CS1800 Day 17

Admin:
- HW5 due today
- HW6 released today
- "Extra" video on BFS / DFS (piazza post 440)
- might end few mins early today, feel free to hang out if you have BFS / DFS or Dijkstra questions

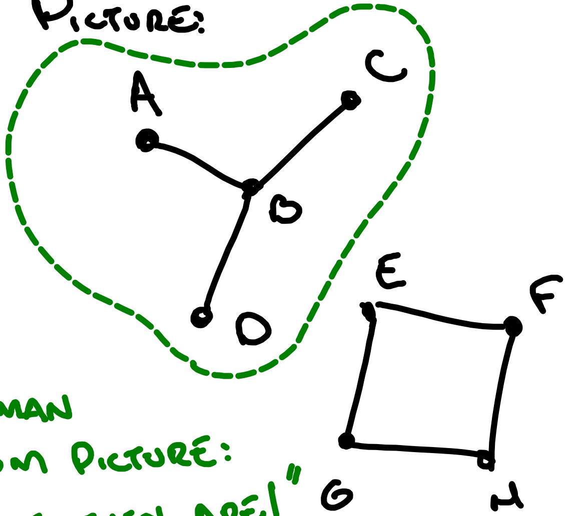
Content:
Searching through all the nodes in a graph:
- Breadth First Search (BFS)
- Depth First Search (DFS)

Finding the shortest path between two nodes in a weighted graph:
- Dijkstra's Algorithm

Searching a graph: (BFS & DFS intro)

Goal: Using a computer, walk (order) to all nodes which are connected to node A

PICTURE:



HUMAN FROM PICTURE: "HERE THEY ARE!"

NEIGHBOR LISTS

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

COMPUTER FROM REPRESENTATION: "NOT SO SIMPLE..."

Depth First Search: Inuition & Animation

Approach: "visit an adjacent, unvisited node as long as possible,
then backup one edge and look for another vertex to visit, using a depth first search."

<view gif>

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

Breadth First Search: Intuition & Animation

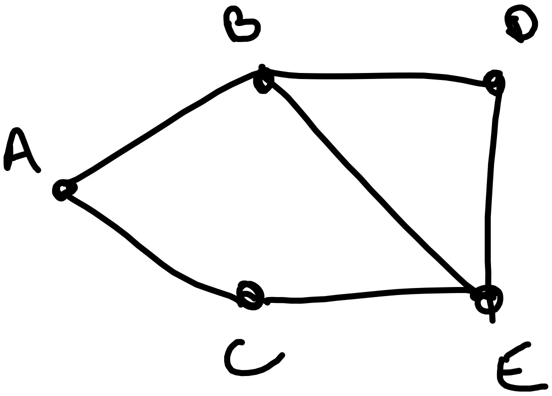
Approach: "Visit all the vertices adjacent to the starting vertex,
then do a breadth first search from each of those vertices."

<view gif>

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

Breadth First Search: Example

Approach: "Visit all the vertices adjacent to the starting vertex,
then do a breadth first search from each of those vertices."



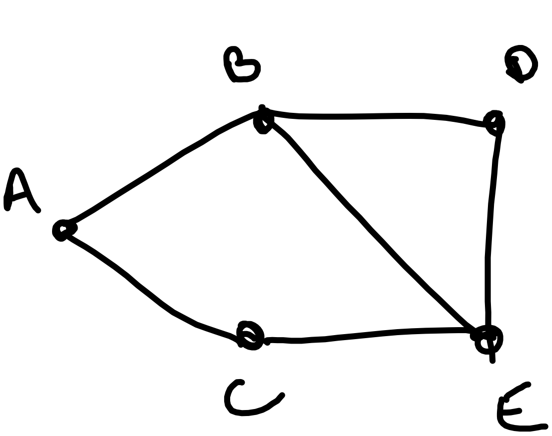
VISITED:

A

BFS / DFS require some starting node be given, where the search is initialized.

Breadth First Search: Example

Approach: "Visit all the vertices adjacent to the starting vertex,
then do a breadth first search from each of those vertices."



VISITED:

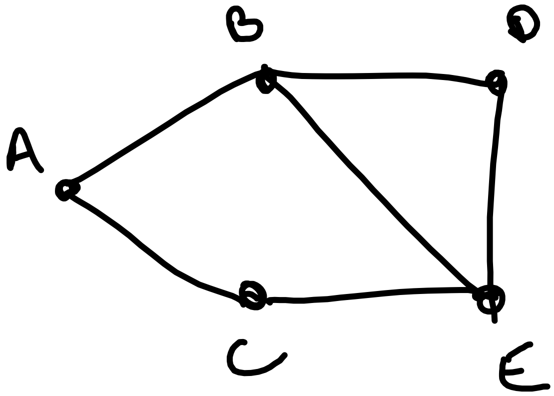
A BC

CURRENT NODE

A's neighbors are {B, C}. We could visit them in any order (BC or CB) as a valid BFS. We choose alphabetical ordering to standardize output

Breadth First Search: Example

Approach: "Visit all the vertices adjacent to the starting vertex, then do a breadth first search from each of those vertices."



