

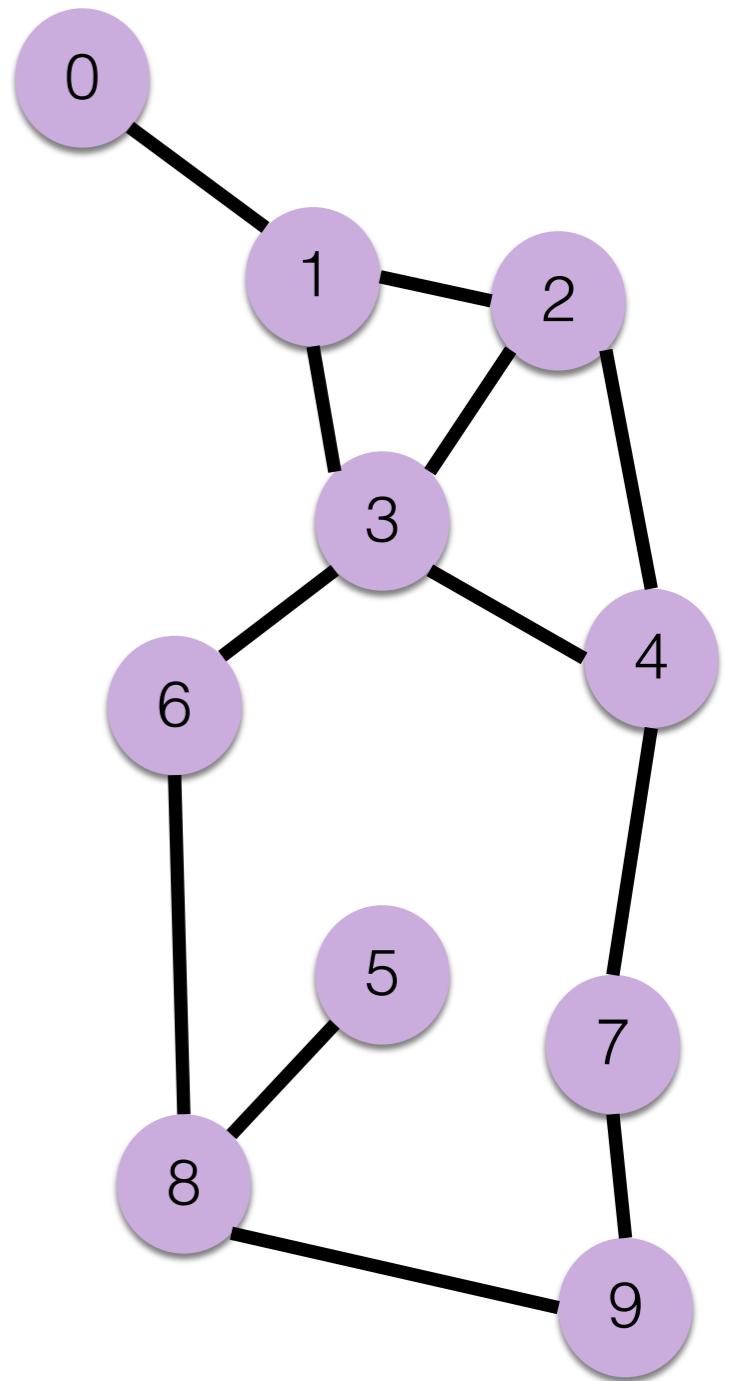
# PG4200: Algorithms And Data Structures

## Lesson 08: Graphs

Prof. Andrea Arcuri

# Graphs

- A set of **vertices** connected by **edges**
- Directed or Undirected graphs
  - If edge from X to Y, implies edge from Y to X?
- Many different problems can be represented with graphs
- Many different algorithms specialized for graphs
- Here just having a very high level view





Map navigation icons: back, forward, zoom, search.

Starting point: Oslo

Destination: Bergen

Leave now

OPTIONS

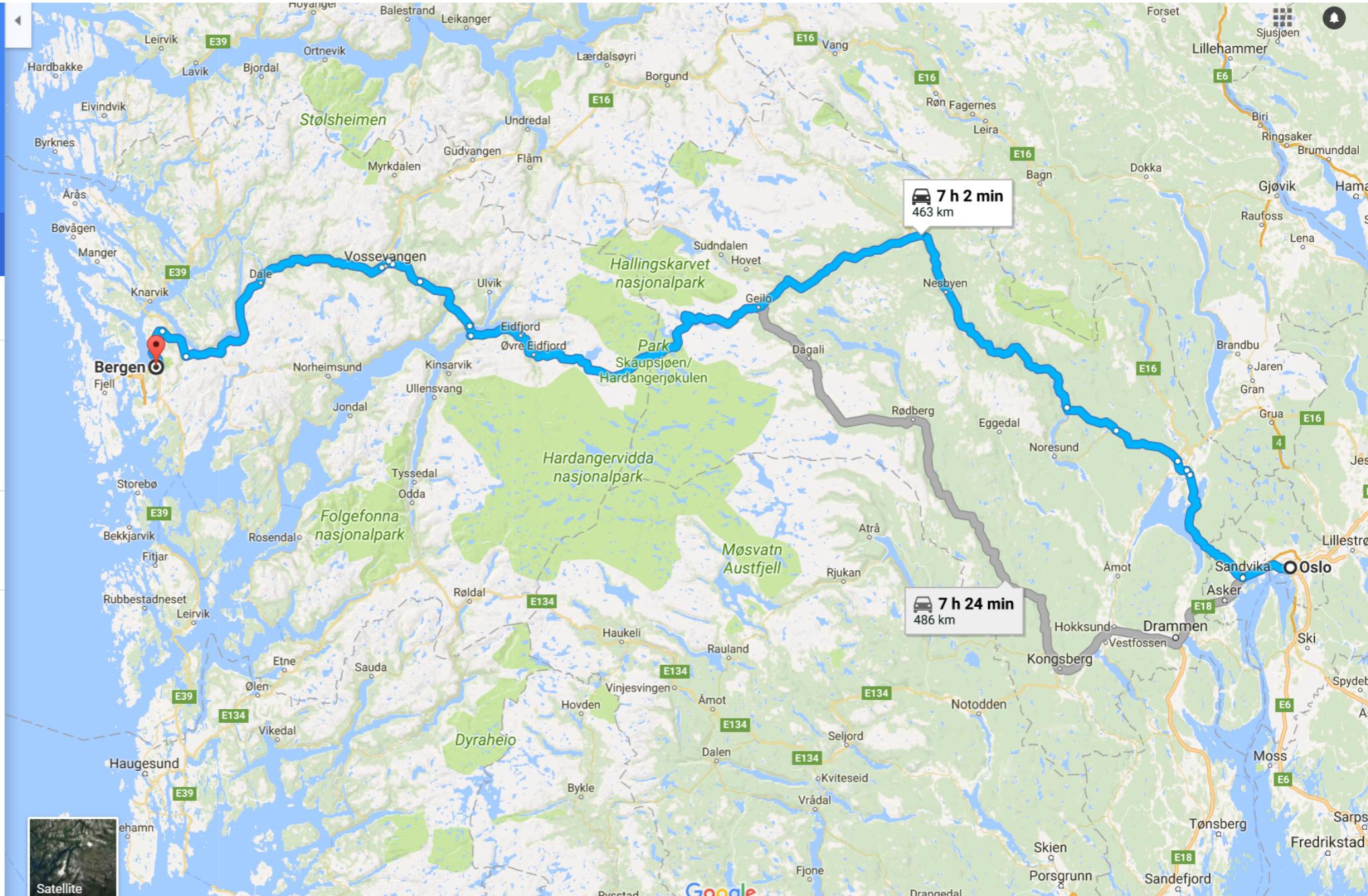
Send directions to your phone

via Rv7  
7 h 2 min  
463 km  
⚠ This route has tolls.

DETAILS

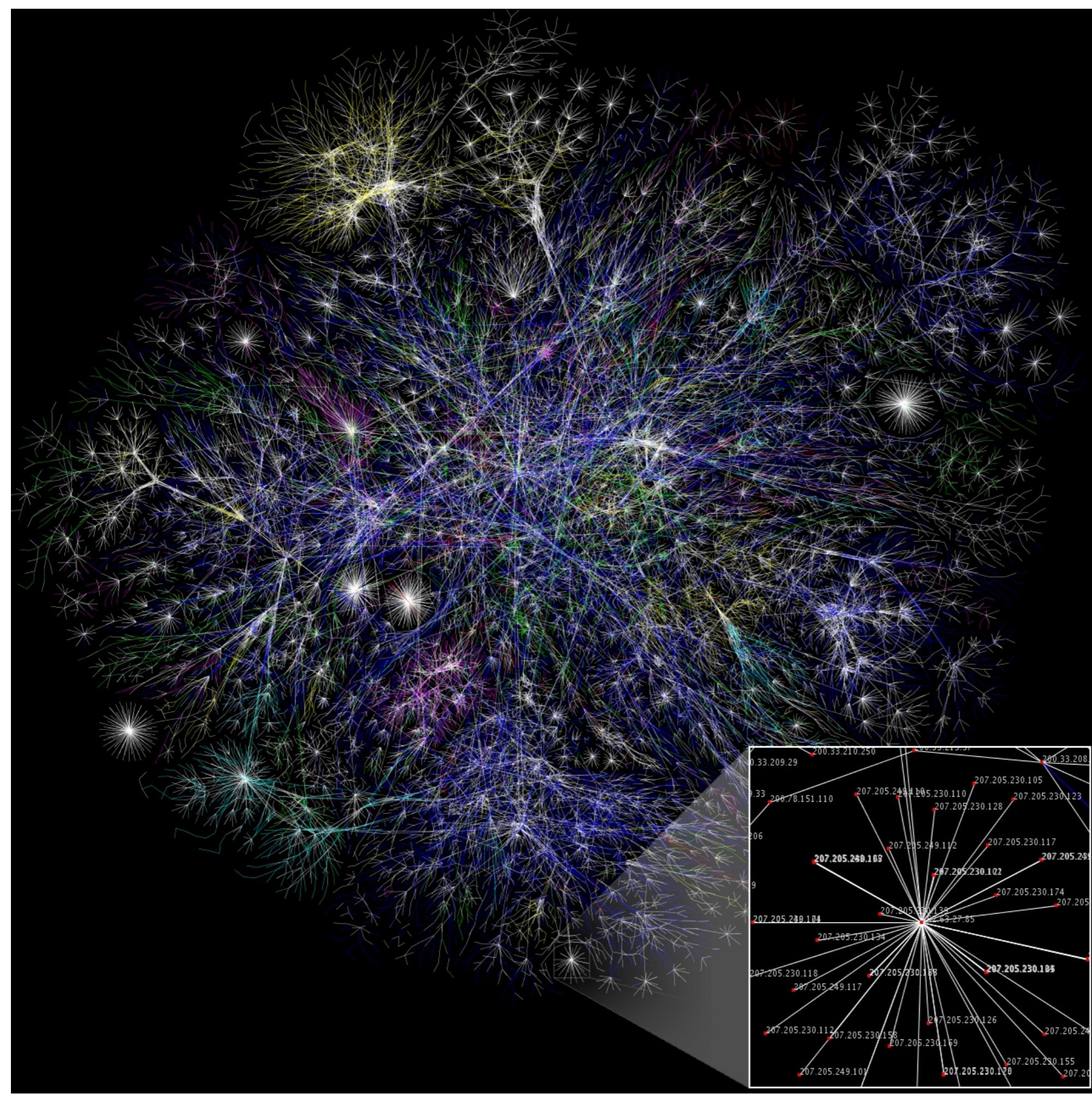
via Fv40  
7 h 24 min  
486 km  
7 h 24 min without traffic

