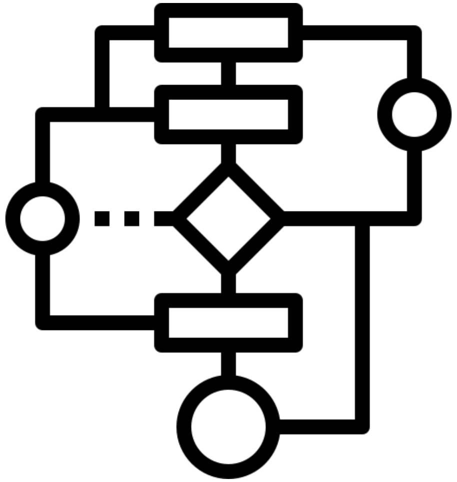
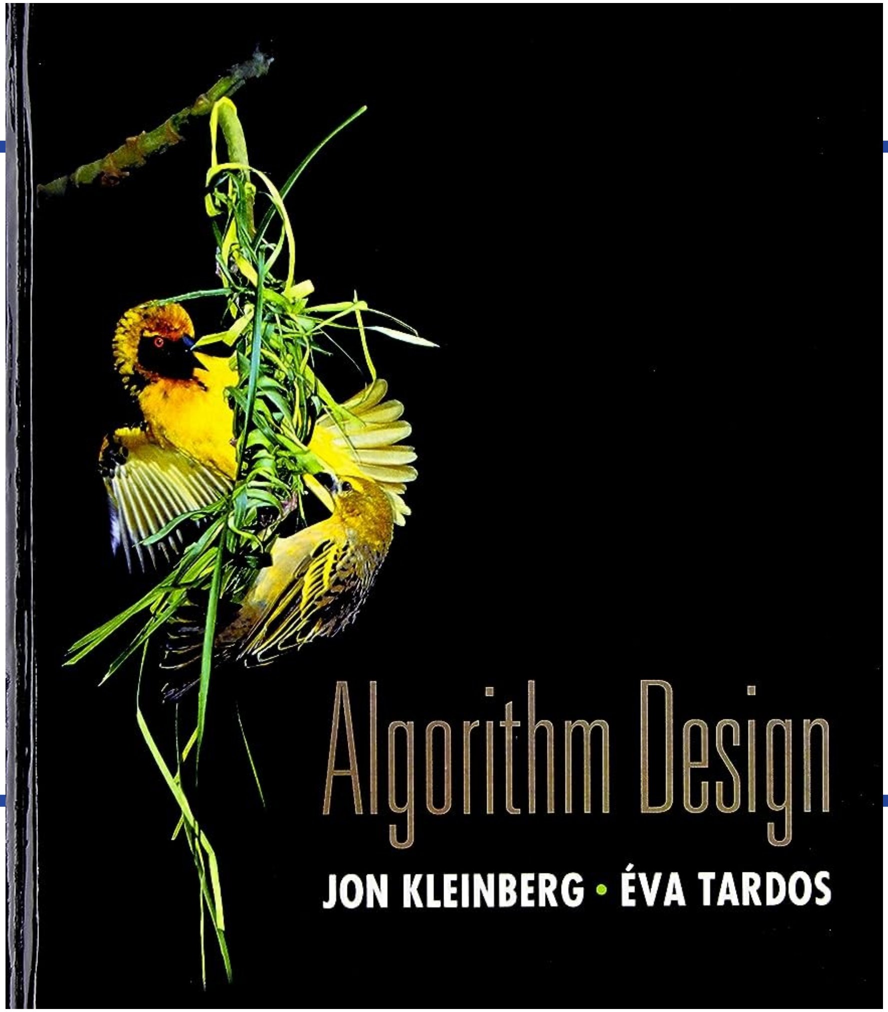


CS 310: Algorithms

Lecture 7

Instructor: Naveed Anwar Bhatti



Chapter 3: Graphs

BFS: Live Poll 1

Consider a complete undirected graph where every vertex ***V*** has an edge with every other vertex. You are going to perform a Breadth-First Search (BFS) on this graph.

Which of the following expressions give equivalent time complexity in terms of the Big O notation of the BFS for this graph?

A. $V^2 = \mathbf{O(\boldsymbol{V^2})}$ ✓

B. 2E = $V(V-1) = V^2 - V = \mathbf{O(\boldsymbol{V^2})}$ ✓ 😈

C. V+2E $= V + V(V-1) = V^2 = \mathbf{O(\boldsymbol{V^2})}$ ✓ 😈

D. V+E $= V + (V(V-1))/2 = V + (V^2 - V)/2 = \mathbf{O(\boldsymbol{V^2})}$ ✓ 😈

E. All of Above

F. None of Above

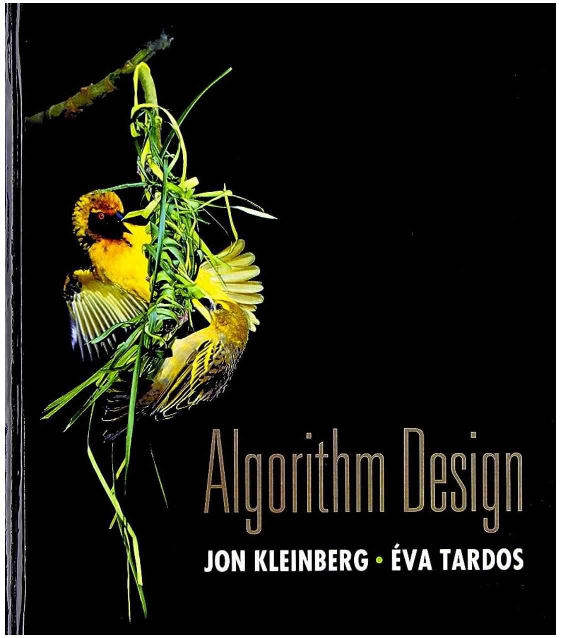
Breadth First Search: Analysis

Theorem: The above implementation of BFS runs in O(m + n) time if the graph is given by its adjacency representation.

- **Proof:**
  - Easy to prove **O(n^2)** running time:
    - at most n lists L[i]
    - each node occurs on at most one list; for loop runs $\leq$ n times
    - when we consider node u, there are $\leq$ n incident edges (u, v), and we spend O(1) processing each edge
  - Actually, runs in **O(m + n)** time:
    - when we consider node u, there are deg(u) incident edges (u, v)
    - total time processing edges is $\Sigma_{u \in V} \deg(u) = 2m$ ▪

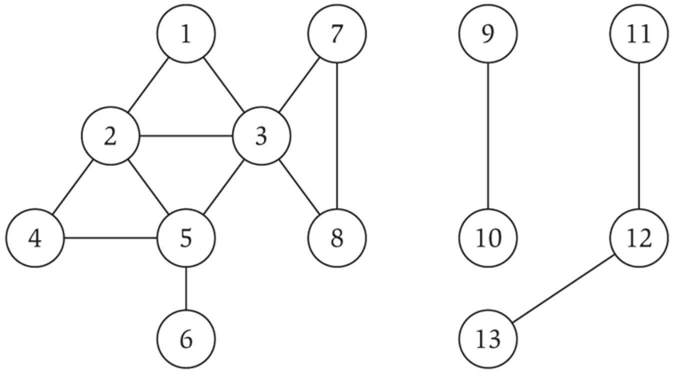
each edge (u, v) is counted exactly twice
in sum: once in deg(u) and once in deg(v)

Read the book



Connected Component

- Connected component. Find all nodes reachable from s.

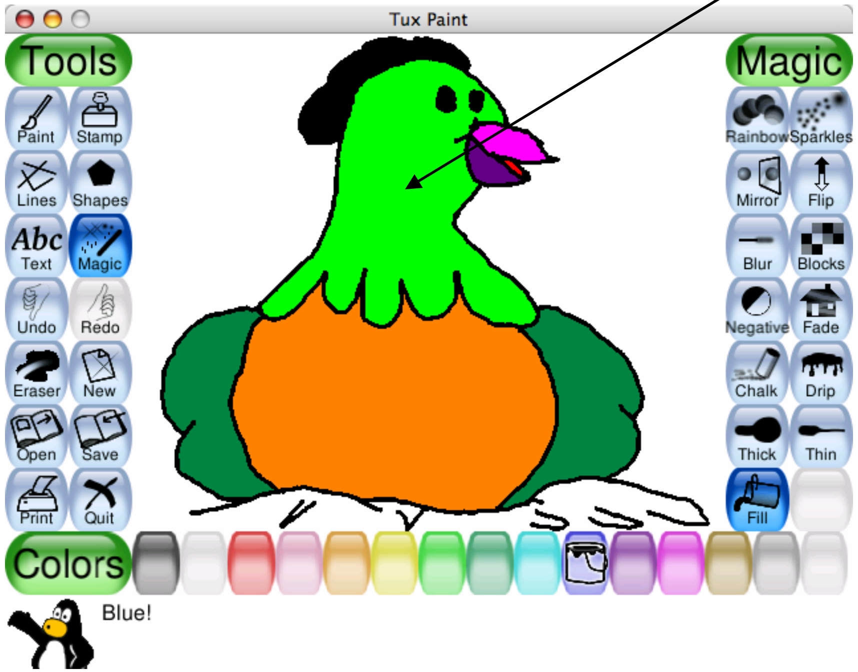


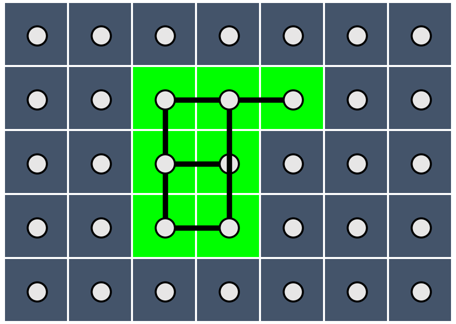
- Connected component containing node 1 = { 1, 2, 3, 4, 5, 6, 7, 8 }.