VISITED:

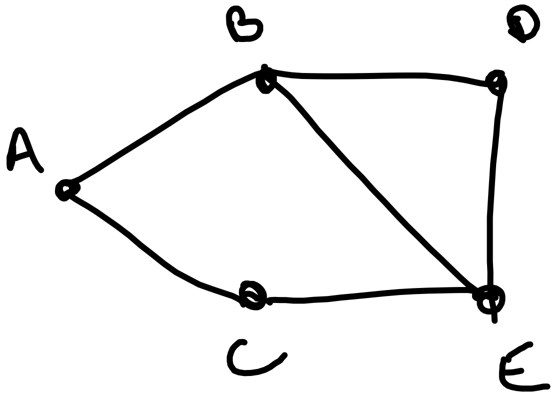
A B C D E

CURRENT NODE

B's neighbors are {A, D, E} but we only add the unvisited nodes to our list (again in alpha order)

Breadth First Search: Example

Approach: "Visit all the vertices adjacent to the starting vertex,
then do a breadth first search from each of those vertices."



VISITED:

A B C D E

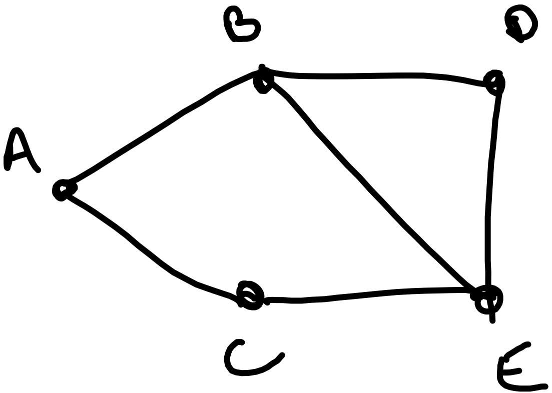
CURRENT NODE

(C has no unvisited neighbors to add)

Looking at the picture, you can tell we're done.
The computer doesn't know ... must finish BFS on visited list

Breadth First Search: Example

Approach: "Visit all the vertices adjacent to the starting vertex,
then do a breadth first search from each of those vertices."



VISITED:

A B C D E

(D has no unvisited neighbors to add)

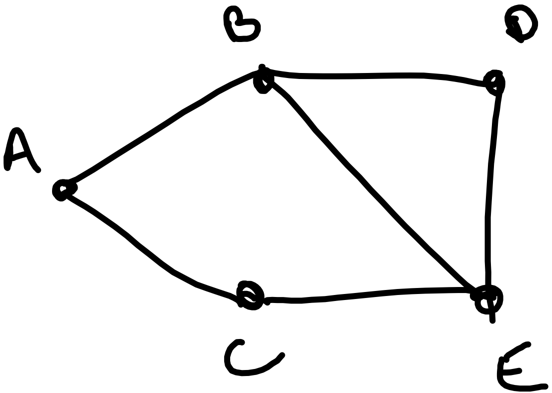
CURRENT NODE

Looking at the picture, you can tell we're done.
The computer doesn't know ... must finish BFS on visited list

Breadth First Search: Example

A B C D E

Approach: "Visit all the vertices adjacent to the starting vertex,
then do a breadth first search from each of those vertices."



VISITED:

A B C D E

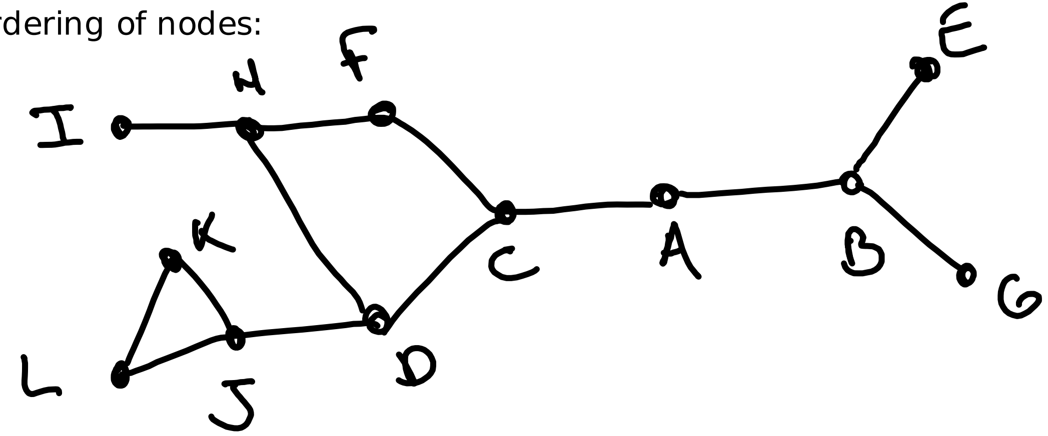
(E has no unvisited neighbors to add)

