Disjoint Sets Data Structure

Disjoint Sets Data Structure

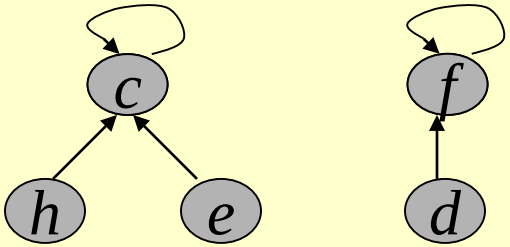
- A disjoint set data structure, also called a union find data structure, is a data structure that stores a collection of disjoint dynamic sets

  $S=\{S_1, S_2,\ldots, S_k\}$

- Each set is identified by a member of the set, called *representative*.
- Disjoint set operations:
  - MAKE-SET(x): create a new set with only x. assume x is not already in some other set.
  - UNION(x,y): combine the two sets containing x and y into one new set. A new representative is selected.
  - FIND-SET(x): return the representative of the set containing x.

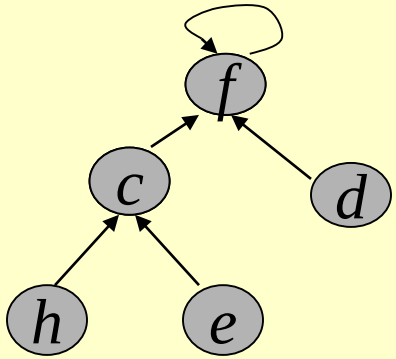
Disjoint-set Implementation: Forests

- Rooted trees, each tree is a set, root is the representative.
- Each node points to its parent. Root points to itself.



