# Graphs

CS 61B Spring 2016: Discussion 11

## Announcements

We are grading your Project 2 (going slowly) and Midterm 2 (almost there).

Thoughts and initial reactions on Midterm 2?

A lot of you said you enjoyed Project 2 and thought it was a challenging but rewarding experience.

"This project was hugely beneficial. On the one hand it introduced me to independent problem solving (I am no longer afraid to search through documentation online) and on the other hand I feel like after finishing this project, nothing at Berkeley can stop me." - Anon.

## Announcements

Project 3 (BearMaps) is released and due Monday, 4/18.

This project is awesome. It incorporates many really cool concepts and applications used in the real world and serves as a pretty nice sneak preview to things you'll see in upper division courses.

The best part? You are fully equipped to tackle this project.

Homework 4 (Gitlet) is released and due Thursday, 4/7.

WE TOLD THEM
HW4 WAS GITLET
imgflip.com

## Announcements

Project 3 (BearMaps) is released and due Monday, 4/18.

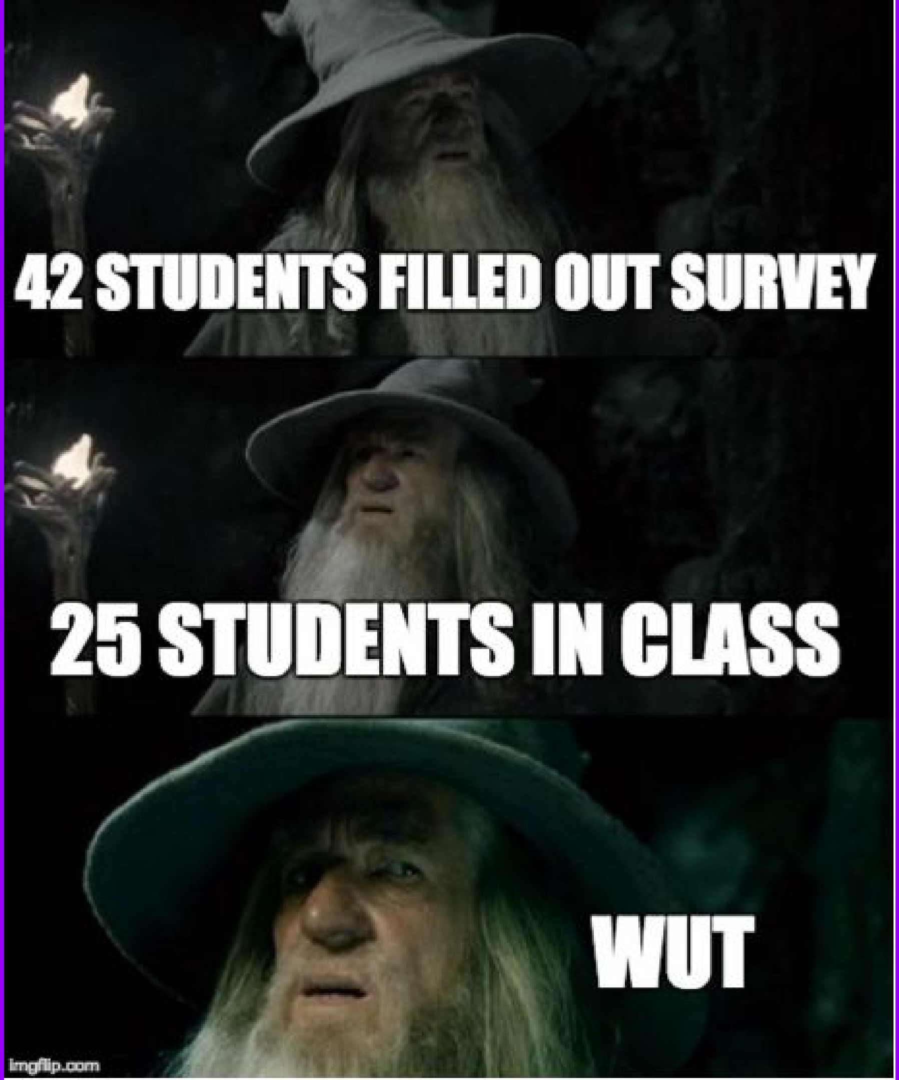
This project is awesome. It incorporates many really cool concepts and applications used in the real world and serves as a pretty nice sneak preview to things you'll see in upper division courses.

The best part? You are fully equipped to tackle this project.

Homework 4 (~~Gitlet~~ 8 Puzzle) is released and due Thursday, 4/7.

## Feedback from Spring Break Survey

42 people filled out the form and reported me as their primary TA.


42 STUDENTS FILLED OUT SURVEY
25 STUDENTS IN CLASS
WUT
imgflip.com

## Feedback from Spring Break Survey

How comfortable do you feel in section? 3.857 / 5

Most people find section to be helpful and enjoyable.

Most people like working in groups, but wish they knew people in section a bit better. Some don’t like working with others at all.

I’ll try some different group-work setups.

Everyone reported they were either keeping up with discussion very well, or still a bit confused at the end. That’s good. :)

## Feedback from Spring Break Survey

It’s pretty mixed between wanting more time to work individually/in groups and wanting to spend more time going over the solution, though there are slightly more votes for the latter.

14 people said they want to cover everything, even if it means in less depth.

28 people said they want to go more in depth, even if it means not covering everything.