CURRENT NODE

Looking at the picture, you can tell we're done.
The computer doesn't know ... must finish BFS on visited list

In Class Activity: Breadth First Search

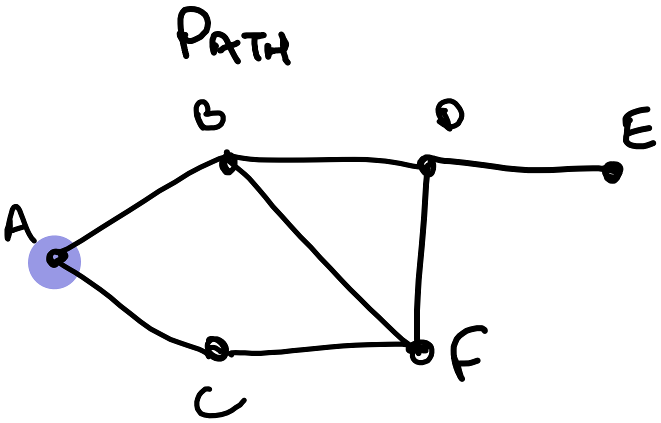
Give the BFS ordering of nodes:
- starting at A
- starting at H
- starting at G



BFS start @ a: ABCE GDFH JIKL
BFS start @ h: HDFI CJAK LBEG
BFS start @ g: GBAE CDFH JIKL

Depth First Search: Example

Approach: "visit an adjacent, unvisited node as long as possible,
then backup one edge and look for another vertex to visit, using a depth first search."

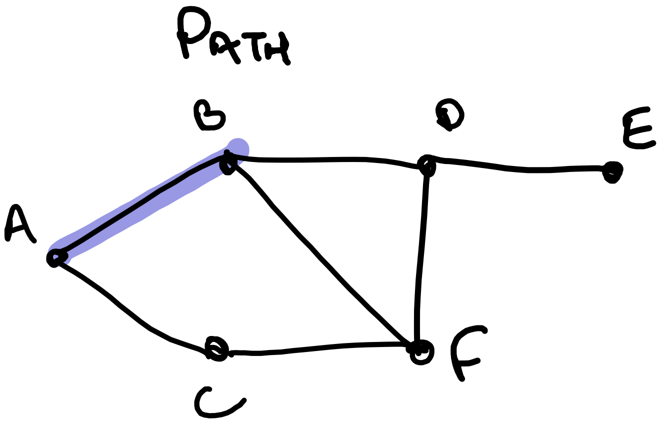


VISITED:

A

Depth First Search: Example

Approach: "visit an adjacent, unvisited node as long as possible,
then backup one edge and look for another vertex to visit, using a depth first search."



VISITED:

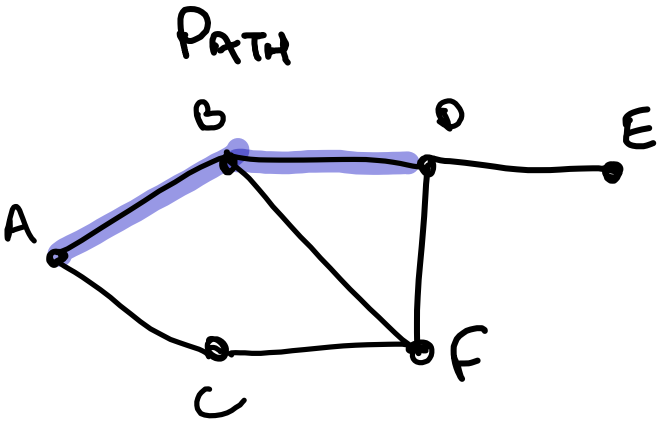
A B

A has two unvisited neighbors {B, C}

Again, we choose to visit the one which is alphabetically first

Depth First Search: Example

Approach: "visit an adjacent, unvisited node as long as possible,
then backup one edge and look for another vertex to visit, using a depth first search."



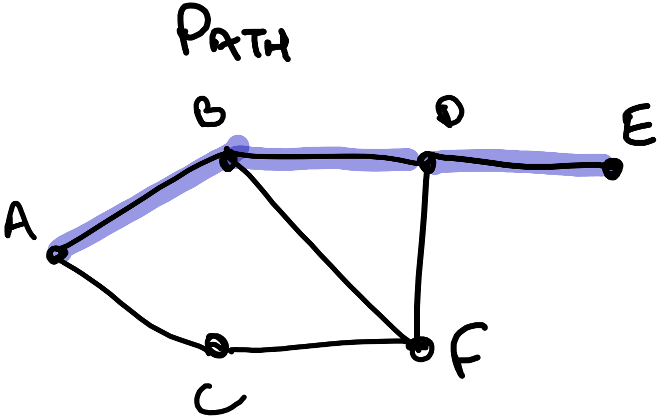
VISITED:

A B D

B has two unvisited neighbors {D, F},
we choose the one which is alphabetically first.

