Graph Traversals

The process of starting at a vertex $u$ and processing all the vertices and edges reachable from $u$ is called a graph traversal.
Algorithm DFS (u)
In: Vertex u of a connected graph G = (V,E)
Out: { A DFS traversal of G starting at u }

Mark(u)
For every edge (u,v) do
  if (u,v) is not labelled then
    if v is marked then
      label (u,v) as "back edge"
  else
    label (u,v) as "discovery edge"
  DFS(v)

DFS(v)
Algorithm path(s, t)

In: Vertices s,t of a graph G = (V, E)
Out: True if there is a path, false otherwise.

// P is an instance or global variable that references an initially empty stack. P will store
// a path from s to t or an empty stack
// if no path exists

Mark(s)
P.push(s)
if s = t then return true
else |
    For each edge (s, u) do
        if u is not marked then
            if path(u, t) = true then return true
        P.pop()
    return false
Algorithm $\text{hasCycles}(u)$

In: Vertex $u$ of a connected graph $G = (V, E)$

Out: true if $G$ has at least 1 cycle; false otherwise

1. $\text{Mark}(u)$
2. For every edge $(u, v)$ do
   - if $(u, v)$ is not labelled then
     - if $v$ is marked then
       - label $(u, v)$ as "back edge"
       - return true
   - else
     - label $(u, v)$ as "discovery edge"
     - if $\text{hasCycles}(v)$ = true then
       - return true
3. return false

Time complexity. Assume that the graph is stored in an adjacency matrix.

One call performs $C_1 + C_2$ operations

Number of calls:
Each vertex is visited once.
All vertices are visited.
There is 1 call per node

$\Rightarrow \sum_{u \in G} (C_1 + C_2 \cdot n) = \sum_{u \in G} C_1 + \sum_{u \in G} C_2 n$

$= C_1 n + C_2 n^2$

is $O(n^2)$
Algorithm \text{count}(u)

In: Vertex \( u \) of a graph \( G = (V, E) \)
Out: Number of vertices reachable from \( u \)

\begin{align*}
\text{C}_1 & \quad \text{Mark}(u) \\
     & \quad c \leftarrow 1 \\
\text{C}_2 & \quad \text{For every edge } (u, v) \text{ do} \\
     & \quad \quad \text{if } v \text{ is not marked then} \\
     & \quad \quad \quad c \leftarrow \text{count}(v) + c \\
\text{return} & \quad c
\end{align*}

Time complexity assuming that the graph is stored in an adjacency list

\[ f(n, m) = \sum_{u \in G} (1 + c_2 \times \text{deg}(u)) = c_1 n + 2m c_2 \]

\[ \leq O(n + m) \]
Algorithm BFS(u)
In: Vertex u of a graph G=(V,E)
Out: {BFS of G starting at u}

Q ← empty queue
Q.enqueue(u)
Mark(u) u.parent ← null

while Q is not empty do

v ← Q.dequeue
Process(v)

for each edge (v, w) do

if (v, w) is not labelled then

if w is marked then

label (v, w) as "cross"

else

label (v, w) as "discovered"
Q.enqueue(w)
Mark(w) w.parent ← v