Aufgabensammlung ADS-Repetitorium WS 24/25

Probabilistische Analyse – Amortisierte Analyse – Datenstrukturen Augmentieren

Aufgabe 1: Indikatorzufallsvariablen und Maximum

Sei A ein Array paarweise unterschiedlicher ganzzahliger Zahlen der Länge n. Algorithmus 1 ermittelt die größte Zahl in A.

Algorithmus 1:

```
1 m = A[1]
2 for i = 2 to n do
3     if A[i] > m then
4         m = A[i]
5     return m
```

(a) Wie häufig wird im worst-case Zeile 4 ausgeführt?

> **Lösung:** Falls A aufsteigend sortiert ist, wird Zeile 4 in jeder Iteration ausgeführt. Zeile 4 wird also $n-1$ mal ausgeführt.

(b) Definieren Sie eine Zufallsvariable X, die die Anzahl der Aktualisierungen von m angibt und definieren Sie zusätzlich Indikatorzufallsvariablen, mit der X gezählt werden kann.

> **Lösung:** Sei $X \in \{1, \ldots, n\}$ die Zufallsvariable, die angibt, wie häufig m aktualisiert wird und seien X_i Indikatorzufallsvariablen, die folgendermaßen definiert sind:
>
> $$X_i = \begin{cases} 1 & m \text{ wird in der } i\text{-ten Iteration aktualisiert} \\ 0 & \text{sonst.} \end{cases}$$
>
> Weil m immer in Zeile 1 aktualisiert wird, ist $X = 1 + X_2 + X_3 + \cdots + X_n$

(c) Berechnen Sie den Erwartungswert von X!

> **Lösung:** A enthält paarweise unterschiedliche Zahlen, somit ist die Wahrscheinlichkeit, dass $A[i]$ im Teilfeld $A[1..i]$ das Maximum ist $\mathbf{Pr}[X_i = 1] = 1/i$. Somit ist der Erwartungswert von X:
>
> $$\begin{aligned} \mathbf{E}[X] &= \mathbf{E}[1 + X_2 + X_3 + \cdots + X_n] \\ &= 1 + \sum_{i=2}^{n} \mathbf{E}[X_i] = 1 + \sum_{i=2}^{n} \mathbf{Pr}[X_i = 1] = 1 + \sum_{i=2}^{n} \frac{1}{i} = \mathcal{H}_n \in \mathcal{O}(\log n) \end{aligned}$$

Aufgabe 2: Algorithmen und Zufall

Bestimmen Sie den Erwartungswert der Ausgabe von Algorithmus 2.

Algorithmus 2:

```
1 s = 0
2 for i = 1 to n do
3     /* Random(a, b) gibt zufällig und gleichverteilt einen Wert zwischen a und b aus. */
4     if Random(1, 2 · i) ≤ i then
5         s = s + i
6 return s
```

Lösung: Sei X die Zufallsvariable, die den Rückgabewert repräsentiert. Sei X_i eine Indikatorzufallsvariable, die angibt, ob in Iteration i die Zeile 5 ausgeführt ($X_i = 1$) wird. Dann können wir X folgendermaßen bestimmen:

$$\mathbf{E}[X] = \mathbf{E}\left[\sum_{i=1}^{n} X_i \cdot i\right] = \sum_{i=1}^{n} \mathbf{E}[X_i] \cdot i$$

Nachdem X_i eine Indikatorzufallsvariable ist, ist $\mathbf{E}[X_i] = \mathbf{Pr}[X_i = 1] = i/(2i) = 1/2$. $\mathbf{E}[X]$ ist also

$$\mathbf{E}[X] = \sum_{i=1}^{n} \mathbf{E}[X_i] \cdot i = \sum_{i=1}^{n} \frac{1}{2} \cdot i = \frac{1}{2} \sum_{i=1}^{n} i = \frac{n(n+1)}{4}.$$

Aufgabe 3: Paranoia

Gegeben sei folgender Algorithmus 3. Dabei ist das Set A eine endliche Menge von ganzzahligen Elementen der Länge n. Sie dürfen davon ausgehen, dass Einfüge-, Lösch- und Suchoperationen in $\mathcal{O}(1)$ möglich sind.