Depth First Search: Example

Approach: "visit an adjacent, unvisited node as long as possible,
then backup one edge and look for another vertex to visit, using a depth first search."



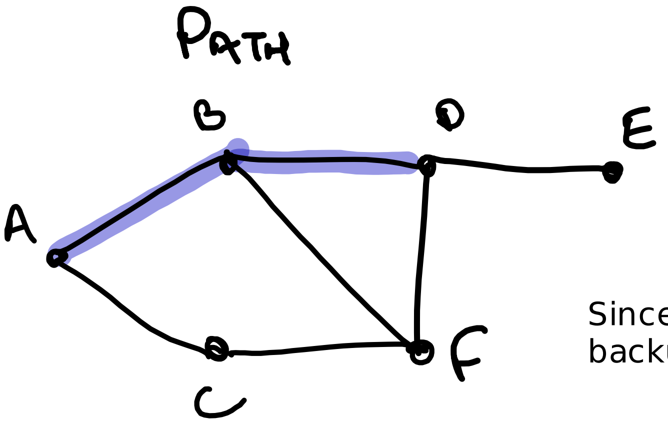
VISITED:

A B D E

D has two unvisited neighbors {E, F},
we choose the one which is alphabetically first.

Depth First Search: Example

Approach: "visit an adjacent, unvisited node as long as possible,
then backup one edge and look for another vertex to visit, using a depth first search."



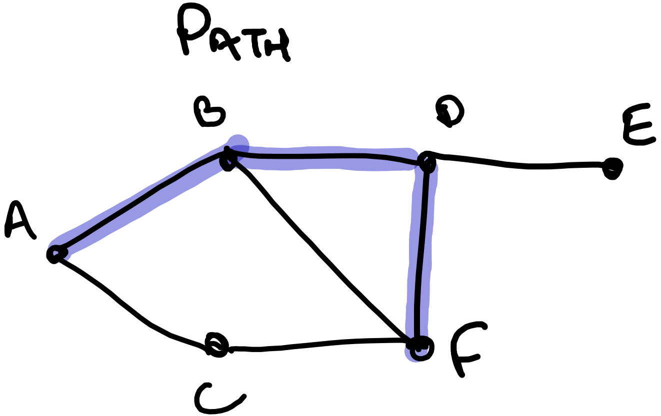
VISITED:

A B D E

Since E has no unvisited neighbors, we
backup our path and repeat the DFS process

Depth First Search: Example

Approach: "visit an adjacent, unvisited node as long as possible,
then backup one edge and look for another vertex to visit, using a depth first search."



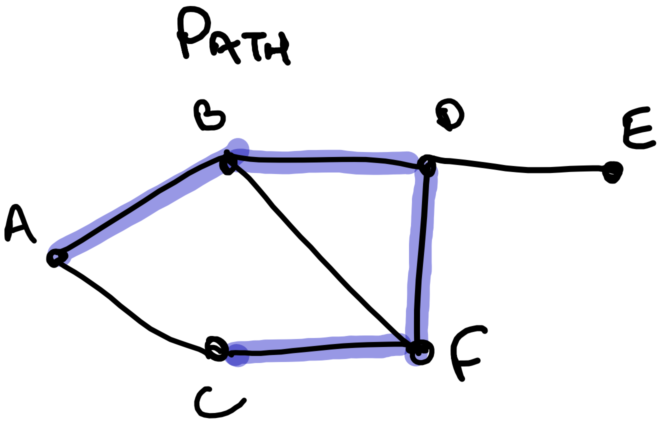
VISITED:

A B D E F

D has 1 unvisited neighbor {F}

Depth First Search: Example

Approach: "visit an adjacent, unvisited node as long as possible,
then backup one edge and look for another vertex to visit, using a depth first search."



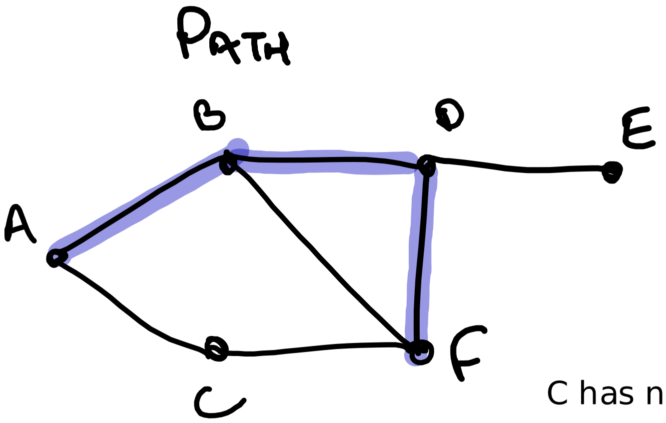
VISITED:

A B D E F C

F has 1 unvisited neighbor {C}

Depth First Search: Example

Approach: "visit an adjacent, unvisited node as long as possible,
then backup one edge and look for another vertex to visit, using a depth first search."



