Professor Hamlin
Class 15

Agenda

1) Admin → extra BFS/DFS video
2) Review
3) Searching through Graph
   - Breadth First Search (BFS)
   - Depth First Search (DFS)
4) Shortest path in weighted graph
   - Dijkstra's Algorithm

Review: Graph: $G = (V, E)$

Vocab: adjacent, degree, walk, path, cycle connected, subgraphs, weighted, directed

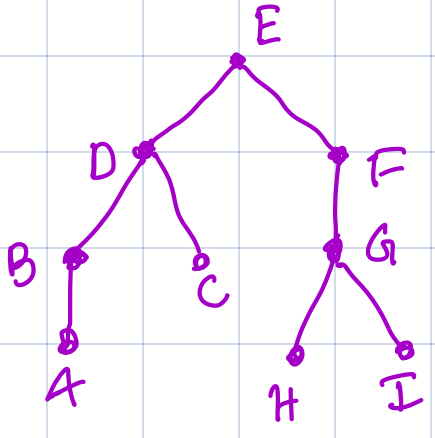
Trees: connected, acyclic, parent, children, leafs, siblings, ancestors, descendents, forest

On computer: adjacency list, adjacency matrix

Graph isomorphism

Exercise: 1. Draw the following adjacency list as E-rooted tree

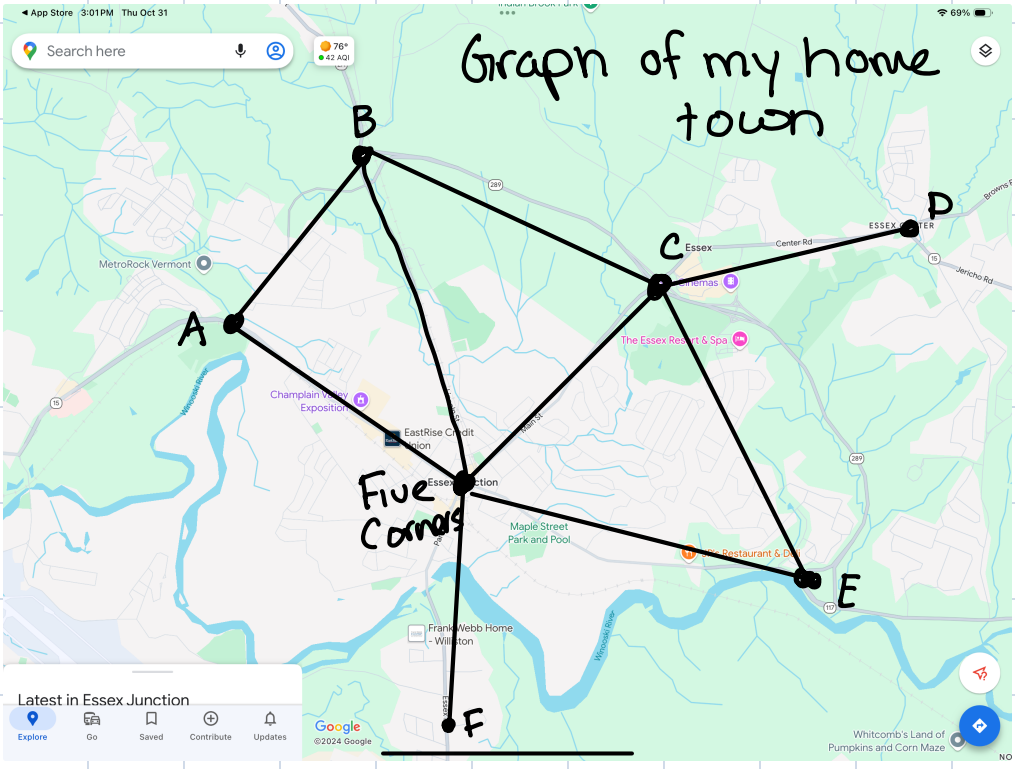
A: [B]
B: [A, D]
C: [D]
D: [B, C, E]
E: [D, F]
F: [G, F]
G: [F, H, I]
H: [G]
I: [G]



2. In the previous graph, what is the path between A & G?

A, B, D, E, F, G

Searching a graph



As a teen could go anywhere within 10 minutes of Scorners. What could I get to?

A, B, C, E, F

D to far away :(

This is easy for a human to do but harder for a computer.

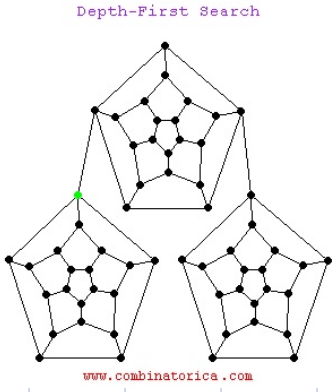
Need to describe a method that works for all graphs: an algorithm

First algorithm...

Depth First Search (DFS) (the brave algorithm)

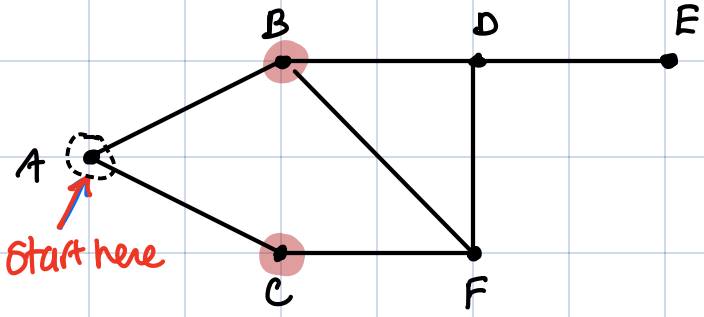
Intuition: keep going far away (via adjacent nodes) as possible before backing up and trying a different path.

https://www3.cs.stonybrook.edu/~skiena/combinatorica/animations/search.html



Formally, visit adjacent, unvisited vertice as long as possible, then back up one edge look for another unvisited vertex to visit using same method.