Satellite view



# Friends in a social network



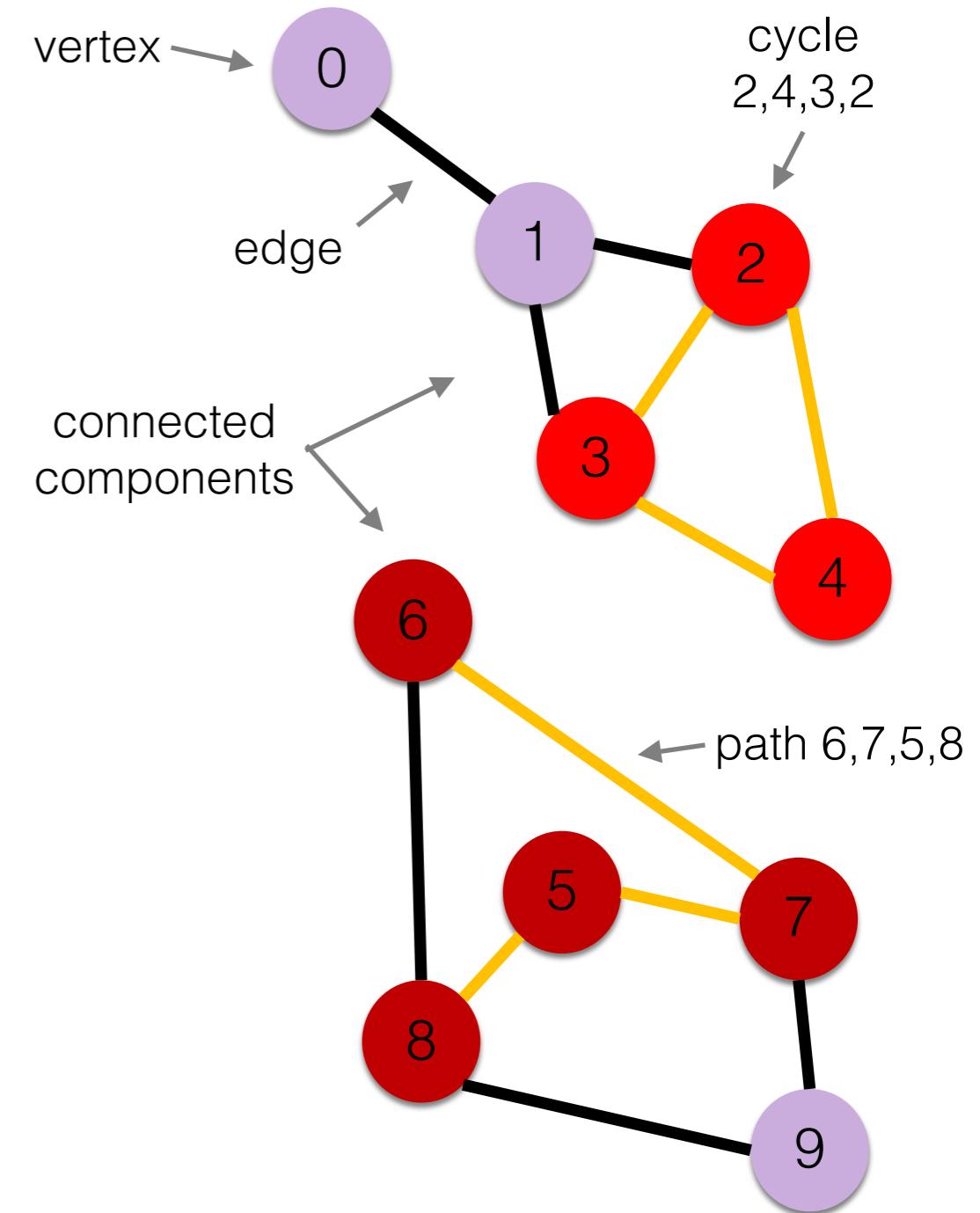


Machines  
connected on  
internet

Note: the picture only show a tiny subset of the whole internet graph...

# Terminology

- **Vertex:** a node, for which we can use a label to identify it
- **Edge:** connection between 2 nodes
- **Path:** a sequence of connected nodes
- **Cycle:** a path starting and ending on the same node
- Note: in a graph, not all nodes are necessarily connected



# How to represent a graph?

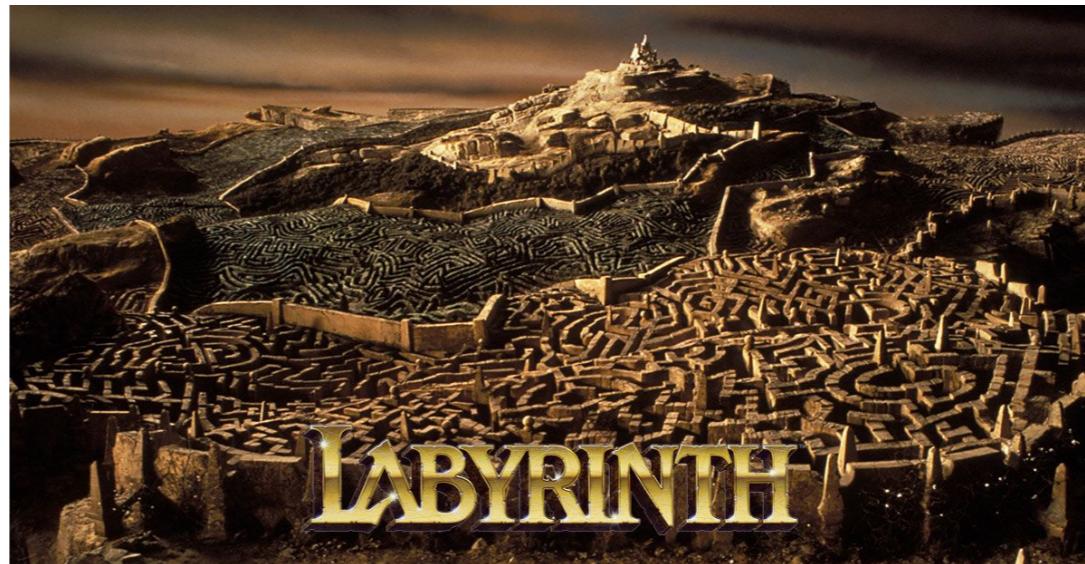
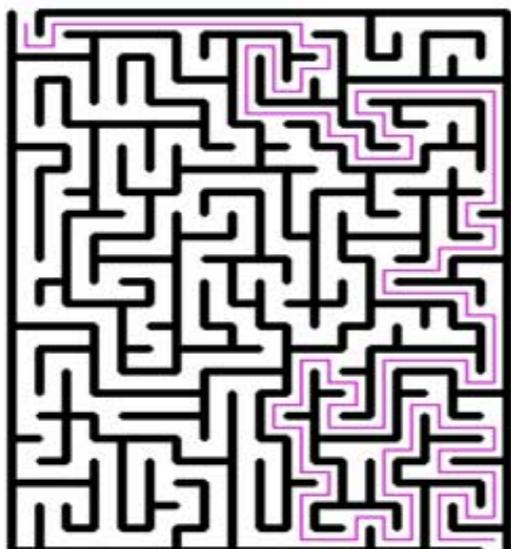
- Vertices can be object with state
  - Eg, name of station, city, friend, IP address
- “Map” from vertex X (key) to a collection of vertices (value) reachable from X
- Note: in this way, do not need objects to explicitly represent edges

# Operations

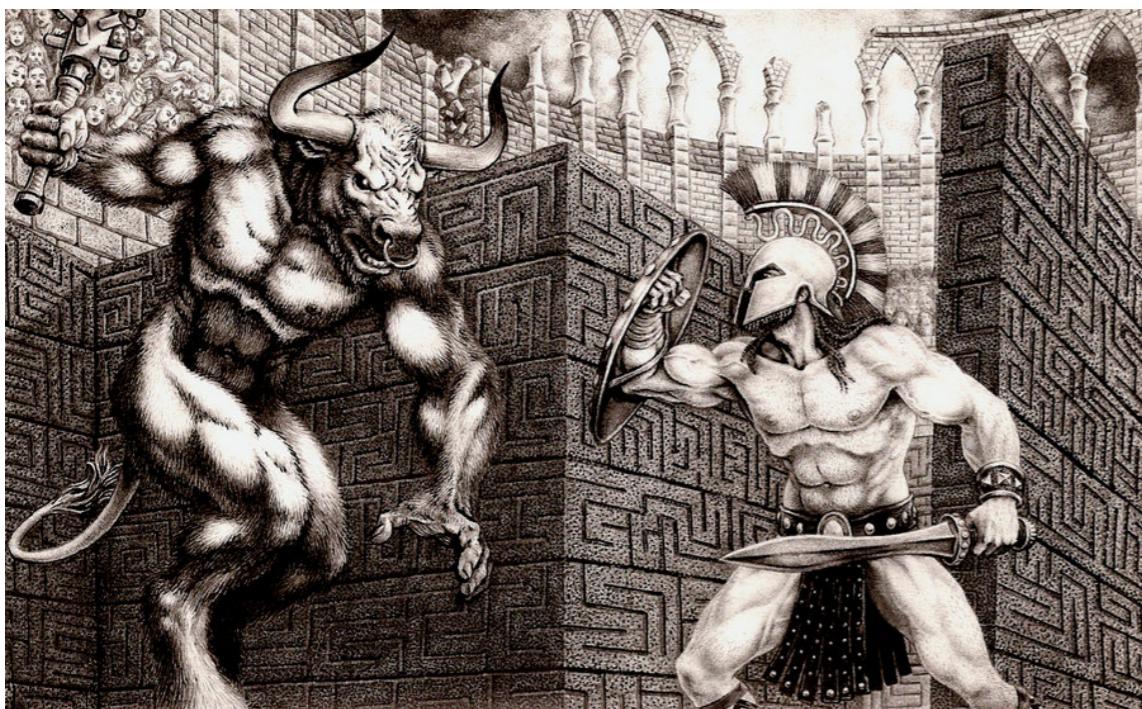
- Given an existing graph, there can be different operations we might need to do
- **Path Finding:** given two vertices (eg, 2 cities), find a path connecting them, and avoiding cycles

# Maze / Labyrinth

- Mazes can be represented with a graph
- Vertices: intersections
- Edges: passages between two intersections
- Find path from starting vertex to the vertex of the exit



# Thesus vs. the Minotaur



- Thesus slays the Minotaur at the center of the labyrinth
- Used a thread to trace back the exit