Algorithmus 3:

```
1  ParanoidMaximum(int[] A, ℓ = 1)
2      if A.length = ℓ then
3          return A[ℓ]
4      swap(A, ℓ, Random(ℓ, n))
5      a = A[ℓ]
6      b = ParanoidMaximum(A, ℓ + 1)
7      if b > a then
8          return b
9      else
10         for i = ℓ + 1 to A.length do
11             if A[i] > a then
12                 a = A[i]
13         return a
```

(a) Was berechnet Algorithmus 3 und ist die for-Schleife in der else Bedingung in den Zeilen 8-11 notwendig?

Lösung: Der Algorithmus berechnet das Maximum von $A[\ell..n]$. Nein, die else Bedingung in den Zeilen 8-11 ist nicht notwendig, denn falls $a \geq b$ ist, so ist a das Maximum in $A[\ell..n]$, weil b bereits das Maximum in $A[\ell + 1..n]$ ist.

(b) Berechnen Sie die worst-case Laufzeit des Algorithmus, indem Sie eine Rekursionsgleichung aufstellen und diese mittels Substitionsmethode beweisen!
Hinweis: Hier eignet es sich in der Rekursionsgleichung die Anzahl der Vergleiche zu zählen.

Lösung: Der worst-case tritt ein, wenn wir in jedem Rekursionsaufrauf in die else Bedingung in den Zeilen 8-11 kommen. Wir stellen eine Rekursionsgleichung für die Anzahl der Vergleiche auf, nachdem diese proportional zu der gesamten Laufzeit des Algorithmus sind:

$$T(n) = \begin{cases} 1 & n = 1 \\ n + 1 + T(n-1) & \text{else.} \end{cases}$$

Wir behaupten, dass $T \in \mathcal{O}(n^2)$. Wir beweisen dies mittels Induktion über n.

Induktionsanfang $(n = 1)$**:** Hier sind wir auch im Basisfall der Rekursionsgleichung und es gilt $1 \leq c \cdot 1^2$ für jede Konstante $c \geq 1$.

Induktionshypothese: Für alle $k < n$ gelte $T(k) \leq c \cdot k^2$ für eine Konstante $c > 0$ und $n \geq 1$.

Induktionsschritt $(n \to n+1)$**:** Es gilt

$$\begin{aligned} T(n) &= (n+1) + T(n-1) \\ &\overset{\text{I.H.}}{\leq} (n+1) + c \cdot (n-1)^2 \\ &= n + 1 + cn^2 - 2c \cdot n + c \\ &= c \cdot n^2 + (1-2c) \cdot n + 1 + c \\ &\leq c \cdot n^2 + (1-2c) \cdot n + (1+c) \cdot n \quad \text{denn } n \geq 1 \text{ und } c > 0 \\ &= c \cdot n^2 + (2-c) \cdot n \\ &\leq c \cdot n^2 \quad \text{falls } c = 2 \end{aligned}$$

Somit gilt die Behauptung, dass $T(n) \leq c \cdot n^2$ für eine Konstante c und damit ist $T \in \mathcal{O}(n^2)$.

Wir nehmen nun im Folgenden an, dass A paarweise verschiedene Zahlen enthält. Weiterhin ist die Anordnung der Zahlen in A zufällig gleichverteilt.

(c) Definieren Sie eine geeignete Zufallsvariable, die beide Fälle in den Zeilen 6-11 in Abhängigkeit der Rekursionstiefe i angibt!

Lösung: Sei X_i eine Indikatorzufallsvariable, die in der i-ten Rekursion angibt, ob wir Zeile 7 oder die Zeilen 8-11 ausführen:

$$X_i = \begin{cases} 1, & a \geq b \\ 0 & \text{else.} \end{cases}$$

(d) Geben Sie eine Formel für die erwartete Laufzeit mit Hilfe der definierten Zufallsvariable aus Teilaufgabe (c) an!