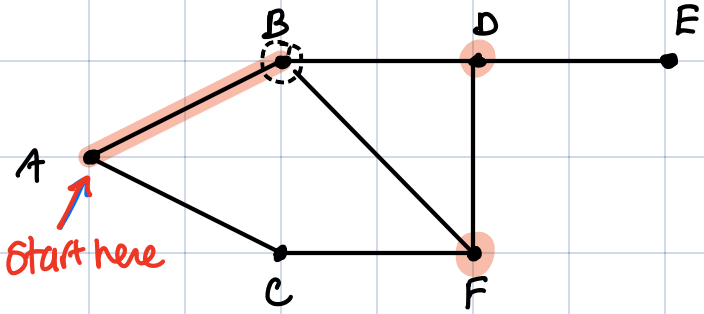
Example



Visited

A

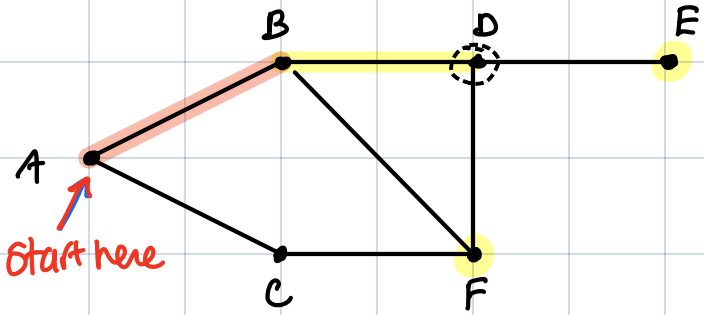
2 neighbors - B, C. Choose 1st alphabetically



Visited

A, B

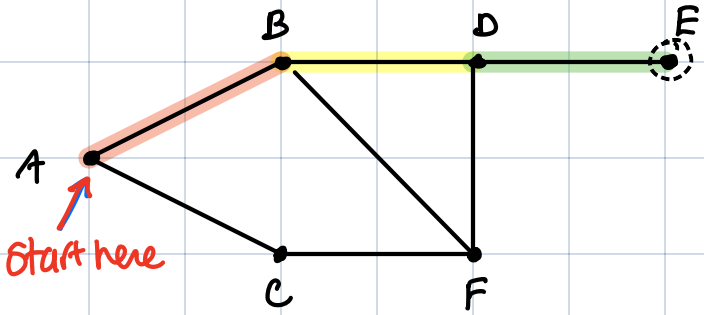
2 neighbors - D, F. Choose D



Visited

A, B, D

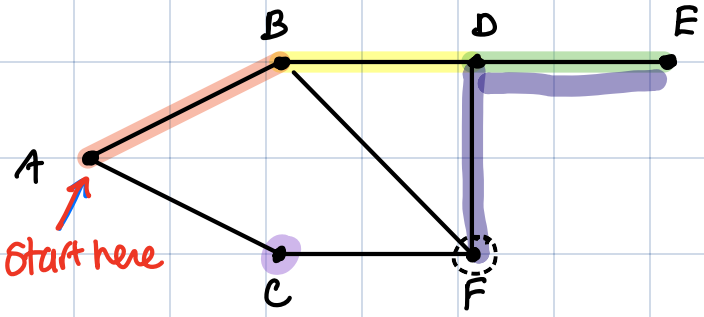
2 neighbors - E, F. Choose E



Visited

A, B, D, E

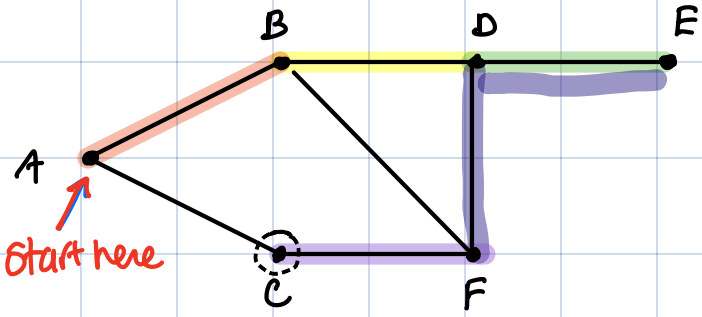
No unvisited neighbors, backtrack to D, F unvisited



Visited

A, B, D, E, F

Two neighbors: B, C, only C unvisited

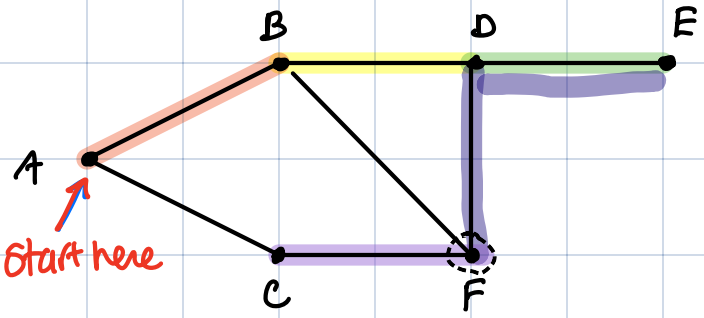


Visited

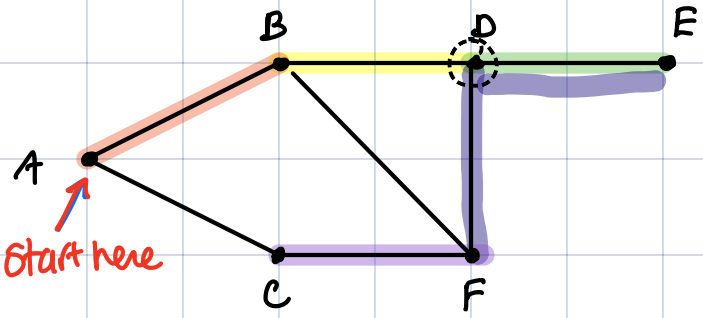
A, B, D, E, F, C

We may be done but Computer isn't!

