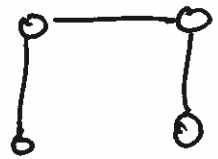
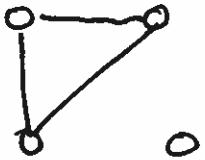
Graphs

- an abstract way of representing connectivity using nodes and edges
- edges connect some pair of nodes
  - can be either directed or undirected
- nodes and edges can carry additional information

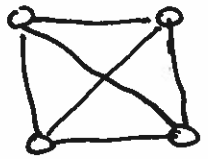
connected



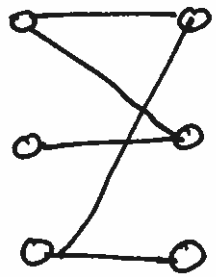
not connected



complete



Bipartite graph



Sparse graph

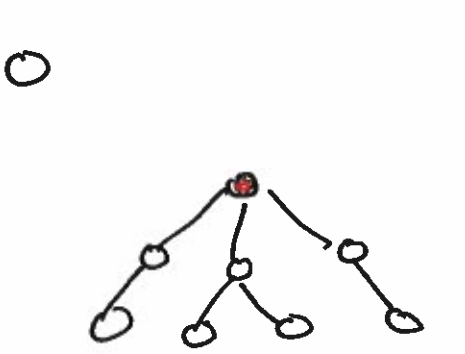
$|E| \in O(|V|)$

Dense graph

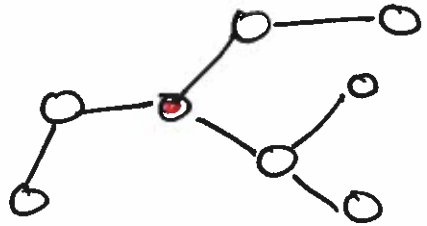
$|E| \in O(|V|^2)$

Tree

- any connected undirected acyclic graph
- any connected graph where $|E| = |V| - 1$



rooted tree



Uses

- Network flow
- pathfinding
- social network
- Maps and navigation (TSP)
- parsing

Graph Representations

- Adjacency List

  each node has a list of outgoing edges

  space: $\Theta(|V| + |E|)$

- Vertex set and Edge set

  space: $\Theta(|V| + |E|)$

- Adjacency Matrix

  $|V| \times |V|$ matrix A where

  $a_{ij} = 1$ if there is an edge between nodes i and j

  $a_{ij} = 0$ otherwise

  space: $\Theta(|V|^2)$

DFS(G, v) Depth-First Search

```
1  Mark node v as visited
2  for each edge (v, u)
3        if u is not marked as visited
4              DFS(G, u)
```

