Bottom-up tree traversal

1 Bottom-up tree traversal

This exercise is about parallelizing the traversal of an unknown tree starting from its leaves.

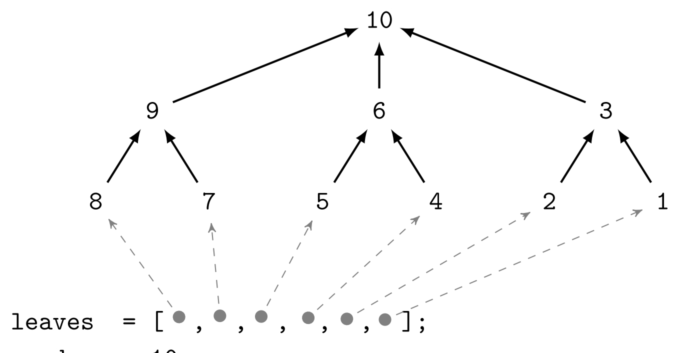
Each node of the tree is defined by the data structure `struct node` containing the following fields

- `unsigned id`: an integer indicating an identifier of the node; identifiers go from 1 to the number of nodes in the tree.
- `struct node *parent`: a pointer to the parent node; for the root of the tree, this parent is set to `NULL`.
- `unsigned data`: a generic data associated with the node; this is used by the `process_node` routine described below. This value is not important for the exercise and need not be used directly.

No information is available about the structure of this tree apart from

- `int nnodes`: the number of nodes it contains;
- `int nleaves`: the number of leaves;
- `struct node **leaves`: an array of pointers to the leaf nodes.

An example for a very small tree with 10 nodes is shown below



```
leaves  = [●,●,●,●,●,●];
nnodes  = 10;
nleaves = 6;
```

Note that the tree can be of any form: the number of children per node may vary, the depth of the branches may be different etc.

The routine `process_node` must be called **once** (and **only once**) on each node of the tree. Because only the leaves of the tree are directly accessible, the only way to do this is to start from the leaves and go up the tree until the root is reached using the `parent` field of the `node` data structure.

2 Package content

In the `tree_bottomup` directory you will find the following files:

- `main.c`: This file contains the main program. This reads from command line the number of nodes in the tree and then generates a random tree accordingly using the `generate_tree` routine. This routine takes as input the number of nodes `nnodes` and returns the number of leaves and the list of leaves in the `nleaves` and `leaves` output arguments. Then it calls the `bottom_up` routine that has to be implemented as described below. Finally it checks if the result of the `bottom_up` routine is correct calling the `check_result` routine.
- `aux.c, aux.h`: these two files contain auxiliary routines and declarations and **must not be modified**. Among other things, they contain the declaration of the `struct node` datastructure and the `process_node` routine whose interface is

  ```
  void process_node(struct node *n);
  ```

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

```
$ ./main nnodes
```

where `nnodes` is the number of nodes in the tree. Upon execution of the main program the tree is printed in the `td_tree.dot` file which can be visualized with the `dotty` tool like this

```
$ ./dotty td_tree.dot
```

3 Assignment

- ⌨ The objective of this exercise is to write a routine called `bottom_up` that visit all the nodes of the tree and, on each of them, calls the `process_node` routine. **This has to be done in parallel**. At the beginning the `bottom_up` routine is empty. When you have a fully functioning code, think about load-balance and see if you can do something to improve it in order to achieve better performance.

- ✎ Describe (briefly) the approach you have used to achieve the parallel tree traversal. Comment quickly on the main difficulties and how you addressed them. Report the execution times for the implemented parallel version with 1, 2 and 4 threads and for different tree sizes. Analyze and comment on your results: is the achieved speedup reasonable or not? Report your answer in the form of comments at the bottom of the `main.c` file.

Advice

- When developing your code, always work on trees of small size (10 nodes, for example) but when you want to evaluate the performance use large size trees (10000 nodes, for example);