## Feedback from Spring Break Survey

Things you like about discussion:

“I'm already pretty satisfied but I think more memes would be cool. Love the tips every week as well.”
“I like the random tips and I like the overall structure.”
“the lame jokes are funny... The random tips are helpful too, I feel like without them I wouldn't really know how to find out about those kinds of shortcuts. Discussion always needs more memes”
“I like the overall structure of the discussion, and those random tips are really useful. PLEASE KEEP THE MEMES”
“I like the overall structure, the random tip of the week, and the joke images. However, the lack of memes in discussion is appalling :/”

My conclusion… you guys are basically this pepe

Please let me know when the memes become too much

## Feedback from Spring Break Survey

Discussion needs to be slightly better paced. I’ll work on this.

Discussion could benefit from more visuals.

Lab is chill.

Lab could use one lab assistant so that a major question from one student doesn’t bottleneck the entire lab.

## Feedback from Spring Break Survey

Thanks for the movie/song/anime/game suggestions! I’ll look into them when I have time this summer. :)

# Questions for me

Q. How do I get an internship at Google ASAP?

A. Apply online, go to the career fairs and tech talks. Goes for every company.

Q. What's your spirit dank meme?

A. This is too hard; it changes every week. This week, it's the "don't trust anybody" meme. Check it out offline. Probably inappropriate to show in class.

Q. WANT TO PLAY LEAGUE???

A. Sure; ask me after the semester is over. :)

# Questions for me

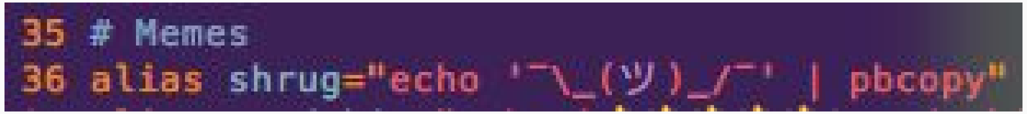
Q. I started playing Neko Atsume because you put it on your discussion slides one time WHY WOULD YOU DO THIS TO ME

A. Whoops. :^)

Q. Why is there random chinese on the survey?

A. ¯_(ツ)_/¯

This is an excerpt of my .bashrc:



```
# Memes
alias shrug="echo '¯\_(ツ)_/¯' | pbcopy"
```

Q. ( ͡° ͜ʖ ͡°)

A. Same.

## Questions for me

Q. Do you have advice for getting an internship?

A. Generic advice: don’t give up, it’s pretty hard as a freshman, don’t be afraid to work for a startup or for little pay; it’s about getting the work experience; go to all of the tech talks and infosessions held on campus (HKN does a lot of these). Ask me offline if you’d like to discuss further!

Q. league???? :3

A. See last answer.

Q. What's your favorite movie?? Superhero? Team Iron Man or Team Cap?

A. Uhh, movie is too hard. Deadpool was pretty great though. I dunno... Ironman?

## Graphs

## Graphs

Graphs are a really useful data structure that can be used to model many real-world problems.

Example uses of graphs:

- Finding shortest routes in car navigation systems (or in Project 3!)
- Search engines use ranking algorithms based on graph theory
- Optimizing time tables for schools or universities
- Analysis of social networks (Think Facebook or LinkedIn)
- Optimizing utilization of railway systems
- Compilers use coloring algorithms to assign registers to variables (see CS 61C)
- Path planning in robotics
- Solving games and puzzles (like the 8Puzzle you're doing for HW4!)

## Graph Terminology

- Graph:
  - Set of ***vertices***, a.k.a. ***nodes***.
  - Set of ***edges***: Pairs of vertices.
  - Vertices with an edge between are ***adjacent***.
  - Optional: Vertices or edges may have ***labels*** (or ***weights***).
- A ***path*** is a sequence of vertices connected by edges.
- A ***cycle*** is a path whose first and last vertices are the same.
  - A graph with a cycle is 'cyclic'.
- Two vertices are ***connected*** if there is a path between them. If all vertices are connected, we say the graph is connected.

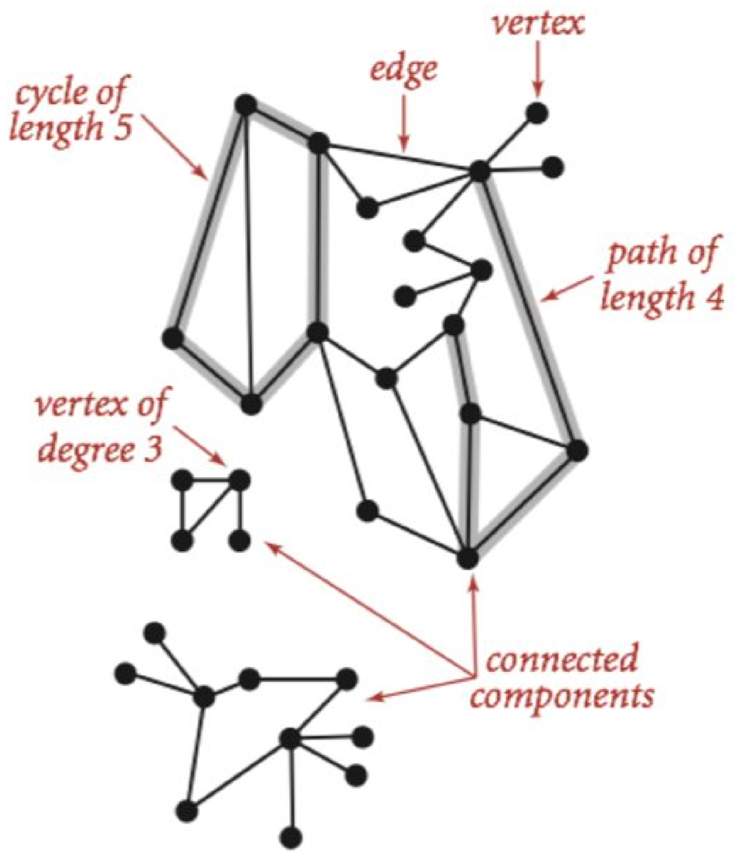


Figure from Algorithms 4th Edition

## Some Graph-Processing Problems

Interested in Graph Theory problems?