Lösung: In jeder Rekursionsstufe wird die Funktion mit einem Element weniger aufgerufen. Nachdem $X_i \in \{0, 1\}$ angibt, ob wir Zeile 7 ausführen $(X_i = 0)$ oder Zeilen 8-11 ausführen $(X_i = 1)$,

können wir die erwartete Laufzeit mit der Linearität des Erwartungswerts ausdrücken:

$$\begin{aligned}\mathbf{E}[T(n)] &= \mathbf{E}\left[\Theta(1) + \sum_{i=2}^{n}(1 - X_i) \cdot \Theta(1) + X_i \cdot \Theta(n - i + 1)\right] \\ &= \Theta(1) + \sum_{i=2}^{n} \mathbf{E}[(1 - X_i)] \cdot \Theta(1) + \mathbf{E}[X_i] \cdot \Theta(n - i + 1) \\ &= \mathcal{O}(n) + \sum_{i=2}^{n} \mathbf{E}[X_i] \cdot \Theta(n - i + 1)\end{aligned}$$

Hierbei entspricht $\Theta(1)$ dem Basisfall und die Summe dem rekursiven Fall. Nachdem der Fall für Zeile 7 insgesamt höchstens Linearzeit in Anspruch nimmt, schätzen wir dies mit $\mathcal{O}(n)$ ab und konzentrieren uns auf den aufwändigen Fall.

(e) Mit welcher Wahrscheinlichkeit tritt der Fall in den Zeilen 8-11 auf? Was ist der Erwartungswert Ihrer Zufallsvariable aus Teilaufgabe (c)?

Lösung: Weil A keine Duplikate enthält, hängt die Wahrscheinlichkeit, dass a das größte Element ist, einzig von der Länge von A ab. Somit ist $\mathbf{Pr}[X_i = 1] = 1/(n - i + 1)$. Nachdem X_i eine Indikatorzufallsvariable ist, ist $\mathbf{E}[X_i] = \mathbf{Pr}[X_i = 1]$.

(f) Berechnen Sie nun die erwartete Laufzeit mit Hilfe der Formel in Teilaufgabe (d) und der Wahrscheinlichkeit aus Teilaufgabe (e).

Lösung: Wir müssen nur noch die Formeln zusammenführen und erhalten:

$$\begin{aligned}\mathbf{E}[T(n)] &= \mathcal{O}(n) + \sum_{i=2}^{n} \mathbf{E}[X_i] \cdot \Theta(n - i + 1) \\ &= \mathcal{O}(n) + \sum_{i=2}^{n} \mathbf{Pr}[X_i = 1] \cdot \Theta(n - i + 1) \\ &= \mathcal{O}(n) + \sum_{i=2}^{n} \frac{1}{n - i + 1} \cdot \Theta(n - i + 1) \\ &= \mathcal{O}(n) + \sum_{i=2}^{n} \mathcal{O}(1) = \mathcal{O}(n)\end{aligned}$$

Die erwartete Laufzeit ist also $\mathcal{O}(n)$.

Aufgabe 4: Dynamic Table

In der Vorlesung sind wir häufig davon ausgegangen, dass unsere elementaren Datenstrukturen, wie eine Schlange oder ein Stapel, statischen Speicher haben. Das heißt, beim Anlegen der Datenstruktur geben wir bereits vor, wie viele Elemente die Struktur insgesamt speichern kann. Diese Größe ist nicht immer zu Beginn bekannt. Im Folgenden untersuchen wir, wie wir den Speicher einer Datenstruktur dynamisch vergrößern bzw. verkleinern können, sodass wir immer genug Platz haben aber gleichzeitig nicht unnötig viel Speicher verschwenden. Eine amortisierte Analyse wird zeigen, dass Insertion und Deletion dennoch $\mathcal{O}(1)$ Zeit benötigt.

Sei zunächst T eine Tabelle, die lediglich die Operation Insert unterstützt. Das Attribut T.table soll einen Pointer auf einen Speicherblock enthalten (ein Array). T.num ist die Anzahl der Elemente, die derzeitig in T enthalten sind und T.size ist die Größe des Speicherblocks. Zu Beginn gilt $T.\text{num} = T.\text{size} = 0$.

(a) Eine herkömmliche Heuristik beim Expandieren einer Tabelle ist, den vorherigen Speicher zu verdoppeln wenn die Tabelle voll ist. Mit anderen Worten: Wenn nur Insert Operationen unterstützt werden, sind immer mindestens T.size/2 Elemente in T enthalten. Schreiben Sie einen Pseudocode der die Operation Insert implementiert.

