DISJOINT SET DATASTRUCTURE

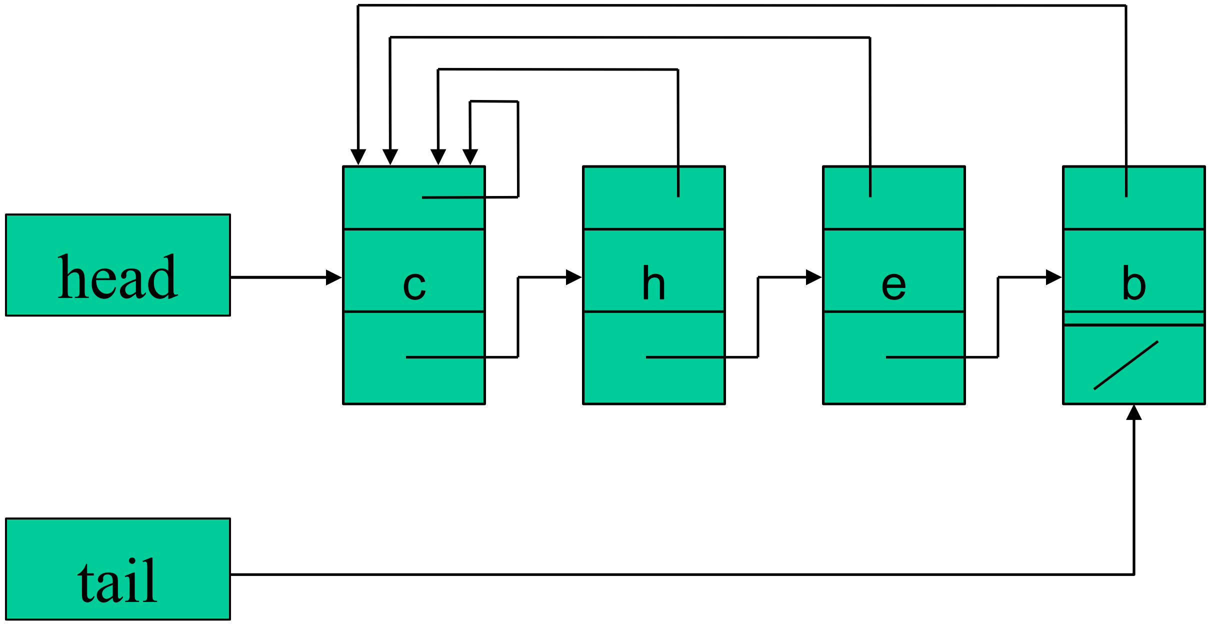
PART II

Linked List Implementation of Disjoint Set

S={s1,s2…}

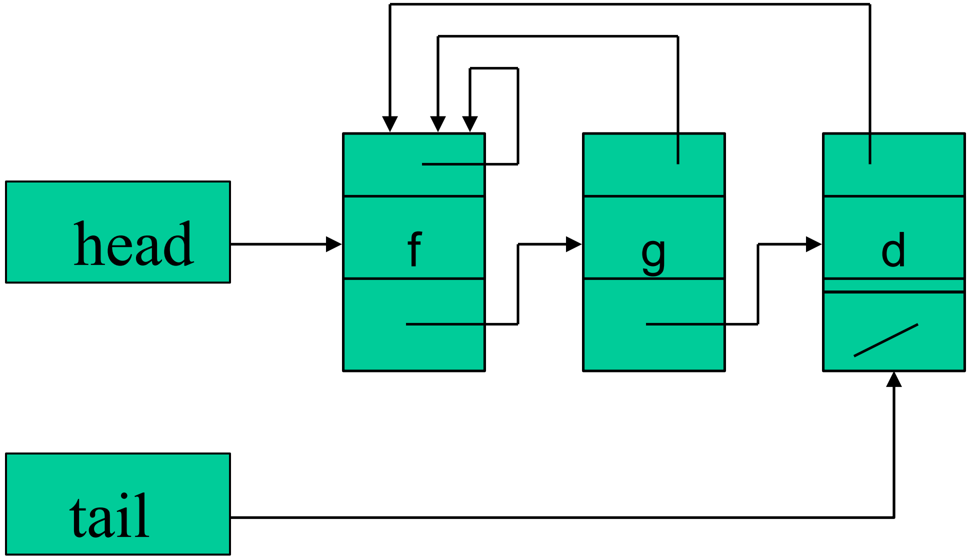
- Each set is represented as a linked list.
- Each node of a list contains:
  - the object
  - a pointer to the next item in the list
  - a pointer back to the representative for the set

