Slides by Kevin Miao (kevinmiao@berkeley.edu)

CS61BL – Tutoring Section 10

Graphs (DFS, BFS, Dijkstra)

- Review (5 min)
- Quiz Review (Optional)
- Worksheet (20 min)
- Questions (5 min)

Friendly reminder: Please participate in order to get marked as 'present'

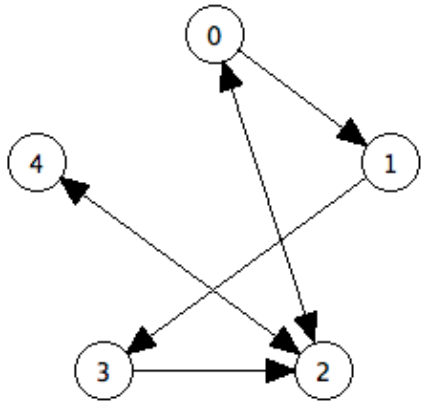
Resources:

- www.cs61bl.org/su20/resources

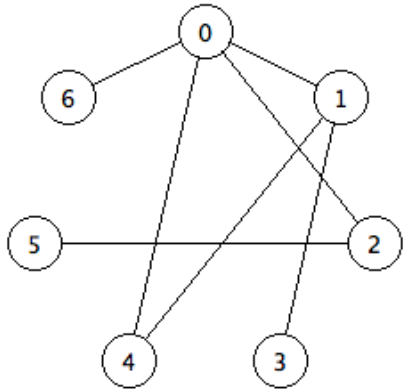
What the Graph?!

- **Objective:** Graphs are extremely powerful and generalizable abstractions of reality (i.e. maps, language, weather)
- **Variants:**

Directed Graphs: Vertices are connected by “one-way” edges



Undirected Graphs: Vertices are connected by “two-way” edges



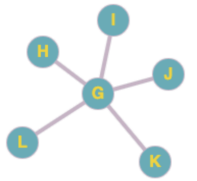
What the Graph?!

Terminology

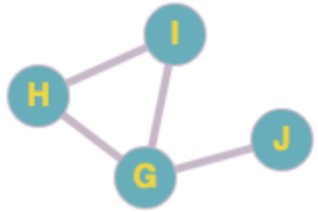
A is **adjacent** to B

F is a **neighbor of** A

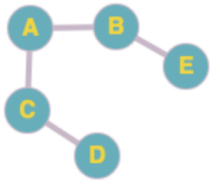
G's **indegree** is 5



[(H,I), (I,G), (G,H)] is a **cycle**

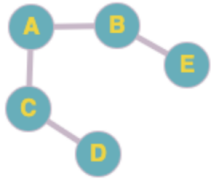


Graph is **connected**



(A,F) is **incident on** A and F

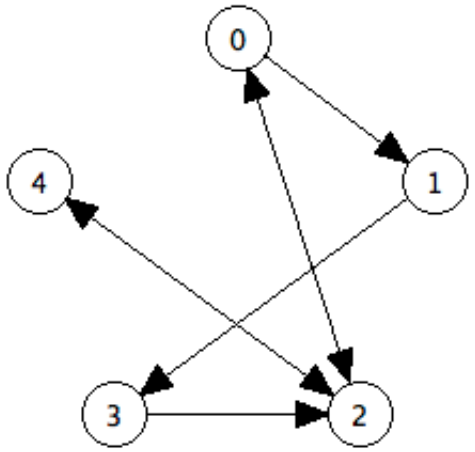
[(A,C), (C,D)] is a **path**



What the Graph?!

- **Representation of** $G = (V, E)$

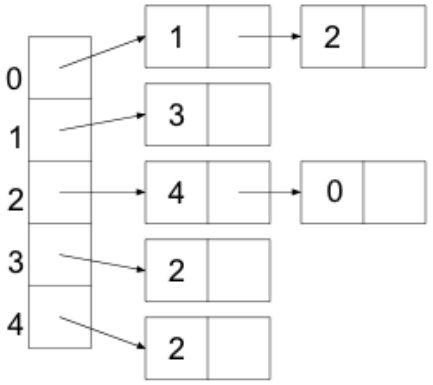
Graph



Adjacency List

Edge (u,v) exists? array[u].contains(v)

Memory, **Runtime**



Adjacency Matri

Edge (u,v) exists? array[u][v]

