Shortest Paths – Unit length edges

With a couple of lines of added code, BFS can compute the length of each vertex's path from s (∞ if not in same connected component)

$$\ell(s)=0; \quad \ell(v) = +\infty \quad \forall v \neq s. \quad \leftarrow \Theta(n)$$

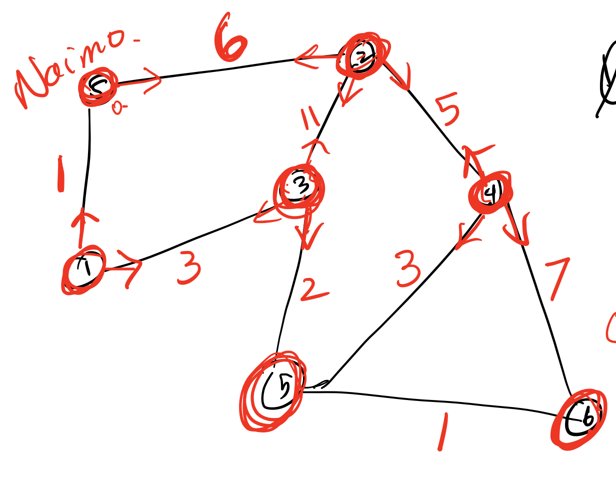
and, when marking w as visited...

$$\ell(w) = \ell(v)+1 \quad \leftarrow \Theta(n)$$

Question: what does this do to running time?

Question: What if the edges have distances...

i.e. not all edges have same value ∞ = INT_MAX



ℓ	0	1	1	~~∞~~ 2	~~∞~~ 2	~~∞~~ 3	~~∞~~ 3	∞
		0	1	2	3	4	5	6

$2+1 = 3$

Q ~~1 2 3 4 5 6~~

... we will figure out an algorithm for this later...

Running time of Augmented BFS

$\Theta(n' + m' + n)$

$\Theta(n + m')$

n_s = number of vertices in connected component of s

m_s = number of edges in connected component of s.

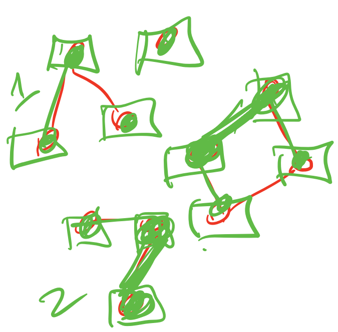
Applications

Detecting Network failures

Data Visualization

Clustering

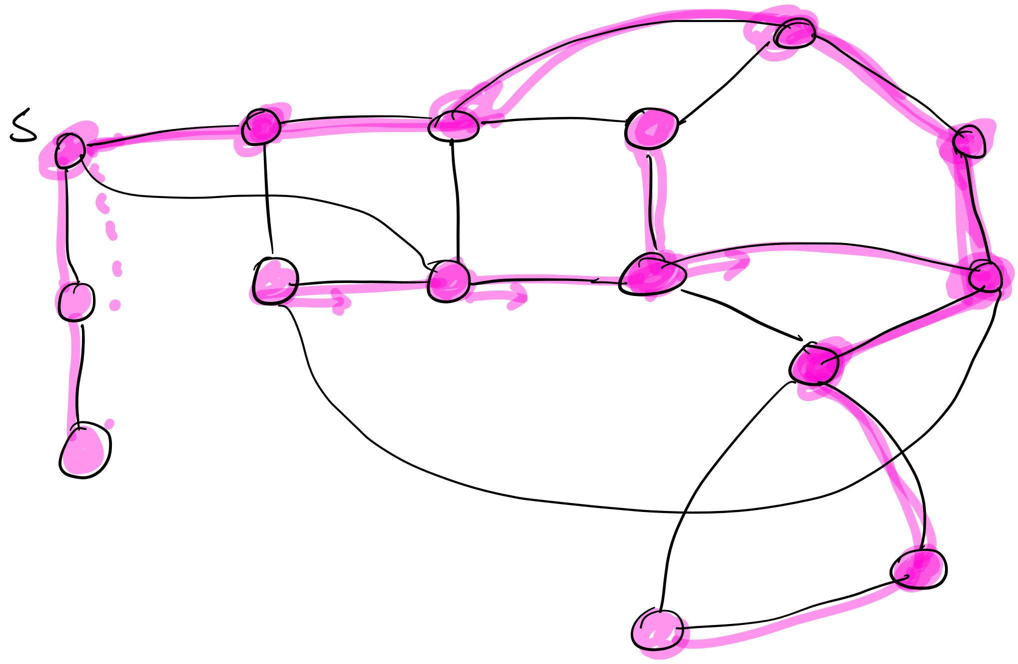
How to compute number of connected components in a graph:



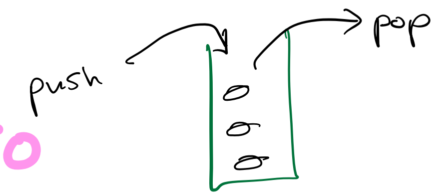
#components = 0.

- While $\exists$ an unmarked vertex v
  - #components ++
  - BFS(v)

Depth-First Search (DFS)



Recall: Stack



LIFO

Iterative version:

DFS

Input: $G=(V,E)$ in adj-list representation; vertex $s \in V$

Post cond: v is reachable from s iff marked[v] = true.

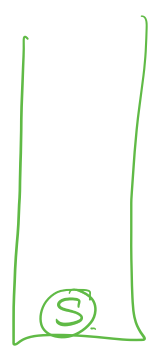
$\forall v \in V$ marked[v] = false

S.init(); S.insert(s) // S is a stack.

while !S.empty()

v = S.front(); S.pop();

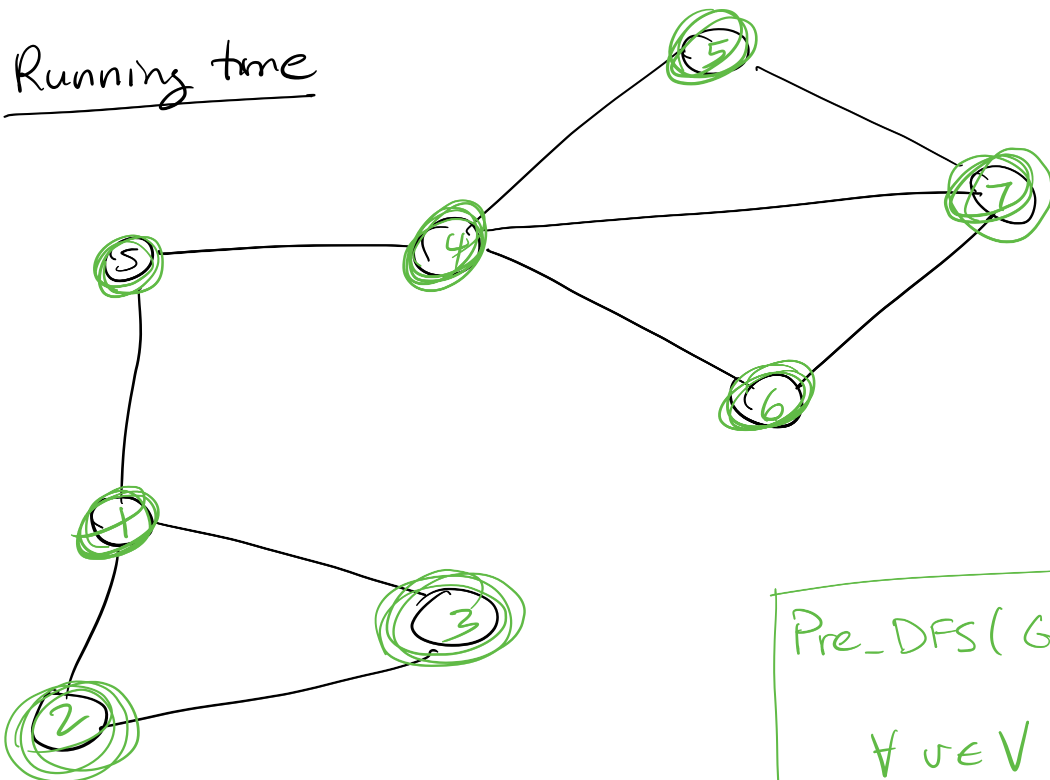
if !marked[v]