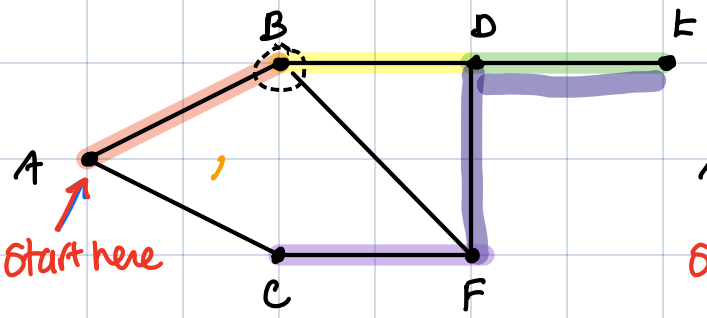
One neighbor, A but visited, so backup...



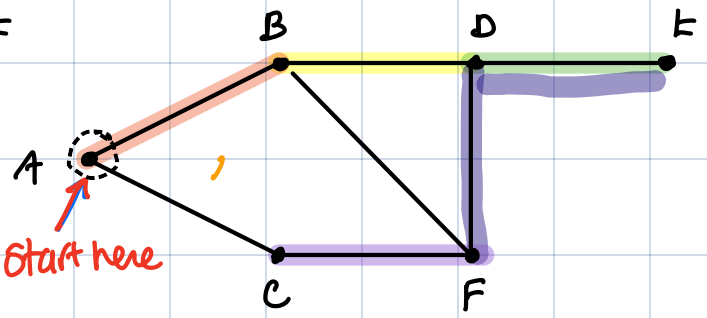
... to F



... to D



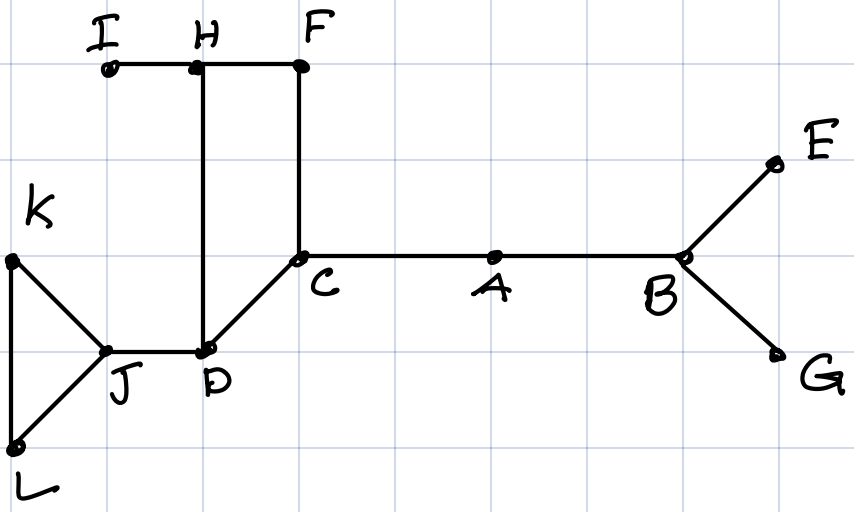
... to B



... to A

Then the computer is done.

Exercise



1) DFS starting at A

A B E G C D H F I J K L

2) DFS starting at H

H D C A B E G F J K L I

3) DFS starting at G

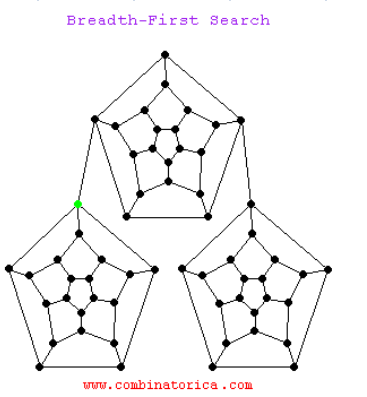
G B A C D H F I J K L E

Breadth First Search (BFS)

(the cautious algorithm)

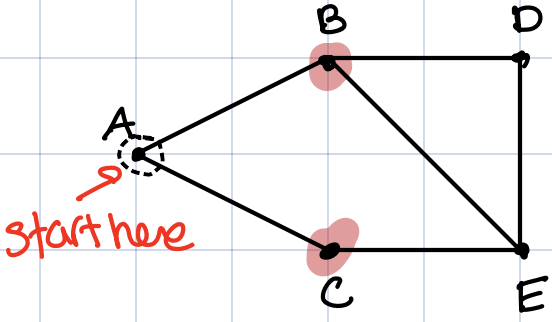
idea: explore all the close things before venturing out the next step

https://www3.cs.stonybrook.edu/~skiena/combinatorica/animations/search.html



Formally: visit all vertices adjacent to the starting vertex then do the same from each of those vertices

Example

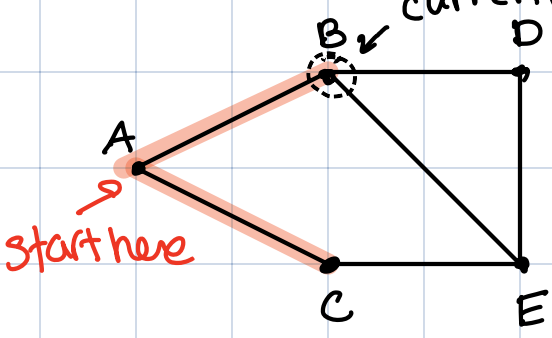


Visited

A

Neighbors are B, C, choose to visit in alphabetic order: so choose B.

