CSE 331:

Algorithms & Complexity

“BFS and DFS Runtime”

Prof. Charlie Anne Carlson (She/Her)

Lecture 13

Friday September 26th, 2025

University at Buffalo

Schedule

1. Course Updates
2. Stacks & Queues
3. Another BFS
4. Runtime Analysis
    1. BFS
    2. DFS
5. Coloring

Course Updates

- **HW 2 Solutions Soon**
- HW 3 Out
- Group Project
  - **Team Emails Soon**
  - No Autolab Registration
- First Quiz NEXT Monday!
  - **Quiz Solutions Soon**
- **Sample Midterms Soon-ish**

Soon = Today… or before I go to sleep

Q: What is a stack?

- A data structure for maintaining a set of elements.
- We can add and remove elements from the stack in constant time.
- When we remove an element, we get the last element that was added.
    - “last-in, first-out” or LIFO

Q: What is Queue?

- A data structure for maintaining a set of elements.
- We can add and remove elements from the stack in constant time.
- When we remove an element, we get the first element that was added (and is still in the set).
    - “first-in, first-out” or FIFO

Stack vs Queue

- Both can be implemented with a (doubly) linked list.
- Let's assume that both implement the remove function to take the first element of the linked list.
- Q: How do we implement the add function for Stack vs Queue?

Stack vs Queue

- Both can be implemented with a (doubly) linked list.
- Let’s assume that both implement the remove function to take the first element of the linked list.
- A: For a Stack we insert at the front and for a Queue we insert at the end.

Breadth First Search

- **Input:** $G = (V, E)$ and $s \ \in \ V$
- **Output:** BFS Tree
- Let $L_0 = \{s\}$
- Assume $L_0, \dots, L_i$ have been constructed:
  - Let $L_{\mathrm{i}+1}$ be nodes do not appear in $L_0, \dots, L_i$ and have an edge to L_i.
  - If $L_{\mathrm{i}+1}$ is empty, stop.
- Return all layers.

A more specific BFS

```
BFS(s):
   Initialize Discovered to be a node index array of false
   Set Discovered[s] = true
   Initialize L[0] to be a linked list with one element s
   Initialize i to be 0
   While L[i] is not empty:
      Initialize L[i+1] to be empty linked list
      For u in L[i]:
            For each edge (u, v) incident to u:
                  If Discovered[v] = false:
                        Set Discovered[u] = true
                        Add v to L[i+1]
      ++i
```

Q: What is the runtime?

```
BFS(s):
   Initialize Discovered to be a node index array of false
   Set Discovered[s] = true
   Initialize L[0] to be a linked list with one element s
   Initialize i to be 0
   While L[i] is not empty:
      Initialize L[i+1] to be empty linked list
      For u in L[i]:
            For each edge (u, v) incident to u:
                  If Discovered[v] = false:
                        Set Discovered[u] = true
                        Add v to L[i+1]
      ++i
```

Total Time:

Q: What is the runtime?

```
BFS(s):
O(n) Initialize Discovered to be a node index array of false
   Set Discovered[s] = true
   Initialize L[0] to be a linked list with one element s
   Initialize i to be 0
   While L[i] is not empty:
      Initialize L[i+1] to be empty linked list
      For u in L[i]:
            For each edge (u, v) incident to u:
                  If Discovered[v] = false:
                        Set Discovered[u] = true
                        Add v to L[i+1]
      ++i
```

Total Time:

Q: What is the runtime?

```
BFS(s):
O(n)  Initialize Discovered to be a node index array of false
O(1)  Set Discovered[s] = true
      Initialize L[0] to be a linked list with one element s
      Initialize i to be 0
      While L[i] is not empty:
          Initialize L[i+1] to be empty linked list
          For u in L[i]:
                For each edge (u, v) incident to u:
                      If Discovered[v] = false:
                            Set Discovered[u] = true
                            Add v to L[i+1]
          ++i
```

Q: What is the runtime?

Total Time:

```
BFS(s):
O(n) Initialize Discovered to be a node index array of false
O(1) Set Discovered[s] = true
O(1) Initialize L[0] to be a linked list with one element s
     Initialize i to be 0
     While L[i] is not empty:
        Initialize L[i+1] to be empty linked list
        For u in L[i]:
              For each edge (u, v) incident to u:
                    If Discovered[v] = false:
                          Set Discovered[u] = true
                          Add v to L[i+1]
        ++i
```

Q: What is the runtime?

Total Time:

```
BFS(s):
O(n)  Initialize Discovered to be a node index array of false
O(1)  Set Discovered[s] = true
O(1)  Initialize L[0] to be a linked list with one element s
O(1)  Initialize i to be 0
      While L[i] is not empty:
         Initialize L[i+1] to be empty linked list
         For u in L[i]:
               For each edge (u, v) incident to u:
                     If Discovered[v] = false:
                           Set Discovered[u] = true
                           Add v to L[i+1]
         ++i
```

Total Time:

Q: What is the runtime?

```
BFS(s):
O(n) Initialize Discovered to be a node index array of false
O(1) Set Discovered[s] = true
O(1) Initialize L[0] to be a linked list with one element s
O(1) Initialize i to be 0
     While L[i] is not empty: Q: How many times does this loop run?
         Initialize L[i+1] to be empty linked list
         For u in L[i]:
               For each edge (u, v) incident to u:
                     If Discovered[v] = false:
                           Set Discovered[u] = true
                           Add v to L[i+1]
         ++i
```

Q: What is the runtime?

Total Time:

```
     BFS(s):
O(n)   Initialize Discovered to be a node index array of false
O(1)   Set Discovered[s] = true
O(1)   Initialize L[0] to be a linked list with one element s
O(1)   Initialize i to be 0
       While L[i] is not empty:  A: $|\cup_i L_i| \leq n$
           Initialize L[i+1] to be empty linked list
           For u in L[i]:
                 For each edge (u, v) incident to u:
                       If Discovered[v] = false:
                             Set Discovered[u] = true
                             Add v to L[i+1]
           ++i
```

Total Time:

Q: What is the runtime?

```
     BFS(s):
O(n)   Initialize Discovered to be a node index array of false
O(1)   Set Discovered[s] = true
O(1)   Initialize L[0] to be a linked list with one element s
O(1)   Initialize i to be 0
O(n)   While L[i] is not empty:
           Initialize L[i+1] to be empty linked list     Q: How many layers max?
           For u in L[i]:
                   For each edge (u, v) incident to u:
                           If Discovered[v] = false:
                                   Set Discovered[u] = true
                                   Add v to L[i+1]
           ++i
```

Total Time:

Q: What is the runtime?

```
     BFS(s):
O(n)   Initialize Discovered to be a node index array of false
O(1)   Set Discovered[s] = true
O(1)   Initialize L[0] to be a linked list with one element s
O(1)   Initialize i to be 0
O(n)   While L[i] is not empty:
O(n)       Initialize L[i+1] to be empty linked list
           For u in L[i]:
                 For each edge (u, v) incident to u:
                       If Discovered[v] = false:
                             Set Discovered[u] = true
                             Add v to L[i+1]
           ++i
```

A: One for each vertex.

Total Time:

Q: What is the runtime?

```
     BFS(s):
O(n)   Initialize Discovered to be a node index array of false
O(1)   Set Discovered[s] = true
O(1)   Initialize L[0] to be a linked list with one element s
O(1)   Initialize i to be 0
O(n)   While L[i] is not empty:
O(n)       Initialize L[i+1] to be empty linked list
           For u in L[i]:
               For each edge (u, v) incident to u:
                   If Discovered[v] = false:
                       Set Discovered[u] = true
                       Add v to L[i+1]
           ++i
```

Q: How many times does this loop run?

Total Time:

Q: What is the runtime?

```
     BFS(s):
O(n)  Initialize Discovered to be a node index array of false
O(1)  Set Discovered[s] = true
O(1)  Initialize L[0] to be a linked list with one element s
O(1)  Initialize i to be 0
O(n)  While L[i] is not empty:
O(n)      Initialize L[i+1] to be empty linked list
O(n)      For u in L[i]: A: $|\bigcup_i L_i| \leq n$
              For each edge (u, v) incident to u:
                  If Discovered[v] = false:
                      Set Discovered[u] = true
                      Add v to L[i+1]
          ++i
```

Total Time:

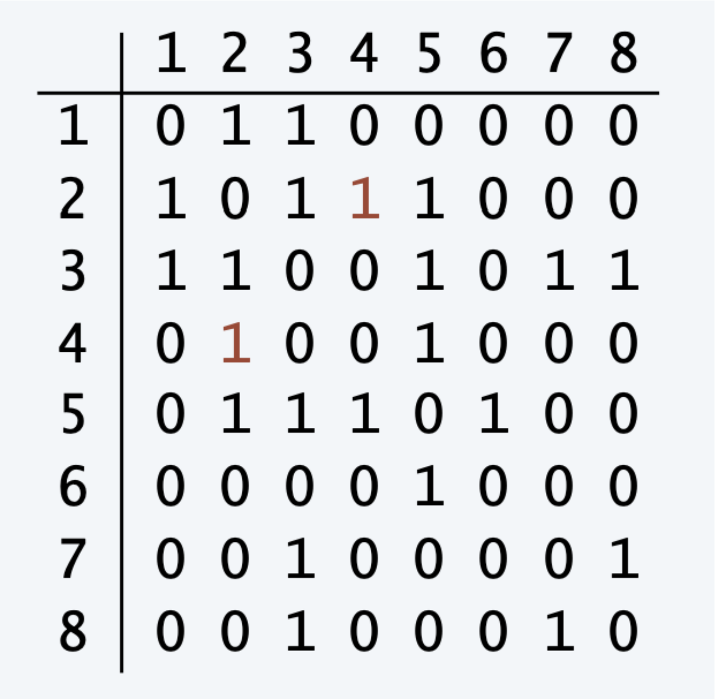
Q: What is the runtime?

```
     BFS(s):
O(n)   Initialize Discovered to be a node index array of false
O(1)   Set Discovered[s] = true
O(1)   Initialize L[0] to be a linked list with one element s
O(1)   Initialize i to be 0
O(n)   While L[i] is not empty:
O(n)       Initialize L[i+1] to be empty linked list
O(n)       For u in L[i]:
               For each edge (u, v) incident to u:
                   If Discovered[v] = false:
                       Set Discovered[u] = true
                       Add v to L[i+1]
           ++i
```

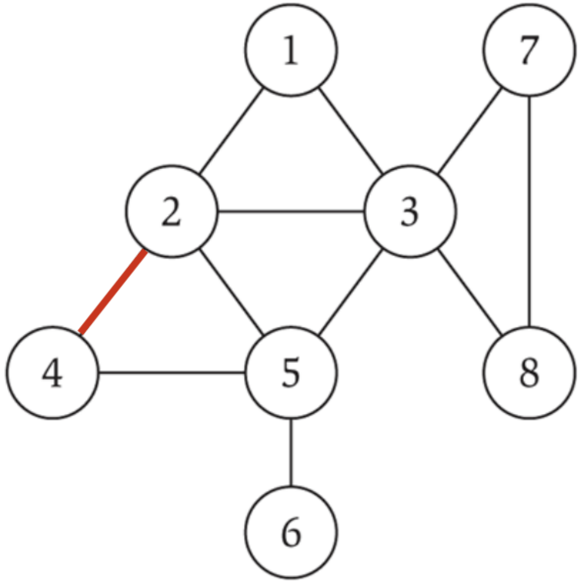
Q: How do we do this?

Q: Which should we use?

