Lecture 7: Strongly Connected Components & Shortest Paths in a Graph

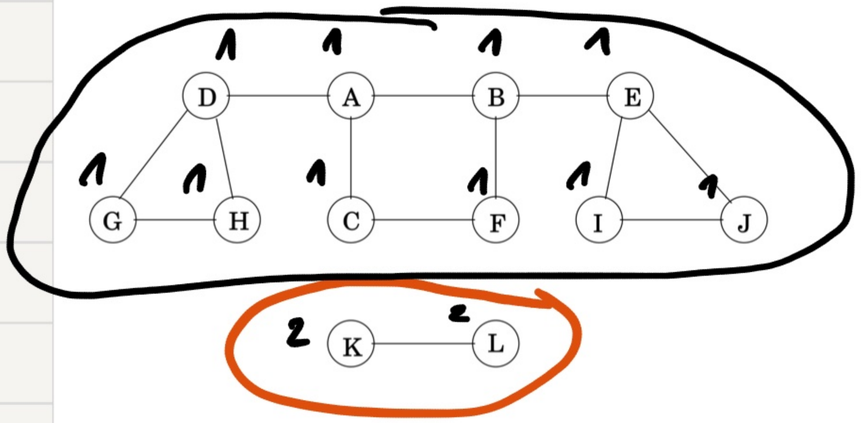
Last Time:
- DFS
  - connected components in undirected graphs.
  - Pre & Post numbers. Back / forward / cross edges.
  - Topological sort.

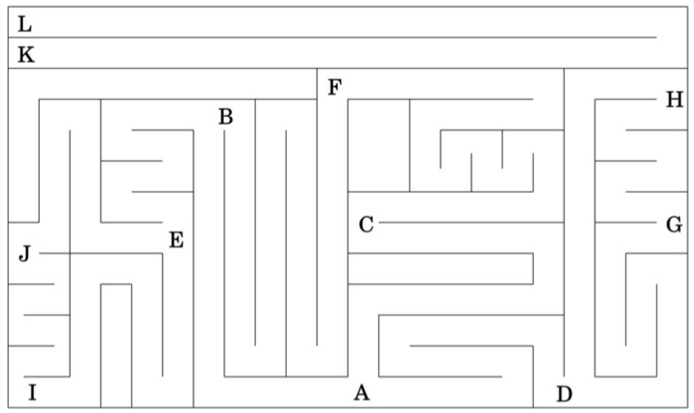
Today:
- Strongly Connected Components.
- BFS = Breadth First Search

(If time permits)
- Dijkstra's Algorithm

Recap: Connected Components in undirected graphs

Figure 3.2 Exploring a graph is rather like navigating a maze.





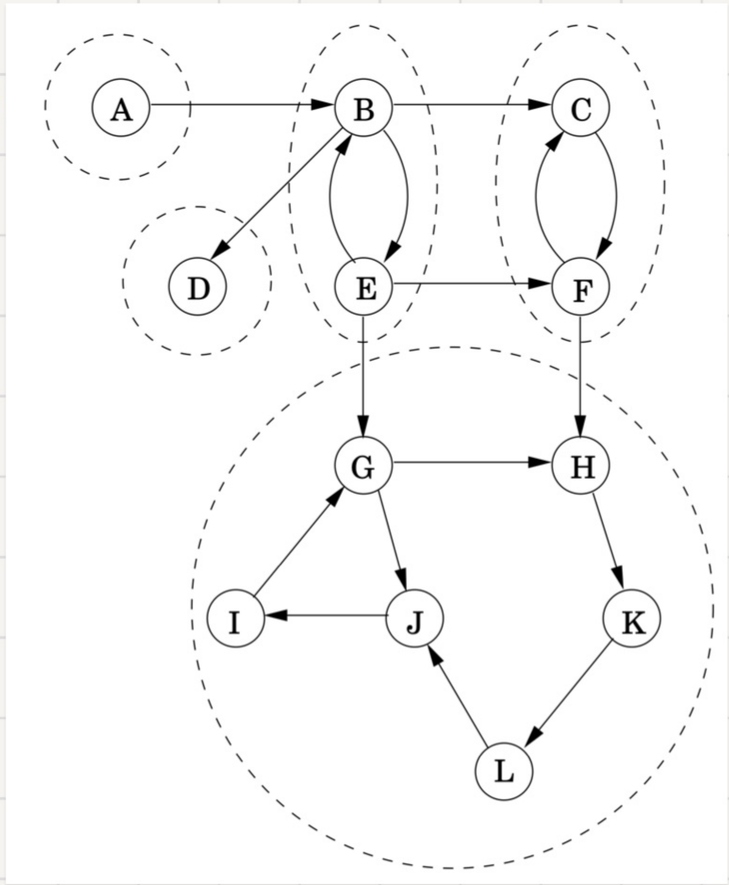
What about directed graphs? (digraphs)

