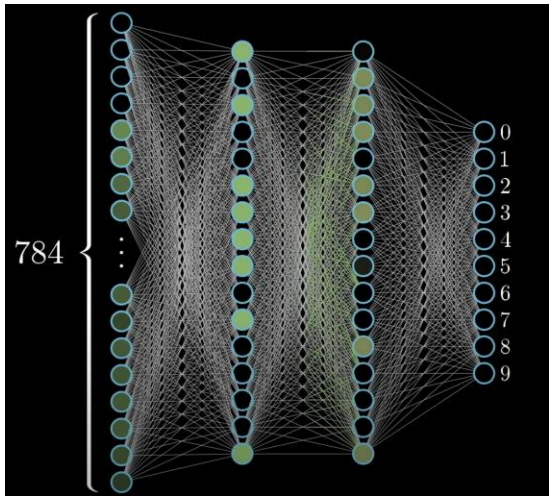
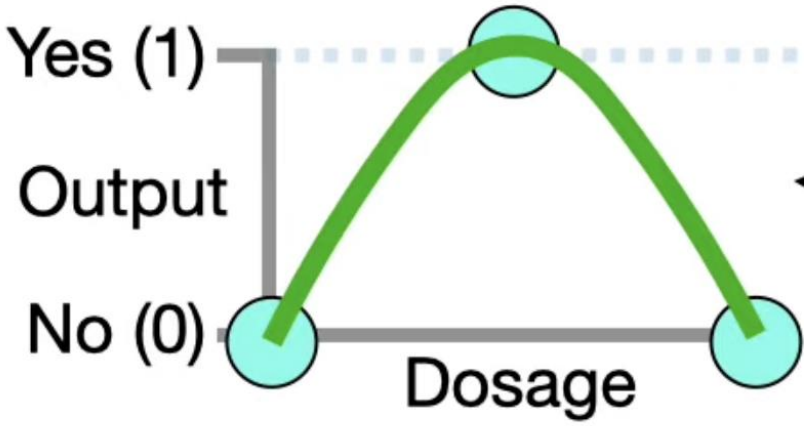
Link to handout: https://bit.ly/NN-DFS-Backprop



PA03: DFS BACKPROP

COMPLEXITY ANALYSIS OF OF GRAPH SEARCH

Neural Net to predict effectiveness of a drug

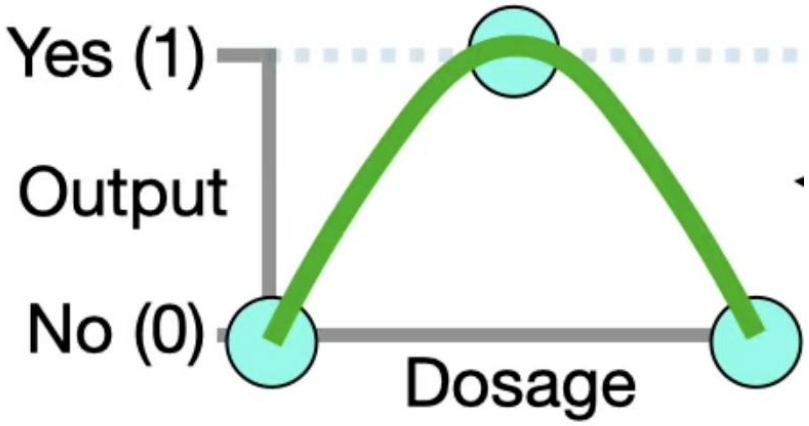


Draw the neural network that was used to predict whether a drug's dosage is effective against a virus.

How many parameters does the NN have?