Definitely check out CS170.

**s-t Path**. Is there a path between vertices s and t?

**Shortest s-t Path.** What is the shortest path between vertices s and t?

**Cycle.** Does the graph contain any cycles?

**Euler Tour.** Is there a cycle that uses every edge exactly once?

**Hamilton Tour.** Is there a cycle that uses every vertex exactly once?

**Connectivity.** Is the graph connected, i.e. is there a path between all vertex pairs?

**Biconnectivity.** Is there a vertex whose removal disconnects the graph?

**Planarity**. Can you draw the graph on a piece of paper with no crossing edges?

**Isomorphism**. Are two graphs isomorphic (the same graph in disguise)?

Graph problems: Unobvious which are easy, hard, or computationally intractable.

## Graph API

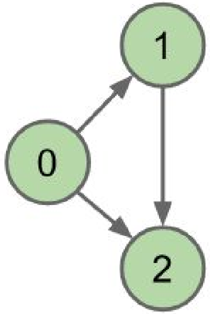
Using a graph in Java:

```
public class Graph {
  public Graph(int V):                     Create empty graph with v vertices
  public void addEdge(int v, int w): add an edge v-w
  Iterable<Integer> adj(int v):            vertices adjacent to v
  int V():                                 number of vertices
  int E():                                 number of edges
...
```

## Graph Representations

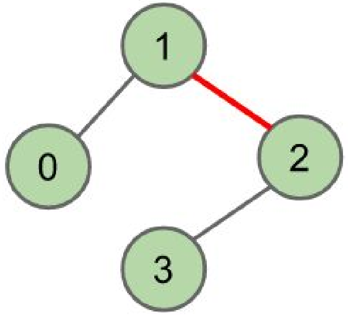
- Representation 1: Adjacency Matrix.

| s \ t | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 |
| 2 | 0 | 0 | 0 |



For undirected graph: Each edge is represented twice in the matrix. Simplicity at the expense of space.

| v \ w | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | **1** | 0 |
| 2 | 0 | **1** | 0 | 1 |
| 3 | 0 | 0 | 1 | 0 |

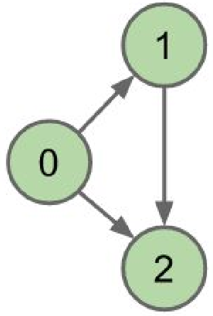


## More Graph Representations

Representation 2: Edge Sets: Collection of all edges.

- Example: HashSet<Edge>, where each Edge is a pair of ints.

```
{(0, 1), (0, 2), (1, 2)}
```

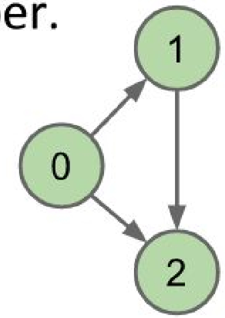


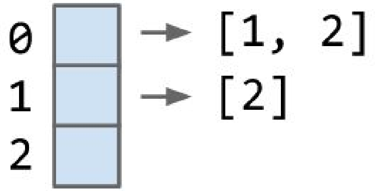
We can imagine an Edge object has two Integer instance variables that represent vertices. They could also be arrays of 2 elements, etc...

## More Graph Representations

Representation 3: Adjacency lists.

- Common approach: Maintain array of lists indexed by vertex number.
- Most popular approach for representing graphs.





For a directed graph, store the vertices that each vertex has an outgoing edge towards.

## Graph Representations

Runtime of some basic operations for each representation:

| idea | addEdge(s, t) | adj(s) | hasEdge(s, t) | space used |
|---|---|---|---|---|
| adjacency matrix | Θ(1) | Θ(V) | Θ(1) | $\Theta(V^2)$ |
| list of edges | Θ(1) | Θ(E) | Θ(E) | Θ(E) |
| adjacency list | Θ(1) | Θ(1) | Θ(degree(v)) | Θ(E+V) |

Note: Our graph implementation doesn't have this operation.

In practice, adjacency lists are most common.

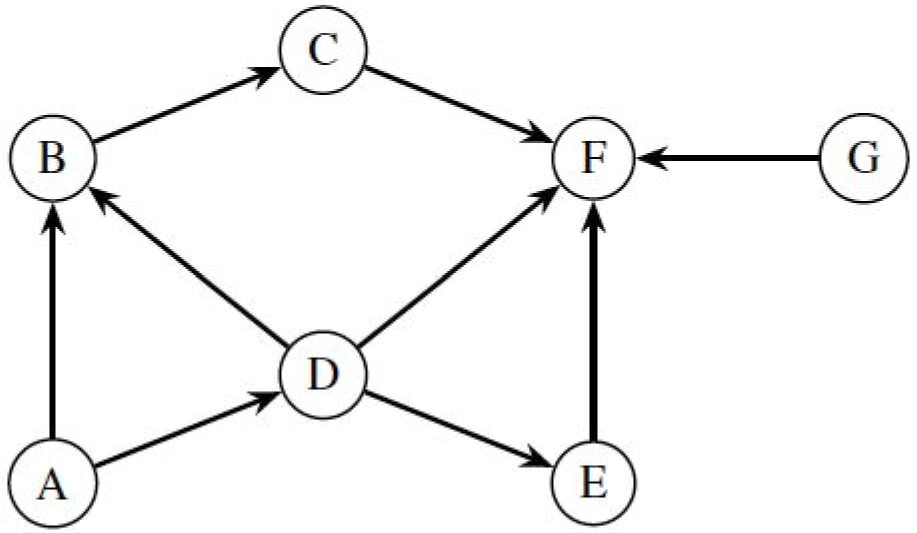
- Many graph algorithms rely heavily on `adj(s)`.
- Most graphs are sparse (not many edges in each bucket).