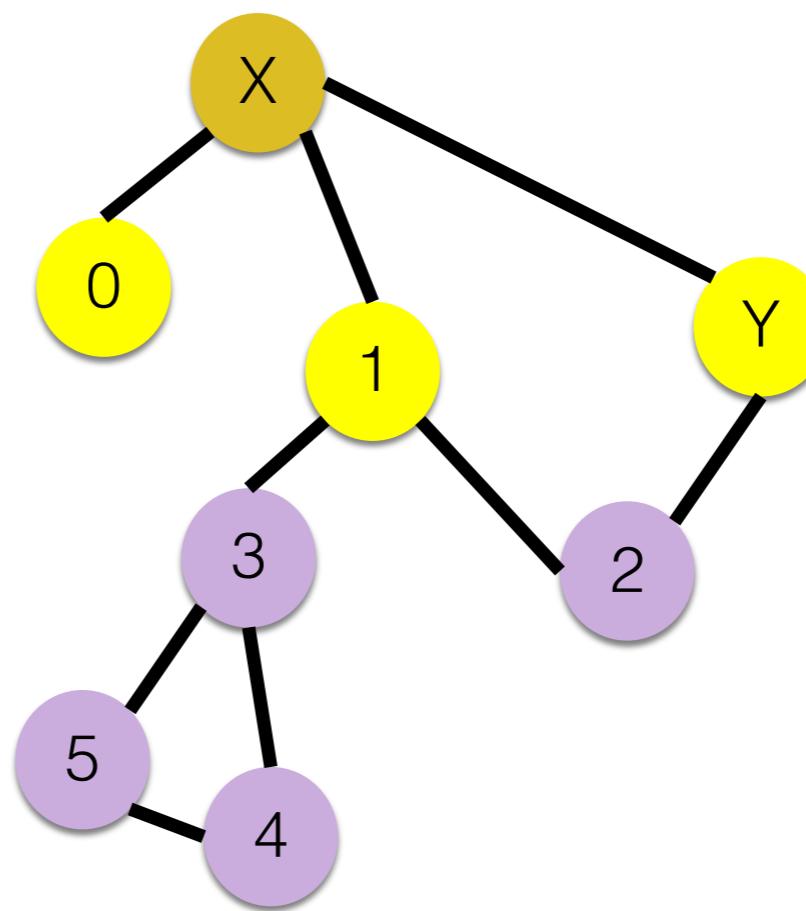
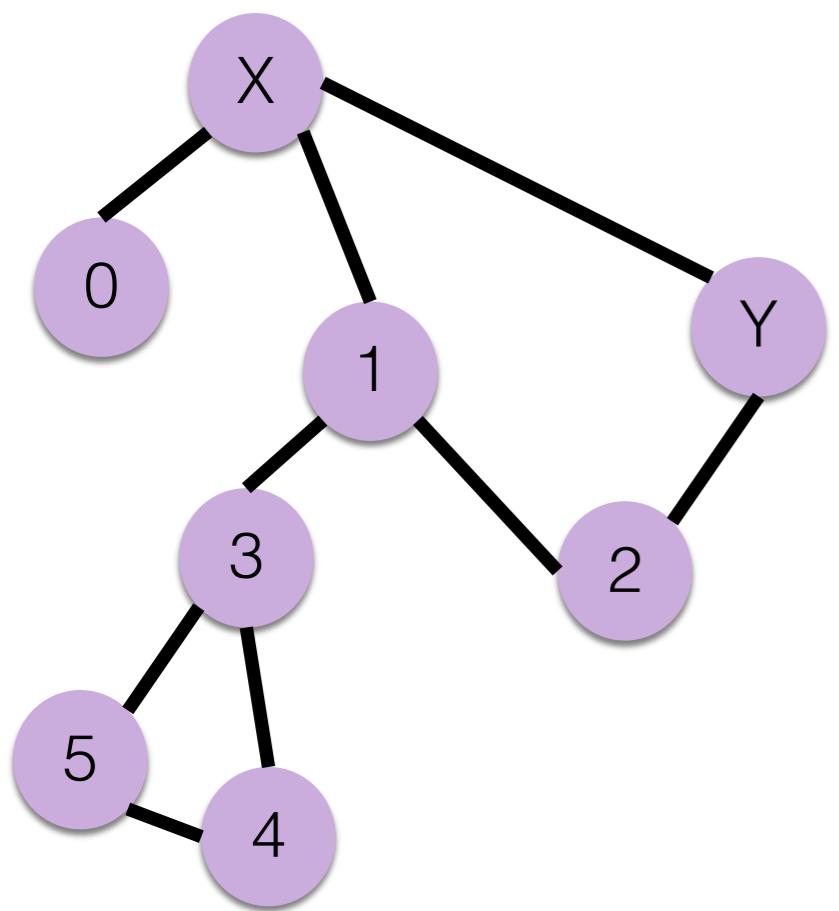
# Trémaux's Algorithm

- Charles Pierre Trémaux, 19th-century
- Method that guarantees to find an exit in a maze
  - Note: talking about actual mazes, not computers...
- Trémaux's Algorithm is an instance of what we now call *Depth-First Search* in graphs

# Depth-First Search (DFS)

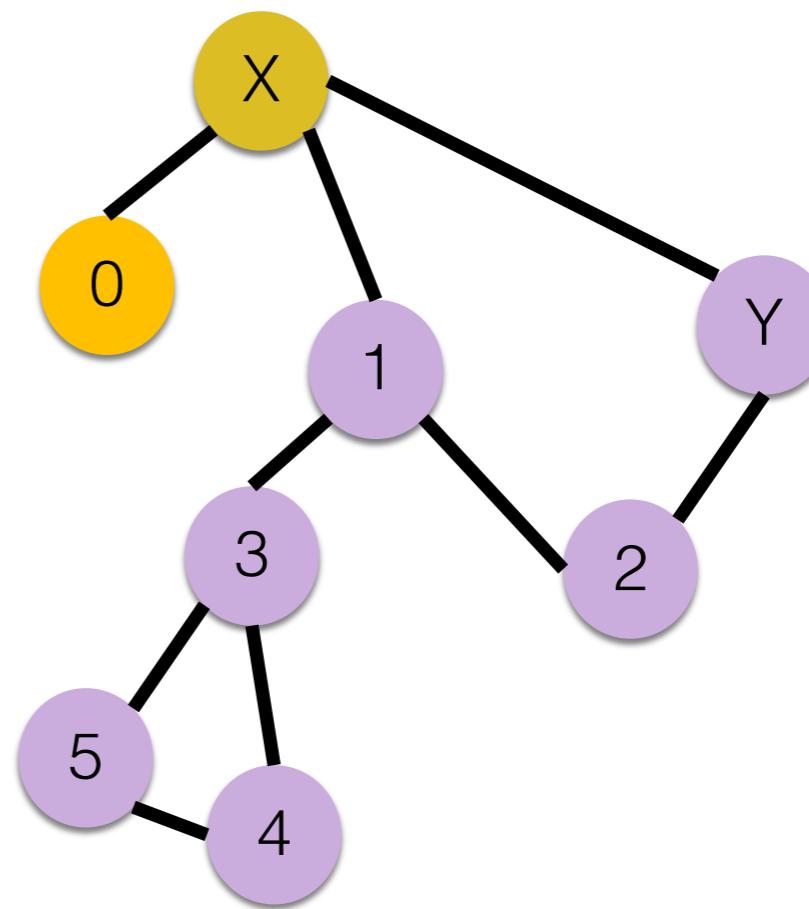
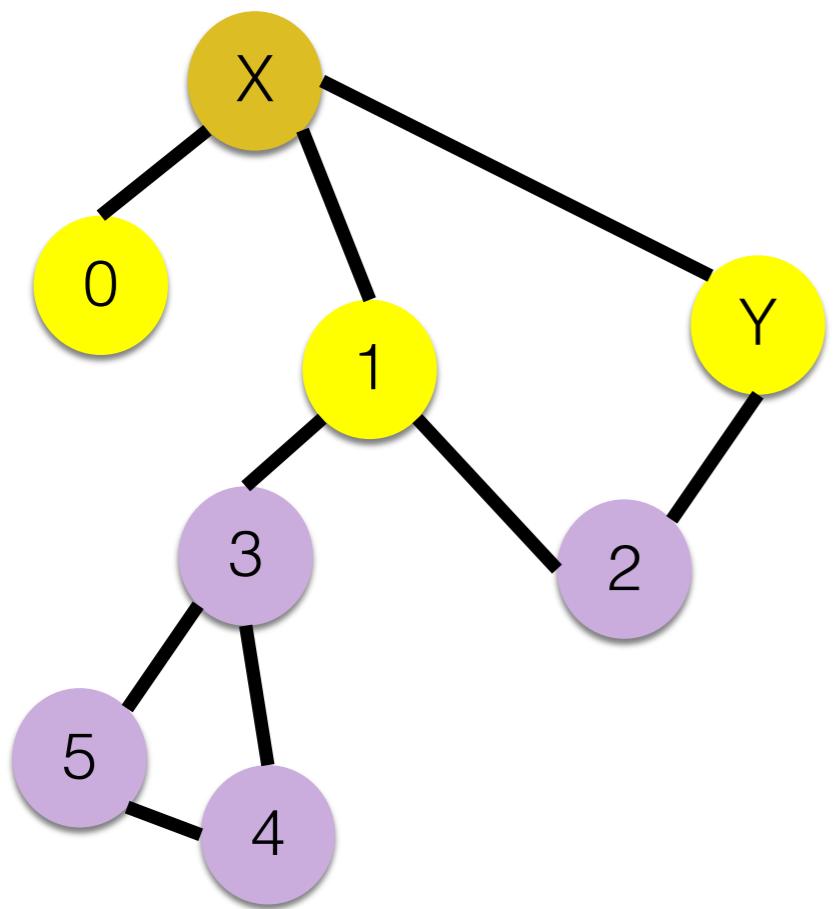
- Try to find a path from vertex X to Y
- Mark current vertex as visited
- *Recursively* look at each connected vertex from the current
  - But skip already marked vertices (eg, by using a set)
- Use stack to represent path from X toward current vertex
  - Push when recursively evaluate a connected vertex
  - Pop when backtrack out of a recursive call

# Example



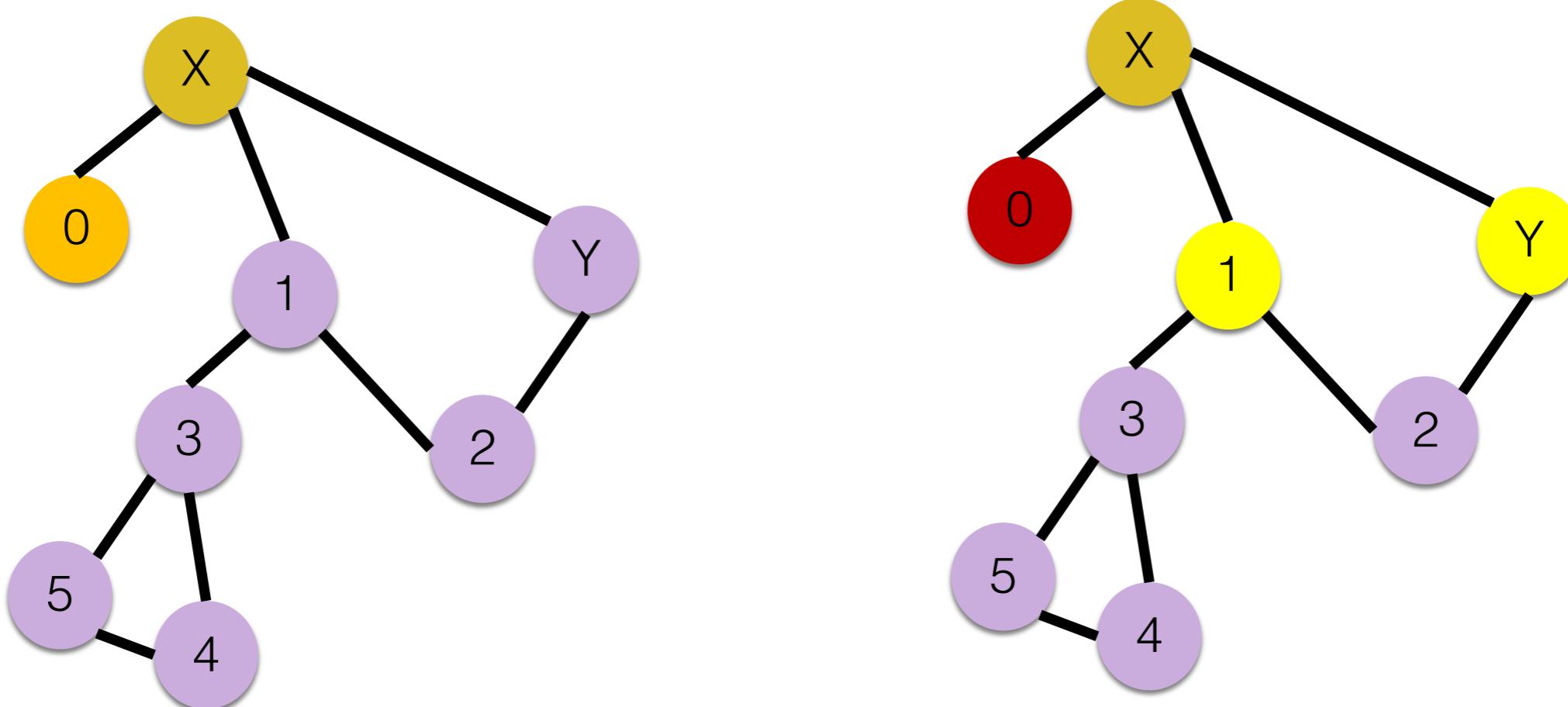
Visited: X  
Stack: X  
Connected: 0, 1, Y