Lösung: Pseudocode:

Algorithmus 4: Insert Operation von T, die ein neues Element x einfügt

```
1  if T.size == 0 then
2      allokiere Speicher für T.table der Länge 1
3      T.size = 1
4  if T.num == T.size then
5      allokiere neuen Speicher A der Länge 2 · T.size
6      Füge alle Elemente von T.table zu A hinzu
7      T.table = A
8      T.size = 2 · T.size
9  füge x zu T.table hinzu
10 T.num = T.num + 1
```

Dieser Pseudocode ist mehr pseudo als code. Man kann zum Beispiel Zeile 6 als For Loop formulieren. In der Lösung wurde T.table absichtlich abstrakt gehalten. Falls das als herkömmliches Array implementiert wurde, ist das in Ordnung.

(b) Was ist die *worst-case* Laufzeit einer einzigen Insert Operation von Teilaufgabe (a)?

Lösung: Sei n die Anzahl der Elemente in T. Im worst-case ist T.table voll und somit T.num == T.size, wodurch T.table verdoppelt wird. Dazu müssen alle n Elemente kopiert werden und das neue Element x hinzugefügt werden. Der Gesamtaufwand beträgt also $\mathcal{O}(n)$.

(c) Betrachten Sie nun eine Folge von n Insert Operationen. Zeigen Sie mit der Aggregationsmethode, dass die Insert Operation *amortisiert* $\mathcal{O}(1)$ Zeit benötigt.
Hinweis: Eine Tabelle verdoppelt sich bei jeder Expansion. Wie häufig kann eine Expansion bei n Insert Operationen auftreten?

Lösung: Die Tabelle wird nur selten expandiert: die i-te Operation hat nur eine Expansion zur Folge, falls $i-1$ eine Zweierpotenz ist! Definiere also die Kosten der i-ten Operation als

$$c_i = \begin{cases} i & \text{falls } i-1 \text{ eine Zweierpotenz} \\ 1 & \text{sonst.} \end{cases}$$

Die Gesamtkosten sind also

$$\sum_{i=1}^{n} c_i \leq n + \sum_{j=0}^{\lfloor \log n \rfloor} 2^j < n + 2n = 3n$$

Damit haben wir amortisierte Kosten von $3 \in \mathcal{O}(1)$ pro Insert Operation.

(d) Zeigen Sie mit Hilfe der Buchhaltermethode, dass die Insert Operation *amortisiert* $\mathcal{O}(1)$ Zeit beansprucht.

Lösung: Die Buchhaltermethode kann gleichzeitig eine Intuition dafür geben, wieso die Insert Operation in der Aggregationsmethode amortisierte Kosten von 3 erhalten hat: Jedes Element das hinzugefügt wird, bezahlt für drei Operationen: die eigenen Kosten durch das Hinzufügen, die eigenen Kosten für das Rüberkopieren von sich selbst in eine erweiterte Tabelle und außerdem übernimmt das Element die Kosten für die Kopie eines weiteren Elements. Da nach einer Expansion $T.\text{size}/2$ Elemente in T enthalten sind und eine Expansion erst stattfindet, wenn $T.\text{size} == T.\text{num}$, übernimmt jedes Element, welches nach der Expansion hinzugefügt wurde genau für ein vorheriges Element die Kosten für die Kopie. Dadurch machen wir nie Schulden.

(e) Zeigen Sie mit Hilfe der Potentialmethode, dass die Insert Operation *amortisiert* $\mathcal{O}(1)$ benötigt.
Hinweis: Entwicklen Sie eine Potentialmethode, die nach einer Expansion 0 ist und sich bis zur nächsten Expansion (pro Insert Operation) auflädt. Um wie viel muss sich die Potentialfunktion pro Operation aufladen?

Lösung: Wenn wir die Potentialmethode verwenden, müssen wir uns eine Potentialfunktion Φ definieren, die uns die freiverfügbare Arbeit angibt. Die verfügbare Arbeit, müssen wir für das Kopieren der Elemente verwenden. Ähnlich wie bei der Buchhaltermethode wollen wir, dass nach einer Expansion das Potential 0 ist, also wenn $T.\text{num} = T.\text{size}/2$. Während Elemente eingefügt werden, muss sich das Potential erhöhen, sodass das Potential nach weiteren $T.\text{size}/2$ Insert Operationen alle Kopien zahlen kann. Die Anzahl der Kopien ist dann $T.\text{num} = T.\text{size}$. Das heißt für $T.\text{size}/2$ Insert Operationen muss das Potential von 0 auf $T.\text{size}$ steigen. Pro Insert Operation sollte sich das Potential also um

