Lecture 14: Shortest Path & Topological sorting

✡ Lecture objective

→ To figure out efficient way to find shortest path from a given vertex to all other vertices.

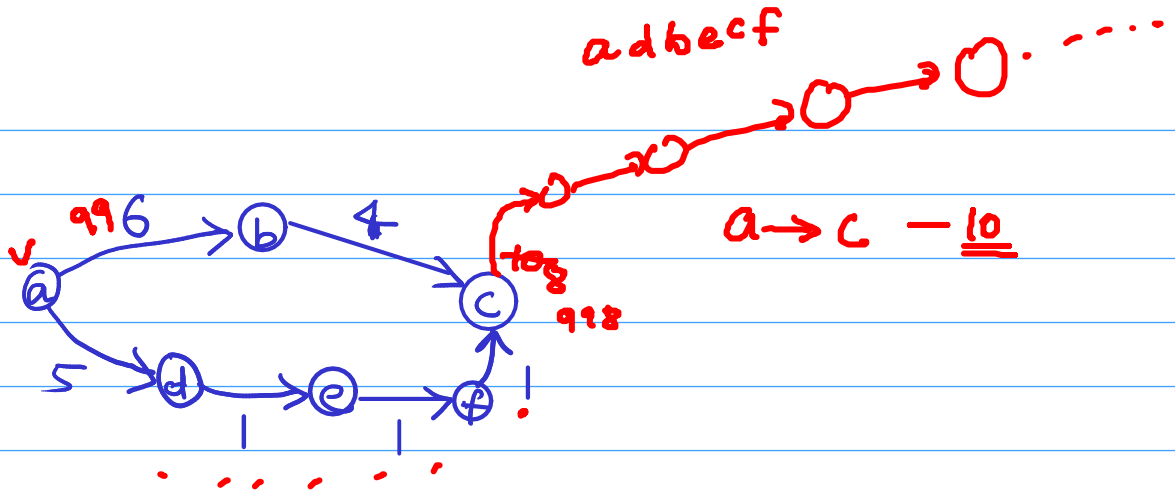
→ To understand topological ordering of DAG. And algorithm to find topological ordering of DAG using DFS.

→ At last, we will discuss problem with solution discussed in class to identify if given graph is semiconnected or not.

✡ Single Source shortest path

↳ Find shortest route from a given vertex to all other vertices.

↳ Can BFS be used?
↳ Yes, only if weights of the edges are same. WHY?



→ According to BFS, path from a to c has weight 6+4=10. While shortest path from a to c is 5+1+1+1=8.

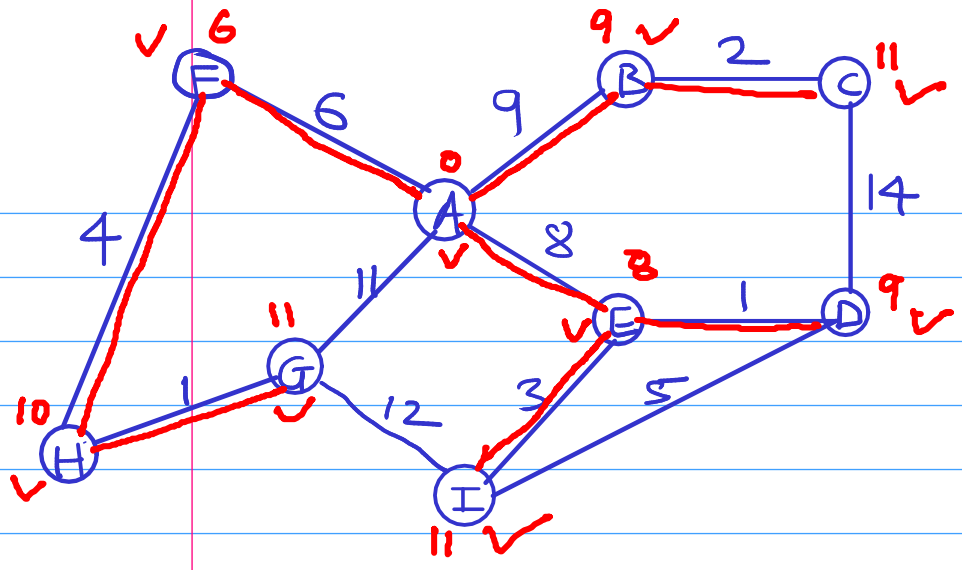
→ You may argue that we can update the weight of path from a to c when c is reached via 5+1+1+1. But then if there were other nodes already visited from c, then you will need to change all of them and so on. So, it is no longer BFS, and its complexity will increase.