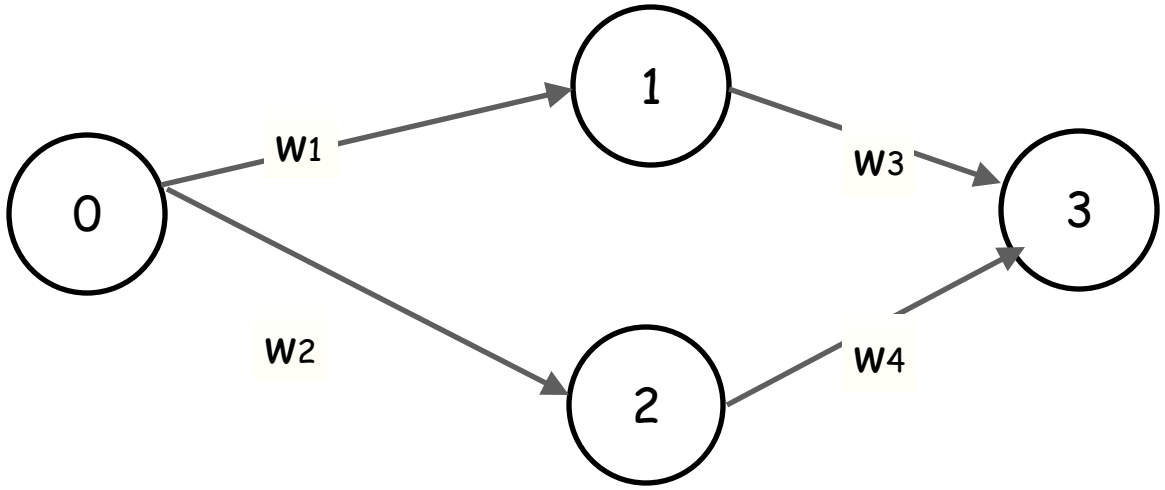
Pre-lecture video on Backpropagation in NNs: https://youtu.be/IN2XmBhILt4?si=bnDft-3T4DQ2iO9X

What is “training” the neural net?



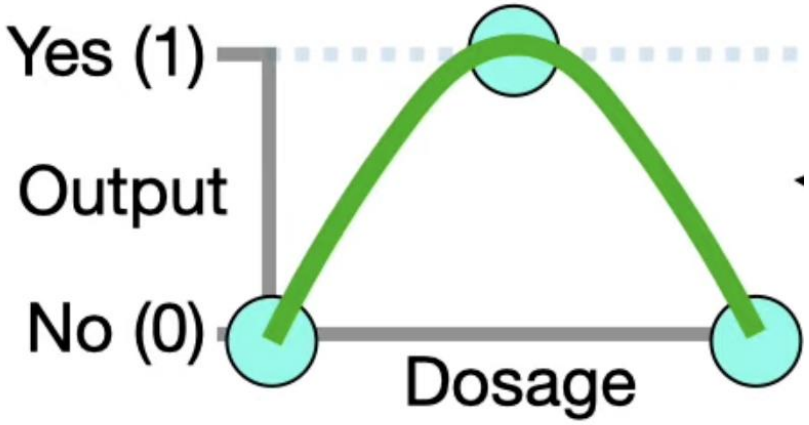
Training a neural net means adjusting its parameters (weights and biases) so it learns to map inputs to the correct outputs.

Show the neural net different example inputs and outputs



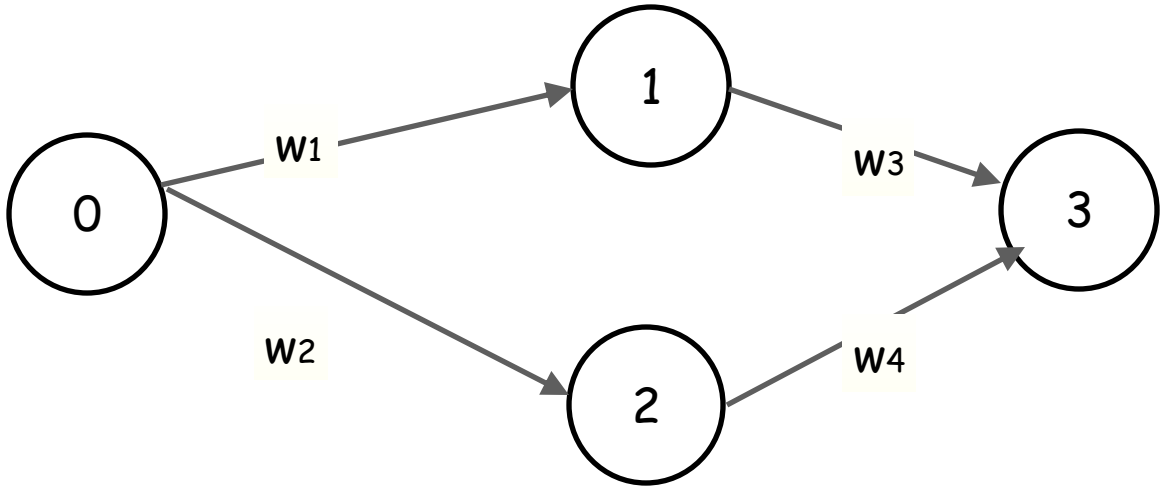
Pre-lecture video on Backpropagation in NNs: https://youtu.be/IN2XmBhILt4?si=bnDft-3T4DQ2iO9X