- assume G is connected

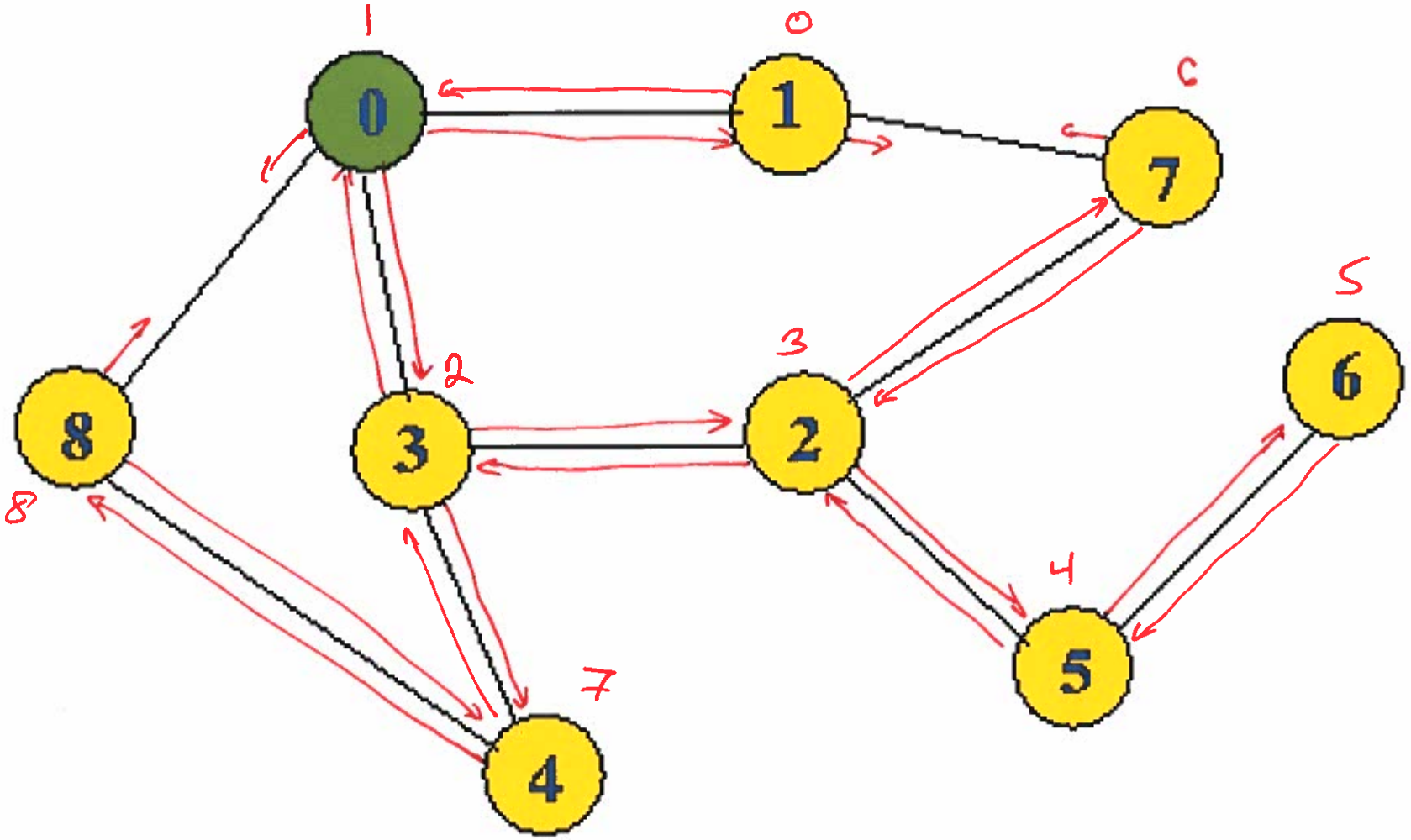
Complexity $\Theta(|V| + |E|)$ $\Theta(n+m)$

Input size: $n = |V|$ $m = |E|$

How many times can a node v be the input to DFS? Once

How many time is an edge (u,v) examined?