current node

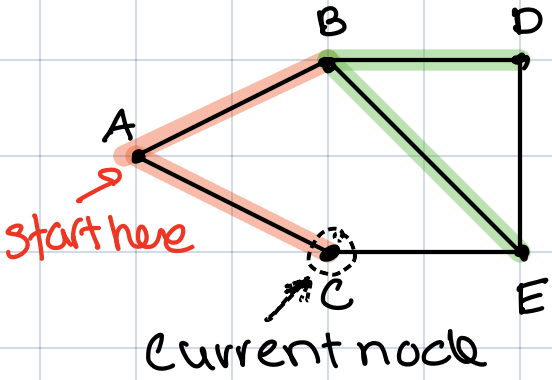


Visited

A B C

B has neighbors D, ~~A~~, E. ~~But have already~~

~~visited all~~, so add D, E

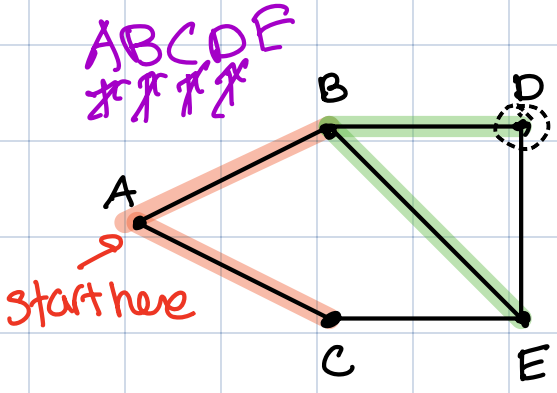


Visited

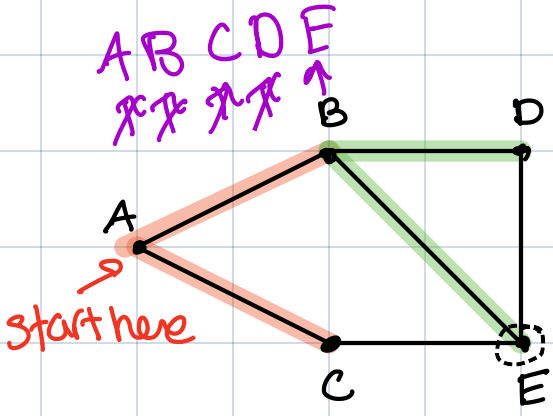
A B C D E

we may be clone but the computer is not!

C has no unvisited neighbors so continue to...



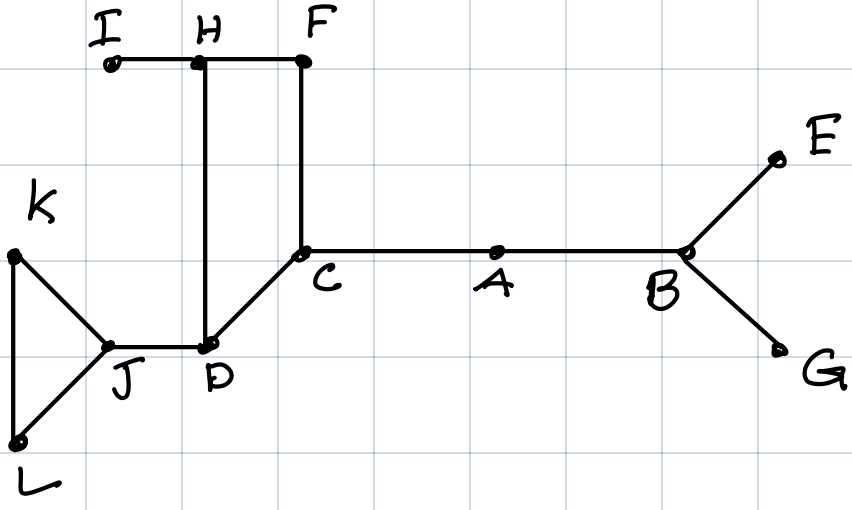
... to D



... to E

Now computer is clone!

Exercise



1) BFS starting at A

A B C E G D F H J I K L

2) BFS starting at H

H D F I C J A K L B E G

3) BFS starting at G

G B A E C D F H J I K L

Why BFS/DFS (again)

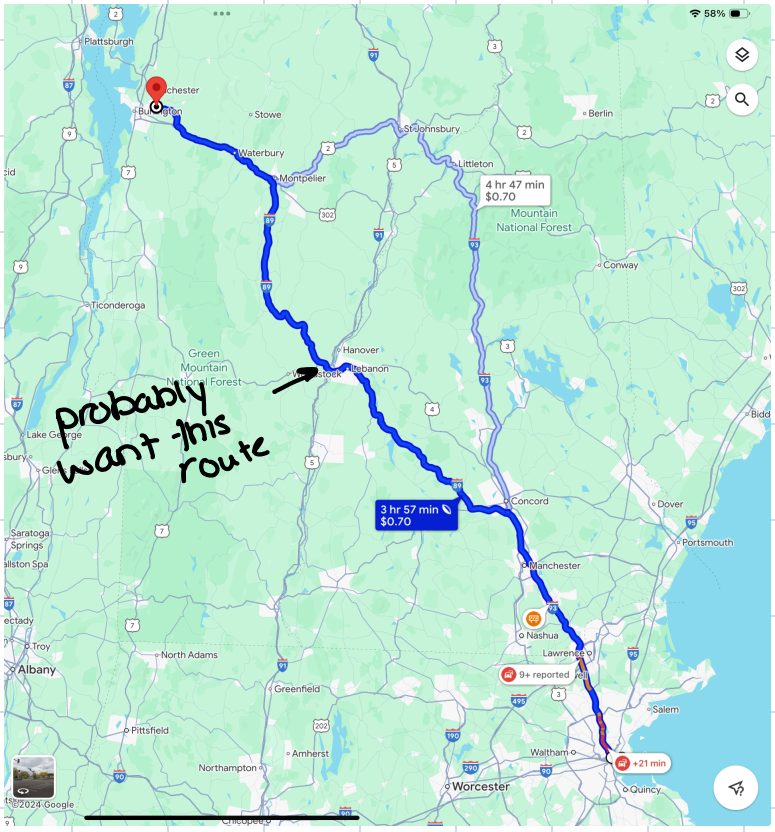
→ DFS/BFS gives you a connected subgraph (where can I get by taking only one airline)

→ DFS can detect cycles in a graph (we bump into an already visited neighbor)

→ BFS orders nodes by how far away they are (1-hop, 2-hops, etc)

→ They can be written recursively ☺

Now a different type of problem: how to get to point A to point B the fastest/cheapest/etc.