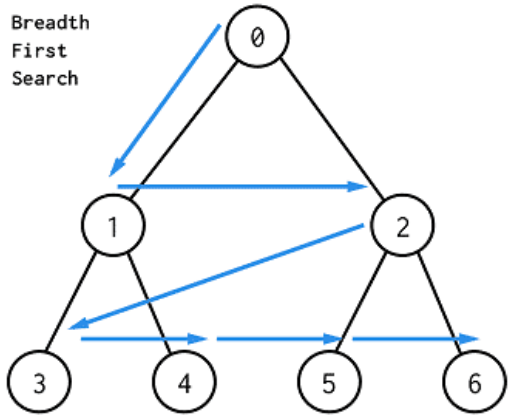
Memory, **Runtime**

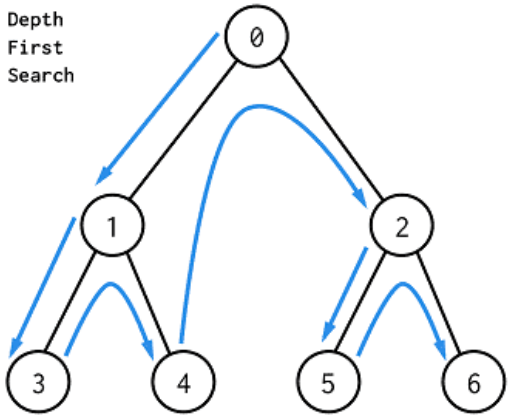
	0	1	2	3	4
0	F	T	T	F	F
1	F	F	F	T	F
2	T	F	F	F	T
3	F	F	T	F	F
4	F	F	T	F	F

Graph Traversals

- **Depth-First Search (DFS)**
  - **Idea:** Explore the deepest nodes first
  - **Fringe:** Stack
  - **Time complexity:** $\boldsymbol{O}(|\boldsymbol{V}| + |\boldsymbol{E}|)$

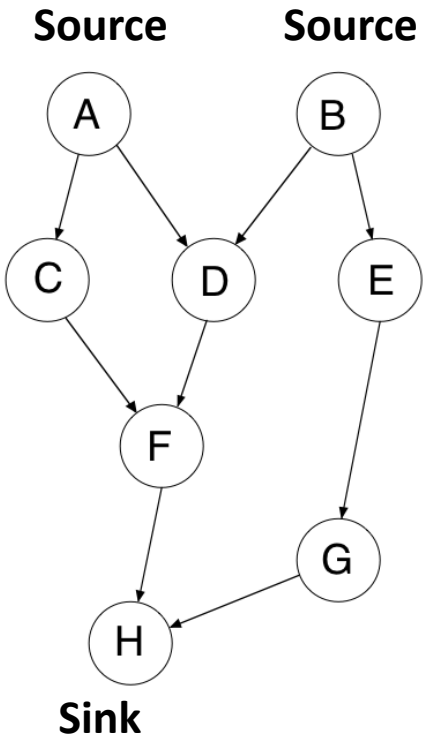
- **Breadth-First Search (BFS)**
  - **Idea:** Explore graph level by level
  - **Fringe:** Queue
  - **Time complexity:** $\boldsymbol{O}(|\boldsymbol{V}| + |\boldsymbol{E}|)$





Topological Sort

- **Linearization of graphs** (Where all arrows point one direction)
- **Only works on *directed, acyclic graphs (DAGs)***



A B C D E F G H

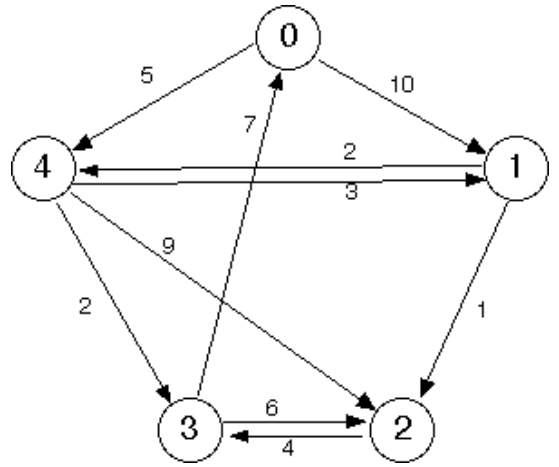
A C B E G D F H

B E G A D C F H

Shortest Path: Dijkstra

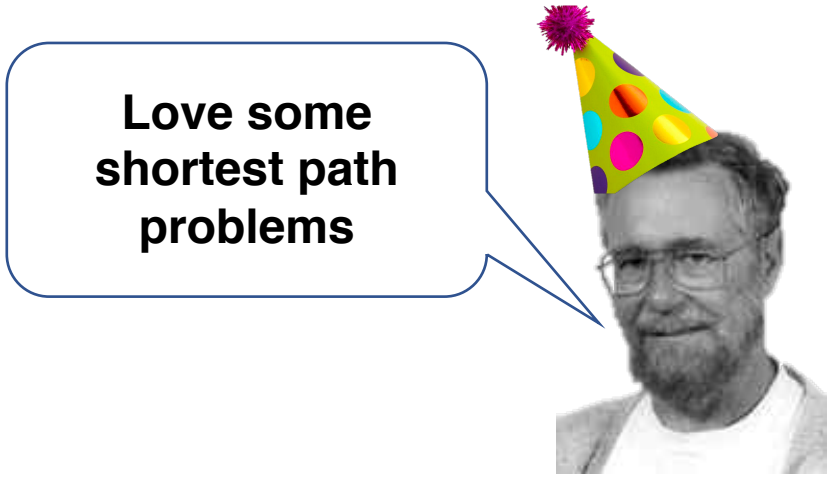
Goal: Find the shortest path from vertex *v* to vertex *u*.
Idea: Finding shortest path starting from a vertex *v*.
Runtime:

- $\boldsymbol{O(|V + E|log(V)}$ generally
- $\boldsymbol{O(|E|log(V)}$ for **<u>connected</u>** graphs

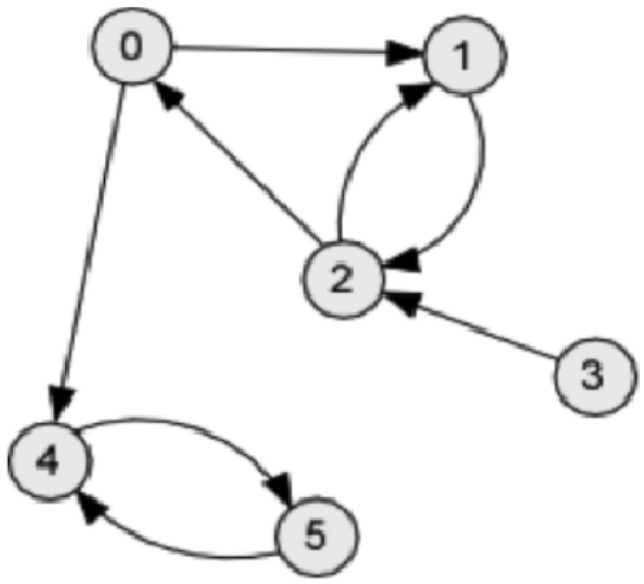


Implementation:

0. Add start vertex *v* to PriorityQueue with distance 0
1. Remove some vertex *y* from PriorityQueue
2. Iterate over neighbors of *y*
   1. ***Terminate if y.equals(u)***
   2. ***Add to PQ if not added already***
   3. ***Update weight in PQ***
3. Repeat 1&2 until PQ empty or terminated



Quiz Q1: Graphs



Adjacency List for node # ...