Space: O(n^2)
Lookup: O(1)
List Neighbors: O(n)



	1	2	3	4	5	6	7	8
1	0	1	1	0	0	0	0	0
2	1	0	1	1	1	0	0	0
3	1	1	0	0	1	0	1	1
4	0	1	0	0	1	0	0	0
5	0	1	1	1	0	1	0	0
6	0	0	0	0	1	0	0	0
7	0	0	1	0	0	0	0	1
8	0	0	1	0	0	0	1	0

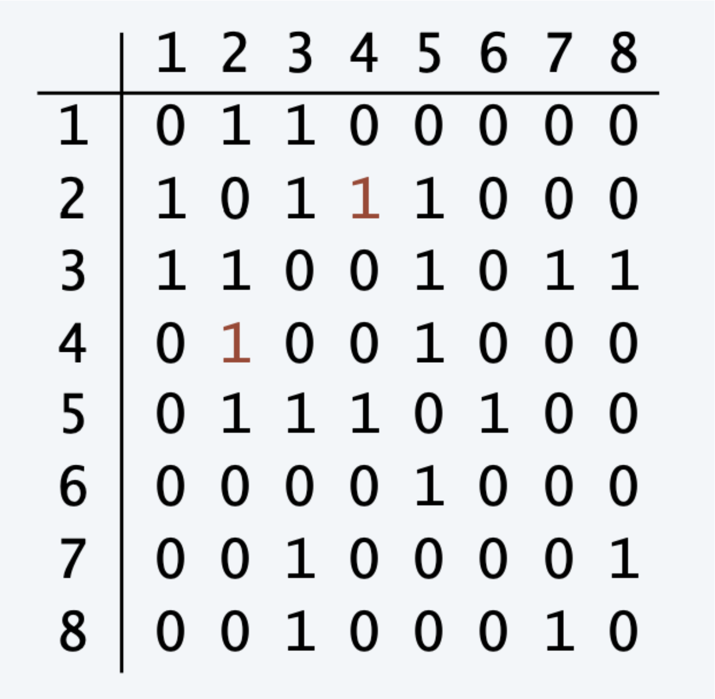


Space: O(n + m)
Lookup: O(d_u)
List Neighbors: O(d_u)

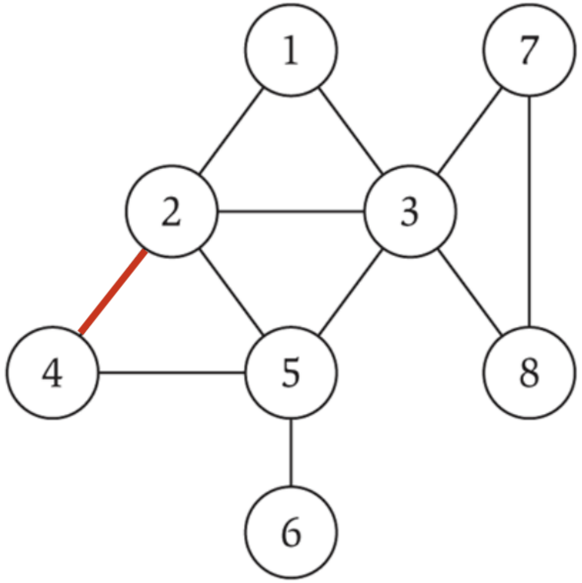
N[1] : [2] -> [3]
N[2] : [1] -> [3] -> [5] -> [4]
N[3] : [1] -> [2] -> [5] -> [7] -> [8]
N[4] : [2] -> [5]
N[5] : [2] -> [3] -> [4] -> [6]
N[6] : [5]
N[7] : [3] -> [8]
N[8] : [3] -> [7]

Adjacency List vs Adjacency Matrix

Space: O(n^2)
Lookup: O(1)
List Neighbors: O(n)



	1	2	3	4	5	6	7	8
1	0	1	1	0	0	0	0	0
2	1	0	1	1	1	0	0	0
3	1	1	0	0	1	0	1	1
4	0	1	0	0	1	0	0	0
5	0	1	1	1	0	1	0	0
6	0	0	0	0	1	0	0	0
7	0	0	1	0	0	0	0	1
8	0	0	1	0	0	0	1	0



Space: O(n + m)
Lookup: O(d_u)
List Neighbors: O(d_u)

N[1] : [2] -> [3]
N[2] : [1] -> [3] -> [5] -> [4]
N[3] : [1] -> [2] -> [5] -> [7] -> [8]
N[4] : [2] -> [5]
N[5] : [2] -> [3] -> [4] -> [6]
N[6] : [5]
N[7] : [3] -> [8]
N[8] : [3] -> [7]

Total Time:

Q: What is the runtime?

```
     BFS(s):
O(n)   Initialize Discovered to be a node index array of false
O(1)   Set Discovered[s] = true
O(1)   Initialize L[0] to be a linked list with one element s
O(1)   Initialize i to be 0
O(n)   While L[i] is not empty:
O(n)       Initialize L[i+1] to be empty linked list
O(n)       For u in L[i]:
                 For each edge (u, v) incident to u: A: Linked List
                       If Discovered[v] = false:
                             Set Discovered[u] = true
                             Add v to L[i+1]
           ++i
```

Total Time:

Q: What is the runtime?

```
BFS(s):
O(n) Initialize Discovered to be a node index array of false
O(1) Set Discovered[s] = true
O(1) Initialize L[0] to be a linked list with one element s
O(1) Initialize i to be 0
O(n) While L[i] is not empty:
O(n)     Initialize L[i+1] to be empty linked list
O(n)     For u in L[i]:
             O($d_u$)For each edge (u, v) incident to u:
                 If Discovered[v] = false:
                     Set Discovered[u] = true
                     Add v to L[i+1]
         ++i
```

Q: How many times does this loop run for u?

Q: What is the runtime?

Total Time:

```
                   BFS(s):
$O(n)$               Initialize Discovered to be a node index array of false
$O(1)$               Set Discovered[s] = true
$O(1)$               Initialize L[0] to be a linked list with one element s
$O(1)$               Initialize i to be 0
$O(n)$               While L[i] is not empty:
$O(n)$                 Initialize L[i+1] to be empty linked list
$O(n)$                 For u in L[i]:
$O(\sum d_u)$              For each edge (u, v) incident to u:   A: One time for each u!
                               If Discovered[v] = false:
                                   Set Discovered[u] = true
                                   Add v to L[i+1]
                       ++i
```

Total Time:

Q: What is the runtime?

```
      BFS(s):
O(n)    Initialize Discovered to be a node index array of false
O(1)    Set Discovered[s] = true
O(1)    Initialize L[0] to be a linked list with one element s
O(1)    Initialize i to be 0
O(n)    While L[i] is not empty:
O(n)        Initialize L[i+1] to be empty linked list
O(n)        For u in L[i]:
O(m)                For each edge (u, v) incident to u:
                        If Discovered[v] = false:
                                Set Discovered[u] = true
                                Add v to L[i+1]
            ++i
```

Total Time:

Q: What is the runtime?

```
     BFS(s):
O(n)   Initialize Discovered to be a node index array of false
O(1)   Set Discovered[s] = true
O(1)   Initialize L[0] to be a linked list with one element s
O(1)   Initialize i to be 0
O(n)   While L[i] is not empty:
O(n)      Initialize L[i+1] to be empty linked list
O(n)      For u in L[i]:
O(m)            For each edge (u, v) incident to u:
            O(1)      If Discovered[v] = false:
                 O(1)       Set Discovered[u] = true
                 O(1)       Add v to L[i+1]
    O(1)  ++i
```

Total Time:

Q: What is the runtime?

```
      BFS(s):
O(n)    Initialize Discovered to be a node index array of false
O(1)    Set Discovered[s] = true
O(1)    Initialize L[0] to be a linked list with one element s
O(1)    Initialize i to be 0
O(n)    While L[i] is not empty:
O(n)        Initialize L[i+1] to be empty linked list
O(n)        For u in L[i]:
O(m)              For each edge (u, v) incident to u:
O(m)                    If Discovered[v] = false:
O(m)                          Set Discovered[u] = true
O(m)                          Add v to L[i+1]
O(n)        ++i
```

Total Time:

Q: What is the runtime? O(n + m)

```
       BFS(s):
O(n)     Initialize Discovered to be a node index array of false
O(1)     Set Discovered[s] = true
O(1)     Initialize L[0] to be a linked list with one element s
O(1)     Initialize i to be 0
O(n)     While L[i] is not empty:
O(n)        Initialize L[i+1] to be empty linked list
O(n)        For u in L[i]:
O(m)              For each edge (u, v) incident to u:
O(m)                    If Discovered[v] = false:
O(m)                          Set Discovered[u] = true
O(m)                          Add v to L[i+1]
O(n)        ++i
```

Q: What if I use a matrix?

```
BFS(s):
   Initialize Discovered to be a node index array of false
   Set Discovered[s] = true
   Initialize L[0] to be a linked list with one element s
   Initialize i to be 0
   While L[i] is not empty:
      Initialize L[i+1] to be empty linked list
      For u in L[i]:
            For each edge (u, v) incident to u:
                  If Discovered[v] = false:
                        Set Discovered[u] = true
                        Add v to L[i+1]
      ++i
```

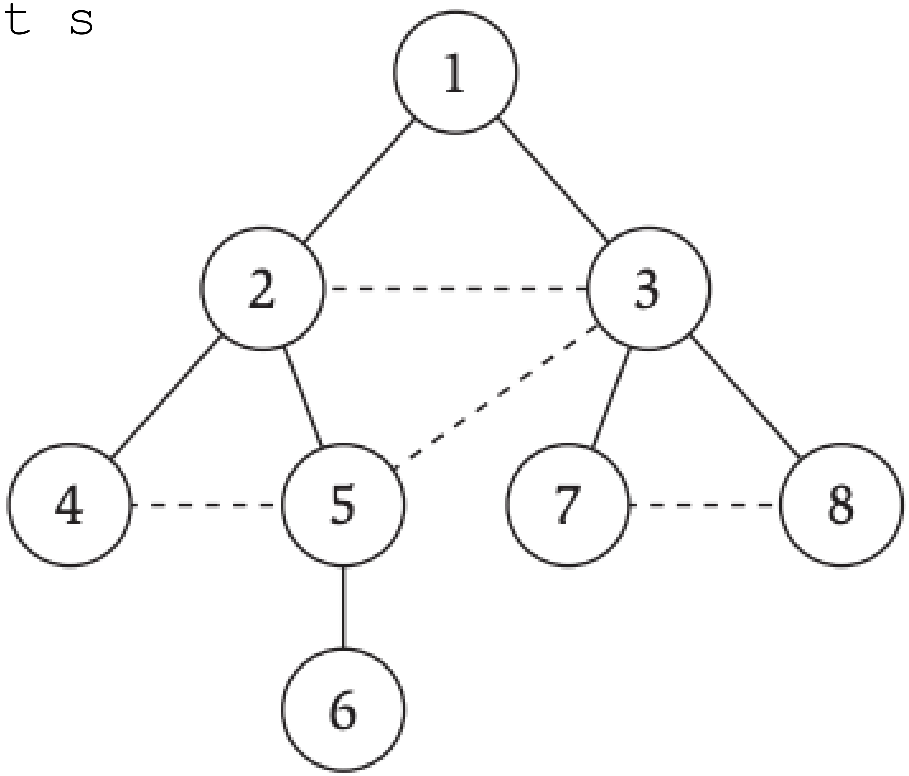
Q: What if I use a matrix? O(n^2)

```
BFS(s):
    Initialize Discovered to be a node index array of false
    Set Discovered[s] = true
    Initialize L[0] to be a linked list with one element s
    Initialize i to be 0
    While L[i] is not empty:
        Initialize L[i+1] to be empty linked list
        For u in L[i]:
O(n^2)          For each edge (u, v) incident to u:
O(n^2)                  If Discovered[v] = false:
O(n^2)                          Set Discovered[u] = true
O(n^2)                          Add v to L[i+1]
        ++i
```

