Problem name: Fuleco and Brazil ant

Language: English

Solution

Fuleco the Armadillo[1] looks like a graph theory problem, but really this is ad-hoc problem. Complexity of the solution is linear and optimal memory complexity is constant (we do not need to store the whole ant-traversal in order to process it). Here we will present couple of possible approaches.

Tree reconstruction

This was probably the most straightforward idea for this problem. Tree can be reconstructed as: up always creates a new node which is added as last child of a current node; down changes current node to be parent of current one. In this way we have a tree and the problem becomes: in a given tree find distance between given two nodes. This can be implemented with any graph traversal algorithm (BFS or DFS for example).

Path length change depending on U/D count

We can keep track of number of ups, in other words height from start node. Up always increases the distance and height, but for down depends and this is something that we should handle by two cases. If height is positive then it decreases both distance and height, but as soon as height becomes zero than down also increases the distance.

Maybe this explanation sounds difficult but it is really very intuitive if you try to think about possible paths between nodes in a given tree. Complexity is linear with O(1) additional memory.

Lengths to the LCA

In this kind of tree, the most efficient path will always be to descend to the nearest common ancestor (may take 0 steps), then ascend to the target. Due to the nature of the ant traversal, the ant will NEVER descend deeper than the nearest common ancestor before reaching the target.

So to find the length, it is only necessary to find the greatest depth (relative to the starting position) to which the ant descends, and then to add the length of the downward portion of the journey to the

[1] Fuleco the Armadillo is the official mascot of the 2014 FIFA World Cup in Brazil. Representing a Brazilian three-banded armadillo, a species of armadillo which is native to Brazil and categorized as a vulnerable species on the IUCN Red List, Fuleco was officially launched as part of Brazilian broadcaster TV Globo's weekly Fantástico entertainment show on 25 November 2012. His name is a portmanteau of the words Futebol ("football") and Ecologia ("Ecology"). The mascot, with his message of environmental concerns, ecology and sport turned out to be very popular with football teams around the world.

length of the upward portion of the journey. Time complexity of this solution is linear, with only O(1) additional memory.

```
int NodeDistance(char[] antTraversal, int nodeIndexA, int nodeIndexB)
{
    int minDepth = 0;
    int curDepth = 0;
    for (int i = nodeIndexA; i < nodeIndexB; i++)
    {
        if (antTraversal[i] == 'U')
        {
            curDepth++;
        }
        else if (antTraversal[i] == 'D')
        {
            curDepth--;
        }
        if (curDepth < minDepth)
        {
            minDepth = curDepth;
        }
    }
    return -(minDepth) + (curDepth - minDepth);

}
```

Pseudo-code for Lengths to the UD parts

Removing UD part

Solution can be described as: traverse from first given node to the second one, and eliminate instances of UD (which the ant wouldn't walk in the minimum distance). This includes instances of UD that are formed after removing a previous UD (such as UUDD).

We can look at these ups and downs and open and closed brackets – so the given string is regular array of brackets. When we remove UD we are removing "()" and recursion leads to removing the regular bracket subarrays. Nice thing is that substring for each subtree is regular bracket subarray. So at the end we only have non regular parts, which is exactly the node distance.

In order to implement this in linear time we need stack. We can use original char array for this or have additional memory for this. Of course, first approach destroys the given string.

```
static int NodeDistance(char[] antTraversal, int nodeIndexA, int nodeIndexB)
{
    int topOfStack = nodeIndexA - 1;
    int stackSize = 0;

    for (int i = nodeIndexA; i < nodeIndexB; ++i)
    {
        if ((antTraversal[i] == 'D') && (stackSize > 0) && (antTraversal[topOfStack] == 'U'))
        {
            stackSize--;
            topOfStack--;
        }
        else
        {
            stackSize++;
            topOfStack++;

            if (i != topOfStack)
            {
                antTraversal[topOfStack] = antTraversal[i];
            }
        }
    }

    return stackSize;
```

```
}
```

Pseudo-code for removing UD parts