HA Fachgebiet Architektur eingebetteter Systeme
Rechnerorganisation – Hausaufgabe WS19/20

AES Embedded Systems Architecture

Hinweise:

- Ihre Lösung muss unbedingt in den **vorgegebenen** Dateien an den **markierten** Stellen erfolgen. Sie erhalten für jede Aufgabe **separate** Vorgabedateien.
- Plagiate führen zum Nichtbestehen des gesamten Moduls.
- Ihr Code muss kommentiert sein, so dass die einzelnen Schritte nachvollziehbar sind.
- Die Abgabe erfolgt über ISIS. Laden Sie die drei Abgabedateien einzeln hoch.
- Tragen Sie Ihren Namen und Ihre Matrikelnummer in die Platzhalter in den Vorgaben ein.
- Ihre Lösung muss auch mit anderen Eingabewerten funktionieren, in MARS ausgeführt werden können und die Registerkonventionen erfüllen.

1. Luhn-Algorithmus (5 Punkte)

Bei der Übertragung von Daten können Übertragungsfehler auftreten. Um diese Fehler erkennen zu können werden Prüfsummen verwendet, die an die eigentliche Übertragung angehängt werden.

Auch Kreditkartennummern sind mit einer Prüfsumme gegen Fehler bei der Eingabe abgesichert. Der verwendete Algorithmus ist der *Luhn-Algorithmus*. Die letzte Ziffer der Kreditkartennummer entspricht einer Prüfziffer. Sie ist so gewählt, dass das Ergebnis des Luhn-Algorithmus über die gesamte Kreditkartennummer stets `0` beträgt. Sie finden weitere Informationen in der Wikipedia [1].

Implementieren Sie den Luhn-Algorithmus zwischen den vorgegebenen Platzhaltern in der Datei *aufgabe_1.s* so, dass eine gegebene 16-stellige Kreditkartennummer auf Korrektheit überprüft werden kann. Achten Sie darauf, dass diese als String, also als Array von einzelnen Characters im Speicher liegt. Leerzeichen sollen übersprungen werden. Bei anderen Zeichen (bspw. Buchstaben) soll abgebrochen werden und `-1` zurückgegeben werden (Funktionsrümpfe sind vorgegeben und müssen verwendet werden).

Teilschritte:

- Betrachten Sie den Pseudocode für den Luhn-Algorithmus.
- Überlegen Sie sich, welche Variablen Sie benötigen.
- Implementieren Sie die Schleife, die über die Zeichenkette iteriert. Die Zeichenkette ist nullterminiert. Laden Sie die einzelnen Zeichen.
- Überprüfen Sie die einzelnen Zeichen auf Validität (nur Ziffern/Leerzeichen erlaubt).
- Implementieren Sie die Gewichtung (x1/x2 der einzelnen Ziffern).
- Implementieren Sie den restlichen Teil der Aufgabe, die Division durch 10, und geben Sie den Rest zurück.

[1] `https://de.wikipedia.org/wiki/Luhn-Algorithmus`

2. Hashfunktion `H3` (7 Punkte)

Um einen größeren Wertebereich auf einen kleineren Wertebereich abzubilden, werden Hashfunktionen verwendet. Eine einfache Hashfunktion ist die Modulo-Operation.

In dieser Aufgabe sollen Sie die Hashfunktionen der Klasse `H3` implementieren. Diese bestehen aus einer einzelnen Hashfunktion, deren Verhalten durch einen Parameter (Seed, Startwert) verändert wird.

Abhängig vom Seed wird auf ein Array mit zufällig generierten Zahlen `q` zugegriffen (in der Vorgabe bereits angelegt). Der Seed gibt die Startposition in `q` an. Indem 32 andere Zahlen in `q` gewählt werden, kann das Verhalten weiter angepasst werden. Für Seed 0 werden die ersten 32 Werte aus `q` benutzt, für Seed 1 die nächsten 32, usw. Der zweite Parameter Schlüssel (key) wird in Binärdarstellung betrachtet: Jedes Bit aus dem Key entspricht einem Eintrag in `q`. Wenn der Schlüssel $32\,\mathrm{bit}$ groß ist, hat `q` 32 Einträge. Wenn ein Bit des Schlüssels auf 1 gesetzt ist, wird der entsprechende Wert aus `q` geladen. Alle geladenen Einträge werden nun ver-`x`-odert, sodass ein einzelner Hashwert entsteht.

Implementieren Sie die Hashfunktion zwischen den vorgegebenen Platzhaltern in der Datei *aufgabe_2.s*!

