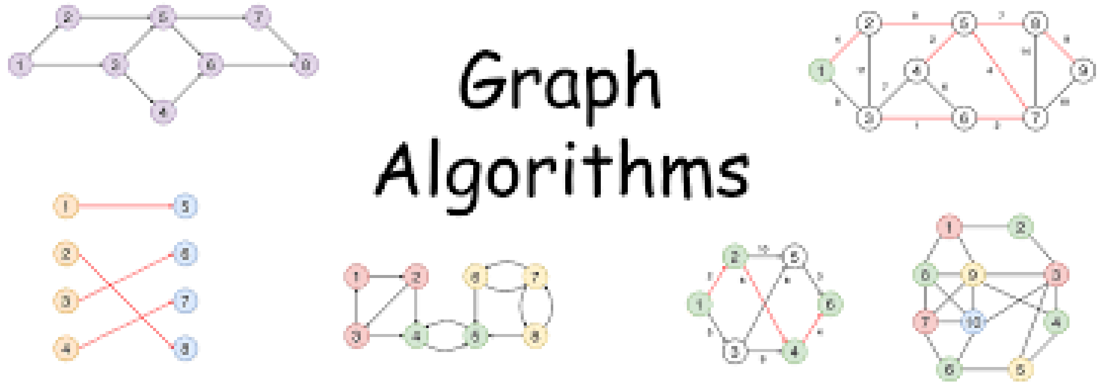
Breadth-first search

COMP215: Design & Analysis of Algorithms

Today

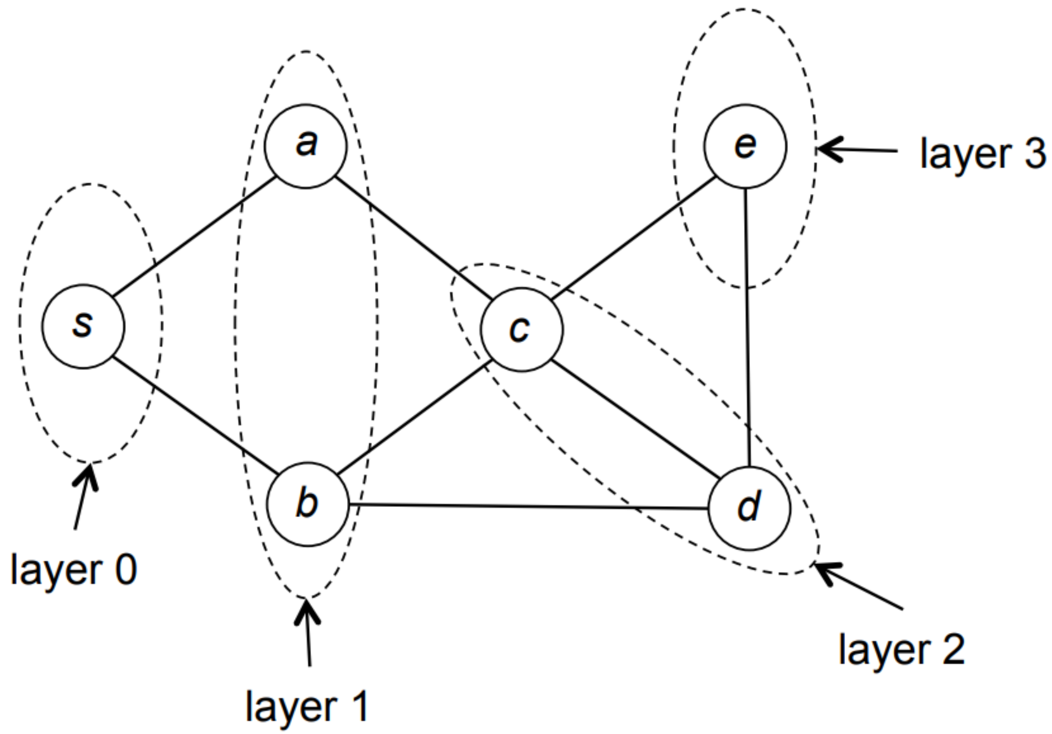


- Breadth-first search
- Shortest Paths
- Computing Connected

Breadth-first search

- Breadth-first search explores the vertices of a graph in **layers**, in order of increasing distance from the starting vertex.
- Layer 0 contains the starting vertex s and nothing else.
- Layer 1 is the set of vertices that are one hop away from s—that is, s's neighbors
