CMPT307: Disjoint Sets & BFS

Week 10-2

Xian Qiu

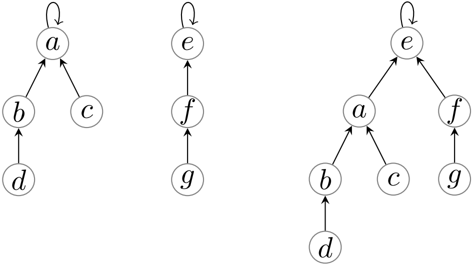
Simon Fraser University

xianq@sfu.ca

represent each dynamic set by a tree

- ▷ MAKE-SET creates a tree with one node

- ▷ FIND-SET?

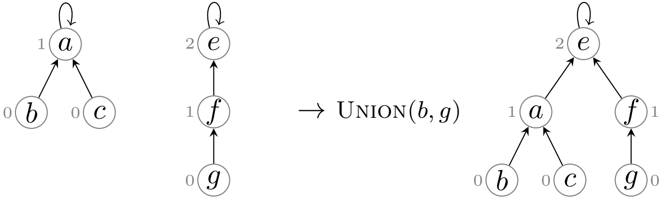


- ▷ UNION(d, f)?

a sequence of $n-1$ UNION may yield a linear chain of n nodes!

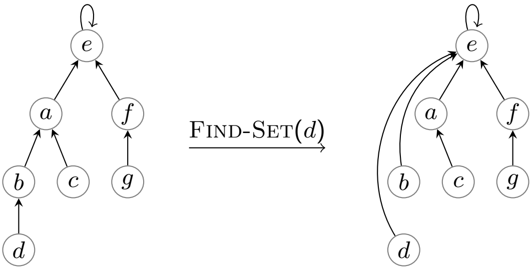
Union by Rank

- intuitive idea: point T_1.root to T_2.root if $|T_1| \leq |T_2|$
- use rank: an upper bound on the height of the node
- point T_1.root to T_2.root if "T_1.root.rank $\leq T_2$.root.rank"



Path Compression

compress path in FIND-SET



- pass compression does not increase any ranks

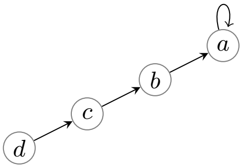
Implementations

MAKE-SET(x) $\{x.\mathsf{p} = x;\ x.\mathsf{rank} = 0; \}$

FIND-SET(x)

---

if $x \neq x.p$ **then**
  $x.\mathsf{p} = \text{FIND-SET}(x.\mathsf{p})$;
return x.p;



Implementations

UNION(x, y) $\{$LINK(FIND-SET(x), FIND-SET(y));$\}$

LINK(x, y)

```
if x.rank > y.rank then
    y.p = x;
else
    x.p = y;
    if x.rank == y.rank then
        y.rank = y.rank + 1;
```

- a node has rank at most $n - 1$
- the rank is an upper bound of node height

Running Time

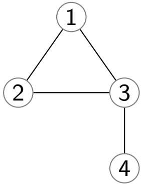
- consider m operations, including n times of MAKE-SET

Theorem

Using union by rank and path compression, the worst case running time is $O(m\alpha(n))$, where $\alpha(n)$ is a very slowly growing function.

- $\alpha(n) \leq 4$ for practical n, say for $n \leq 10^{80}$
- amortize running time per operation is $O(\alpha(n))$
- proof *cf.* Section 21.4

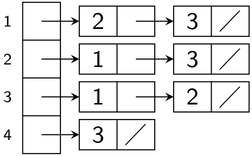
Representations of Graphs



adjacency matrix

$$\begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

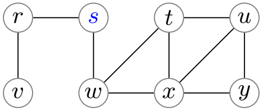
adjacency list



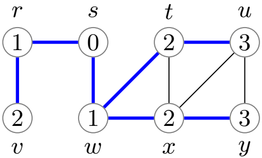
directed graph?

weighted graph?

- given $s \in V$, to discover reachable nodes from s
- compute the shortest distance from s to any reachable v
- also output a search tree



breadth first, then depth



w	r	t	x	v	u	y

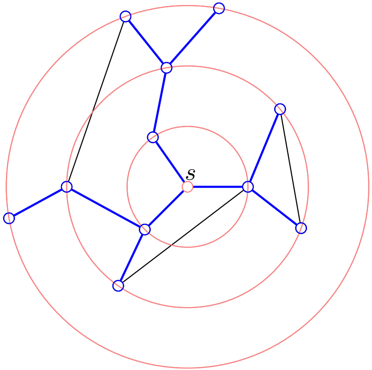
how to identify discovered nodes?

Pseudocode

BFS(G, s)

```
initialize u ∈ V − s: u.d = ∞; u.p = nil;          // ".d" = distance
s.d = 0; s.p = nil ;                                // ".p" = parent
Q = ∅;                                              // initialize queque
ENQUEUE(Q, s);
while Q ≠ ∅ do
    u = DEQUEUE(Q);
    for each v ∈ V that is adjacent to u do
        if v.d == ∞ then
            v.d = u.d + 1;
            v.p = u;                                // record searching path
            ENQUEUE(Q, v);
```

running time?



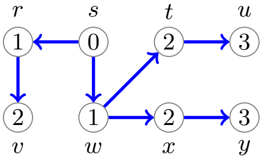
grow a “ball” centerred at s

Breadth-First Tree

breadth-first tree: $T_{\text{bfs}} = (V_p, E_p)$, where

$$V_p = \{v \in V \mid v.\text{p} \neq \text{nil}\} + s$$ reachable nodes from s

$$E_p = \{(v.\text{p}, v) \mid v \in V - s\}$$



PRINT-PATH(G, s, v)

```
if v == s then
    print s;
else if v.p == nil then
    print "no available path";
else
    PRINT-PATH(G, s, v.p);
    print v;
```