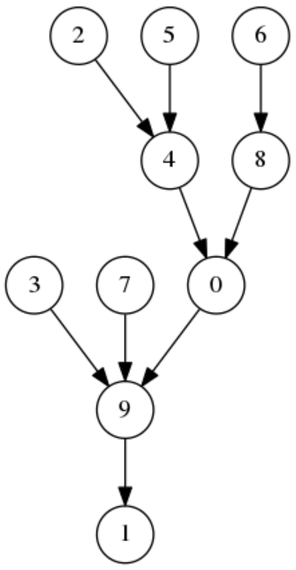
Min Router Peripherality

It's time to show you how the Codility challenge code-named λ (Lambda), can be solved. You can still give it a try, although no certificate will be granted.

In this task you are given a tree: a graph comprising N nodes (called *routers*) and $N-1$ edges (called *links*). It is more convenient to consider the given tree as a rooted tree: that is, with one chosen router (called the *root*) and all the links suspended downwards from that root. Any choice of root is equally valid; it is just a matter of the order in which the routers are processed. Actually, the tree is already presented in the form of a rooted tree. We are given array T such that $T(K)$ is linked with K, unless $T(K) = K$ (for $V = 0, 1, \ldots, N-1$).

There have to be $N-1$ links; otherwise, two distinct routers will either not be connected at all, or will have more than one (possibly indirect) connection between them. Hence, there is exactly one such K where $T(K) = K$: it is the root of the tree. The links, directed from K to $T[K]$, lead in the bottom-up direction. The example graph shown in the problem statement can be directed as follows:



The task is to find the router with minimum *peripherality*: that is, minimum average distance to all other routers in the network. Since the number of routers is fixed, instead of the average, we can think about the total length of paths leading to other routers. For any given router, paths starting at that router can be twofold: they can go down the tree, or up the tree and then possibly down, into some other branch. Let us first focus on the paths going down the tree. For each router, we will compute the total length of all the paths going down the tree. However, to preserve linear running time, we have to calculate such total lengths for all the routers in a single scan of the tree. It is not that difficult, if we scan the tree in a bottom-up order and calculate the sizes of subtrees at the same time. So, first let us fix some bottom-up order in which the routers will be processed. Of course, the root must be the last one. Still, there can be many such orderings. We can put all the leaves first, then all the routers from which only leaves descend, and so on. To implement it in an efficient way, we should calculate, for each router K, the number of such routers I for which $T[I] = K$; that is, the *degree* of K. Whenever one of K's direct descendants is put in the ordering, K's degree is decreased. Once it becomes zero, K can be put in the ordering. For the example tree from the problem statement, the routers can be ordered as: $2, 3, 5, 6, 7, 4, 8, 0, 9, 1$. The following piece of code (in Python) calculates the degrees of routers (array D) and then the ordering of routers (array O):

```python
N = len(T)
D = [0]*N
for U in xrange(N):
    V = T[U]
    if V != U:
        D[V] = D[V] + 1
O = [-1]*N
M = 0
for U in xrange(N):
    if D[U] == 0:
        O[M] = U
        M = M + 1
for K in xrange(N-1):
    U = O[K]
    V = T[U]
    D[V] = D[V] - 1
    if D[V] == 0:
        O[M] = V
        M = M + 1
```

Now, for each router V we have to calculate the size of the subtree rooted in V (denoted by $S[V]$) and the total length of paths going down from V (denoted by $P(V)$). For the example tree from the problem statement, arrays S and P are as follows:

$S[0] = 6$	$S[1] = 10$	$S[2] = 1$	$S[3] = 1$	$S[4] = 3$
$S[5] = 1$	$S[6] = 1$	$S[7] = 1$	$S[8] = 2$	$S[9] = 9$
$P[0] = 8$	$P[1] = 25$	$P[2] = 0$	$P[3] = 0$	$P[4] = 2$
$P[5] = 0$	$P[6] = 0$	$P[7] = 0$	$P[8] = 1$	$P[9] = 16$

Let U be a direct descendant of V, that is $T[U] = V$. The total length of paths going down from V through U equals $P[U] + S[U]$. Using this observation, we can calculate arrays S and P in a single scan of the tree:

```python
P = [0]*N
S = [1]*N
for K in xrange(N-1):
    U = O[K]
    V = T[U]
    S[V] = S[V] + S[U]
    P[V] = P[V] + P[U] + S[U]
```

Now, array P should be increased to include also the lengths of paths going up the tree. We can do this by processing the tree in a top-down order (that is, reversal of O). For the root, no change is needed. Let U be a direct descendant of V; that is, $T[U] = V$. Let us assume that the total length of all the paths going from V equals $P[V]$. What should be the value of $P[U]$? If we move from V to U, then all the paths going from V through U shorten by one link, all the paths going from V and not through U extend by one link, and, instead of a path from V to U, there is a path from U to V. So, $P[U] = P[V] - S[U] + (N - S[U])$. For the example tree from the task statement, the final values in array P should be as follows:

$P[0] = 15$	$P[1] = 25$	$P[2] = 27$	$P[3] = 25$	$P[4] = 19$
$P[5] = 27$	$P[6] = 29$	$P[7] = 25$	$P[8] = 21$	$P[9] = 17$

```
for K in xrange(N-2, -1, -1):
    U = O[K]
    V = T[U]
    P[U] = P[V] - S[U] + N - S[U]
```

Finally, it is sufficient to find the position of the minimum value in array P.

```
C = 0
for U in xrange(1, N):
    if P[U] < P[C]:
        C = U
return C
```