# Practice with Graph Representations

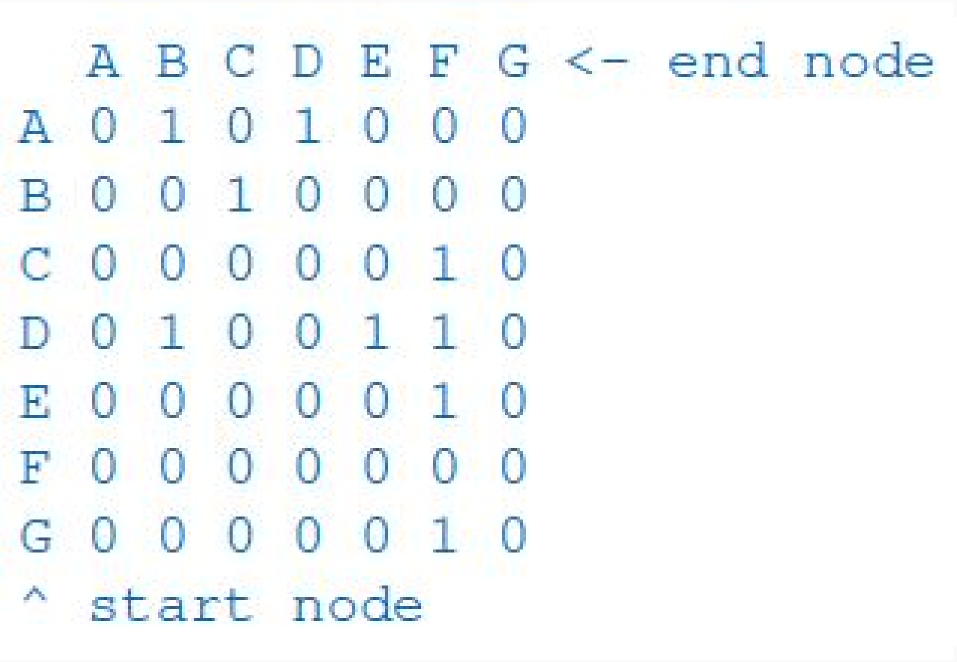
Problem 1

## Problem 1: Adjacency Matrix

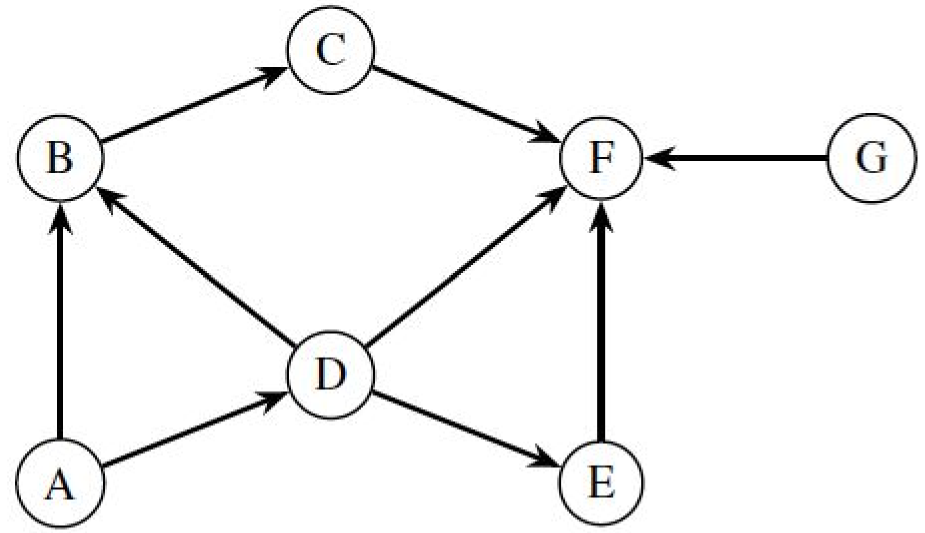
```
  A B C D E F G <- end node
A
B
C
D
E
F
G
^ start node
```



## Problem 1: Adjacency Matrix



```
  A B C D E F G <- end node
A 0 1 0 1 0 0 0
B 0 0 1 0 0 0 0
C 0 0 0 0 0 1 0
D 0 1 0 0 1 1 0
E 0 0 0 0 0 1 0
F 0 0 0 0 0 0 0
G 0 0 0 0 0 1 0
^ start node
```



## Problem 1: Adjacency List

A:

B:

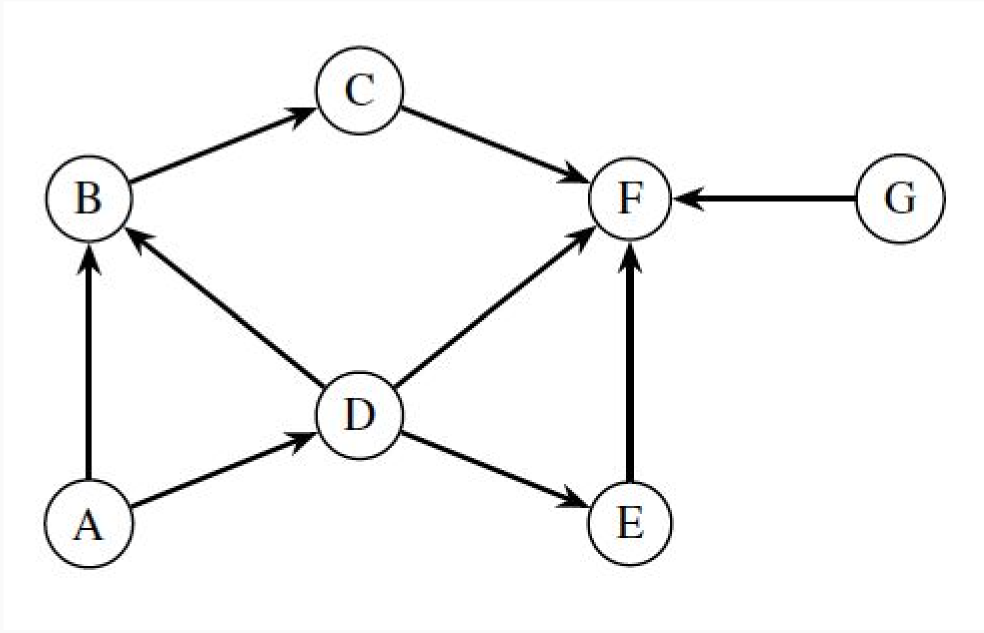
C:

D:

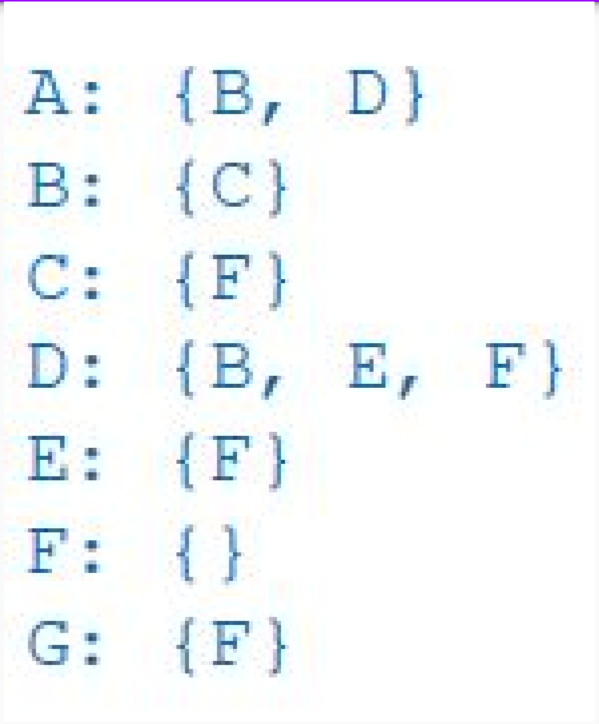
E:

F:

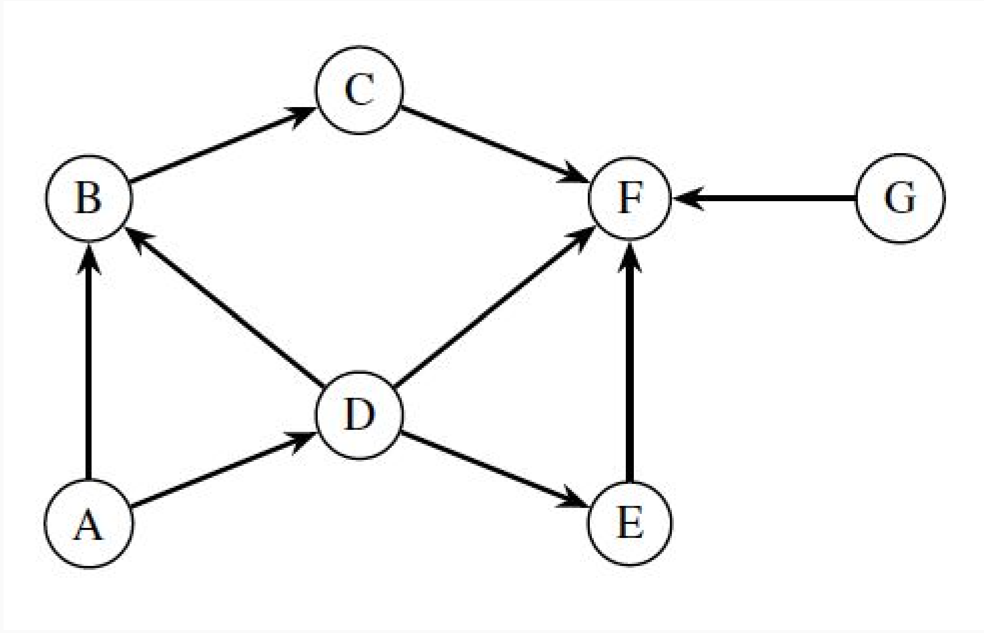
G:



## Problem 1: Adjacency List



```
A:  {B, D}
B:  {C}
C:  {F}
D:  {B, E, F}
E:  {F}
F:  {}
G:  {F}
```

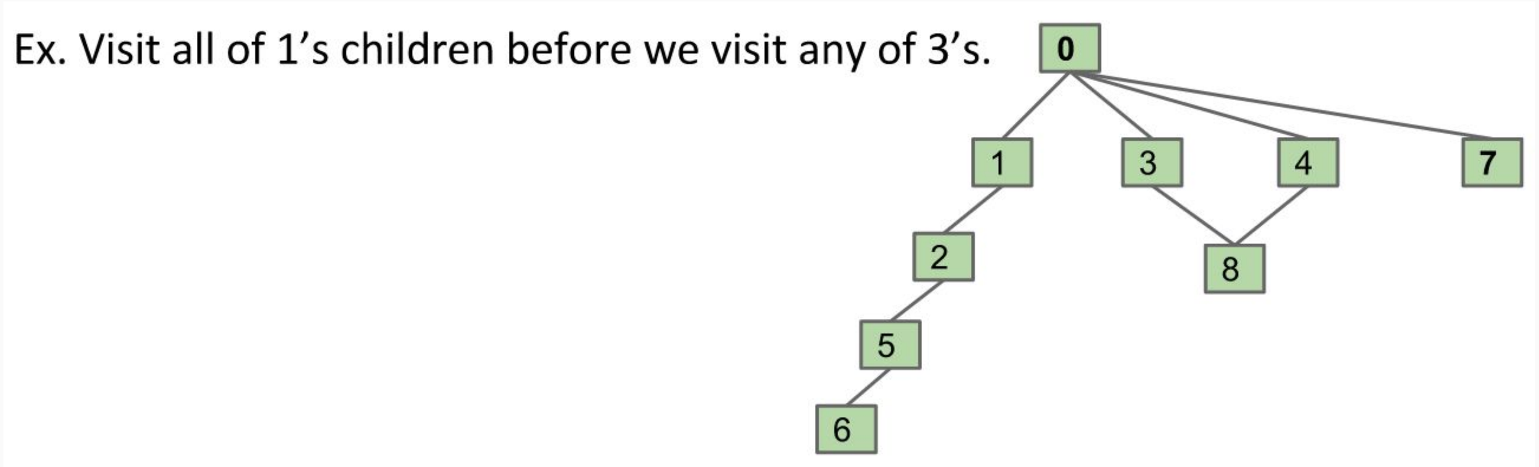


## Graph Traversals

## Depth-First Search (DFS)

One way of traversing a graph is with a depth-first search (DFS).

In DFS, we explore the entire subgraph of each child as we find them.


Ex. Visit all of 1’s children before we visit any of 3’s.

## Depth-First Search (DFS)

Many ways to do DFS:

- Preorder: same order as the DFS calls
- Postorder: same order as the returns from the DFS calls
- Level-order and inorder are also ways to traverse graphs

A DFS, regardless of the ordering of the traversal, is:

- Guaranteed to reach every node
- Runs in O(V+E) time and takes O(V) space

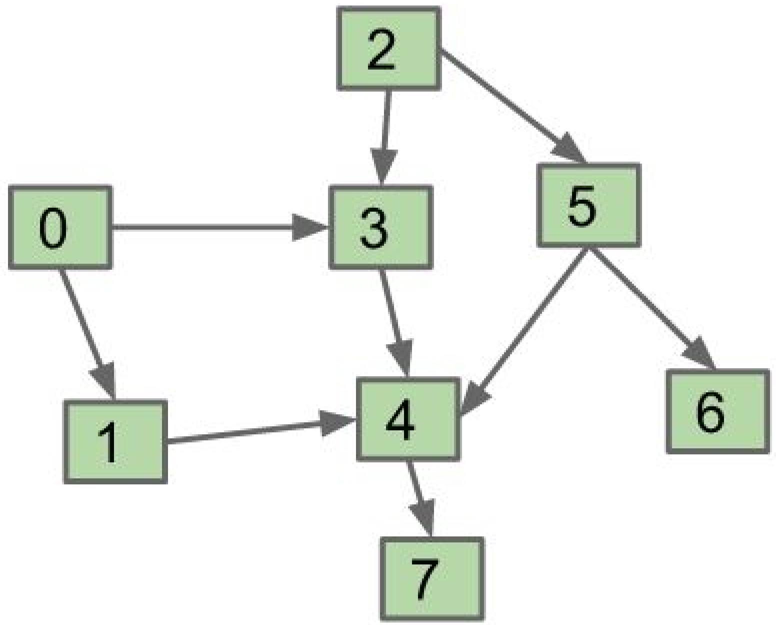
## Breadth-First Search (BFS)

A breadth-first search will visit vertices in level order.

It examines vertices in increasing order from the source vertex.

Like DFS, it runs in O(V+E) time and takes O(V) space.

## Breadth-First Search (BFS)



In this graph, a BFS from vertex 0 would visit vertices 0, followed by 1 and 3, before visiting any other vertices.

The complete BFS order is:

0 -> 1 -> 3 -> 4 -> 7

Notice that since we started at vertex 0, we won’t see vertices 2, 5, and 6.

