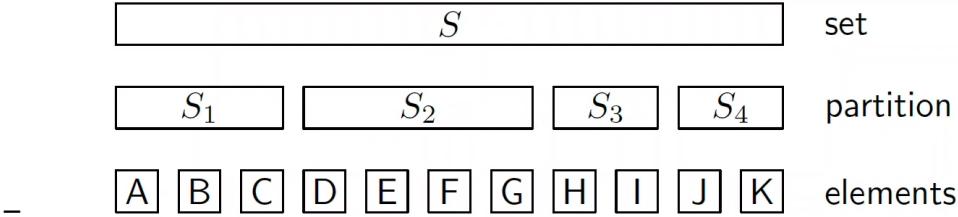


## Definition

- set partitioned into disjunct subsets
  - Set  $S$  of  $n$  elements:  $S = \{s_1, s_2, \dots, s_n\}$
  - $S$  is partitioned into subsets  $S_1, S_2, \dots, S_k$ , that is,  $S = \cup_{i=1, \dots, k} S_i$  and  $S_i \cap S_j = \emptyset$  for  $i \neq j$ .



- needed operations

**Union( $S_i, S_j$ ):** Combine two subsets  $S_i$  and  $S_j$  to a new subset:  $S_{new} = S_i \cup S_j$ .

— **Find( $s_x$ ):** Find the subset  $S_i$  to which  $s_x$  belongs

- How fast can a sequence of  $n - 1$  unions and  $f$  finds (in arbitrarily interleaved order) be performed?

- performance of merging subsets into one big set

- example

**Initial setting:**

- Initially there are  $n$  subsets  $S_i = \{s_i\}$ ,  $i = 1, \dots, n$
- Example:  $S = \{A, B, C, D, E\}$   
 $S_1 = \{A\}$ ,  $S_2 = \{B\}$ ,  $S_3 = \{C\}$ ,  $S_4 = \{D\}$ ,  $S_5 = \{E\}$ ,

FIND(D)= $S_4$

UNION( $S_3, S_4$ ) %New set always gets smaller index

FIND(A)= $S_1$

FIND(D)= $S_3$

UNION( $S_1, S_3$ )

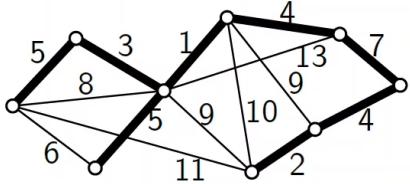
UNION( $S_2, S_5$ )

FIND(D)= $S_1$

FIND(E)= $S_2$      $\Rightarrow$      $S_1 = \{A, C, D\}$ ,  $S_2 = \{B, E\}$

## Motivation: Minimum Spanning Tree

- Let  $G$  be a weighted graph.
- Compute a spanning tree  $\in G$  of minimum weight



More details about minimum spanning trees (algorithms, correctness etc.) in the lecture  
*Design and Analysis of Algorithms*

- Greedy Algorithm: Add short edges (in increasing order), but only if they **don't close a cycle**, until we get a tree.
- We need to check if an edge  $e = p_i, p_j$  closes a cycle!

### How to check if an edge $e = p_i, p_j$ closes a cycle?

- Use UNION-FIND
- Initially each point is in its own set:  $S_i = \{p_i\}$
- When inserting the edge  $e = p_i, p_j$ :  $\text{UNION}(S_{p_i}, S_{p_j})$
- Edge  $e = p_i, p_j$  closes a cycle  $\Leftrightarrow \text{FIND}(p_i) = \text{FIND}(p_j)$
- If  $G$  has  $n$  vertices and  $m$  edges, then we need  $n - 1$  unions and  $O(m)$  finds