# Recursion on 0



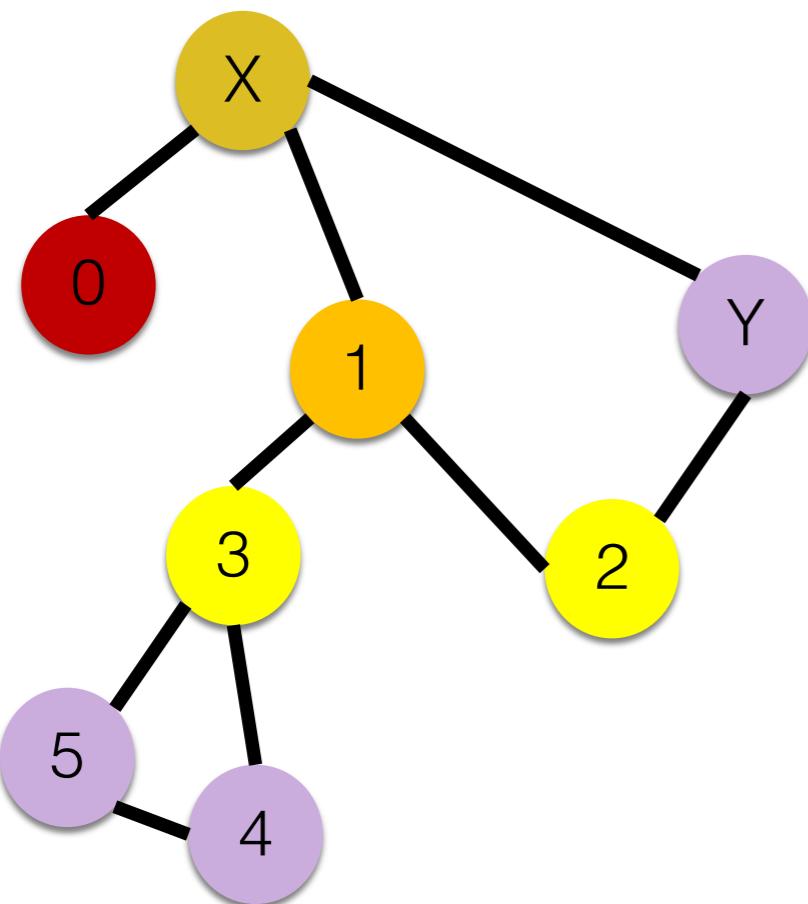
Visited: X, 0  
Stack: X, 0  
Connected: (X)