Flood Fill

- Flood fill. Given lime green pixel in an image, change color of entire blob of neighboring lime pixels to blue.
  - Node: pixel.
  - Edge: two neighboring lime pixels.
  - Blob: connected component of lime pixels.

recolor lime green blob to blue

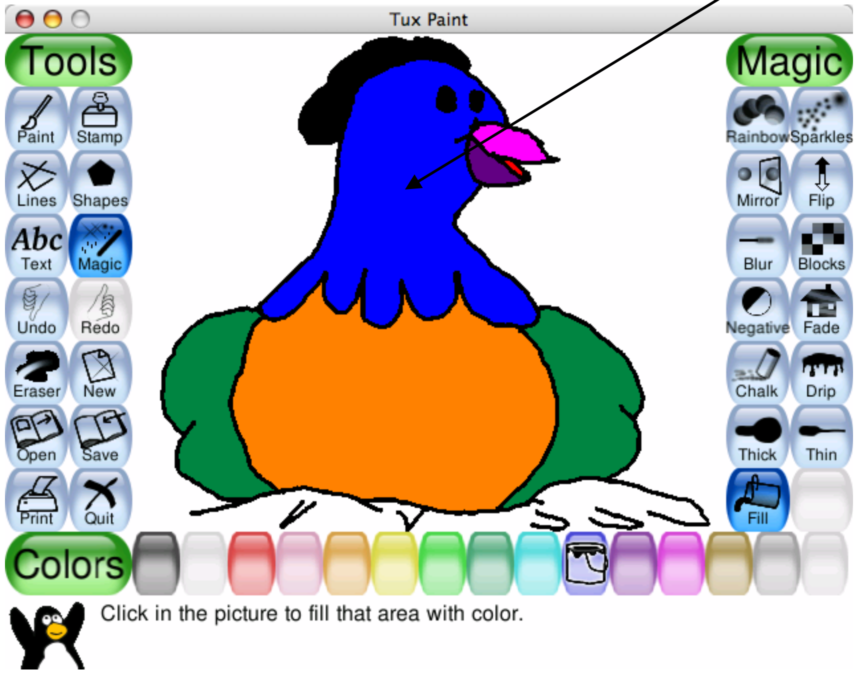


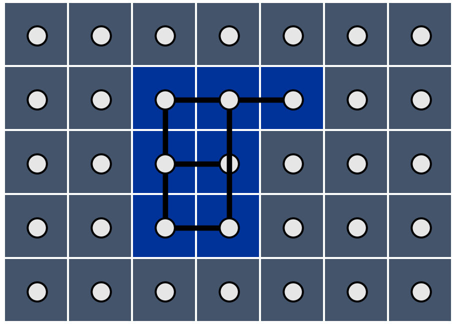


Flood Fill

- Flood fill. Given lime green pixel in an image, change color of entire blob of neighboring lime pixels to blue.
  - Node: pixel.
  - Edge: two neighboring lime pixels.
  - Blob: connected component of lime pixels.

recolor lime green blob to blue

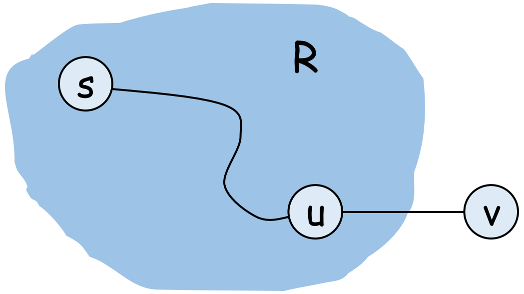




Connected Component

- Connected component. Find all nodes reachable from s.

```
R will consist of nodes to which s has a path
Initially R = {s}
While there is an edge (u, v) where u ∈ R and v ∉ R
  Add v to R
Endwhile
```



it's safe to add v

- Theorem. Upon termination, R is the connected component containing s.
  - BFS = explore in order of distance from s.
  - DFS = explore in a different way.

Section 3.4:
Testing Bipartiteness

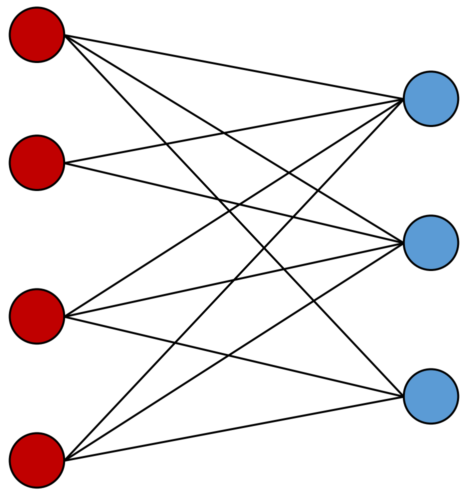
Administrivia

- **Assignment 1 – Released**
- **Quiz 2 on Wednesday 27th**

Bipartite Graphs

- **Def:** An undirected graph G = (V, E) is bipartite if the nodes can be colored red or blue such that every edge has one red and one blue end.

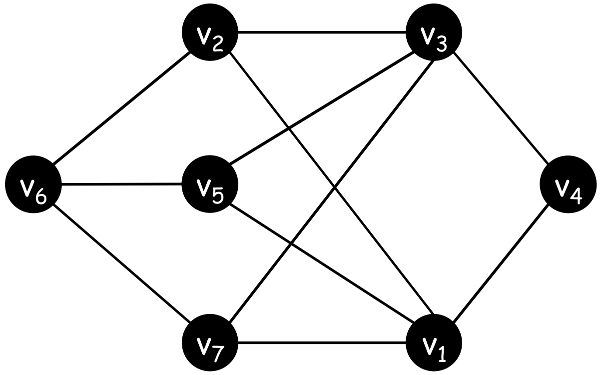
- Applications.
  - Stable matching: courses = blues, TAs = red.
  - Scheduling: jobs = blue, machines = red



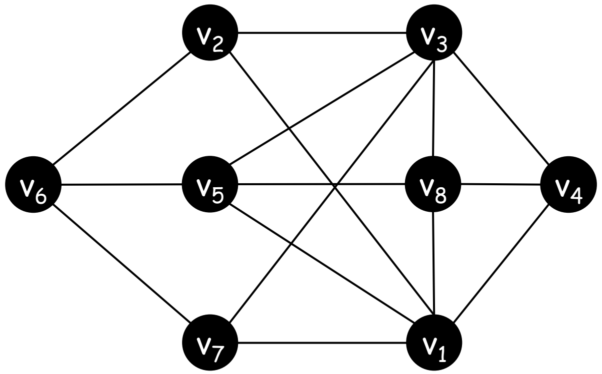
a bipartite graph

Testing Bipartiteness

Given a graph **G1** and **G2**, which one is bipartite?

