3.2 Graph Traversal

Connectivity

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

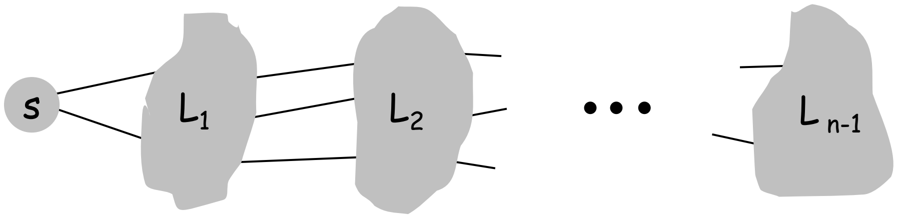
s-t shortest path problem. Given two node s and t, what is the length of the shortest path between s and t?

Applications.
- Hyves.
- Maze traversal.
- Kevin Bacon number.
- Fewest number of hops in a communication network.

Goal: find an algorithm that can answer these questions.

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.

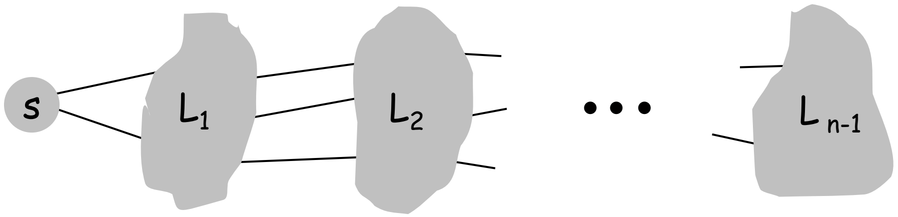
Q. Is there always a path for any node in L_i to s?

Q. If so, what is the length of the path of a node in L_i to s?

TU Delft

Breadth First Search

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



BFS algorithm.

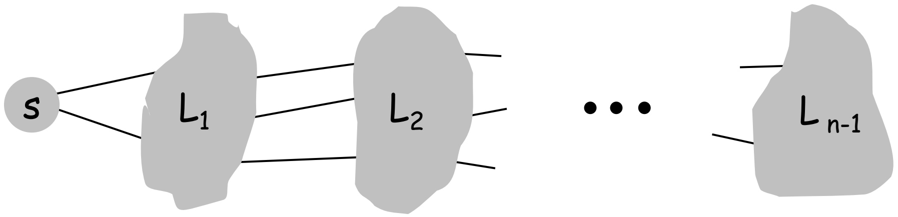
- $L_0 = \{ s \}$.
- L_1 = all neighbors of L_0.
- L_2 = all nodes that do not belong to L_0 or L_1, and that have an edge to a node in L_1.
- L_{i+1} = all nodes that do not belong to an earlier layer, and that have an edge to a node in L_i.

Q. Is there always a path for any node in L_i to s?
A. Yes, only nodes are added that have an edge to the previous node.
Q. If so, what is the length of the path of a node in L_i to s?
A. Exactly i.

TU Delft

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.

TU Delft

Breadth First Search: Analysis

```
i = 0;   L[0]={s}
discovered[s]=true and false for all others
while (L[i] not empty) {
     for (each node u in L[i])
         for (each edge (u,v))
                  if (discovered[v] = false) {
                           discovered[v] = true
                           add v to list L[i+1]
                  }
     i=i+1
}
```

Q. What is the run-time of the BFS algorithm? (1 min)

Breadth First Search: Analysis

```
i = 0;   L[0]={s}
discovered[s]=true and false for all others
while (L[i] not empty) {
    for (each node u in L[i])
        for (each edge (u,v))
            if (discovered[v] = false) {
                discovered[v] = true
                add v to list L[i+1]
            }
    i=i+1
}
```

Q. What is the run-time of the BFS algorithm?

A. $O(n^2)$

- at most n lists L[i]
- each node occurs on at most one list; outermost **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

TU Delft

Breadth First Search: Analysis

```
i = 0;   L[0]={s}
discovered[s]=true and false for all others
while (L[i] not empty) {
    for (each node u in L[i])
        for (each edge (u,v))
            if (discovered[v] = false) {
                discovered[v] = true
                add v to list L[i+1]
            }
    i=i+1
}
```

