Chapter 4
Data Structures

Algorithm Theory
WS 2014/15

Fabian Kuhn

Implementation of Kruskal Algorithm

1. Go through edges in order of increasing weights

2. For each edge e:

   if *e* does not close a cycle then

   add *e* to the current solution

Union-Find Data Structure

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

Manages partition of a set of elements

- set of disjoint sets

Operations:

- $\mathbf{make\_set}(\boldsymbol{x})$**:** create a new set that only contains element x

- $\mathbf{find}(\boldsymbol{x})$**:** return the set containing x

- $\mathbf{union}(\boldsymbol{x}, \boldsymbol{y})$**:** 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**

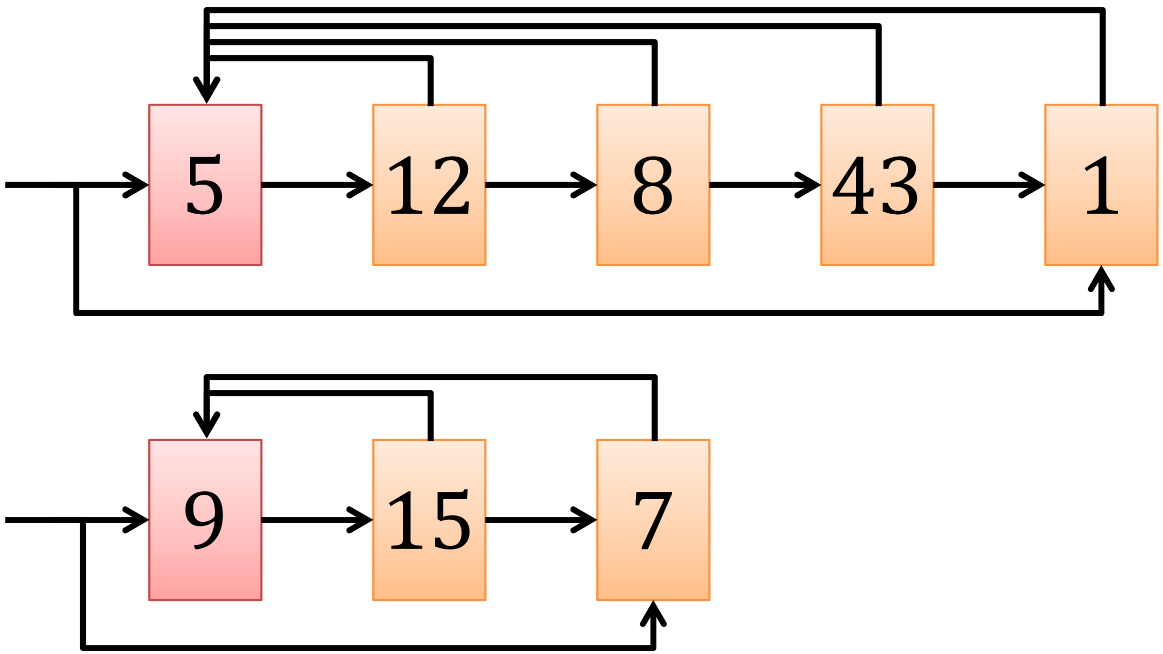