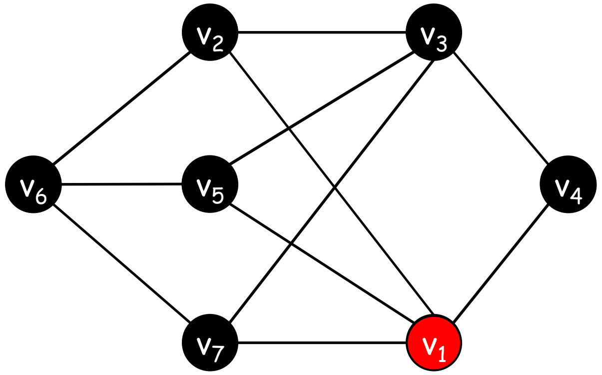


G1

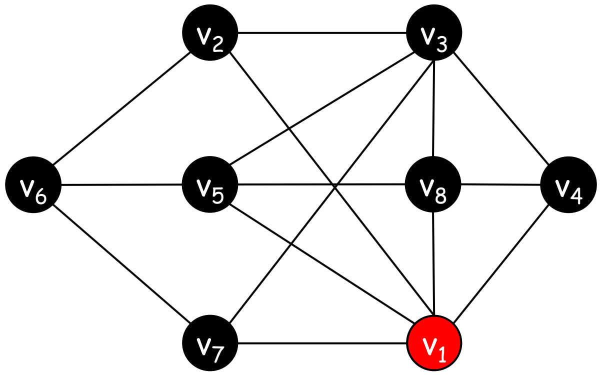


G2

Testing Bipartiteness

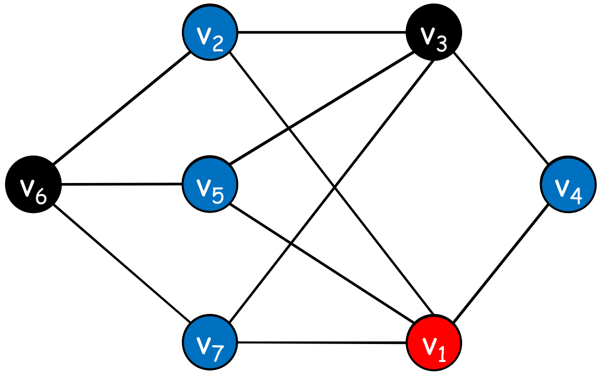


G1

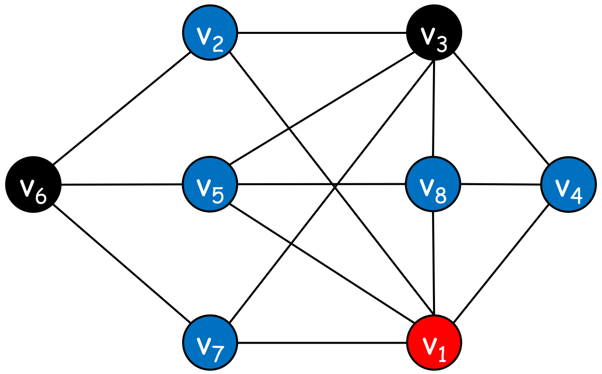


G2

Testing Bipartiteness

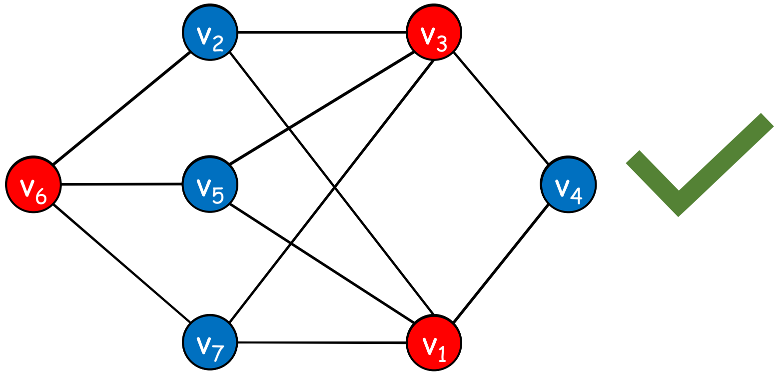


G1

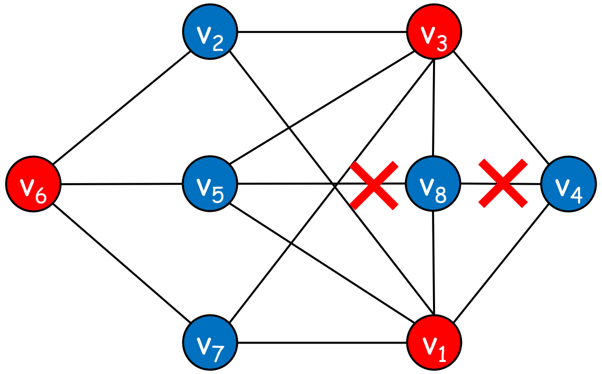


G2

Testing Bipartiteness

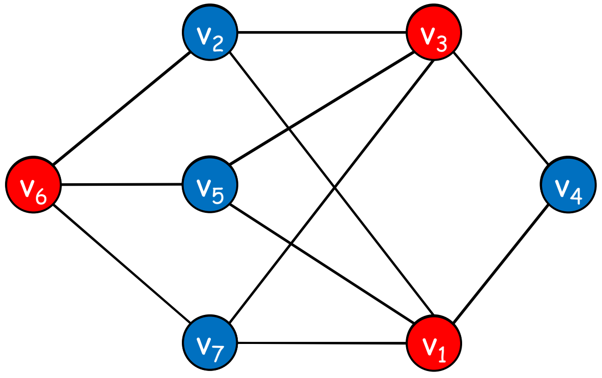


G1

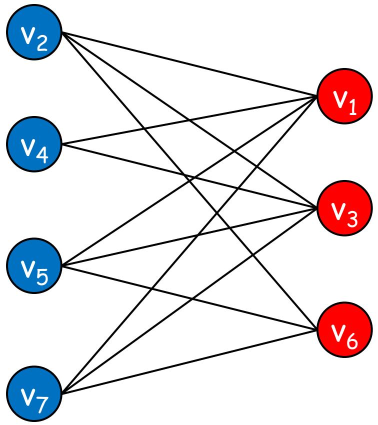


G2

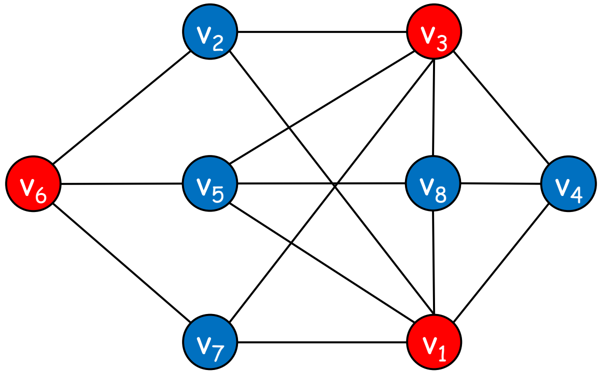
Testing Bipartiteness



G1



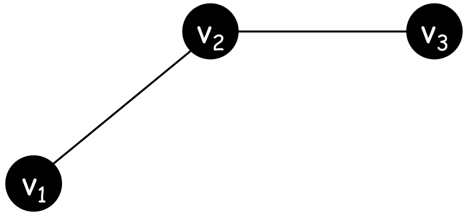
another drawing of G1



G2

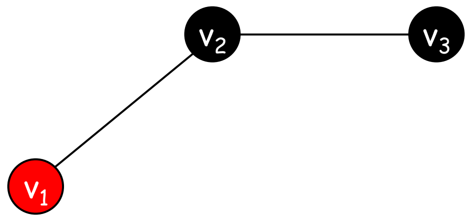
Testing Bipartiteness

Before attempting to design an algorithm, we need to understand structure of bipartite graphs



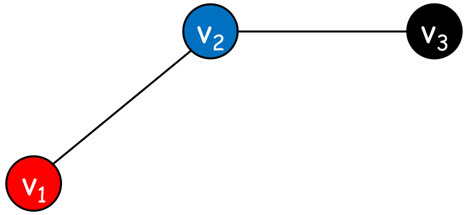
Testing Bipartiteness

Before attempting to design an algorithm, we need to understand structure of bipartite graphs



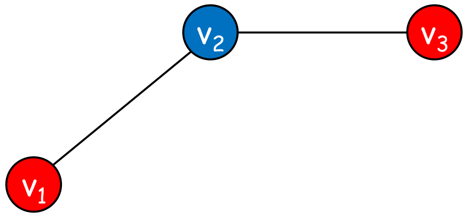
Testing Bipartiteness

Before attempting to design an algorithm, we need to understand structure of bipartite graphs



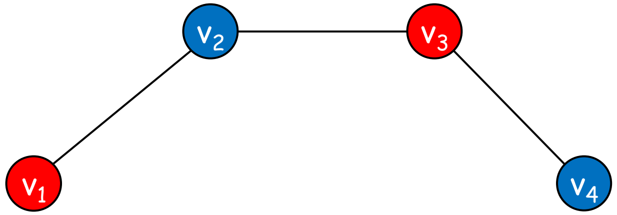
Testing Bipartiteness

Before attempting to design an algorithm, we need to understand structure of bipartite graphs



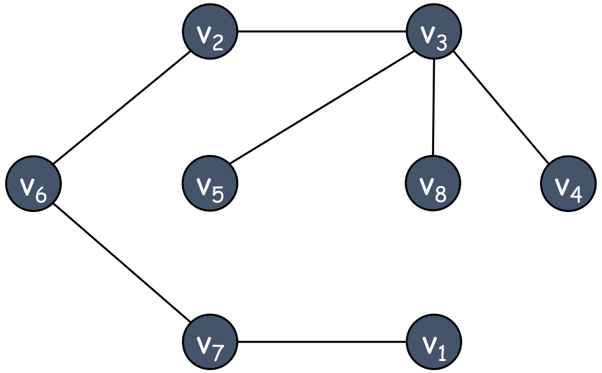
Testing Bipartiteness

Before attempting to design an algorithm, we need to understand structure of bipartite graphs



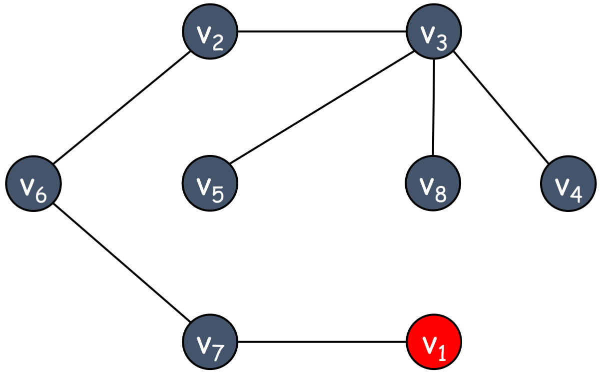
Testing Bipartiteness

Before attempting to design an algorithm, we need to understand structure of bipartite graphs