$$\frac{T.\text{size}}{T.\text{size}/2} = 2$$

erhöhen. Wir können die Potentialfunktion also folgendermaßen definieren:

$$\Phi(T) = 2(T.\text{num} - T.\text{size}/2)$$

Die Funktion ist 0 gleich nach einer Expansion, also wenn $T.\text{num} == T.\text{size}/2$ und es steigt um zwei mit jeder Insertion Operation, bis sie $T.\text{size}$ groß ist. Nachdem unsere Tabelle immer mindestens $T.\text{num} \geq T.\text{size}/2$ groß ist, ist unsere Funktion immer ≥ 0 und wir kommen nie in die Mießen.
Die amortisierten Kosten einer Insert Operation falls keine Expansion stattfindet sind dann:

$$\hat{c}_i = c_i + \Delta\Phi = 1 + 2 = 3$$

Sei num_i die Anzahl der Elemente nach der i-ten Operation und sei size_i die Größe der Tabelle nach der i-ten Operation, d.h. es gilt $\Phi(D_{i-1}) = 2(\text{size}_{i-1} - \text{size}_{i-1}/2) = \text{size}_{i-1} = i - 1$. Gleich nach der Expansion ist das Potential 0 aber gleich danach wird das Element x hinzugefügt, weshalb $\Phi(D_i) = 2$ und wir für $\Delta\Phi = 2 - (i-1) = 3 - i$ erhalten. Wenn wir die Tabelle expandieren müssen, haben wir echte Kosten von i (Kopie von $i-1$ Elementen + das Einfügen von Element i). Die amortisierten Kosten sind also

$$\hat{c}_i = c_i + \Delta\Phi = i + (3 - i) = 3$$

Bisher haben wir uns lediglich die Insert Operation angeschaut. Nun betrachten wir auch die Delete Operation, die ein Element von unserer Tabelle löscht. Um nicht unnötig Speicher zu besetzen, wollen wir außerdem unsere Tabelle verkleinern, wenn $T.\text{num}$ einen bestimmten Schwellenwert unterschreitet. Diese Verkleinerung funktioniert analog zur Expansion. Falls der Schwellenwert unterschritten wird, wird ein kleinerer Speicher allokiert und die Elemente in diesen Speicher rüberkopiert, damit anschließend der vorherige Speicher freigegeben werden kann.

(f) Ein Kommilitone von Ihnen schlägt vor, dass der Schwellenwert für eine Verkleinerung bei $T.\text{num}/2$

sein sollte, dessen Unterschreitung in einer Halbierung der Tablle resultiert. Zeigen Sie anhand einer Beispielfolge, dass diese Strategie nicht zu amortisiert konstanten Laufzeiten für die Insert und Delete Operation führt.
Hinweis: Sie dürfen davon ausgehen, dass eine Verdopplung der Tabelle stattfindet, wenn die Tabelle voll ist, wie in den vorherigen Teilaufgaben der Fall war.

Lösung: Sei n eine Zweierpotenz und T eine Tabelle der Größe $n/2$. Die ersten $n/2$ Operationen sind Insert Operationen, sodass am Ende dieser Operationen $T.\text{num} = T.\text{size} = n/2$. Anschließend führen wir Operationen in der Folge Insert, Delete, Delete, Insert, Insert, ... aus.
Die erste Insertion Operationen hat eine Expansion zur Folge, sodass $T.\text{size} = n$. Die zwei aufeinanderfolgenden Delete Operationen führen zu einer Verkleinerung der Tabelle. Dies wiederholt sich bis zum Ende der Operationen. Insgesamt gibt es dann $\Theta(n)$ Expansionen bzw. Verkleinerungen, die jeweils $\Theta(n)$ Zeit benötigen, wodurch diese Folge von Operationen $\Theta(n^2)$ beansprucht, wodurch amortisierte Kosten von $\Theta(n)$ resultieren.