Q. What is the run-time of the BFS algorithm?

A. O(n+m)

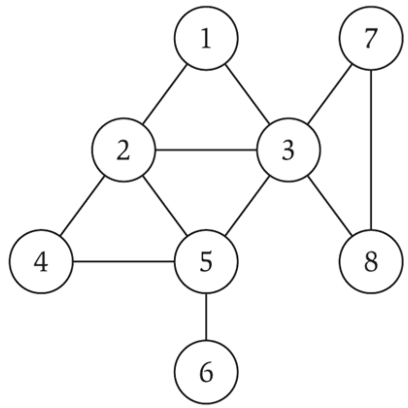
- when we consider node u, there are deg(u) incident edges (u, v)
- total time processing edges is $\Sigma_{u \in V} \deg(u) = 2m$ ← each edge (u, v) is counted exactly twice in sum: once in deg(u) and once in deg(v)
- n steps to setup **`discovered`** array

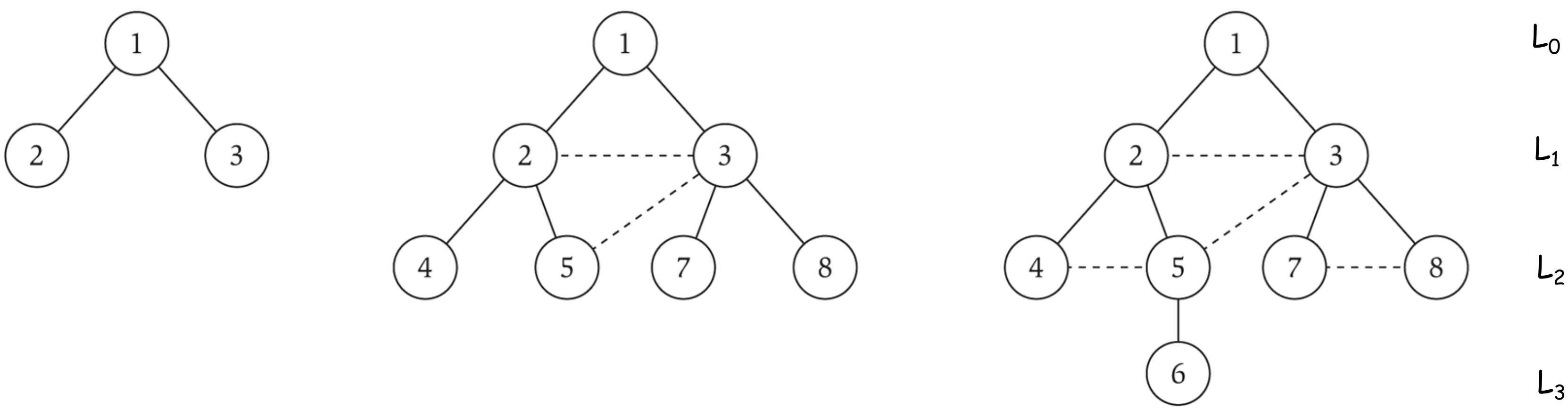
Q. Which graph datastructure is assumed here?

TU Delft

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.

