1 Assignment No. 6: Multi-way Trees

Transforms between different representations
Allocated time: 2 hours

1.1 Implementation

1. You are required to implement **correctly** and **efficiently** *iterative* (can be found in Lecture 6) and *recursive* binary tree traversal. You may find any necessary information and pseudo-code in your course and seminar notes.

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

   1. **R1**: *Parent representation*: for each index, the value in the vector represents the parent's index, e.g.: $\Pi = \{2, 7, 5, 2, 7, 7, -1, 5, 2\}$
   2. **R2**: *Multi-way tree representation*: each node contains the key and a vector of child nodes.
   3. **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 3).

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

1.2 Minimal requirements for grading

The lack of any of the minimum requirements (even partially) may result in a lower grade through penalties or refusal to accept the assignment resulting in a grade of 0.

- *Demo*: Prepare a demonstration of correctness for each algorithm implemented. The correctness of each algorithm is demonstrated through a simple example (maximum 10 values).

- Write your observations in the header (block comments) section at the beginning of your *main.cpp* file.

- We do not accept assignments without code indentation and with code not organized in functions (for example where the entire code is in the main function).

- *The points from the requirements correspond to a correct and complete solution, quality of interpretation from the block comment and* ***the correct answer to the questions from the teacher.***

1.3 Requirements

1.3.1 Implementation of *iterative* (*with constant additional memory*) and *recursive* binary tree traversal in *O(n)* (3p)

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

Only one tree traversal order is required— Preorder, Inorder, or Postorder—and you may choose whichever you prefer.

You may generate a sorted array that can be transformed into a binary tree through a recursive approach.

1.3.2 Transformations between tree representations

1. **Correct implementation for Pretty-print for R1 (2p)**

   Demo: The correctness of the algorithms should be demonstrated using the example $\Pi = \{2, 7, 5, 2, 7, 7, -1, 5, 2\}$ with a pretty-print, as shown in the Figure below.

2. **Correct implementation for *T1* (from R1 to R2) and pretty-print for *R2* (1p) + *T1* in linear time (1p)**

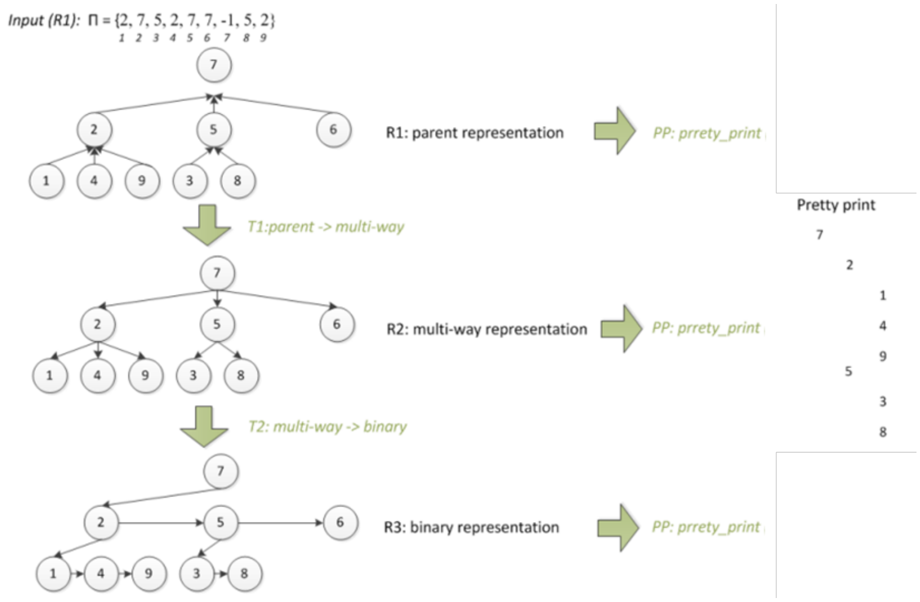
   Demo: The correctness of the algorithms should be demonstrated using the example $\Pi = \{2, 7, 5, 2, 7, 7, -1, 5, 2\}$ with a pretty-print, as shown in the Figure below.

3. **Correct implementation for *T2* (from R2 to R3) and pretty-print for *R3* (2p) + *T2* in linear time (1p)**

   Demo: The correctness of the algorithms should be demonstrated using the example $\Pi = \{2, 7, 5, 2, 7, 7, -1, 5, 2\}$ with a pretty-print, as shown in the Figure below.

Use Pretty Print for all three representations. Each representation (R1, R2, R3) should have a pretty print of its own with a different implementation but with the same result.

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



Input (R1): Π = {2, 7, 5, 2, 7, 7, -1, 5, 2}
R1: parent representation
PP: prrety_print
T1:parent -> multi-way
R2: multi-way representation
PP: prrety_print
T2: multi-way -> binary
R3: binary representation
PP: prrety_print
Pretty print