contains verts
add one vertex
remove one vertex

```
WHATEVERFIRSTSEARCH(s):
    put s into the bag
    while the bag is not empty
        take v from the bag
        if v is unmarked
            mark v
            for each edge vw    v→w
                put w into the bag
```

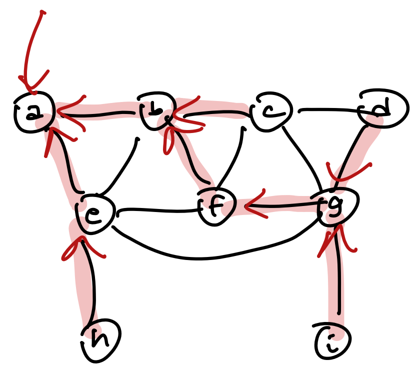
O(E) iterations

```
WHATEVERFIRSTSEARCH(s):
    put (∅, s) in bag
    while the bag is not empty
        take (p, v) from the bag            (⋆)
        if v is unmarked
            mark v
            parent(v) ← p
            for each edge vw                (†)
                put (v, w) into the bag     (⋆⋆)
```

≤2E times

O(V) times

Running time?



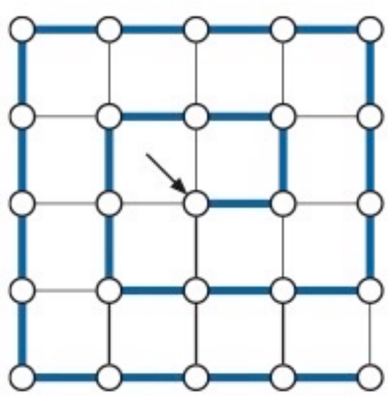
Parent pointers define a spanning tree

Each vertex is marked ≤ once
Each edge is put into bag ≤ twice
taken out ≤ twice

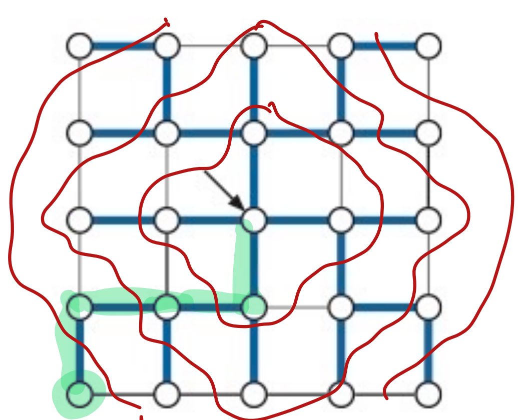
$\boxed{O(V+E) \text{ time}} = O(V^2)$ time

assuming O(1)-time bags

Connected $\Rightarrow E \geq V-1 \Rightarrow V = O(E) \Rightarrow O(E)$ time



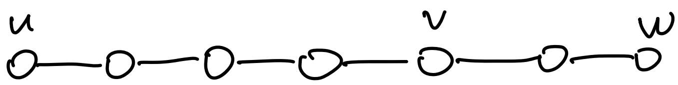
bag = stack
depth-first search



bag = queue
breadth-first search

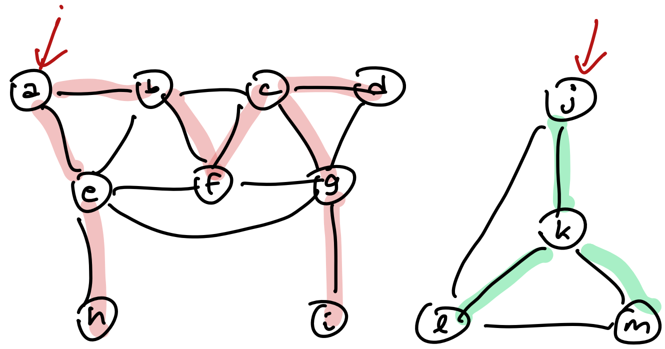
Shortest path tree

```
WFSAll(G):
    for all vertices v
        unmark v
    for all vertices v
        if v is unmarked
            WhateverFirstSearch(v)
```




```
CountComponents(G):
    count ← 0
    for all vertices v
        unmark v
    for all vertices v
        if v is unmarked
            count ← count + 1
            WhateverFirstSearch(v)
    return count
```