Set {*c*,*h*,*e*}

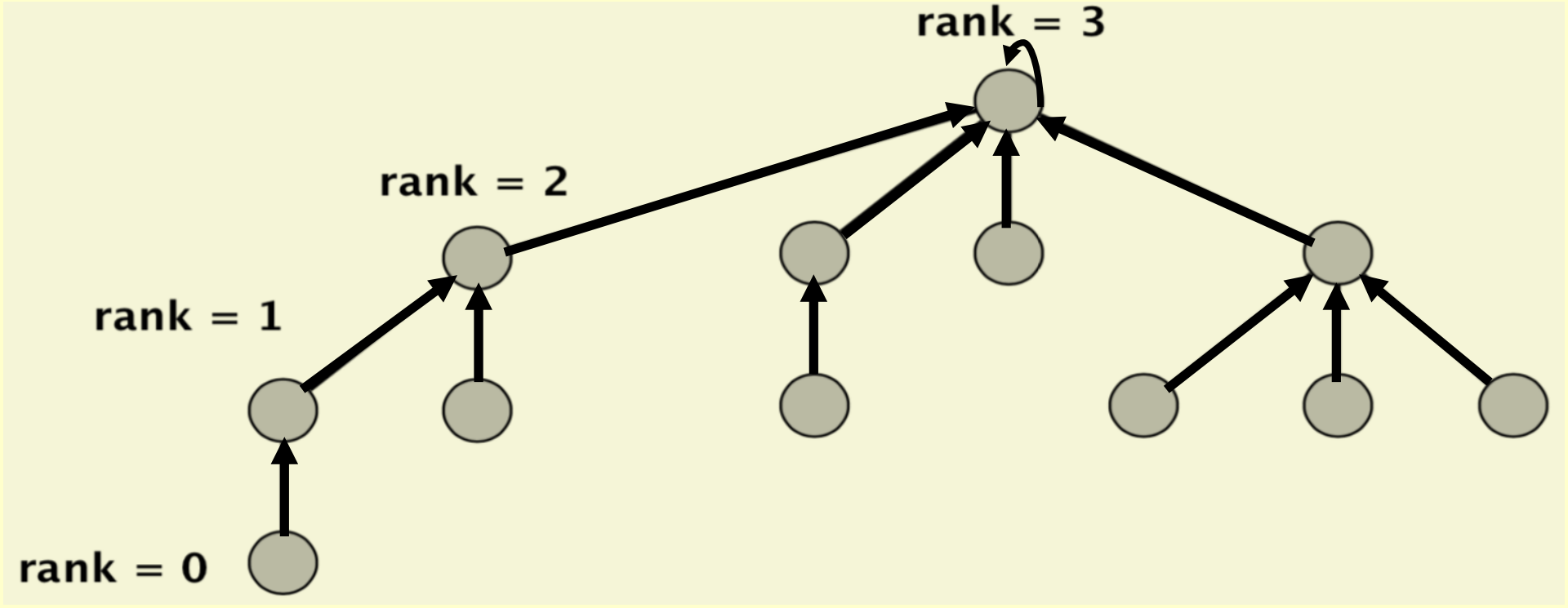
Set {*f*,*d*}



UNION

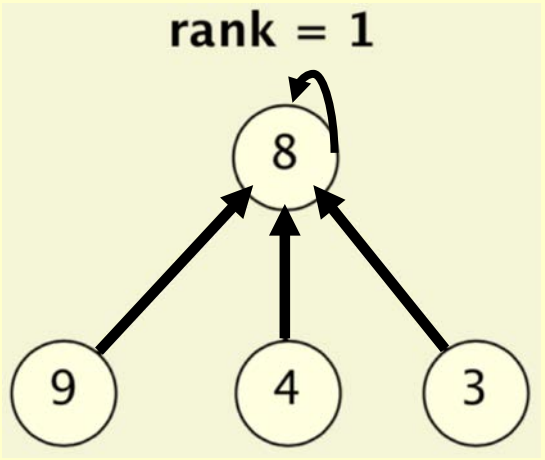
Heuristics for disjoint set operations

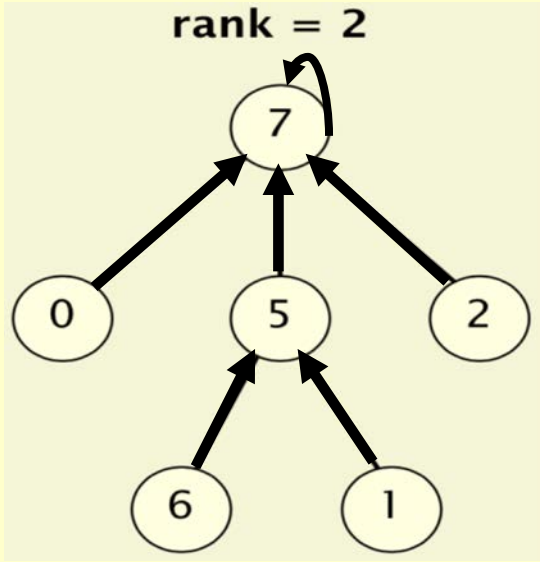
- Union by Rank: Each node is associated with a rank, which is the upper bound on the height of the node (i.e., the height of subtree rooted at the node), in UNION, let the root with smaller rank point to the root with larger rank.