↳ so, what else can we do!
→ There already is well known algorithm — Dijkstra's algo.

→ It is similar to Prim's algo.
→ But it chooses edges differently.

→ Lets look into example.

Note → Does not work if graph has (-)ve weights



How does Dijkstra's algo work?

(1) pick a source and marks it's distance 0.

(2) Mark distance of all other vertices as max value (∞)

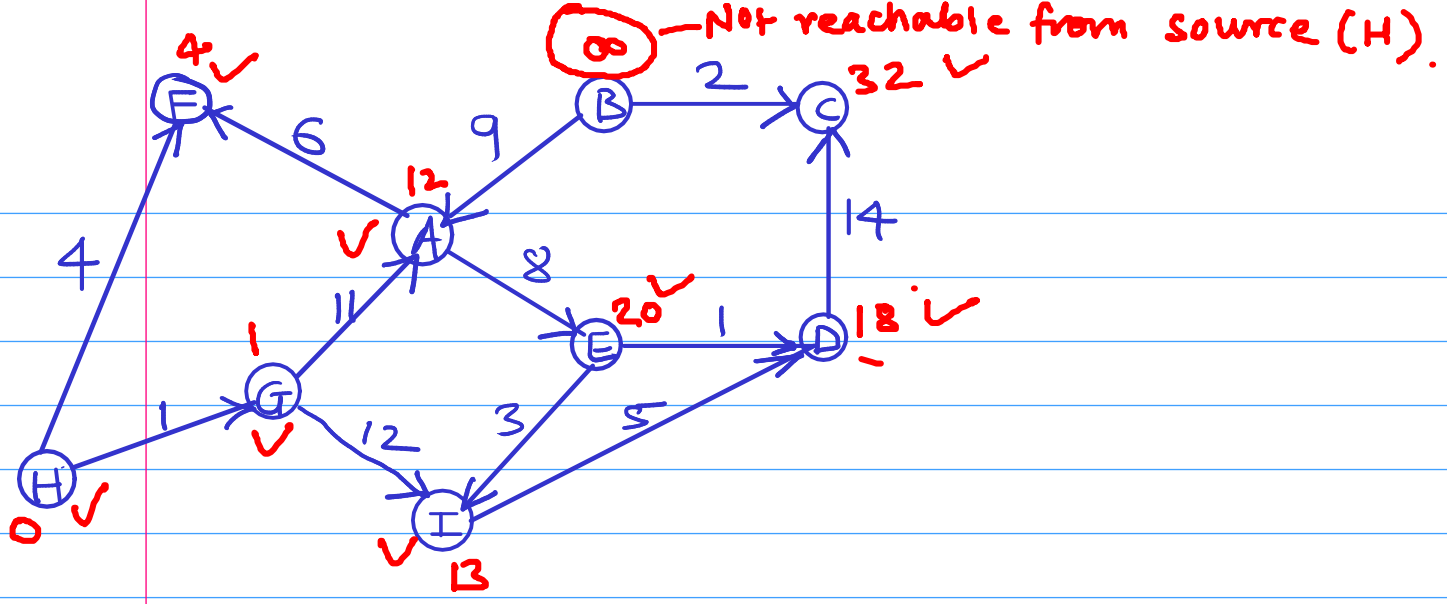
(3) Put all the vertices to S' (no mark)

(4) Repeat following still S' is empty

(4.1) pick vertex with smallest distance from S'. Move it to S (mark it)

(4.2) Update distance of neighbours (if required)

→ It works for directed graph too.



