L4: Hashtabelle

Bekommen in L5

Abgabe in L6

Implementiere in C++ den gegebenen **Container (ADT)** mithilfe der gegebenen **Repräsentierung** und mit einer **Hashtabelle mit der gegebenen Strategie für Kollisionsauflösung** (unabhängige Verkettung, coalesced Verkettung, offene Adressierung) als Datenstruktur. Für die Implementierung dürft ihr keine Containers oder Datenstrukturen aus STL (oder aus anderen Bibliotheken) benutzen.

Bemerkung. Ihr müsst nicht eine separate Klasse für Hashtabelle (oder dynamisches Array oder etwas Anderes) implementieren, implementiere den Container direkt!

1. **ADT Matrix –** repräsentiert als schwachbesetzte Matrix (sparse) mithilfe einer Hashtabelle mit unabhängiger Verkettung wo Tupel der Form (Zeile, Spalte, Wert) (Wert $\neq 0$) gespeichert werden.
2. **ADT Matrix –** repräsentiert als schwachbesetzte Matrix (sparse) mithilfe einer Hashtabelle mit coalesced Verkettung wo Tupel der Form (Zeile, Spalte, Wert) (Wert $\neq 0$) gespeichert werden.
3. **ADT Matrix –** repräsentiert als schwachbesetzte Matrix (sparse) mithilfe einer Hashtabelle mit offener Adressierung, mit doppeltem Hashing wo Tupel der Form (Zeile, Spalte, Wert) (Wert $\neq 0$) gespeichert werden.
4. **ADT Bag** – repräsentiert mithilfe einer Hashtabelle mit unabhängiger Verkettung, wo die Elemente gespeichert sind. Falls ein Element mehrmals vorkommt, dann wird es mehrmals in der Hashtabelle gespeichert.
5. **ADT Bag** – repräsentiert mithilfe einer Hashtabelle mit coalesced Verkettung, wo die Elemente gespeichert sind. Falls ein Element mehrmals vorkommt, dann wird es mehrmals in der Hashtabelle gespeichert.
6. **ADT Bag** – repräsentiert mithilfe einer Hashtabelle mit offener Adressierung, mit doppeltem Hashing, wo die Elemente gespeichert sind. Falls ein Element mehrmals vorkommt, dann wird es mehrmals in der Hashtabelle gespeichert.
7. **ADT Bag** – repräsentiert mithilfe einer Hashtabelle mit unabhängiger Verkettung, wo Paare der Form (*eindeutiges Element*, *Frequenz*) gespeichert werden.
8. **ADT Bag** – repräsentiert mithilfe einer Hashtabelle mit coalesced Verkettung, wo Paare der Form (*eindeutiges Element*, *Frequenz*) gespeichert werden.
9. **ADT Bag** – repräsentiert mithilfe einer Hashtabelle mit offener Adressierung, mit quadratischem Sondieren, wo Paare der Form (*eindeutiges Element*, *Frequenz*) gespeichert werden.
10. **ADT Set** – repräsentiert mithilfe einer Hashtabelle mit unabhängiger Verkettung

