Lecture 2:

Data structures for disjoint sets

October 7, 2020

Preliminaries

MAIN IDEA: Group n distinct elements into a collection of disjoint sets; the following operations should be efficient:

- Finding the set to which a given element belongs.
- Uniting two sets.

CONTENT OF THIS LECTURE

1. The disjoint-set data structure + specific operations
2. A simple application
3. Concrete implementations based on
   - linked lists
   - rooted trees
4. Discussion: the Ackermann function

Container for a collection $\mathcal{S} = \{S_1, S_2, \ldots, S_n\}$ of disjoint dynamic sets. (A, B are disjoint sets if $A \cap B = \emptyset$.)

- Each set is identified by some member of the set, called its representative
  - ▷ REQUIREMENT: If we ask for the representative of a dynamic set twice without modifying the set, we should get the same answer.

DESIRABLE OPERATIONS

- ▷ MAKESET(x): creates a new set consisting of x only. (Requirement: x is not already in another set.)
- ▷ UNION(x, y): unites the sets that contain x and y, say S_x and S_y, into a new set that is their union. The sets S_x and S_y can be destroyed.
- ▷ FINDSET(x): returns a pointer to the representative of the unique set containing element x.

Disjoint-set data structure

Application: Determining the connected components of an undirected graph

ASSUMPTION: $G = (V, E)$ is an undirected graph.

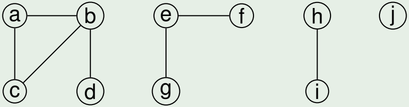
1. Computing the connected components of G:

   CONNECTEDCOMPONENTS(G)
   ```
   for each node v ∈ V
     MAKESET(v)
   for each edge (u, v) ∈ E
     if FINDSET(u) ≠ FINDSET(v)
       UNION(u, v)
   ```

2. Determine if two elements are in the same component:

   SAMECOMPONENT(u, v)
   ```
   if FINDSET(u) = FINDSET(v)
     return TRUE
   return FALSE
   ```

Example (A graph with 4 connected components)



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)	$\{a,b,c,d\}$				$\{e,f,g\}$			$\{h,i\}$		$\{j\}$

MAIN IDEAS

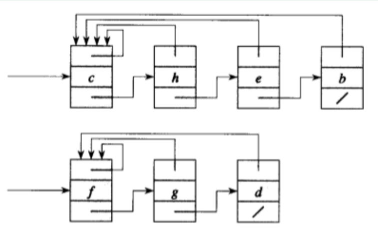
- Each set is represented by a linked list.
- The first element in each linked list is the representative of the set.
- Each object in the linked list contains
  - A pointer to the next set element
  - A pointer back to the set representative
- MAKESET(x) and FINDSET(x) are straightforward to implement
  - They require $O(1)$ time.

Q1: How to implement UNION(x, y)?

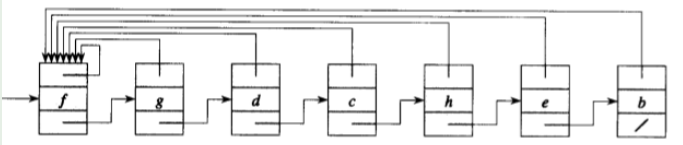
Q2: What is the time complexity of UNION(x, y)?

Example

1. Linked-list representations of sets $\{b, c, h, e\}$ and $\{d, f, g\}$



2. Linked-list representation of their union



Implementation of UNION(x, y)

- Append x-s list onto the end of y-s list and update all elements from x-s list to point to the representative of the set containing y

  $\Rightarrow$ time linear in the length of x-s list.

Implementation of UNION(x, y)

- Append x-s list onto the end of y-s list and update all elements from x-s list to point to the representative of the set containing y

  $\Rightarrow$ time linear in the length of x-s list.

Some sequences of m operations may require $\Theta(m^2)$ time (see next slide)

Disjoint sets represented by linked lists

Example

A sequence of m operations that takes $\Theta(m^2)$ time

Operation	Number of objects updated
$\textsc{MakeSet}(x_1)$	1
$\textsc{MakeSet}(x_2)$	1
$\vdots$	$\vdots$
$\textsc{MakeSet}(x_q)$	1
$\textsc{Union}(x_1, x_2)$	1
$\textsc{Union}(x_2, x_3)$	2
$\textsc{Union}(x_3, x_4)$	3
$\vdots$	$\vdots$
$\textsc{Union}(x_{q-1}, x_q)$	$q-1$

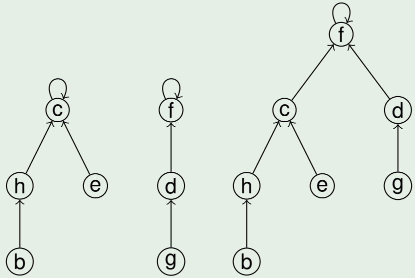
The number of MAKESET ops. is $n = \lceil m/2 \rceil + 1$, and $q = m - n$.

Total time spent: $\Theta(n + q^2) = \Theta(m^2)$ because $n = \Theta(m)$ and $q = \Theta(m) \Rightarrow$ amortized time of an operation is $\Theta(m)$.