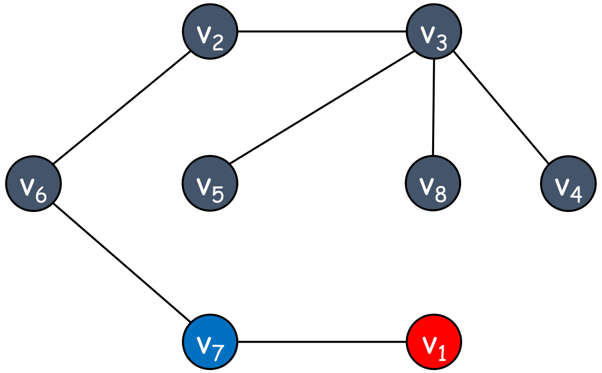
Testing Bipartiteness

Before attempting to design an algorithm, we need to understand structure of bipartite graphs



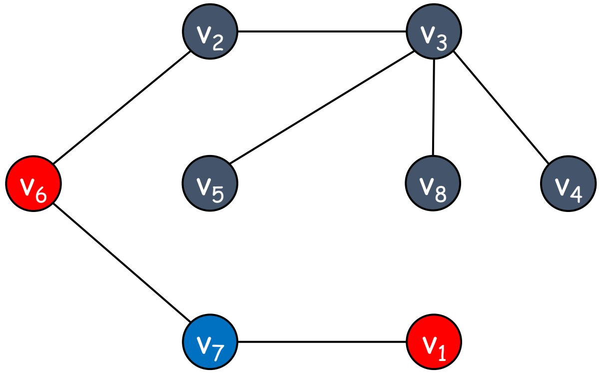
Testing Bipartiteness

Before attempting to design an algorithm, we need to understand structure of bipartite graphs



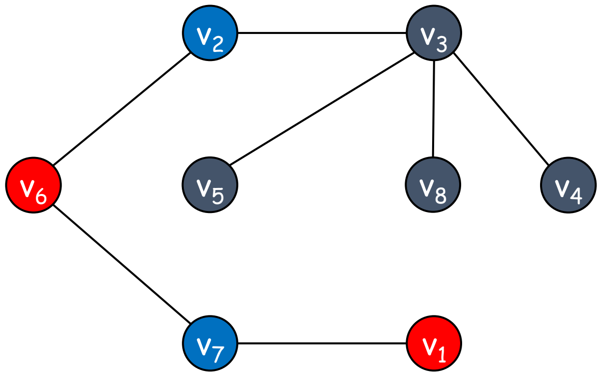
Testing Bipartiteness

Before attempting to design an algorithm, we need to understand structure of bipartite graphs



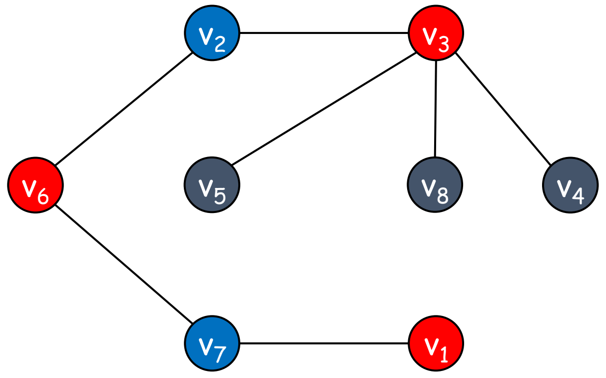
Testing Bipartiteness

Before attempting to design an algorithm, we need to understand structure of bipartite graphs



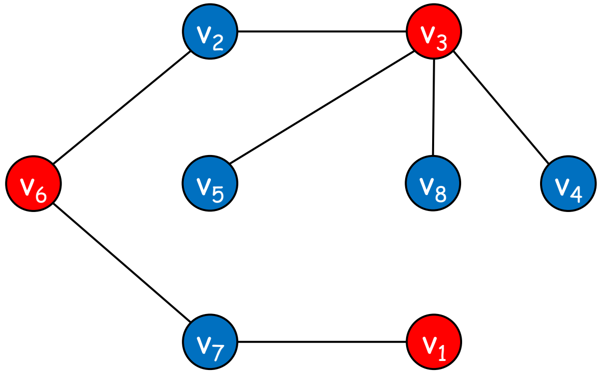
Testing Bipartiteness

Before attempting to design an algorithm, we need to understand structure of bipartite graphs



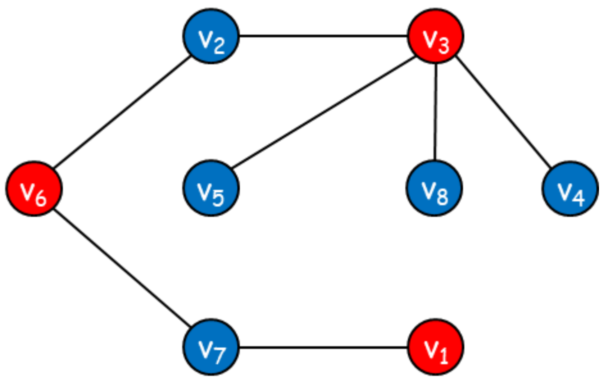
Testing Bipartiteness

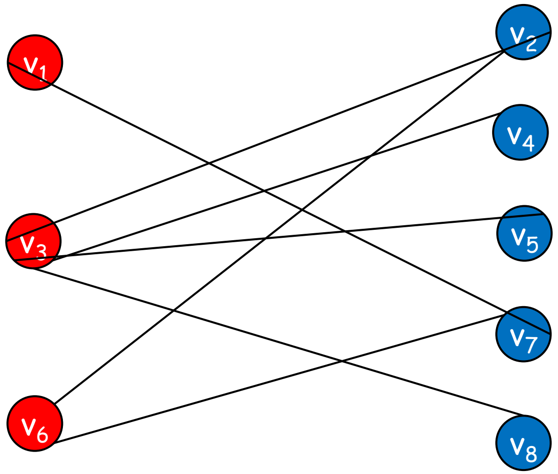
Before attempting to design an algorithm, we need to understand structure of bipartite graphs



Testing Bipartiteness

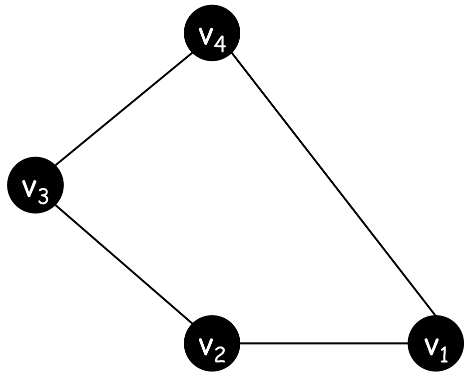
Observation: *A* graph ***G*** without cycles is always **bipartite**.





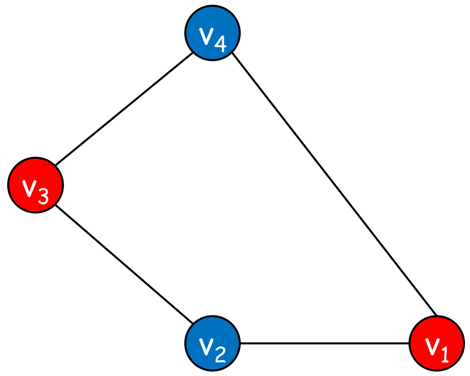
Testing Bipartiteness

What about Cycles? Let explore **even cycles**



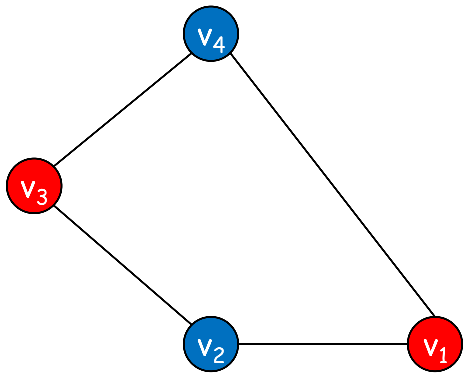
Testing Bipartiteness

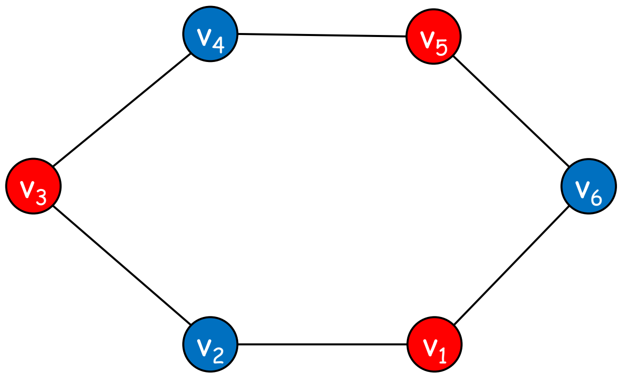
What about Cycles? Let explore **even cycles**