- In general, the vertices in a layer i are those that neighbor a vertex in layer i - 1 and that do not already belong to one of the layers 0, 1, 2,...,i – 1.

Breadth-first search



Breadth-first search

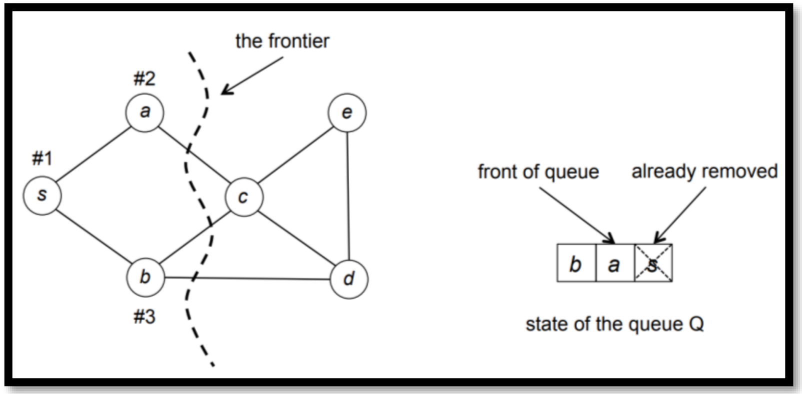
BFS

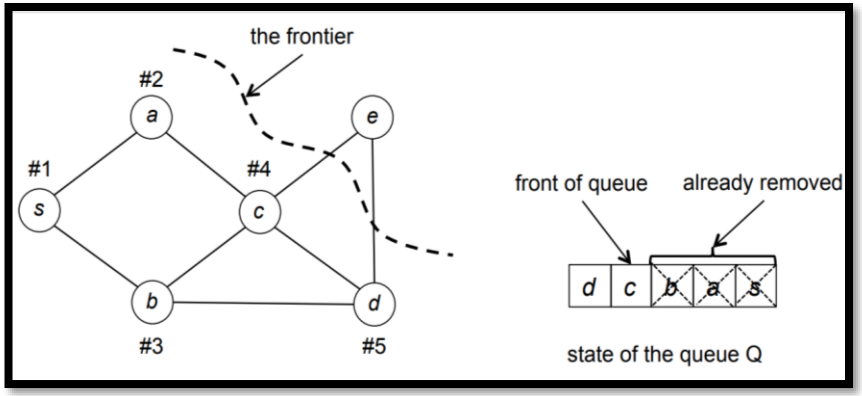
Input: graph $G = (V, E)$ in adjacency-list representation, and a vertex $s \in V$.
Postcondition: a vertex is reachable from s if and only if it is marked as "explored."

