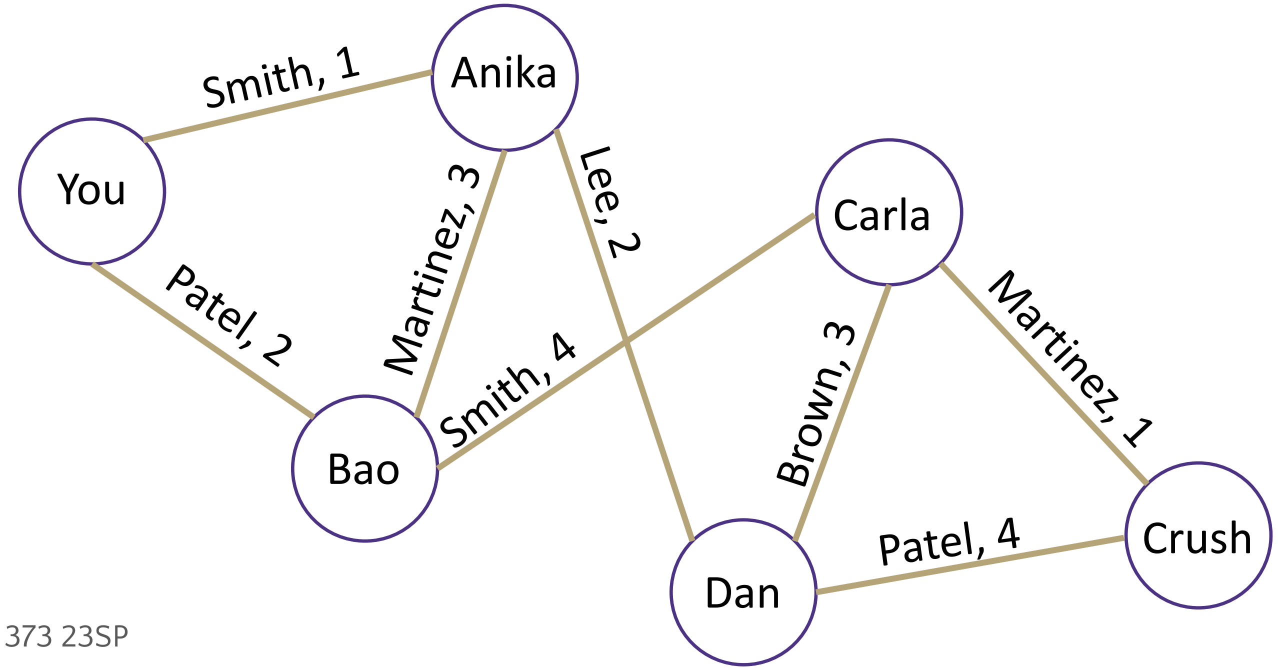
students teachers

Possible Design

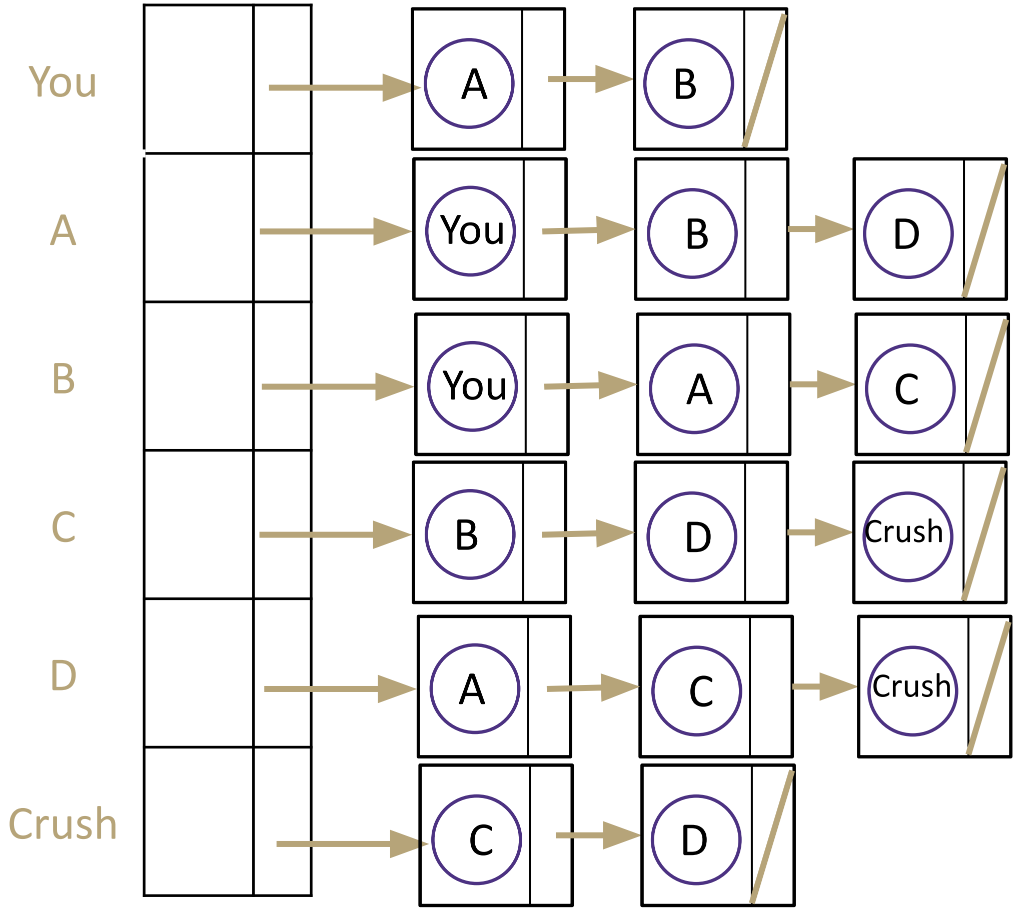
Algorithm

BFS or DFS to see if you and your Crush are connected

- **Vertices**
  - Students
  - Fields: Name, have note
- **Edges**
  - Classes shared by students
  - Not directed
  - Could be left without weights
  - Fields: vertex 1, vertex 2, teacher, period



Adjacency List



More Design

Note Passing – Part II

Now that you know there exists a way to get your note to your crush, we can work on picking the best hand off path possible.