## Searching… for something

Problem 2

## Graph Traversal Reference

Postorder: the order of returns from DFS calls

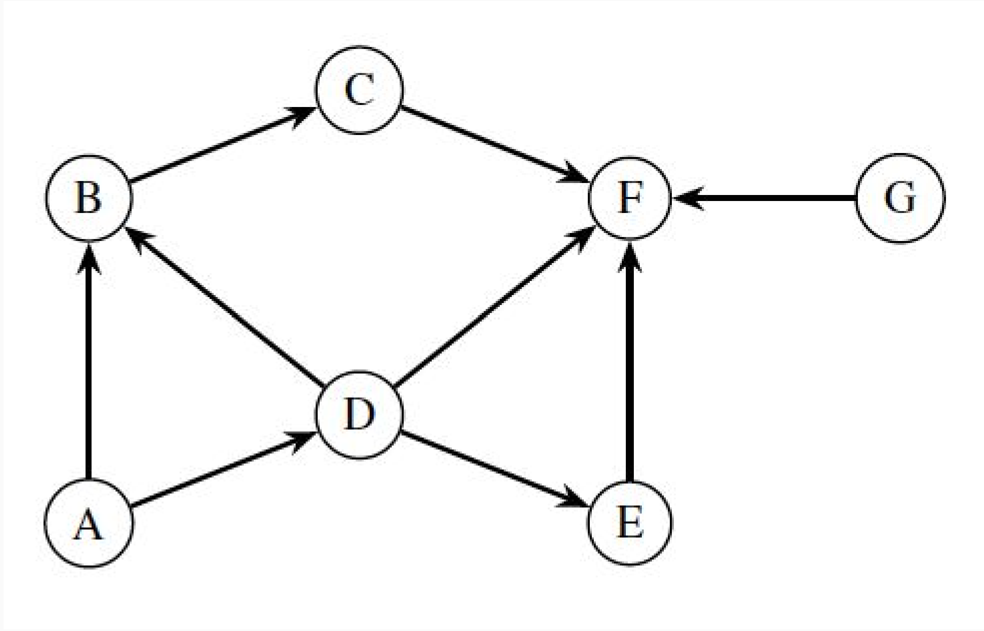
Preorder: the order in which the DFS calls are made

BFS: level order

## Problem 2: DFS Preorder

DFS preorder (starting at A):

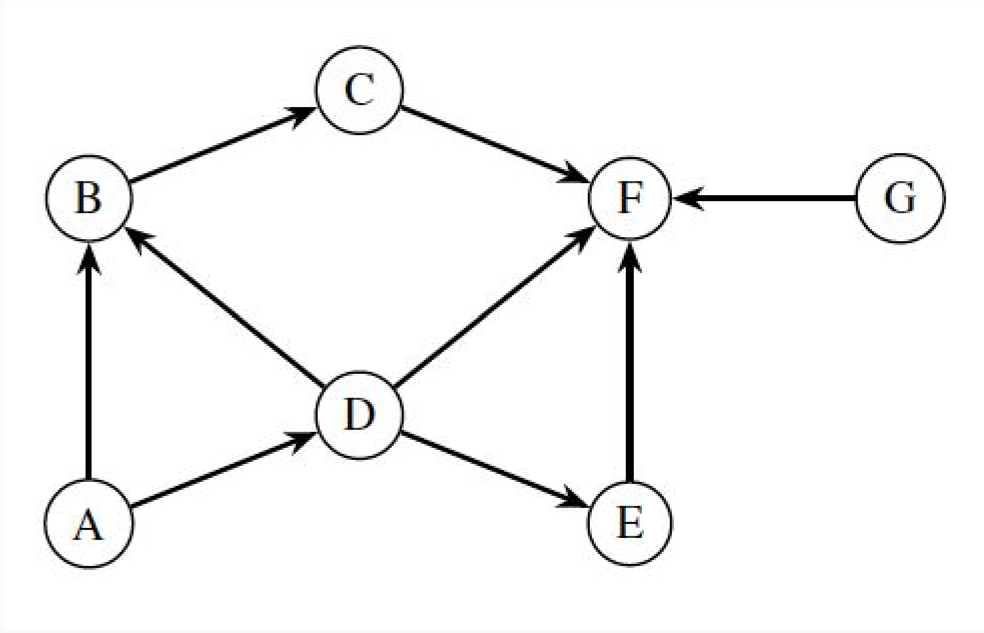
A -> B -> C -> F -> D -> E



## Problem 2: DFS Postorder

DFS postorder (starting at A):