(g) Das Problem der Strategie des Kommilitonen war, dass nach einer Verkleinerung der Tabelle nicht genug Insertion Operationen stattfinden konnten, um für eine Expansion zu zahlen und auch umgekehrt nicht genug Delete Operationen stattfinden konnten, um für eine Verkleinerung zu zahlen. Zeigen Sie mit der Potentialmethode, dass folgende Strategie aufgeht: Wenn $T.\text{num} < T.\text{size}/4$, dann halbiere die Größe der Tabelle.

Lösung: Siehe CLRS, Amortisierte Analyse

Aufgabe 5: Monotoner Stapel

Wir wollen in einem Stapel eine Menge von aufsteigenden Zahlen verwalten. Dabei wurde die `Push` Operation folgendermaßen mittels Algorithmus 5 modifiziert:

Algorithmus 5:

```
1 PushMonotone(Stack S, int x)
2     y = S.Pop()
3     while y > x and S ≠ ∅ do
4         y = S.Pop()
5     S.Push(x)
```

(a) Was ist die worst-case Laufzeit von `PushMonotone`?

Lösung: Falls der Stack nur Elemente enthält, die größer sind als x, wird der gesamte Stapel geleert, bevor x eingefügt wird. Dementsprechend ist die worst-case Laufzeit linear in der Anzahl der Elemente im Stack.

(b) Zeigen Sie, dass `PushMonotone` bei einer Sequenz von n Operationen amortisiert konstant ist.

Lösung: In einem Stack kann ein Element nur eingefügt und gelöscht werden. Wenn wir jedem Element beim Einfügen Kosten von 2 geben, so benutzen wir Kosten von 1, um das Element einzufügen und Kosten von 1, um dieses Element später wieder zu entfernen. Nachdem ein Element erst eingefügt werden muss, bevor es entfernt werden kann, kommen wir nicht in die Miesen. Allen anderen Operationen geben wir konstante Kosten und erhalten folglich ingesamt über eine Sequenz von n Operationen maximale Kosten von $\mathcal{O}(n)$. Demnach ist `PushMonotone` amortisiert konstant.

Aufgabe 6: Leichte Klausuren

Jede ADS-Klausur wird von den Studierenden nach Ihrer Schwierigkeit bewertet. Die erste Klausur wurde im Semster 0 geschrieben, die letzte Klausur im Semester $n+1$. Jede Klausur hat einen eindeutigen Rang erhalten, sodass sich Bewertungen von 0 bis $n+1$ ergeben. Wir betrachten eine Klausur im Semester i als *leicht*, falls für deren Bewertung b_i gilt, dass $b_i < b_{i-1}$ und $b_i < b_{i+1}$ ist.

Wir nehmen an, dass jede mögliche Folge der Bewertungen gleich wahrscheinlich ist, also alle Permutationen von $(b_0, b_1, \ldots, b_n, b_{n+1})$ mit gleicher Wahrscheinlichkeit auftreten können. Als Studierender möchten Sie berechnen, wie viele leichte Klausuren Sie in n Semestern erwarten können.

(a) Sei K eine Zufallsvariable, die die Anzahl der leichten Klausuren in den Semestern 1 bis n beschreibt. Angenommen, Sie hätten eine Formel, die die Wahrscheinlichkeit $P(K = k)$ für $0 \leq k \leq n$ berechnet. Wie könnten Sie dann $E[K]$ für ein festes n ausrechnen?

Lösung: In diesem Fall könnten Sie die allgemeine Definition des Erwartungswerts verwenden: $E[K] = \sum_{k=0}^{n} k \cdot P(K = k)$.

(b) Im Allgemeinen ist es schwierig, eine Formel für $P(K = k)$ zu finden. Die Fragestellung können Sie aber trotzdem lösen, indem Sie mit Indikatorzufallsvariablen arbeiten. Definieren Sie eine Indikatorzufallsvariable K_i die einem Semester $0 < i < n+1$ den Wert 0 oder 1 zuweist, je nachdem, ob die Klausur einfach war oder nicht. Wir können Sie K in Abhängigkeit von K_i für $0 < i < n+1$ berechnen?

Lösung:

$$K_i = \begin{cases} 1 \text{ falls } b_i < b_{i-1} \text{ und } b_i < b_{i+1} \\ 0 \text{ sonst} \end{cases}$$

$$K = \sum_{i=1}^{n} K_i$$

(c) Geben Sie außerdem eine Formel an, mit der Sie $E[K]$ in Abhängigkeit von $E[K_i]$ für $0 < i < n+1$ berechnen können. Welche Gesetzmäßigkeit des Erwartungswerts machen Sie sich zu Nutze?