11. **ADT Set** – repräsentiert mithilfe einer Hashtabelle mit coalesced Verkettung
12. **ADT Set** – repräsentiert mithilfe einer Hashtabelle mit offener Adressierung, mit doppeltem Hashing
13. **ADT Map** – repräsentiert mithilfe einer Hashtabelle mit unabhängiger Verkettung
14. **ADT Map** – repräsentiert mithilfe einer Hashtabelle mit coalesced Verkettung
15. **ADT Map** – repräsentiert mithilfe einer Hashtabelle mit offener Adressierung, mit quadratischem Sondieren
16. **ADT SortedMap** – repräsentiert mithilfe einer Hashtabelle mit unabhängiger Verkettung. In dem Constructor des Iterators erstelle ein sortiertes Array von Elementen und benutze es für die Iterierung.
17. **ADT SortedMap** – repräsentiert mithilfe einer Hashtabelle mit coalesced Verkettung. In dem Constructor des Iterators erstelle ein sortiertes Array von Elementen und benutze es für die Iterierung.
18. **ADT SortedMap** – repräsentiert mithilfe einer Hashtabelle mit offener Adressierung, mit doppeltem Hashing. In dem Constructor des Iterators erstelle ein sortiertes Array von Elementen und benutze es für die Iterierung.
19. **ADT MultiMap** – repräsentiert mithilfe einer Hashtabelle mit unabhängiger Verkettung wo Paaren der Form (*key*, *value*) gespeichert werden. Wenn ein Schlüssel mehrere entsprechende Werte hat, dann kommt es in mehreren Paaren vor.
20. **ADT MultiMap** – repräsentiert mithilfe einer Hashtabelle mit coalesced Verkettung wo Paaren der Form (*key*, *value*) gespeichert werden. Wenn ein Schlüssel mehrere entsprechende Werte hat, dann kommt es in mehreren Paaren vor.
21. **ADT MultiMap** – repräsentiert mithilfe einer Hashtabelle mit offener Adressierung, mit quadratischem Sondieren, wo Paaren der Form (*key*, *value*) gespeichert werden. Wenn ein Schlüssel mehrere entsprechende Werte hat, dann kommt es in mehreren Paaren vor.
22. **ADT MultiMap** – repräsentiert mithilfe einer Hashtabelle mit unabhängiger Verkettung wo eindeutige Schlüssel zusammen mit einem dynamischen Array von Werten gespeichert werden.
23. **ADT MultiMap** – repräsentiert mithilfe einer Hashtabelle mit coalesced Verkettung wo eindeutige Schlüssel zusammen mit einem dynamischen Array von Werten gespeichert werden.
24. **ADT MultiMap** – repräsentiert mithilfe einer Hashtabelle mit offener Adressierung, mit doppeltem Hashing, wo eindeutige Schlüssel zusammen mit einem dynamischen Array von Werten gespeichert werden.
25. **ADT SortedMultiMap** – repräsentiert mithilfe einer Hashtabelle mit unabhängiger Verkettung wo Paaren der Form (*key*, *value*) gespeichert werden. Wenn ein Schlüssel mehrere entsprechende Werte hat, dann kommt es in mehreren Paaren vor. In dem Constructor des Iterators erstelle ein sortiertes Array von Elementen und benutze es für die Iterierung.

26. **ADT SortedMultiMap** – repräsentiert mithilfe einer Hashtabelle mit coalesced Verkettung wo Paaren der Form (*key*, *value*) gespeichert werden. Wenn ein Schlüssel mehrere entsprechende Werte hat, dann kommt es in mehreren Paaren vor. In dem Constructor des Iterators erstelle ein sortiertes Array von Elementen und benutze es für die Iterierung.
27. **ADT SortedMultiMap** – repräsentiert mithilfe einer Hashtabelle mit offener Adressierung, mit quadratischem Sondieren, wo Paaren der Form (*key*, *value*) gespeichert werden. Wenn ein Schlüssel mehrere entsprechende Werte hat, dann kommt es in mehreren Paaren vor. In dem Constructor des Iterators erstelle ein sortiertes Array von Elementen und benutze es für die Iterierung.
28. **ADT SortedMultiMap** – repräsentiert mithilfe einer Hashtabelle mit unabhängiger Verkettung wo eindeutige Schlüssel zusammen mit einem dynamischen Array von Werten gespeichert werden. In dem Constructor des Iterators erstelle ein sortiertes Array von Elementen und benutze es für die Iterierung.
29. **ADT SortedMultiMap** – repräsentiert mithilfe einer Hashtabelle mit coalesced Verkettung wo eindeutige Schlüssel zusammen mit einem dynamischen Array von Werten gespeichert werden. In dem Constructor des Iterators erstelle ein sortiertes Array von Elementen der Form (Schlüssel, dynamisches Array von Werten) und benutze es für die Iterierung.
30. **ADT SortedMultiMap** – repräsentiert mithilfe einer Hashtabelle mit offener Adressierung, mit doppeltem Hashing, wo eindeutige Schlüssel zusammen mit einem dynamischen Array von Werten gespeichert werden. In dem Constructor des Iterators erstelle ein sortiertes Array von Elementen der Form (Schlüssel, dynamisches Array von Werten) und benutze es für die Iterierung.

---

31. **ADT Map** – repräsentiert mithilfe einer Hashtabelle mit Cuckoo Hashing
32. **ADT SortedMap** – repräsentiert mithilfe einer verketteten Hashtabelle / Linked Hash Table
33. **ADT Map** – repräsentiert mithilfe einer Hashtabelle mit Hopscotch Hashing
34. **ADT List** – repräsentiert mithilfe einer verketteten Hashtabelle / Linked Hash Table