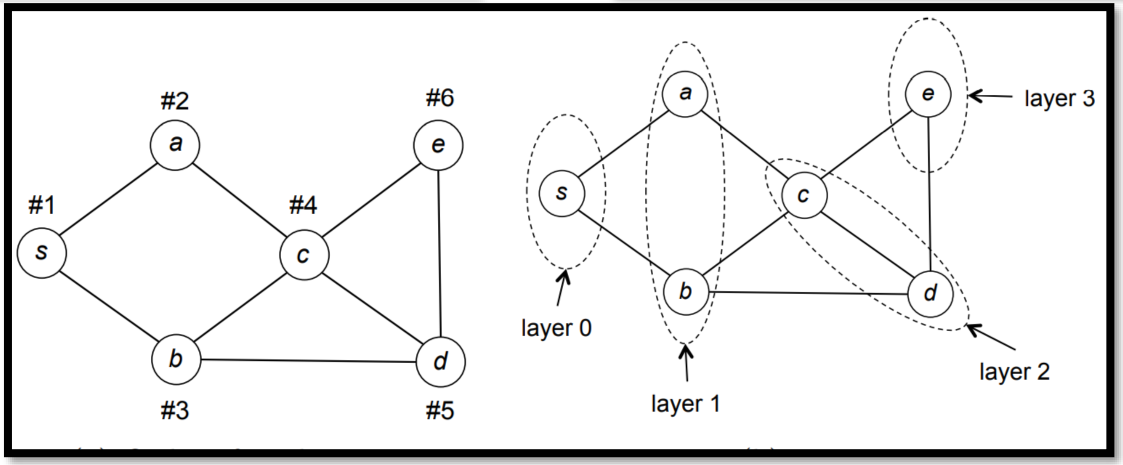
---

mark s as explored, all other vertices as unexplored
$Q :=$ a queue data structure, initialized with s
while Q is not empty **do**
    remove the vertex from the front of Q, call it v
    for each edge (v, w) in v's adjacency list **do**
        if w is unexplored **then**
            mark w as explored
            add w to the end of Q

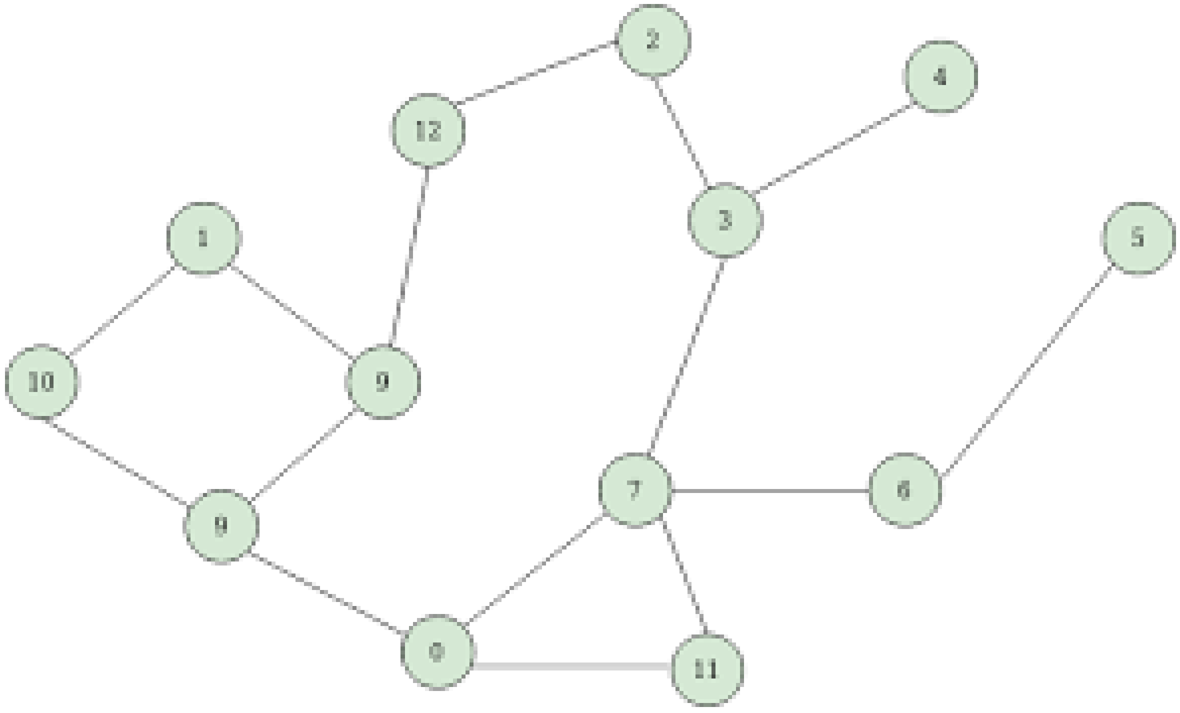
Breadth-first search







Example



Correctness and Running Time

Theorem 8.2 (Properties of BFS) *For every undirected or directed graph $G = (V, E)$ in adjacency-list representation and for every starting vertex $s \in V$:*

