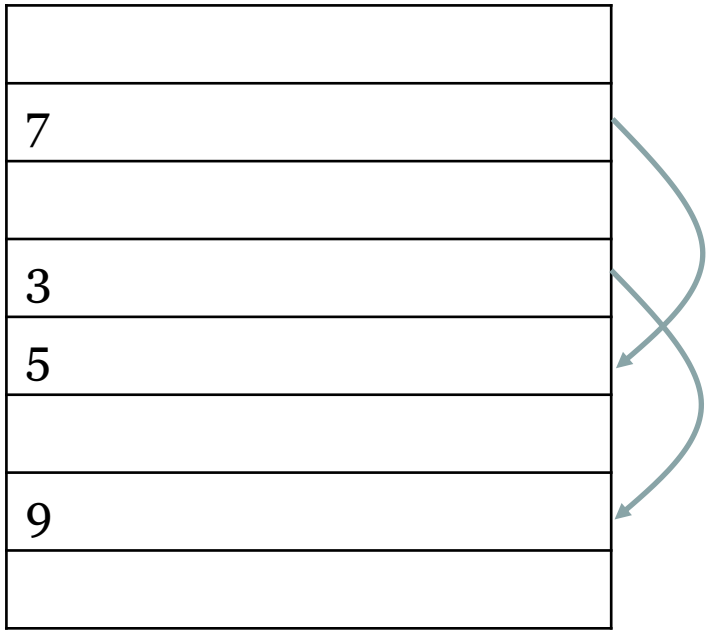
Coalesced Hashing
(Coalesced Chaining)

Eine Kombination der Ideen von separate chaining und double hashing

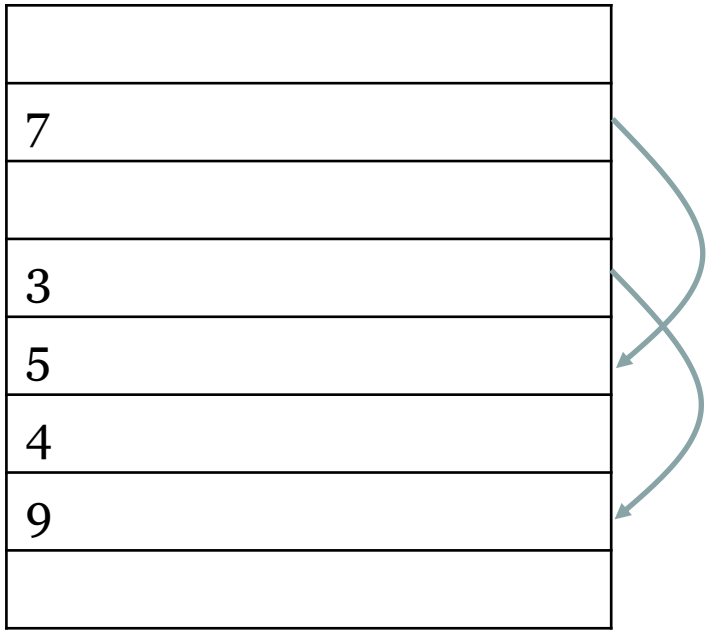
Statt extern immer neuen Speicher anzufordern (wie bei separate chaining) werden einfach Ketten in den schon in der Tabelle vorhandenen Positionen gebildet.



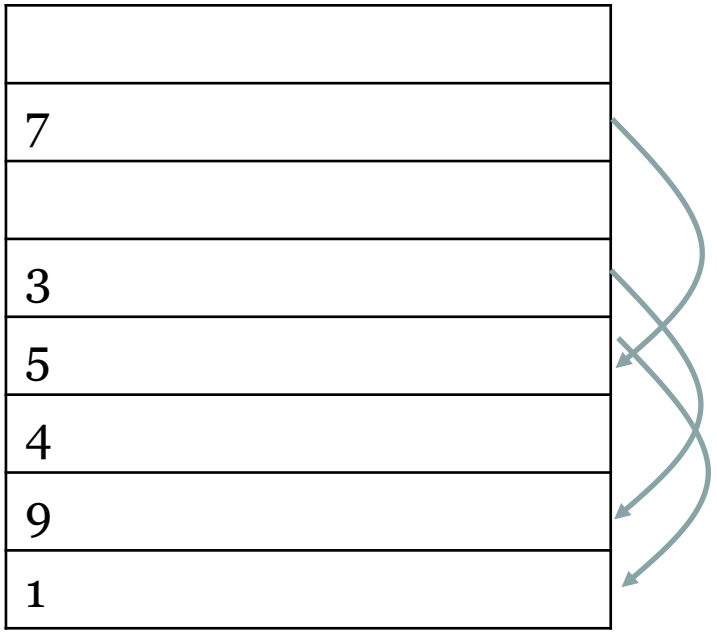
Anmerkung:
Es müssen nicht unbedingt Pointer verwendet werden, da ja nur Array-Indizes benötigt werden. Ein entsprechend großer integraler Datentyp kann dann auch zur Markierung von Positionen verwendet werden. Etwa -1: Ende der verketteten Liste, -2: frei, …