- What's the right analog of CC in the directed case?
- How can we compute these components?

Connedivity for Directed Graphs

Defn: We say that u, v are <u>strongly connected</u> if there is a path from u to v, and a path from v to u.

Example:



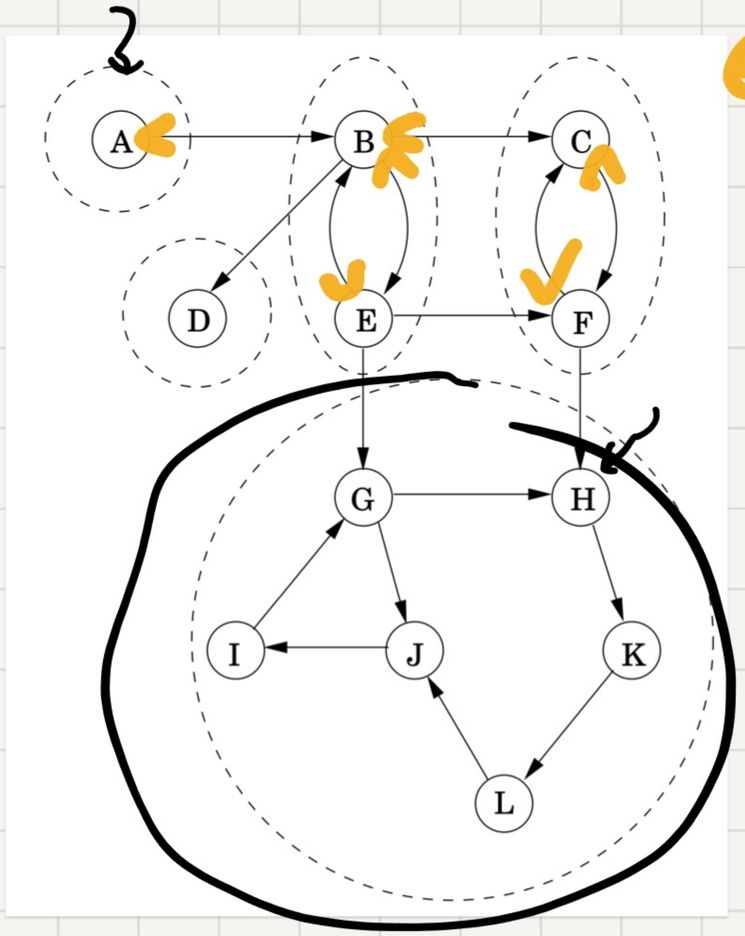
Claim: This relation is an equivalence relation

- reflexive: $\forall v: v \sim v$
- Symmetric: $u \sim v \Downarrow v \sim u$
- transitive: $u \sim v$, $v \sim w \Downarrow u \sim w$

Given a Graph G, shrink every SCC to a meta-node and connect meta-nodes $C \to C'$ iff there an edge $(u,v) \in E(G)$ $u \in C$, $v \in C'$.

