csce750 — Analysis of Algorithms
Fall 2020 — Lecture Notes: Disjoint Sets

This document contains slides from the lecture, formatted to be suitable for printing or individual reading, and with some supplemental explanations added. It is intended as a supplement to, rather than a replacement for, the lectures themselves — you should not expect the notes to be self-contained or complete on their own.

1 *Data Structures for Disjoint Sets*

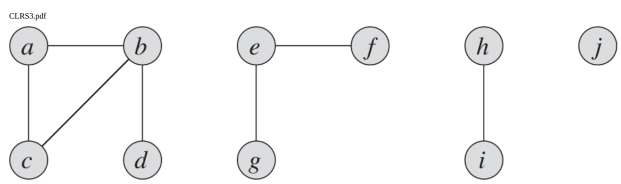
CLRS 21

Data structures for **disjoint sets** support these operations:

- MAKESET(x) — create a new set containing only x.
- UNION(x, y) — union the set containing x with the set containing y.
- FIND(x) — return a unique *representative* of the set containing x.

For analysis, we consider sequences of m total operations, of which n are calls to MAKESET.

2 *Example Application: Connected components of a graph*



(a)

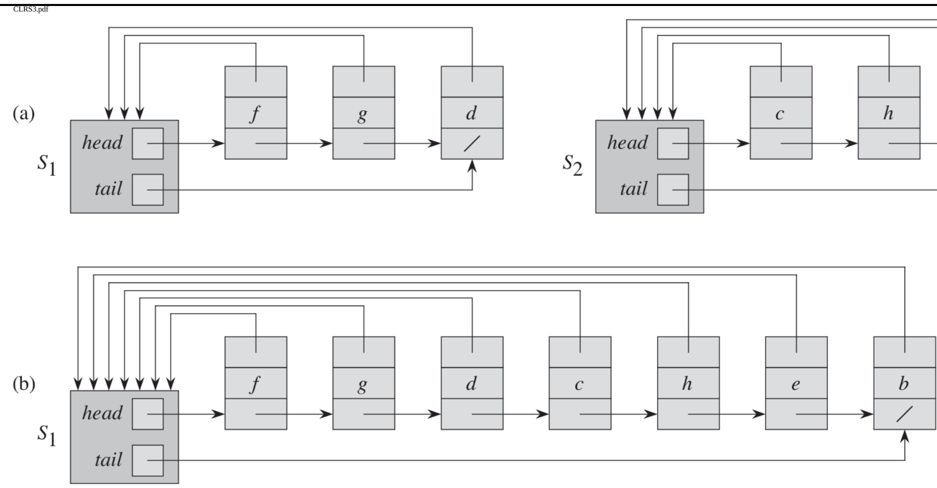
Edge processed	Collection of disjoint sets									
initial sets	{*a*}	{*b*}	{*c*}	{*d*}	{*e*}	{*f*}	{*g*}	{*h*}	{*i*}	{*j*}
(*b*,*d*)	{*a*}	{*b*,*d*}	{*c*}		{*e*}	{*f*}	{*g*}	{*h*}	{*i*}	{*j*}
(*e*,*g*)	{*a*}	{*b*,*d*}	{*c*}		{*e*,*g*}	{*f*}		{*h*}	{*i*}	{*j*}
(*a*,*c*)	{*a*,*c*}	{*b*,*d*}			{*e*,*g*}	{*f*}		{*h*}	{*i*}	{*j*}
(*h*,*i*)	{*a*,*c*}	{*b*,*d*}			{*e*,*g*}	{*f*}		{*h*,*i*}		{*j*}
(*a*,*b*)	{*a*,*b*,*c*,*d*}				{*e*,*g*}	{*f*}		{*h*,*i*}		{*j*}
(*e*,*f*)	{*a*,*b*,*c*,*d*}				{*e*,*f*,*g*}			{*h*,*i*}		{*j*}
(*b*,*c*)	{*a*,*b*,*c*,*d*}				{*e*,*f*,*g*}			{*h*,*i*}		{*j*}

(b)

Pseudocode: CLRS 563

3 *A simple option: Linked lists*

We can implement these operations using a linked list to represent each set:



4 Weighted unions

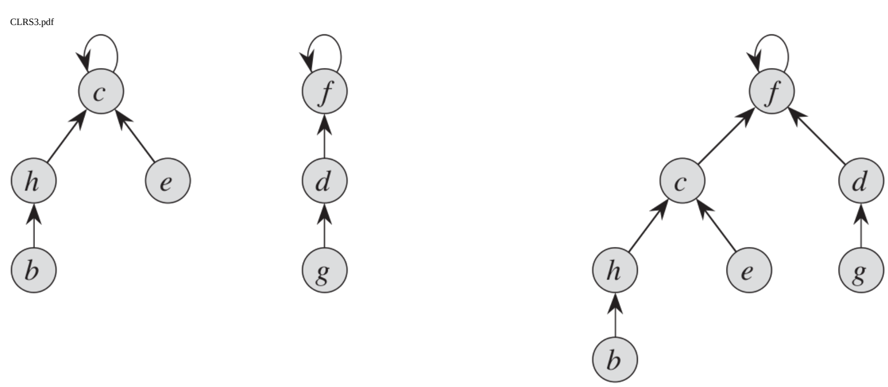
For each UNION, we need to update the pointers on each element of one of the two lists.

- Without this step, we could not do FIND in $O(1)$ time.
- If we always append the shorter list to the longer one, then the entire sequence of operations takes $O(m + n \lg n)$ time.

5 Disjoint Set Forests

We can to better than the linked list approach if we use **trees** instead of lists.

- Each element has a pointer to its **parent**.
- Elements do not keep track of their **children**.
- Root elements are **their own parents**.
- The root of each tree is its **representative**.



6 Disjoint set operations (Simple version)

```
MAKESET(a)
  a.parent = a
```

```
FIND(a)
  if a ≠ a.parent then
    return FIND(a.parent)
  else
    return a
  end if
```

```
UNION(a, b)
  FIND(a).parent = FIND(b)
```

7 Speeding things up

To improve upon the linked list version, we need two enhancements to this basic idea.

- **Union-by-rank** — Each node keeps an upper bound, called its **rank**, on the height of its subtree. For UNION, make the lower-ranked tree a child of the higher-ranked one.
- **Path compression** — During each FIND, rewire the parent pointers to go directly to the root.

8 Disjoint set operations (Real version)

```
MAKESET(a)
  a.parent = a
  a.rank = 0
```

```
FIND(a)
  if a ≠ a.parent then
    a.parent = FIND(a.parent)
  end if
  return a.parent
```

9 Disjoint set operations (Real version)

```
UNION(a, b)
  x = FIND(a)
  y = FIND(b)
  if x.rank > y.rank then
    y.parent = x
  else if x.rank < y.rank then
    x.parent = y
  else
    x.parent = y
    y.rank = y.rank +1
  end if
```

10 Analysis

In a disjoint set forest with union-by-rank and path compression, any sequence of m operations, including n MAKESETs, takes time $O(m\alpha(n))$, in which $\alpha(n)$ is the **inverse Ackermann function**. (Details: CLRS 21.4)

If $n < 16^{512} \approx 10^{616}$ then $\alpha(n) \leq 4$.

(Note: There are only about 10^{80} atoms in the observable universe.)