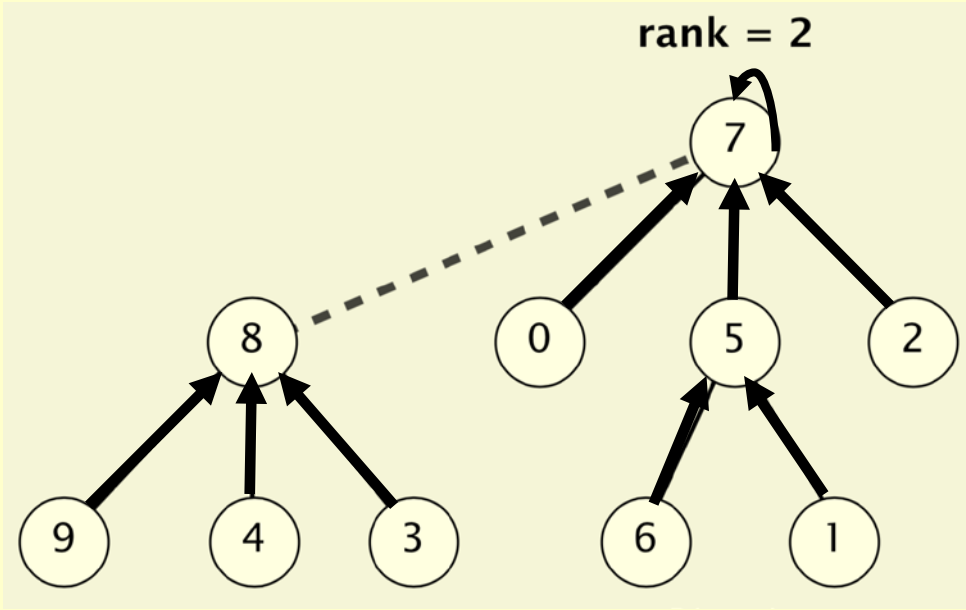


Heuristics for disjoint set operations

- Union by Rank: Each node is associated with a rank, which is the upper bound on the height of the node (i.e., the height of subtree rooted at the node), in UNION, let the root with smaller rank point to the root with larger rank.