VISITED:

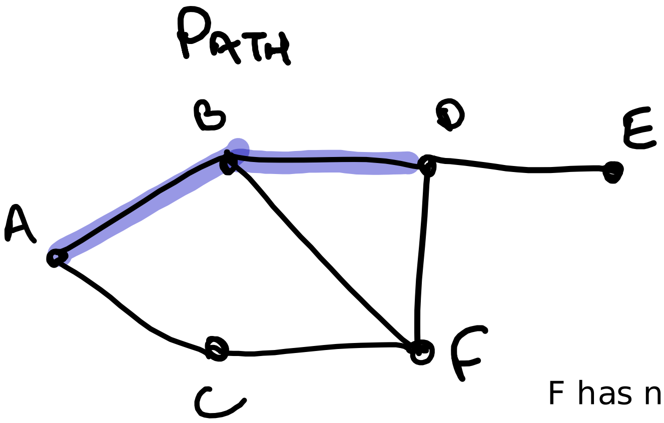
A B D E F C

C has no unvisited neighbors so we backup

(You can tell from the picture we're done ...
the computer can't)

Depth First Search: Example

Approach: "visit an adjacent, unvisited node as long as possible,
then backup one edge and look for another vertex to visit, using a depth first search."



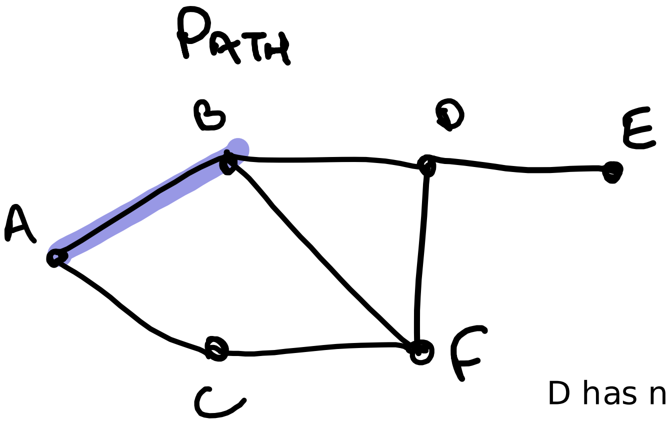
VISITED:

A B D E F C

F has no unvisited neighbors so we backup

Depth First Search: Example

Approach: "visit an adjacent, unvisited node as long as possible,
then backup one edge and look for another vertex to visit, using a depth first search."



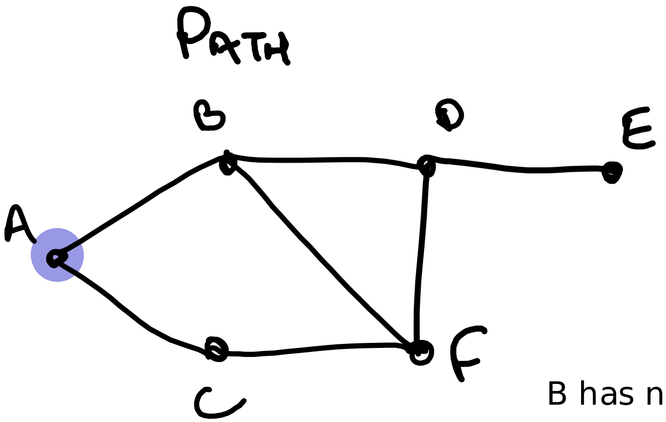
VISITED:

A B D E F C

D has no unvisited neighbors so we backup

Depth First Search: Example

Approach: "visit an adjacent, unvisited node as long as possible,
then backup one edge and look for another vertex to visit, using a depth first search."



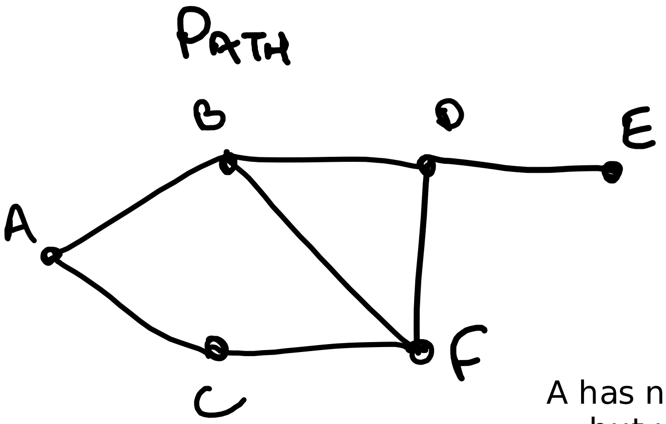
VISITED:

A B D E F C

B has no unvisited neighbors so we backup

Depth First Search: Example

Approach: "visit an adjacent, unvisited node as long as possible,
then backup one edge and look for another vertex to visit, using a depth first search."