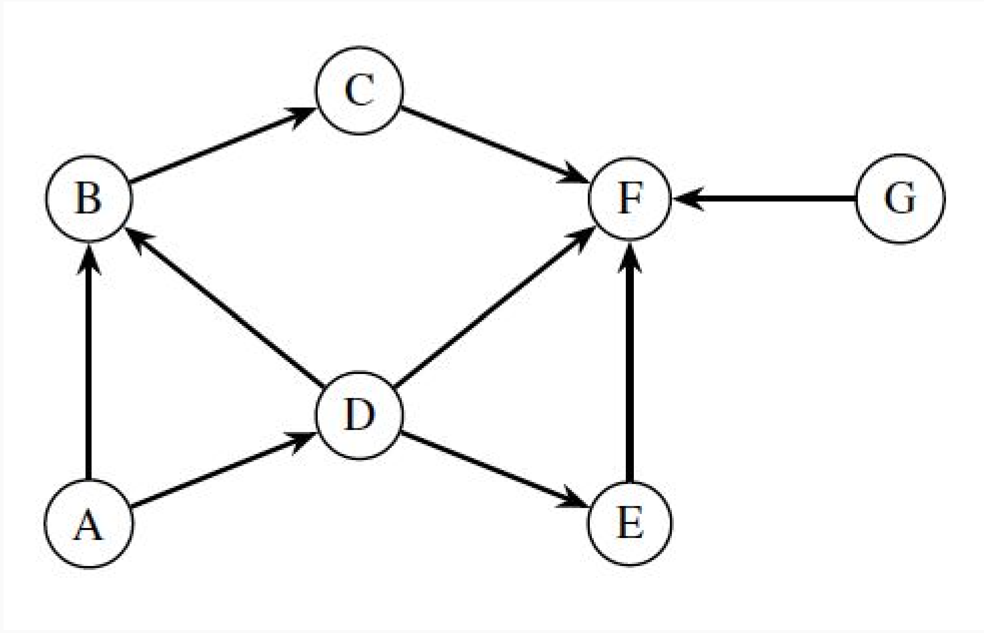
F -> C -> B -> E -> D -> A



## Problem 2: BFS

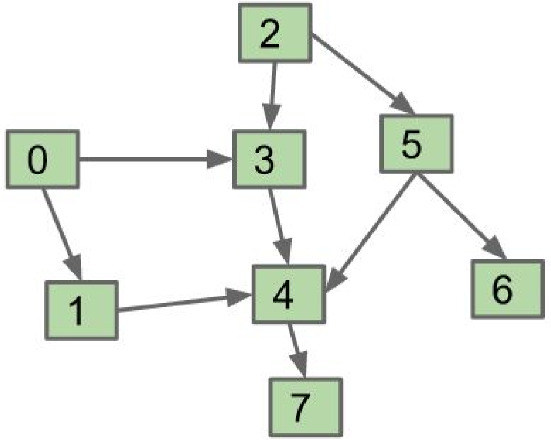
BFS (starting at A):

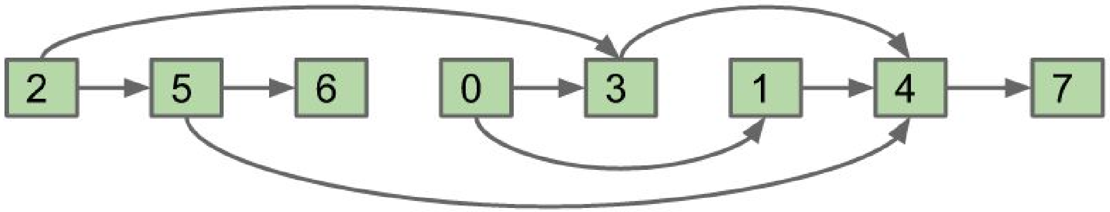
A -> B -> D -> C -> E -> F



## Topological Sort

A topological sort of a graph is an ordering of the vertices such that they appear in an order consistent with edges.





## Topological Sort

To get a valid topological ordering:

1. Perform DFS traversal from every vertex with indegree 0. Do not clear markings between traversals.
2. Record DFS postorder in a list.
3. Topological order is the reverse of the postorder list.

# Topological Sort

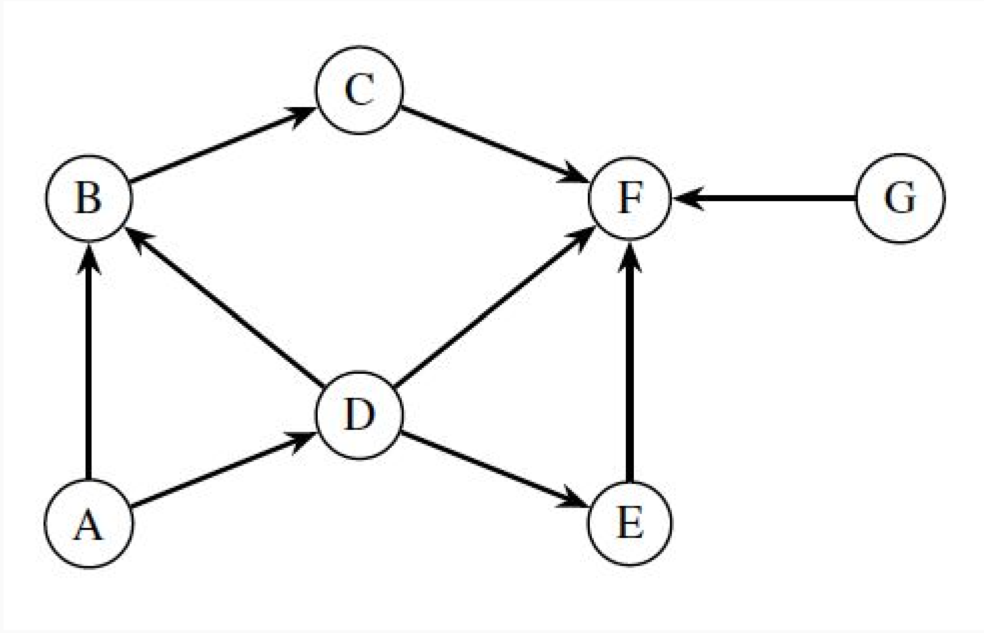
Problem 3

## Problem 3

One valid topological order is:

G -> A -> D -> E -> B -> C -> F

Note: G can go anywhere in the order, since it has indegree 0.



# Bipartite Graphs

Problem 4

## Problem 4

To determine whether a graph is bipartite, we perform a special version of DFS or BFS from any vertex.

Mark start vertex with a label U. For each of its children, mark them with a label V. For each of those children, mark them with a label U, and so on. If at any point, a child is labelled already but has a conflicting label, then we return False.

If graph is not fully connected, repeat for each connected component.

If algorithm completes without returning False early, then it is bipartite.

## Recap: Graphs

Graphs are a really useful data structure that can be used to model many real-world problems.

Unfortunately, they can be pretty complicated to grasp at first. Things to review/study:

- Graph terminology (vertex, edge, cycle, path, connected, etc.)
- Pseudocode/implementation of graphs (adjacency matrix, adjacency list, set of edges)
- Pseudocode/implementation of graph traversals (BFS/DFS)
- Know and understand the runtime and space efficiency of graph implementations and graph traversals (BFS/DFS) in postorder, preorder, inorder, level-order.
- Practice doing graph traversals on arbitrary graphs
- Practice doing topological sorts on graphs (runtime/space?)
- Be familiar with different problems we can solve with graphs (e.g. s-t paths, shortest paths, cycles, isomorphism, etc.)

# Thanks for coming!

`What color should I use next week?`