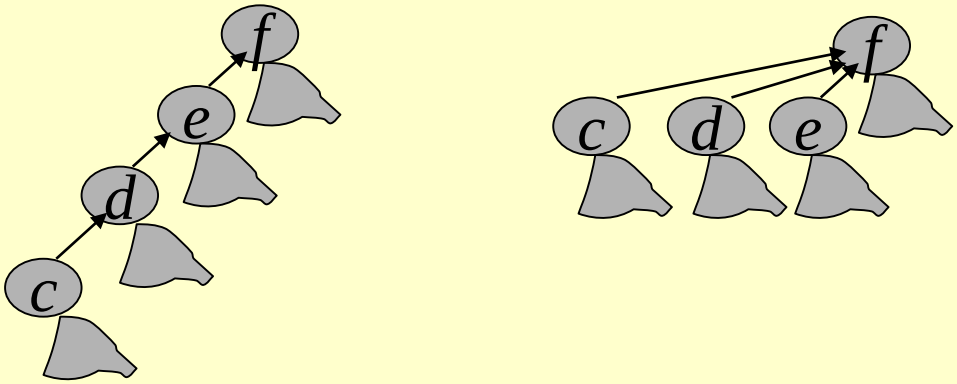


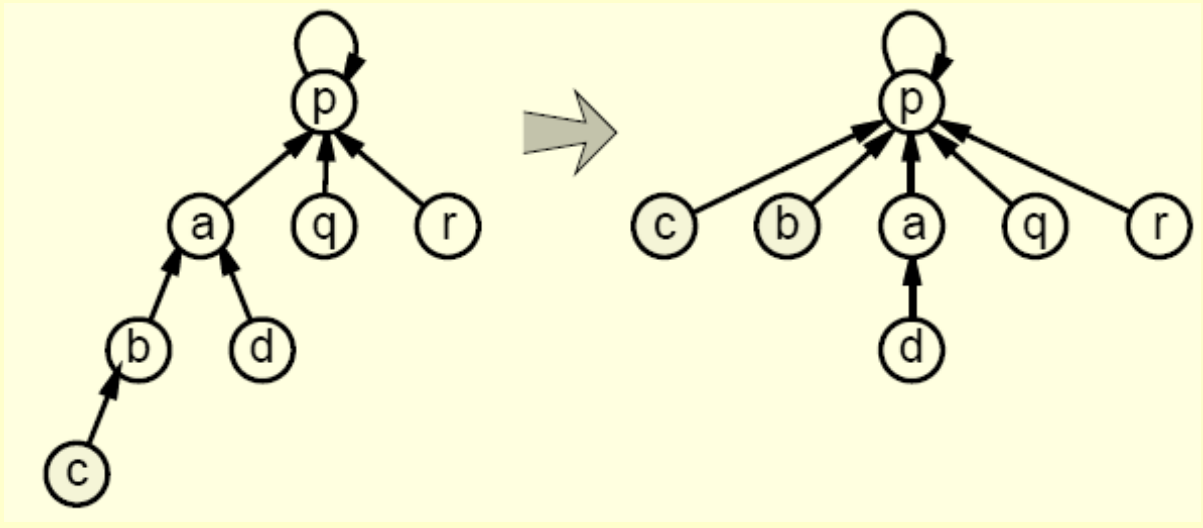




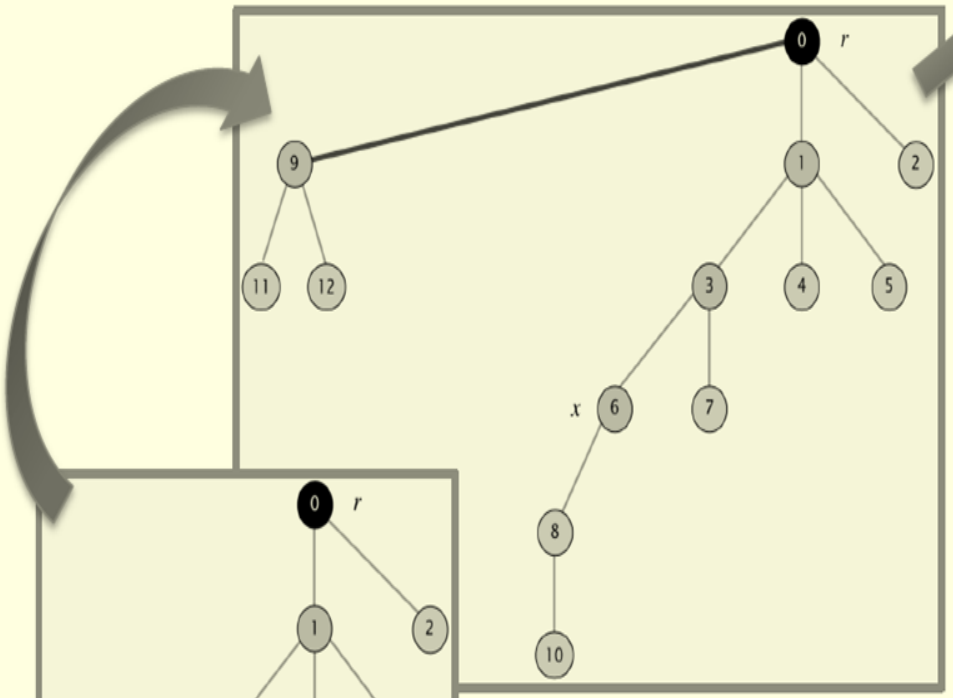
Heuristics for disjoint set operations

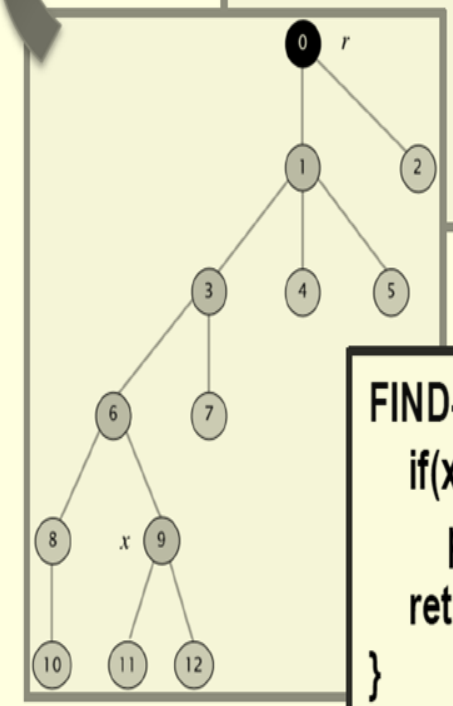
- Path Compression: used in FIND-SET(x) operation, make each node in the path from x to the root directly point to the root. Thus reduce the tree height.