V → max. E ~ V^2

→ What will be the complexity of it?

→ Using heap, it can be reduced to $O(E \log(E))$ $O(E \log(V^2))$ $O(2E \log(V))$

sometimes referred as $O(E \log(V))$

→ Both are correct. why?

⇒ How to find all-pairs shortest path?

↳ V times execution of Dijkstra. once for each vertex.

↳ complexity!

$V \times O(E * \log(E))$

✓ $O(V * E * \log(E))$

↳ Floyd-warshall algo also gives, all-pairs shortest path.

It's complexity is $O(V^3)$. time

So, in theory its not better than running Dijkstra's algo V times.

Ignore it for now!

But, in practice, Floyd-Warshall performs better.

✓ https://stackoverflow.com/questions/10779054/time-complexity-of-floyd-warshall-algorithm

i [j] ✓

→ Floyd-warshall algo will be covered in DAA subject.

→ what is path matrix?

→ can we use Dijkstra to generate path matrix?

A A^2 i j A^3 ✓ (H.W.)

✡ Topological ordering (sorting)

→ For a given DAG, arrange (order) vertices in a sequence

$V_1, V_2, V_3, V_4 \ldots\ldots V_n$

Such that there does not exist a path from V_i to any vertex before it (V_1 to V_{i-1}).

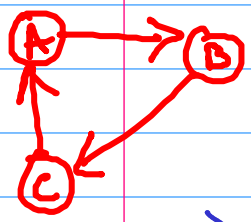
→ In other words for each edge

$V_i \rightarrow V_j$

V_i appears before V_j in the sequence

→ Is topological ordering possible if it is not DAG (if graph has cycle)?

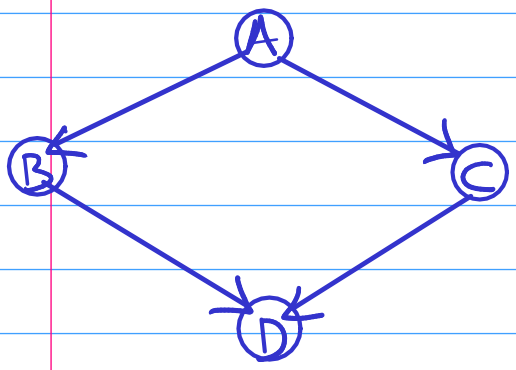
AB



No. Why?

→ Topological sort for undirected graph is also not possible. As edges have no direction, there is no one order for an edge.

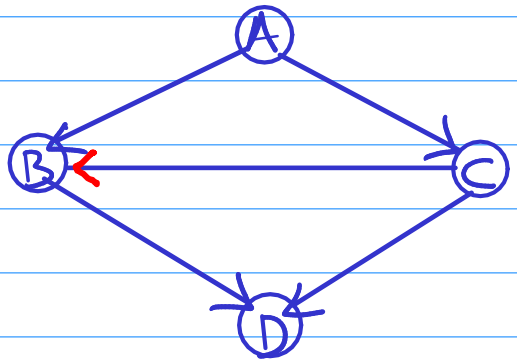
a — b
b — a
same



Which of these are valid topological sorts for given graph?