Testing Bipartiteness

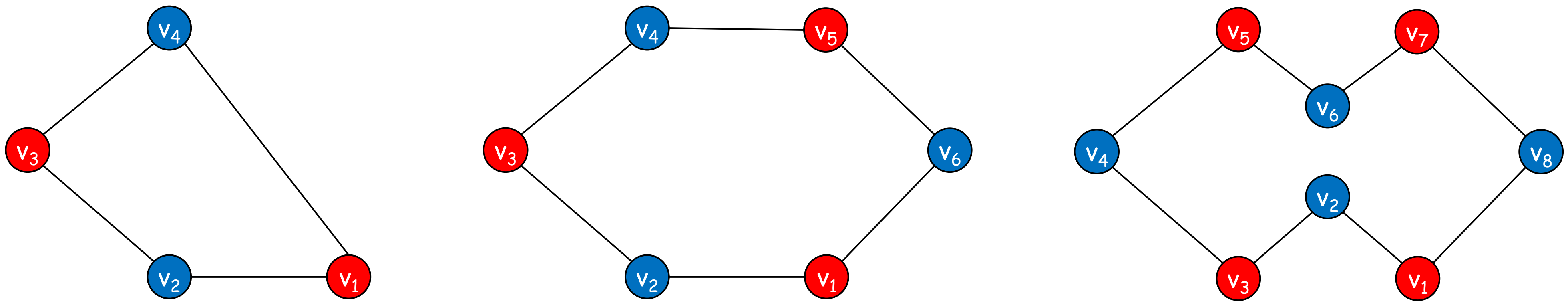
What about Cycles? Let explore **even cycles**





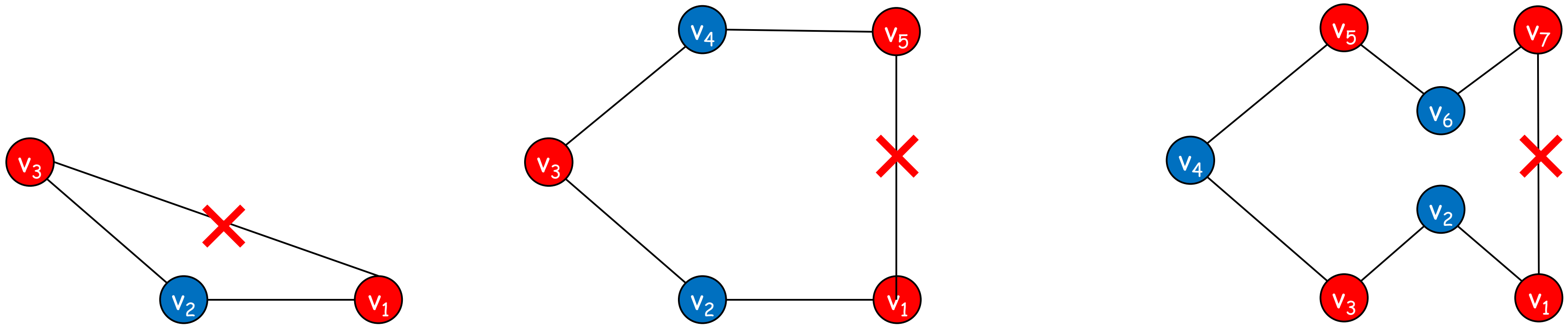
Testing Bipartiteness

What about Cycles? Let explore **even cycles**



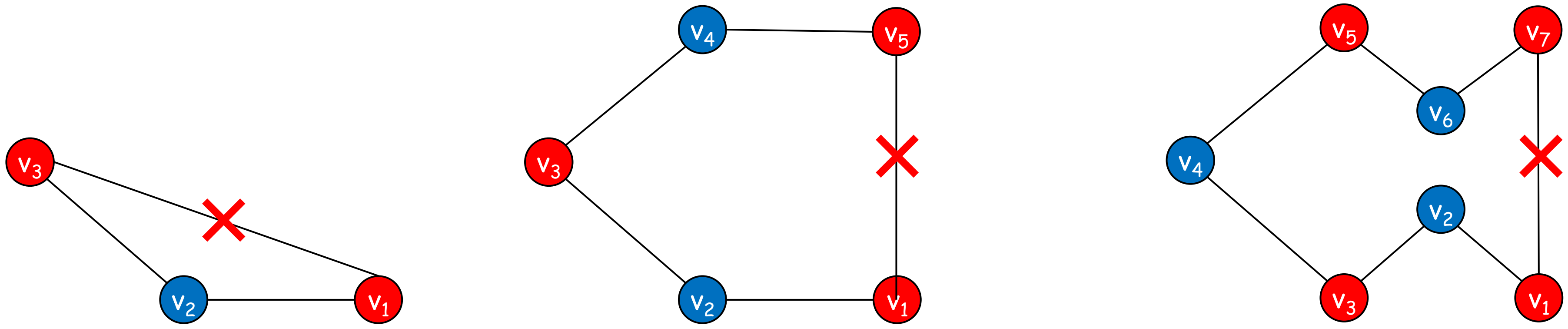
Testing Bipartiteness

What about **odd cycles?**

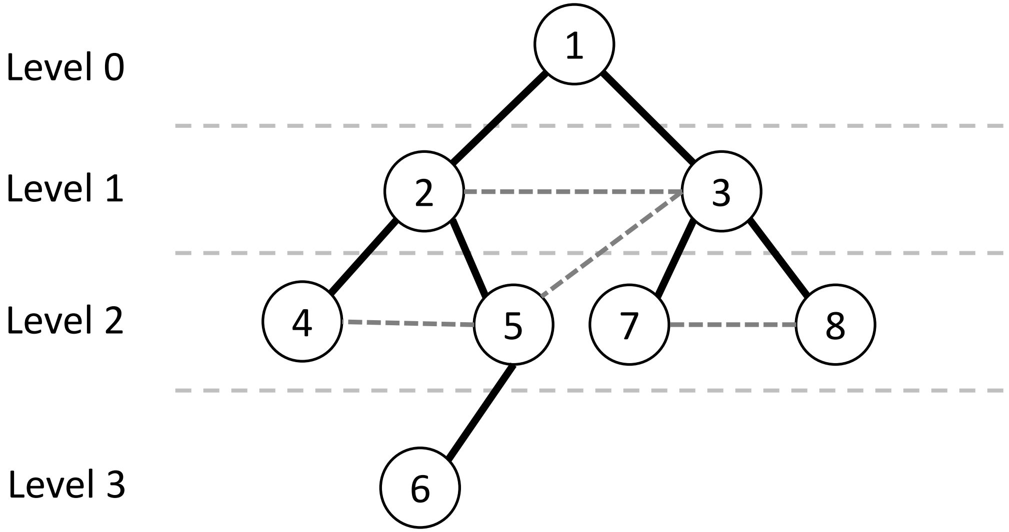


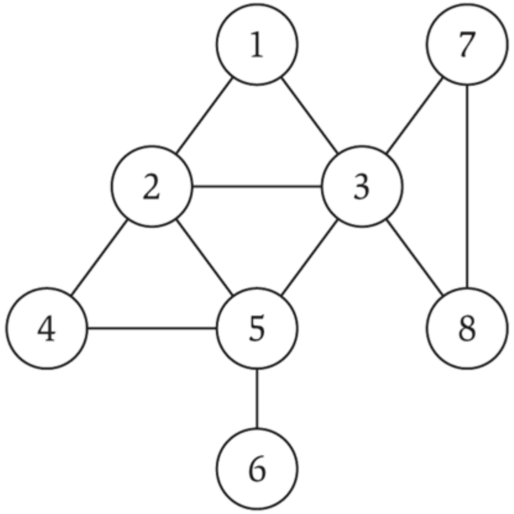
Testing Bipartiteness

Lemma 1: If a graph **G** is bipartite, it cannot contain an odd length cycle.

