Breadth-First Search (BFS)

- For searching a graph and the archetype for many important graph algorithms

\[ G = (V, E) \]

\[ d[v] = \text{distance from } s \text{ to } v \text{ for all } v \in V. \]

- distance = fewest number of edges = shortest path

- BFS basic idea:
  - discovers all vertices at distance \( k \) from the source vertex before discovering any vertices at distance \( k + 1 \)
  - expanding frontier – "greedy" – propagate a wave 1 edge-distance at a time.
Breadth-First Search (BFS)

- For searching a graph and the archetype for many important graph algorithms
- **Input:** $G = (V, E)$ and a source vertex $s$,

\[
d[v] = \text{distance from } s \text{ to } v \text{ for all } v \in V.
\]

- distance = fewest number of edges = shortest path

- BFS basic idea:
  - discovers all vertices at distance $k$ from the source vertex before discovering any vertices at distance $k+1$
  - expanding frontier – “greedy” – propagate a wave 1 edge-distance at a time.
Breadth-First Search (BFS)

- For searching a graph and the archetype for many important graph algorithms

- **Input:** $G = (V, E)$ and a source vertex $s$,

- **Output:** $d[v] =$ distance from $s$ to $v$ for all $v \in V$. 

  distance = fewest number of edges = shortest path

- BFS basic idea:
  - discovers all vertices at distance $k$ from the source vertex before discovering any vertices at distance $k + 1$.
  - expanding frontier – “greedy” – propagate a wave 1 edge-distance at a time.
Breadth-First Search (BFS)

- For searching a graph and the archetype for many important graph algorithms

- **Input:** $G = (V, E)$ and a source vertex $s$,
  
  **Output:** $d[v] = \text{distance}$ from $s$ to $v$ for all $v \in V$.

- **distance** = fewest number of edges = shortest path
Breadth-First Search (BFS)

- For searching a graph and the archetype for many important graph algorithms

- **Input:** \( G = (V, E) \) and a source vertex \( s \),

  **Output:** \( d[v] = \text{distance} \) from \( s \) to \( v \) for all \( v \in V \).

- **distance** = fewest number of edges = shortest path

- **BFS basic idea:**
  - discovers all vertices at distance \( k \) from the source vertex before discovering any vertices at distance \( k + 1 \)
Breadth-First Search (BFS)

- For searching a graph and the archetype for many important graph algorithms

- **Input:** $G = (V, E)$ and a source vertex $s$, 
  
  **Output:** $d[v] = \text{distance}$ from $s$ to $v$ for all $v \in V$.

- **distance** = fewest number of edges = shortest path

- **BFS basic idea:**
  - discovers all vertices at distance $k$ from the source vertex before discovering any vertices at distance $k + 1$
  - expanding frontier – “greedy” – propagate a wave 1 edge-distance at a time.
Review: queue and stack data structure

Queues and stacks are dynamic sets in which the elements removed from the set is prescribed.

The queue implements a First-In-First-Out (FIFO) policy.
The stack implements a Last-In-First-Out (LIFO) policy.

Queue supports the following operations:
Enqueue(Q,v): insert element v into the queue Q
Dequeue(Q,v): delete element v from the queue Q

There are several efficient ways to implement queues and stacks, see section 10.1.
Review: queue and stack data structure

- Queues and stacks are dynamic sets in which the elements removed from the set is prescribed.
Review: queue and stack data structure

- Queues and stacks are dynamic sets in which the elements removed from the set is prescribed.

- The queue implements a First-In-First-Out (FIFO) policy. The stack implements a Last-In-First-Out (LIFO) policy.
Review: queue and stack data structure

- Queues and stacks are dynamic sets in which the elements removed from the set is prescribed.

- The queue implements a First-In-First-Out (FIFO) policy. The stack implements a Last-In-First-Out (LIFO) policy.

- Queue supports the following operations:
  - Enqueue(Q,v): insert element v into the queue Q
  - Dequeue(Q,v): delete element v from the queue Q
Review: queue and stack data structure

- **Queues** and **stacks** are dynamic sets in which the elements removed from the set is prescribed.

- The **queue** implements a First-In-First-Out (**FIFO**) policy. The **stack** implements a Last-In-First-Out (**LIFO**) policy.

- Queue supports the following operations:
  - **Enqueue**($Q,v$): insert element $v$ into the queue $Q$
  - **Dequeue**($Q,v$): delete element $v$ from the queue $Q$

- There are several efficient ways to implement queues and stacks, see section 10.1.
Breadth-First Search (BFS)

BFS(G,s)
for each vertex u in V-{s}
    d[u] = +infty
endfor
d[s] = 0
Q = empty  // create FIFO queue
Enqueue(Q, s)
while Q not empty
    u = Dequeue(Q)
    for each v in Adj[u]
        if d[v] = +infty
            d[v] = d[u] + 1
            Enqueue(Q, v)
        endif
    endfor
endwhile
return d
Breadth-First Search (BFS)

- Breadth-First spanning tree
- Running time: $O(|V| + |E|)$

Note: not $\Theta(|V| + |E|)$!
Breadth-First Search (BFS)

- Breadth-First spanning tree
- Running time: $O(|V| + |E|)$

  $O(|V|)$: every vertex enqueued at most once

Note: not $\Omega(|V| + |E|)$!
Breadth-First Search (BFS)

- Breadth-First spanning tree
- Running time: \(O(|V| + |E|)\)

\(O(|V|)\): every vertex enqueued at most once

\(O(|E|)\): every vertex dequeued at most once and we examine \((u, v)\) only when \(u\) is dequeued at most once if directed, at most twice if undirected.

Note: not \(\Theta(|V| + |E|)\)!
Breadth-First Search (BFS)

- Breadth-First spanning tree
- Running time: $O(|V| + |E|)$

$O(|V|)$: every vertex enqueued at most once

$O(|E|)$: every vertex dequeued at most once and we examine $(u, v)$ only when $u$ is dequeued at most once if directed, at most twice if undirected.

Note: not $\Theta(|V| + |E|)$!
Breadth-First Search (BFS)

- Breadth-First spanning tree
- Running time: \( O(|V| + |E|) \)
  
  \( O(|V|) \): every vertex enqueued at most once
  
  \( O(|E|) \): every vertex dequeued at most once and we examine \((u, v)\) only when \(u\) is dequeued at most once if directed, at most twice if undirected.

  **Note:** not \( \Theta(|V| + |E|) \)!

- Correctness of BFS
  shortest path proof – see pp.597-600 of [CLRS,3rd ed.]
  similar with weighted edges – Dijkstra’s algorithm – *to be discussed*