# Backtrack to X

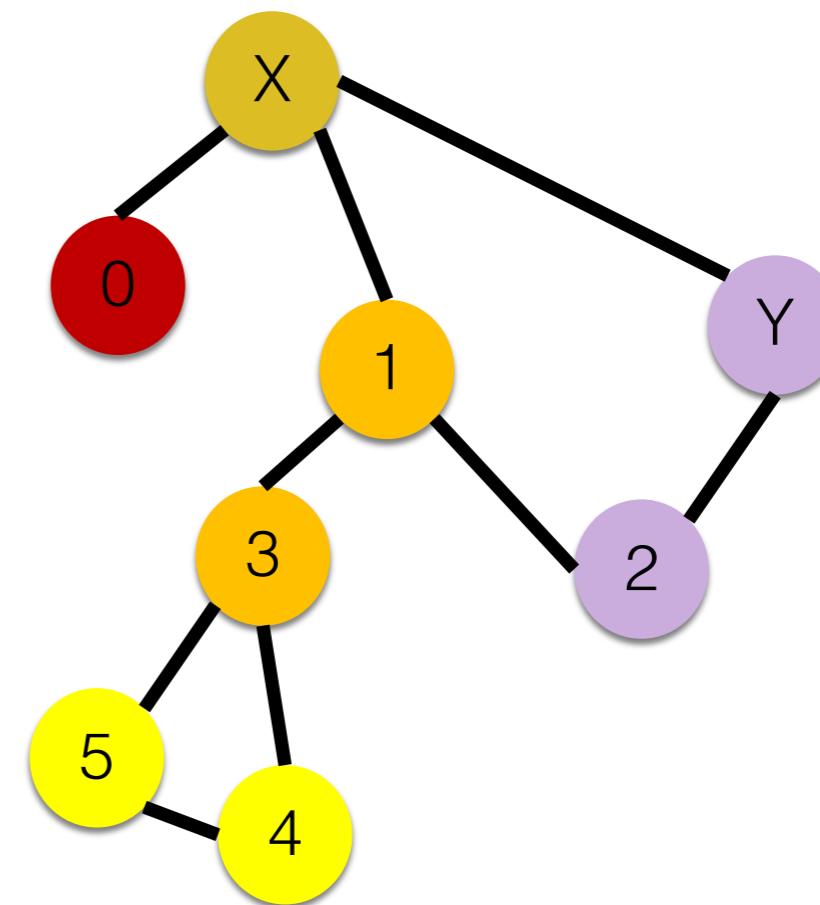


Visited: X, 0  
Stack: X  
Connected: (0), 1, Y

# Evaluating 1 and then 3

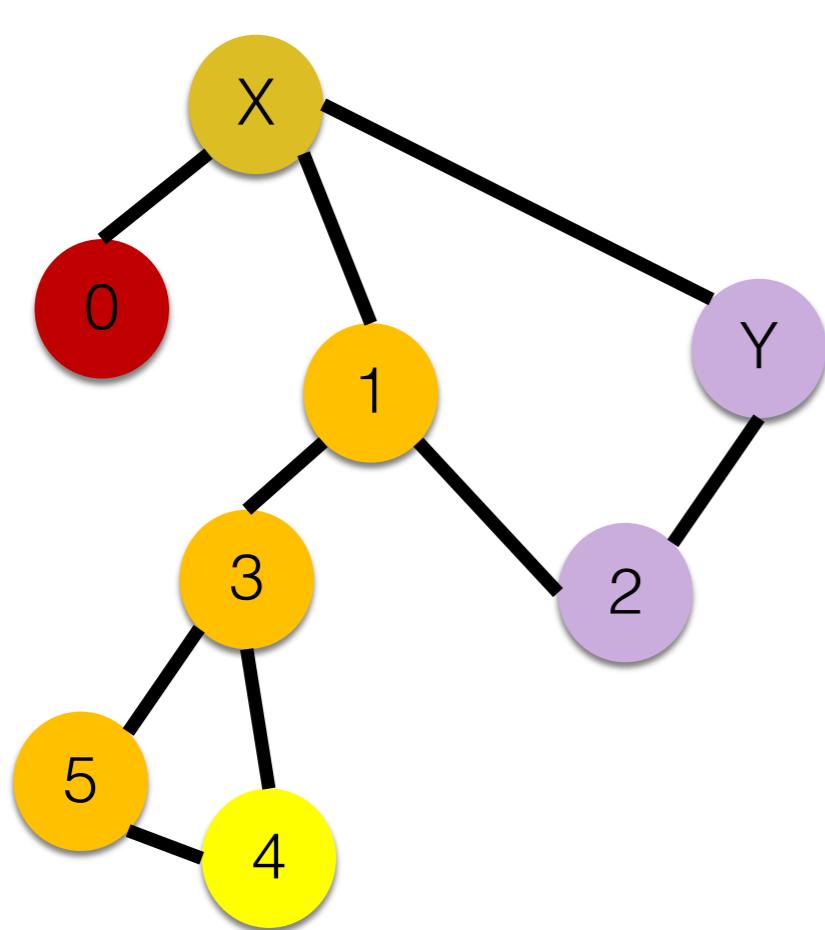


Visited: X, 0, 1  
Stack: X, 1  
Connected: (X), 3, 2



Visited: X, 0, 1, 3  
Stack: X, 1, 3  
Connected: (1), 4, 5

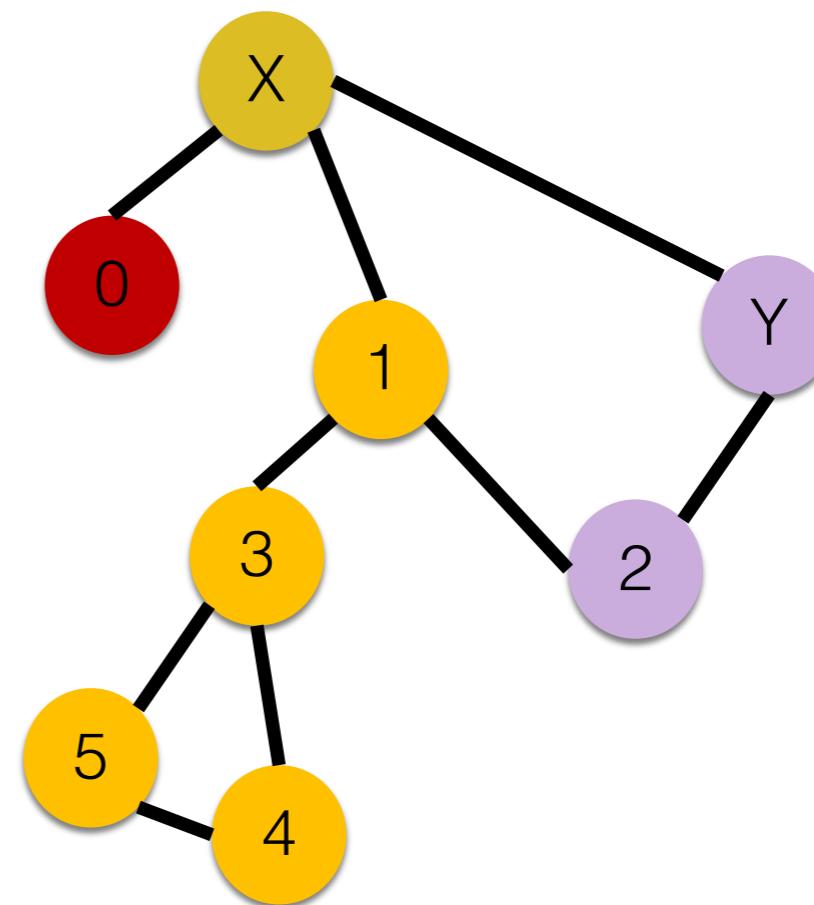
# Evaluating 5 and then 4



Visited: X, 0, 1, 3, 5

Stack: X, 1, 3, 5

Connected: (3), 4

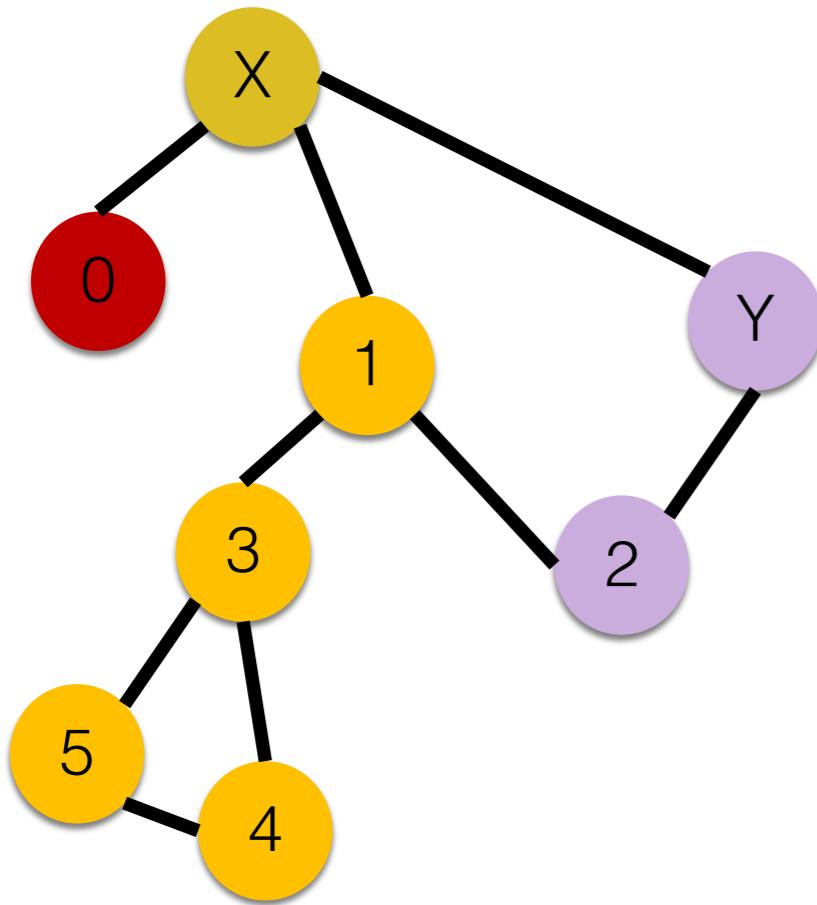


Visited: X, 0, 1, 3, 5, 4

Stack: X, 1, 3, 5, 4

Connected: (3), (5)

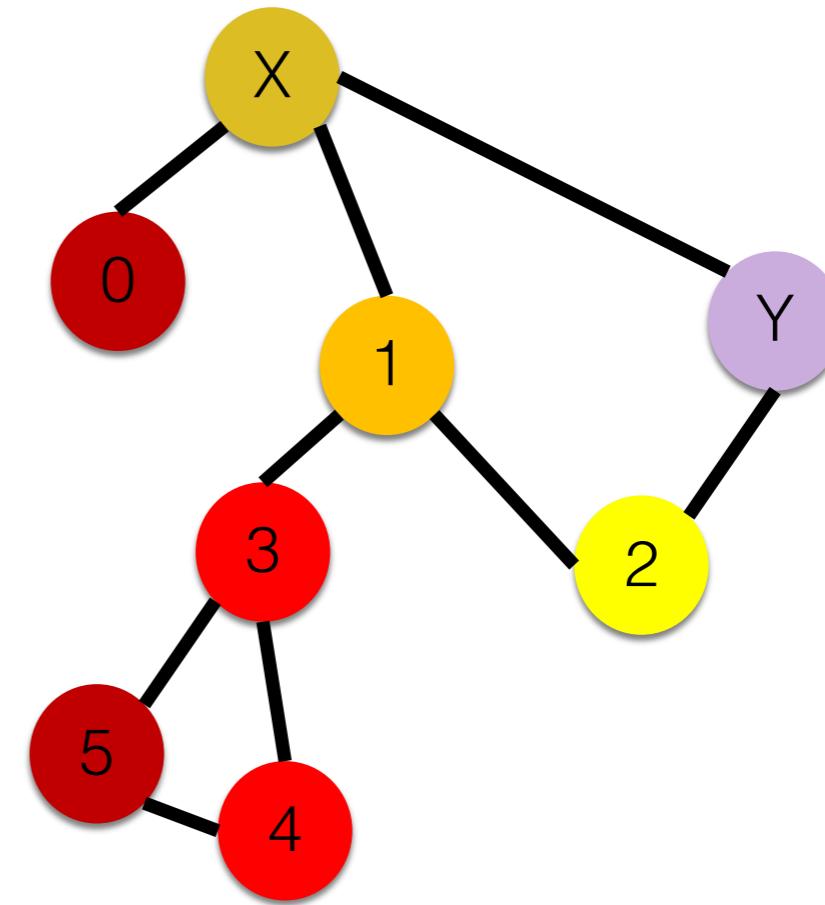
# Backtracking Till 1



Visited: X, 0, 1, 3, 5, 4

Stack: X, 1, 3, 5, 4

Connected: (3), (5)

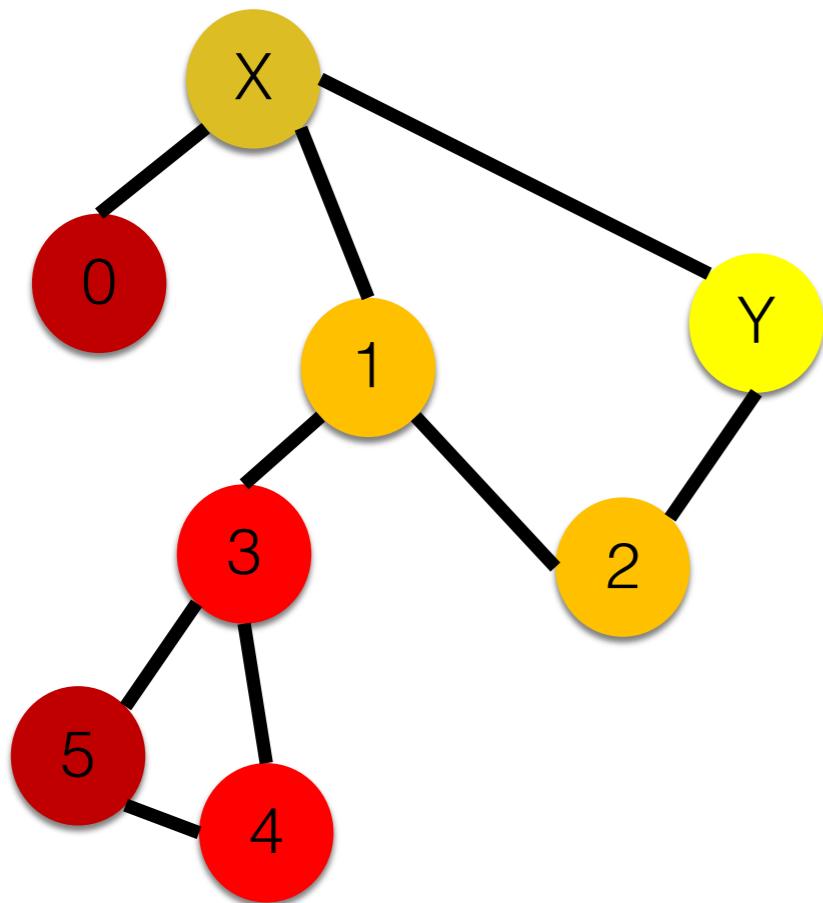


Visited: X, 0, 1, 3, 5, 4

Stack: X, 1

Connected: (X), (3), 2

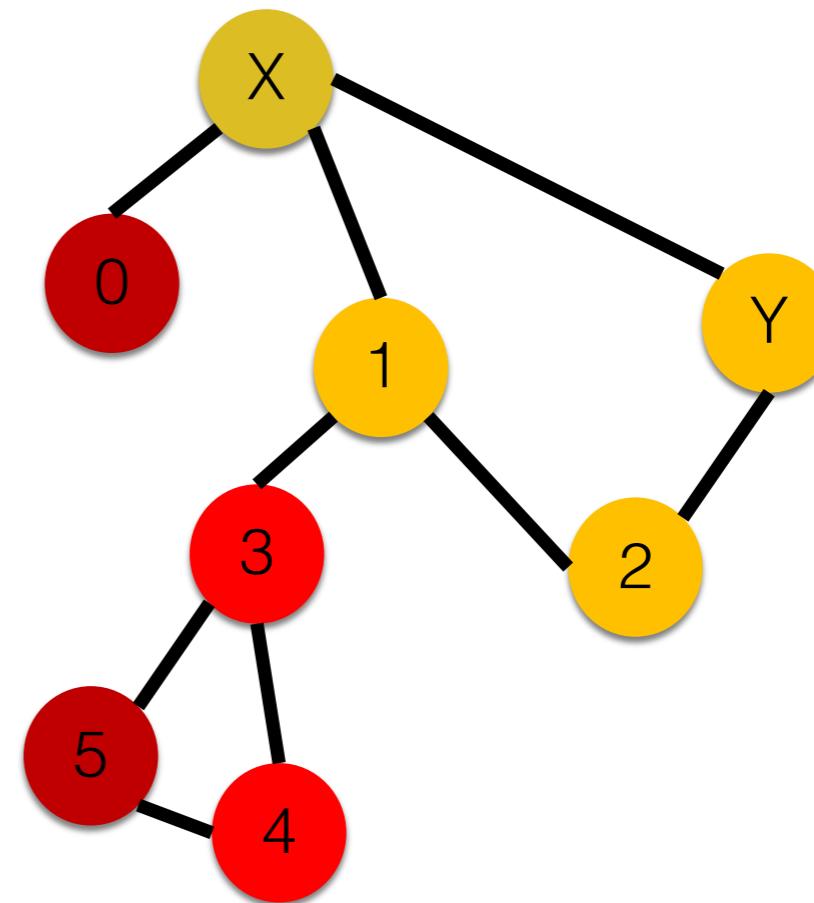
# Evaluating 2 and 1



Visited: X, 0, 1, 3, 5, 4, 2

Stack: X, 1, 2

Connected: (1), Y

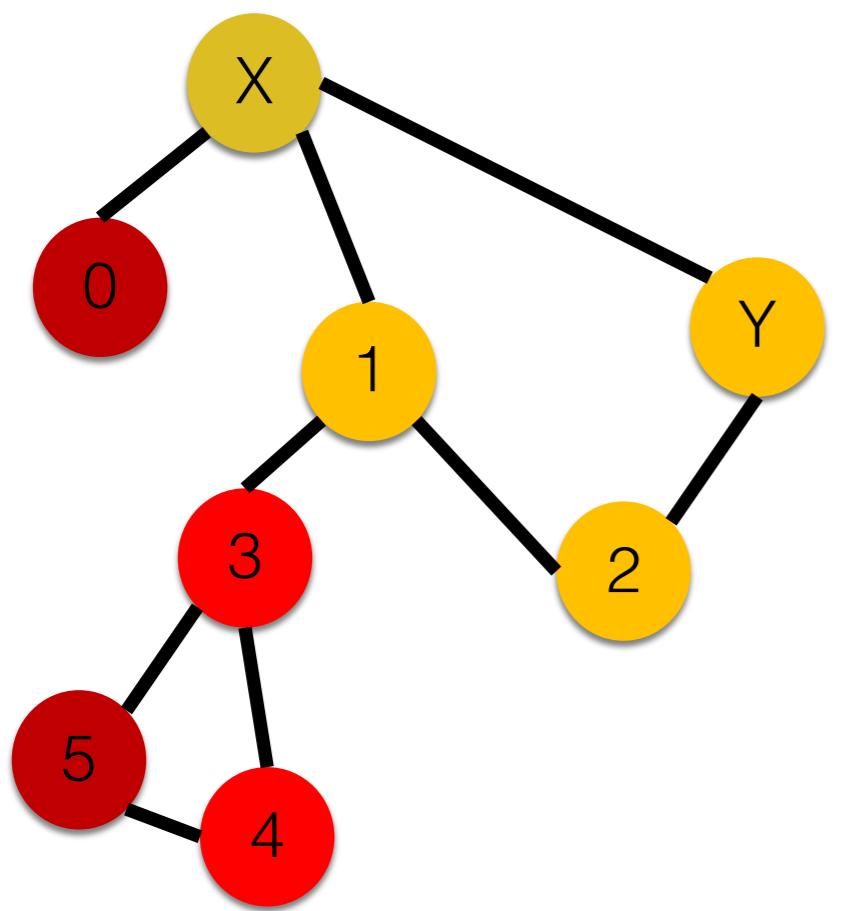


Visited: X, 0, 1, 3, 5, 4, 2, Y

Stack: X, 1, 2, Y

Connected: (X), (2)