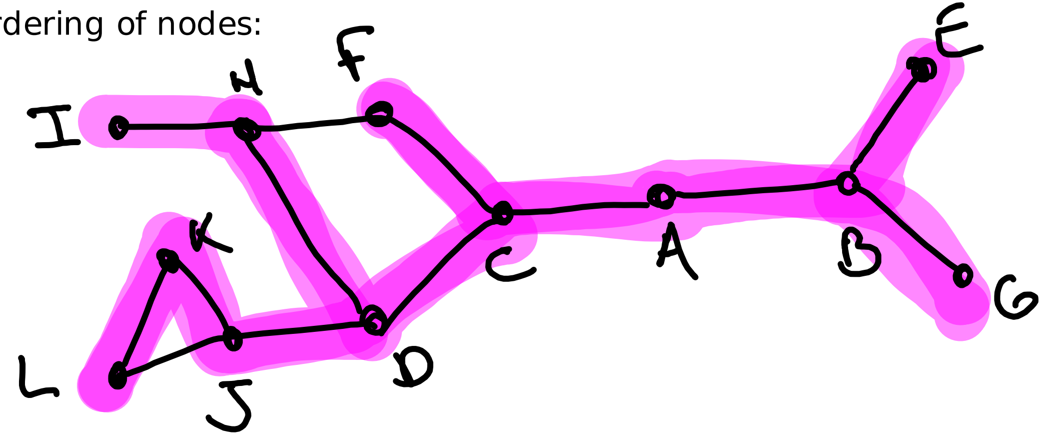
VISITED:

A B D E F C

A has no unvisited neighbors so we backup ...
... but we can't backup as A was our starting node.
DFS is complete

In Class Activity: Depth First Search

Give the DFS ordering of nodes:
- starting at A
- starting at H
- starting at G



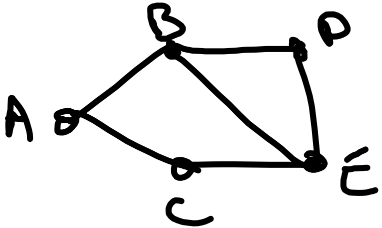
DFS start @ a: ABEG CDHF IJKL
DFS start @ h: HDCA BEGF JKLI
DFS start @ g: GBAC DHFI JKLE

BFS / DFS: Why did we do this again?

- BFS/DFS gives you the largest, connected subgraph
  - "What are all the cities I can get to taking flights from only one airline?"
  - computer can tell if a graph is connected
  - one run gives a connected component ... repeat again for others

- DFS detects cycles in a graph
  - cycle exists if and only if we bump into a neighbor which has already been visited

- BFS orders all nodes from nearest to furthest starting point



BFS ORDERING: A B C D E

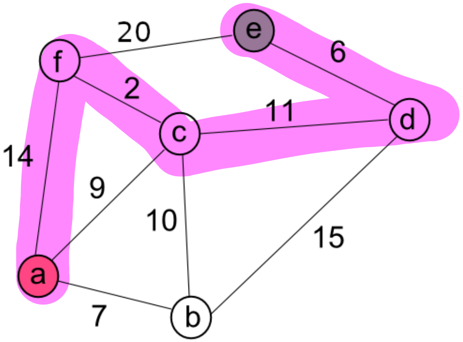
PATH LENGTH FROM A: 0 1 1 2 2

- Comp Sci Education:
  - They're very similar to many other graph algorithms
  - They can be built recursively (a function which calls itself). super useful pattern

Shortest Path Problem

What path (sequence of unique, adjacent edges) has the lowest total weight from a to e?

