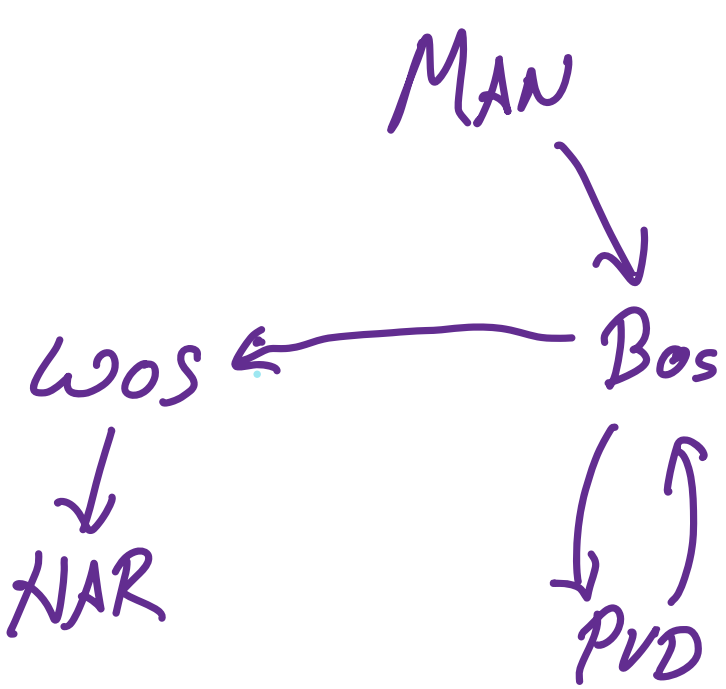


BOS.toCities = [PVD, WOS]
WOS.toCities = [HAR]
PVD.toCities = [BOS]

HOW TO FIND IF WE CAN REACH ONE CITY FROM ANOTHER? FIRST TRY:

```
public boolean canReach(CityVertex dest) {
    if (this.equals(dest)) {
        return true;
    }

    for (CityVertex neighbor : this.toCities) {

        if (neighbor.canReach(dest)) {
            return true;
        }
    }
    return false;
}
```

```
Method calls:
bos.canReach(har)
   pvd.canReach(har)
      bos.canReach(har)
         …
```

If we don't keep track of the cities we've already asked about, then the loop between Boston and Providence may result in an infinite recursion

IDEA: KEEP TRACK OF VISITED VERTICES IN A SET.

```
With helper:
bos.canReachHelper(har, {}) -> true
   pvd.canReachHelper(har, {bos}) -> false
   wos.canReachHelper(har, {bos, pvd}) -> true
      har.canReachHelper(har, {bos, pvd, wos}) -> true
```

Use canReachHelper to keep a HashSet of visited nodes (i.e. nodes that we've already called canReachHelper on)

This example uses recursion to "queue up" neighbors that we should examine.

Instead of using recursion, we can use a list (toCheck) where we add neighbors to the end of the list. (See written notes for a more detailed explanation)

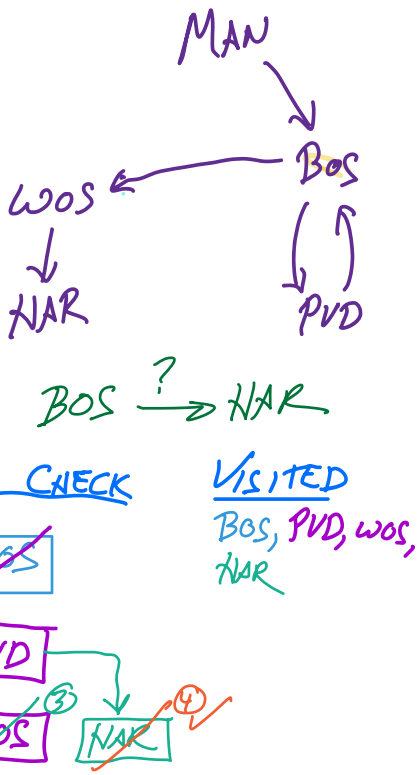
```
public boolean canReachDFS(CityVertex source, CityVertex dest) {
    HashSet<CityVertex> visited = new HashSet<CityVertex>();
    LinkedList<CityVertex> toCheck = new LinkedList<CityVertex>();

    visited.add(source);
    toCheck.addLast(source);

    while (!toCheck.isEmpty()) {
        CityVertex checkingCity = toCheck.removeLast();

        if (checkingCity.equals(dest)) {
            return true;
        }
        // Check all the neighbors
        for (CityVertex neighbor : checkingCity.getNeighbors()) {
            if (!visited.contains(neighbor)) {
                visited.add(neighbor);
                toCheck.addLast(neighbor);
            }
        }
    }
    return false;
}
```

(SEE CODE LINK OR TYPED NOTES FOR FULL VERSION)

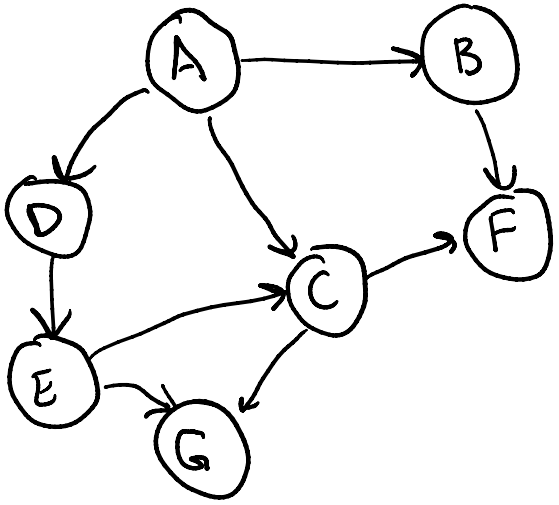


1. Initial step: add Boston
2. Remove and check Boston != Hartford; add neighbors [Providence, Worcester] to end of list
3. Remove from Worcester from end of list, check Worcester != Hartford; add neighbors [Hartford] to end of list
4. Remove Hartford from end of list, check Hartford == Hartford => done!

BFS/DFS example

Wednesday, October 19, 2022 1:18 PM

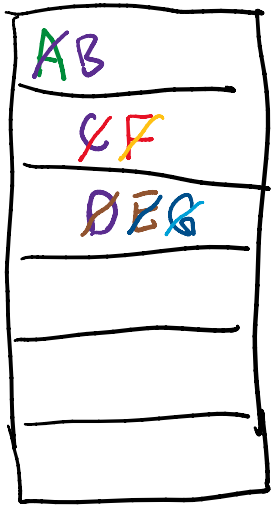
A → F??



Visited:

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

To Check:



DFS
(depth-first search)

Step 1: add A to toCheck and visited
Step 2: remove last element (A) from toCheck; add A's neighbors to toCheck and visited
Step 3: remove last element (D) from toCheck; add D's neighbor to toCheck and visited
Step 4: remove last element (E) from toCheck; add E's unvisited neighbor to toCheck and visited
Step 5: remove last element (G) from toCheck (has no neighbors)
Step 6: remove last element (C) from toCheck; add C's unvisited neighbor to toCheck and visited
Step 7: remove last element (F) from toCheck; we are done because F is the destination