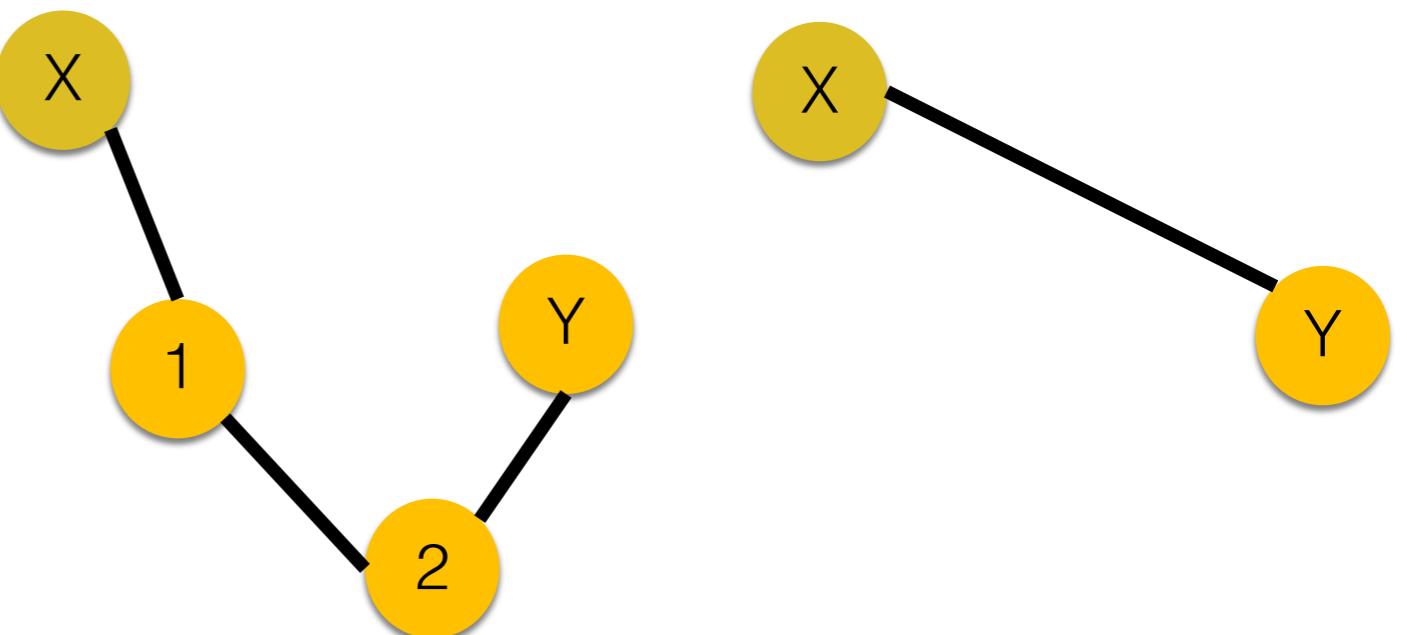
# Final Path X, 1, 2, Y



Visited: X, 0, 1, 3, 5, 4, 2, Y

Stack: X, 1, 2, Y

Connected: (X), (2)

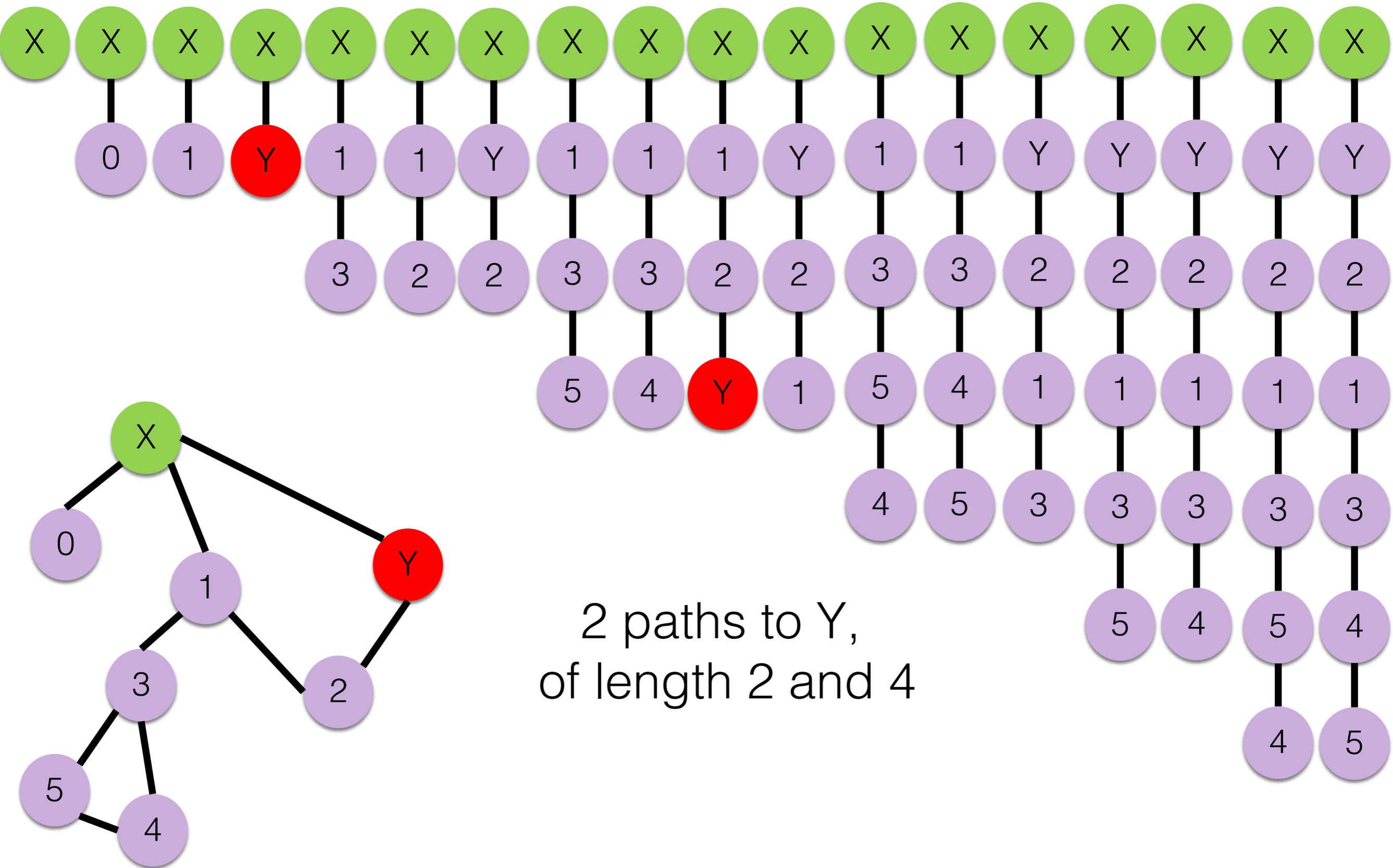


We found a path  
(which is stored in  
the stack), but it is  
not necessarily the  
shortest (which would  
be X-Y)

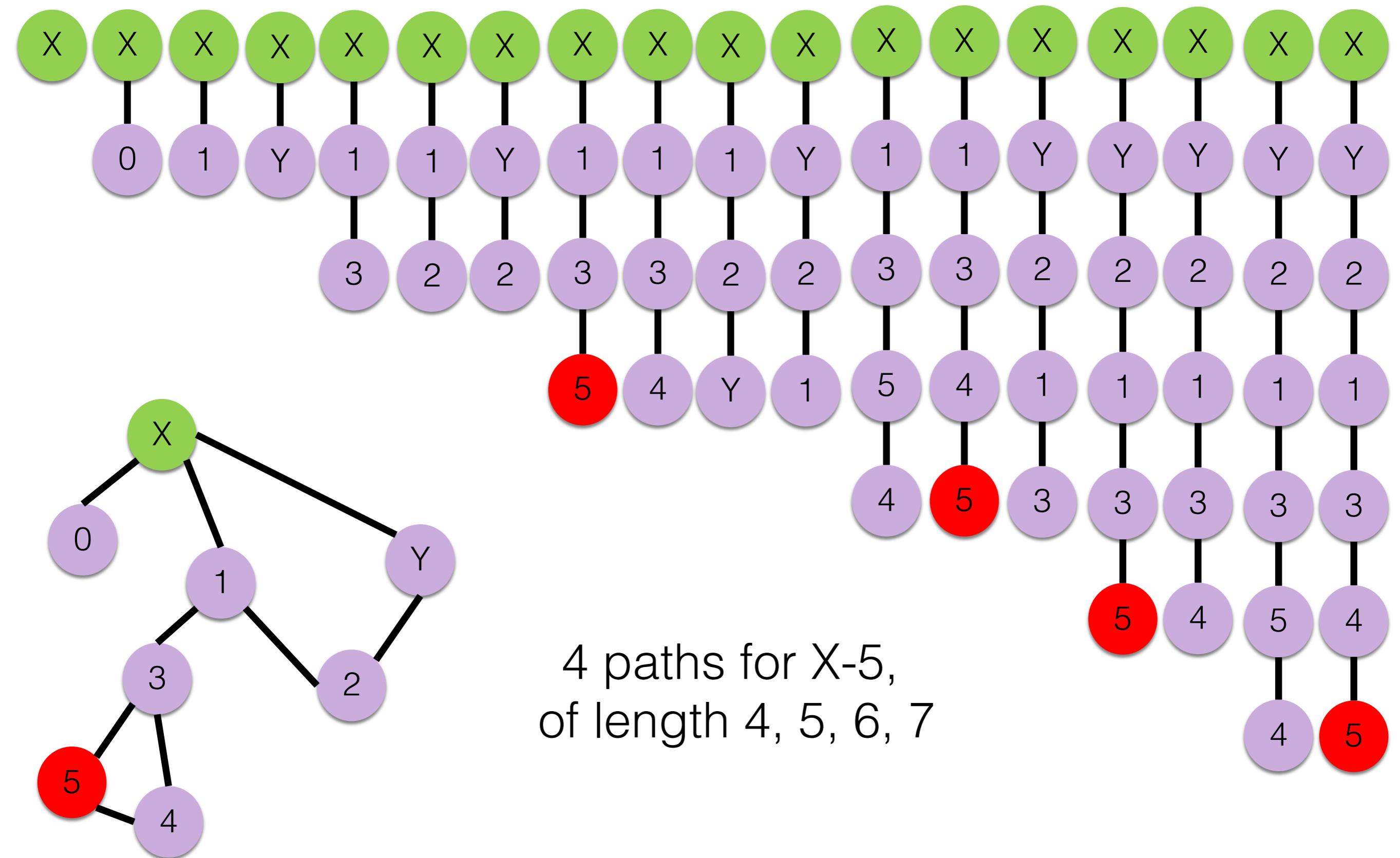
# Breadth-First Search (BFS)