Motivation: Suppose each node is a location and the edges weights are times to travel between the location. The shortest path gets us from a to e quickest



An example path (not shortest):

a → f → c → d → e

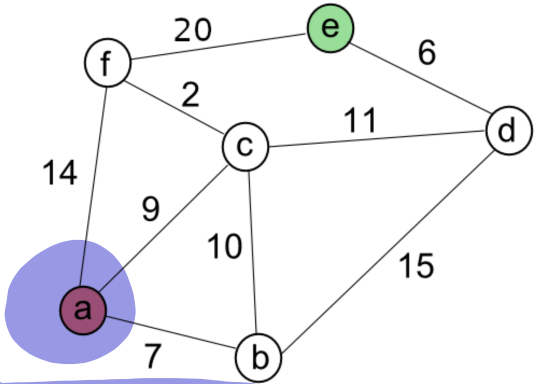
+14 +2 +11 +6 = 33

TOTAL PATH COST

Shortest Path Problem: Dijkstra's Algorithm

What path (sequence of unique, adjacent edges) has the lowest total weight from a to e?
(Assumes all edge weights are non-negative)

Approach: - Track shortest path from a, potentially through a subgraph, to all other nodes
- Add node to subgraph with shortest path weight
- Stop when there is no node outside subgraph with lowest weight to destination



SUBGRAPH

The shortest path from a, potentially through {a}:

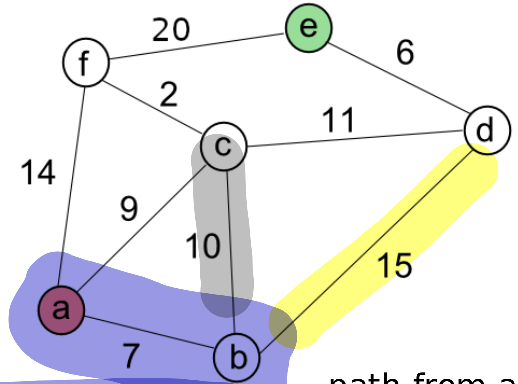
a	b	c	d	e	f
0	7	9			14

The 7 above tells us we can get from a to b at a cost of 7

Shortest Path Problem: Dijkstra's Algorithm

What path (sequence of unique, adjacent edges) has the lowest total weight from a to e?
(Assumes all edge weights are non-negative)

Approach: - Track shortest path from a, potentially through a subgraph, to all other nodes
- Add node to subgraph with shortest path weight
- Stop when there is no node outside subgraph with lowest weight to destination



SUBGRAPH

NEW

The shortest path from a, potentially through {a,b}:

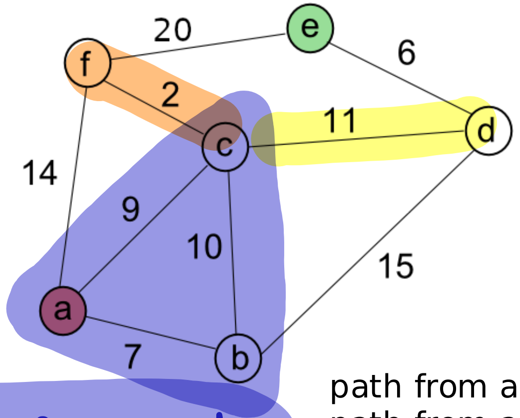
a	b	c	d	e	f
0	7	9	22		14

path from a to c (through b) has cost of 7 + 10: worse than previous: leave it.
path from a to d (through b) has cost of 7 + 15: better than previous: replace

Shortest Path Problem: Dijkstra's Algorithm

What path (sequence of unique, adjacent edges) has the lowest total weight from a to e?
(Assumes all edge weights are non-negative)

Approach: - Track shortest path from a, potentially through a subgraph, to all other nodes
- Add node to subgraph with shortest path weight
- Stop when there is no node outside subgraph with lowest weight to destination



SUBGRAPH

NEW

The shortest path from a, potentially through {a,b,c}:

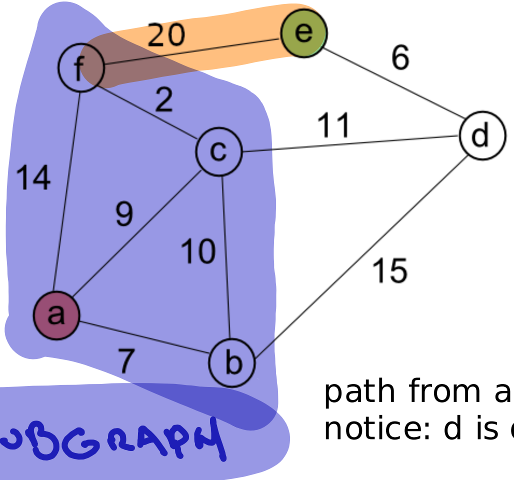
a	b	c	d	e	f
0	7	9	20		11

path from a to f (through c) has cost of 9 + 2: better than previous: replace.
path from a to d (through c) has cost of 9 + 11: better than previous: replace

Shortest Path Problem: Dijkstra's Algorithm

What path (sequence of unique, adjacent edges) has the lowest total weight from a to e?
(Assumes all edge weights are non-negative)

Approach:
- Track shortest path from a, potentially through a subgraph, to all other nodes
- Add node to subgraph with shortest path weight
- Stop when there is no node outside subgraph with lowest weight to destination



NEW

The shortest path from a, potentially through {a,b,c,f}:

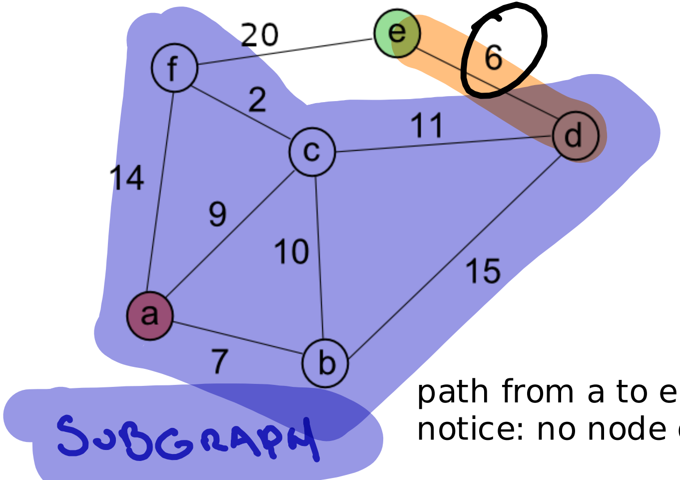
a	b	c	d	e	f
0	7	9	20	31	11

path from a to e (through f) has cost of 11 + 20: better than previous: replace
notice: d is outside subgraph w/ lower weight … we should continue

Shortest Path Problem: Dijkstra's Algorithm

What path (sequence of unique, adjacent edges) has the lowest total weight from a to e?
(Assumes all edge weights are non-negative)

Approach: - Track shortest path from a, potentially through a subgraph, to all other nodes
- Add node to subgraph with shortest path weight
- Stop when there is no node outside subgraph with lowest weight to destination



NEW

The shortest path from a, potentially through {a,b,c,f,d}:

a	b	c	d	e	f
0	7	9	20	26	11

path from a to e (through d) has cost of 20 + 6: better than previous: replace
notice: no node outside subgraph has lower weight, we may stop

In this example, we visited all nodes but our destintion.

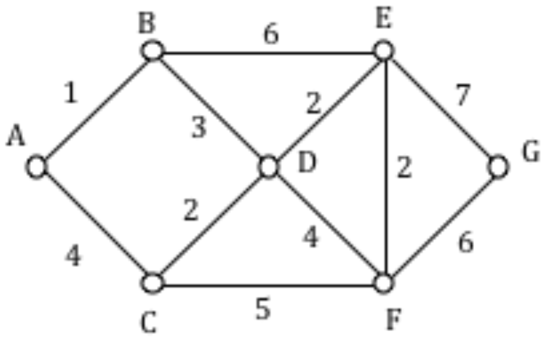
In others, we needn't visit all nodes but our destination.
(stopping early = less computation = faster runtime = good news!)

In Class Activity: Dijkstra's Algorithm

Find the shortest path weight from A to G.

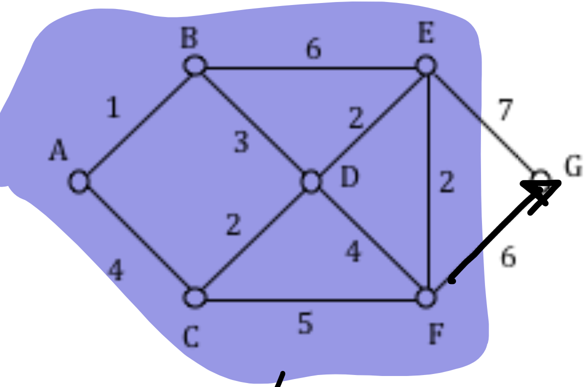
Please write out each step of your algorithm (erasing work makes it tough to find errors!)

- clearly label nodes which nodes are in the "sub-graph" (those you've visited)
- write path weight from starting node to all others through the subgraph (i.e. previous table)



Visited?

A	B	C	D	E	F	G
0	1	4				



Visited?

A	B	C	D	E	F	G
✓						
0	1	4				

Visited?

A	B	C	D	E	F	G
✓	✓					
0	1	4	4	7		

Visited?

A	B	C	D	E	F	G
✓	✓	✓				
0	1	4	4	7	9	

Visited?

A	B	C	D	E	F	G
✓	✓	✓	✓			
0	1	4	4	6	8	

Visited?

A	B	C	D	E	F	G
✓	✓	✓	✓	✓		
0	1	4	4	6	8	13

Visited?

A	B	C	D	E	F	G
✓	✓	✓	✓	✓	✓	
0	1	4	4	6	8	13

$$P(X=K) = \binom{N}{K} p^K (1-p)^{N-K}$$

PROB EITHER TEAMS SCORES K GOALS FROM N ATTEMPTS

$$= \binom{2}{2} .6^2 (1-.6)^0 = .36$$

$$P(X=2) \cdot P(X=2) = .36^2$$

$$P((0,0)) + P((1,1)) + P((2,2))$$