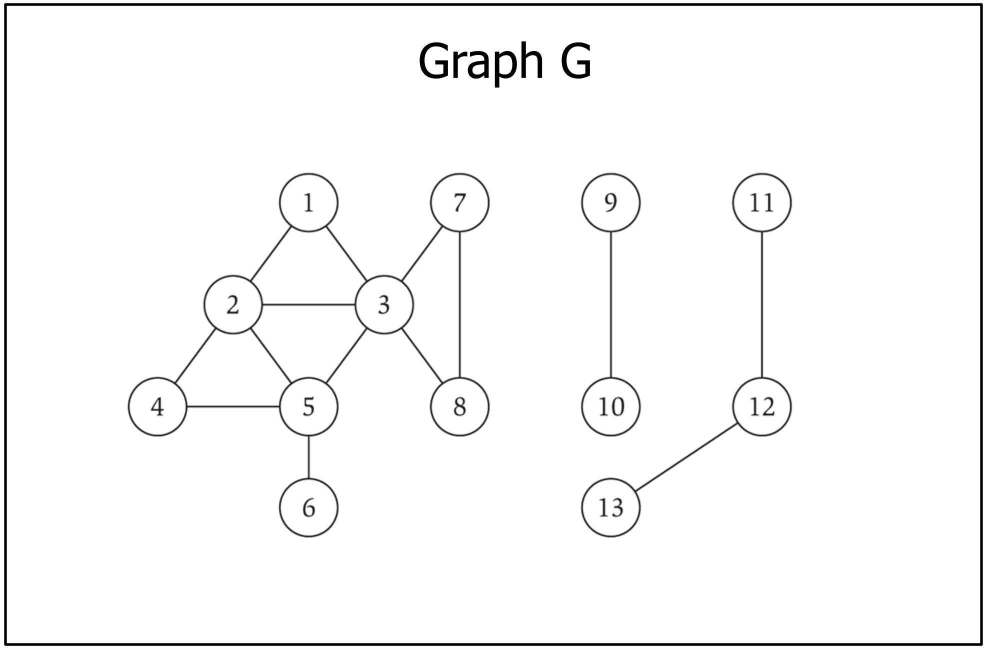




(a) (b) (c)

Connected Component

Connected component. Find all nodes reachable from s.



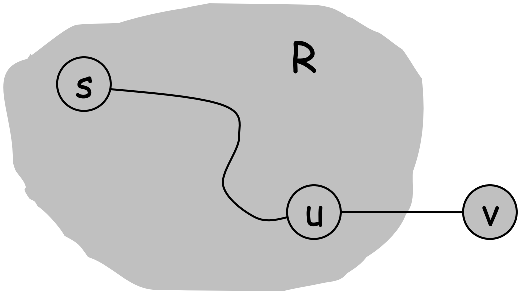
Connected component containing node 1 = { 1, 2, 3, 4, 5, 6, 7, 8 }.