- DFS no guarantee to find shortest path, whereas BFS does
- From starting point X, look at all paths of length 1, then all paths of length 2, then 3, ... then N, until found Y or visited whole graph
- Considering paths without cycles

# Paths Starting From X



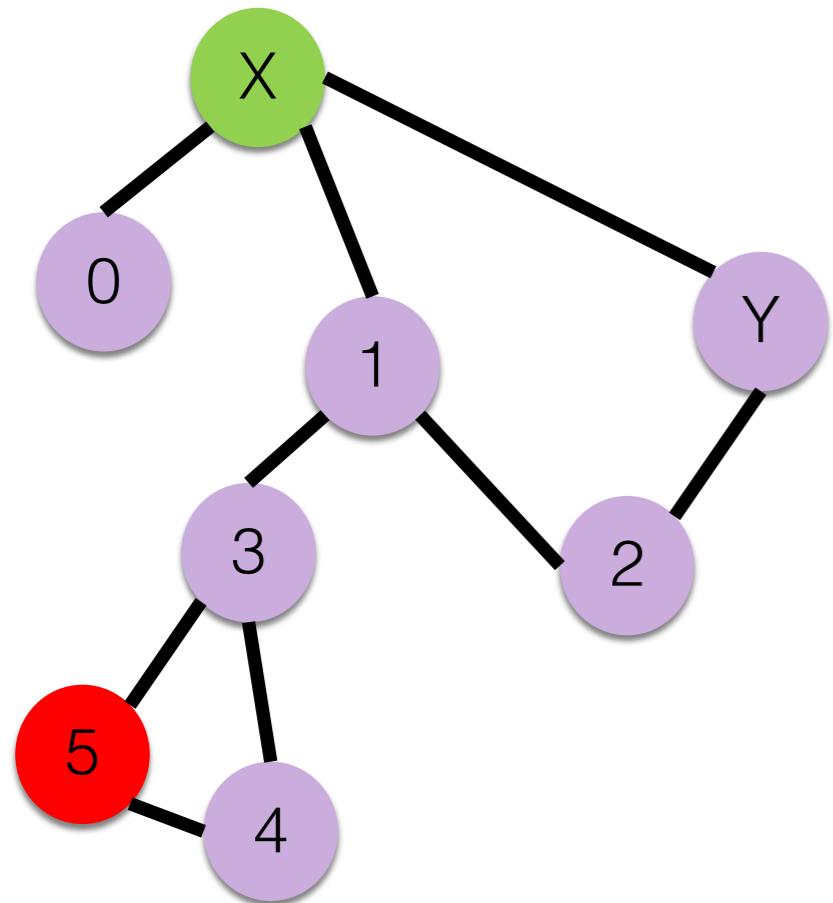
# From X to 5



# BFS Details

- Need special algorithm to keep track and visit all paths of length  $N$  in increasing order
- BFS: use a *queue* of yet to visit vertices
- Pull vertex from queue, add connected vertices to back, if not already visited
- Keep track of which pulled vertex  $X$  added a connected vertex  $Y$

# Example

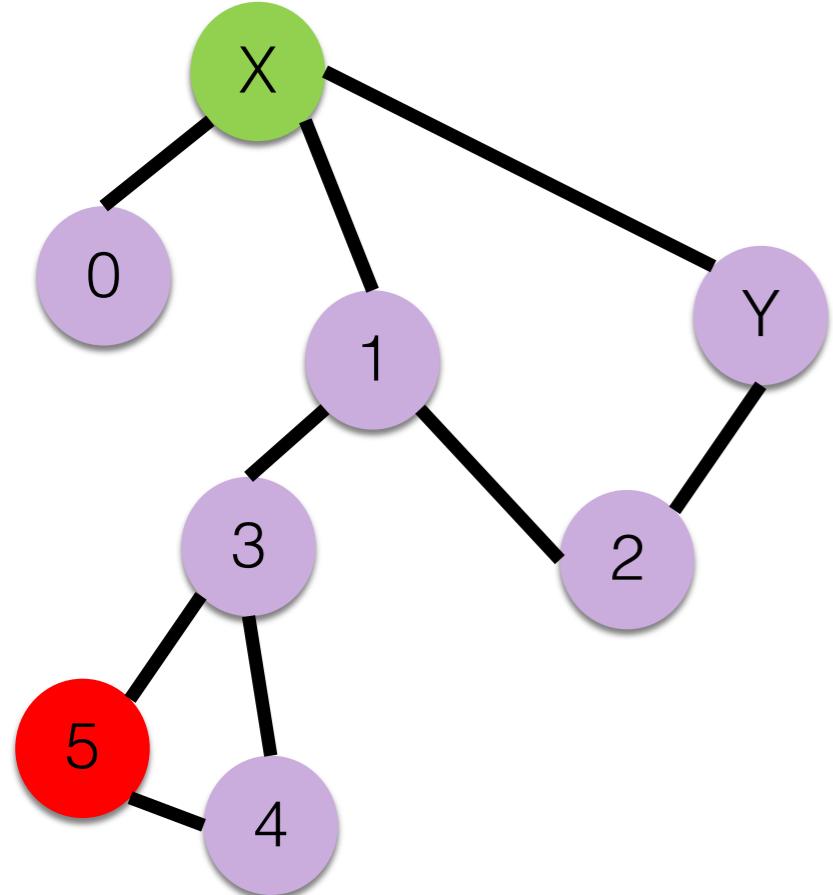


Queue: X

Map:

X	root
Y	null
0	null
1	null
2	null
3	null
4	null
5	null

# Cont.



Queue: 

0	1	Y
---	---	---

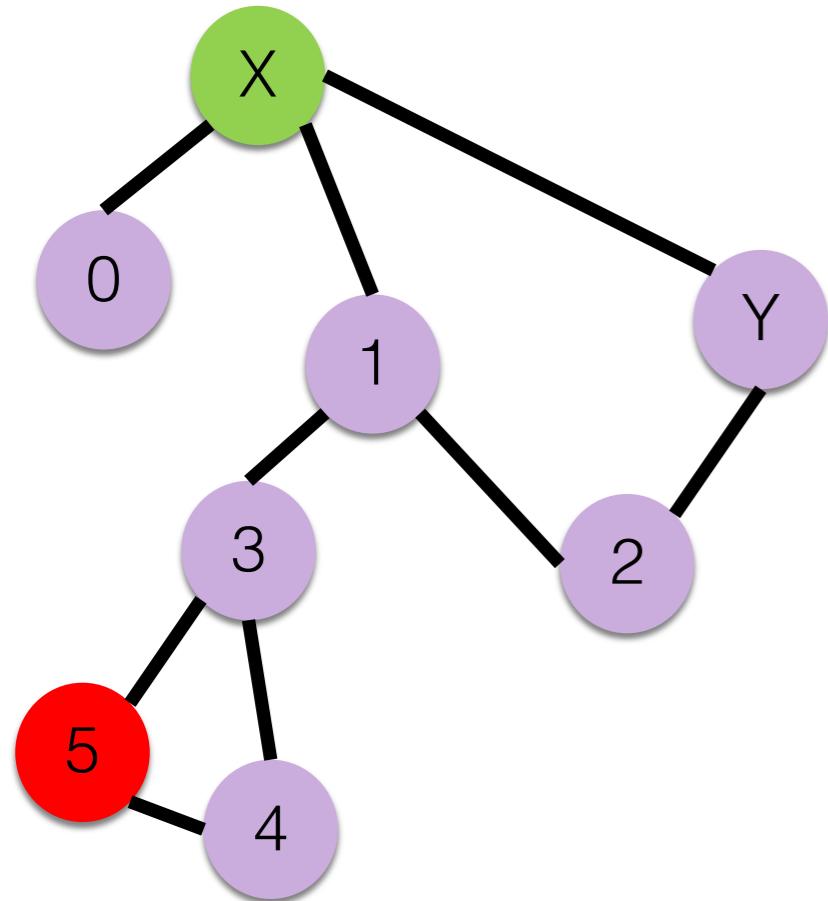
Map:

X	root
Y	X
0	X
1	X
2	null
3	null
4	null
5	null

Pulled X, added 0, 1, and Y.

Those represent all paths of length 2

# Cont.



Queue: 

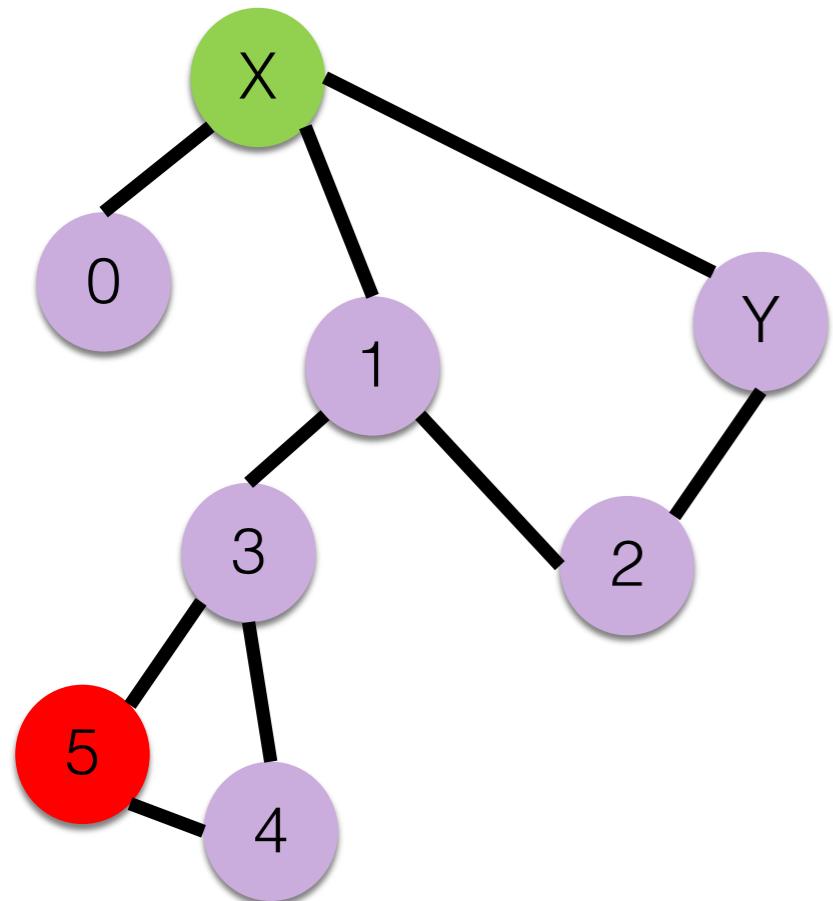
Map:

X	root
Y	X
0	X
1	X
2	1
3	1
4	null
5	null

Pulling 0 has no effect, as not adding X back

Pulling 1 results in adding 3 and 2

# Cont.



Queue: 

