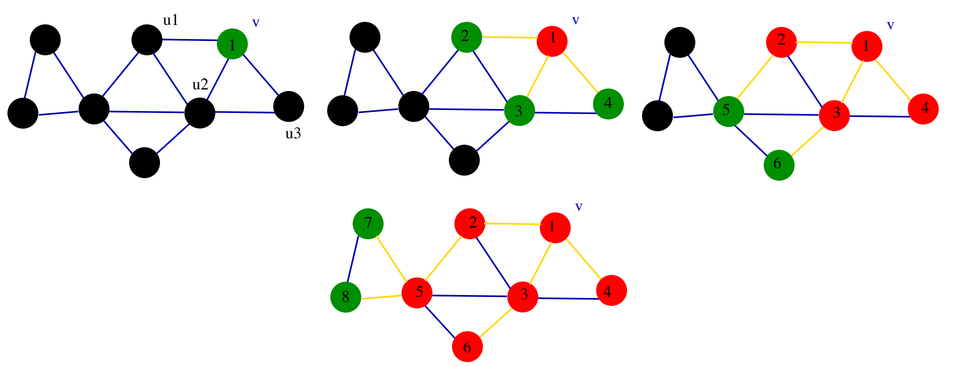
Breadth-First Search (BFS)

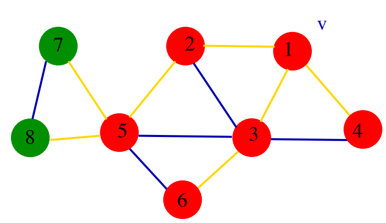
Intuition: **BFS**(vertex v)
To start, all vertices are *unmarked*.

- *Start* at v. *Visit* v and mark as *visited*.
- *Visit* every *unmarked* neighbour u_i of v and mark each u_i as visited.
- *Mark* v finished.
- *Recurse* on each vertex marked as *visited* in the order they were visited.



BFS

Q: *What information about the graph can a BFS be used to find?*



- the *shortest path* from v to any other vertex u and this distance $d(v)$.
- Whether the graph is *connected*.
- Number of *connected components.*

Q: *What does the BFS construct?*

A. a *BFS tree* that visits every node *connected* to v, we call this a *spanning-tree*.

Implementing BFS

Q: What is an appropriate ADT to implement a *BFS* given an adjacency list representation of a graph?

A. A FIFO (first in, first out) queue. which has the operations:

- `ENQUEUE(Q,v)`,
- `DEQUEUE(Q)`,
- `ISEMPTY(Q)`

Q: What *information* will we need to store along the way for each v?

- the current node u and it's *state* (ie, visited, **not visited**, finished)
- the predecessor $p[u]$
- possibly the *distance* $d[u]$ from v if needed
- $o[u]$ the *order* of discovery

The BFS Algorithm

```
BFS(G=(V,E),v):      # Start BFS on G at vertex v
   for u in V:  # Initialize arrays
      state[u]=not_visited; o[u]=-1; d[u]=infinity; p[u]=NIL
   new queue Q
   i = 1
   state[v] = visited                      green = visited
   o[v] = i; d[v] = 0; p[v] = NIL          black = not visited
   ENQUEUE(Q,v)                            red = finished
   while not ISEMPTY(Q):
      u = DEQUEUE(Q)
      for each edge (u,w) in E:
         if (state[w] == not_visited):
            state[w] = visited
            i += 1
            o[w] = i
            d[w] = d[u] + 1
            p[w] = u
            ENQUEUE(Q,w)
      state[u] = finished
```

COMPLEXITY OF BFS(G,V)

Q: *How many times is each node* `ENQUEUE`*ed?*

A. At *most once*, when it is **not visited**, at which point it is marked *visited*.

$\Rightarrow$ the *adjacency list of each node* is traversed at most once.

$\Rightarrow$ so that the total running time of `BFS` is $O(n+m)$ or linear in the size of the adjacency list.

NOTES:

- BFS will visit only those vertices that are *reachable* from v.
- If the graph is *connected (in the undirected case)* or *strongly-connected (in the directed case)*, then this will be all the vertices.
- If not, then we may have to call *BFS* on more than *one start vertex* in order to see the whole graph.

Exercise *Prove that $d[u]$ really does represent the length of the shortest path (in terms of number of edges) from v to u.*