A New BFS

```
BFS(s):
   Initialize Discovered to be a node index array of false
   Set Discovered[s] = true
   Initialize Q to be a Queue with one element s
   While Q is not empty:
      u = Q.dequeue()
      For each edge (u, v) incident to u:
            If Discovered[v] = false:
                  Set Discovered[u] = true
                  Add v to Q
```

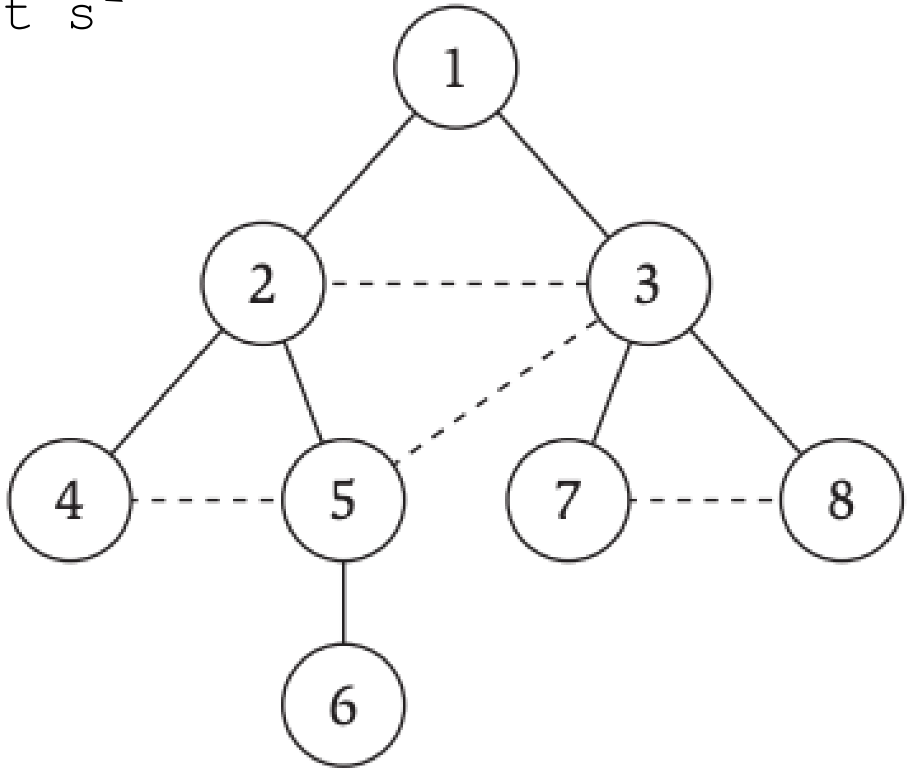
Discovered:
Queue:
u:
v:



A New BFS

```
BFS(s):
   Initialize Discovered to be a node index array of false
   Set Discovered[s] = true
   Initialize Q to be a Queue with one element s
   While Q is not empty:
      u = Q.dequeue()
      For each edge (u, v) incident to u:
            If Discovered[v] = false:
                  Set Discovered[u] = true
                  Add v to Q
```

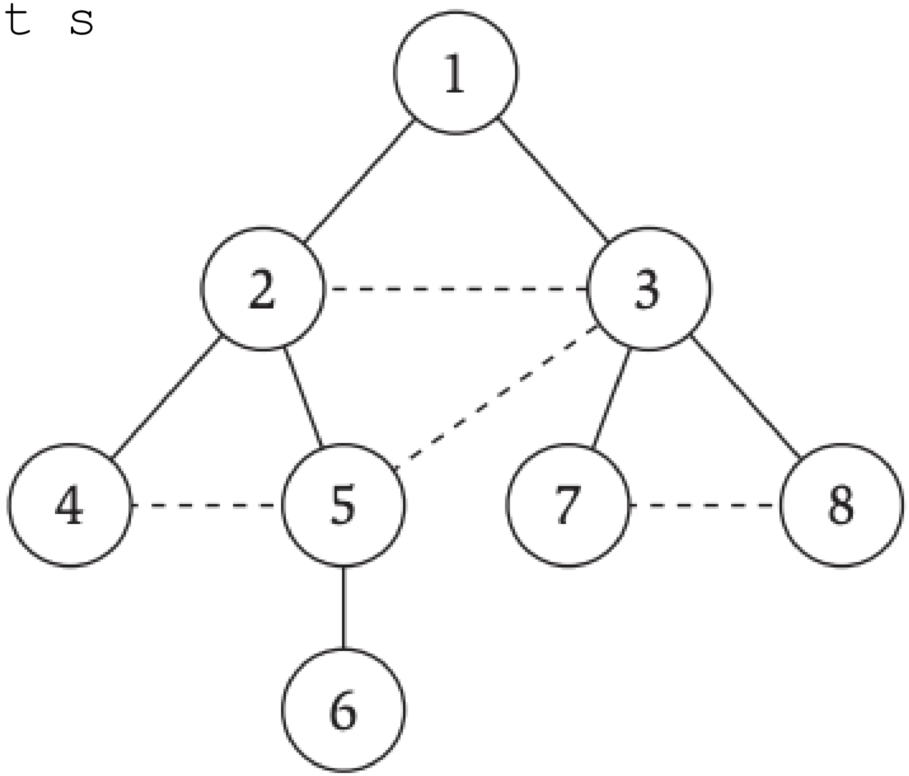
Discovered : {1}
Queue : {1}
u :
v :



A New BFS

```
BFS(s):
   Initialize Discovered to be a node index array of false
   Set Discovered[s] = true
   Initialize Q to be a Queue with one element s
   While Q is not empty:
      u = Q.dequeue()
      For each edge (u, v) incident to u:
            If Discovered[v] = false:
                  Set Discovered[u] = true
                  Add v to Q
```

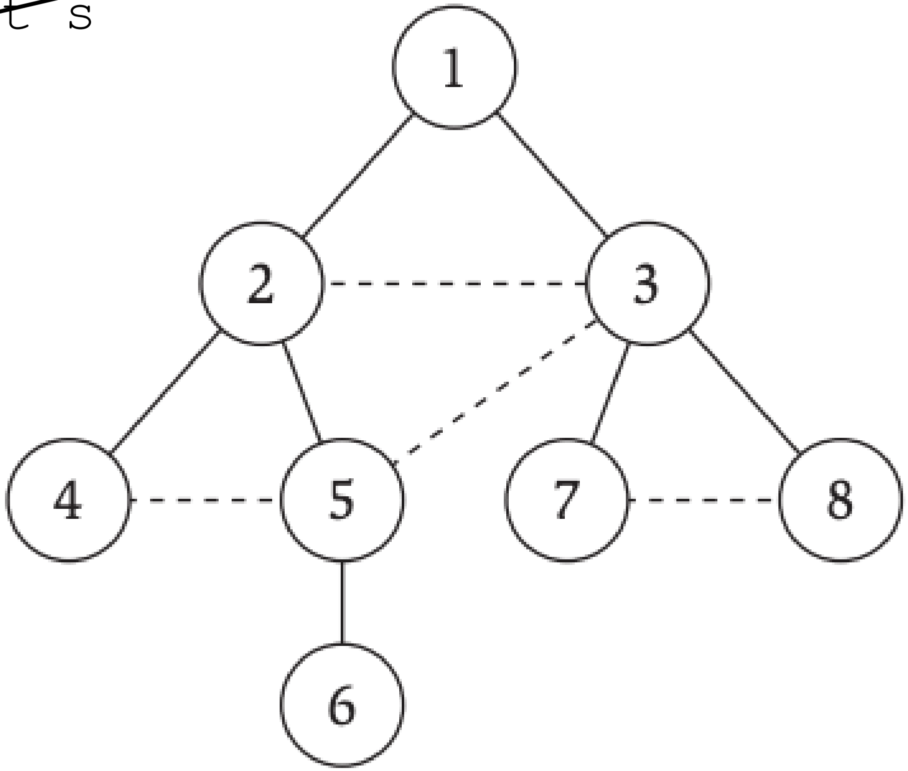
Discovered : {1}
Queue : {1}
u :
v :



A New BFS

```
BFS(s):
   Initialize Discovered to be a node index array of false
   Set Discovered[s] = true
   Initialize Q to be a Queue with one element s
   While Q is not empty:
      u = Q.dequeue()
      For each edge (u, v) incident to u:
            If Discovered[v] = false:
                  Set Discovered[u] = true
                  Add v to Q
```

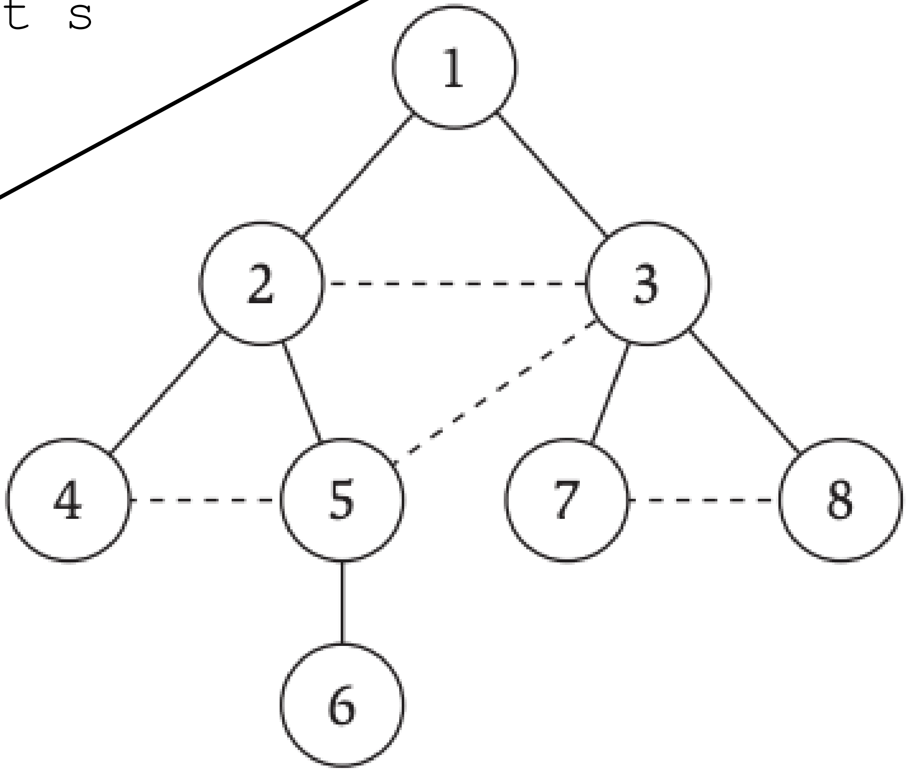
Discovered : {1}
Queue : {}
u : {1}
v :



A New BFS

```
BFS(s):
    Initialize Discovered to be a node index array of false
    Set Discovered[s] = true
    Initialize Q to be a Queue with one element s
    While Q is not empty:
        u = Q.dequeue()
        For each edge (u, v) incident to u:
                If Discovered[v] = false:
                        Set Discovered[u] = true
                        Add v to Q
```

