Disjoint-Set data structures

disjoint | path compression | set | union by rank

Original Article:

- Disjoint-set Data Structures (by vlad_D)

If you need to store disjoint sets dynamically the disjoint-set data structure or the union-find data structure is extremely well suited to this task.

Some operations

- CREATE-SET(x) – creates a new set with one element {x}.
- MERGE-SETS(x, y) – merge into one set the set that contains element x and the set that contains element y (x and y are in different sets). The original sets will be destroyed.
- FIND-SET(x) – returns the representative or a pointer to the representative of the set that contains element x.

Implementation with linked lists

Maintain multiple linked lists, each representing one disjoint set. For the merge operation, merge one linked list to the other. We need to maintain a representative of each element in a step, here in merge the representatives of all elements in one set need to be changed.
Time Complexity O(M^2) where M is the number of operations MERGE-SETS(x, y). With this implementation the complexity will average O(N) per operation where N represents the number of elements in all sets.

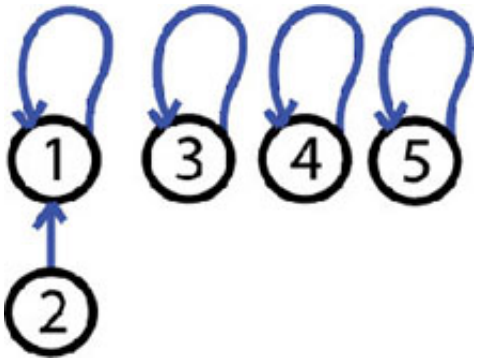
Weighted-union heuristic

In the merge step append the list having less number of elements to the list having more number of elements. In case of equality decide arbitrarily.
This will bring the complexity of the algorithm to O(M + NlogN)

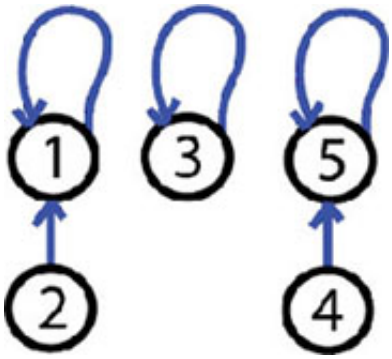
Implementation with rooted trees

Step 1. Initialisation

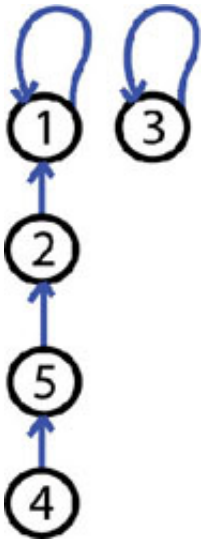
Step 2. 1 and 2 are friends, MERGE-SETS(1, 2):



Step 3. 5 and 4 are friends, MERGE-SETS(5, 4)



Step 4. 5 and 1 are friends, MERGE-SETS(5, 1)



Union by rank

We maintain a quantity called rank that "sort of" represents the height of the tree. Whenever the merge is called we point the root of the tree having lower rank to the root of the tree having higher rank. If the rank of both are equal then the value of rank is incremented by 1. Initially the rank is set to 0.

Path Compression

The idea of path compression is that during the FIND-SET(x) operation we connect all nodes lying in the path of the given node x, directly to the root of the tree. This flattens the tree.

```
1 CREATE-SET(x){
2   P[x] = x;
```

```
  rank[x] = 0;
}

FIND-SET(x){
  if P[x] = x:
    return x;
  P[x] = FIND-SET(P[x]);
  return P[x];
}

MERGE-SET(x, y){
  PX = FIND-SET(x);
  PY = FIND-SET(y);
  if(PX == PY)
    return;
  if(rank[PX] > rank[PY])
    P[PY] = PX;
  else if(rank[PY] > rank[PX])
    P[PX] = PY;
  else{
    P[PY] = PX;
    rank[PX] = rank[PX] + 1;
  }
}
```

The worst running time is O(m α(m,n)), where α(m,n) is the very slowly growing inverse of Ackermann's function. In application α(m,n) <= 4 that's why we can say that the running time is linear in terms of m, in practical situations.