Einfügen

Keine Kollision -> Wert einfach einfügen



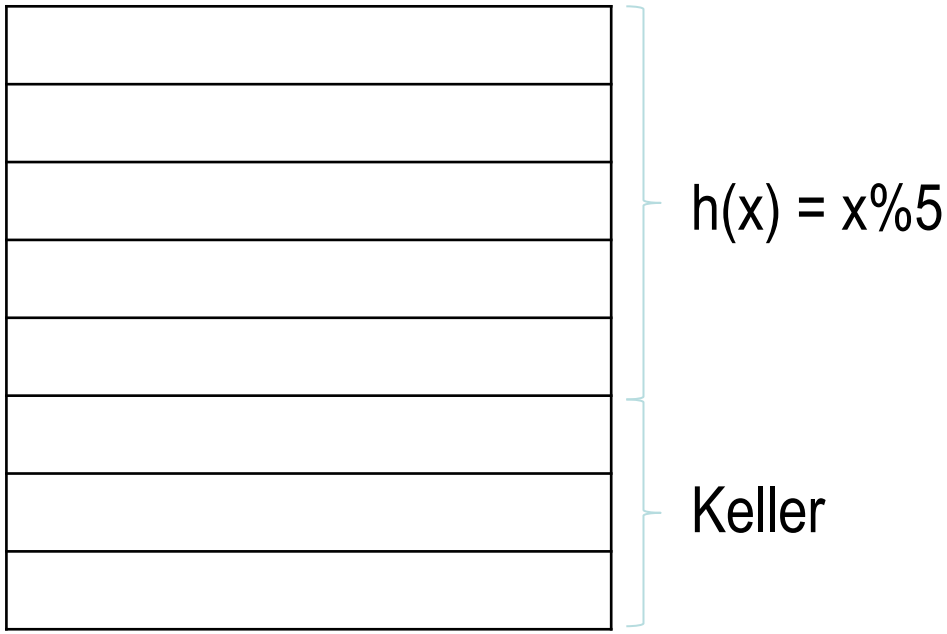
Bei Kollision wird der Wert in die letzte freie Tabellenposition eingetragen und ans Ende der Liste, die von der Hashadresse ausgeht, gehängt (LICH)



Late insertion coalesced hashing vereinfacht das Löschen. Varianten sind early insertion coalesced hashing (EICH) und varied insertion coalesced hashing (VICH). Wir beschränken uns hier auf LICH.

Keller

Eine simple Optimierung stellt die Verwendung eines Kellers dar. Dabei wird das Array größer dimensioniert, als der Wertebereich der Hashfunktion ist. Der durch die Hashfunktion nicht erreichbare Teil des Arrays wird als Keller bezeichnet.



Die optimale Kellergröße ist abhängig vom Belegungsgrad (load factor) der Tabelle. Die mathematische Analyse zeigt, dass für einen weiten Bereich an Belegungsgraden die optimale Kellergröße K=N*0.1628 ist. Dabei ist N die Größe des Arrays ohne Keller.

Um nicht immer die gesamte Tabelle nach einer leeren Position durchsuchen zu müssen, empfiehlt sich die Verwendung eines "Zeigers" auf die nächste freie Position, der das Array zyklisch vom höchsten Index beginnend durchläuft.