Linked List Implementation of Disjoint Set



x = {c, h, e, b}

Linked List Implementation of Disjoint Set



y = {f, g, d}

Implementation of Operations

MAKE-SET (x):

- Create new linked list whose only object is *x*.

FIND-SET (x):

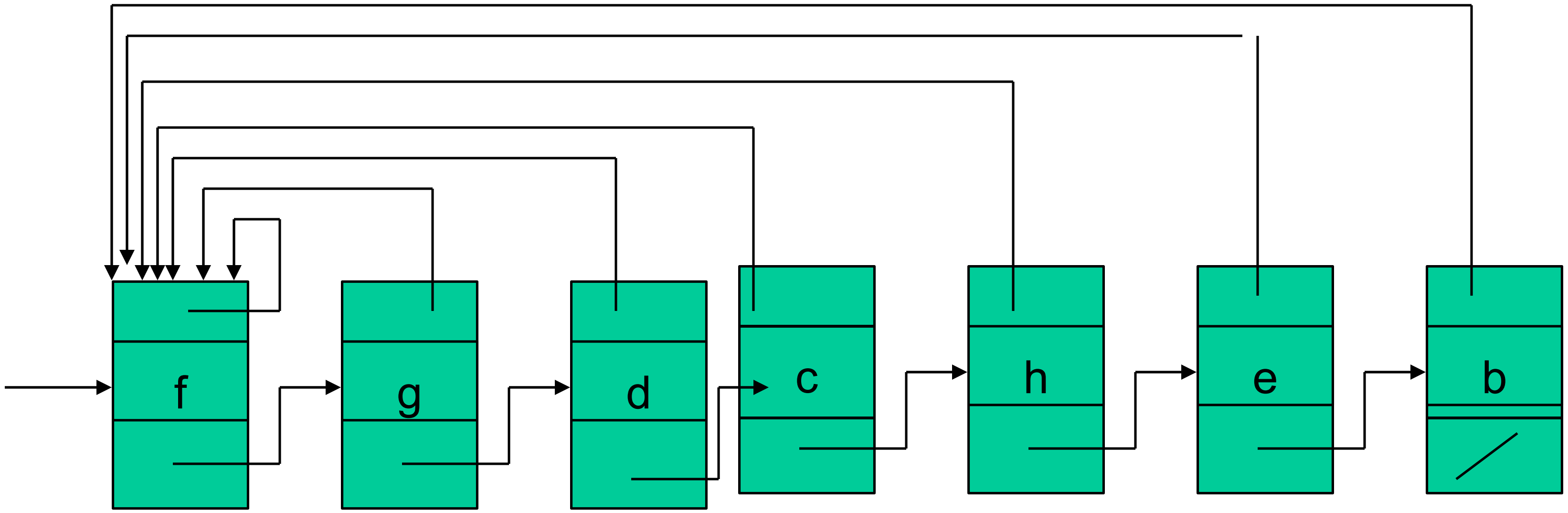
- Return the pointer from *x* back to the representative.

Implementation of Operations

UNION (x, y):

- *Append x's list to y's list using the tail pointer for y's* list. Update the representative pointer for each object that was in *x*'s list.
  - **Weighted-union heuristic:** Store length of list in each list so we can be sure to append the shorter list to the end of the longer list.