Backpropagation = smart feedback



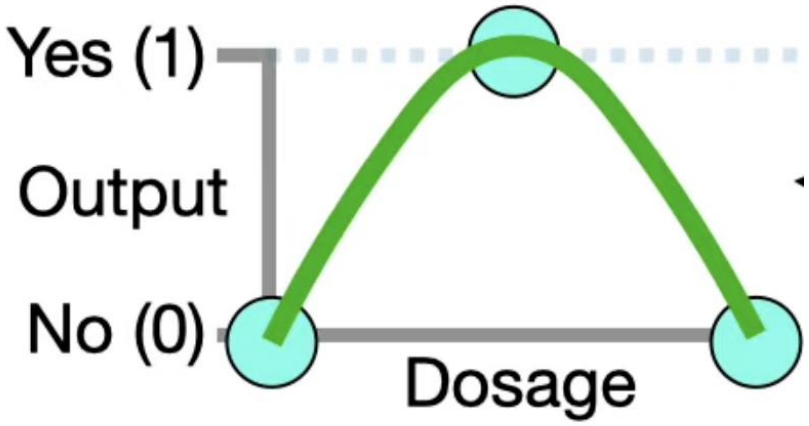
Backprop: Guiding feedback that tells each weight and bias how to slightly nudge to reduce the error.

In PA03, that feedback is abstracted into a call to contribute(node, y, p)



PA03 Starter: https://github.com/ucsb-cs24-s25/STARTER-pa03/blob/main/NeuralNetwork.cpp

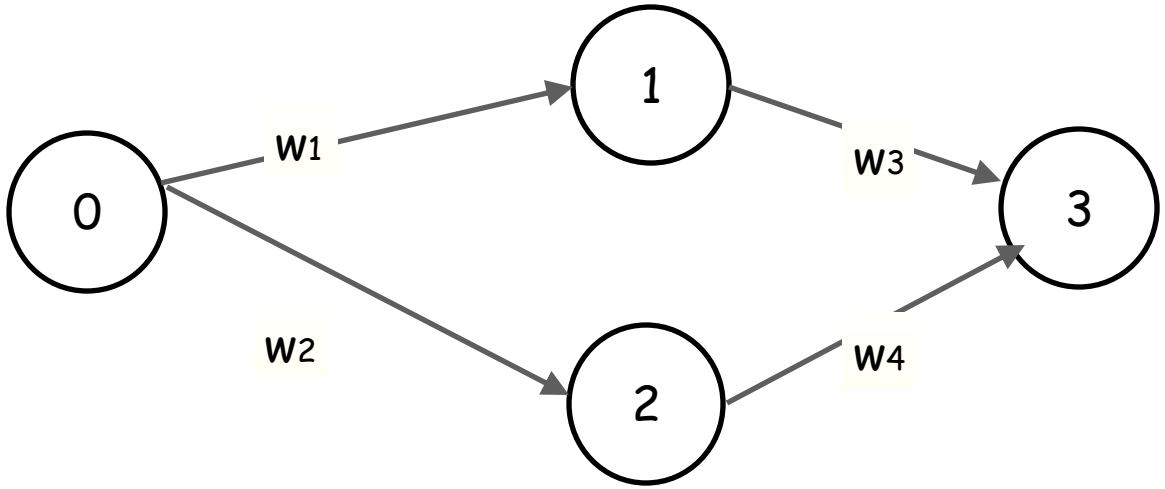
Backpropagation: PA03 contribute()