Suchen

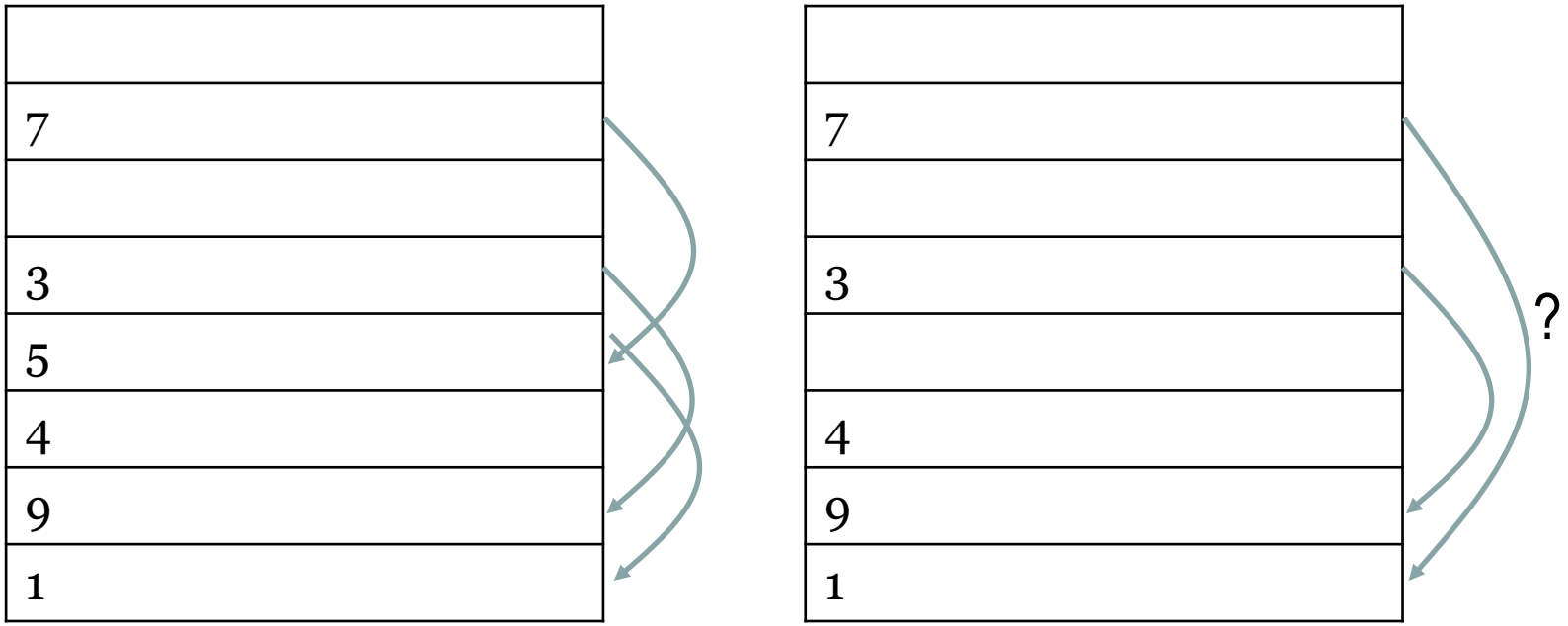
Trivial.

Suche an der Position beginnen, an die das gesuchte Element hasht. Dann entlang der Kette fortsetzen. Wird das Ende der Kette erreicht, ist das Element nicht in der Tabelle vorhanden.

Löschen

Ist das zu löschende Element nicht Teil einer Kette, wird die Position einfach als frei markiert.

Bei Elementen, die Teil einer Kette sind, ist zu beachten, dass Werte, die genau an die frei werdende Position hashen, in der Überlaufliste nicht mehr gefunden würden.



Man könnte die Position als wiederfrei markieren, das führt aber eventuell zu Problemen bei späteren Einfügeoperationen.

Löschen (2)

Simples Verfahren: Alle Werte in der Kollisionskette nach dem zu löschenden Element neu einfügen (für alle Einfügeverfahren möglich).

Etwas effizienter (nur für LICH):

Wir suchen den zu löschenden Wert und merken uns bei der Suche immer den jeweiligen Vorgänger. (Für den ersten Wert in der Kette ist der Vorgänger leer).

Wurde der Wert nicht gefunden, return.

Der Wert wurde an einer Position pos gefunden und der Vorgänger pre ist bekannt.

Wenn pre nicht leer ist, dann kappe den Link von pre (pre ist dann das Ende der Kette, in der es vorkommt).

Merken des Nachfolgers von pos in einer Variablen kette (das ist quasi die Kette aller Elemente, die an dem zu löschenden hängen)

pos löschen (status auf leer setzen, kein Nachfolger).

Damit haben wir eine Lücke in der Tabelle an der Position pos erzeugt.

Wir durchlaufen nun alle Elemente von kette.

Für jedes Element berechnen wir den Hashwert h

Es gibt zwei Fälle:

h trifft die Lücke:

Wir kopieren das Element von seiner aktuellen Position in die Lücke (diese hat keinen Nachfolger)

Dadurch entsteht eine neue Lücke an dieser aktuellen Position. Diese wird wieder geleert (Inhalt löschen, kein Nachfolger)

h trifft eines der zuvor schon abgearbeiteten Elemente aus unserer Kette

Das Element bleibt an seiner aktuellen Posiiton und wird dann einfach an das Ende der Kette, die von diesem getroffenen Element ausgeht, gehängt. Die Kette hört bei dem angehängten (aktuellen) Element auf.

Am Ende bleibt eine Lücke, die schon ordnungsgemäß geleert worden sein sollte. Das ist quasi die neue leere Stelle, die durch das Löschen erzeugt wurde.