Breath-First Search (BFS)

Graphs

Undirected

V = {a, b, c}

E = {{a,b}, {a,c}, {b,c}}

Directed

V = {a, b, c}

E = {(b,a), (a,c), (b,c), (c,b)}

Graph representation:

Adjacency list (most common):

For all $u \in V$, $Adj[u]$ stores all of u's neighbors, $Adj[u] = \{v \in V : (u, v) \in E\}$.

space: $\Theta(|V| + |E|)$

Adjacency matrix:

A is a two-dimensional array where $A[u, v] = 1$ iff $(u, v) \in E$.

space: $\Theta(|V|^2)$

Breath-First Search (BFS)

Input: a graph $G = (V, E)$ either directed or undirected given as adjacency list, and a "source" vertex s

Output: all vertices reachable from s.

Runtime: $O(|V| + |E|)$ (linear time)

```
BFS(s, Adj)
    level = {s : 0}
    parent = {s : None}   // s.level = 0
    i = 1
    frontier = [s]
    while frontier:
        next = []
        for u in frontier:
            for v in Ajd[u]:
                if v not in level:   // not seen before
                    level[v] = i
                    parent[v] = u
                    next.append(v)
        frontier = next
        i += 1
```

BFS tree with all the parent edges: (parent[v], v)

This tree allows to find the shortest path to s from any u.

BFS works equally well with **directed graphs**.

Slightly simpler, essentially equivalent.

```
BFS(s, Adj)
    frontier = [s]
    s.color = BLACK
    for u ∈ V − {s}:     // all other vertices are white
        u.color = WHITE
    while frontier not empty:
        next = []
        for u in frontier:
            for v in Adj[u]:
                if v.color = WHITE:
                    v.color == BLACK
                    next.append(v)
        frontier = next
```