Known as "~~shortest~~ cheapest path" problem - can be solved using Dijkstra's algorithm

(best path is not always shortest)

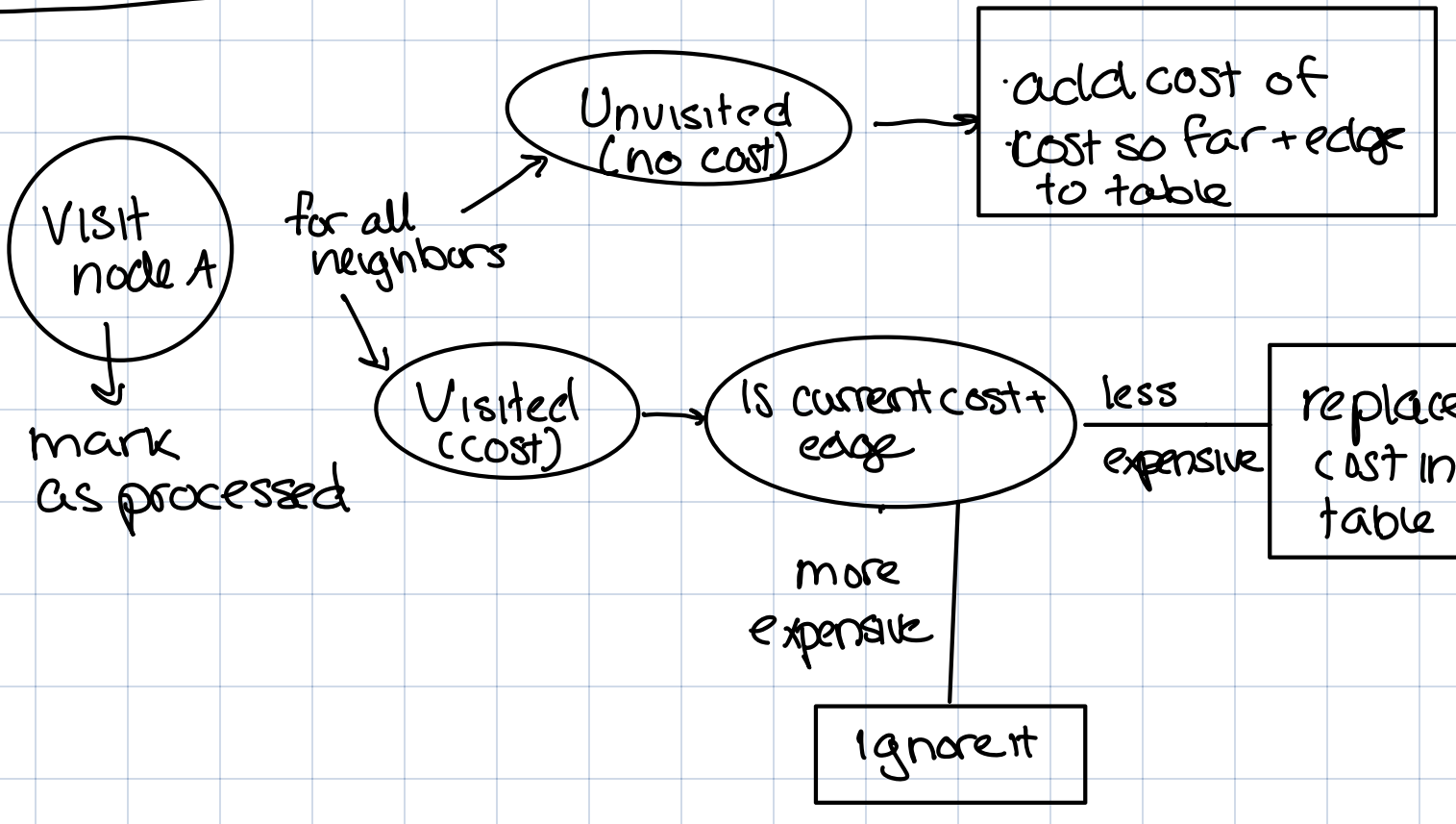
Formally: - weighted graph w/ positive edges
- what is least cost from A to B if we add up the edges along the path

Approach: keep track of cheapest path so far in table

Warning flow chart below is for reference will build up intuition w/ example

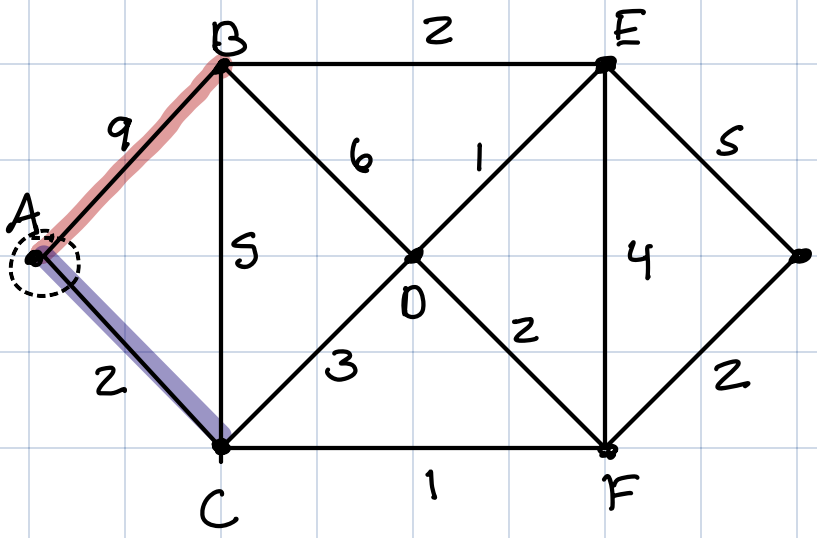
---

For reference



Then go to next node not processed yet that has minimum cost

Cheapest path from A to G:



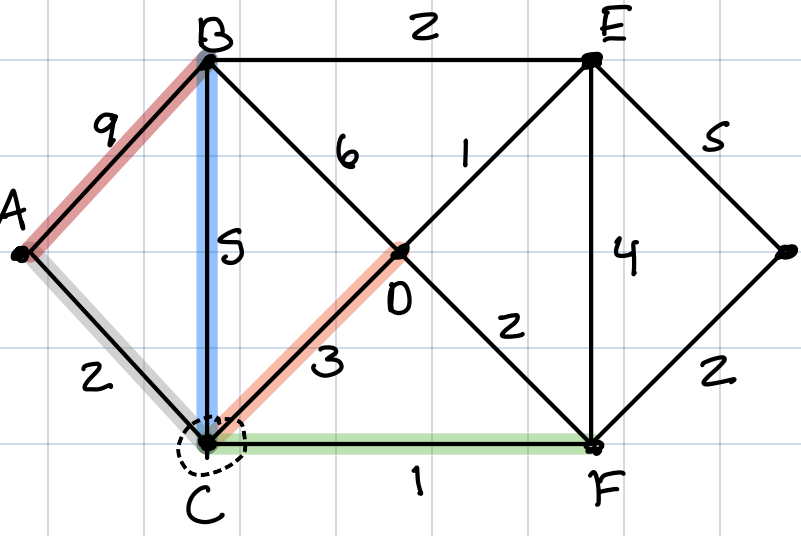
	A	B	C	D	E	F	G
Proc	X						
cost	0	9	2				

↑ currently cheapest path between A & B is 9

1. Mark A as processed
2. Examine all neighbors
   no cost? add to table

Next vertex is
- ☐ unvisited
- ☐ lowest cost

so C



	A	B	C	D	E	F	G
Proc	X		X				
cost	0	~~9~~ 7	2	5		3	

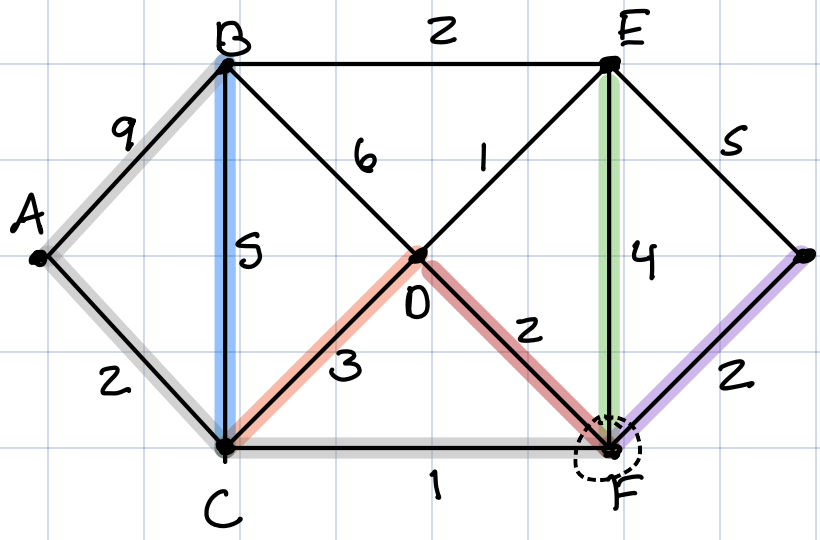
Cost for D is 2+3

Cost for F is 2+1

Cost for B is 2+5
7 < 9, update

1. Mark vertex as processed
2. Examine all neighbors
   no cost? add to table
   cost? lower, add to table
   higher, ignore

Next vertex is F



	A	B	C	D	E	F	G
Proc	X		X			X	
cost	0	7	2	5	7	3	5

Cost for D is 3+2

5=5, ignore

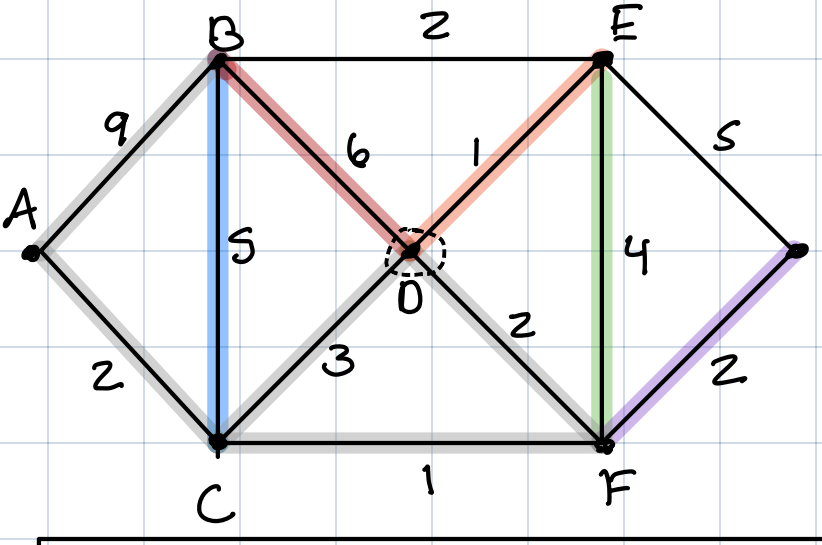
Cost for E is 3+4=7

Cost for G is 3+2

next vertex is D

D & G have same value but we go alphabetically

1. Mark vertex as processed
2. Examine all unvisited neighbors
   - no cost? add to table
   - cost? lower, add to table
   - higher, ignore
   - or =