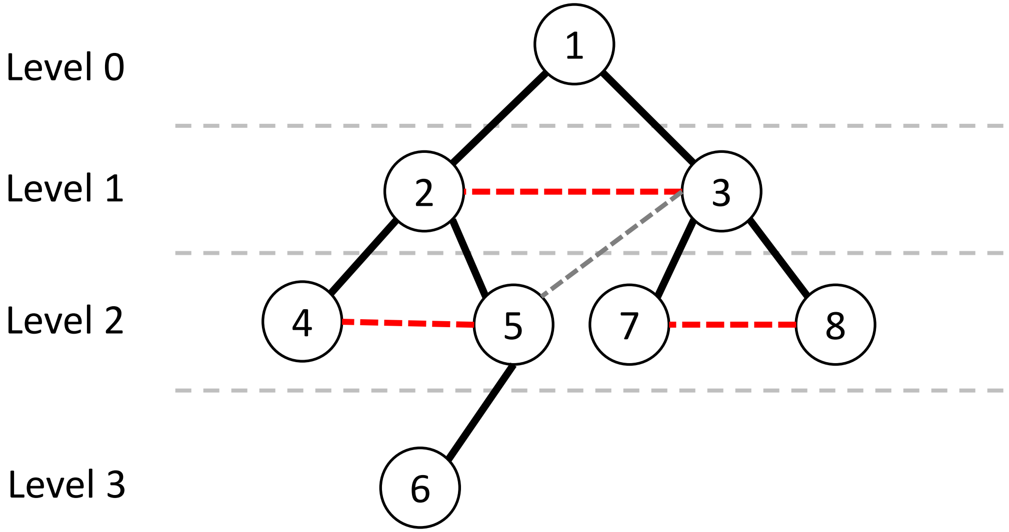


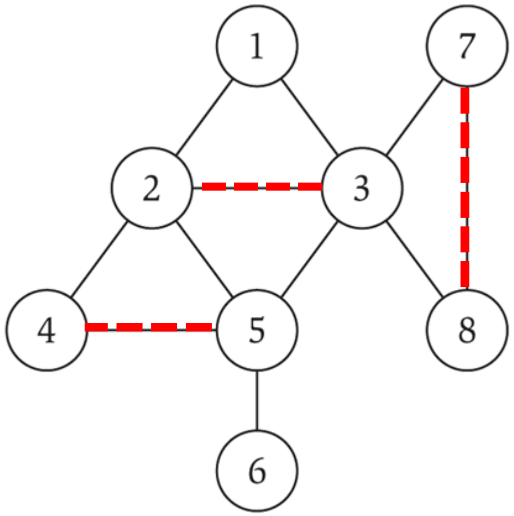
Bipartiteness and BFS



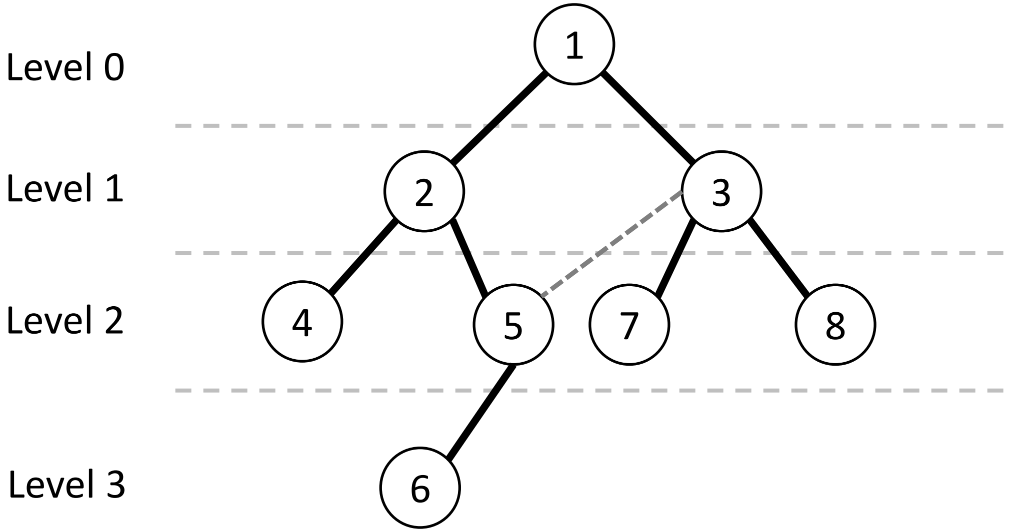


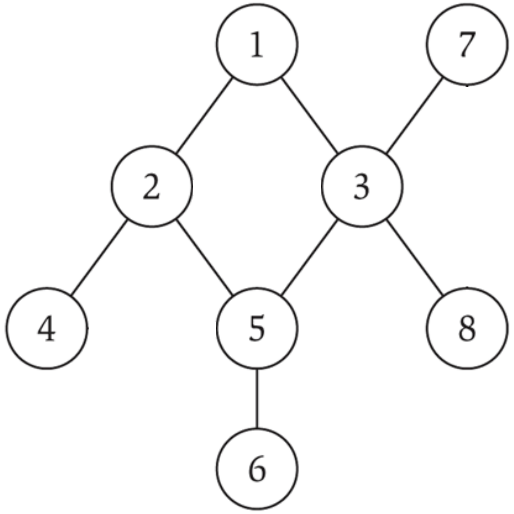
Bipartiteness and BFS





Bipartiteness and BFS



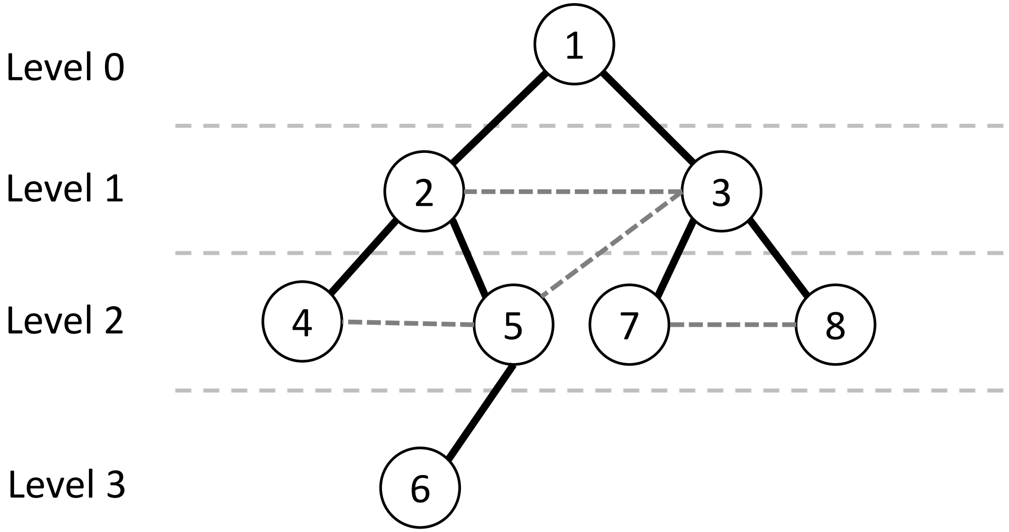


Bipartite Graphs and BFS

Hence…

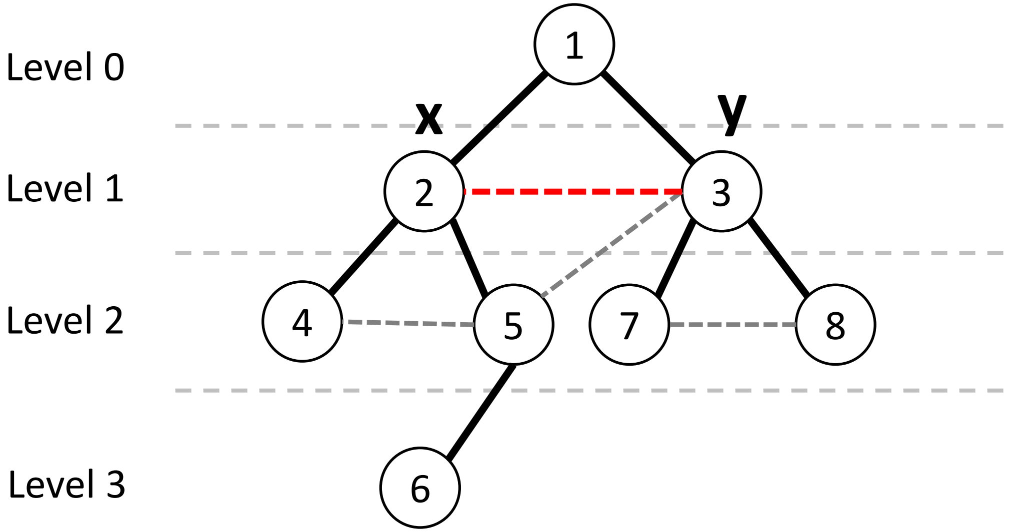
- **Lemma 2**. Let $\boldsymbol{G}$ be a connected graph, and let $\mathbf{L_0}$, …, $\mathbf{L_k}$ be the layers produced by **BFS** starting at node $\boldsymbol{s}$. Exactly one of the following holds.
  (i) No edge of $\boldsymbol{G}$ joins two nodes of the same layer, and $\boldsymbol{G}$ is bipartite.
  (ii) An edge of $\boldsymbol{G}$ joins two nodes of the same layer, and $\boldsymbol{G}$ contains an odd-length cycle (and hence is not bipartite).

Bipartiteness and BFS



Proof:

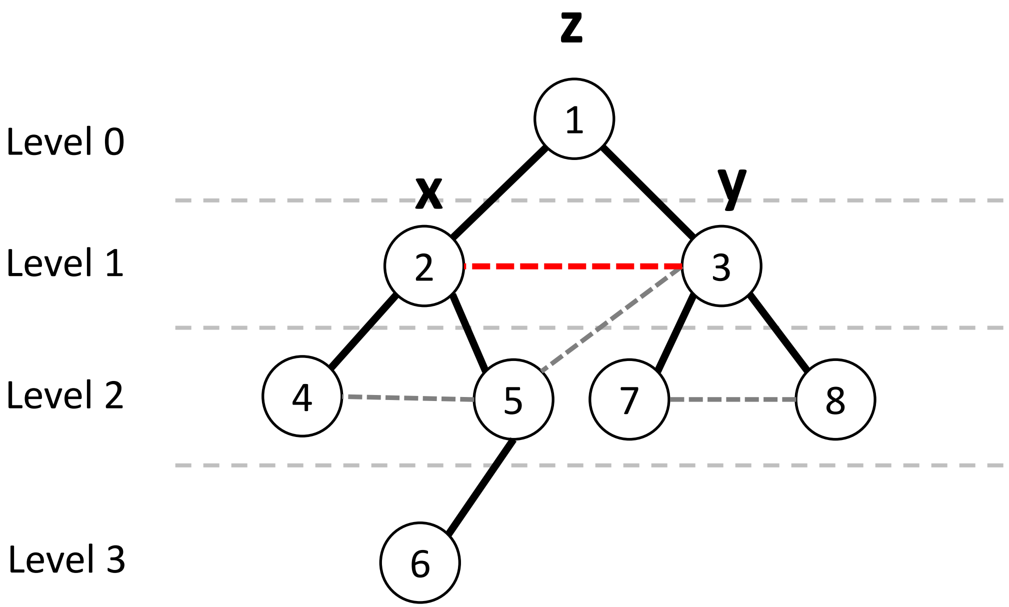
Bipartiteness and BFS



Proof:

- Suppose **(x, y)** is an edge with x, y in same level $\mathbf{L_j}$

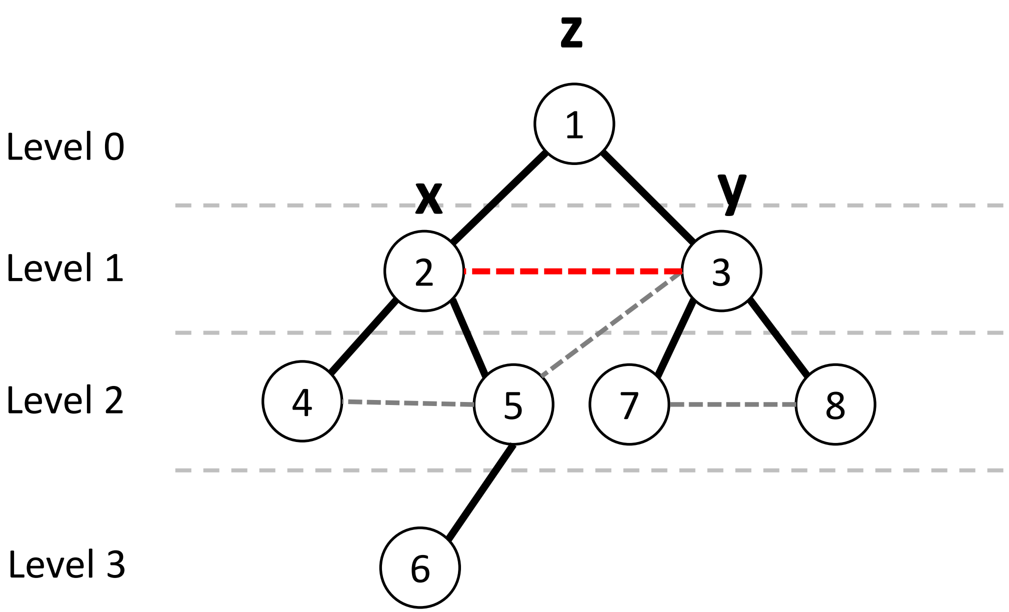
Bipartiteness and BFS



Proof:

- Suppose **(x, y)** is an edge with x, y in same level $\mathbf{L_j}$.
- Let **z** = lca(x, y) = lowest common ancestor.

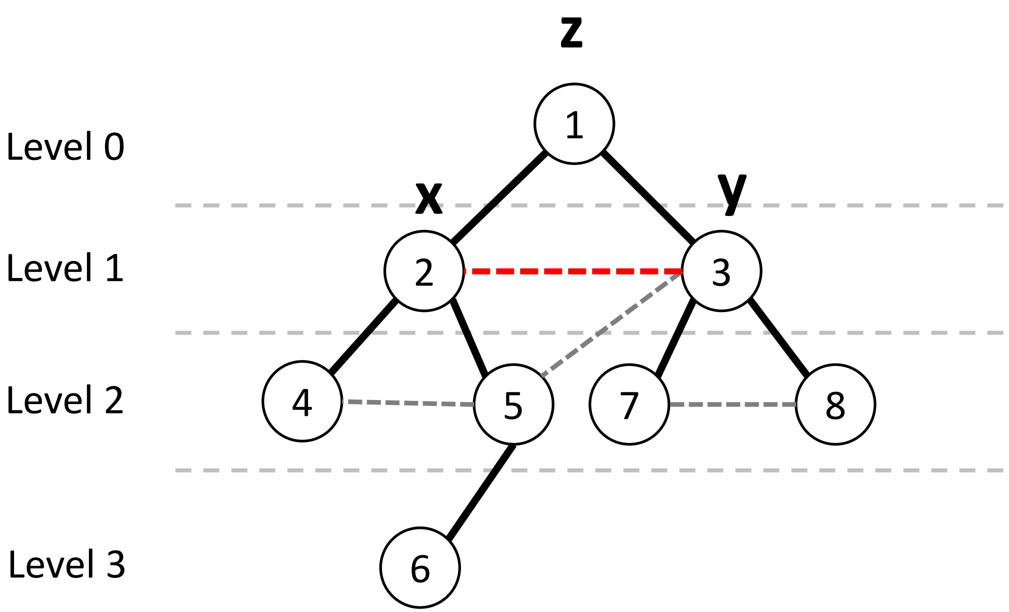
Bipartiteness and BFS



Proof:

- Suppose **(x, y)** is an edge with x, y in same level $\mathbf{L_j}$.
- Let **z** = lca(x, y) = lowest common ancestor.
- Let $\mathbf{L_i}$ be level containing **z**

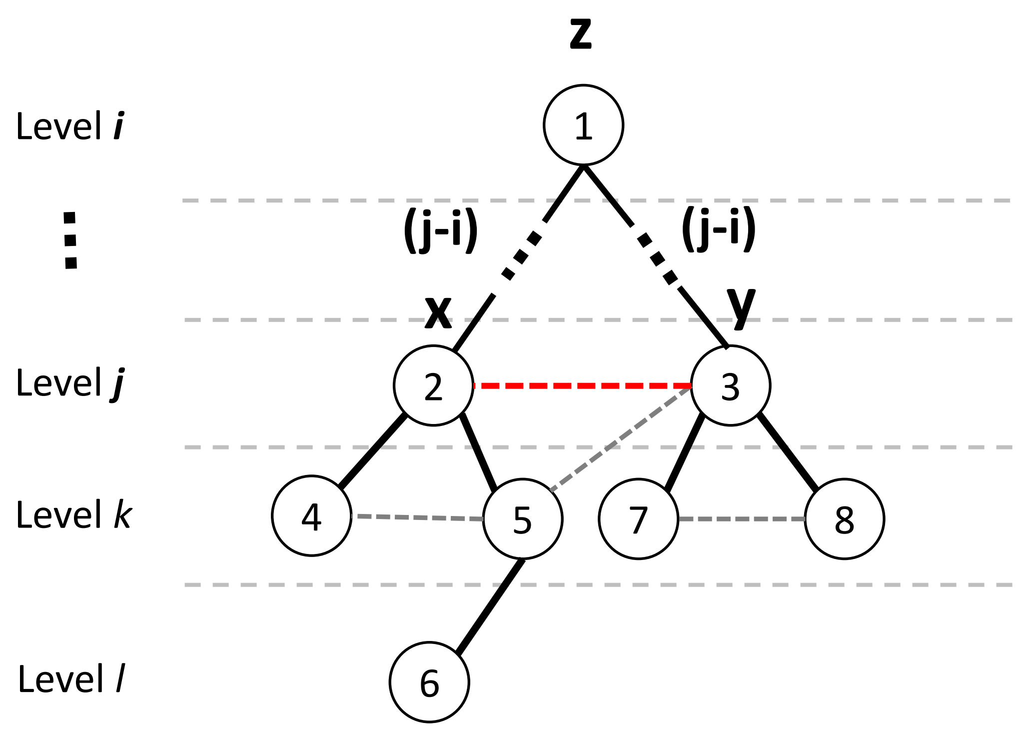
Bipartiteness and BFS



Proof:

- Suppose **(x, y)** is an edge with x, y in same level $\mathbf{L_j}$.
- Let **z** = lca(x, y) = lowest common ancestor.
- Let $\mathbf{L_i}$ be level containing **z**
- Consider cycle that takes edge from **x to y**, then path from **y to z**, then path from **z to x**.

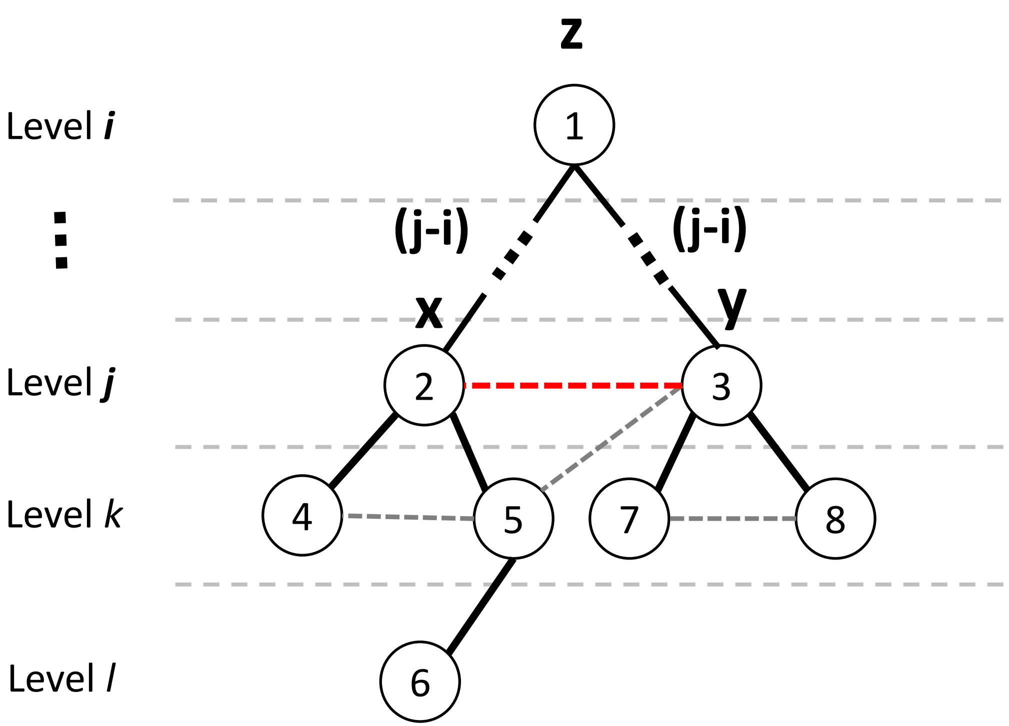
Bipartiteness and BFS



Proof:

- Suppose **(x, y)** is an edge with x, y in same level $\mathbf{L_j}$.
- Let **z** = lca(x, y) = lowest common ancestor.
- Let $\mathbf{L_i}$ be level containing **z**
- Consider cycle that takes edge from **x to y**, then path from **y to z**, then path from **z to x**.

