Algorithm Theory

Chapter 5
Data Structures

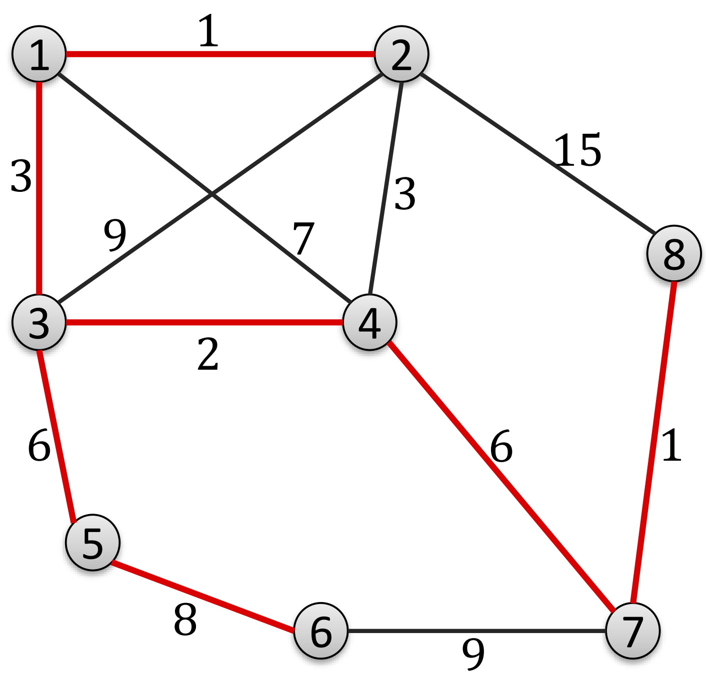
Part I:
Union Find: Basic Implementation

Fabian Kuhn

Minimum Spanning Trees

Kruskal Algorithm:

1. Start with an empty edge set
2. In each step:
   Add minimum weight edge e such that e does not close a cycle



Implementation of Kruskal Algorithm

1. Go through edges in order of increasing weights

   sort edges by weight : $O(m \log n)$ time

2. For each edge $e = \{u, v\}$:

   if e does not close a cycle then

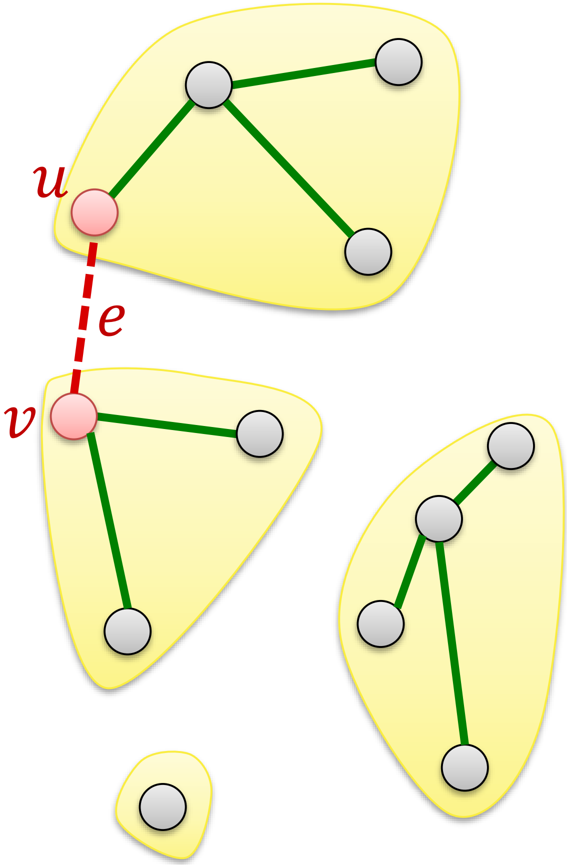
      need to check if e closes a cycle

      $\Downarrow$

      are u and v in same conn. comp.?

      add e to the current solution

      merge the connected components containing nodes u and v.