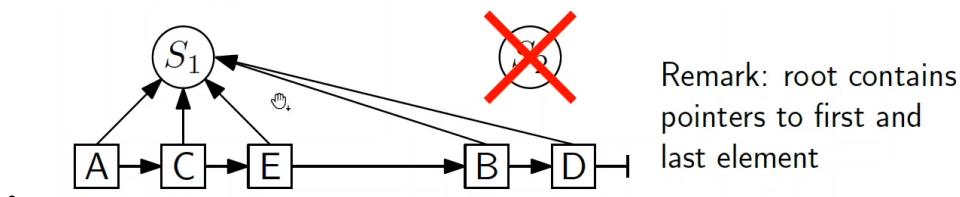
## Fast FIND

### Store each set as a tree:

- Each set  $S_i$  is a tree of height 1
- Leaves of the tree are the elements (in arbitrary order)
- Each leaf points to the root and to the next element



- $\text{FIND}(x)$ : inspect pointer to root in  $\Theta(1)$  time
- $\text{UNION}(S_i, S_j)$ : update pointers of one set and link elements in time  $\Theta(|S_i|)$  or  $\Theta(|S_j|)$



## Analysis UNION operation:

- Assume UNION( $S_i, S_j$ ) updates the pointers of  $S_j$
- The new sets gets index  $i$

Then the sequence UNION( $S_{i-1}, S_i$ ) for  $i = n$  down to 2 needs  $1 + 2 + 3 + \dots + (n - 1) = \Theta(n^2)$  pointer updates

- Then a UNION() operation needs  $\Theta(n)$  time in average.

**Idea: update smaller set** (make  $u$  UNION() operations)

- For a fixed element  $x \in S$  in each UNION() step which 'moves'  $x$  the size of the resulting set at least doubles
- After  $u_x$  UNION() steps of  $x$  we get  $2^{u_x} \leq |S_{u_x}| \leq u$
- $u_x = O(\log u)$ : each element is updated  $O(\log u)$  times
- So  $u$  UNION() operations need  $O(u \log u)$  time

- performance of merging subsets into one big set

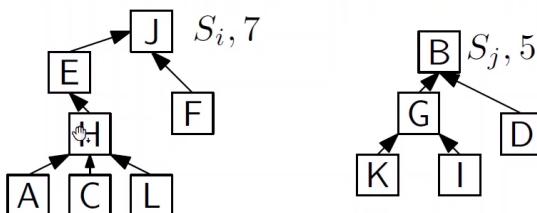
A sequence of  $n - 1$  unions and  $f$  finds can be performed in  $O(n \log n + f)$  time.

Remark: The minimum spanning tree of a graph with  $n$  vertices and  $m$  edges can be computed in  $O(n \log n + m)$  time.

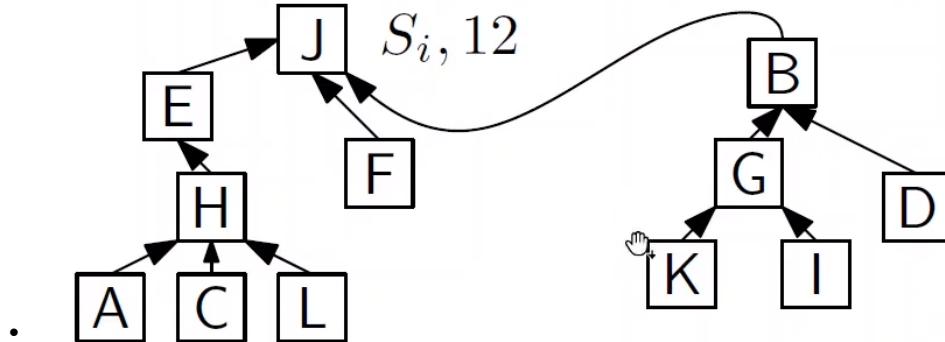
## Fast UNION

**Store each set as a tree:**

- Each set  $S_i$  is a tree of height  $h_i$
- Nodes are the elements with pointers towards the root
- Root contains the id  $S_i$  and the number of nodes



- FIND( $x$ ): follow pointers towards the root, then inspect root. Time needed:  $\Theta(h_i)$
- UNION( $S_i, S_j$ ): connect root of the smaller tree as child to the root of the larger tree, update treesize. Time  $\Theta(1)$