MAIN IDEA: Represent sets by rooted trees, with each node containing one member and each tree representing one set.

- A disjoint-set forest is a set of rooted trees, where each member points only to its parent.

Example



Towards a faster implementation

Disjoint-set forests

Implementation of disjoint set operations:

- MAKESET(x): creates a tree with just one node.
- FINDSET(x): follows the parent pointers from a node until it reaches the root of the tree.
  - The nodes visited on the path towards the root constitute the find path.
- UNION(x, y): causes the root of one tree to point to the root of the other tree.

Remarks

1. A sequence of n UNION operations may create a tree which is just a linear chain of nodes

   $\Rightarrow$ Disjoint-set forests have not improved the linked list representation.

2. We need 2 more heuristically improvements: union by rank and path compression.

Implementation of UNION(x, y)

- MAIN IDEA: make the root of the tree with fewer nodes point to the root of the tree with more nodes.
  - Each node has a rank that approximates the logarithm of the size of the subtree rooted at each node and also an upper bound of the height of the node.

  $\Rightarrow$ perform union by rank: the root with smaller rank is made to point to the root with larger rank during the operation UNION(x, y).

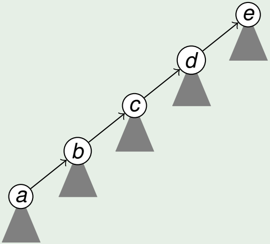
Disjoint-set forests

Heuristic 2: path compression

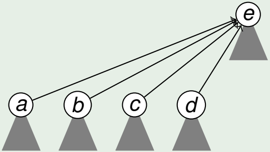
MAIN IDEA: During FINDSET operations, each node on the find path will be made to point directly to the root.

Example

Path compression during the operation FINDSET(a).



(a) before executing FINDSET(a)



(b) after executing FINDSET(a)

Disjoint-set forests

Pseudocode for main operations (1)

- With each node x, we maintain the `int` value $x.rank$ which is an upper bound on the height of x (the number of edges on the longest path between x and a descendant leaf) The initial rank of a node in a newly created singleton tree is 0.

```
MAKESET(x)
x.p = x
x.rank = 0
```

```
UNION(x, y)
LINK(FINDSET(x),FINDSET(y))
```

```
LINK(x, y)
if x.rank > y.rank
    y.p = x
else x.p = y
        if x.rank == y.rank
            y.rank = y.rank + 1
```

Disjoint-set forests

Pseudocode for FINDSET(x)

FINDSET is a **two-pass method**:

1. It makes one pass up the find path to find the root
2. it makes a second pass back down the path to update each node so that it points directly to the root.

FINDSET(x)

```
1 if x ≠ x.p
2     x.p = FINDSET(x.p)
3 return x.p
```

- ▷ Each call of FINDSET(x) returns $x.p$ in line 3.
- ▷ If x is the root then line 2 is not executed and $p[x] = x$ is returned.
  - This is the case when recursion bottoms out.
- ▷ Otherwise, line 2 is executed and the recursive call with parameter $x.p$ returns (a pointer to) the root.
- ▷ Line 2 updates x to point directly to the root.

ASSUMPTIONS:

n = number of MAKESET operations,
m = total number of MAKESET, UNION and FINDSET operations.

- Union by rank has time complexity $O(m \log n)$ [Cormen *et al.*, 2000]
- When we use both path compression and union by rank, the operations have worst-case time complexity $O(m \cdot \alpha(m, n))$ where $\alpha(m, n)$ is the *very slowly growing inverse of Ackermann's function* (see next slides.)
    - On all practical applications of a disjoint-set data structure, $\alpha(m, n) \leq 4$.
    - $\Rightarrow$ we can view the running time as linear in m in all practical situations.

- ▷ Let $g : \mathbb{N} \to \mathbb{N}$ be the function defined recursively by

$$g(i) = \begin{cases} 2^1 & \text{if } i = 0, \\ 2^2 & \text{if } i = 1, \\ 2^{g(i-1)} & \text{if } i > 1. \end{cases}$$

  INTUITION: i gives the *height* of the stack of 2s that make up the exponent.

- ▷ For all $i \in \mathbb{N}$ we define

$$\lg^{(i)}(n) = \begin{cases} n & \text{if } i = 0, \\ \lg(\lg^{(i-1)}(n)) & \text{if } i > 0 \text{ and } \lg^{(i-1)}(n) > 0, \\ \text{undefined} & \text{if } i > 0 \text{ and } \lg^{(i-1)}(n) \leq 0 \text{ or } \lg^{(i-1)}(n) \text{ is undefined.} \end{cases}$$

  where lg stands for $\log_2$

- ▷ $\lg^*(n) = \min\{i \geq 0 \mid \lg^{(i)}(n) \leq 1\}$.

REMARK: $\lg^*(2^{g(n)}) = n + 1$.

References

- Chapter 22: "Data Structures for Disjoint Sets" of
  - T. H. Cormen, C. E. Leiserson, R. L. Rivest. *Introduction to Algorithms*. MIT Press, 2000.