Algorithms and Data Structures

EADS 2023Z LAB/103

Task #3 (term 2023Z)

Task #3 deadline is Wednesday, January 10th, 2024, 2 pm.

There are several steps to complete:

- Design `avl_tree` class according to very limited specification below.
- Implementing your class write unit tests in parallel (try to write test before actual implementation of the behaviour of your tree).
- You should implement your class template in the `avl_tree.h` file, and provide tests of your container in `avl_tree_test.h` and `avl_tree_test.cpp` files.
- Implement external function template `maxinfo_selector`.
- Implement external function `count_words`.
- Be prepared to modify fragments of your solution or write additional function/template during the lab class on January 10th.

1 PART1 - CLASS DESIGN

Design a class to represent AVL tree. Write unit tests for the designed class, at least one test per method.

```
template <typename Key, typename Info>
class avl_tree
{
public:
	avl_tree();
	avl_tree (const avl_tree& src);
	~avl_tree();
	avl_tree& operator=(const avl_tree& src);

	// insert (key, info) pair

	// remove given key

	// check if given key is present

	// print nicely formatted tree structure

	Info& operator[](const Key& key);
	const Info& operator[](const Key& key) const;

	// what else can be useful in such a collection?
};
```

2 PART 2 – ADDITIONAL FUNCTIONS

Implement two additional functions:

```
template <typename Key, typename Info>
std::vector<std::pair<Key, Info>>
      maxinfo_selector(const avl_tree<Key, Info>& tree, unsigned cnt);
```

`maxinfo_selector` returns *cnt* elements of the *tree* with the highest values of the info member.

```
avl_tree<string, int> count_words(istream& is);
```

`count_words` creates a dictionary mapping words found in the is stream to number of occurrences of the word in this stream. You can use beagle_voyage.txt (yes, this is The Voyage of the Beagle by Charles Darwin as published on the Gutenberg Project page - https://www.gutenberg.org/) as a slightly larger test for your `count_words` function.

How the execution time of your `count_words` compares to the word counter implemented with the map collection from the standard library?

You can use following code to measure the execution time (don't forget to include chrono header):

```
    for (int rep = 0; rep < 5; ++rep)
    {
        ifstream is("beagle_voyage.txt");
        if (!is)
        {
            cout << "Error opening input file.\n";
            return 1;
        }

        auto start_time = std::chrono::high_resolution_clock::now();

        string word;
        map<string, int> wc;  // counting word occurrences in the stream
        while (is >> word)
        {
            wc[word]++;
        }

        auto end_time = std::chrono::high_resolution_clock::now();
        auto time = end_time - start_time;

        std::cout << "Ellapsed time: " << time/std::chrono::milliseconds(1)
                  << " ms.\n";
    }
```

If the computations are not lengthy it is good to repeat time measurements several times. The result on my machine is following:

```
Ellapsed time: 40 ms.
Ellapsed time: 41 ms.
Ellapsed time: 49 ms.
Ellapsed time: 42 ms.
Ellapsed time: 43 ms.
```

(Is your avl_tree implementation slower ☹ or faster ☺?).