**Analysis of depth of tree:**

- For a (sub)tree  $T$  let  $h_T$  be its height, and  $|T|$  its size
- Claim:  $|T| \geq 2^{h_T}$
- Proof by induction over  $h_T$
- Induction base:  $h_T = 0$  (single node),  $|T| = 1 \geq 2^0 = 1$
- Induction step:  $h_T > 0$
- Let  $s$  be the son with highest subtree  $T_s$ ,  $h_{T_s} = h_T - 1$
- By induction we have  $|T_s| \geq 2^{h_{T_s}} = 2^{h_T - 1}$
- Consider UNION() when  $s$  got a son of the root of  $T \setminus T_s$ : by the 'smaller' rule we know  $|T \setminus T_s| \geq |T_s|$
- Thus  $|T| \geq 2|T_s| = 2 \times 2^{h_T - 1} = 2^{h_T}$ , q.e.d.
- From  $|T| \geq 2^{h_T}$  we have  $h_T = \log_2 |T| = O(\log n)$
- FIND() needs only  $O(\log n)$  time
- performance of merging subsets into one big set

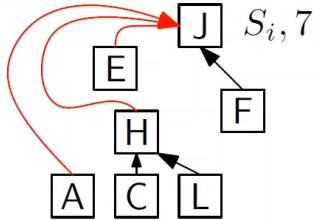
A sequence of  $n - 1$  unions and  $f$  finds can be performed in  $O(n + f \log n)$  time.

Remark: For few finds ( $f = o(n)$ ) this is faster than the previous approach

## Fast UNION and FIND

### Idea: shorten path to root

- For each FIND() of the previous approach link the visited nodes directly to the root



- First search for the root  $r$  as usual
- Then search a second time and relink all pointers to  $r$
- Further FIND() operations get potentially much faster
- performance of merging subsets into one big set
  - A sequence of  $n - 1$  unions and  $f \geq n$  finds can be performed in  $O(f \cdot \alpha(f, n))$  time.
  - Lemma without proof
  - $\alpha(f, n)$  is the inverse of the Ackermann-function  $A(i, n)$
  - $A(1, n) = 2^n$
  - $A(i, 1) = A(i - 1, 2)$
  - $A(i, n) = A(i - 1, A(i, n - 1))$  for  $i, n \geq 2$

$$\text{Example: } A(2, n) = A(1, A(2, n - 1)) = 2^{A(2, n - 1)} = \\ 2^{2^{A(2, n - 2)}} = \dots = 2^{2^{2^{\dots^2}}} \quad (\text{n times power of 2})$$

**Reasonable input:**  $\alpha(f, n) < 4$

- $\alpha(f, n)$  grows more than moderate:
- For  $f \geq 3$  we have  $\alpha(f, n) < \alpha(2, n) = \log_2^* n - 1$
- $\log_2^* n = \min\{i \in \mathbb{N} \mid \log_2^{(i)} n \leq 1\}$  (iterative logarithm)
- $\log_2^{(i)} n = \log_2(\log_2(\log_2(\dots \log_2 n)))$   $i$  times
- Example:  $\log_2^*(65536) = 1 + \log_2^*(16) = 2 + \log_2^*(4) = 3 + \log_2^*(2) = 4$
- For  $n \leq 2^{65536} \approx 10^{19728}$  we have  $\log_2^* n \leq 5$ .
- So for any reasonable input  $\log_2^* n \leq 5$
- Thus  $O(f\alpha(f, n))$  is linear in  $f$  for all practical problems
- A lower bound of  $\Omega(f\alpha(f, n))$  can be shown

## Runtimes

**Lemma (Fast FIND):** A sequence of  $n - 1$  unions and  $f$  finds can be performed in  $O(n \log n + f)$  time.

**Lemma (Fast UNION):** A sequence of  $n - 1$  unions and  $f$  finds can be performed in  $O(n + f \log n)$  time.

**Lemma (Almost Linear):** A sequence of  $n - 1$  unions and  $f \geq n$  finds can be performed in  $O(f \cdot \alpha(f, n))$  time.