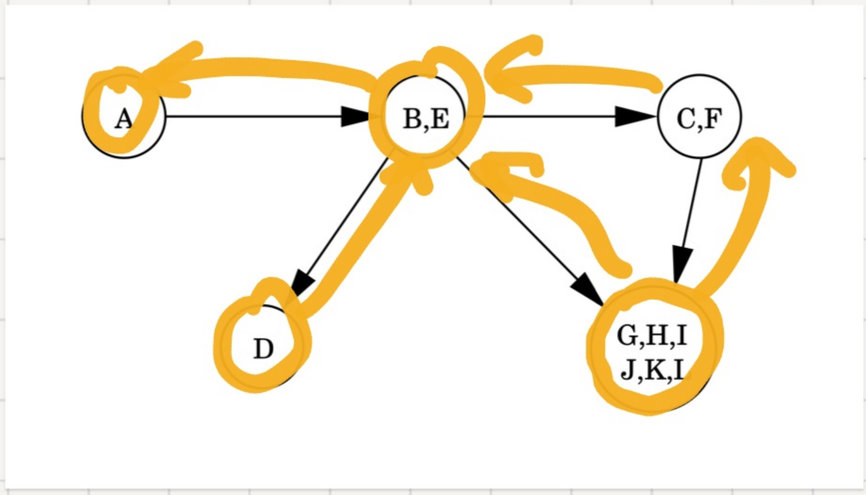
This produces the SCC meta-graph G'.

G^R



Example:

$(G^R)'$.