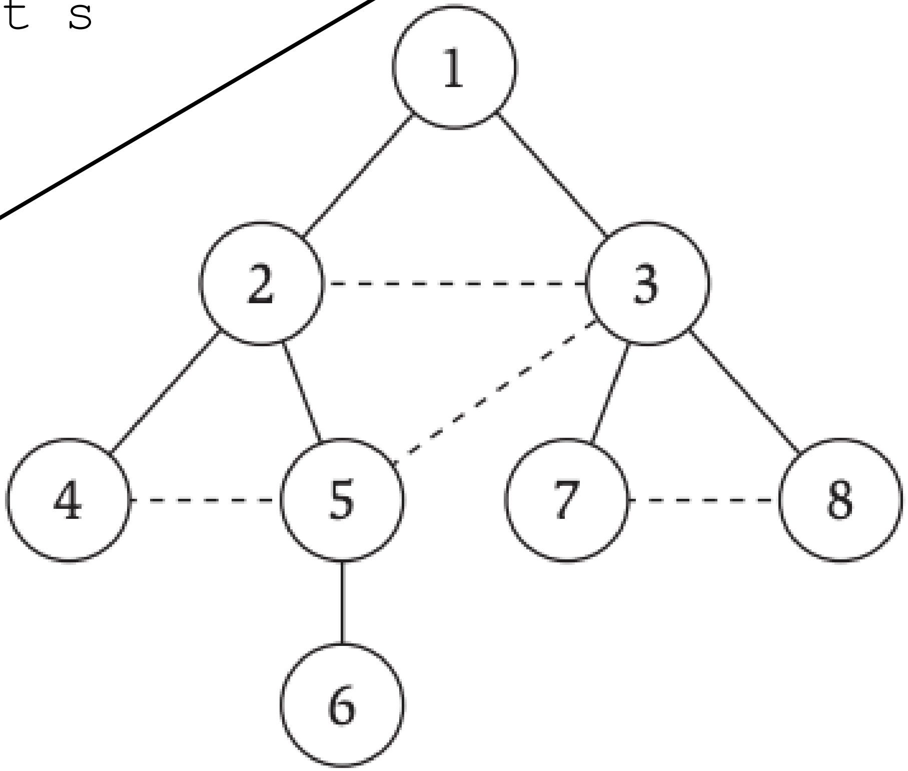
Discovered : {1}
Queue : {}
u : 1
v : 2



A New BFS

```
BFS(s):
   Initialize Discovered to be a node index array of false
   Set Discovered[s] = true
   Initialize Q to be a Queue with one element s
   While Q is not empty:
      u = Q.dequeue()
      For each edge (u, v) incident to u:
            If Discovered[v] = false:
                  Set Discovered[u] = true
                  Add v to Q
```

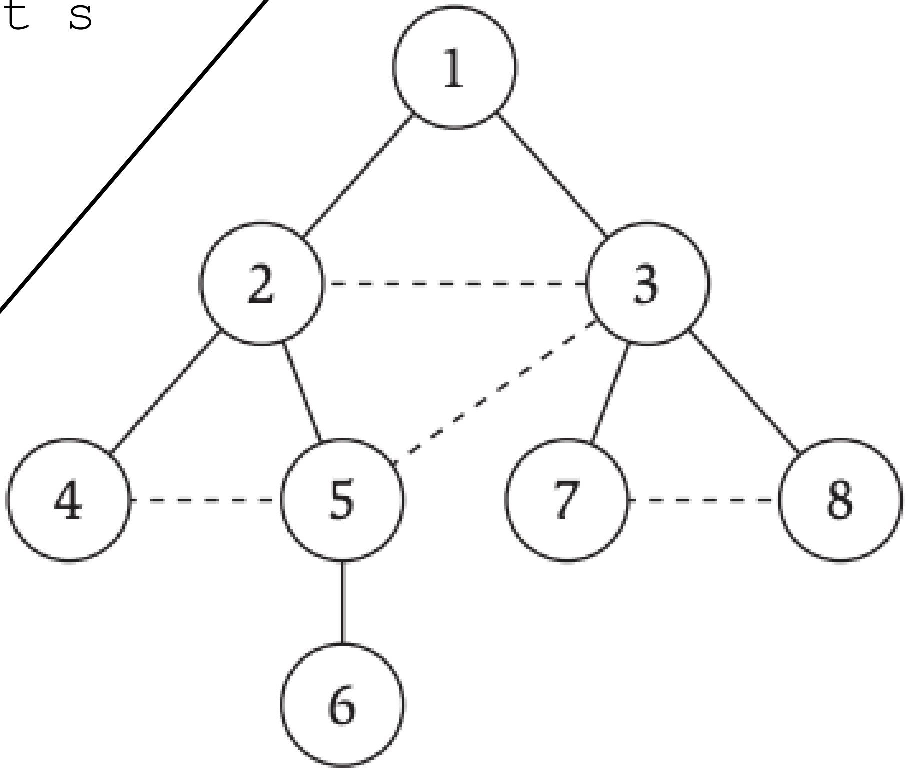
Discovered : {1}
Queue : {}
u : 1
v : 2



A New BFS

```
BFS(s):
    Initialize Discovered to be a node index array of false
    Set Discovered[s] = true
    Initialize Q to be a Queue with one element s
    While Q is not empty:
        u = Q.dequeue()
        For each edge (u, v) incident to u:
                If Discovered[v] = false:
                        Set Discovered[u] = true
                        Add v to Q
```

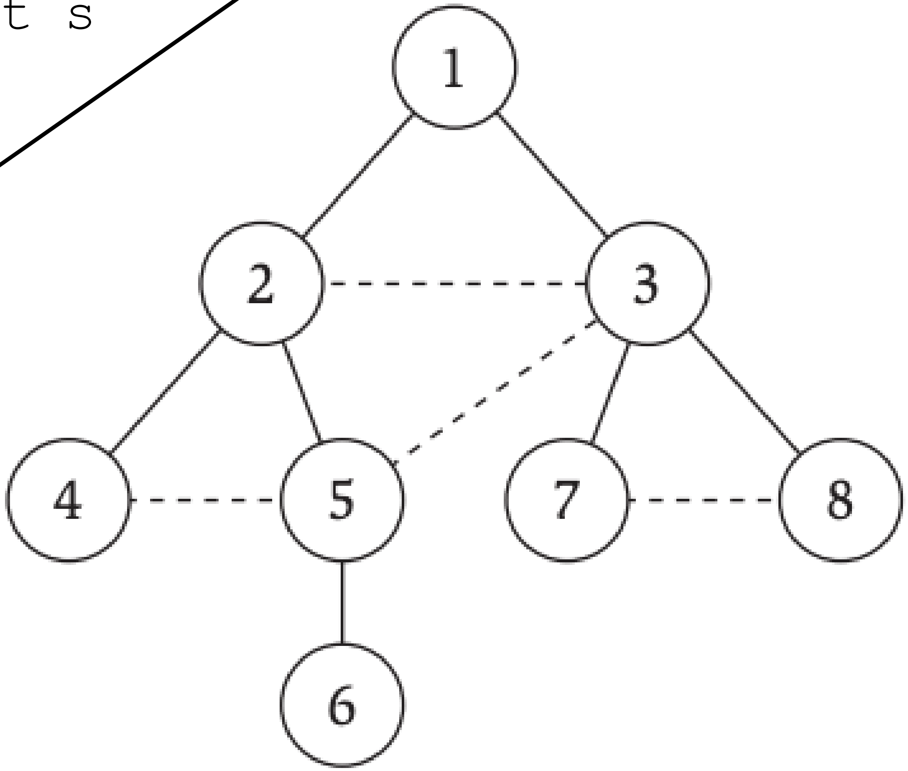
Discovered : {1,2}
Queue : {}
u : 1
v : 2



A New BFS

```
BFS(s):
   Initialize Discovered to be a node index array of false
   Set Discovered[s] = true
   Initialize Q to be a Queue with one element s
   While Q is not empty:
      u = Q.dequeue()
      For each edge (u, v) incident to u:
            If Discovered[v] = false:
                  Set Discovered[u] = true
                  Add v to Q
```

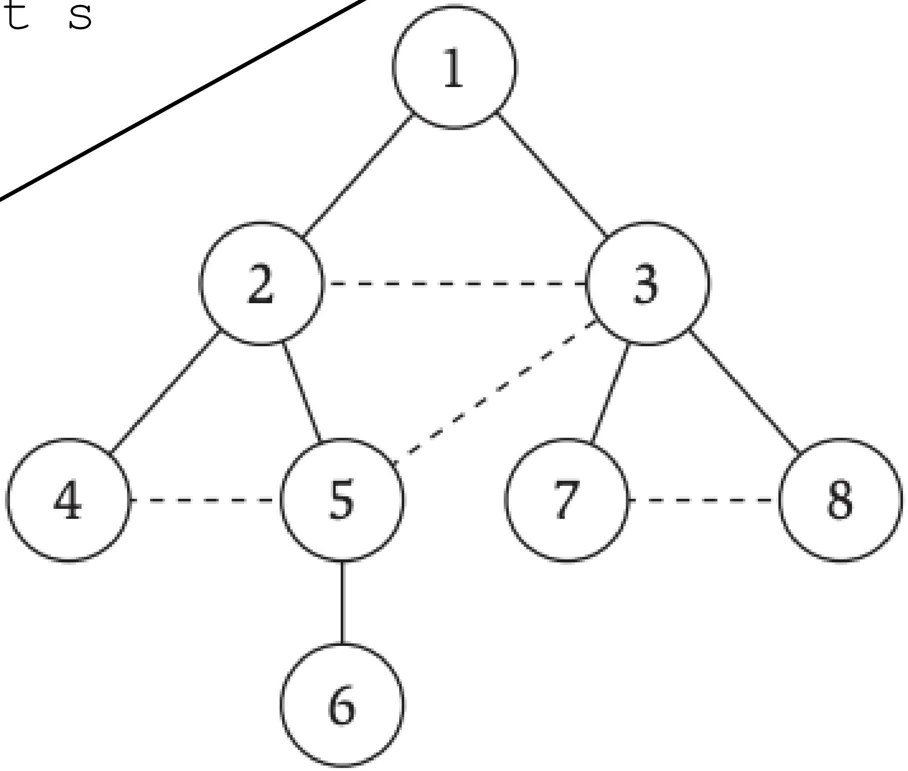
Discovered : {1,2}
Queue : {2}
u : 1
v : 2



A New BFS

```
BFS(s):
   Initialize Discovered to be a node index array of false
   Set Discovered[s] = true
   Initialize Q to be a Queue with one element s
   While Q is not empty:
      u = Q.dequeue()
      For each edge (u, v) incident to u:
            If Discovered[v] = false:
                  Set Discovered[u] = true
                  Add v to Q
```

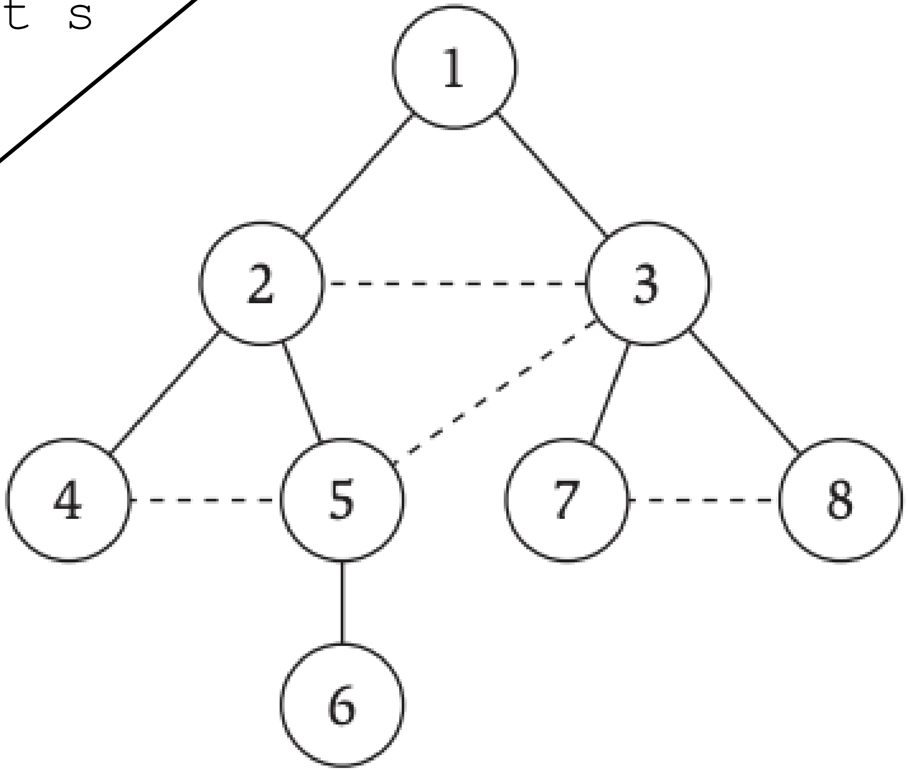
Discovered : {1,2}
Queue : {2}
u : 1
v : 3