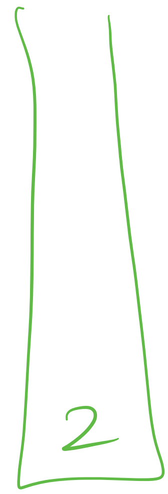


$\forall$ edge (v, w)

$S.\text{push}(w)$

Running time





Pre_DFS(G)

$\forall\, v \in V$

marked[v] = false

Recursive version

Input: graph $G = (V, E)$ in adj list representation

$s \in V$; "set up" call is Pre_DFS(G)

Postcondition: $\forall\, v \in V$, v is reachable from s iff marked[v] == true.

marked[s] = true ✓✓

$\forall$ edge (s, v)
if $!\text{marked}[v]$
DFS (G, v)

Oct 5

Note: "marked" is global to all recursive calls. otherwise it won't work (or will need to be passed as a parameter).

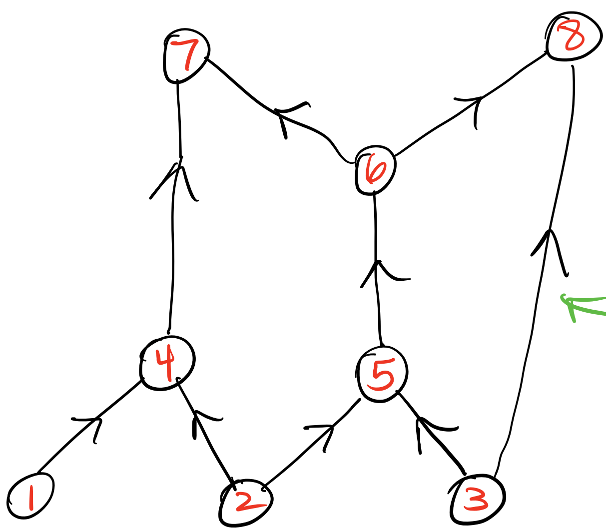
Correctness of DFS:

- it halts because DFS will only be called $\leq$ once on each vertex
- it is correct, as it is a version of generic Search.

(a proof by induction, on distance (in # of edges) will work even better for recursive DFS)

Applications for DFS

Topological Sort



can't do task 8 till have completed task 3.

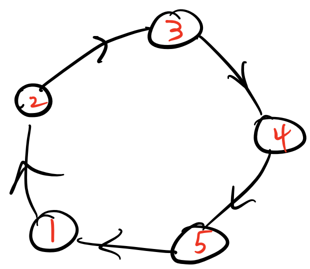
1 2 4 3 5 6 7 8

1 2 3 4 5 6 7 8

3 2 5 6 8 1 4 7

⋮

Can't find a topological sort $\Longleftrightarrow$ $\exists$ directed cycle.



Directed Acyclic graph (DAG)

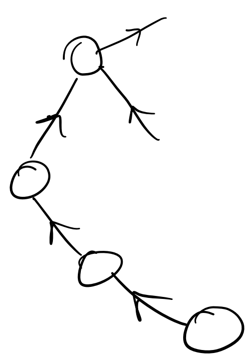
– not necessarily a tree (differs from undirected case)

$\forall$ DAG has a topological sort

$\forall$ DAG has ≥ 1 "source" (no in-edges)

– Keep going backwards along edges

– will either



Setup_DFSTopo(G)

$\forall v \in V$ marked[v] = false

curLabel = $|V|$

$\forall v \in V$ do

if !marked[v]

DFSTopo(G, v)

DFSTopo(G, v)

Input: $G = (V, E)$ in adj-list form; $v \in V$

post cond: f[1] f[2] ... f[|V|] is a topo-sort

marked[v] = true

$\forall$ edge (v, w)

if ! marked[w]

DFSTopo(G, w)

f[v] = curLabel--