Lab 4: 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.

If you need to sort elements, you have to implement your own sorting algorithm.

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 SortedBag** – 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.
11. **ADT SortedBag** – 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. In the constructor of the iterator create a sorted array of the elements and use that array for iterating.

12. **ADT SortedBag** – using a hashtable with open addressing (linear probing) in which the elements are stored. If an element appears multiple times, it will be stored multiple times in the hashtable. In the constructor of the iterator create a sorted array of the elements and use that array for iterating.
13. **ADT SortedBag** – using a hashtable with separate chaining in which (unique element, frequency) pairs are stored.
14. **ADT SortedBag** – using a hashtable with coalesced chaining in which (unique element, frequency) pairs are stored. In the constructor of the iterator create an array of (element, frequency) pairs, sorted by the element and use that array for iterating.
15. **ADT Bag** – using a hashtable with open addressing (quadratic probing) in which (unique element, frequency) pairs are stored. In the constructor of the iterator create an array of (element, frequency) pairs, sorted by the element and use that array for iterating.
16. **ADT Set** – using a hashtable with separate chaining.
17. **ADT Set** – using a hashtable with coalesced chaining.
18. **ADT Set** – using a hashtable with open addressing and double hashing.
19. **ADT SortedSet** – using a hashtable with separate chaining.
20. **ADT SortedSet** – using a hashtable with coalesced chaining. In the constructor of the iterator create a sorted array of the elements and use that array for iterating.
21. **ADT SortedSet** – using a hashtable with open addressing and double hashing. In the constructor of the iterator create a sorted array of the elements and use that array for iterating.
22. **ADT Map** – using a hashtable with separate chaining.
23. **ADT Map –** using a hashtable with coalesced chaining.
24. **ADT Map –** using a hashtable with open addressing and quadratic probing.
25. **ADT Sorted Map** – using a hashtable with separate chaining.
26. **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.
27. **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.
28. **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.
29. **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.
30. **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.
31. **ADT MultiMap** – using a hashtable with separate chaining in which unique keys are stored with a dynamic array of the associated values.
32. **ADT MultiMap** – using a hashtable with coalesced chaining in which unique keys are stored with a dynamic array of the associated values.
33. **ADT MultiMap** – using a hashtable with open addressing and linear probing in which unique keys are stored with a dynamic array of the associated values.

34. **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.
35. **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.
36. **ADT SortedMultiMap** – using a hashtable with open addressing and linear 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.
37. **ADT SortedMultiMap** – using a hashtable with separate chaining in which unique keys are stored with a dynamic array of the associated values.
38. **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.
39. **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.