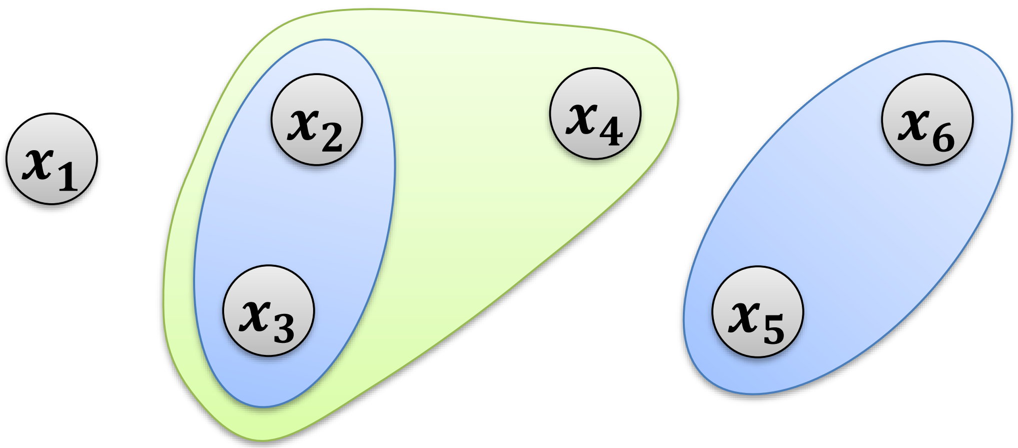
Union-Find Data Structure

Also known as **Disjoint-Set Data Structure**...

Manages partition of a set of elements (a set of disjoint sets)

Operations:

- $\mathbf{make\_set}(x)$**:** create a new set that only contains element x
- $\mathbf{find}(x)$**:** return the set containing x
- $\mathbf{union}(x, y)$**:** merge the two sets containing x and y



Implementation of Kruskal Algorithm

1. Initialization:
   For each node v: $\mathrm{make\_set}(v)$

2. Go through edges in order of increasing weights:
   Sort edges by edge weight

3. For each edge $e = \{u, v\}$:

   if $\mathbf{find}(\boldsymbol{u}) \neq \mathbf{find}(\boldsymbol{v})$ **then**

   add e to the current solution

   $\mathbf{union}(\boldsymbol{u}, \boldsymbol{v})$

Managing Connected Components

- Union-find data structure can be used more generally to manage the connected components of a graph

  … if edges are added incrementally

- $\text{make\_set}(v)$ for every node v
- $\text{find}(v)$ returns component containing v
- $\text{union}(u, v)$ merges the components of u and v
  (when an edge is added between the components)

- Can also be used to manage biconnected components

Basic Implementation Properties

Representation of sets:

- Every set S of the partition is identified with a representative, by one of its members $x \in S$

Operations:

- $\mathrm{make\_set}(x)$: x is the representative of the new set $\{x\}$

- $\mathrm{find}(x)$: return representative of set S_x containing x

- $\mathrm{union}(x, y)$: unites the sets S_x and S_y containing x and y and returns the new representative of $S_x \cup S_y$

Observations

Throughout the discussion of union-find:

- n: total number of make_set operations
- m: total number of operations (make_set, find, and union)

Clearly:

- $m \geq n$
- There are at most $n - 1$ union operations

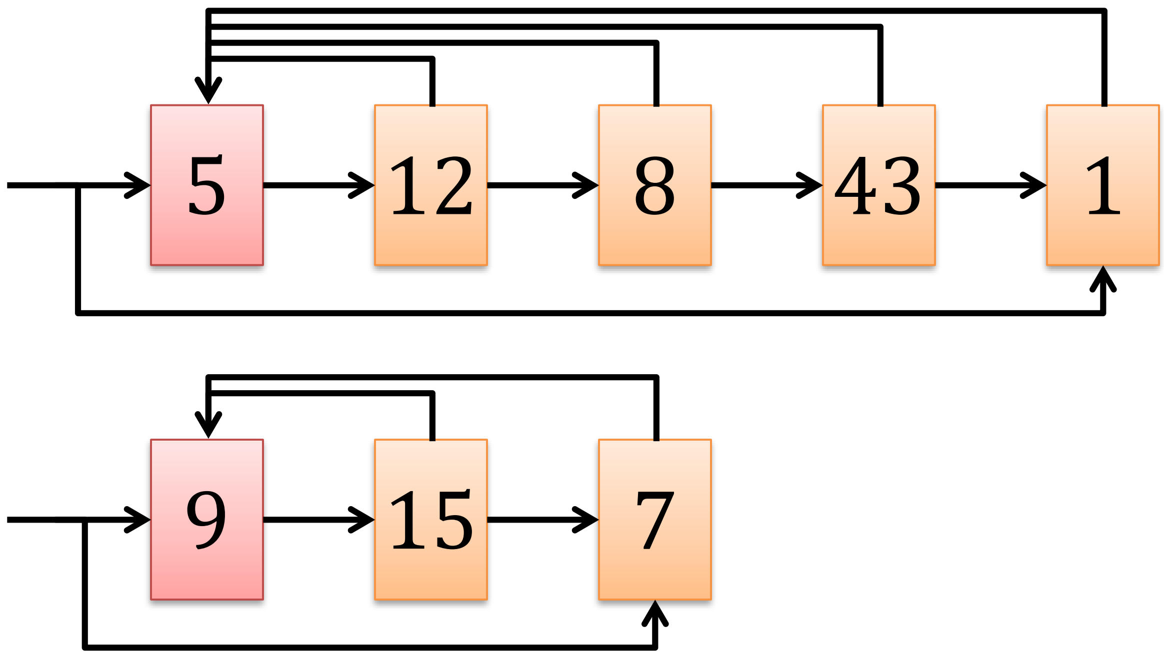
Remark:

- We assume that the n make_set operations are the first n operations
  - Does not really matter…

Linked List Implementation

Each set is implemented as a linked list:

- representative: first list element (all nodes point to first elem.)
  in addition: pointer to first and last element



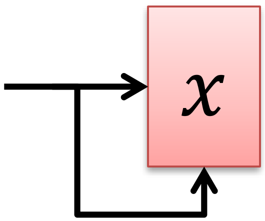
- sets: $\{1,5,8,12,43\}, \{7,9,15\}$; representatives: $5, 9$

Linked List Implementation

make_set($\boldsymbol{x}$):

- Create list with one element:

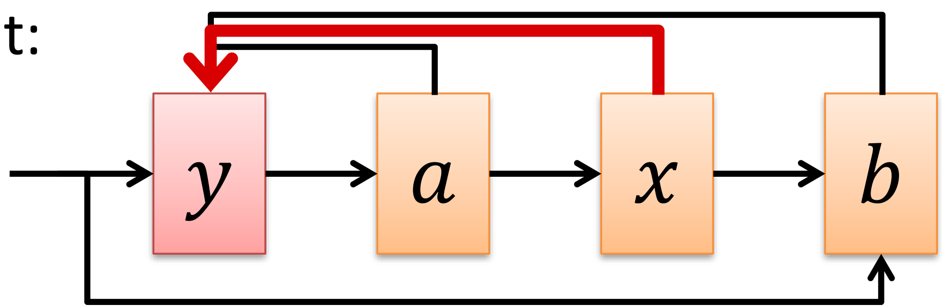
  time: $\boldsymbol{O(1)}$



find($\boldsymbol{x}$):

- Return first list element:

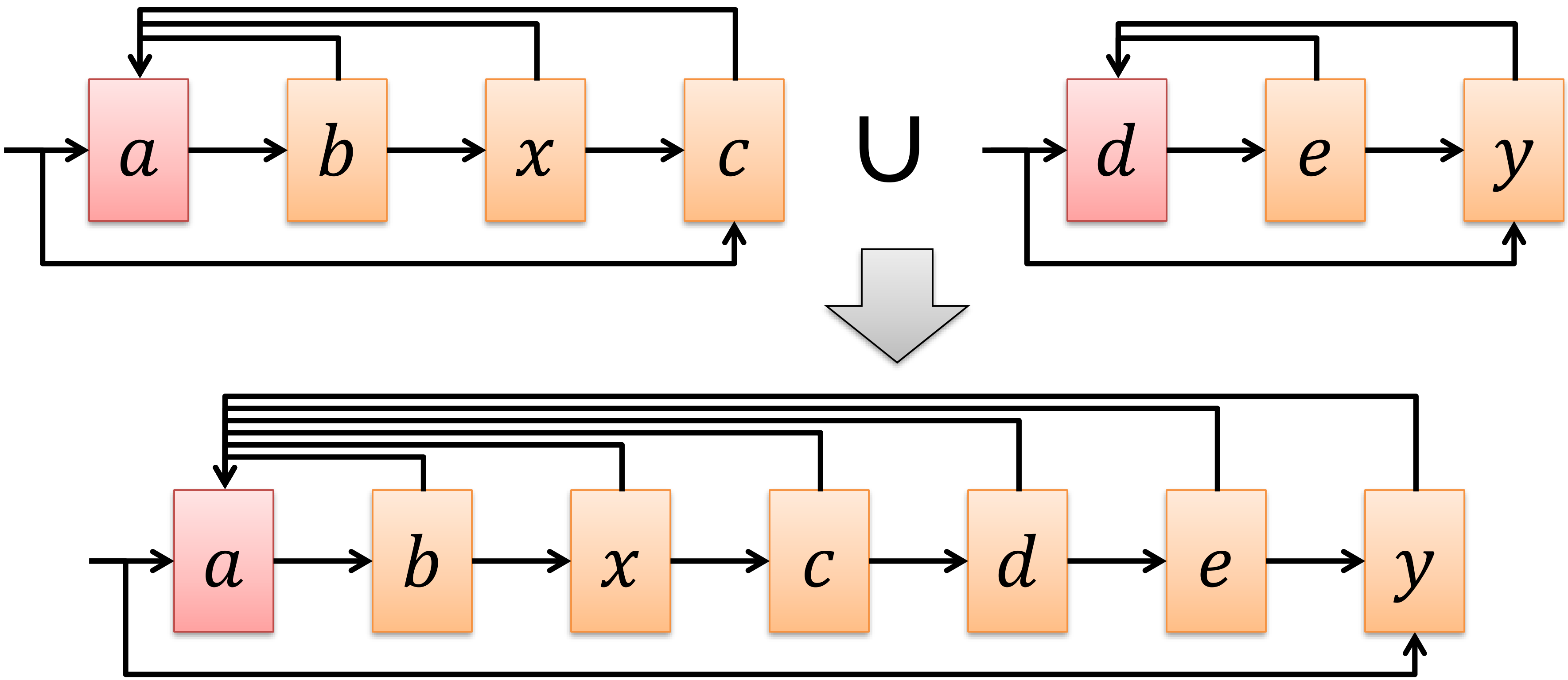
  time: $\boldsymbol{O(1)}$



Linked List Implementation

union(x, y):

- Append list of y to list of x:

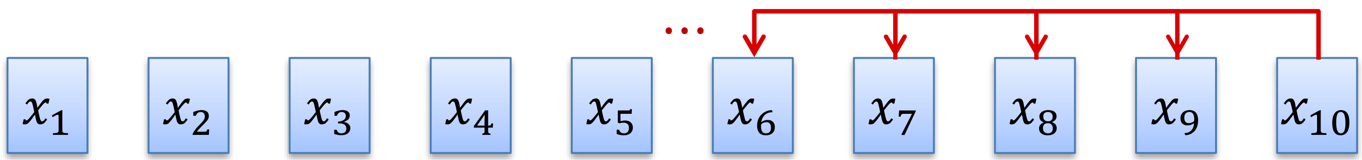


Time: O(length of list of y)

Cost of Union (Linked List Implementation)

Total cost for $n-1$ union operations can be $\Theta(n^2)$:

- $\text{make\_set}(x_1), \text{make\_set}(x_2), \dots, \text{make\_set}(x_n),$ $\text{union}(x_{n-1}, x_n), \text{union}(x_{n-2}, x_{n-1}), \dots, \text{union}(x_1, x_2)$



- #pointer redirections: $1+2+3+\cdots+n-1=\Theta(n^2)$

Union-By-Size Heuristic

- In a bad execution, average cost per union can be $\Theta(n)$
- Problem: The longer list is always appended to the shorter one

Idea:

- In each union operation, append shorter list to longer one!

Cost for union of sets S_x and S_y: $O\big(\min\{|S_x|, |S_y|\}\big)$

Theorem: The overall cost of m operations of which at most $u \leq n$ are union operations is $\boldsymbol{O(m + u \cdot \log n)}$.

n: #make_set op.

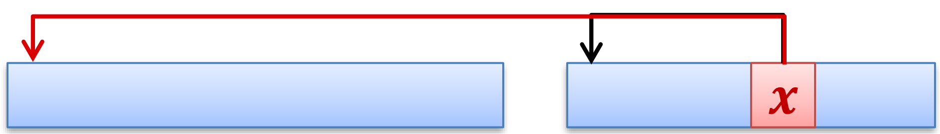
- There are at most $n - 1$ union operations
- Amortized and worst-case cost of make_set, find: $O(1)$
- Amortized cost of union operation: $O(\log n)$

Union-By-Size Heuristic

Theorem: The overall cost of m operations of which at most $u \leq n$ are union operations is $\boldsymbol{O(m + u \cdot \log n)}$.

Proof:

- Total cost of make-set & find operations: $O(m)$
- Total cost of union operations: $O(\#\text{pointer redirections})$
- Consider a fixed element x
- How often do we redirect the pointer of x?



- When redirecting the pointer of x, the size of the set of x at least doubles.
  $\Longrightarrow \leq \log_2 n$ pointer redir. for element x
  - But only if x ends up in a set of size > 1
- **Total union cost:** $O(u \cdot \log n)$

Kruskal Algorithm:

Sorting edges by weight:
$O(m \log n)$

Union-find part:
$O(m + n \log n)$