Gruppe **14**
Datenstrukturen und Algorithmen

Phil Pützstück, 377247
Benedikt Gerlach, 376944
Sebastian Hackenberg, 377550

Hausaufgabe 8

Aufgabe 1

Eingabearray `[3,6,2,4,5,5,0,2]`.
Histogramm `[1,0,2,1,1,2,1]`.
Positionsarray `[1,1,3,4,5,7,8]`.
Ergebnisarray:

```
[0,0,0,0,0,0,0,0] -> [0,0,2,0,0,0,0,0] -> [0,0,2,0,0,0,0,0] -> [0,0,2,0,0,0,5,0]
[0,0,2,0,0,5,5,0] -> [0,0,2,0,4,5,5,0] -> [0,2,2,0,4,5,5,0] -> [0,2,2,0,4,5,5,6]
[0,2,2,3,4,5,5,6]
```

Aufgabe 2

a) Wir berechnen der Erwartungswert der Anzahl der Versuche um einen freien Platz durch Zufall zu wählen, wobei $1-\alpha$ die Wkeit ist, einen freien Platz zu finden. Es ist $0 \leq \alpha < 1$, also:

$$A_{\text{ins}}(\alpha) = \sum_{i=0}^{\infty} i\alpha^{i-1}(1-\alpha) = \frac{1-\alpha}{\alpha}\sum_{i=0}^{\infty} i\alpha^{i} = \frac{1-\alpha}{\alpha}\cdot\frac{\alpha}{(\alpha-1)^2} = \frac{1}{1-\alpha}$$

b) Dies ist analog zu Teilaufgabe a). Wir betrachten $0 < \alpha = 1 - \frac{p}{m} < 1$, wo p die Anzahl der Vorkommnisse eines Schlüssels k in der Hashtabelle ist. Das finden des Schlüssels ist dann analog zum finden eines freien Platzes beim Einfügen, da $1-(1-\frac{p}{m})$ dann die Wkeit darstellt den Schlüssel zu finden, und $1-\frac{p}{m} = \alpha$ die Wkeit bei der wir weiter suchen müssen.
Mit a) folgt also:

$$A_{\text{find}}(p) = A_{\text{ins}}(1-\frac{p}{m}) = \frac{1}{1-(1-\frac{p}{m})} = \frac{m}{p}.$$

c) Die erfolglose Suche wird nicht terminieren, da im angegebenen Algorithmus einfach immer der erste Schritt wiederholt wird wenn wir einen Schlüssel k im letzten Versuch nicht gefunden haben. Also $A_{\neg\text{find}} = \infty$.

d) Es ist grundsätzlich nicht so einfach (bzw. effizient), gute zufällige zahlen zu generieren. Ferner ist weder eine Gleichverteilung noch Surjektivität garantiert, jedoch sehr wahrscheinlich auf Dauer. Die Schlüssel werden auch sehr wahrscheinlich gut breit verteilt. Jedoch ist diese Methode problematisch, in der Hinsicht, dass stets die Möglichkeit besteht extrem lange (oder auch unendlich oft) zu benötigen einen Schlüssel einzufügen, oder zu Suchen. Vorallem der letze Fall der erfolglosen Suche stellt dies dar. Bei einer Hashtabellengröße von 10 und einem einmaligem Schlüssel kann es auch gut schonmal sein 30 Anläufe zu brauchen um diesen zu finden, während bei den anderen Verfahren z.B. linearer sondierrung dies niemals länger als 10 Schritte dauern kann. Insgesamt ist der Worstcase nicht vertretbar.

Aufgabe 3

Die Funktion $f : [1, 100] \to [1, 10], x \mapsto \lfloor \frac{10}{x} \rfloor$ ist grundsätzlich einfach zu berechnen, wir nehmen schließlich oft an, dass Operationen wie $\div$ konstante Zeit benötigen. Weiter sind keine zu großen Zahlen im Spiel. f ist jedoch nicht surjektiv, da f monoton fallend ist und $f(1) = 10, f(2) = 5$ gilt, also z.B. $9, 8, 7, 6$ nicht getroffen werden. Weiter ist f bei weitem nicht gleichverteilt, wir haben $\forall x \in [11, 100] : f(x) = 0$, also ca 90% der möglichen Eingaben werden auf 0 abegebildet. Letztlich werden auch ähnliche Schlüssel auf ähnliche Bereiche verteil, insgesamt ist die Funktion nicht sehr gut geeignet.