   add e to the current solution

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

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: $5, 9$

Weighted-Union Heuristic

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

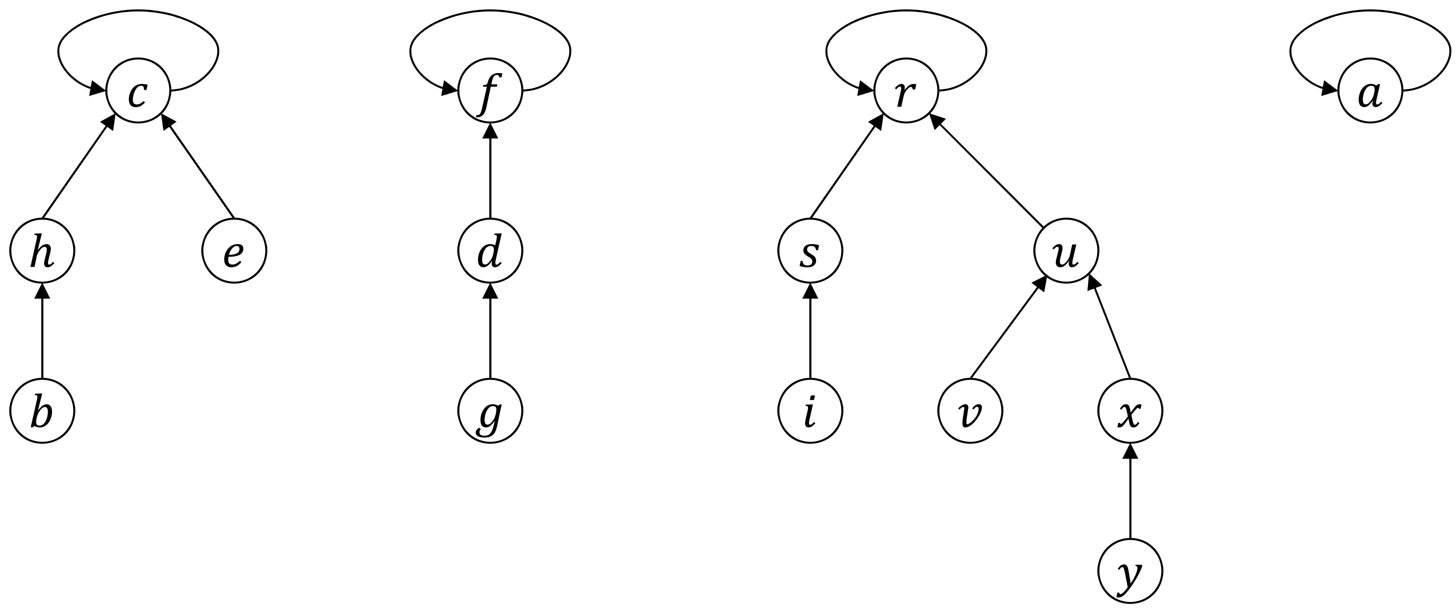
Idea:

- 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 n are make_set operations is $\boldsymbol{O(m + n \log n)}$.

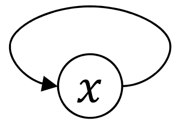
Disjoint-Set Forests



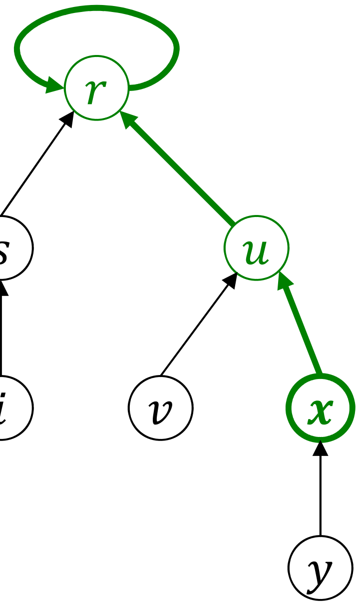
- Represent each set by a tree
- Representative of a set is the root of the tree

Disjoint-Set Forests

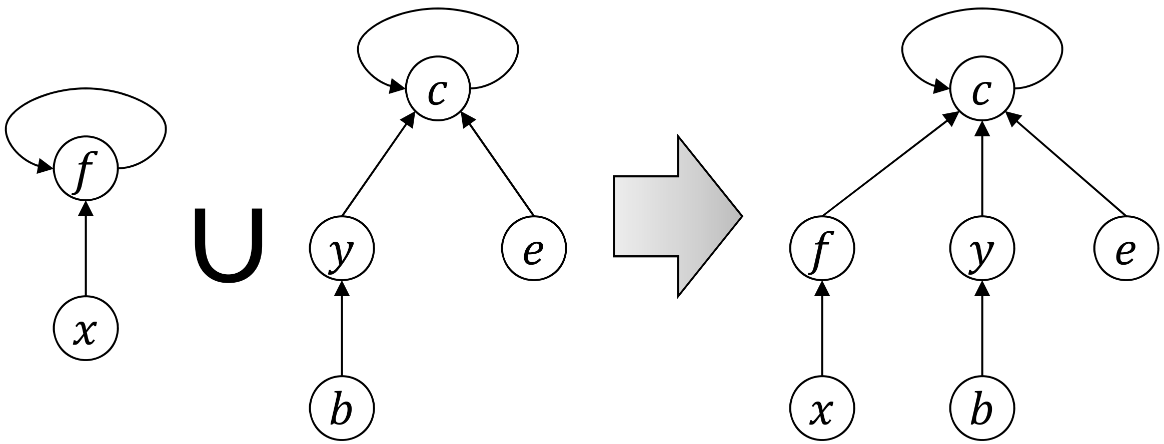
make_set(x): create new one-node tree



find(x): follow parent point to root
(parent pointer to itself)