exactly twice


0
1
7
8
3
2
6
5
4

BFS(G, v) Breadth-first Search

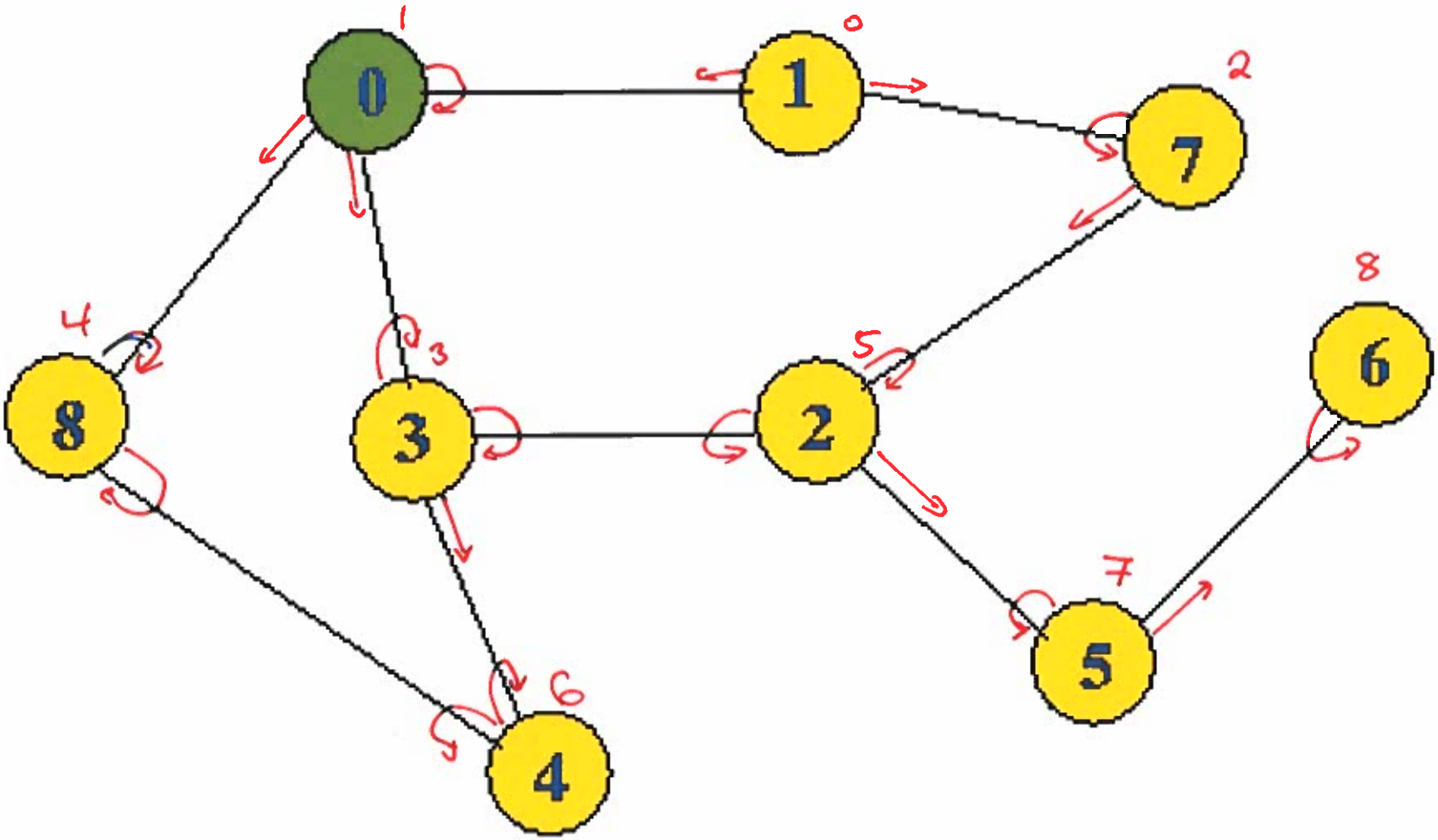
```
1  Q = empty queue
2  Mark node v as visited
3  Q.enqueue(v)
4  while Q is not empty
5        v = Q.dequeue()
6        for each edge (v, u)
7              if u is not marked as visited
8                    Mark node u as visited
9                    Q.enqueue(u)
```

Input size: $n = |V|$ $m = |E|$

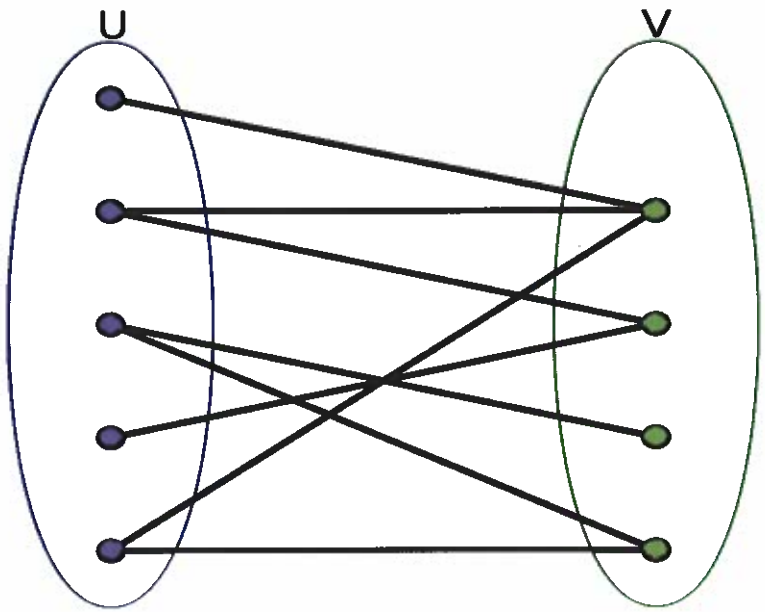
Complexity: $\Theta(n + m)$

- Each node can only be added to Q once
- Each edge will be examined twice

Q: ~~1 0 7 3 8 2 4 5 6~~



A **Bipartite Graph** is a graph whose vertices can be divided into two disjoint sets, **U** and **V**, such that every edge connects a vertex in **U** to one in **V**.



Design an Algorithm that uses either DFS or BFS to test whether a given connected undirected graph is bipartite.

PotD: Bipartite Graph Tester

Input: A connected undirected graph G

Output: True if G is bipartite and False otherwise

```
func isBipartite(V, Color)
    V.color with Color
    for neighbor U of V
        if U is unlabeled
            return isBipartite(V, otherColor)
        else if U.color = Color
            return False
    return True
```