*(a) At the conclusion of **BFS**, a vertex $v \in V$ is marked as explored if and only if there is a path from s to v in G.*

*(b) The running time of **BFS** is $O(m + n)$, where $m = |E|$ and $n = |V|$.*

*(c) The running time of lines 2–8 of **BFS** is*

$$O(m_s + n_s),$$

where m_s and n_s denote the number of edges and vertices, respectively, reachable from s in G.

Shortest Paths

- What is unique about BFS is that, with just a couple extra lines of code, it efficiently computes **shortest-path distances**.

Problem Definition

In a graph G, we use the notation $dist(v, w)$ for the fewest number of edges in a path from v to w (or $+\infty$, if G contains no path from v to w).[15]

Problem: Shortest Paths (Unit Edge Lengths)

Input: An undirected or directed graph $G = (V, E)$, and a starting vertex $s \in V$.

Output: $dist(s, v)$ for every vertex $v \in V$.[16]

Shortest Paths

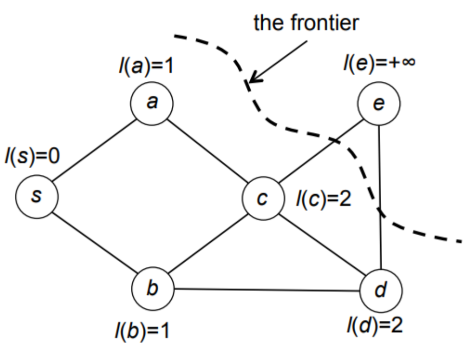
Augmented-BFS

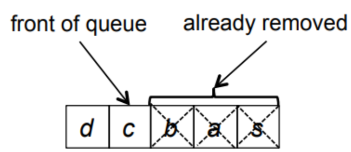
Input: graph $G = (V, E)$ in adjacency-list representation, and a vertex $s \in V$.

Postcondition: for every vertex $v \in V$, the value $l(v)$ equals the true shortest-path distance $dist(s, v)$.