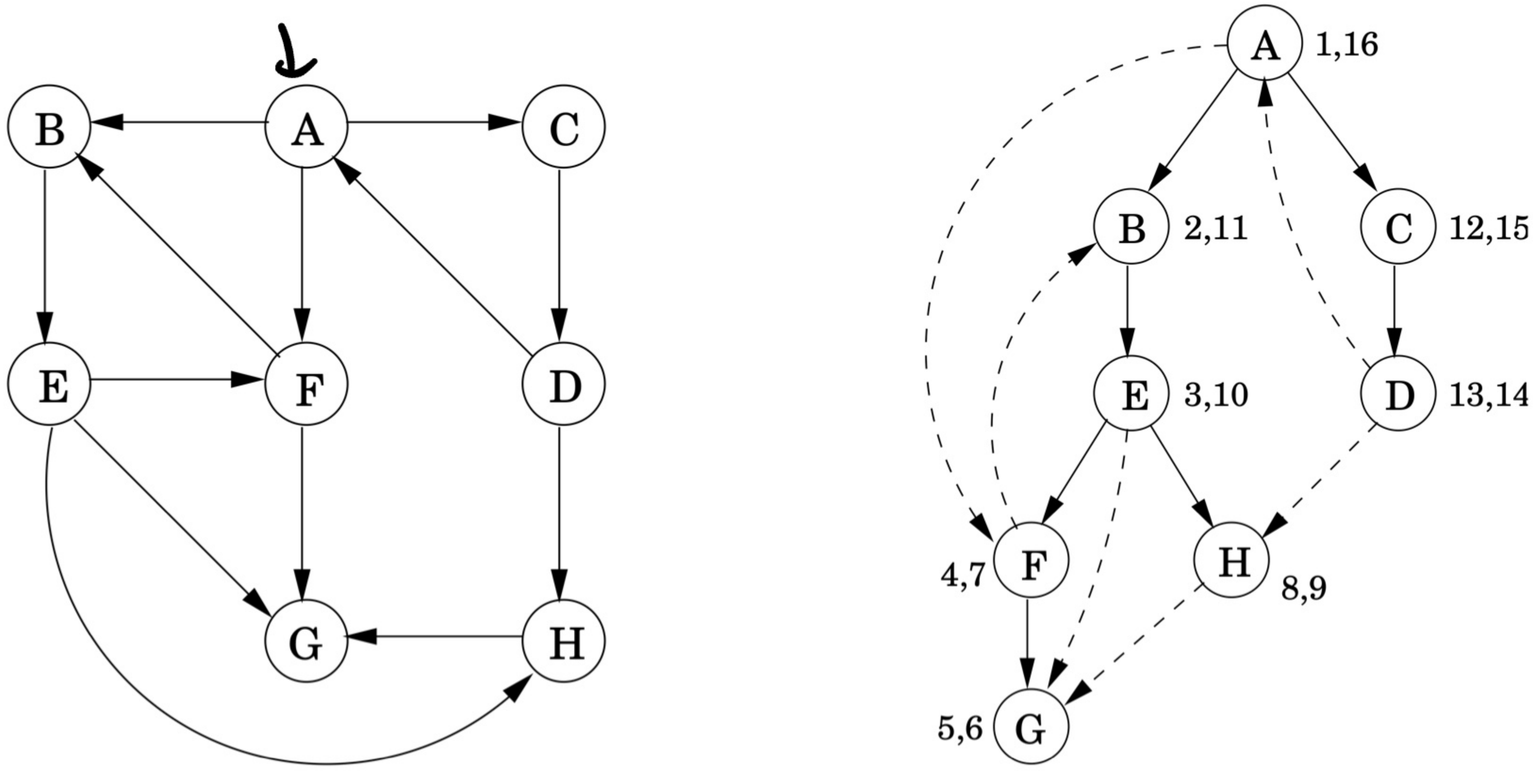


↑

Claim: ∀ digraph G, the SCC metagraph G' is a DAG.

Recall: pre & post numbers in DFS.

Figure 3.7 DFS on a directed graph.



[[[[[ ]] [ ]]] [[ ]]]

A B E F G G F H H E B C D D C A

1 2 3 4 . . . 16

u,v is a back edge if

[[ ]]

v u u v

[ ] [ ] cross

H H D D

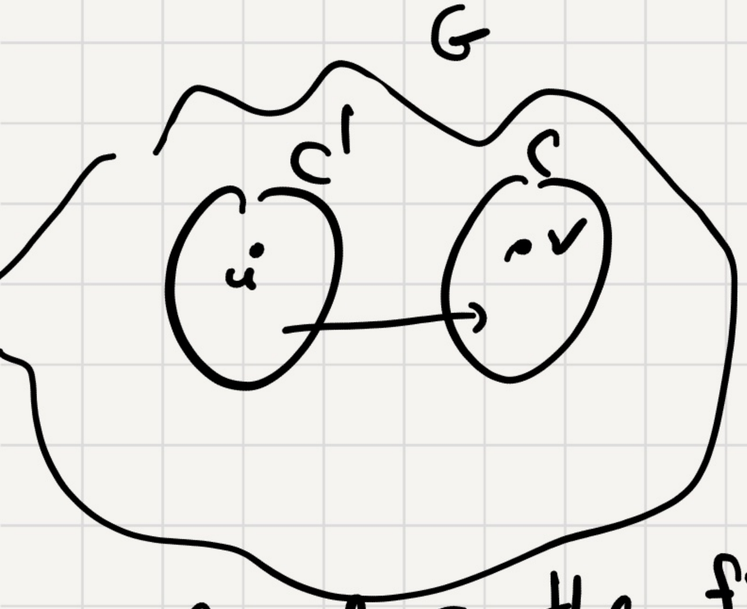
D → H

Q: Can we find a node $v \in V$ s.t. v is in a sink SCC?

_____ " _____ source SCCs?

Claim: Run DFS on G. The node with highest post number, then this node is in a source SCC.

Proof: Assume not. Let v be the node with highest post number.



C - SCC of v.

There exists another component C'

$C' \longrightarrow C$.

Consider the first vertex $u \in C \cup C'$ that is explored by DFS(G).