Thought Experiments:

1. What if you want to optimize for time to get your crush the note as early in the day as possible?
   - How can we use our knowledge of which period students share to calculate for time knowing that period 1 is earliest in the day and period 4 is later in the day?
   - How can we account for the possibility that it might take more than a single school day to deliver the note?

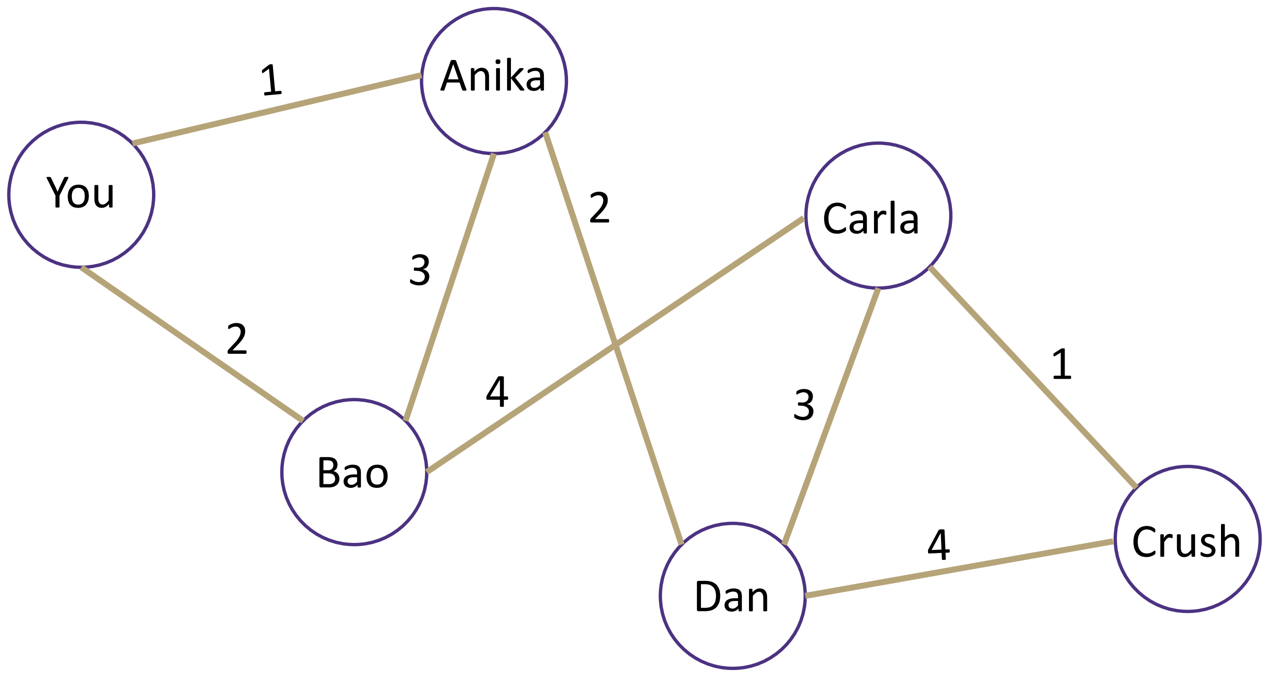
2. What if you want to optimize for rick avoidance to make sure your note only gets passed in classes least likely for it to get intercepted?