	A	B	C	D	E	F	G
Proc	X		X	X		X	
cost	0	7	2	5	~~7~~ 6	3	5

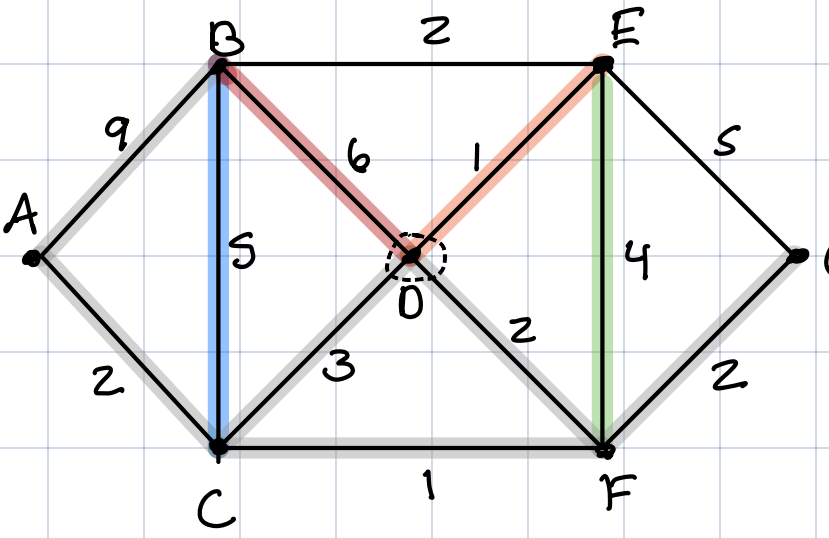
cost for B is 5+6,

cost for E is 5+1

6<7

1. Mark vertex as processed
2. Examine all neighbors
   - no cost? add to table
   - cost? lower, add to table
   - higher, ignore
   - or =

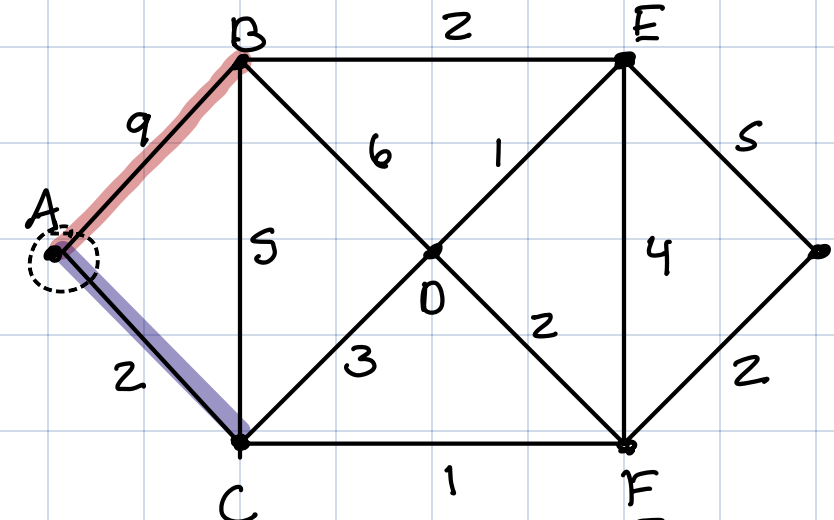
Next node is G!



	A	B	C	D	E	F	G
Proc	X		X	X		X	X
cost	0	7	2	5	6	3	5

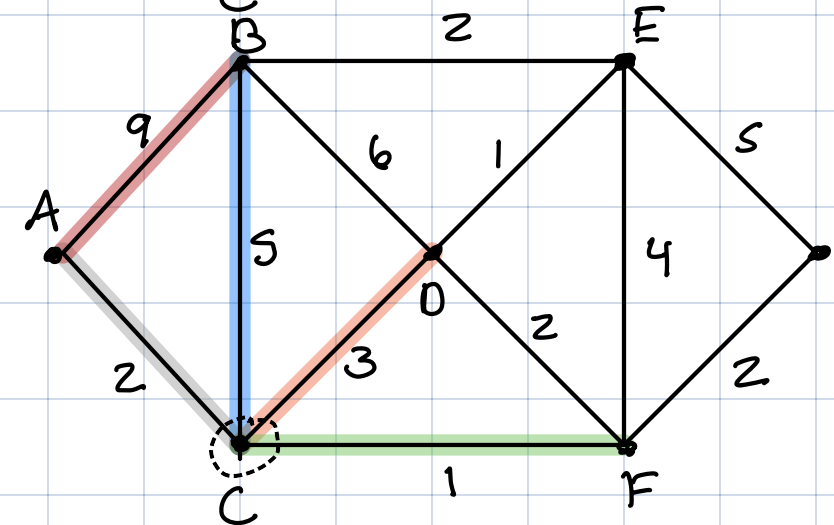
1. Mark vertex as processed
2. Examine all neighbors
   - no cost? add to table
   - cost? lower, add to table
   - higher, ignore
   - or =

We now know our shortest path to G cost 5, but what is that path?