---

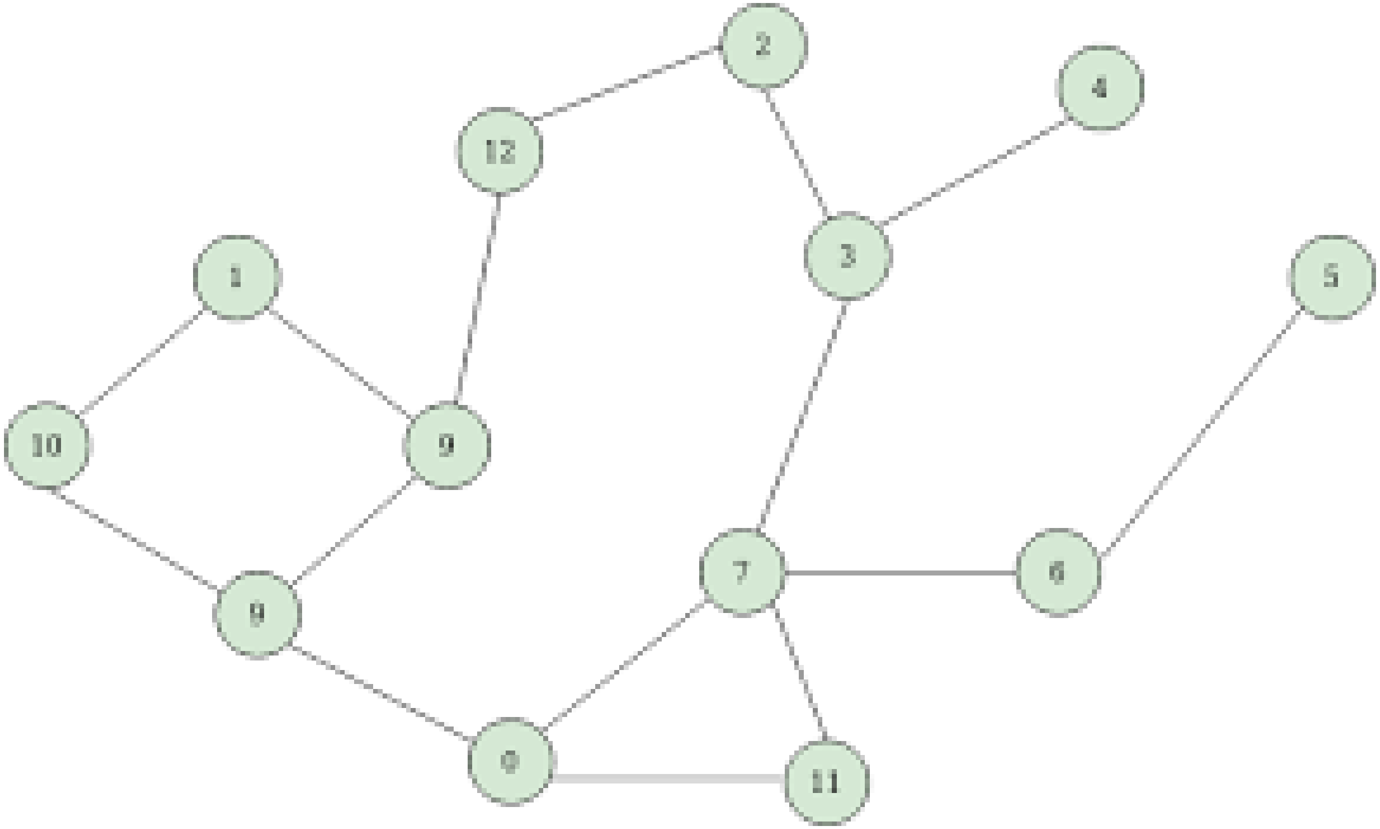
mark s as explored, all other vertices as unexplored
$l(s) := 0$, $l(v) := +\infty$ for every $v \neq s$
Q := a queue data structure, initialized with s
while Q is not empty **do**
    remove the vertex from the front of Q, call it v
    for each edge (v, w) in v's adjacency list **do**
        if w is unexplored **then**
            mark w as explored
            $l(w) := l(v) + 1$
            add w to the end of Q





state of the queue Q

Example



Shortest Paths

Theorem 8.3 (Properties of `Augmented-BFS`) *For every undirected or directed graph $G = (V, E)$ in adjacency-list representation and for every starting vertex $s \in V$:*

*(a) At the conclusion of **`Augmented-BFS`**, for every vertex $v \in V$, the value of $l(v)$ equals the length $dist(s, v)$ of a shortest path from s to v in G (or $+\infty$, if no such path exists).*

*(b) The running time of **`Augmented-BFS`** is $O(m+n)$, where $m = |E|$ and $n = |V|$.*