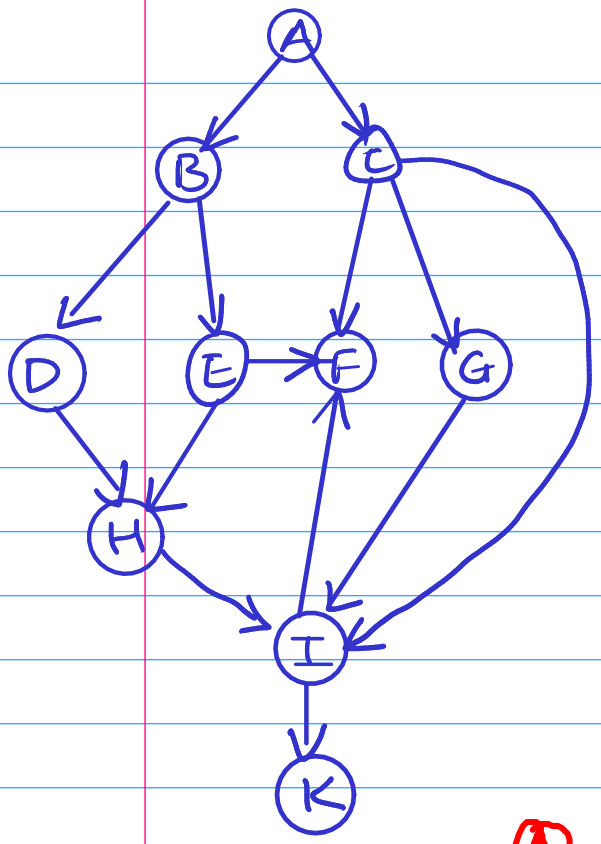
ABCD ✓	X
ACBD ✓	✓
BACD X	X
BCAD X	X
DBCA X	X

→ What if graph was as follow? Which of above are still topological sorts?

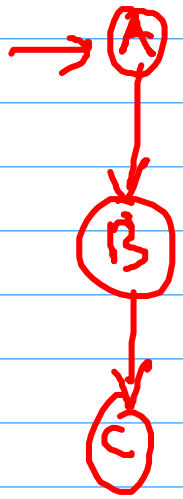


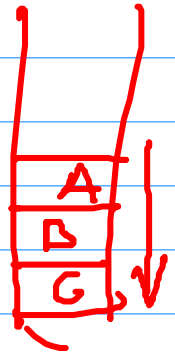
→ How can we use DFS for this?
→ Will see next.

→ Can BFS be used? Yes. Figure out yourself, how BFS can be used to come up with topological ordering for a given graph.



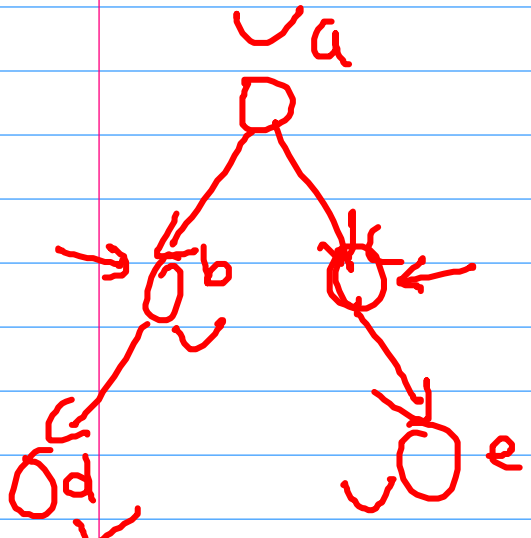
1. Pick any unvisited vertex at random and apply DFS (recursive)
2. While backtracking from a vertex, insert it into a separate stack.
3. If all vertices are not visited, goto Step 1.
4. Pop vertices from stack one by one and print them. It will produce topological sort.