Beispiel für Berechnung des Hashwertes:

$$q := [110, 001, 111, 101, 011, 100, 001, 000]$$

$$\begin{aligned} h_q(53) &= h_q(00110101) \\ &= q(0) \oplus q(2) \oplus q(4) \oplus q(5) \\ &= 110 \oplus 111 \oplus 011 \oplus 100 \\ &= 110 = 6 \text{ (decimal)} \end{aligned}$$

Das Array `q` hat in diesem Beispiel 8 Werte, dies bedeutet, dass wir nur Schlüssel mit 8 Bit betrachten. Zudem sind die Werte im Array `q` alle 3 Bit lang, dies bedeutet, dass der Hashwert im Bereich zwischen 0 und 7 liegt. Da beim Schlüssel die Bits 0, 2, 4 und 5 gesetzt sind, werden die korrespondierenden Werte (bei Index 0 anfangend) aus `q` geladen und dann mit `xor` verknüpft. So erhält man den Hashwert 6.

Um zwei Hashfunktionen zu implementieren wäre hier das Array `q` doppelt so groß, also 16 Einträge und die Hashfunktion mit dem Seed 1 würde hier mit den Indizes 8-15 korrespondieren.

Hinweis:

In der Vorgabe finden Sie einige Hashwerte bereits vorberechnet und als Kommentar. Diese können Sie nutzen, um Ihre Implementierung zu verifizieren.

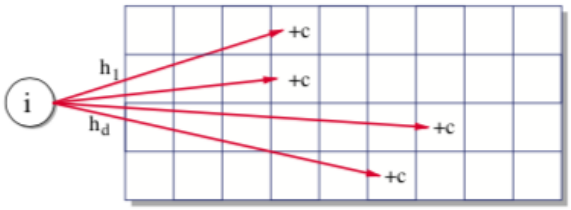
3. Count-Min-Sketch (8 Punkte)

Count-Min-Sketch ist eine Datenstruktur zum Speichern von Häufigkeiten. Die Struktur besteht aus einem zweidimensionalen Array, das durch Breite und Höhe beschrieben wird.

In dieser Aufgabe sollen Sie die Insert- und Update-Funktion implementieren.

3.1 Insert-Funktion für einen Schlüssel (4 Teilpunkte)

Soll ein Schlüssel (key) eingefügt werden, so wird pro Reihe (Zeile) des 2D-Arrays ein Eintrag inkrementiert. Die Spalte des Eintrags liefert die Hashfunktion. Pro Reihe wird die Hashfunktion mit dem Reihenindex als Seed verwendet.



Rufen Sie hierzu Ihre zuvor implementierten Hashfunktionen `H3` (Aufgabe 2) auf, beachten Sie auch die Hinweise unterhalb dieser Aufgabe!

Implementieren Sie Ihre Lösung zwischen den vorgegebenen Platzhaltern der Funktion `insert` in der Datei *aufgabe_3.s*. Beachten Sie die Kommentare in der Vorgabedatei, die Teilschritte und die Hinweise am Ende!

Teilschritte:

- Schleife über alle Reihen des 2D-Arrays
- Berechnung des Spaltenindex mit `H3`
- Laden des zu erhöhenden Wertes aus dem Count-Min Sketch
- Inkrementieren und Zurückspeichern des jeweiligen Eintrag

3.2 Update-Funktion für eine Menge an Schlüsseln (4 Teilpunkte)

Diese Funktion bekommt ein Array an Schlüsseln übergeben. Die Schlüssel sollen mit der Funktion aus dem Aufgabenteil 1 eingefügt werden.

Implementieren Sie Ihre Lösung zwischen den vorgegebenen Platzhaltern der Funktion `update` in der Datei *aufgabe_3.s*. Beachten Sie hierzu die Kommentare in der Vorgabedatei!

Hinweise:

Falls Sie Aufgabe 2 nicht lösen konnten, lassen Sie den Funktionsrumpf von `H3` so wie in der Vorgabe gegeben. So können Sie davon unabhängig trotzdem diese Aufgabe lösen.

Um die Insert-Funktion zu testen, finden Sie bereits Befehle in der Update-Funktion vorimplementiert, die den ersten Key laden und `insert` aufrufen. Diese müssen Sie für Ihre eigene Implementierung von `update` auskommentieren.

Mehrdimensionale Arrays werden im Speicher stets linear dargestellt. Überlegen Sie sich zuerst, wie Sie auf einzelne Koordinaten zugreifen können und wie sich die Position berechnet.

Sie können Ihre Implementierung mittels der `print_sketch`-Funktion verifizieren.