Project Advanced Data Structures Summer Term 2022

In this document, we describe the programming task fro the summer term 2022. Main objective of the project is to implement multiple data structures that have been presented in the **lecture Advanced Data Structures**. In addition to the implementation, the project consists of the following: documentation, evaluation, and a final presentation of the project/implementation.

Dynamic Bit Vectors and BP

For this year's project, the following data structures have to be implemented:

1. A dynamic bit vector, which supports the operations *access*, *insert* , *delete*, and *flip*. Additionally, the bit vector has to support the operations *rank* and *select*.
2. A dynamic BP data structure that supports the following operations, which we did discuss in lecture 03: *i-th child*, *parent*, *suptree size*, *deletenode* and *insertchild*.

Input and Output

Your program has to be usable via the command line. The input has the following format.

```
ads_programm_a [bv|bp] input_file output_file
```

The parameter *bv* is used to create a dynamic bit vector w.r.t. the input file and the parameter *bp* is used to create a dynamic succinct tree using the BP representation w.r.t. the input file.

bv. The first line of the input is an integer $n \in \mathbb{N}_0$ followed by n lines consisting of either `0` or `1`. This is the initial configuration of the bit vector. Next, there is an arbitrary number of lines containing the following commands:

- `insert` i [0|1] (insert a 0 or 1 at the i-th position of the bit vector)
- `delete` i (delete the i-th bit)
- `flip` i (flip the i-th bit)
- `rank` [0|1] i (write $rank_0$ or $rank_1$ up to position i to the output file)
- `select` [0|1] i (write $select_0$ or $select_1$ for the i-th occurrence to the output file)

All commands will be feasible, for example, there will be no select query that cannot be answered. Example:

```
4
0
1
1
1
insert 3 1
insert 0 0
select 0 2
```

In addition to the output file, there should be an output on the command line, which is structured as followed:

```
RESULT algo=bv name<first_last_name> time=<running_time_without_output_in_ms> space=<required_space_in_bits>
```

Example:

```
RESULT algo=bv name<florian_kurpicz> time=<512>
```

bp. Here, the input file consists of an arbitrary number of lines. Each line will contain one of the following commands:

- `deletenode` v (delete node v)
- `insertchild` v i k (defined in Lecture 05)

- `child` v i (write i-th child of v to output file)
- `subtre_size` v (write subtree size of v (including v) to output file)
- `parent` v (write parent of v to output file)

Again, all commands will be feasible, e.g., there will be no command trying to delete the root of the tree. Initially, the tree only consists of a root.

After computing the tree, the program should write the degree[1] of every node to the output. Nodes are visited in depth-first order and outputs should occur in preorder. Each degree should be written in a new line.
Example:

```
insertchild 0 1 0
insertchild 0 1 0
insertchild 0 1 0
insertchild 0 2 2
deletenode 1
```

In addition to the output file, there should be an output on the command line, which is structured as followed:

```
RESULT algo=bv name<first_last_name> time=<running_time_without_output_in_ms> space=<required_space_in_bits>
```

Example:

```
RESULT algo=bp name<florian_kurpicz> time=<512>
```

Minimal Requirements

The project can be implemented in a programming language of your choice. However, it is important that the program runs on a Linux system (Ubuntu 20.04.3 LTS)! Your project should come with instructions on how to compile/run the project (including required software to do so). The program has to support the input and output format described above.

Important: **You are not allowed to use any additional libraries**, which have not been approved by Florian Kurpicz. Pleas ask (via mail `kurpicz@kit.edu`) before you include any external library. Standard libraries cab be used without asking.

Documentation, Evaluation, and Presentation

The code has to be documented in such a way that a reader can easily understand what happens where. To this end, you should consider also documenting *why* something is done and not only *what* is done.

The Evaluation is part of the presentation. Here, you can compare your running times for different inputs. A comparison with other implementations is also possible. (Note that the command line output can be used to create plots, see `https://github.com/bingmann/sqlplot-tools/` for more details).

The results have to be presented in a **exactly** five minute long presentation. Here, you should discuss what your have implemented, how you have implemented, and what is special about your implementation, e.g., where did you have problems or what part in your implementation do you like.

Competition

There is a small competition, every project automatically takes part in. The result in the competition does not influence the grade of the project at all! During the competition, the running time of all tests are weighted and added. The smallest running time wins. The weights are: 45 % running time and 55 % memory requirements.

Deadline

The project has to be submitted no later than 15.07.2022 at 23:59 (MESZ) via mail (`kurpicz@kit.edu`). Preferably the mail contains a link to a repository. Submissions after the deadline will not be considered.

[1] only out-degree