A New BFS

```
BFS(s):
   Initialize Discovered to be a node index array of false
   Set Discovered[s] = true
   Initialize Q to be a Queue with one element s
   While Q is not empty:
      u = Q.dequeue()
      For each edge (u, v) incident to u:
            If Discovered[v] = false:
                  Set Discovered[u] = true
                  Add v to Q
```

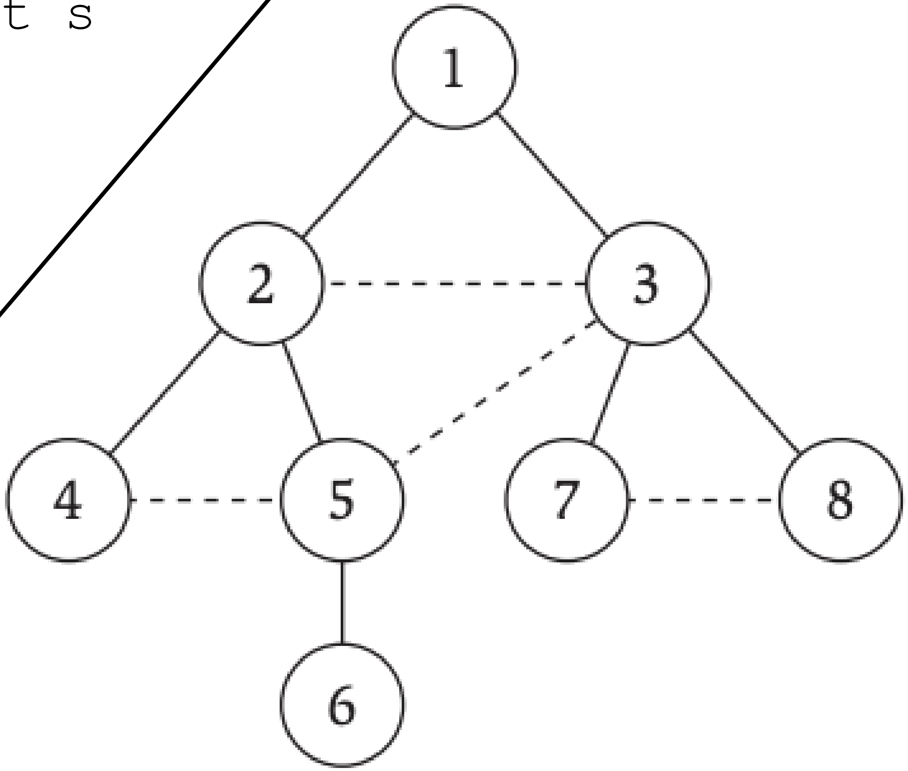
Discovered : {1,2}
Queue : {2}
u : 1
v : 3



A New BFS

```
BFS(s):
   Initialize Discovered to be a node index array of false
   Set Discovered[s] = true
   Initialize Q to be a Queue with one element s
   While Q is not empty:
      u = Q.dequeue()
      For each edge (u, v) incident to u:
            If Discovered[v] = false:
                  Set Discovered[u] = true
                  Add v to Q
```

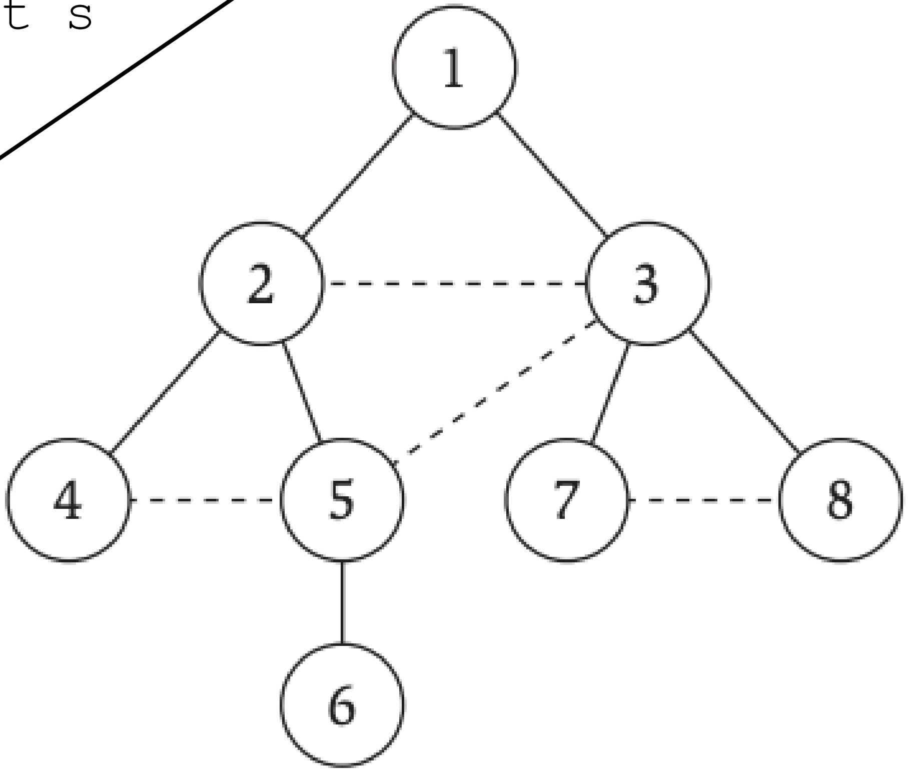
Discovered : {1,2,3}
Queue : {2}
u : 1
v : 3



A New BFS

```
BFS(s):
   Initialize Discovered to be a node index array of false
   Set Discovered[s] = true
   Initialize Q to be a Queue with one element s
   While Q is not empty:
      u = Q.dequeue()
      For each edge (u, v) incident to u:
            If Discovered[v] = false:
                  Set Discovered[u] = true
                  Add v to Q
```

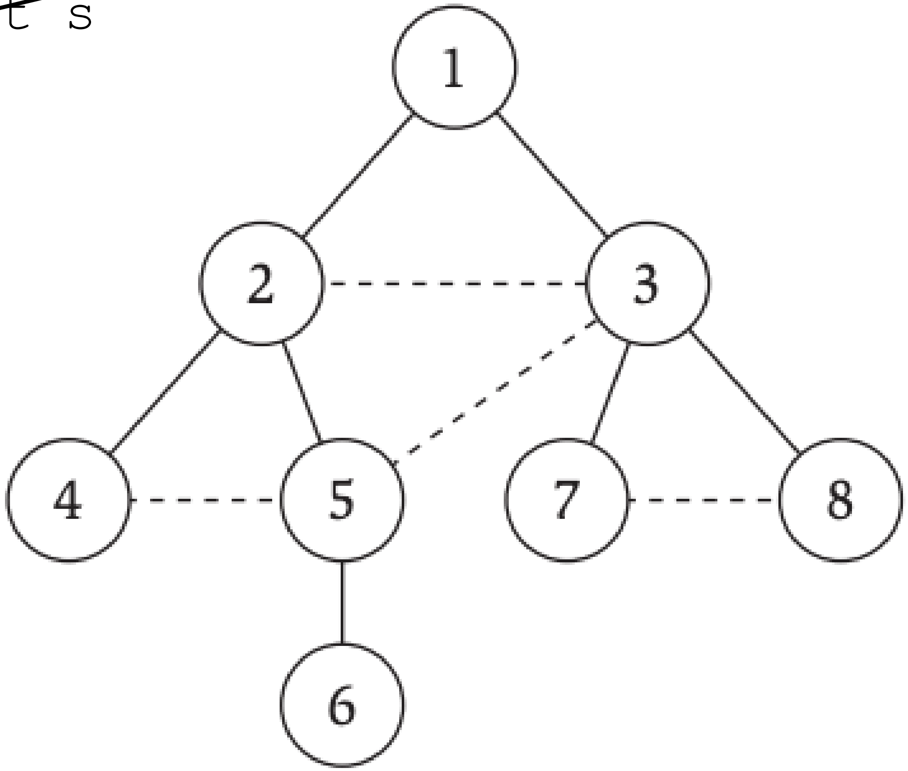
Discovered : {1,2,3}
Queue : {2, 3}
u : 1
v : 3



A New BFS

```
BFS(s):
   Initialize Discovered to be a node index array of false
   Set Discovered[s] = true
   Initialize Q to be a Queue with one element s
   While Q is not empty:
      u = Q.dequeue()
      For each edge (u, v) incident to u:
            If Discovered[v] = false:
                  Set Discovered[u] = true
                  Add v to Q
```

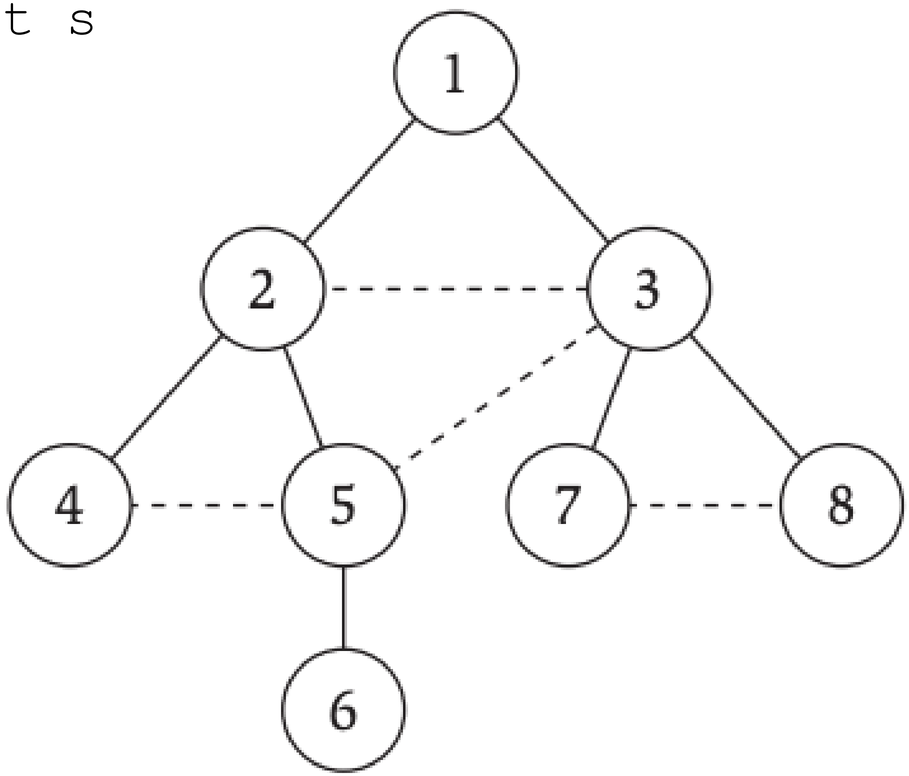
Discovered : {1,2,3}
Queue : {3}
u : 2
v : ...