Q. How many connected components are there in this graph?

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
```

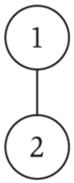


it's safe to add v

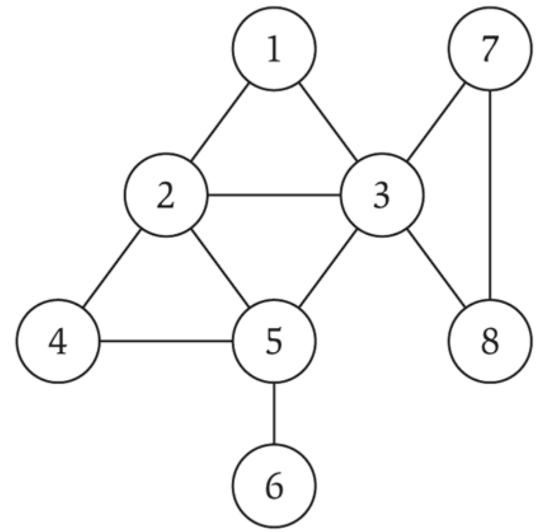
Theorem. Upon termination, R is the connected component containing s.

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

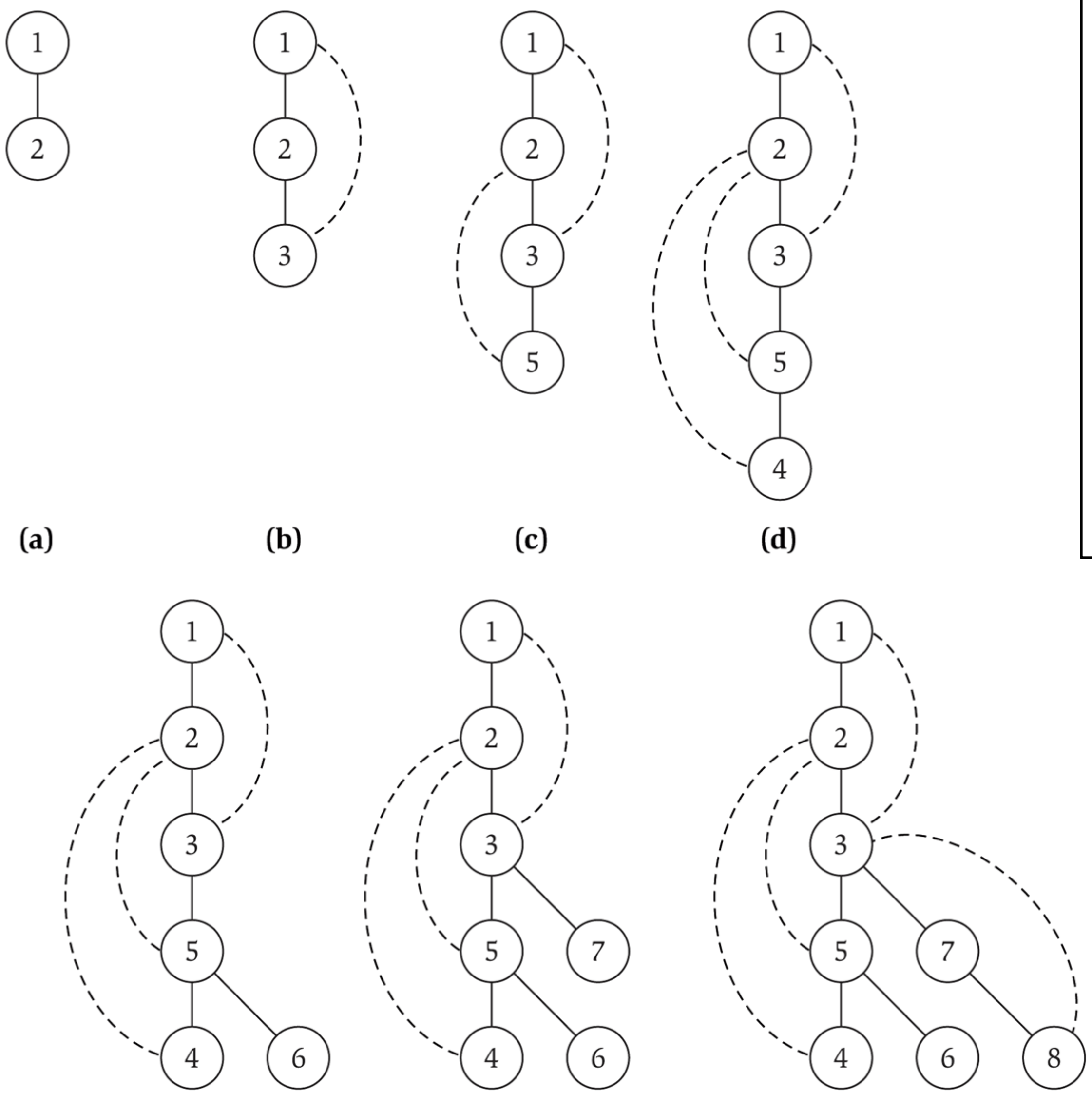
DFS



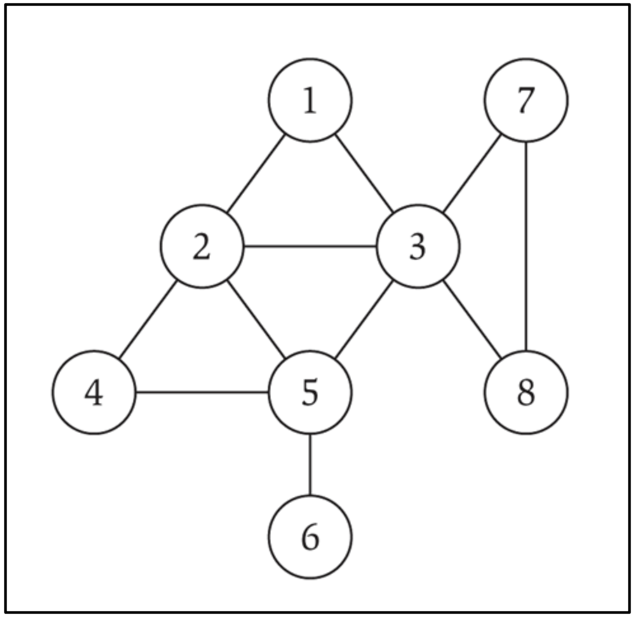
(a)



DFS



(a) (b) (c) (d) (e) (f) (g)



Q. Does this remind you of a search method you've seen last year?

Q. How could you implement this?

DFS

(Like backtracking)

```
DFS(u):
  Mark u as "Explored" and add u to R
  For each edge (u, v) incident to u
    If v is not marked "Explored" then
      Recursively invoke DFS(v)
    Endif
  Endfor
```

TU Delft