CS 430 Lecture 23 Activities

Disjoint-set Data Structure

It is useful in many applications to have a structure that handles groups of disjoint sets. In particular, we support the following three operations:

- MAKE-SET(x): Creates a new set consisting of a single element x. Since the sets are disjoint, we assume that this element is not already contained in any set.
- UNION(x, y): Given elements in two different sets, forms the union of two existing sets into one new set.
- FIND-SET(x): Returns a pointer to the representative of the set containing element x.

To identify a set, we return a pointer to any element in the set. The only constraint we make on the element chosen is that if the set does not change between calls to FIND-SET, we must return the same representative.

We analyze the algorithms implementing these operations in terms of n, the number of MAKE-SET operations (all MAKE-SETs are usually assumed to run first), and m, the total number of MAKE-SET, UNION, and FIND-SET operations ($n \leq m$).

1. Give then above, how many possible UNION operations might there be? n make sets; $n - 1$ unions; findsets?

Disjoint-set Application

A graph data structure is a set of vertices and edges between those vertices, and supports problems where there can be relationships between any two items (vertices).

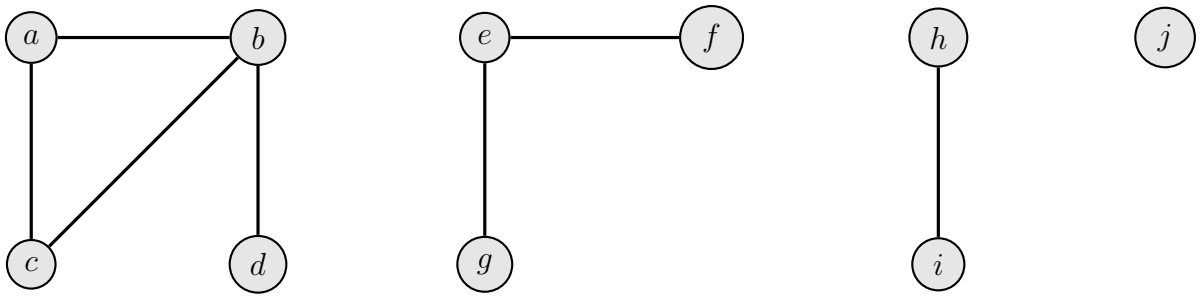


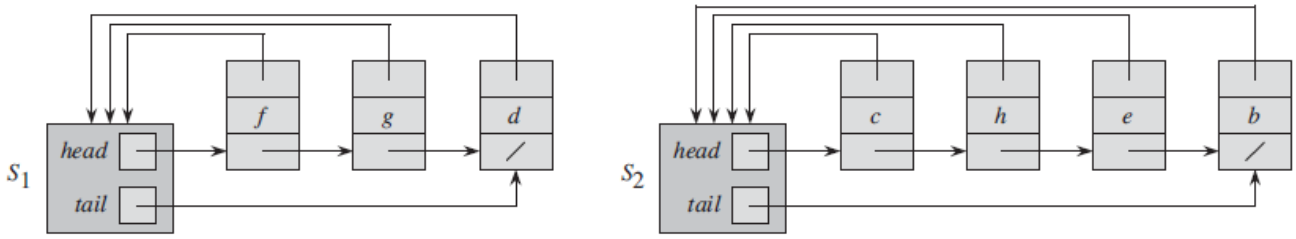
Figure 23.1: $V = \{a, b, c, d, e, f, g, h, i, j\}$, $E = \{\{a, b\}, \{a, c\}, \{c, b\}, \{b, d\}, \{e, g\}, \{e, f\}, \{h, i\}\}$

1. It is easy to see the disjoint sets (connected components) of the graph. Given a graph $G = (V, E)$ write an algorithm using MAKE-SET, UNION, and FIND-SET to find the connected components of any undirected graph.

Algorithm 23.1

```
1: function ConnectedComponents(V, E)
2:     for all v in V do
3:         MakeSet(v)
4:     end for
5:     for all e in E do
6:
7:     end for
8: end function
```

Linked-List Representation



One simple approach is to represent a set as an unordered linked list. Each element contains two pointers, one to the next element (as in a simple linked list) and one to the head of the list. The head serves as the set representative.

2. Describe the algorithms for Make-Set(x) and Find-Set(x) including runtime. For Find-Set(x), assume you already have a reference to x. Make-Set runs in $O(1)$, since you're creating the element, setting the *tail* pointer to this new element, and creating a pointer back to the head. Find-Set also runs in $O(1)$, since the function is not searching for a set element, instead it follows the back pointer to head to get the set label.

3. How do you think Union(x, y) is implemented?

   - Link the two set element lists $O(1)$ together.
   - Update the back pointers O(length of the 2nd list of elments)

4. For n total elements, what is the maximum number of Make-Set and Union operators that would need to be called in the worst case to get all the elements in one set? What is the maximum amortized runtime of each union?

   n Make-Sets $O(1)$ each

   $n-1$ Unions depend on how many back pointers need updating $\Rightarrow \dfrac{O(n^2)}{n-1} = O(n)$ amortized.

5. For n total elements, what is the minimum number of MAKE-SET and UNION operators that would need to be called in the best case to get all the elements in one set? What is the lower bound, worst case amortized runtime of each union? Try the "pairwise approach" to get a lower bound.

Table 23.1:

$\frac{n}{2}$	$1+1$ unions	$1 \times \frac{n}{2} = \frac{n}{2}$
$\frac{n}{4}$	$2+2$ unions	$2 \times \frac{n}{4} = \frac{n}{2}$
$\frac{n}{8}$	$4+4$ unions	$4 \times \frac{n}{8} = \frac{n}{2}$
	$\vdots$	
1	$\frac{n}{2} + \frac{n}{2}$ unions	$1 \times \frac{n}{2} = \frac{n}{2}$

The summation would be $O\left(\frac{n}{2} \lg n\right)$