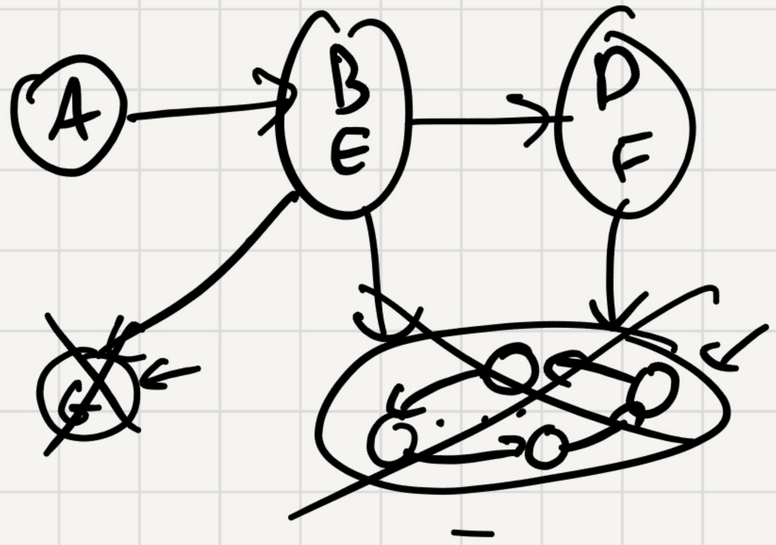
- $u \in C$: We process all vertices in C & only after we would start exploring C'. => contradiction
- $u \in C'$: $post(u) > post(v)$ => contradiction. □

We actually proved something stronger:

Claim: Ran DFS on G. $\forall$ SCCs $C' \to C$

the highest post number in C' > highest post number in C.

Strongly Connected Components - The Algorithm



Explore (G, v):

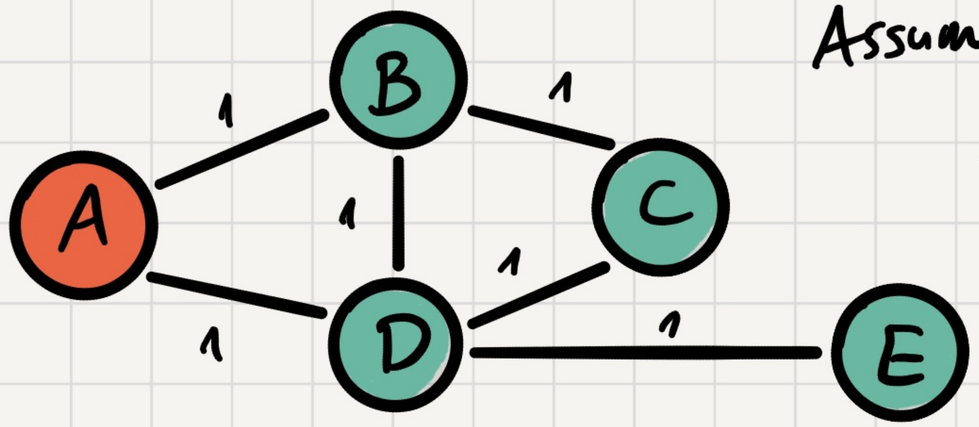
- visited[v] = true.
- comp[v] = scc
- for all (v,u) ∈ E:
  - if not visited[v]:
    - Explore(G,u)

SCC(G)

- Run DFS on G^R ⟹ post numbers.
- for all v∈V: visited[v] = false.
  comp[v] = null
- scc = 1
- for all v∈V, in descending order ordered by post on G^R:
  - if not visited[v]
    - Explore(G,v)
    - scc += 1.

New Topic: Distances in Graphs, Shortest Paths

Chapter 4 in DPV.



Assume: unit distances on edges
↑ for now

Q: What are the distances from A to all other vertices?

