**Union by rank:**

MakeSet(u)
u.p := nil
u.rank := 0

```
Union(u,v):
    r := Find(u)
    s := Find(v)
    if r ≠ s then
        if r.rank < s.rank then
            r.p := s
        if r.rank > s.rank then
            s.p := r
        else // if r.rank = s.rank
            r.p := s
            s.rank ++
```

rank == depth at any node *provided* no path compression during Finds.

But path compression may decrease depth. (never increases)

So: rank ≥ depth at each node.

Set system with union by rank path compression has good performance

m — Union, Find operations
n — items

Total time for such a sequence is $O(m\,\alpha(n))$, where $\alpha(n)$ is very slowly increasing ($\alpha(n)$ amortized time per op).

$\alpha(n) \leq \log^* n$

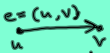
$\log^* n$ = max $i$ such that $\lg^{(i)} n \geq 1$

$\log^* 2^{2^{2^{\cdots^2}}} = i$ (a tower of $i$ 2's)

## Graph Algos

Directed graph $G$
is a pair $(V,E)$
$V$ finite set ("vertices" / "vertex")
$E$ set of edges (ordered pairs of vertices)

Draw: vertices are points (or circles)
edges are arrows connecting the vertices




u tail, v head
e goes from u to v
u is adjacent to v
v is adjacent from u.

An *undirected* graph is similar except edges are unordered pairs $e = \{u,v\}$


Consider an undir. graph as a special sort of directed graph

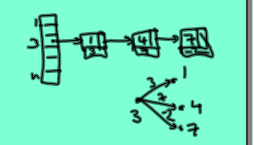

Representations (two):

Adjacency matrix:
$G = (V,E)$
assume $V = \{1,\ldots,n\}$
$n \times n$ boolean matrix

$A_G[i,j] = \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{if } (i,j) \notin E \end{cases}$


usually forbid self-loops in undirected graphs; often in digraphs as well.

Adjacency list (edge list) rep:




Edge weights — often associate some quantity with each edge.

Adj list rep is more compact for sparse graphs
Adj. matrix: $\Theta(n^2)$ space
Adj list: $\Theta(n+m)$ space where $m$ = # edges

Nov 15-12:27 PM

# Breadth-First Search (BFS)


Graph Search in general:
start at some source $s$
follow edges to visit
vertices reachable from $s$
in some order.


In the midst of a search,
partition the vertices into
3 sets: white, gray, black

```
BFS(G, s)   // G=(V,E)
// search starting from s
// V=G.V, E=G.E
foreall v≠s
    v.color := white
    v.d     := ∞
    v.π     := nil
initialize empty queue Q
    // Q is a simple FIFO queue
    s.π := nil
    s.d := 0
    s.color := gray
    insert s into Q
    while Q not empty, do
    → remove a vx u from Q
        u.color := black
        // follow edges from u
        for each v adjacent from u do
            if v.color ≠ white
            go onto the next v
            // v.color = white
            v.color := grey
            v.π := u
            // v was discovered
            // by following an
            // edge from u
            v.d := u.d + 1
            //v.d gives distance
            // (#edges) from
            // source to v.
            insert v into Q
    end foreach
end while
```

It can be proved:
For any $v \in V$
v.d gives shortest distance
from $s \to v$
(# edges)
v.π gives v's predecessor
along some shortest path $s \to v$.


Q: 


Time: $O((n+m)(n+m^2))$ } linear time
n = # vertices
m = # edges

---

Skip Depth-First Search

---

## Minimum Spanning trees

Each edge has a cost
u.wt (weight)
connected, undirected graph
Find a subset of edges
connecting all vertices of
minimum total weight.
(If all wt's are positive.)
Such a set of edges will be a tree

A tree is a connected graph
with no cycles

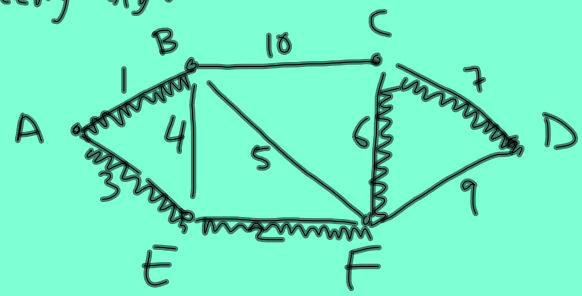

Fact: Let G be a graph with
n vertices and m edges
1. If $m \geq n$, then G has
a cycle.
2. If $m < n-1$, then G
is not connected
3. (therefore) If G is a
tree, then $m = n-1$.

## Kruskal's MST algo:

$T \subseteq E$
$T := \emptyset$
Sort E in ascending order
by edge weight.

Nov 15-12:59 PM

while $|T| < n-1$ do
look at the next edge
$e=(u,v)$ in the sorted list.
if adding $e$ to $T$ would create a cycle in $T$, then ignore $e$.
otherwise, add $e$ to $T$. //
greedy algo




Squiggle an edge when adding it to $T$.
add AB to $T$
add EF to $T$
" AE " "
(ignore BE)

How does the algo know that it is not creating a cycle

Maintain ~~edges~~ vertices in a disjoint set system.
Initially, each ~~edge~~ vertex is its own singleton set

when considering edge $e=(u,v)$ to add to $T$
ignore if Find(u)=Find(v)
otherwise add and Union(u,v)

Nov 15-1:33 PM