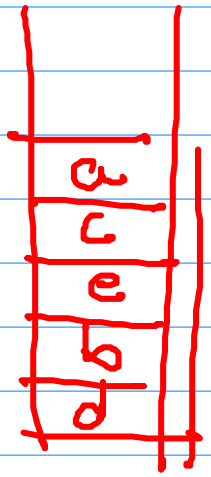




ABC

B



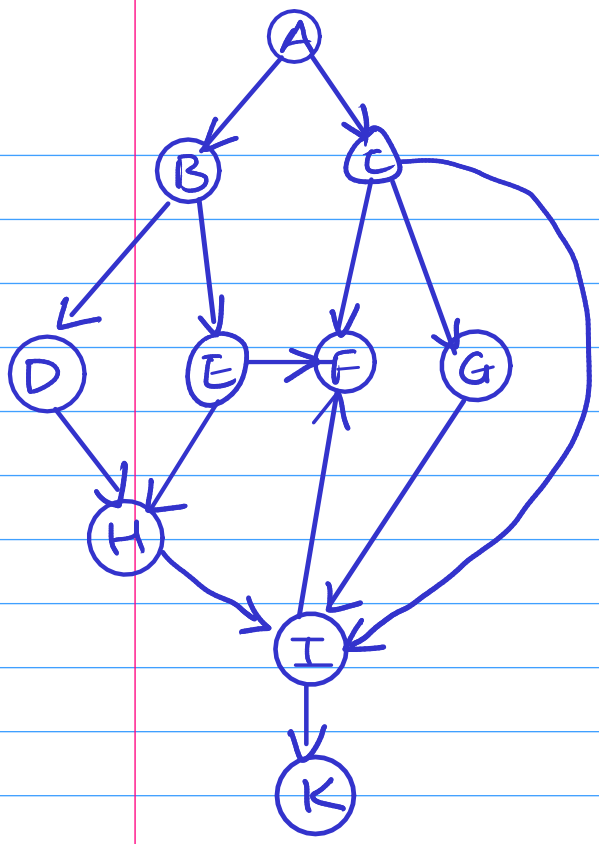


acebd

Stack content

→ dbeca

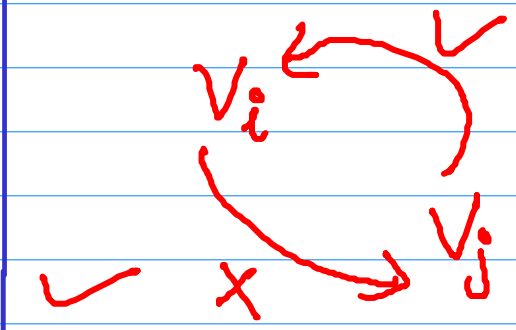
→ ecdba


A
B
C
D
E
F
G
H
I
K

☆ Issue with solution discussed in class to check if given digraph is semiconnected or not?

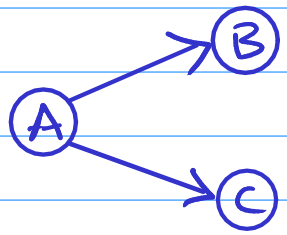
→ Do you remember the soln discussed in class?

Semiconnected graph	There exist a vertex from which all other vertices are reachable
T	T
T	F → Not possible. We argued this
F	T → We missed this ☹
F	F

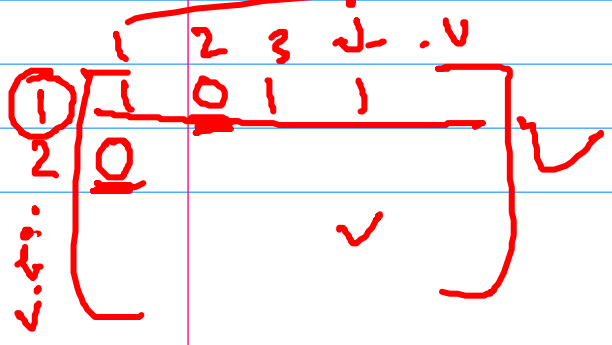


$V \times O(V+E)$

e.g.



$O(V^2+VE)$



→ All vertices are reachable from A.

→ But no path from B to C or C to B. So, it is not semiconnected.

→ Link to better soln

→ what is the solution then?

(1) Path matrix can be used as discussed above.

(2) There is better solution using strongly connected components and topological ordering.

https://stackoverflow.com/questions/30642383/determine-if-a-graph-is-semi-connected-or-not

NOTE: Notes with this color have been added after lecture.