Die Funktion $g : \mathbb{N} \to \mathbb{Z}/101\mathbb{Z}, x \mapsto 2^x \mod 101$ ist nicht so einfach zu berechnen wie die anderen der Liste, jedoch ist eine Zweierpotenz letzendlich nur ein Left-shift, also auch nicht sehr kostspielig. g ist nicht surjektiv, da es kein $x \in \mathbb{N}$ mit $2^x \mod 101 = 0 \in \mathbb{Z}/101\mathbb{Z}$ gibt. Da wir $2^{100} \mod 101 = 1$ haben, gilt stets $2^{x+100y} \equiv 2^x \mod 101$ für $x \in [0, 99], z \in \mathbb{N}$. Damit haben wir (bis auf 0) eine perfekte Gleichverteilung über $\mathbb{N}$. Hinzukommend werden aufeinanderfolgende Werte breit verteilt, beispielsweise gilt $g(22) = 77, g(23) = 53$. Insgesamt ist g eine geeignete Hashfunktion.

Die Funktion $h : [0, 100] \to [0, 10], x \mapsto x \mod 11$ ist sehr einfach zu berechnen. Ferner ist es surjektiv, da z.b. $h_{[0,10]} = \mathrm{id}_{[0,10]}$. Offensichtlich haben wir für $x \in [2, 10]$ stets 9 werte in $[0, 100]$ welche unter h auf x abgebildet werden, für $x = 0$ oder $x = 1$ gibt es 10. Das ist eine ausgeglichene Gleichverteilung über $[0, 100]$. Jedoch werden Schlüssel nicht sehr breit verteilt. Ist $x \in [0, 100]$ Vielfaches von 10, so wird der Nachfolger von x auch auf den Nachfolger des Bildes von x unter h abgebildet. Grundsätzlich ist diese letzte Eigenschaft eine der wichtigsten, weswegen Ich diese Funktion trotz ihrer restlichen Qualitäten nicht als gute Hashfunktion betiteln würde.

Die Funktion $i : \mathbb{N} \to [0, 50], x \mapsto \lfloor \frac{x}{2} \rfloor \mod 51$ ist (analog zu f) wahrscheinlich in konstanter Zeit berechnbar. Weiter ist sie surjektiv, denn es ist $i(\{2x \mid x \in [0, 50]\}) = [0, 50]$. Ferner haben wir eine ausgeglichene Gleichverteilung über ganz $\mathbb{N}$, da für $x \in [0, 50]$ immer
$\{y \in \mathbb{N} | y = 2(x + 51z) \vee y = 2(x + 51z) + 1, z \in \mathbb{N}\}$ das Urbild von x darstellt. Jedoch ist wieder analog zu h eine eher schlechte Verteilung der Werte gegeben. Die Folge der Werte über $\mathbb{N}$ folgt dem Schema $0, 0, 1, 1, 2, 2, \cdots, 50, 50, 0, 0, 1, 1, \cdots$. Aus gleichem Grund wie für h würde ich also i nicht als gute Hashfunktion bezeichnen.

Aufgabe 4

$c = 0.01, m = 11$.

```
[5, , , , , , , , , , ] -> [5, ,21, , , , , , , , ] -> [5, ,21,23, , , , , , , ]
[5,17,21,23, , , , , , ] -> [5,17,21,23,11, , , , , ]
[5,17,21,23,11,7, , , , ] -> [5,17,21,23,11,7,1, , , , ]
```

Aufgabe 5

$c = 0.01, c_1 = 2, c_2 = 1, m = 11$.

```
[5, , , , , , , , , , ] -> [5, ,21, , , , , , , , ] -> [5, ,21, , ,23, , , , , ]
[5,17,21, , ,23, , , , ] -> [5,17,21, ,11,23, , , , , ]
[5,17,21,7,11,23, , , , , ] -> [5,17,21,7,11,23, , ,1, , ]
```

Aufgabe 6

```
// Angenommen alles wird abgerundet zum nächsten Int

void A(int n) {
    for (int i = 1; i <= 2*n/3+1; ++i)
            print(".");
}

void B(int n) {
    for(int i = 1; i <= n*n + n*n*n; ++i)
        print(".");
}

void C(int n) {
    int acc1 = 1, acc2 = 1;
    for (int i = 1; i <= n; ++i) {
        acc1 *= 3;
        acc2 *= n;
    }

    for (int i = 1; i <= acc1+acc2; ++i)
        print(".");
}

void E(int n) {
    int acc = 0;
    for (int i = 0; i < 2; ++i) {
        while (n > 1) {
            n /= 2;
            ++acc;
        }
        n = acc; acc = 0;
    }

    for (int i = 1; i <= n; ++i)
        print(".");
}

void F(int n) {
    int acc = 0;
    while (n > 1) {
        n /= 2;
        ++acc;
    }

    for (int i = 1; i*i*i <= acc; ++i)
        print(".");
}
```