A New BFS

```
BFS(s):
   Initialize Discovered to be a node index array of false
   Set Discovered[s] = true
   Initialize Q to be a Queue with one element s
   While Q is not empty:
      u = Q.dequeue()
      For each edge (u, v) incident to u:
            If Discovered[v] = false:
                  Set Discovered[u] = true
                  Add v to Q
```

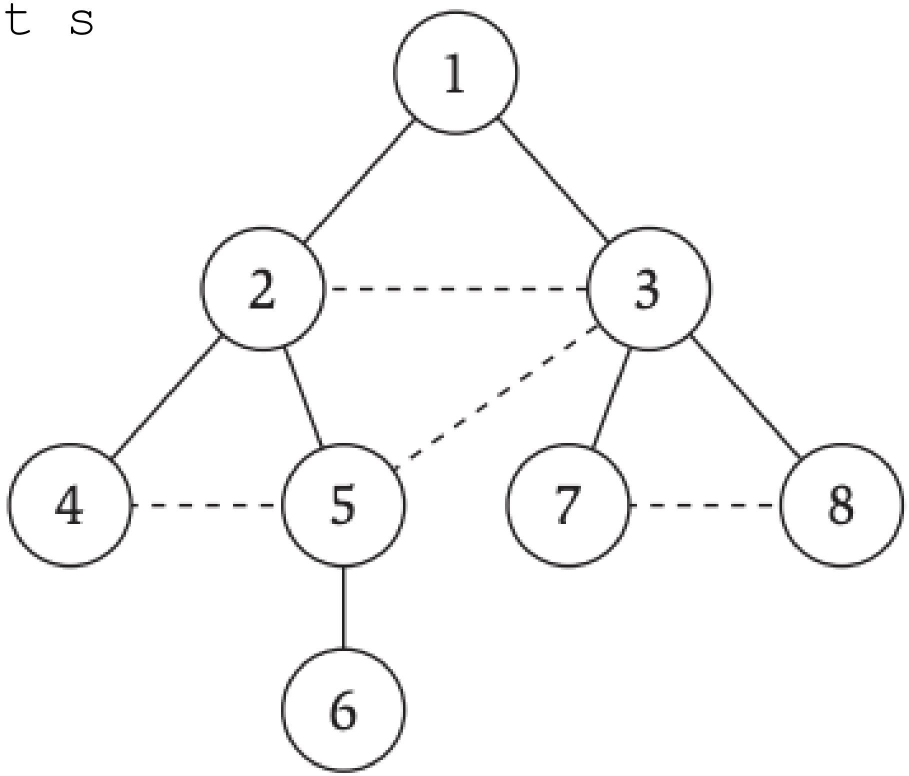
Discovered : {1,2,3, 4, 5}
Queue : {3, 4, 5}
u : 2
v : ...



A New BFS

```
BFS(s):
   Initialize Discovered to be a node index array of false
   Set Discovered[s] = true
   Initialize Q to be a Queue with one element s
   While Q is not empty:
      u = Q.dequeue()
      For each edge (u, v) incident to u:
            If Discovered[v] = false:
                  Set Discovered[u] = true
                  Add v to Q
```

Discovered : {1,2,3, 4, 5}
Queue : {4, 5}
u : 3
v : ...



A New BFS

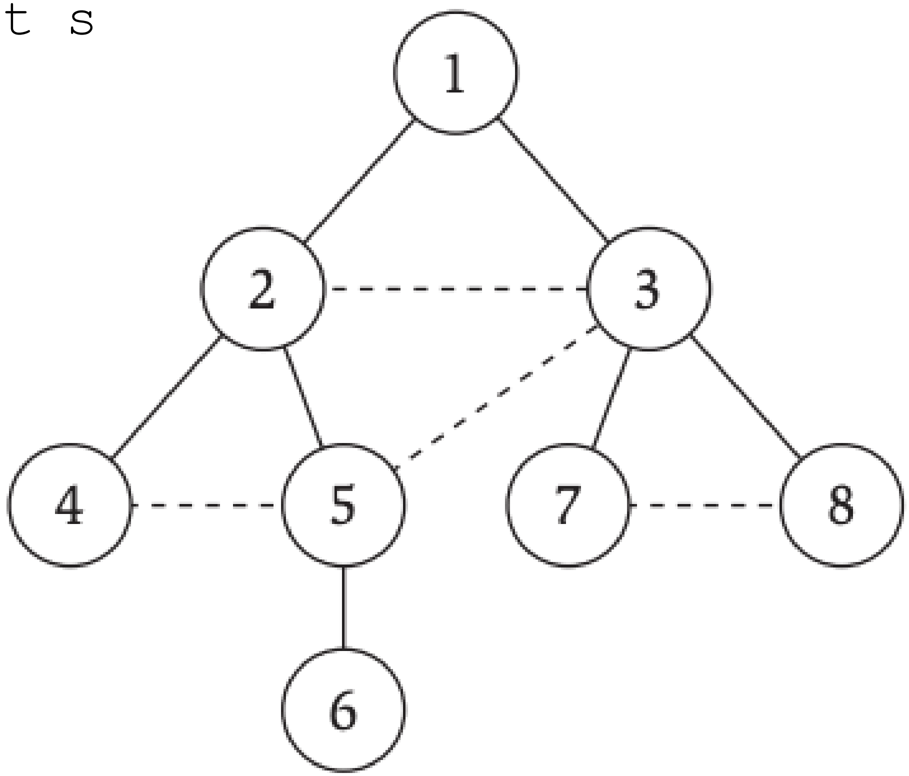
```
BFS(s):
   Initialize Discovered to be a node index array of false
   Set Discovered[s] = true
   Initialize Q to be a Queue with one element s
   While Q is not empty:
      u = Q.dequeue()
      For each edge (u, v) incident to u:
            If Discovered[v] = false:
                  Set Discovered[u] = true
                  Add v to Q
```

Discovered : {1,2,3,4,5,7,8,6}

Queue : {7, 8}

u : 5

v : ...



A New BFS

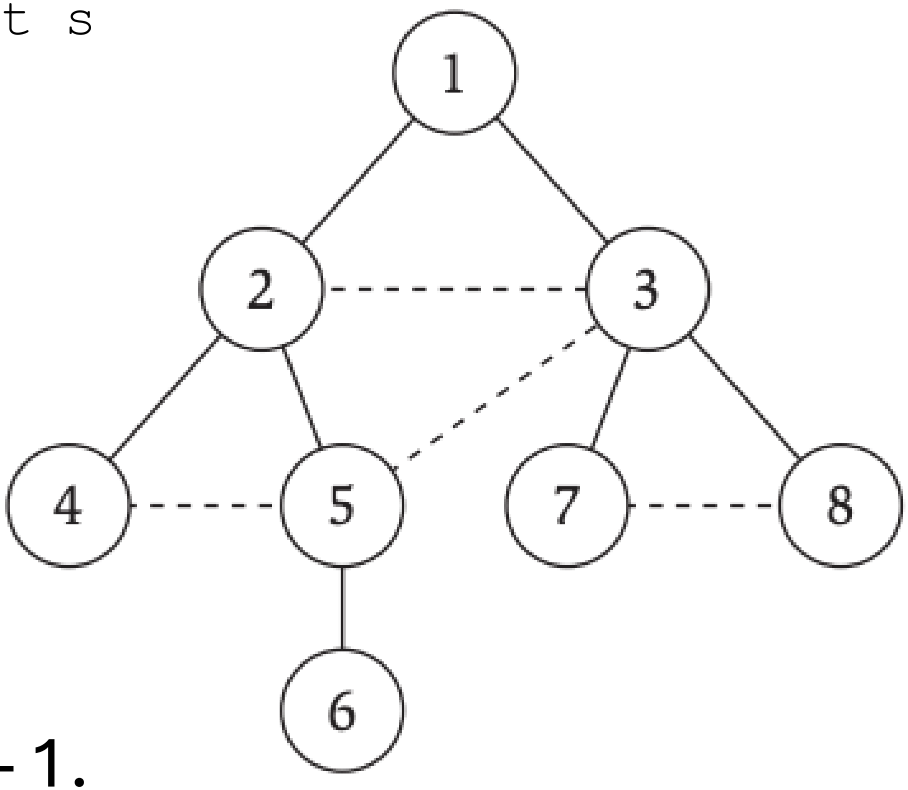
```
BFS(s):
   Initialize Discovered to be a node index array of false
   Set Discovered[s] = true
   Initialize Q to be a Queue with one element s
   While Q is not empty:
      u = Q.dequeue()
      For each edge (u, v) incident to u:
            If Discovered[v] = false:
                  Set Discovered[u] = true
                  Add v to Q
```

Discovered : {1,2,3,4,5,7,8,6}

Queue : {}

u : …

v : …



Note: We never visit anything in level i before i – 1.

Q: What is the runtime? (Assume Linked List)

```
BFS(s):
   Initialize Discovered to be a node index array of false
   Set Discovered[s] = true
   Initialize Q to be a Queue with one element s
   While Q is not empty:
      u = Q.dequeue()
      For each edge (u, v) incident to u:
            If Discovered[v] = false:
                  Set Discovered[u] = true
                  Add v to Q
```

Q: What is the runtime? O(m+n)

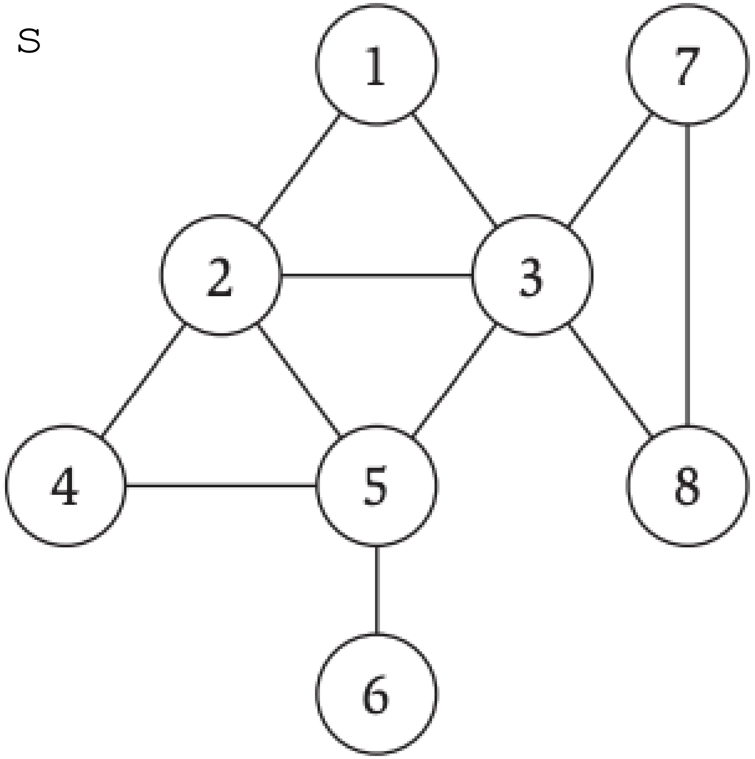
```
BFS(s):
    Initialize Discovered to be a node index array of false
    Set Discovered[s] = true
    Initialize Q to be a Queue with one element s
    While Q is not empty:
        u = Q.dequeue()
        For each edge (u, v) incident to u:
                If Discovered[v] = false:
                        Set Discovered[u] = true
                        Add v to Q
```

Note: We only do this outer loop at most once per vertex.

Note: We only do this inner loop at most twice per edge.