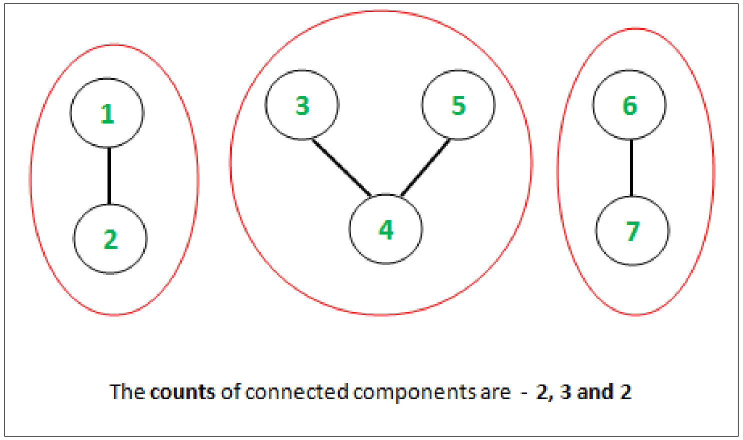
Computing Connected Components

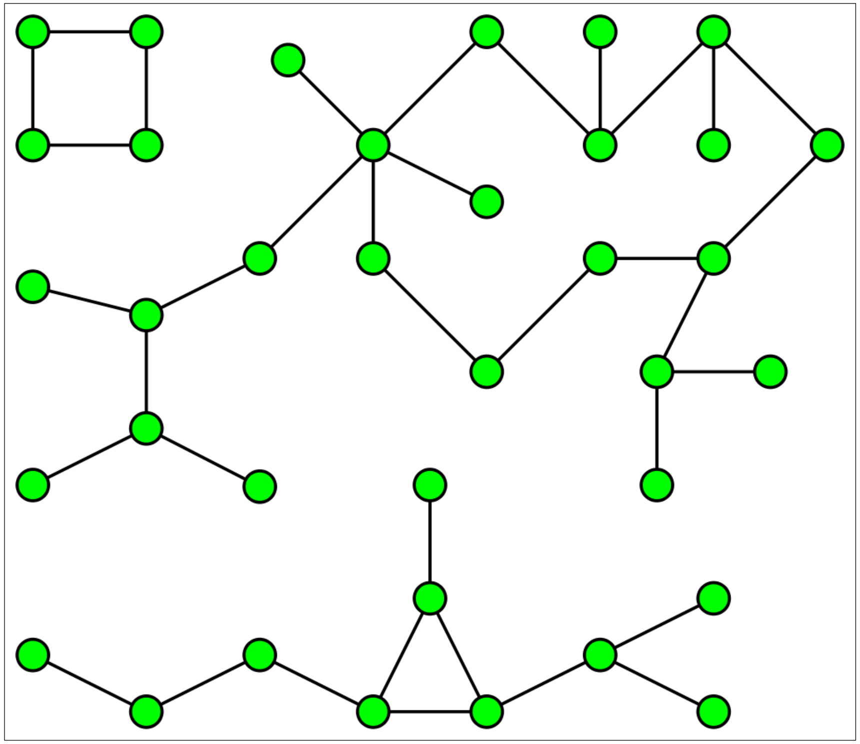
- An undirected graph G = (V,E) naturally falls into "pieces," which are **called connected components.**

- A connected component is a maximal subset **S $\subseteq$ V** of vertices such that there is a path from any vertex in S to any other vertex in S.

- We will use **breadth-first search** to compute the connected components of a graph in linear time

Computing Connected Components





Computing Connected Components

Problem: Undirected Connected Components

Input: An undirected graph $G = (V, E)$.

Goal: Identify the connected components of G.

Quiz 8.2

Consider an undirected graph with n vertices and m edges. What are the minimum and maximum number of connected components that the graph could have, respectively?