O(V+E) time

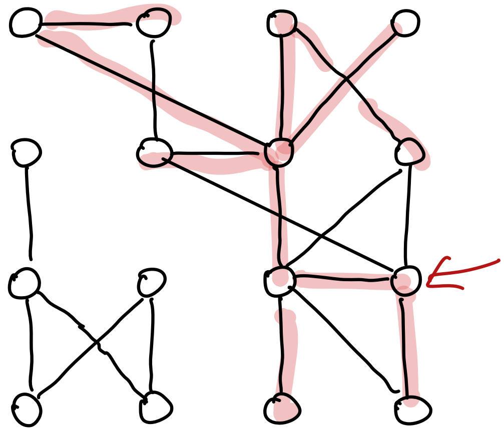


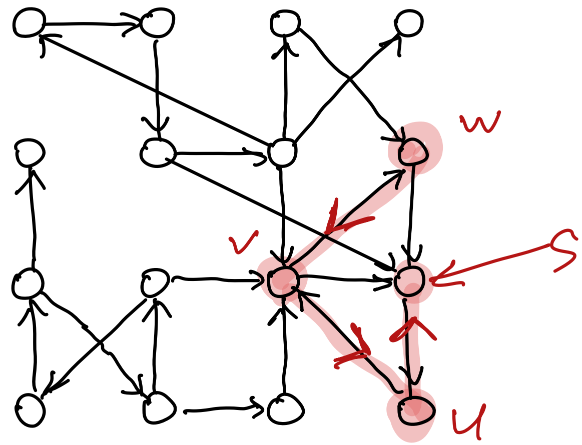
~~0~~ ~~1~~ 2

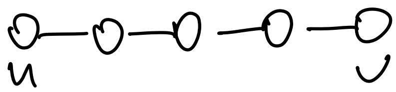
```
CountAndLabel(G):
    count ← 0
    for all vertices v
        unmark v
    for all vertices v
        if v is unmarked
            count ← count + 1
            LabelOne(v, count)
    return count
```

```
⟨⟨Label one component⟩⟩
LabelOne(v, count):
    while the bag is not empty
        take v from the bag
        if v is unmarked
            mark v
            comp(v) ← count
            for each edge vw
                put w into the bag
```

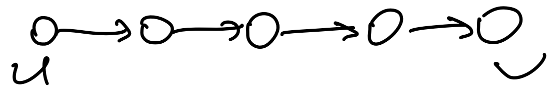
v.comp







u and v are connected



u can reach v

DEPTH-FIRST SEARCH

```
DFS(v):
    mark v
    PreVisit(v)
    for every edge v→w
        if w is unmarked
            parent(w) ← v
            DFS(w)
    PostVisit(v)
```

```
DFSALL(G):
    Preprocess(G)
    for all vertices v
        unmark v
    for all vertices v
        if v is unmarked
            DFS(v)
```

DFS(v):

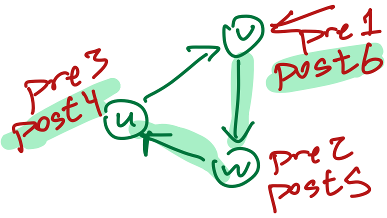
```
mark v
v.pre ← clock++
for every edge v→w
    if w is unmarked
        parent(w) ← v
        DFS(w)
v.post ← clock++
```

DFSALL(G):

```
clock ← 0
for all vertices v
    unmark v

for all vertices v
    if v is unmarked
        DFS(v)
```

