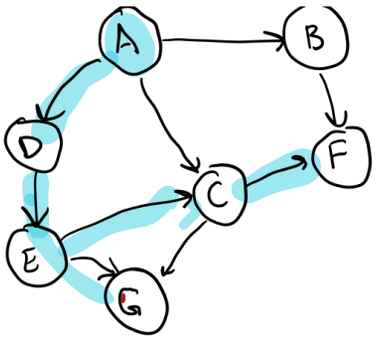
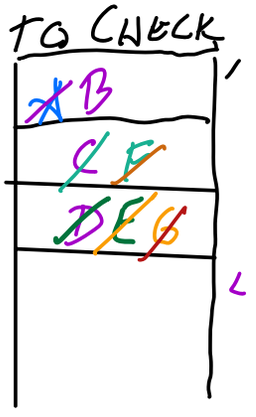


A → F ?

VISITED A, B, C, D, E, G, F

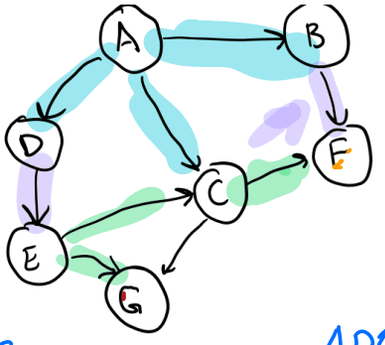


DFS: TO CHECK

DFS: Removing most recent item to have been added to list
This is called "last in/first out" (LIFO) order
=> This is a <u>stack</u>
(which happens to be implemented with a linked list)

DFS (DEPTH-FIRST SEARCH)