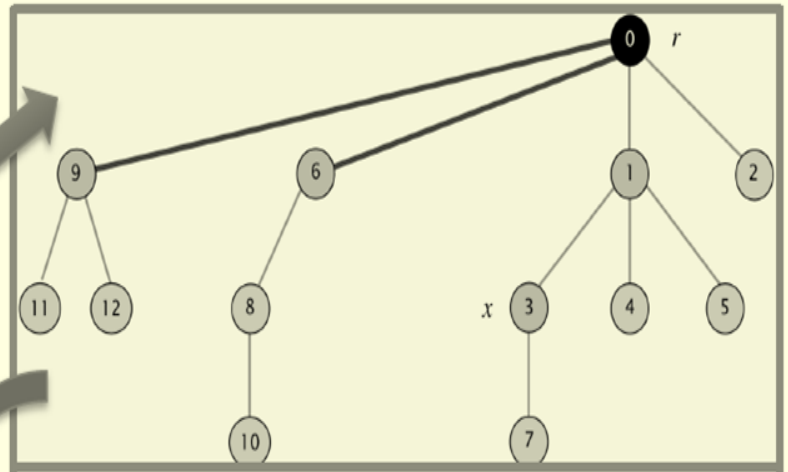


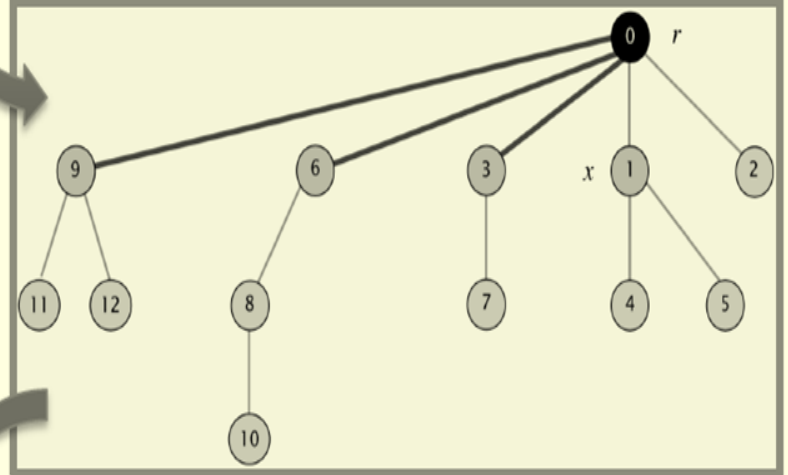
Path Compression: Example

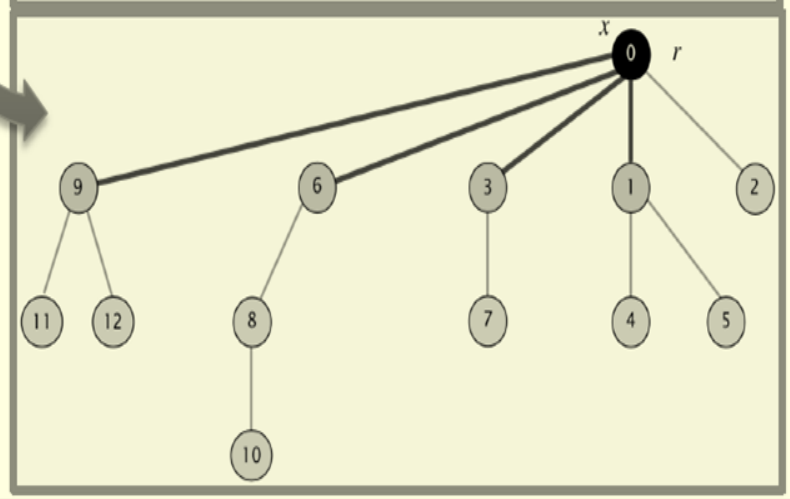





```
FIND-SET(x) {
    if(x is not parent)
        parent[x] ← FIND-SET(parent[x]);
    return x;
}
```







Performance Parameters

- n: number of MAKE-SET operations (executed at beginning).
- m: number of MAKE-SET, UNION, FIND-SET operations.
- $m \geq n$
- Number of UNION operation is at most n-1.
- n essentially gives the number of data items to be stored in the disjoint set data structure.

Algorithm for Disjoint-Set Forest

```
MAKE-SET(x)
1.  p[x]←x
2.  rank[x]←0
```

```
UNION(x', y')
1.  x ← FIND-SET (x')
2.  y ←FIND-SET (y')
3.  if rank[x]>rank[y]
4.  then p[y] ←x
5.  else  p[x] ←y
6.        if rank[x]=rank[y]
7.        then rank[y]++
```

```
FIND-SET(x)
1.  if x≠ p[x]
2.     then p[x] ←FIND-SET(p[x])
3.  return p[x]
```

Worst case running time for MAKE-SET is $O(1)$, UNION and FIND-SET operations is: $O(lg\ n)$.

