WIKIPEDIA

Tarjan's off-line lowest common ancestors algorithm

In computer science, **Tarjan's off-line lowest common ancestors algorithm** is an algorithm for computing lowest common ancestors for pairs of nodes in a tree, based on the union-find data structure. The lowest common ancestor of two nodes *d* and *e* in a rooted tree *T* is the node *g* that is an ancestor of both *d* and *e* and that has the greatest depth in *T*. It is named after Robert Tarjan, who discovered the technique in 1979. Tarjan's algorithm is an offline algorithm; that is, unlike other lowest common ancestor algorithms, it requires that all pairs of nodes for which the lowest common ancestor is desired must be specified in advance. The simplest version of the algorithm uses the union-find data structure, which unlike other lowest common ancestor data structures can take more than constant time per operation when the number of pairs of nodes is similar in magnitude to the number of nodes. A later refinement by Gabow & Tarjan (1983) speeds the algorithm up to linear time.

Pseudocode

The pseudocode below determines the lowest common ancestor of each pair in *P*, given the root *r* of a tree in which the children of node *n* are in the set *n.children*. For this offline algorithm, the set *P* must be specified in advance. It uses the *MakeSet*, *Find*, and *Union* functions of a disjoint-set forest. *MakeSet(u)* removes *u* to a singleton set, *Find(u)* returns the standard representative of the set containing *u*, and *Union(u,v)* merges the set containing *u* with the set containing *v*. TarjanOLCA(*r*) is first called on the root *r*.

```
function TarjanOLCA(u) is
    MakeSet(u)
    u.ancestor  := u
    for each v in u.children do
        TarjanOLCA(v)
        Union(u, v)
        Find(u).ancestor  := u
    u.color  := black
    for each v such that {u, v} in P do
        if v.color == black then
            print "Tarjan's Lowest Common Ancestor of " + u +
                  " and " + v + " is " + Find(v).ancestor + "."
```

Each node is initially white, and is colored black after it and all its children have been visited.

For each node pair *{u,v}* to be investigated:

- **When v is already black** (viz. when v comes before u in a post-order traversal of the tree): After u is colored black, the lowest common ancestor of this pair is available as Find(v).ancestor, but only while the LCA of u and v is not colored black.
- **Otherwise:** Once v is colored black, the LCA will be available as Find(u).ancestor, while the LCA is not colored black.

For reference, here are optimized versions of *MakeSet*, *Find*, and *Union* for a disjoint-set forest:

```
function MakeSet(x) is
    x.parent  := x
    x.rank    := 1
```

```
function Union(x, y) is
    xRoot  := Find(x)
    yRoot  := Find(y)
    if xRoot.rank > yRoot.rank then
        yRoot.parent  := xRoot
    else if xRoot.rank < yRoot.rank then
        xRoot.parent  := yRoot
    else if xRoot.rank == yRoot.rank then
        yRoot.parent  := xRoot
        xRoot.rank  := xRoot.rank + 1

function Find(x) is
    if x.parent  != x then
       x.parent  := Find(x.parent)
    return x.parent
```

References

- Gabow, H. N.; Tarjan, R. E. (1983), "A linear-time algorithm for a special case of disjoint set union", Proceedings of the 15th ACM Symposium on Theory of Computing (STOC), pp. 246–251, doi:10.1145/800061.808753 (https://doi.org/10.1145%2F800061.808753).
- Tarjan, R. E. (1979), "Applications of path compression on balanced trees", Journal of the ACM, **26** (4): 690–715, doi:10.1145/322154.322161 (https://doi.org/10.1145%2F322154.322161).