Lab 5: Hashtable

Implement in C++ the given **container** (ADT) using a given representation and a **hashtable** with a given collision resolution (separate chaining, coalesced chaining, open addressing) as a data structure. You are not allowed to use any container or data structure from STL or from any other library.

Do not implement a separate class for the hashtable (or dynamic array, or anything), implement the container directly!

The hashtable has to be dynamic: no matter what collision resolution has to be used, set a threshold for α and resize and rehash the table when the actual load factor is higher than α.

1. **ADT Matrix** – represented as a sparse matrix where <line, column, value> triples (value ≠ 0) are memorized, ordered lexicographically considering the line and column of every element. The elements are stored in a hashtable with separate chaining.
2. **ADT Matrix** – represented as a sparse matrix where <line, column, value> triples (value ≠ 0) are memorized, ordered lexicographically considering the line and column of every element. The elements are stored in a hashtable with coalesced chaining.
3. **ADT Matrix** – represented as a sparse matrix where <line, column, value> triples (value ≠ 0) are memorized, ordered lexicographically considering the line and column of every element. The elements are stored in a hashtable with open addressing, quadratic probing.
4. **ADT Bag** – using a hashtable with separate chaining in which the elements are stored. If an element appears multiple times, it will be stored multiple times in the hashtable.
5. **ADT Bag** – using a hashtable with coalesced chaining in which the elements are stored. If an element appears multiple times, it will be stored multiple times in the hashtable.
6. **ADT Bag** – using a hashtable with open addressing (double hashing) in which the elements are stored. If an element appears multiple times, it will be stored multiple times in the hashtable.
7. **ADT Bag** – using a hashtable with separate chaining in which (unique element, frequency) pairs are stored.
8. **ADT Bag** – using a hashtable with coalesced chaining in which (unique element, frequency) pairs are stored.
9. **ADT Bag** – using a hashtable with open addressing (quadratic probing) in which (unique element, frequency) pairs are stored.
10. **ADT Set** – using a hashtable with separate chaining.
11. **ADT Set** – using a hashtable with coalesced chaining.
12. **ADT Set** – using a hashtable with open addressing and double hashing.
13. **ADT Map** – using a hashtable with separate chaining.
14. **ADT Map –** using a hashtable with coalesced chaining.
15. **ADT Map –** using a hashtable with open addressing and quadratic probing.
16. **ADT Sorted Map** – using a hashtable with separate chaining.

17. **ADT Sorted Map** – using a hashtable with coalesced chaining. In the constructor of the iterator create a sorted array of the elements and use it for iterating.

18. **ADT Sorted Map** – using a hashtable with open addressing and double hashing. In the constructor of the iterator create a sorted array of the elements and use it for iterating.

19. **ADT MultiMap** – using a hashtable with separate chaining in which (key, value) pairs are stored. If a key has multiple values, it appears in multiple pairs.

20. **ADT MultiMap** – using a hashtable with coalesced chaining in which (key, value) pairs are stored. If a key has multiple values, it appears in multiple pairs.

21. **ADT MultiMap** – using a hashtable with open addressing and quadratic probing in which (key, value) pairs are stored. If a key has multiple values, it appears in multiple pairs.

22. **ADT MultiMap** – using a hashtable with separate chaining in which unique keys are stored with a dynamic array of the associated values.

23. **ADT MultiMap** – using a hashtable with coalesced chaining in which unique keys are stored with a dynamic array of the associated values.

24. **ADT MultiMap** – using a hashtable with open addressing and double hashing in which unique keys are stored with a dynamic array of the associated values.

25. **ADT SortedMultiMap** – using a hashtable with separate chaining in which (key, value) pairs are stored. If a key has multiple values, it appears in multiple pairs.

26. **ADT SortedMultiMap** – using a hashtable with coalesced chaining in which (key, value) pairs are stored. If a key has multiple values, it appears in multiple pairs. In the constructor of the iterator create a sorted array of pairs and use it for iterating.

27. **ADT SortedMultiMap** – using a hashtable with open addressing and quadratic probing in which (key, value) pairs are stored. If a key has multiple values, it appears in multiple pairs. In the constructor of the iterator create a sorted array of pairs and use it for iterating.

28. **ADT SortedMultiMap** – using a hashtable with separate chaining in which unique keys are stored with a dynamic array of the associated values.

29. **ADT SortedMultiMap** – using a hashtable with coalesced chaining in which unique keys are stored with a dynamic array of the associated values. In the constructor of the iterator create a sorted array of (key, dynamic array of values) pairs and use it for iterating.

30. **ADT SortedMultiMap** – using a hashtable with open addressing and double hashing in which unique keys are stored with a dynamic array of the associated values. In the constructor of the iterator create a sorted array of (key, dynamic array of values) pairs and use it for iterating.