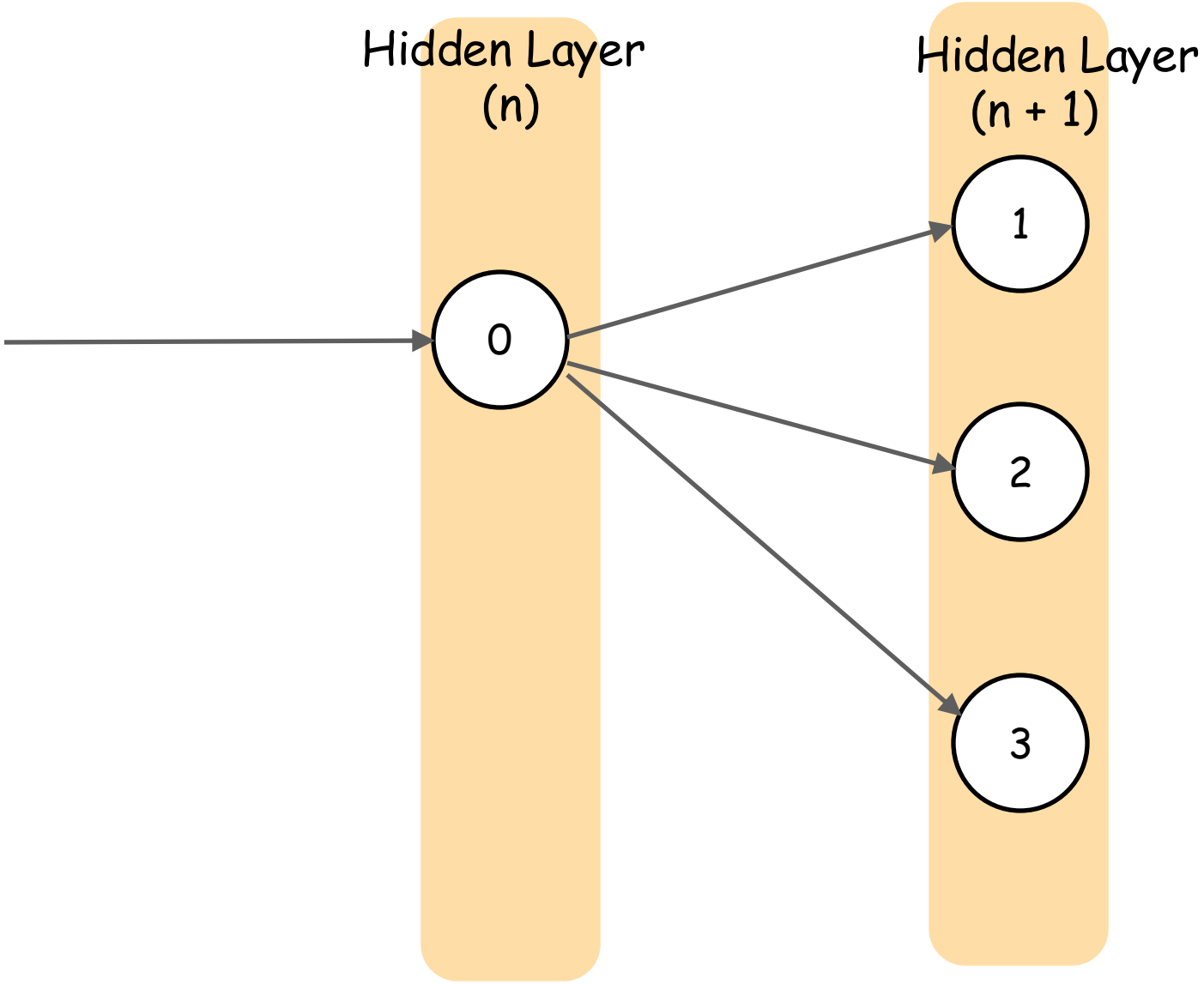
Node 3's contribution = Nudge to node 3's bias

Contributions in one layer are used to compute the contributions in the previous layer

Node 3's outgoing contribution is used to compute the contributions of which nodes?



