Algorithms: Graph Search (BFS and Connected Components)

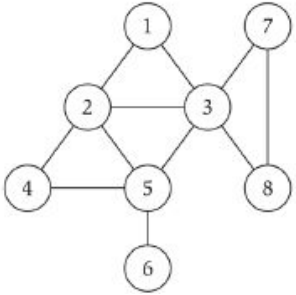
Connectivity

s-t connectivity problem. Given two node s and t, is there a path between s and t?

s-t shortest path problem. Given two node s and t, what is the length of the shortest path between s and t?

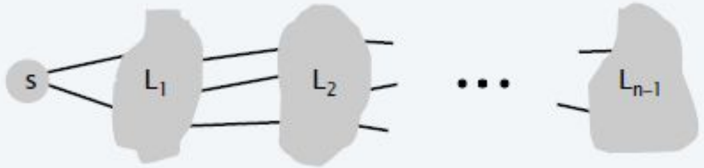
Applications.

- Navigation (Google Maps).
- Maze traversal.
- Kevin Bacon number (or Erdős Number).
- Fewest number of hops in a communication network.



Breadth-first search

BFS intuition. Explore outward from s in all possible directions, adding nodes one "layer" at a time.



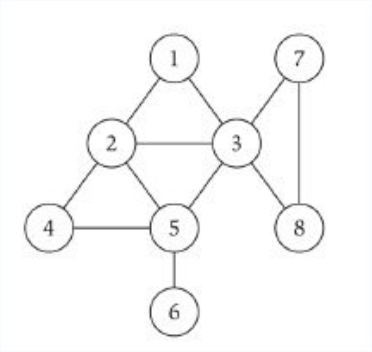
BFS algorithm.

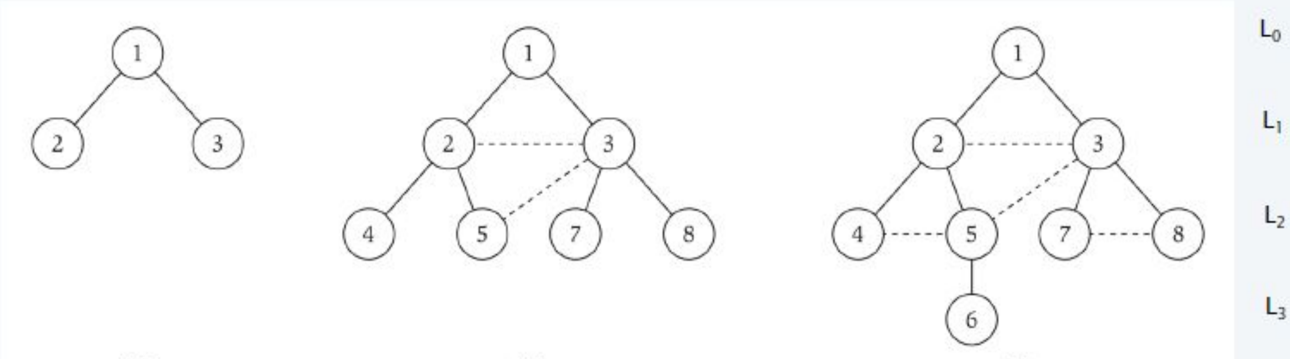
- $L_0 = \{ s \}$.
- L_1 = all neighbors of L_0.
- L_2 = all nodes that do not belong to L_0 or L_1, and that have an edge to a node in L_1.
- L_{i+1} = all nodes that do not belong to an earlier layer, and that have an edge to a node in L_i.

Theorem. For each i, L_i consists of all nodes at distance exactly i from s. There is a path from s to t iff t appears in some layer.

Breadth-first search

Property. Let T be a BFS tree of $G = (V, E)$, and let (x, y) be an edge of G. Then, the levels of x and y differ by at most 1.