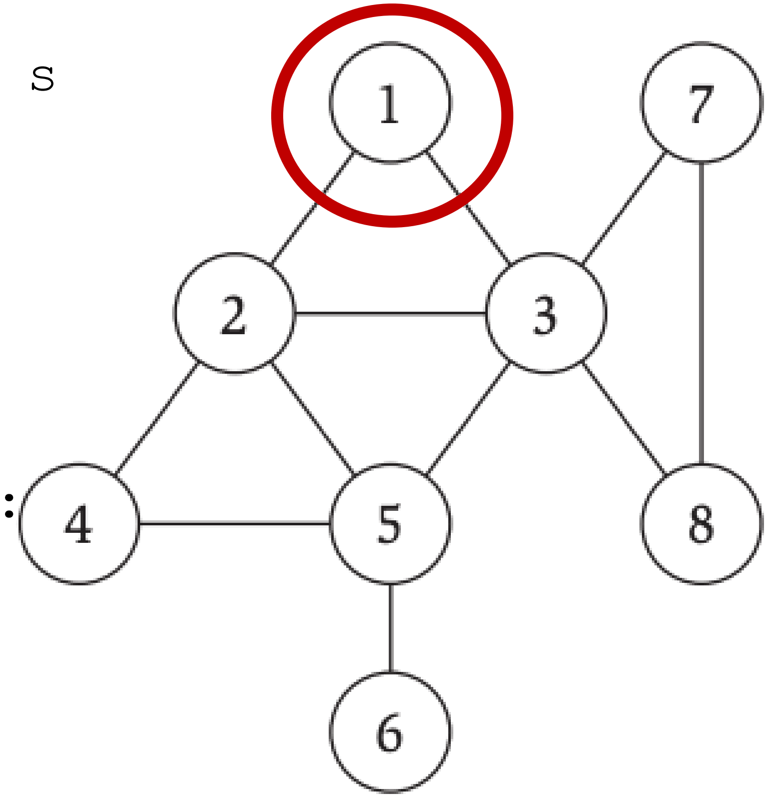
Q: What if we change the queue to a stack?

```
???(s):
   Initialize Explored to be a node index array of false
   Set Explored[s] = true
   Initialize Q to be a **Stack** with one element s
   While Q is not empty:
      u = Q.remove()
      If Explored[u] = false:
            Explored[u] = true
            For each edge (u, v) incident to u:
                  Add v to Q
```



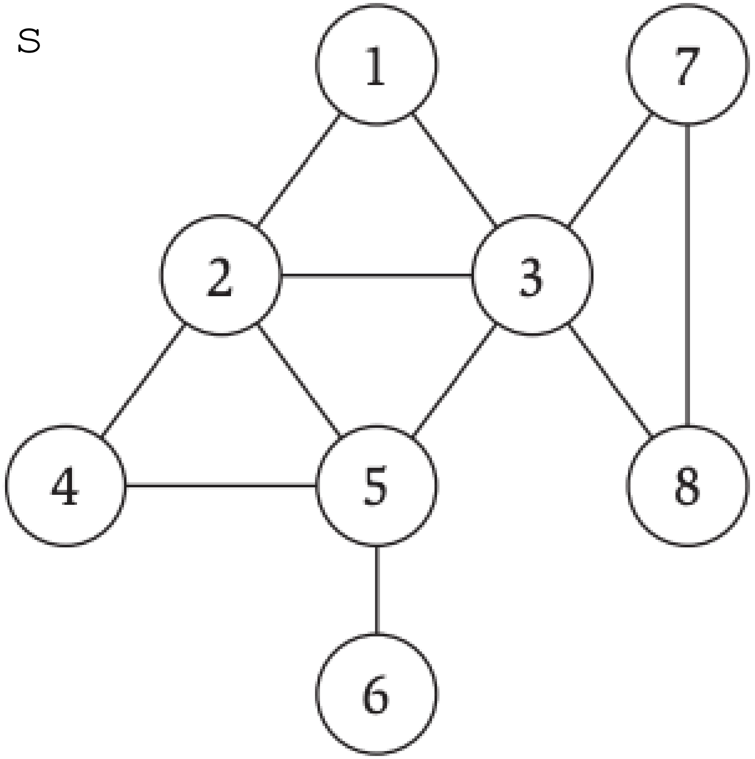
Q: What if we change the queue to a stack?

```
???(s):
   Initialize Explored to be a node index array of false
   Set Explored[s] = true
   Initialize Q to be a **Stack** with one element s
   While Q is not empty:
      u = Q.remove()
      If Explored[u] = false:
            Explored[u] = true
            For each edge (u, v) incident to u:
                  Add v to Q
```



Q: What if we change the queue to a stack?

```
???(s):
   Initialize Explored to be a node index array of false
   Set Explored[s] = true
   Initialize Q to be a **Stack** with one element s
   While Q is not empty:
      u = Q.remove()
      If Explored[u] = false:
            Explored[u] = true
            For each edge (u, v) incident to u:
                  Add v to Q
```



A: We DFS

```
DFS(s):
   Initialize Explored to be a node index array of false
   Set Explored[s] = true
   Initialize Q to be a Stack with one element s
   While Q is not empty:
      u = Q.remove()
      If Explored[u] = false:
            Explored[u] = true
            For each edge (u, v) incident to u:
                  Add v to Q
```

Q: What is the runtime of DFS?

```
DFSs):
   Initialize Explored to be a node index array of false
   Set Explored[s] = true
   Initialize Q to be a Stack with one element s
   While Q is not empty:
      u = Q.remove()
      If Explored[u] = false:
            Explored[u] = true
            For each edge (u, v) incident to u:
                  Add v to Q
```

Q: What is the runtime of DFS? O(m+n)

```
DFSs):
    Initialize Explored to be a node index array of false
    Set Explored[s] = true
    Initialize Q to be a Stack with one element s
    While Q is not empty:
        u = Q.remove()
        If Explored[u] = false:
                Explored[u] = true
                For each edge (u, v) incident to u:
                        Add v to Q
```

Q: How many times does this loop run for u?

Q: What is the runtime of DFS? O(m+n)

```
DFSs):
   Initialize Explored to be a node index array of false
   Set Explored[s] = true
   Initialize Q to be a Stack with one element s
   While Q is not empty: A: At most once for each edge
      u = Q.remove()
      If Explored[u] = false:
            Explored[u] = true
            For each edge (u, v) incident to u:
                  Add v to Q
```

Problem: Bipartite Graph

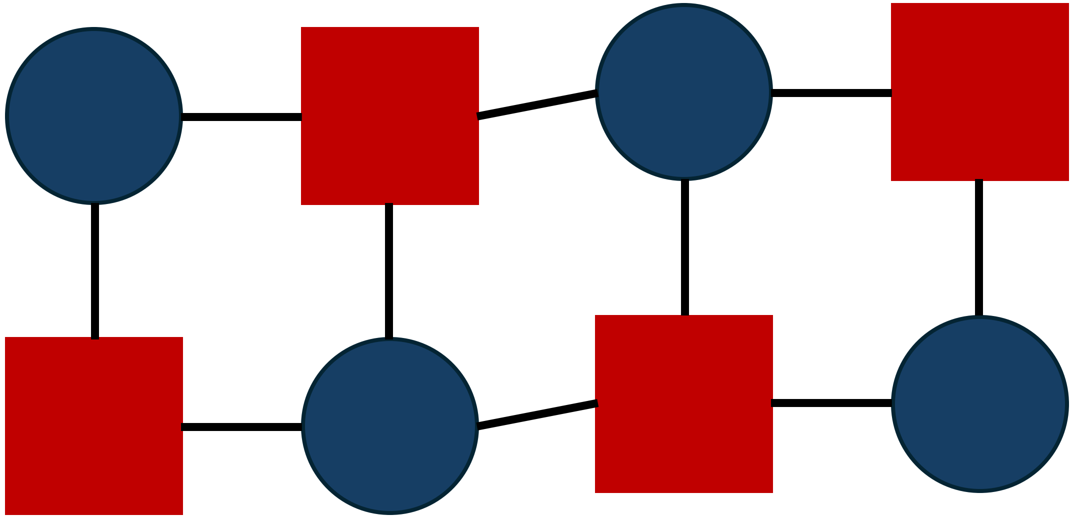
Input: A graph $\mathrm{G} = (\mathrm{V}, \mathrm{E})$

Output: True if G is bipartite and False otherwise.

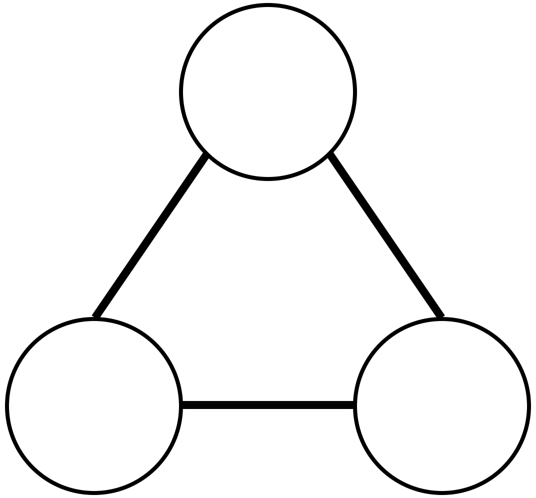
Def: We say that a graph is bipartite if we can partition the vertices V into two groups L and R such that each edge has one endpoint in L and the other endpoint in R.

Problem: Bipartite Graph

Def: We say that a graph is bipartite if we can partition the vertices V into two groups L and R such that each edge has one endpoint in L and the other endpoint in R.



Bipartite



Not Bipartite

Problem: Bipartite Graph

Input: A graph $G = (V, E)$

Output: True if G is bipartite and False otherwise.

Algorithm Ideas:

Problem: Bipartite Graph

Input: A graph $G = (V, E)$

Output: True if G is bipartite and False otherwise.

Algorithm Ideas: We will find the layering produced by BFS and color odd levels Red and even layers Blue.

Q: When will this fail?

Problem: Bipartite Graph

Input: A graph $G = (V, E)$

Output: True if G is bipartite and False otherwise.

Algorithm Ideas: We will find the layering produced by BFS and color odd levels Red and even layers Blue. This will fail if there is a cross edge in one of the layers.

Q: What does this imply?

Problem: Bipartite Graph

Input: A graph $G = (V, E)$

Output: True if G is bipartite and False otherwise.

Algorithm Ideas: We will find the layering produced by BFS and color odd levels Red and even layers Blue. This will fail if there is a cross edge in one of the layers. This implies there is an odd cycle in the graph.

Q: Is that a problem?

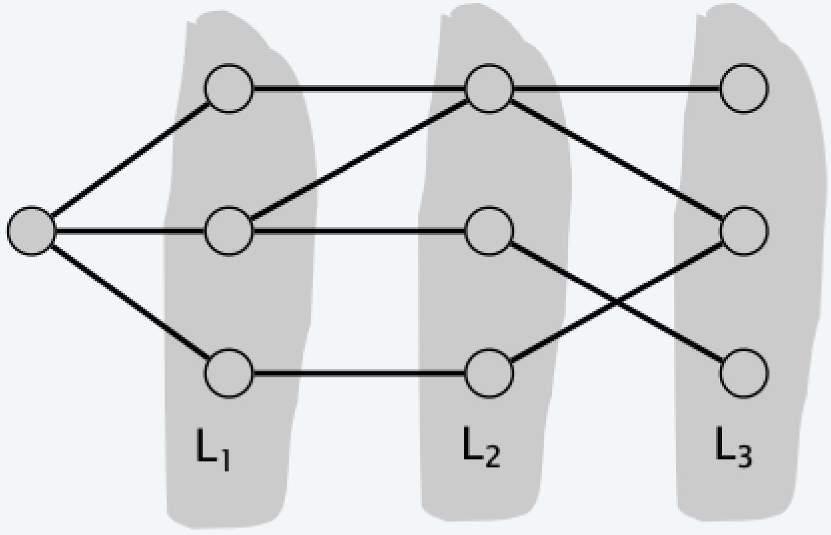
Problem: Bipartite Graph

Input: A graph $G = (V, E)$

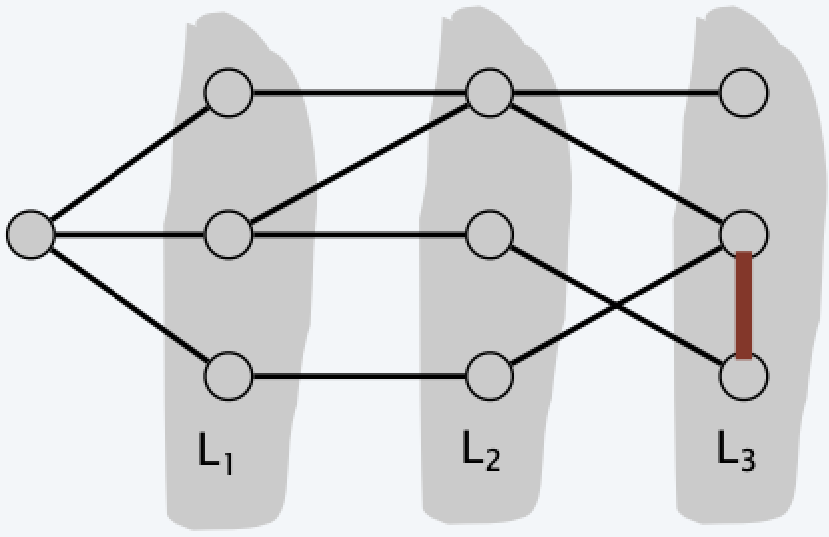
Output: True if G is bipartite and False otherwise.