Backpropagation: Order of operations for one node



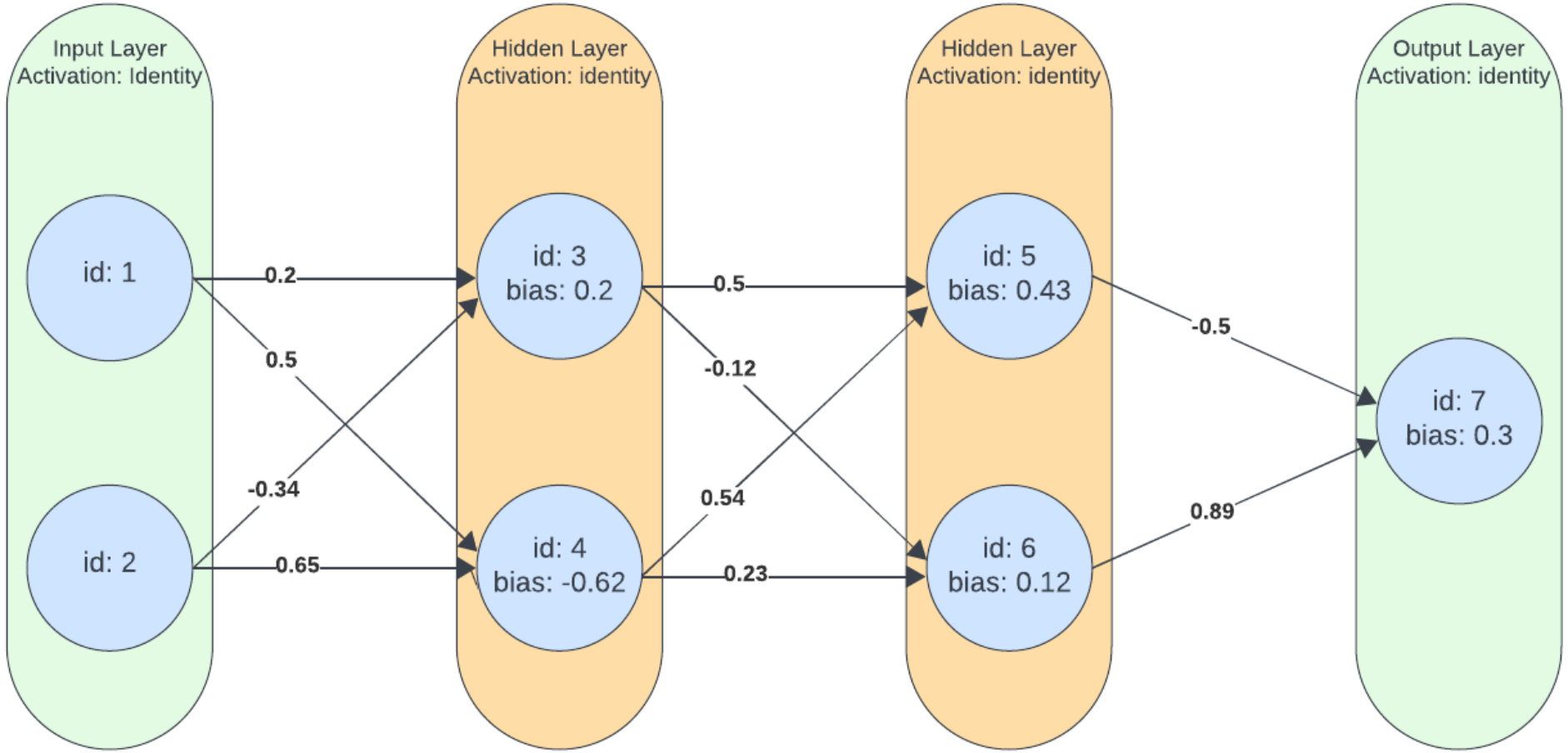
Backprop using DFS (you need to implement)

```
contribute(int nodeId,
        const double& y,
        const double& p)
```

Helper visit functions that do the math for you:

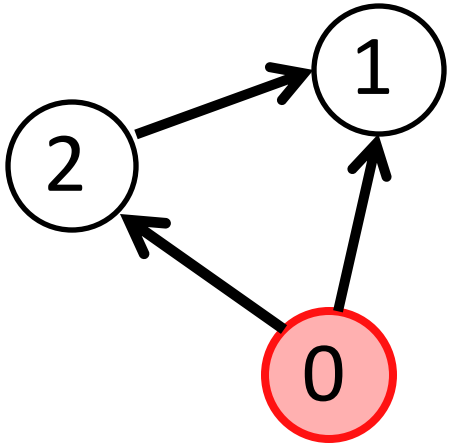
```
visitContributeNeighbor(
  Connection& c,
  double& incomingContribution,
  double& outgoingContribution)

visitContributeNode(
  int vId,
  double& outgoingContribution)
```

PA03: Backpropagation data flow



Trace the order in which **contribute(...)**, **visitContributeNode (…)**, and **visitContributeNeighbor(…)** are called. For each node, write down: when it's visited, what contributions it receives, and when those are passed on to others.