a) 1 and $n - 1$

b) 1 and n

c) 1 and $\max\{m, n\}$

d) 2 and $\max\{m, n\}$

Computing Connected Components

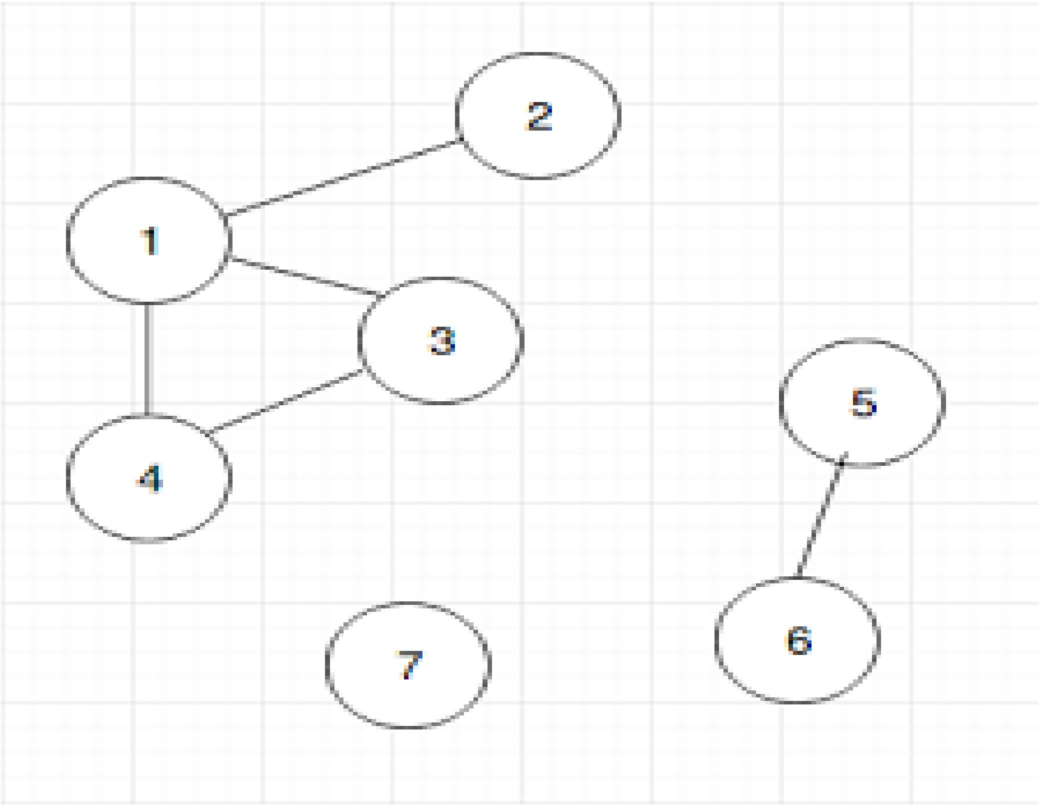
UCC

Input: undirected graph $G = (V, E)$ in adjacency-list representation, with $V = \{1, 2, 3, \ldots, n\}$.
Postcondition: for every $u, v \in V$, $cc(u) = cc(v)$ if and only if u, v are in the same connected component.

---

mark all vertices as unexplored
$numCC := 0$
for $i := 1$ to n **do** `// try all vertices`
  if i is unexplored **then** `// avoid redundancy`
    $numCC := numCC + 1$ `// new component`
    `// call BFS starting at` i `(lines 2-8)`
    $Q :=$ a queue data structure, initialized with i
    while Q is not empty **do**
      remove the vertex from the front of Q, call it v
      $cc(v) := numCC$
      for each (v, w) in v's adjacency list **do**
        if w is unexplored **then**
          mark w as explored
          add w to the end of Q

Computing Connected Components



Computing Connected Components

8.3.5 Correctness and Running Time

The `UCC` algorithm correctly computes the connected components of an undirected graph, and does so in linear time.

Theorem 8.4 (Properties of `UCC`) *For every undirected graph $G = (V, E)$ in adjacency-list representation:*

(a) At the conclusion of `UCC`, for every pair u, v of vertices, $cc(u) = cc(v)$ if and only if u and v belong to the same connected component of G.

(b) The running time of `UCC` is $O(m + n)$, where $m = |E|$ and $n = |V|$.