Application of Disjoint-Set

- One of the many applications of disjoint set data structures arises in determining the connected components of an undirected graph
- Used to determine whether two vertices belong to the same component, or whether adding an edge between them would result in a cycle
- It is also a key component in implementing Kruskal's algorithm to find the minimum spanning tree of a graph

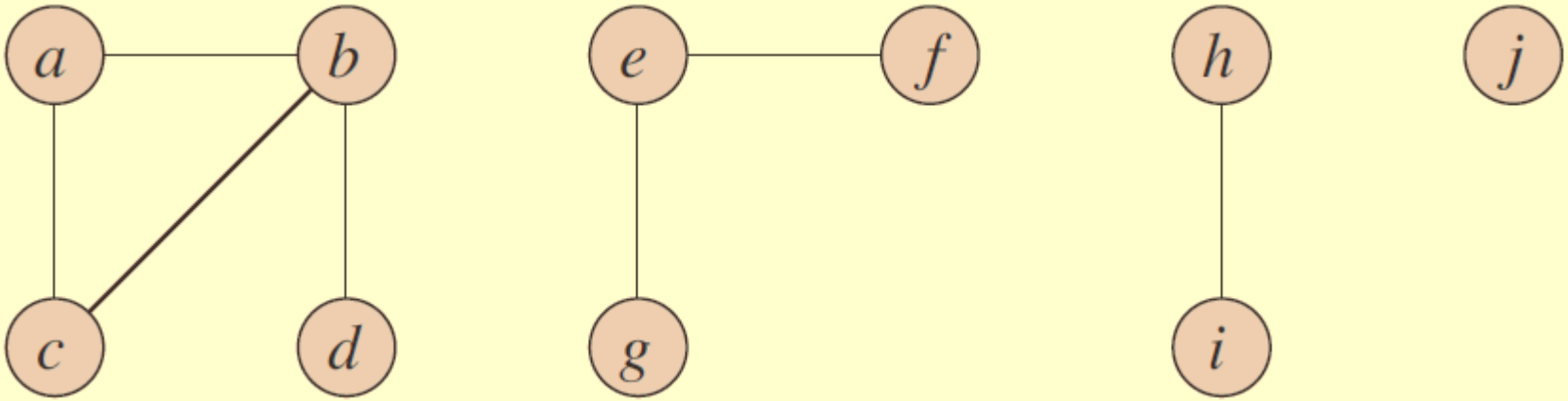
An Application of Disjoint-Set

- Determine the connected components of an undirected graph.

```
CONNECTED-COMPONENTS(G)
1.  for each vertex v ∈ V[G]
2.      do MAKE-SET(v)
3.  for each edge (u,v) ∈ E[G]
4.      do if FIND-SET(u) ≠ FIND-SET(v)
5.          then UNION(u,v)
```

```
SAME-COMPONENT(u,v)
1.  if FIND-SET(u)=FIND-SET(v)
2.      then return TRUE
3.      else return FALSE
```

An Application of Disjoint-Set



1.{a ,b ,c ,d} 2.{e ,f , g} 3.{h ,i } 4.{j}

An Application of Disjoint-Set

Edge processed	Collection of disjoint sets
initial sets	{a} {b} {c} {d} {e} {f} {g} {h} {i} {j}
(b, d)	{a} { b, d} {c} {e} {f} {g} {h} {i} {j}
(e , g)	{a} { b, d} {c} { e, g} {f} {h} {i} {j}
(a , c)	{ a, c} { b ,d} { e, g} {f} {h} {i} {j}
(h , i)	{a, c} { b ,d } { e ,g } {f} {h ,i} {j}
(a, b)	{a ,b ,c ,d },{e , g} ,{f} {h ,i} {j}
(e , f)	{a ,b , c, d} {e ,f , g} { h, i} {j}
(b, c)	{a ,b ,c , d} {e ,f , g} { h , i} {j}

The collection of disjoint sets after each edge is processed