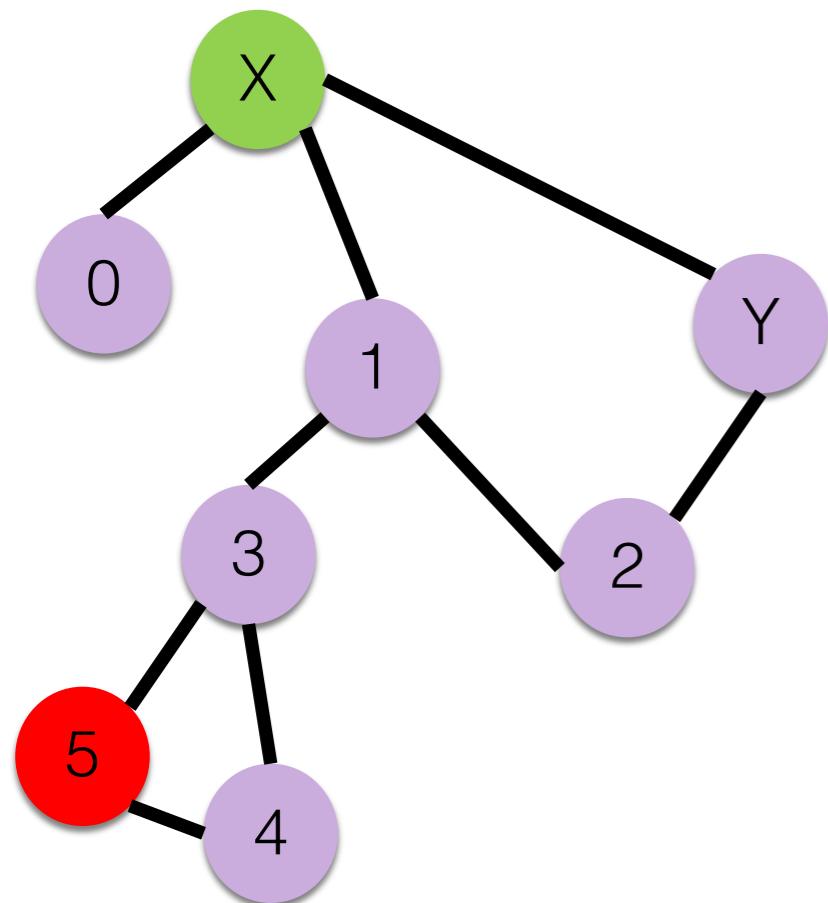
2	3
---	---

Map:

X	root
Y	X
0	X
1	X
2	1
3	1
4	null
5	null

Pulling Y has no effect, as connected 2 and X have already been handled

# Cont.



Queue: 

Pulling 2 has no effect (1 and Y already handled).

Queue: 

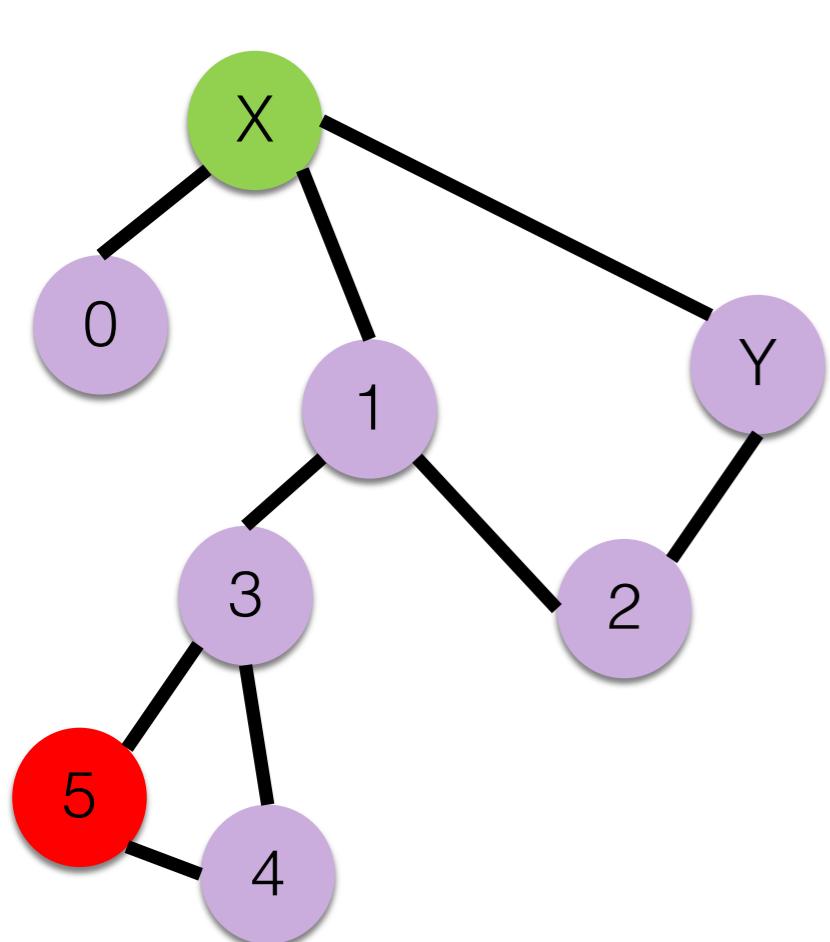
Queue: 

Pulling 3 leads to push 4 and 5 (but not 1 that has already been handled)

	Map:
X	root
Y	X
0	X
1	X
2	1
3	1
4	3
5	3

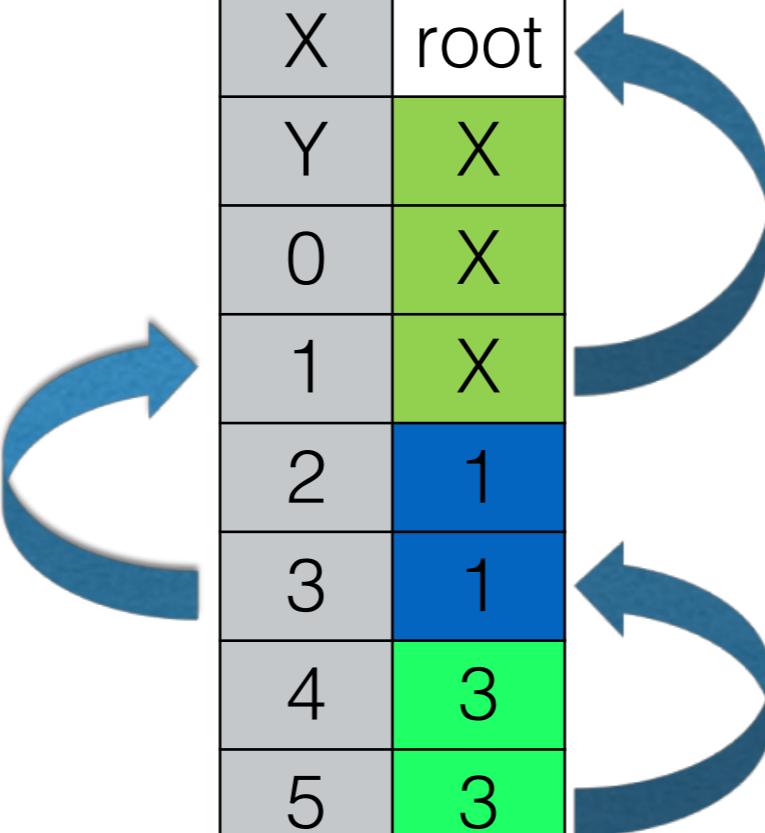
5 is our target

# Retrieve Path

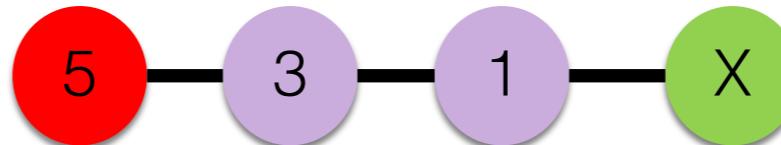


Map:

X	root
Y	X
0	X
1	X
2	1
3	1
4	3
5	3



From 5, follow links backward in the map till X

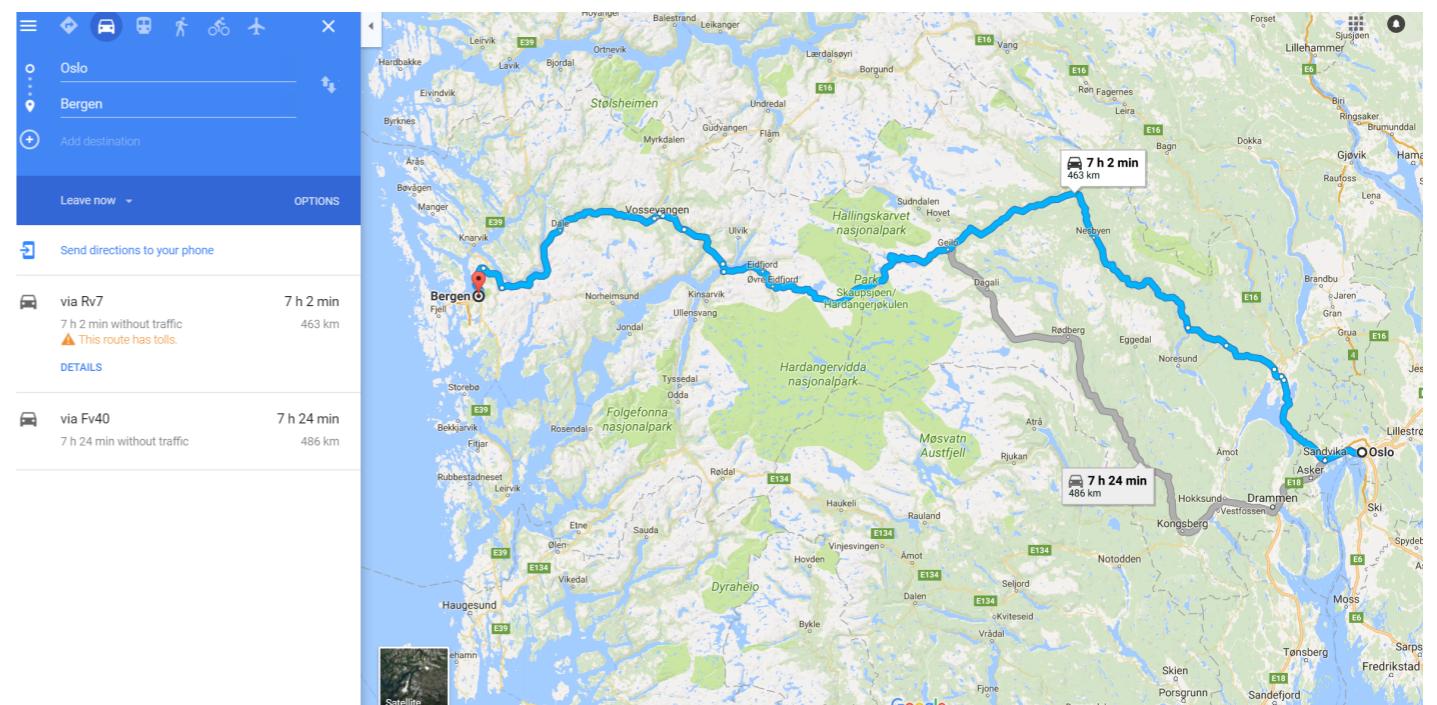


# DFS or BFS?

- BFS guarantees to find minimum path, whereas DFS does not
- But BFS is “usually” more expensive, both in terms of time and memory
  - given  $|A|$  the average arity of the node connections, it needs to compute an exponential number  $|A|^N$  of paths, if the target optimal path has length  $N$ .
  - recall that exponential cost is really bad... for already not so large  $N$ , its cost can become prohibitive

# Weighted Graphs

- You can have graphs where edges have weights
  - Eg, distance between two cities, road tolls, etc.
- Find paths with shortest weight/cost on the traversed edges, even if traversing more vertices



# Homework

- Study Book Chapter 4.1
- Study code in the *org.pg4200.les08* package
- Do exercises in *exercises/ex08*
- Extra: do exercises in the book