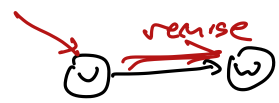
Sort by v.pre —— preorder
Sort by v.post —— postorder



Lemma: G has a directed cycle iff for some edge v→w we have $\boxed{v.post < w.post}$

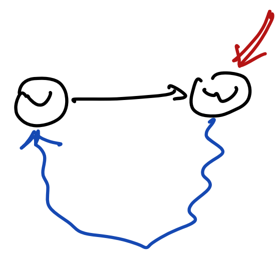
Proof: Let v→w be an arbitrary edge

3 cases:



(1) DFS(v) called before DFS(w)

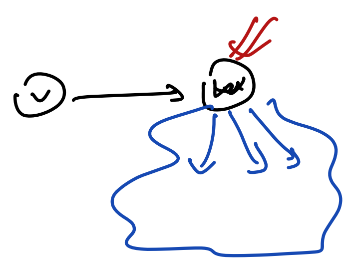
$$v.pre < w.pre < w.post < v.post$$



(2) DFS(w) called before DFS(v) and w can reach v

→ DIRECTED CYCLE!

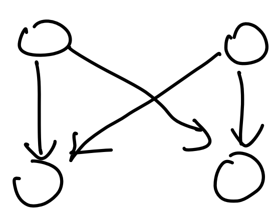
$$w.pre < v.pre < \boxed{v.post < w.post}$$



(3) DFS(w) before DFS(v)
w cannot reach v

$$w.pre < w.post < v.pre < v.post$$

Is v.post > w.post for all v→w then G is a dag



see "DAG" or "dag" $\Longrightarrow$ think "topological sort"

Order the vertices s.t. $num(v) < num(w)$ for all $v \to w$

G is a dag $\Longleftrightarrow$ top. order exists

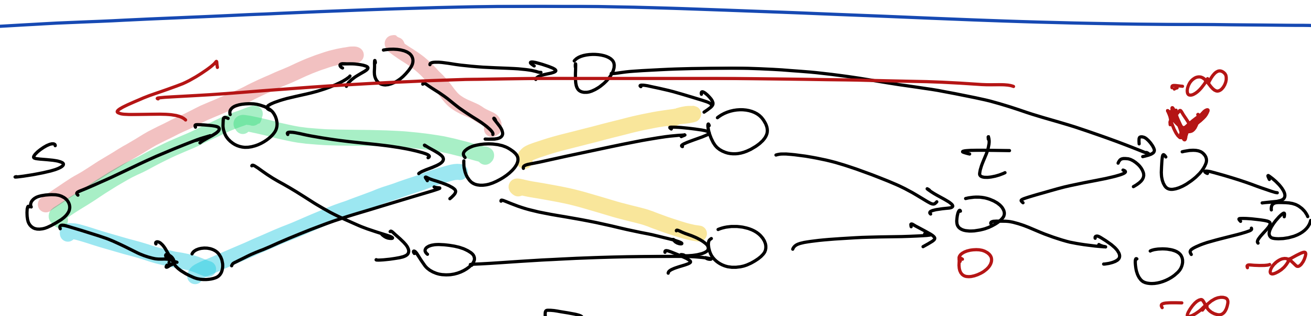
$num(v) = 2V - v.post$

Top Sort(G)

Preprocess(G): Clock $\leftarrow V$ (# vertices)

Previsit(v): return

Postvisit(v): Top[clock--] $\leftarrow v$



longest path from s to t in dag

Dynamic Programming!!

$LP(v)$ = length of the longest path in G from v to t

$$LP(v) = \begin{cases} 0 & \text{if } v = t \\ \max\limits_{v \to w} \{1 + LP(w)\} & \text{if } v \neq t \end{cases}$$

$\max \emptyset = -\infty$

Memoize? Use the graph! v.LP

Eval order? reverse top. order = postorder

Running time? $O(V+E)$