union(x, y): attach tree of x to tree of y



Bad Sequence

Bad sequence leads to tree(s) of depth $\Theta(n)$

- $\text{make\_set}(x_1), \text{make\_set}(x_2), \dots, \text{make\_set}(x_n)$, $\text{union}(x_1, x_2), \text{union}(x_1, x_3), \dots, \text{union}(x_1, x_n)$

Union-By-Size Heuristic

Union of sets S_1 and S_2:

- Root of trees representing S_1 and S_2: r_1 and r_2
- W.l.o.g., assume that $|S_1| \geq |S_2|$
- Root of $S_1 \cup S_2$: r_1 (r_2 is attached to r_1 as a new child)

Theorem: If the union-by-size heuristic is used, the **worst-case cost of a $\mathbf{find}$-operation is $O(\log n)$**

Proof:

Similar Strategy: union-by-rank

- rank: essentially the depth of a tree

Union-Find Algorithms

Recall: m operations, n of the operations are make_set-operations

Linked List with Weighted Union Heuristic:

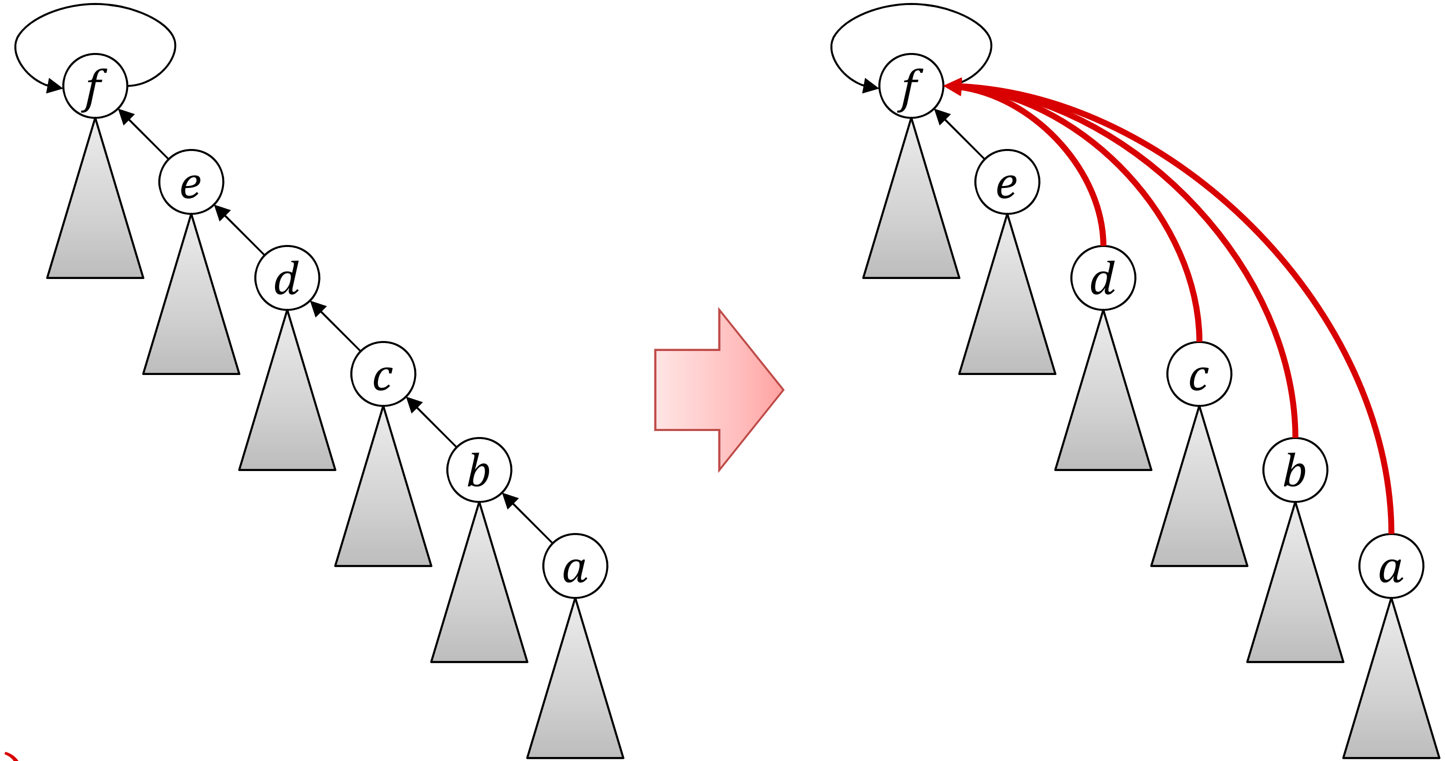
- make_set: worst-case cost $O(1)$
- find : worst-case cost $O(1)$
- union : amortized worst-case cost $O(\log n)$