BFS: Running Time Complexity



Algo exploreBFS (Graph G, vertex s):

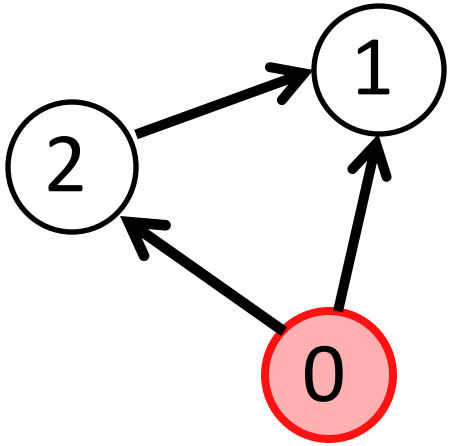
- Mark all the vertices as “not visited”
- Mark ***s*** as visited
- push ***s*** into a queue
- while the queue is not empty:
  - pop the vertex ***u*** from the front of the queue
  - for each of ***u***’s neighbor (***v***)
    - If ***v*** has not yet been visited:
      - Mark ***v*** as visited
      - Push ***v*** in the queue

n: number of vertices
m: number of edges

How many times does the while loop run?

A. n
B. m
C. n + m
D. nm
E. None of the above

BFS: Running Time Complexity



Algo exploreBFS (Graph G, vertex s):

For each iteration of the while loop, the for loop runs a variable number of times. How should we proceed to analyze the Big-O running time?

- while the queue is not empty:
  - pop the vertex ***u*** from the front of the queue
  - for each of ***u***'s neighbor (***v***):
    - If ***v*** has not yet been visited:
      - Mark ***v*** as visited
      - Push ***v*** in the queue

A. Bound the maximum number of times the for loop runs **per iteration** of the while loop

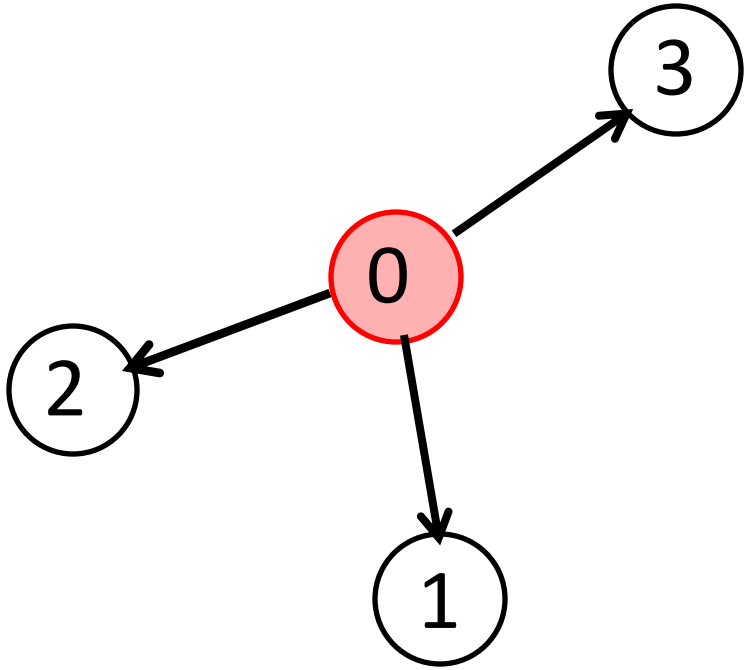
B. Compute the total number of times the for loop runs over **the entire run of exploreBFS**

