DSBA Introduction to Programming // Workshops 11 & 12: Maps and Sets

Spring semester 2020/21

Date: 16.02.2021

Ed. 1.0 as of 16.02.2021

by *Sergey Shershakov*

General instructions

Proposed tasks can be divided into classwork and homework.

Task Description

Consider the proper problem decomposition.

Task 1. Welcome `std::map`

You need to develop a program that asks a user for input a non-negative integer `n` and two integers `r<=q`. The program generates randomly a sequence of `n` integers from the range `[r, q)` using a uniform distribution. The numbers are put in a vector. The vector is printed on the screen in the forward and backward order (consider using iterators for this).

> While testing the program, consider a narrow range `r<=q` s.t. the probability of duplicates is higher.

The program puts the vector's elements in an `std::map<int, size_t>` object, where the *key* of the mapping is another vector's element `x` and the *value* of the mapping is the number of times `x` appears in the sequence.

> NB: consider different approaches for filling the map up with vector's elements. Use operator[], insert() method. Discuss the difference and side effects of using these approaches. Also, discuss the time complexity of the approaches.

The program provides the following functionality upon request.

- Outputs the elements of the map (pairs) in the forward and backward order.
- Outputs only keys of the map.
- Outputs each `t`-s element of the map, where `t` is a parameter of the corresponding method.

> While outputting the map's elements, consider the difference between using iterators and the range-based for loop. How one can properly bind a variable with another map's pair without copying of it?

- Queries the map for existence of some *key* `k`. Create an appropriate method, which returns `true` **by value** and the *value* through a **reference variable** if the given key `k` exists. Otherwise the method returns false and the value of reference variable is undefined.
    - Modify the method (create another (*overloaded?*) variant) that returns a pair of `<bool, int>` with corresponding values.
- Modifies the mapping adding some given number for all *values* of map.
    - Can we do the same for map's *keys*? Why?
- Removes all negative *keys* from the map. Consider proper iterator re-initialization on erasing.

Task 2. Welcome `std::set`

Consider another (second) mapping as follows: `std::map<int, true>`. The map consists of *keys* in range `r<=q` and the corresponding *values* are `true` if the *key* appears in the initial vector/the map from the **Task 1**, and `false` otherwise.

- Create a method that fills up such a map by using the map from **Task 1**. Use the *developed query method* to check whether another *key* is represented in the first map or not. Discuss the time complexity of this method.
- Create a method that queries the second map for presence of an element with the given *key* `k`. It must work similarly for the corresponding query method from **Task 1**, yet it returns the only boolean value indicating presence of the *key*.

Consider the following claim: "a *key* is mapped to `true` in the map iff it belongs to the corresponding set of existing keys". So, there is **equivalence** between the following definitions:

```
std::map<T,bool> ↔ std::set<T>
```

Creates a set of integers, `std::set<int>`. The set consists the keys from the first map from **Task 1**.

- Create a method that makes the set from the first map.
- Create a method that prints the set in forward and backward order.
    - Consider both approaches, the iterators- and the range-based for loop.
- Create a method that queries the set for the presence of an element with the given *key* `k`. It must work similarly for the corresponding method for the second map above.

> Note, that any of `map<T, bool>`, for any `T`, is almost always the sign of *anti-pattern*. Consider the substitution to `set<T>` in such cases.

Task 3. Multimaps and Multisets

Consider the object of type `std::multimap<int, size_t>`. It represents a map, which may contains duplicate keys. Consider its first element (*key*) as an integer from the initial vector, and the second as the position (index) in vector where it appears.

- Create a function that makes the multimap from the initial vector.
- Create a function that prints the multimap (both *keys* and *value*; separately only *keys* or *values*) in forward and backward order.
    - Consider both approaches, the iterator- and the range-based for loop.
- Create a method that prints the multimap's elements with the given key `k` (if ones exist). Consider the methods `equal_range`, `lower_bound` and `upper_bound` to do this.

Similarly to a multimap, a multiset can store duplicate of keys.

Create a `std::multiset<int>` object for storing *keys* from the multimap above.

- Create a method that makes a multiset object from the multimap object.
- Create a method for printing out the values of the multiset.
- Create a method that prints out the keys that are equal to the provided key `k`. Use an approach similar to one for the multimap.

Task 4. Unordered associative containers

Consider using the following containers in the similar contexts as described above: `std::unodered_map` and `std::unodered_set`.

Is there any difference between the sequence of enumerated elements for corresponding containers (e.g. ordered `set` and `unordered_set`)?

> Explore the mentioned difference while printing out the containers.

Discuss with the trainer the difference in requirements for the stored keys for ordered and unordered containers. Why we consider `operator <` in the former case and the `operator ==` and `std::hash<TKey>` in the latter case? Explore the following methods while adding elements in the container: `bucket_count`, `max_bucket_count`, `bucket_size`, `bucket`, `load_factor`.