Assignment No. 6: Multi-way Trees
Transforms between different representations

Allocated time: 2 hours

Implementation

You are required to implement **correctly** and **efficiently** *iterative* and *recursive* binary tree traversal. You may find any necessary information and pseudo-code in your course and seminar notes.

Moreover, the **correct** and **efficient** implementation of *linear* complexity algorithms is required for transforming multi-way trees between the following representations:

> **R1**: *Parent representation*: for each index, the value in the vector represents the parent's index, e.g.: $\Pi = \{2,7,5,2,7,7,-1,5,2\}$
> **R2**: *Multi-way tree representation*: each node contains the key and a vector of child nodes.
> **R3**: *Binary representation*: each node contains the key and two pointers, one to the first child and the second to the right sibling (e.g., the next sibling).

Therefore, you need to define transformation **T1** from the parent representation (**R1)** to the multi-way tree representation (**R2**), and then the transformation **T2** from the multi-way tree representation (**R2**) to the binary representation (**R3**). For all representations (**R1**, **R2**, **R3**), you need to implement the Pretty Print (**PP**) display (see page 2).

Define the data structures. You can use intermediate structures (e.g., additional memory).

Requirements

1. **Implementation of *iterative* and *recursive* binary tree traversal in *O(n)* and *with constant additional memory* (2p)**

You will have to prove your algorithm(s) work on a small-sized input.

2. **Comparative analysis of the *iterative* vs *recursive* tree traversal from the perspective of the number of operations (2p)**

! Before you start to work on the algorithms evaluation code, make sure you have a **correct algorithm**!

In the comparative analysis of the iterative vs recursive version, you have *to count only the print key operations*, varying the number of nodes from the tree in the [100, 10000] range with an increment of maximum 500 (we suggest 100).

For binary tree construction you can start from an array with a variable size and pick a random node as root.

3. **Correct implementation for Pretty-print for R1 (1p)**

4. **Correct implementation for *T1* and pretty-print for *R2* (1p) + *T1* in linear time (1p)**

5. **Correct implementation for *T2* and pretty-print for *R3* (2p) + *T2* in linear time (1p)**

The correctness of the algorithms should be demonstrated using the example from **R1** (*Π*). Use Pretty Print for all three representations.

Explain the data structures you used for representations **R2** and **R3**.

Analyse the time and space efficiency of the two transformations. Did you achieve *O(n)*? Did you use additional memory?