Algorithm Ideas: We will find the layering produced by BFS and color odd levels Red and even layers Blue. This will fail if there is a cross edge in one of the layers. This implies there is an odd cycle in the graph. We can show that a graph is not bipartite if it contains an odd cycle.

Problem: Bipartite Graph

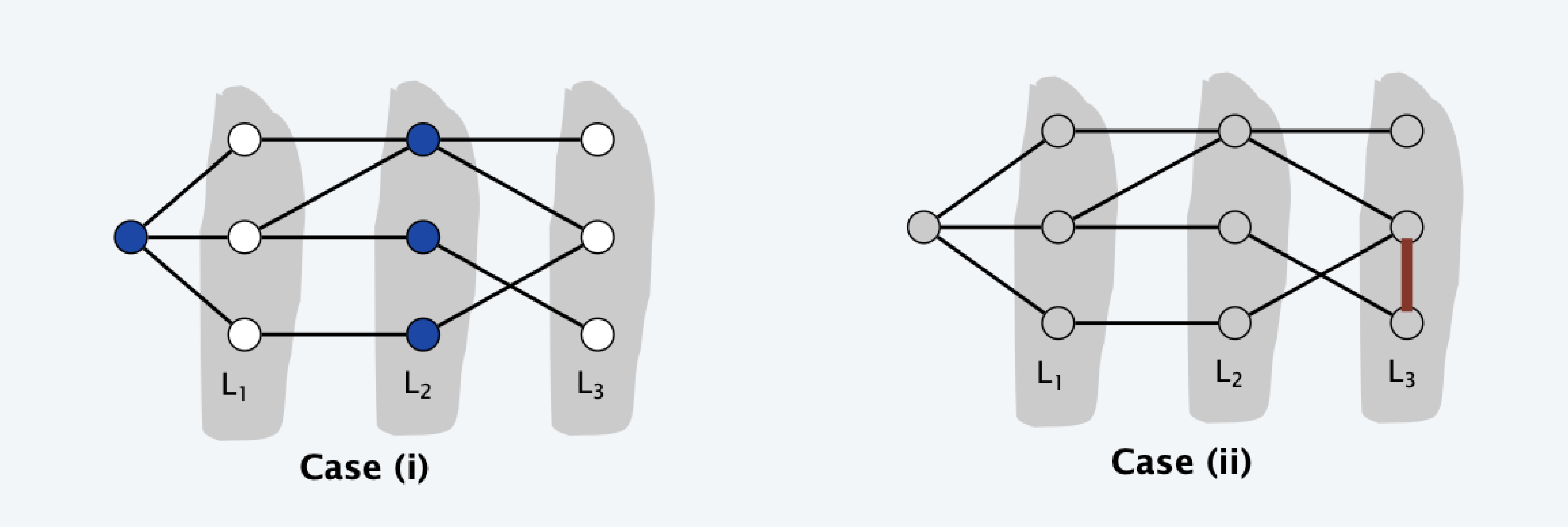


Case (i)



Case (ii)

Problem: Bipartite Graph

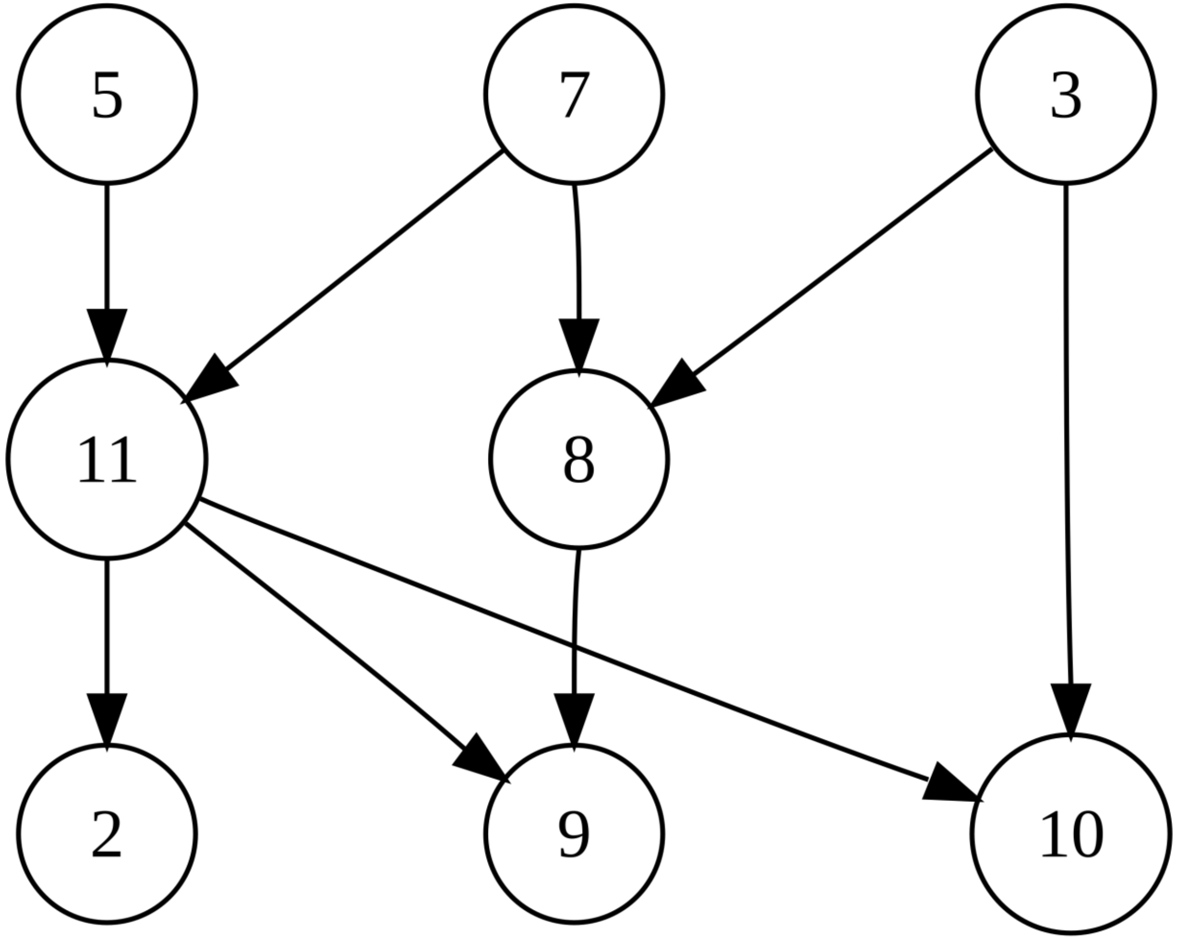


Is bipartite!

Is not bipartite!

Directed Graphs

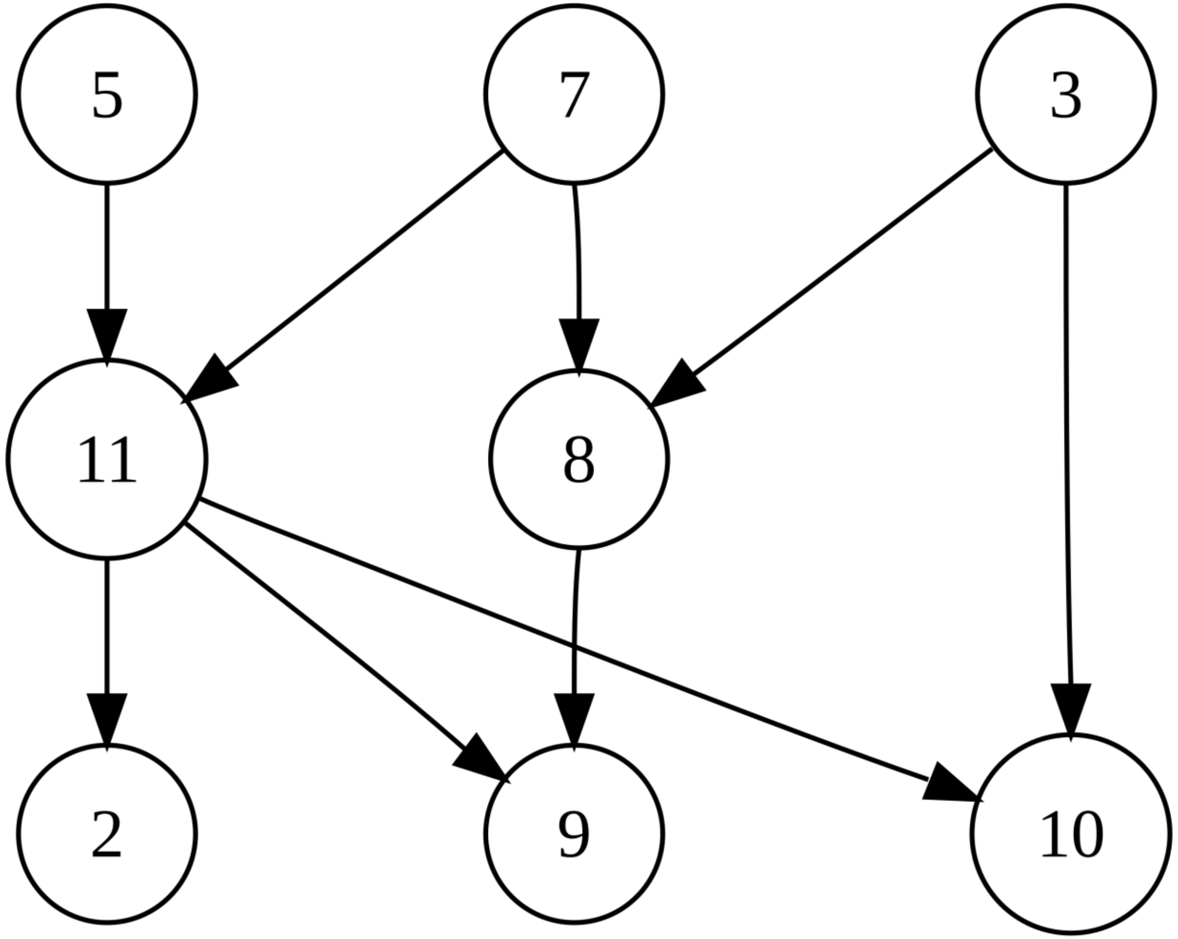
- A directed graph is a graph $G = (V, E)$ such that the edges are directed:
  - That is, each edge $(u, v) \in E$ has an ordering (i.e., a start and an end vertex).



By Johannes Rössel (talk) - Own work, Public Domain, https://commons.wikimedia.org/w/index.php?curid=5559952

Examples Directed Graphs

- What might the nodes represent?
- What might the edges represent?



By Johannes Rössel (talk) - Own work, Public Domain, https://commons.wikimedia.org/w/index.php?curid=5559952