MA 515: Introduction to Algorithms & MA353 : Design and Analysis of Algorithms [3-0-0-6]

Lecture 17

http://www.iitg.ernet.in/psm/indexing_ma353/y09/index.html

Partha Sarathi Manal

psm@iitg.ernet.in

Dept. of Mathematics, IIT Guwahati

Mon 10:00-10:55 Tue 11:00-11:55 Fri 9:00-9:55

Class Room : 2101

Disjoint Sets

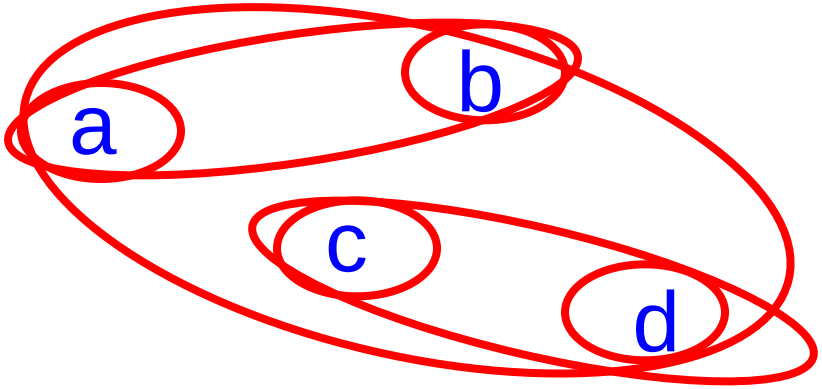
- Disjoint set data structure maintains a collection $S = \{S_1, S_2, \ldots, S_n\}$ of disjoint dynamic sets.
- Each set is identified by a representative, which is some member of the set.
- Three important operations
  - making a new set
  - finding which set a given element belongs to
  - uniting two sets.

Disjoint Sets

- State: collection of disjoint dynamic sets
  - number of sets and composition can change, but must always be disjoint
  - each set has representative element, serves as name of the set
- Operations:
  - Make-Set(x): creates {x} and adds to collection of sets
  - Union(x,y): replaces x's set S_x and y's set S_y with $S_x \cup S_y$
  - Find-Set(x): returns (pointer to) the representative of the set containing x

Disjoint Sets Example

- Make-Set(a)
- Make-Set(b)
- Make-Set(c)
- Make-Set(d)
- Union(a,b)
- Union(c,d)
- Find-Set(b) returns a
- Find-Set(d) returns c
- Union(b,d)



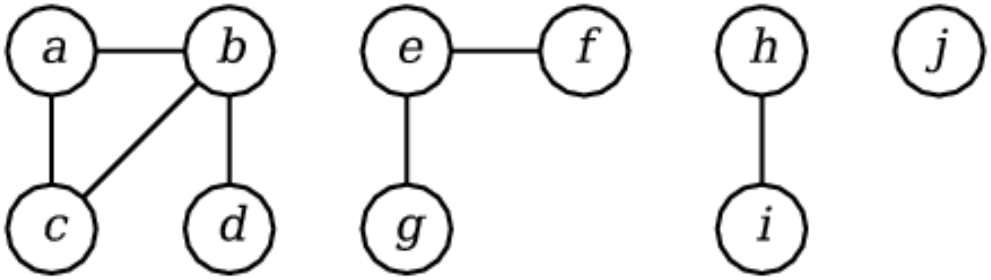
Linked List Representation

- Store set elements in a linked list
  - each list node has a pointer to the next list node
- First list node is set representative (rep)
- Each list node also has a pointer to the first list node (rep)
- Keep external pointers to first list node (rep) and last list node (tail)

Algorithm for connected components

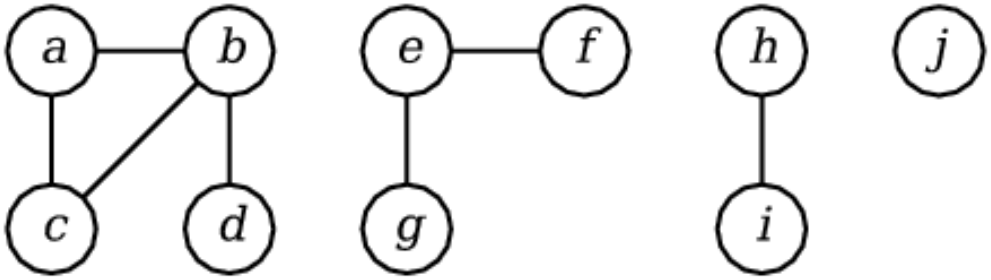
```
CONNECTED-COMPONENTS(G)
1  for each v ∈ V
2      do MAKE-SET(v)
3  for every edge (u, v) ∈ E
4      do if FIND-SET(u) ≠ FIND-SET(v)
5             then UNION(u, v)
```

```
SAME-COMPONENTS(u, v)
1  if FIND-SET(u) = FIND-SET(v)
2     then return TRUE
3  return FALSE
```