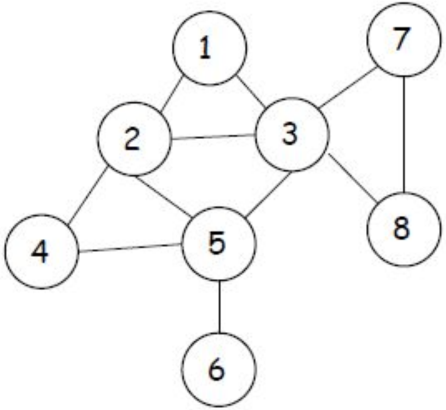


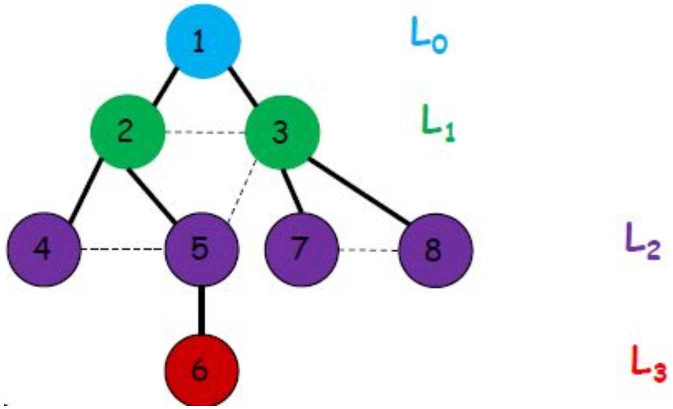


(a) (b) (c)

Breadth First Search

Property. Let T be a BFS tree of $G = (V, E)$, and let (x, y) be an edge of G. Then the level of x and y differ by at most 1.





Breadth-first search: analysis

Theorem. The above implementation of BFS runs in $O(m+n)$ time if the graph is given by its adjacency representation.

Pf.

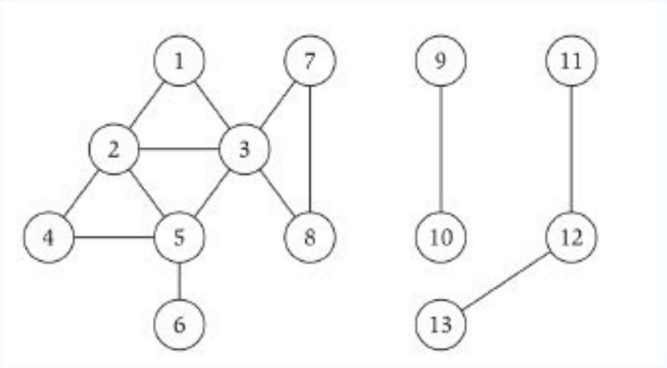
- Easy to prove $O(n^2)$ running time:
  - at most n lists $L[i]$
  - each node occurs on at most one list; for loop runs $\leq n$ times
  - when we consider node u, there are $\leq n$ incident edges (u, v), and we spend $O(1)$ processing each edge

- Actually runs in $O(m+n)$ time:
  - when we consider node u, there are $degree(u)$ incident edges (u, v)
  - total time processing edges is $\Sigma_{u \in V}\ degree(u) = 2m$. ▪

each edge (u, v) is counted exactly twice
in sum: once in degree(u) and once in degree(v)

Connected component

Connected component. Find all nodes reachable from s.

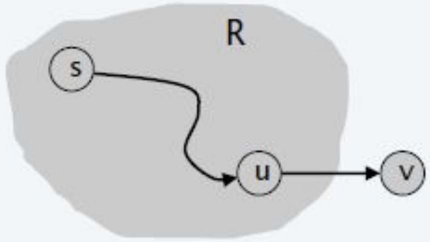


Connected component containing node $1 = \{ 1, 2, 3, 4, 5, 6, 7, 8 \}$.

Connected component

Connected component. Find all nodes reachable from s.

```
R will consist of nodes to which s has a path
Initially R = {s}
While there is an edge (u, v) where u ∈ R and v ∉ R
  Add v to R
Endwhile
```



it's safe to add v

Theorem. Upon termination, R is the connected component containing s.

- BFS = explore in order of distance from s.
- DFS = explore in a different way.

Suggested Reading

- Algorithm Design by Jon Kleinberg, Eva Tardos
  - Chapter 3
    - Section: 3.2