   – Some teachers are better at intercepting notes than others. The more notes a teacher has intercepted, the more likely it is they will take yours and it will never get to your crush. If we knew how many notes each teacher has intercepted how might we incorporate that into our graph to find the least risky route?

Optimize for Time

“Distance” will represent the sum of which periods the note is passed in, because smaller period values are earlier in the day the smaller the sum the earlier the note gets there except in the case of a “wrap around”

1. Add the period number to each edge as its weight
2. Run Dijkstra’s from You to Crush

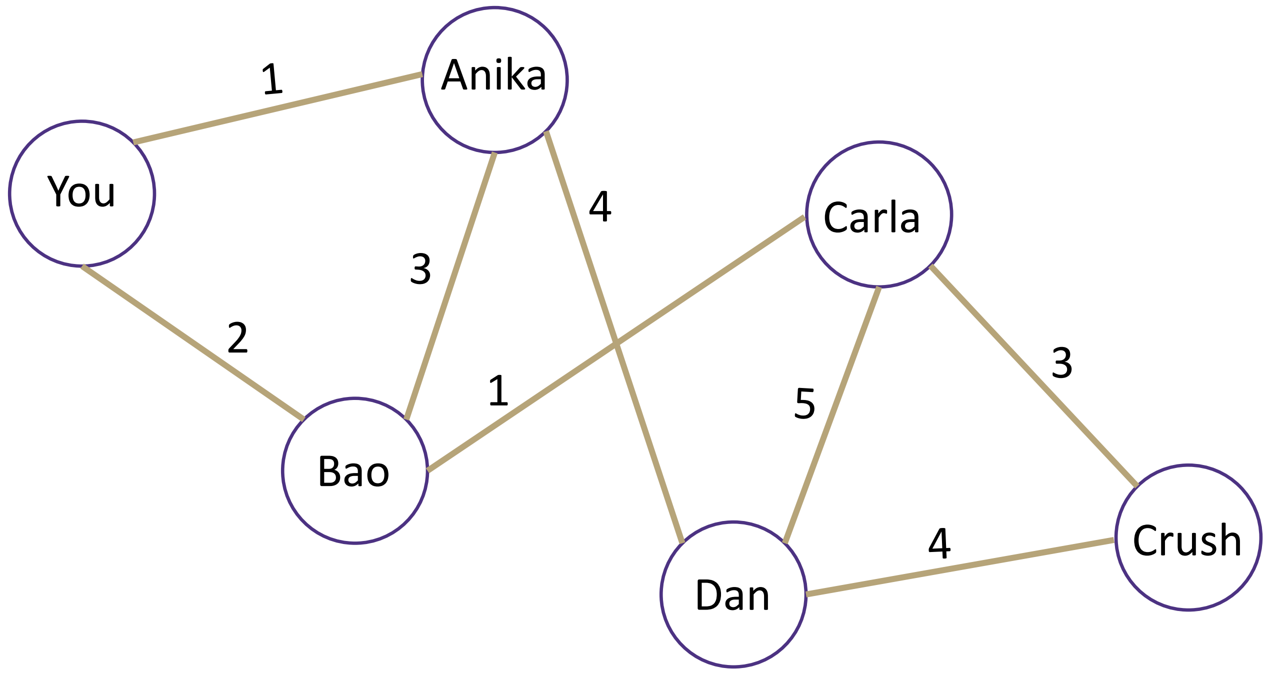


Vertex	Distance	Predecessor	Process Order
You	0	--	0
Anika	1	You	1
Bao	2	You	5
Carla	6	Dan	3
Dan	3	Anika	2
Crush	7	Carla	4*

*The path found wraps around to a new school day because the path moves from a later period to an earlier one
– We can change our algorithm to check for wrap arounds and try other routes

Optimize for Risk

"Distance" will represent the sum of notes intercepted across the teachers in your passing route. The smaller the sum of notes the "safer" the path.