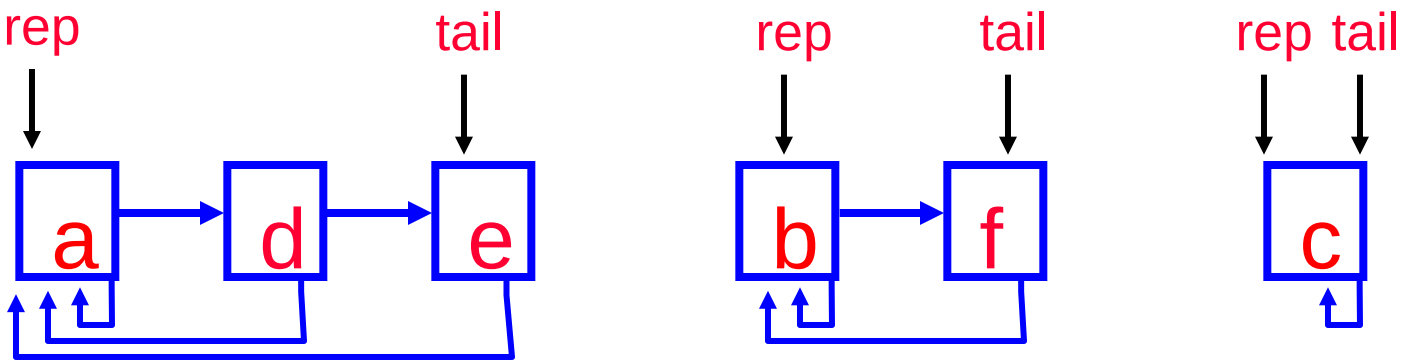


Algorithm for connected components

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



Linked List Representation



Linked List Representation

- Make-Set(x): make a new linked list containing just a node for *x*
  - O(1) time
- Find-Set(x): given (pointer to) linked list node containing *x*, follow rep pointer to head of list
  - O(1) time
- Union(x,y): append *x*'s list to end of *y*'s list and update all rep pointers in *x*'s old list to point to head of *y*'s list
  - O(size of *x*'s old list) time

Time Analysis

- What is worst-case time for any sequence of Disjoint Set operations, using the linked list representation?
- Let m be number of ops in the sequence
  - Make-Set, Union, Find-Set
- Let n be number of Make-Set ops (i.e., number of elements)
- $m \geq n$

Expensive Case

- MS(x_1), MS(x_2), …, MS(x_n),
  U(x_1,x_2), U(x_2,x_3), …, U(x_{n-1},x_n)

- Total time is $O(n^2)$, which is $O(m^2)$
  since $m = 2n - 1$
- So amortized time per operation is $O(m^2)/m = O(m)$

Linked List with Weighted Union

- Always append smaller list to larger list
- Need to keep count of number of elements in list (weight) in rep node
- Calculate worst-case time for a sequence of m operations.
- Make-Set and Find-Set operations contribute $O(m)$ total

Analyzing Time for All Unions

- How many times can the rep pointer for an arbitrary node *x* be updated ?
- First time *x*'s rep pointer is updated, the new set has at least 2 elements
- Second time *x*'s rep pointer is updated, the new set has at least 4 elements
  - *x*'s set has at least 2 and the other set is at least as large as *x*'s set

Analyzing Time for All Unions

- The maximum size of a set is n (the number of Make-Set ops in the sequence)
- So x's rep pointer can be updated at most $\log n$ times.
- Thus total time for all unions is $O(n \log n)$.
- Note style of counting - focus on one element and how it fares over all the Unions

Amortized Time

- Grand total for sequence is $O(m+n \log n)$
- Amortized cost per Make-Set and Find-Set is $O(1)$
- Amortized cost per Union is $O(\log n)$ since there can be at most $n - 1$ Union ops.