Disjoint-Set Forest with Union-By-Size Heuristic:

- make_set: worst-case cost $O(1)$
- find : worst-case cost $O(\log n)$
- union : worst-case cost $O(\log n)$

Can we make this faster?

Path Compression During Find Operation



find(a):

1. **if** $a \neq a.parent$ **then**
2. $\quad a.parent \coloneqq \text{find}(a.parent)$
3. **return** $a.parent$

Complexity With Path Compression

When using only path compression (without union-by-rank):

m: total number of operations

- f of which are find-operations
- n of which are make_set-operations
  → at most $n-1$ are union-operations

Total cost: $\mathbf{O}\left(n + f \cdot \left\lceil \log_{2+f/n} n \right\rceil\right) = O\left(m + f \cdot \log_{2+m/n} n\right)$

Union-By-Size and Path Compression

Theorem:

Using the combined union-by-rank and path compression heuristic, the running time of m disjoint-set (union-find) operations on n elements (at most n $\mathrm{make\_set}$-operations) is

$$\boldsymbol{\Theta}\big(\boldsymbol{m} \cdot \boldsymbol{\alpha}(\boldsymbol{m}, \boldsymbol{n})\big),$$

Where $\alpha(m, n)$ is the inverse of the Ackermann function.

Ackermann Function and its Inverse

Ackermann Function:

For $k, \ell \geq 1$,

$$\boldsymbol{A(k,\ell) := \begin{cases} 2^{\ell}, & \text{if } k = 1, \ell \geq 1 \\ A(k-1,2), & \text{if } k > 1, \ell = 1 \\ A\big(k-1, A(k,\ell-1)\big), & \text{if } k > 1, \ell > 1 \end{cases}}$$

Inverse of Ackermann Function:

$$\boldsymbol{\alpha(m,n) := \min\{k \geq 1 \mid A(k, \lfloor m/n \rfloor) > \log_2 n\}}$$

Inverse of Ackermann Function

- $\alpha(m,n) \coloneqq \min\{k \geq 1 \mid A(k, \lfloor {}^{m}/_{n} \rfloor) > \log_2 n\}$

  $m \geq n \Longrightarrow A(k, \lfloor {}^{m}/_{n} \rfloor) \geq A(k,1) \Longrightarrow \alpha(m,n) \leq \min\{k \geq 1 | A(k,1) > \log n\}$

- $A(1,\ell) = 2^{\ell}, \quad A(k,1) = A(k-1,2),$
  $A(k,\ell) = A\big(k-1, A(k,\ell-1)\big)$

- $A(2,1) = A(1,2) = 4$
- $A(3,1) = A(2,2) = A\big(1, A(2,1)\big) = 2^4$
- $A(4,1) = A(3,2) = A\big(2, A(3,1)\big) = A(2,2^4)$

  $= A\big(1, A(2,2^4-1)\big) = 2^{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}} \Big\} \; c+1$ times

- $A(5,1) = \; \ldots$