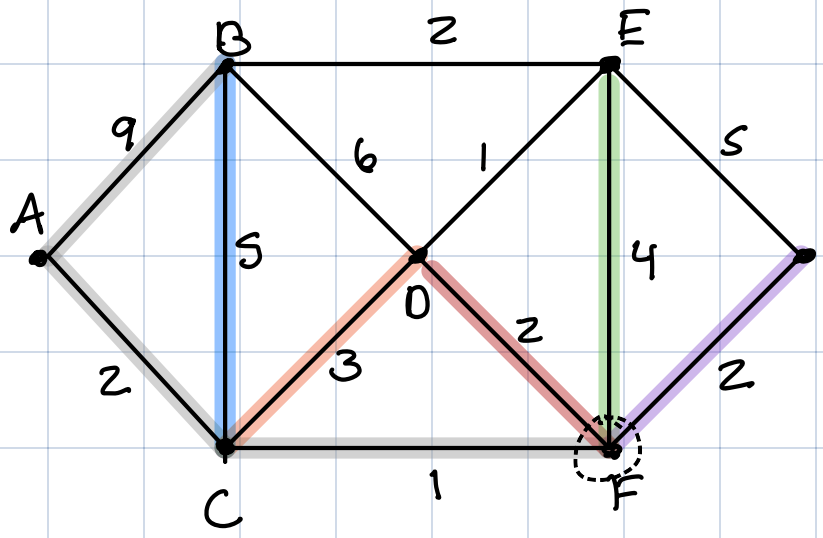


	A	B	C	D	E	F	G
Proc	X						
cost	0	A:9	A:2				

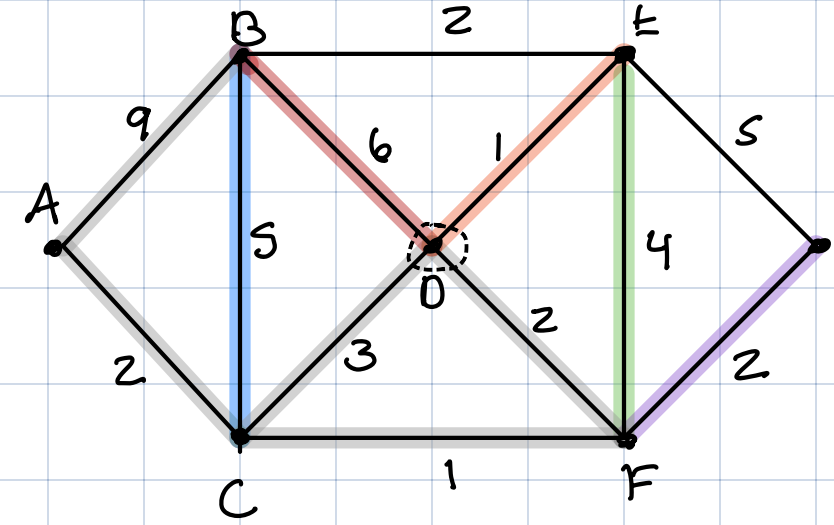
Keep track of proceeding vertex



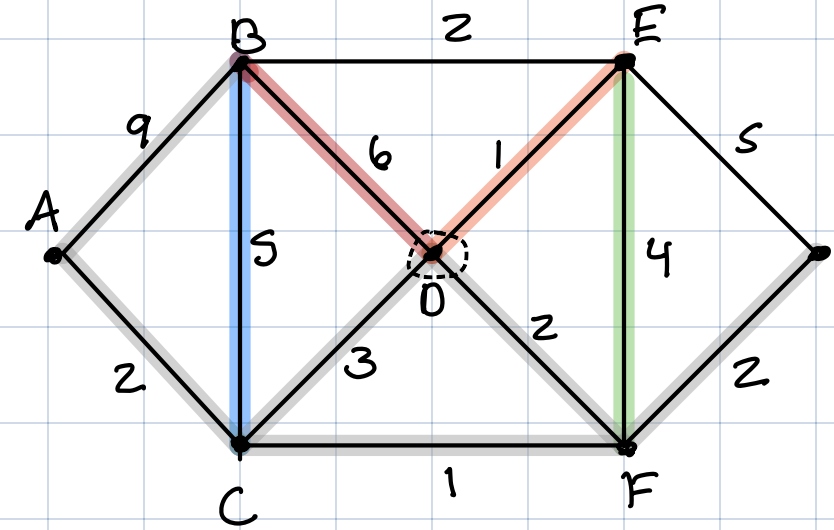
	A	B	C	D	E	F	G
Proc	X		X				
cost	0	C:7	A:2	C:5		C:3	



	A	B	C	D	E	F	G
Proc	X		X			X	
Cost	0	C:7	A:2	C:5	F:7	C:3	F:5



	A	B	C	D	E	F	G
Proc	X		X	X		X	
Cost	0	C:7	A:2	C:5	D:6	C:3	F:5



	A	B	C	D	E	F	G
Proc	X		X	X		X	X
Cost	0	C:7	A:2	C:5	D:6	C:3	F:5

	A	B	C	D	E	F	G
Proc	X		X	X		X	X
Cost	0	7	A:2	C:5	6	C:3	F:5

More backward through table to find path

A → C → F → G

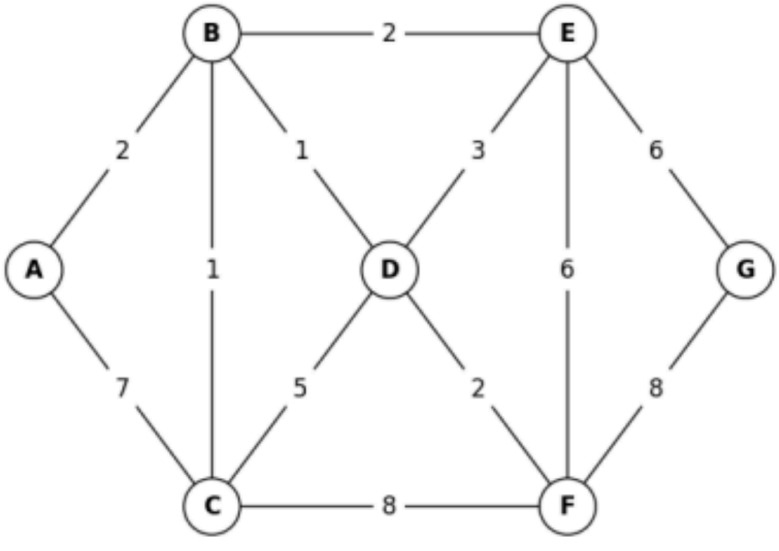
On Hw/Exam

iteration	node visited	A	B	C	D	E	F	G
0	A	start:0	A: 9	A: 2	none	none	none	none
1	C	start:0	C: 7	A: 2	C: 5	none	C: 3	none
2	F	start:0	C: 7	A: 2	C: 5	F: 7	C: 3	F: 5

The path with min weight is: G ← F ← C ← A

Example posted in detail on course web-page

Exercise 1



Shortest path from A to G

iteration	visited	A	B	C	D	E	F	G
0	A	0	A:2	A:7	-	-	-	-
					.			
					.		.	
			.