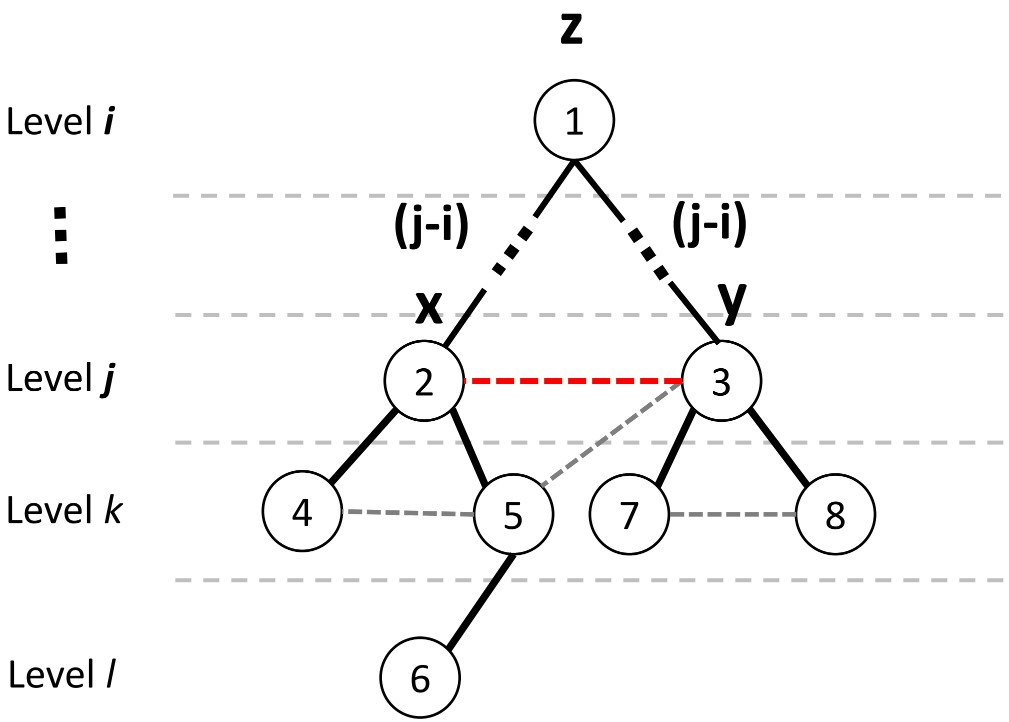
Bipartiteness and BFS



Proof:

- Suppose **(x, y)** is an edge with x, y in same level L_j.
- Let **z** = lca(x, y) = lowest common ancestor.
- Let L_i be level containing **z**
- Consider cycle that takes edge from **x to y**, then path from **y to z**, then path from **z to x**.
- Its length is **1** + **(j-i)** + **(j-i)**.

Bipartiteness and BFS



Proof:

- Suppose **(x, y)** is an edge with x, y in same level $\mathbf{L_j}$.
- Let **z** = lca(x, y) = lowest common ancestor.
- Let $\mathbf{L_i}$ be level containing **z**
- Consider cycle that takes edge from **x to y**, then path from **y to z**, then path from **z to x**.
- Its length is **1** + **(j-i)** + **(j-i)**.

$$\mathbf{1} + \underbrace{\mathbf{2\,(j\text{-}i)}}_{\text{Any number multiplied by 2 is even}}$$ Which always gives **Odd** length

Bipartite Graphs

Corollary (Based on *Lemma 1* and *2*)**:** A graph **G** is bipartite ***iff*** it contain no odd length cycle.

Testing Bipartite Graphs – Designing the Algorithm

```
procedure BFS(G,s)

    for each vertex v ∈ V[G] do
        explored[v] ← false
        d[v] ← ∞
    end for
    explored[s] ← true
    d[s] ← 0
    Q:= a queue data structure, initialized with s
    while Q ≠ ϕ do
        u ← remove vertex from the front of Q
        for each v adjacent to u do
            if not explored[v] then
                explored[v] ← true
                d[v] ← d[u] + 1
                insert v to the end of Q
            end if
        end for
    end while

end procedure
```

Replace it with **color[v] = Black**

Replace it with **color[s] = Red**

color it with the opposite color of ‘u’

Add new ***if*** block. If **v** is already explored and has the same color as **u**, then the graph is not bipartite. Exit.

Testing Bipartite Graphs – Designing the Algorithm

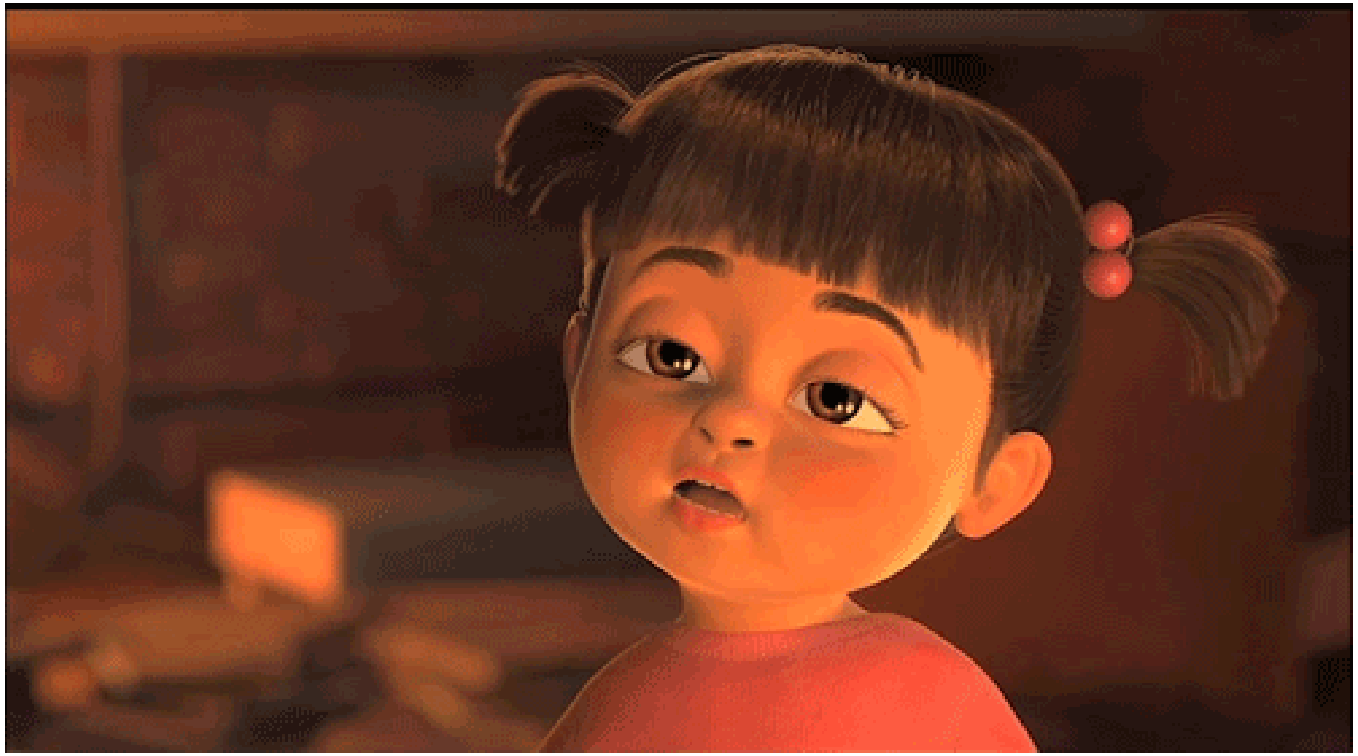
```
procedure BFS(G,s)

    for each vertex v ∈ V[G] do
        explored[v] ← false
        d[v] ← ∞
    end for
    explored[s] ← true
    d[s] ← 0
    Q:= a queue data structure, initialized with s
    while Q ≠ ϕ do
        u ← remove vertex from the front of Q
        for each v adjacent to u do
            if not explored[v] then
                explored[v] ← true
                d[v] ← d[u] + 1
                insert v to the end of Q
            end if
        end for
    end while

end procedure
```

Same Time Complexity

Thanks a lot



If you are taking a Nap, **wake up**.......Lecture Over