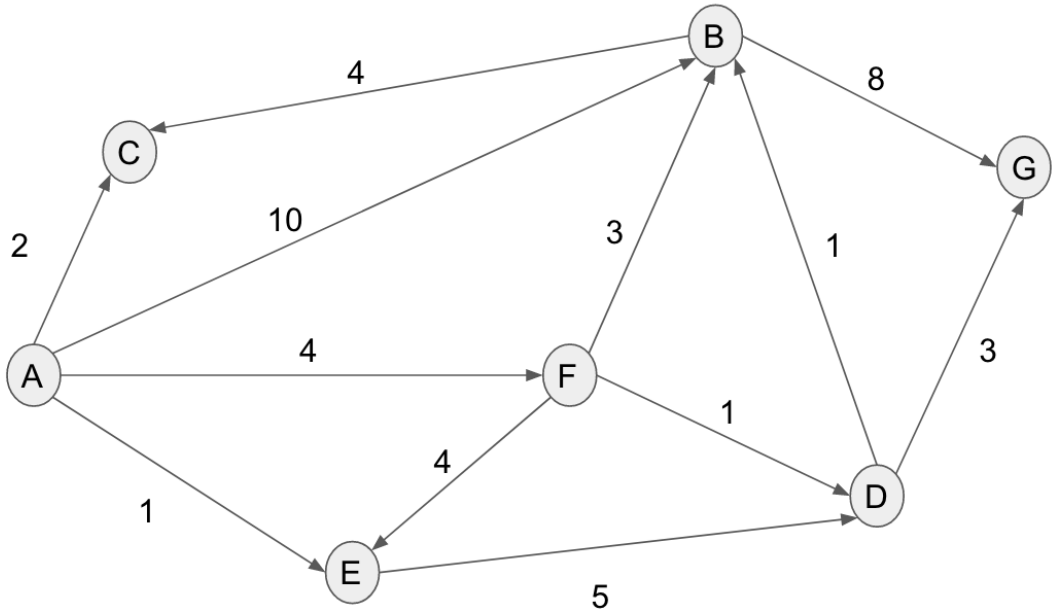
1

2

3

4

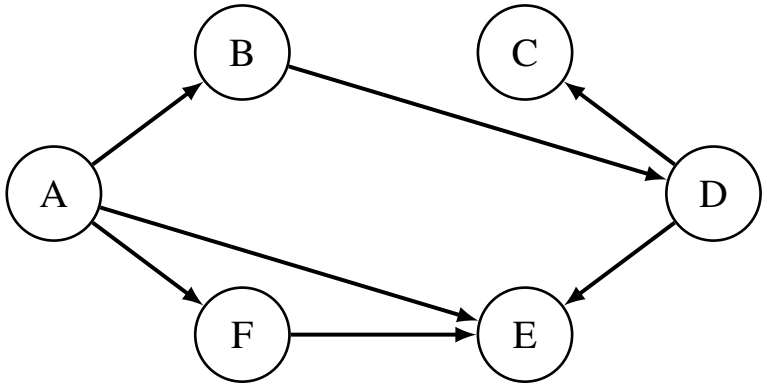
Quiz Q2: Dijkstra



Starting Vertex = A

DistTo

EdgeTo

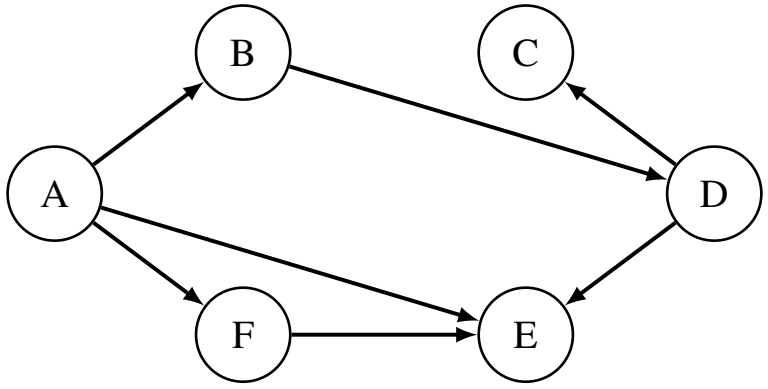


1 Graph Representation

Represent the graph above with an adjacency list and an adjacency matrix representation.

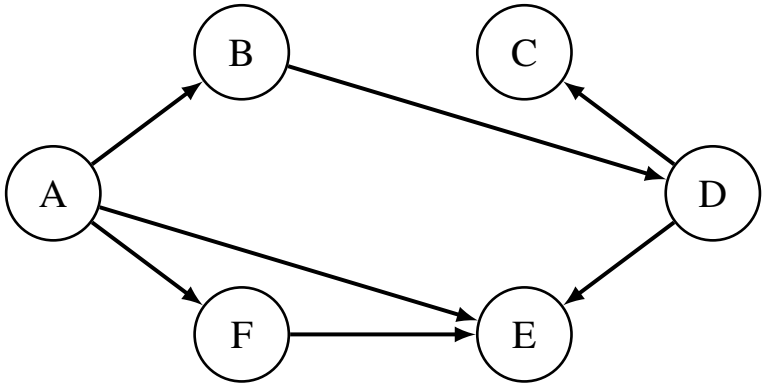
2 Searches and Traversals

Run depth first search (DFS) preorder and breadth first search (BFS) on the graph above, starting from node A. List the order in which each node is first visited. Whenever there is a choice of which node to visit next, visit nodes in alphabetical order.



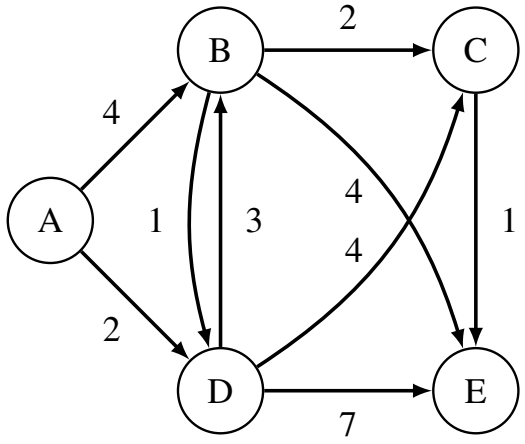
3 Topological Sorting

Give a valid topological ordering of the graph. Is it unique?



4 Dijkstra's Algorithm

(a) Given the following graph, run Dijkstra's algorithm starting at node *A*. For each iteration, write down the entire state of the algorithm. This includes the value `dist(`*v*`)` for all vertices *v*. Keep track of the vertices traversed along the shortest paths from A to every other node in the graph. You will need to maintain an `edgeTo` array.



(b) What must be true about our graph in order to guarantee Dijkstra's will return the shortest path's tree to every vertex? Draw an example of a graph that demonstrates why Dijkstra's might fail if we do not satisfy this condition.