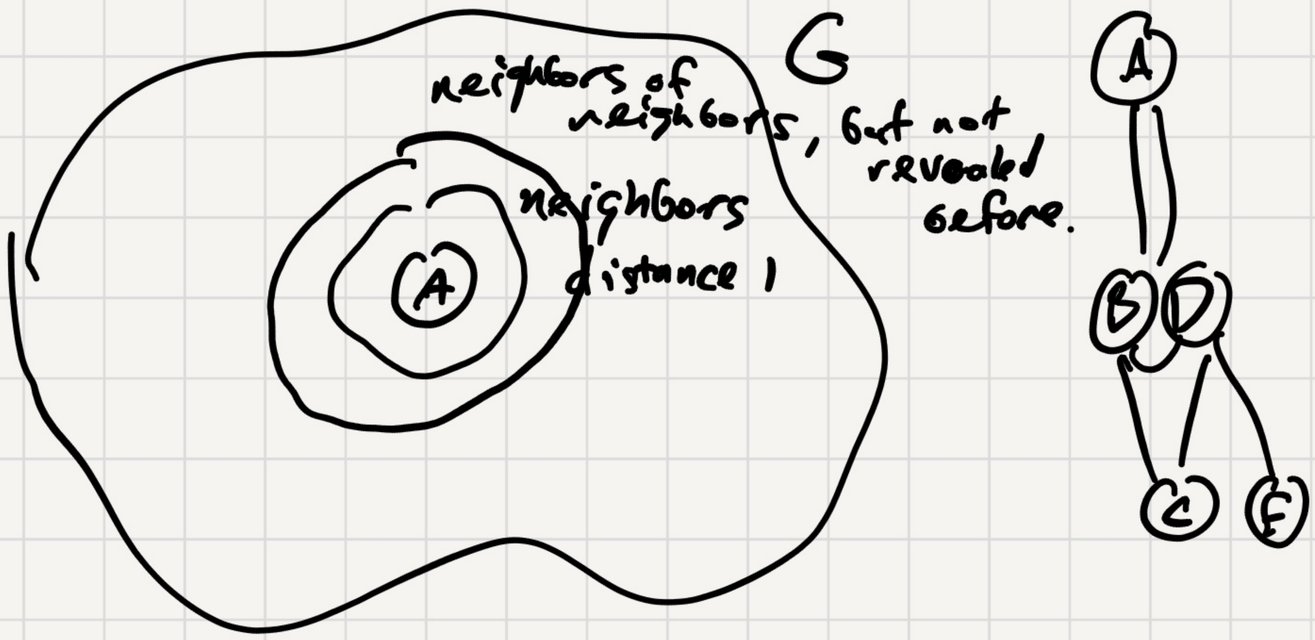
A → A
0

A → B
1

A → D
1

A → C
2

A → E
2



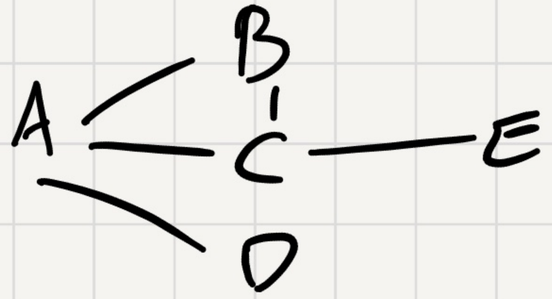
start vertex

BFS(G, s):

- dist[s] = 0.
- $\forall v \in V$, $v \neq s$: dist[v] = ∞
- ~~CurLayer~~ queue = [s]
- While ~~CurLayer~~ queue ≠ []:
  - v = queue.eject()
  - ~~NextLayer = []~~.
  - ~~For v in CurLayer:~~
    - For (v, u) ∈ E:
      - if dist[u] = ∞:
        - dist[u] = dist[v] + 1
        - ~~NextLayer.append(u)~~
        - queue.inject(u)
  - ~~CurLayer = NextLayer~~.

Claim: In iteration i of the while loop we identify all vertices of distance exactly i from s.

Proof: By induction on i.