1. Add the number of letters intercepted by the teacher to each edge as its weight
2. Run Dijkstra's from You to Crush

Teacher	Notes Intercepted
Smith	1
Martinez	3
Lee	4
Brown	5
Patel	2

Vertex	Distance	Predecessor	Process Order
You	0	--	0
Anika	1	You	1
Bao	4	Anika	2
Carla	5	Bao	3
Dan	10	Carla	5
Crush	8	Carla	4

Graph Modeling Activity

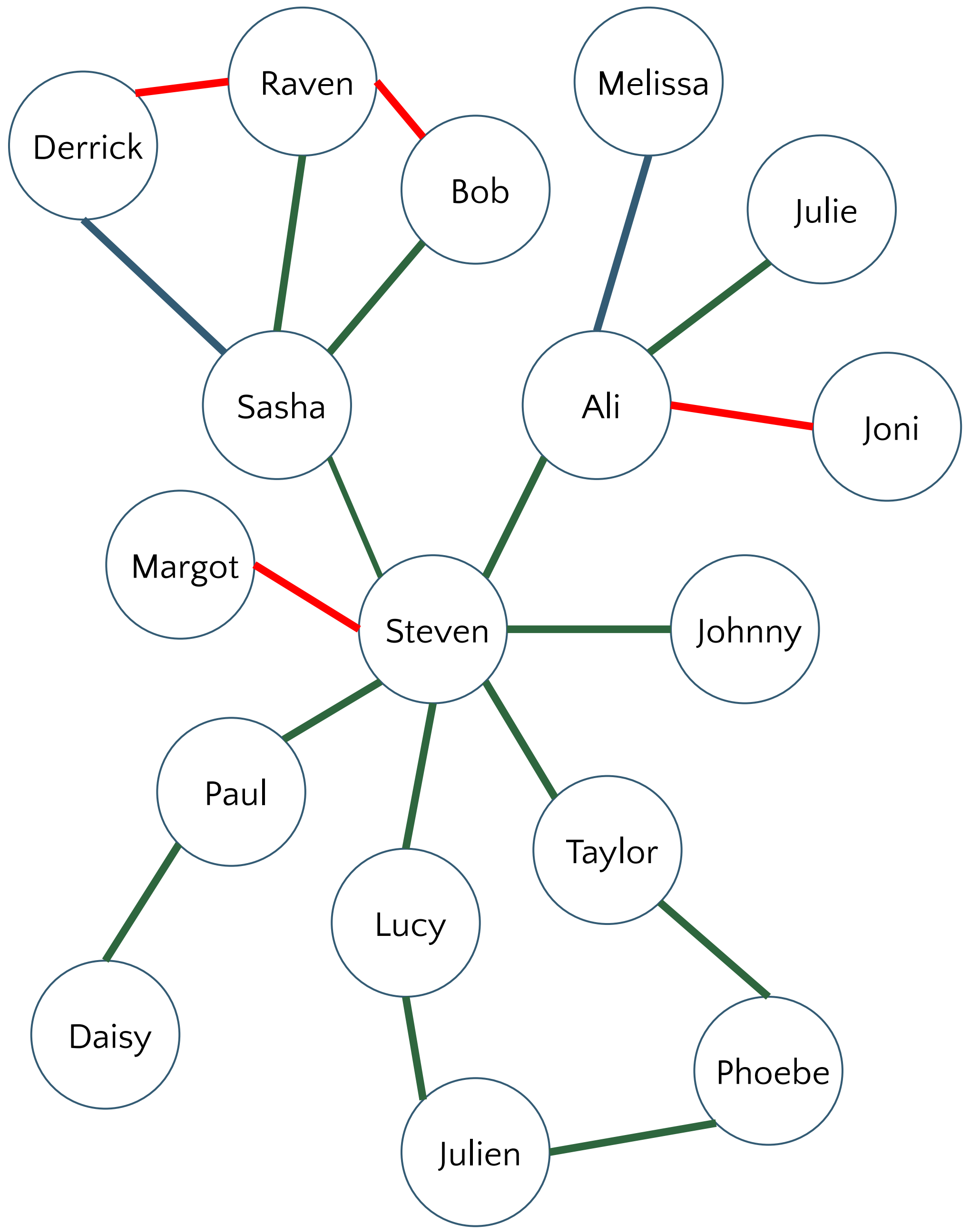
Party Planning

Imagine you are planning a party for your friend Steven. You are trying to figure out who you should invite from his big group of friends.

You want to invite as many people as possible, but you only want to invite either people he knows or friends of his friends, aka if they have at least one mutual friend.

You also don't want to invite two people if they dislike each other.

How could you model this situation as a graph?



Graph Modeling Activity

Movie Theater Marathon

Imagine (for some reason) your goal for the day is to watch as many movies as possible at your local movie theater.

Your rules are that you have to be there for the start of the trailers (aka when the listed movie time is) and can't leave a movie until it is completely finished (aka after the listed runtime).

How would you model this situation as a graph?

Questions?

That's all!