SEARCHING IN GRAPHS

We already did the most basic form of search: (neighbor query)

In general, given a vertex \( s \),
we want to locate some vertex \( t \),
find a path in \( G \)
or we want to visit all vertices,
in a “local” organized manner.
BREADTH FIRST SEARCH (BFS) - The polite way to search

Start by checking if \( t \) is a neighbor of \( s \).

\[ \Rightarrow \text{look one step away from } s. \]

If yes, done. If not, then check all neighbors-of-neighbors

\[ \Rightarrow \text{one step from each.} \]

Either done, or repeat (dig deeper) ... only on unexplored neighbors!
Search follows a tree pattern.

BFS extends depth by 1 at all possible nodes

(always processing nodes closer to $s$ first)

(each node is processed only once (e.g. $u$))
If $s$ and $t$ are in the same connected component, then the search will find $t$.

Even better, BFS will find the shortest path $s \rightarrow t$.
(prove by contradiction)

time? (supposing we can tell instantly whether a vertex is "new") $O(|V|)$ (in component of $s$)
Algorithm:

1. mark $s$

2. While Q not empty,
   - remove first vertex $v_f$ in Q
   - check $\text{Adj}[v_f]$: $u_1, ..., u_p$
     - if $u_i \neq t$ & unmarked
       - put $u_i$ in Q.
       - mark $u_i$
- mark $s$ & put in $Q$.
- depth($s$) = 0

While $Q$ not empty,
  
  $x = \text{dequeue}(Q)$
  
  check Adj[x]: $u, ..., u_p$
  
  if $u_i$ is unmarked
    mark $u_i$ & put in $Q$
    depth($u_i$) = 1 + depth($x$)
    parent($u_i$) = $x$; $u_i \rightarrow \text{child}(x)$
  
  if $v_i = t$ done
  if $v_i \neq t$
    mark as visited
    put in queue $: Q$

(0) mark $s$

(1) check Adj[$s$]: $v_1, ..., v_k$

  if $v_i = t$ done

  if $v_i \neq t$
    mark as visited
    put in queue $: Q$

(2) While $Q$ not empty,

- remove first vertex $v_f$ in $Q$
- check Adj[$v_f$]: $u_1, ..., u_p$

  if $u_i = t$ done

  if $u_i \neq t$ & unmarked
    put $u_i$ in $Q$.