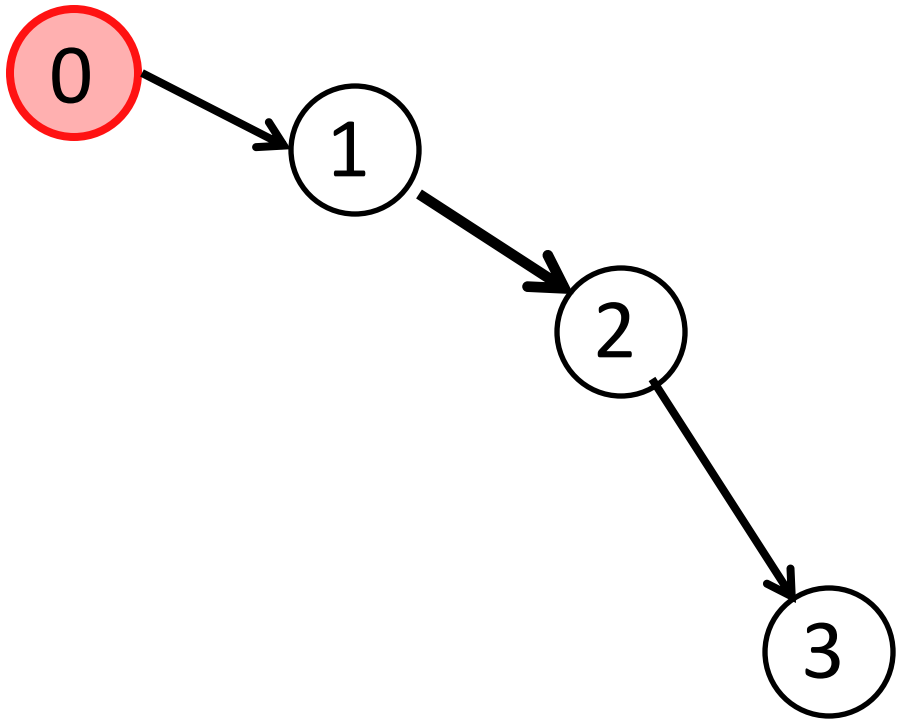
C. Cannot compute Big-O because running time depends on two parameters (n, m)

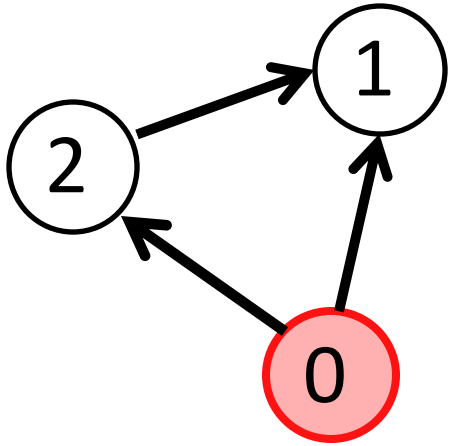
BFS: Running Time Complexity

(A) The total number of times the for loop runs over **the entire run of exploreBFS**

(B) The total number of times each neighbor (u) is checked for all vertices over **the entire run of exploreBFS**







BFS: Time Complexity

Algo exploreBFS (Graph G, vertex s):

- Mark all the vertices as “not visited”
- Mark ***s*** as visited
- push ***s*** into a queue
- while the queue is not empty:
  - pop the vertex ***u*** from the front of the queue
  - for each of ***u***’s neighbor (***v***)
    - If ***v*** has not yet been visited:
      - Mark ***v*** as visited
      - Push ***v*** in the queue

n: number of vertices
m: number of edges

What is the time complexity of exploreBFS?

A. O(n)
B. O(m)
C. O(n + m)
D. O(nm)
E. None of the above

BFS Traverse: Space Complexity

Algo exploreBFS (Graph G, vertex s):

- Mark all the vertices as “not visited”
- Mark ***s*** as visited
- push ***s*** into a queue
- while the queue is not empty:
  - pop the vertex ***u*** from the front of the queue
  - for each of ***u***’s neighbor (***v***)
    - If ***v*** has not yet been visited:
      - Mark ***v*** as visited
      - Push ***v*** in the queue

n: number of vertices
m: number of edges

What is the Big -O auxiliary space complexity of exploreBFS?

A. O(n)
B. O(m)
C. O(n + m)
D. O(n^2)
E. None of the above

- Auxiliary Space complexity: Additional space usage (not including input and output)

exploreDFS: Time Complexity

```
exploreDFS(v)

  v.visited ← true

  For each edge (v, w):

    If not w.visited

      exploreDFS(w)
```

n: number of vertices
m: number of edges

What is the time complexity of exploreDFS?

A. O(n)
B. O(m)
C. O(n + m)
D. O(n^2)
E. None of the above

exploreDFS: Space Complexity

```
exploreDFS(v)

  v.visited ← true

  For each edge (v,w)

    If not w.visited

      exploreDFS(w)
```

n: number of vertices
m: number of edges

What is the worst-case space complexity of exploreDFS?

A. O(n)
B. O(m)
C. O(n + m)
D. O(n^2+ n.m)
E. None of the above