# DFS and BFS – non-recursive versions

## $\mathbf{DFS(G, v_0)}$

**Input:** graph $G$ and starting vertex $v_0$

**Output:** labels $L(v)$ for each vertex $v$ that give the order in which the search visits the vertices. (Each vertex label is initially zero, with a zero label designating an unvisited vertex.)

- $T \longleftarrow \{v_0\}$ ($T$ is maintained as a stack)
- For all vertices $v$, $L(v) \longleftarrow 0$.
- $Count \longleftarrow 1$.
- $L(v_0) \longleftarrow Count$
- While $T \neq \emptyset$ do
  - $v \longleftarrow$ top of stack $T$
  - If there is a vertex $w$ adjacent to $v$ with $L(w) = 0$ then do
    - $Count \longleftarrow Count + 1$
    - $L(w) \longleftarrow Count$
    - Push $w$ onto stack $T$
  - Else pop $v$ from stack $T$

---

## $\mathbf{BFS(G, v_0)}$

**Input:** graph $G$ and starting vertex $v_0$

**Output:** labels $L(v)$ for each vertex $v$ that give the length of the shortest path from that vertex to $v_0$.

- $T \longleftarrow \{v_0\}$ ($T$ is maintained as a queue)
- For all vertices $v$, $L(v) \longleftarrow 0$.
- $L(v_0) \longleftarrow 0$
- While $T \neq \emptyset$ do
  - $v \longleftarrow$ first item on queue $T$
  - Remove $v$ from $T$
  - For all $w$ adjacent to $v$ with $L(w) = 0$ and $w \neq v_0$ do
    - add $w$ to queue $T$
    - $L(w) \longleftarrow L(v) + 1$

**Efficiency analysis of DFS:**

- We'll measure input size by the number $n$ of vertices in $G$.
- The basic operation will be the label check implicit in the line "If there is a vertex $w$ adjacent to $v$ with $L(w) = 0$ then ..."
- There are at most $n - 1$ of these label checks each time through the "While ..." loop.
- But the "While ..." loop is executed at most $2n$ times: each time either adds a vertex to the stack or takes on off, and each vertex is added once and taken off once.
- This makes the total number of basic steps no more than $n \cdot 2n$, so the algorithm has efficiency in $O(n^2)$.

**Efficiency analysis of BFS:**

- Again, we'll measure input size by the number $n$ of vertices in $G$.
- The basic operation is again a label check, this time implicit in the line "For all $w$ adjacent to $v$ with $L(w) = 0$ do ..."
- There are at most $n - 1$ label checks each time through the "While ..." loop.
- But the "While ..." loop is executed at most $n$ times, since each time one vertex is removed from the queue.
- This makes the total number of basic steps no more than $n^2$, so the algorithm has efficiency in $O(n^2)$.

**Note:** Can you see how the label checks would be handled differently depending on if $G$ is input as an adjacency matrix or as adjacency lists? Is one better than the other?