

Tree nodes distance

1 Description of problem

This exercise is about parallelizing a random tree traversal to compute the distance of each node from the root. We assume that each node has a weight which corresponds to the time it takes to process it; this weight is not known in advance but computed when the node is visited. The distance of a node from the root is, simply, the sum of the weights of all nodes along the path connecting that node to the root.

The tree is made of nodes of type `node_t` which contains the following members

- **weight**: the weight of the node;
- **dist**: the distance of the node from the root;
- **id**: the node number;
- ***p**: a pointer to the parent of the node.

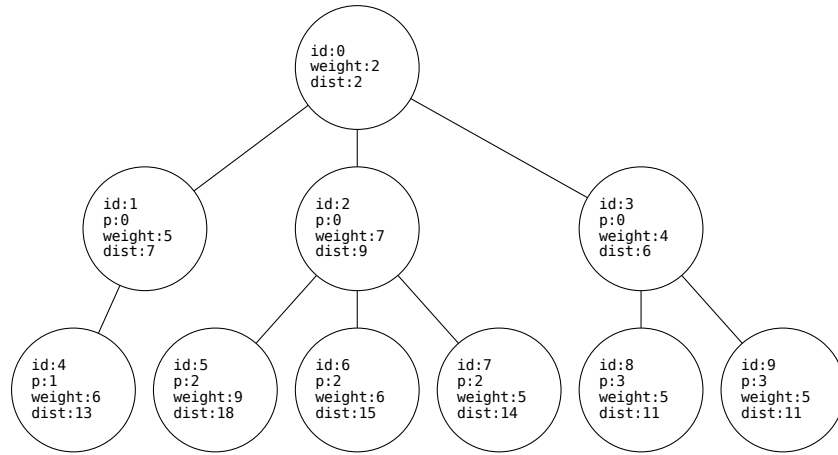
The tree contains `n` nodes and all the nodes of the tree are stored in a `nodes` array of size `n`. We assume that the nodes are numbered in a top-down fashion and stored in the `nodes` array in the corresponding order. This means that a simple natural order traversal of the `nodes` array ensures that each node is visited **after** its parent.

The `tree_dist_seq` routine performs a sequential traversal of the tree through the following pseudo-code:

```
int node;

for(node=0; node<n; node++){
    nodes[node].weight = process(nodes[node]);
    nodes[node].dist   = nodes[node].weight;
    if(nodes[node].p) nodes[node].dist += (nodes[node].p)->dist;
}
```

In this code, every time a node is visited, its weight is computed and then its distance is computed as the sum of its weight plus the distance of its parent (if it exists). In the end, the result of this code is like in the figure below:



2 Package content

In the `tree.dist` directory you will find the following files:

- `main.c`: this file contains the main program which first initializes the tree for a provided number of nodes. The main program then calls a sequential routine `tree.dist.seq` containing the above code, then calls the `tree.dist.par` routine which is supposed to contain a parallel version of the traversal code.
- `tree.dist.seq.c`: contains a routine implementing a sequential traversal with the code presented above.
- `tree.dist.par.c` contains a routine implementing a parallel tree traversal. **Only this file has to be modified for this exercise.**
- `aux.c`, `aux.h`: these two files contain auxiliary routines and **must not be modified**.


The code can be compiled with the `make` command: just type `make` inside the `tree.dist` directory; this will generate a `main` program that can be run like this:

```
$ ./main n s
```

where `n` is the number of nodes in the tree. The argument `s` is the seed for the random number generation (which is used to build the tree), and can be used to create trees of different shapes for a fixed number of nodes. Upon execution, the `main` program generates a `tree.dot` containing a graphical representation of the tree. This can be visualized using the `xdot` program:

```
$ xdot tree.dot
```

3 Assignment

-  At the beginning, the `tree_dist_par` routine contains an exact copy of the `tree_dist_seq` one. Modify these routine in order to parallelize it. Make sure that the result computed by the two routines (sequential and parallel ones) is consistently (that is, at every execution of the parallel code) the same; a message printed at the end of the execution will tell you whether this is the case. **This exercise must be done using OpenMP tasks.**