VISITED A, B, C, D, F, G, E



BFS: TO CHECK

ADD

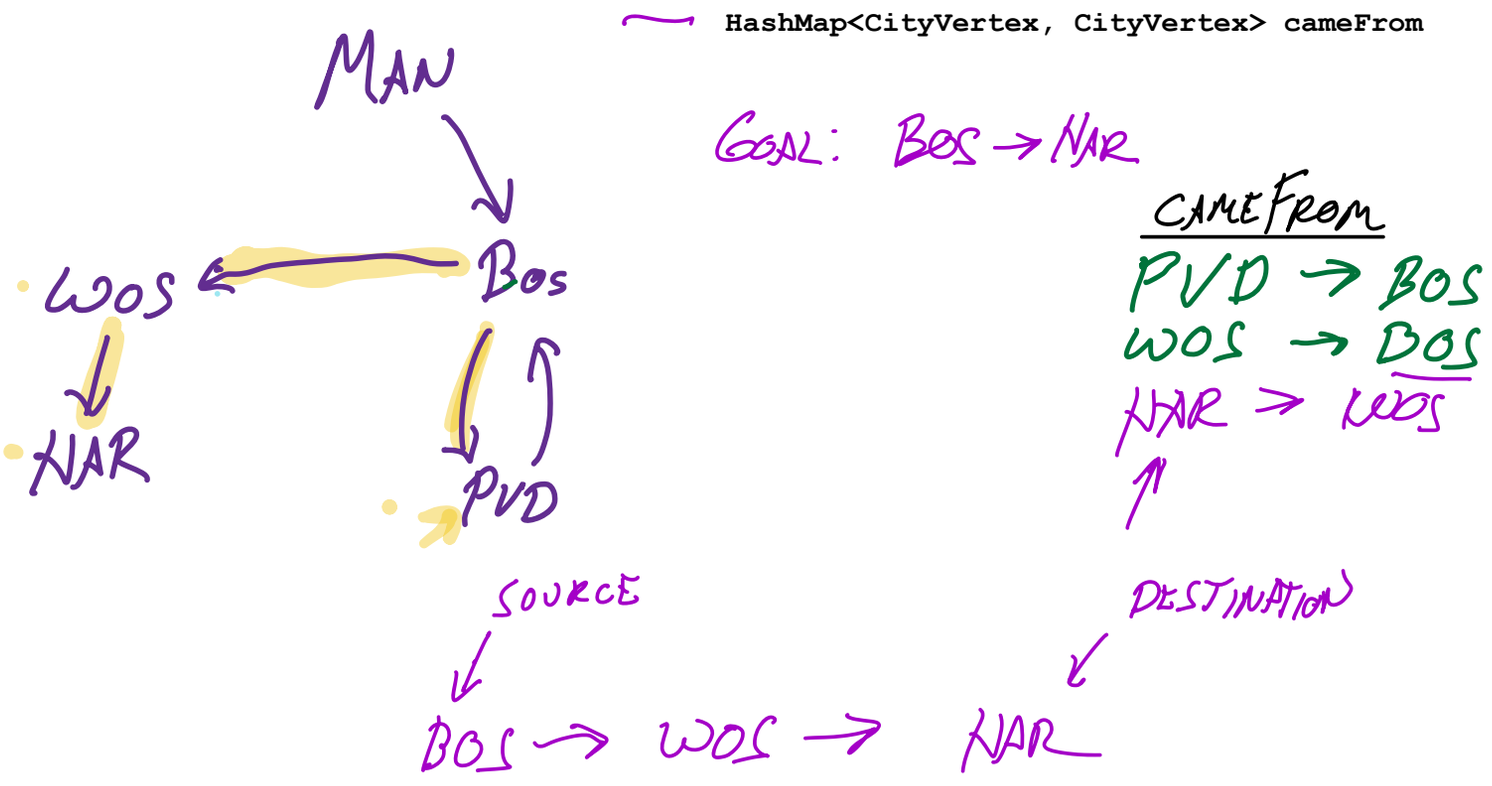
REMOVE

BFS: Removing the least recent item to have been added to the list
This is called "first in/first out" (FIFO) order
=> This is a <u>queue</u>
(Which also happens to be implemented with a linked list)

BFS (BREADTH-FIRST SEARCH)

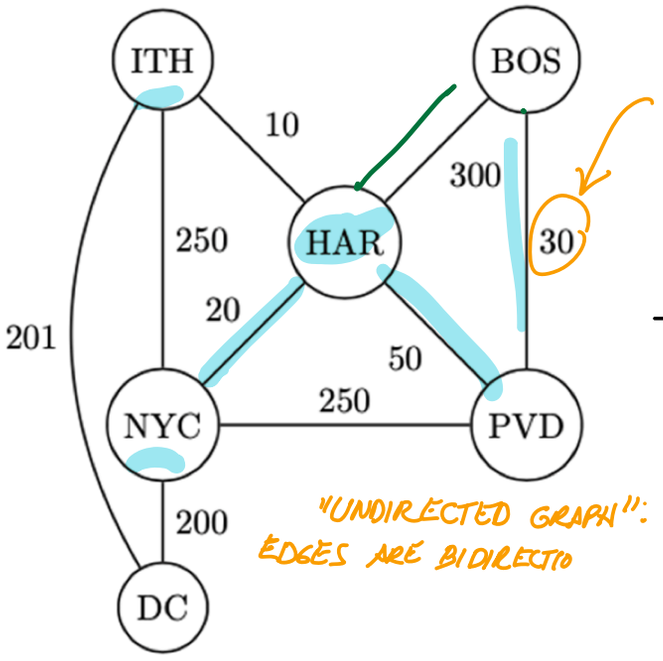
What if canReach didn't return a boolean? What if we wanted to know what path we found to get to the destination?

Use a HashMap to record which node we came from when we considered each node—(for more details, see typed notes)



A look at Dijkstra's algorithm

CHEAPEST WAY BOS → NYC?



"COSTS", "WEIGHTS"

"UNDIRECTED GRAPH": EDGES ARE BIDIRECTIO

TO CHECK PQ ← DISTANCE FROM SOURCE

NODE	BEST DISTANCE	CAME FROM
BOS	0	
DC	~~∞~~ 291	PVD → BOS ②
HAR	~~∞~~ ~~300~~ 80 ①	HAR → ~~BOS~~ PVD NYC → ~~PVD~~ HAR
ITH	~~∞~~ 90 (See note 1, 2)	ITH → HAR
NYC	~~∞~~ ~~280~~ 100	DC → ITH
PVD	~~∞~~ 30	

① 30 + 50 < 300

Here, the new cost to HAR is 30+50 (ie. [cost BOS-PVD] + [cost PVD-HAR]). This is less than the cost we have stored to HAR, so we update our cost table and cameFrom

② Since we found that it was cheaper to go to HAR from PVD, we update canFrom to store HAR->PVD. This is how we keep track of the best path we know about to each destination

Dijkstra's algorithm summary:
Choose a source node (BOS), and at every step, keep track of the **best distance you've seen so far** from the source node to every other node
For every step, consider the cheapest places you can reach, and update the costs to the source node based on the neighbor costs

For more info and a step-by-step example, see the typed notes. We'll also talk more about this on Monday.