[A]

[B, C, D]

[E]

Running Time of BFS

Very similar to DFS. Also in BFS

every vertex is inserted to the queue at most once
ejected from the queue

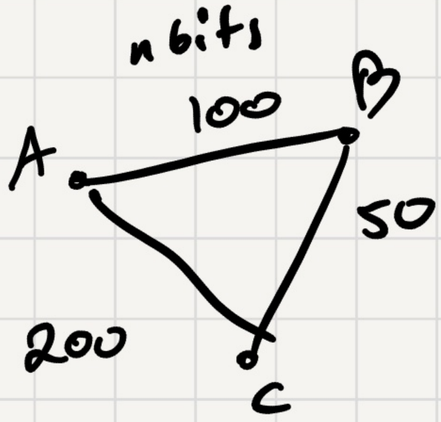
every edge is considered at most once (digraph)
or at most twice
(undirected graph)

$O(n+m)$.

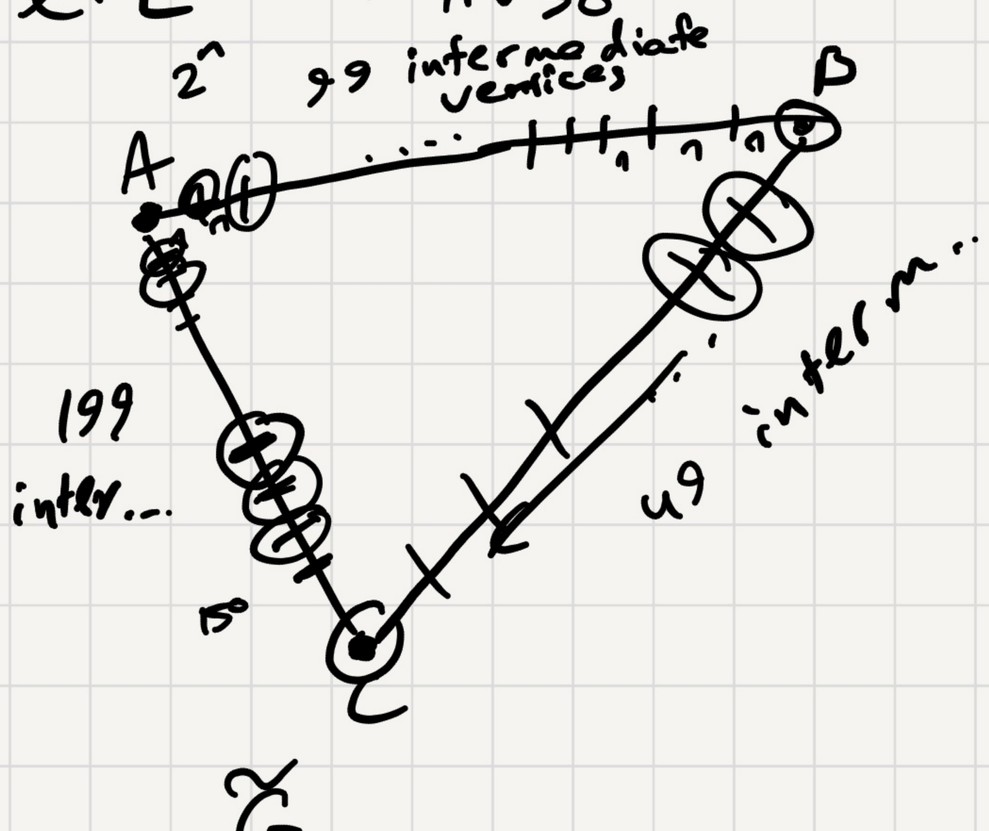
$n = |V|$ $\quad$ $m = |E|$.

Shortest Paths with Weights on Edges.

$G = (V, E)$ $\quad \ell : E \to \mathbb{N}_{>0}$



G



$\mathcal{G}$

This is an exponential

Next time:
- Dijkstra's Algorithm.
- Proof of Correctness.
- Runtime Analysis.
- What to do with negative weights?