Union/find

10/28/24

Recall: Kruskal's MST algorithm

create a "tree" for each node

for each edge (u,v) in order of non-decreasing weight

    if u and v are in different components

        add (u,v) to the MST

        merge their components

Recall: Kruskal's MST algorithm

create a "tree" for each node

for each edge (u,v) in order of non-decreasing weight

    if u and v are in different components

        add (u,v) to the MST

        merge their components

How do we perform these operations?

Disjoint set operations

- Make-set(v): Create a set {v}
- Find(v): Return representative member of the set containing v
  - Allows comparison to determine if two values are in the same set
- Union(u,v): Join the sets containing u and v

Basic uptree implementation

Data structure: Each node has a “parent” pointer p

Make-set(v): v.p = v

Find(v):

```
curr = v;
while(curr.p != curr)
        curr = curr.p;
return curr
```

Union(u,v):

```
u_rep = Find(u)
v_rep = Find(v)
u_rep.p = v_rep
```

Basic uptree implementation

Data structure: Each node has a “parent” pointer p

Make-set(v): v.p = v

Find(v):

```
curr = v;
while(curr.p != curr)
        curr = curr.p;
return curr
```

Union(u,v):

```
u_rep = Find(u)
v_rep = Find(v)
u_rep.p = v_rep
```

What is the worst case running time for a Union or Find?

A. O(1)
B. O(log n)
C. O(n)
D. O(n log n)
E. None of the above

Basic uptree implementation

Data structure: Each node has a “parent” pointer p

Make-set(v): v.p = v

Find(v):

```
curr = v;
while(curr.p != curr)
        curr = curr.p;
return curr
```

Union(u,v):

```
u_rep = Find(u)
v_rep = Find(v)
u_rep.p = v_rep
```

What is the worst case running time for a Union or Find?

A. O(1)
B. O(log n)
C. O(n)
D. O(n log n)
E. None of the above

Union by rank

Data structure: Each node has a “parent” pointer p and an integer rank

Make-set(v): v.p = v; v.rank = 0

Find(v):

```
curr = v;
while(curr.p != curr)
        curr = curr.p;
return curr
```

Union(u,v):

```
u_rep = Find(u)
v_rep = Find(v)
if(u_rep.rank < v_rep.rank)
        u_rep.p = v_rep
else
        v_rep.p = u_rep
        if(u_rep.rank == v_rep.rank)
                u_rep.rank++
```

What does union by rank get us?

Claim: With union by rank, an uptree containing n nodes has height O(log n)

Path compression

Data structure: Each node has a “parent” pointer p and an integer rank

Make-set(v): v.p = v; v.rank = 0

Find(v):

```
if(v != v.p)
        v.p = Find(v.p)
return v.p
```

Union(u,v):

```
u_rep = Find(u)
v_rep = Find(v)
if(u_rep.rank < v_rep.rank)
        u_rep.p = v_rep
else
        v_rep.p = u_rep
        if(u_rep.rank == v_rep.rank)
                u_rep.rank++
```

What does path compression (+ union by rank) get us?

- Define “tower” function:
  - $F(0) = 1$
  - $F(i+1) = 2^{F(i)}$

What does path compression (+ union by rank) get us?

- Define “tower” function:
  - $F(0) = 1$
  - $F(i+1) = 2^{F(i)}$

- $\log^* n$ = least i such that $F(i) \geq n$

  = least i such that $\underbrace{\log \log \ldots \log}_{i \text{ times}} n \leq 1$

What does path compression (+ union by rank) get us?

- Define "tower" function:
  - $F(0) = 1$
  - $F(i+1) = 2^{F(i)}$

- $\log^* n$ = least i such that $F(i) \geq n$

  = least i such that $\underbrace{\log \log \ldots \log}_{\text{i times}} n \leq 1$

What is $\log^* 1{,}000{,}000$?

A. 5 B. 10 C. 15 D. 20 E. None of these

What does path compression (+ union by rank) get us?

- Define "tower" function:
  - $F(0) = 1$
  - $F(i+1) = 2^{F(i)}$

- $\log^* n$ = least i such that $F(i) \geq n$

  = least i such that $\underbrace{\log \log \ldots \log}_{\text{i times}} n \leq 1$

What is $\log^* 1{,}000{,}000$?

A. 5 B. 10 C. 15 D. 20 E. None of these

What does path compression (+ union by rank) get us?

- Define "tower" function:
  - $F(0) = 1$
  - $F(i+1) = 2^{F(i)}$

- $\log^* n$ = least i such that $F(i) \geq n$

  = least i such that $\underbrace{\log \log \ldots \log}_{i \text{ times}} n \leq 1$

Claim: With union by rank and path compression, amortized time for find (and thus for make-set and union) is $O(\log^* n)$