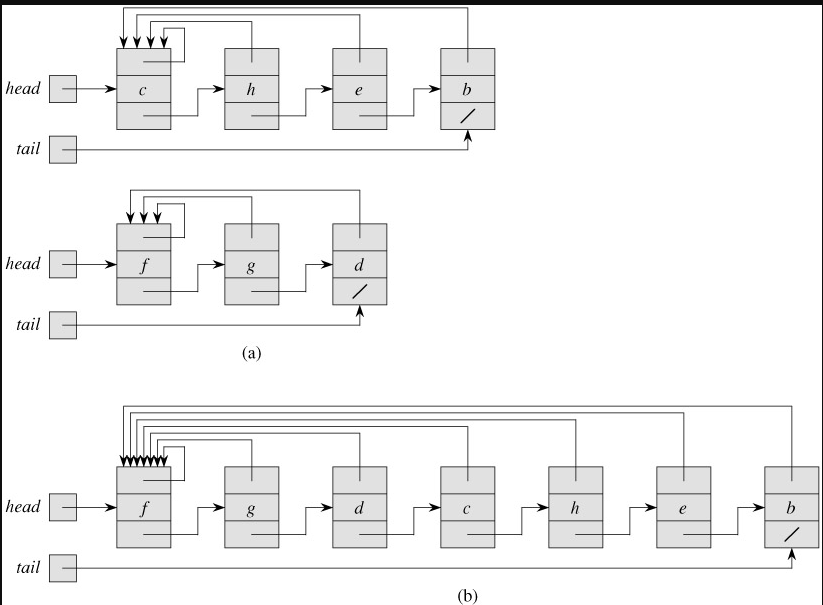
1)How do we represent disjoint set using Linked List?

A simple way to implement a disjoint-set data structure is to represent each set by a linked list. The first object in each linked list serves as its set's representative. Each object in the linked list contains a set member, a pointer to the object containing the next set member, and a pointer back to the representative. Each list maintains pointers *head*, to the representative, and *tail*, to the last object in the list. [Figure 21.2(a)](http://www.euroinformatica.ro/documentation/programming/!!!Algorithms_CORMEN!!!/DDU0124.html#ch21fig02) shows two sets. Within each linked list, the objects may appear in any order (subject to our assumption that the first object in each list is the representative).



**Figure 21.2:***(a)* Linked-list representations of two sets. One contains objects *b*, *c*, *e*, and *h*, with *c* as the representative, and the other contains objects *d*, *f*, and *g*, with *f* as the representative. Each object on the list contains a set member, a pointer to the next object on the list, and a pointer back to the first object on the list, which is the representative. Each list has pointers *head* and *tail* to the first and last objects, respectively. *(b)* The result of UNION(*e*, *g*). The representative of the resulting set is *f*.

With this linked-list representation, both MAKE-SET and FIND-SET are easy, requiring *O*(1) time. To carry out MAKE-SET(*x*), we create a new linked list whose only object is *x*. For FIND-SET(*x*), we just return the pointer from *x* back to the representative

2)How can we union two disjoint sets according to their rank?

The  *union()*and *find()*in naïve way , the worst case time complexity is linear.

The operations can be optimized to *O(Log n)* in worst case. The idea is to always attach smaller depth tree under the root of the deeper tree. This technique is called ***union by rank***.

Let us see the above example with union by rank

Initially, all elements are single element subsets.

0 1 2 3

Do Union(0, 1)

1 2 3

/

0

Do Union(1, 2)

1 3

/ \

0 2

Do Union(2, 3)

1

/ | \

0 2 3

And here is the implementation of union by rank based on the depth of the trees:

void make\_set(int v) {

parent[v] = v;

rank[v] = 0;

}

void union\_sets(int a, int b) {

a = find\_set(a);

b = find\_set(b);

if (a != b) {

if (rank[a] < rank[b])

swap(a, b);

parent[b] = a;

if (rank[a] == rank[b])

rank[a]++;

}

}