Union of the Two Sets x and y

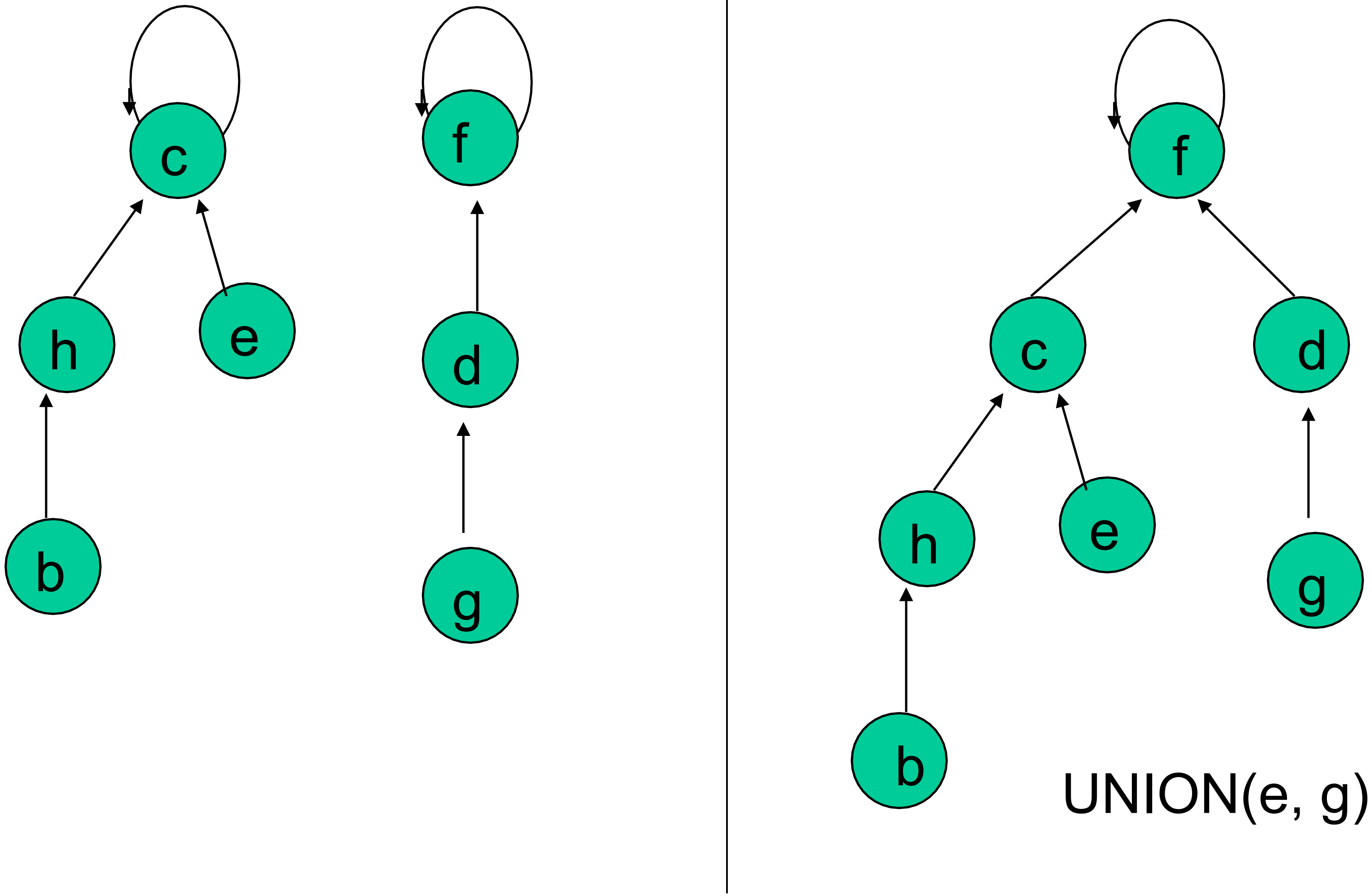


x ∪ y = {c, h, e, b, f, g, d}

Tree Implementation of Disjoint Set

- **Disjoint-set forests:**
  - Each tree represents one set.
  - Each node contains one member.
  - Each member points only to its parent.
  - Each root contains the representative and is its own parent.

Example of Disjoint-Set Forest



UNION(e, g)

Implementation of Operations

MAKE-SET (x):

- Create new tree whose only object is *x*.

FIND-SET (x):

- Follow parent pointers from *x* to the root; return pointer to the root.

UNION (x, y):

- Make root of one tree point to root of the other.