Lösung: Um $E[K]$ zu berechnen, bilden wir den Erwartungswert der Summe $E[K] = E[\sum_{i=1}^{n} K_i]$. Dank der *Linearität* des Erwartungswerts können wir auch erst alle Erwartungswerte berechnen und dann die Summe bilden $E[K] = \sum_{i=1}^{n} E[K_i]$.

(d) Nehmen Sie an, Sie kennen eine Formel, die $P(K_i = 1)$ für ein festes i berechnet. Geben Sie eine Formel an, mit der Sie $E[K_i]$ berechnen können. Formulieren Sie in Worten, was die Wahrscheinlichkeit $P(K_i = 1)$ ausdrückt.

Lösung: Auch hier können wir die allgemeine Definition des Erwartungswerts verwenden. Da K_i nur 0 oder 1 annehmen kann, besteht die Formel nur aus einem Term $E[K_i] = 1 \cdot P(K_i = 1) + 0 \cdot P(K_i = 0) = P(K_i = 1)$, da der Faktor 0 wegfällt. Die gegebene Wahrscheinlichkeit $P(K_i = 1)$ ist die Wahrscheinlichkeit, dass die Klausur im Semester i eine leichte Klausur war.

(e) Wie groß ist die Wahrscheinlichkeit, dass die Bewertung b_i des Semesters i einen bestimmten Wert j aufweist *und* diese Klausur in Semester i eine leichte ist?

Lösung: Die gesuchte Wahrscheinlichkeit besteht aus zwei Teilen, die mit einem „und“ verknüpft sind. Wir betrachten beide Teile einzeln und verknüpfen Sie unter Beachtung der Pfadregel mit einer

Multiplikation. Die Wahrscheinlichkeit, dass die Bewertung b_i den Wert j aufweist, ist $1/(n+2)$, da es $n+2$ verschiedene Bewertungen gibt.

Es gibt offenbar j Zahlen, die kleiner als j sind, da unsere Zählung bei 0 beginnt. Also bleiben für die Bewertung b_{i-1} noch j Möglichkeiten und für die Bewertung b_{i+1} noch $j-1$ Möglichkeiten. Es bleiben $(n-1)!$ Möglichkeiten der anderen Werte $b_0, \ldots, b_{i-2}$ und $b_{i+2}, \ldots, b_{n+1}$. Da die Bewertung b_i den festen Wert j hat, gibt es insgesamt noch $(n+1)!$ Möglichkeiten, um die restlichen Bewertungen zu verteilen.

Die gesuchte Wahrscheinlichkeit ist also:

$$P(b_i = j \wedge (b_i < b_{i-1} \wedge b_i < b_{i+1}) = \frac{1}{n+2} \cdot \frac{j \cdot (j-1) \cdot (n-1)!}{(n+1)!} = \frac{j(j-1)}{(n+2)(n+1)n}$$

(f) Wie groß ist die Wahrscheinlichkeit $P(K_i = 1)$ für ein gegebenes i?

Lösung: Die Bewertung b_i kann $n+2$ verschiedene Werte j annehmen und für jeden dieser Werte j kennen wir die Wahrscheinlichkeit, dass $b_i = j$ und die Klausur ist eine leichte Klausur. Gemäß der Baumregel müssen wir die Wahrscheinlichkeiten für alle möglichen Werten von j aufaddieren. Dies können wir dann vereinfachen:

$$\begin{aligned}
P(X_i = 1) &= \sum_{j=0}^{n+1} \frac{1}{n+2} \cdot \frac{j \cdot (j-1) \cdot (n-1)!}{(n+1)!} = \frac{(n-1)!}{(n+2)(n+1)!} \sum_{j=2}^{n+1} j \cdot (j-1) \\
&= \frac{1}{(n+2)(n+1)n} \sum_{j=1}^{n} (j+1)j = \frac{1}{(n+2)(n+1)n} \left(\sum_{j=1}^{n} j^2 + \sum_{j=1}^{n} j \right) \\
&= \frac{1}{(n+2)(n+1)n} \Big(\frac{n(n+1)(2n+1)}{6} + \frac{n(n+1)}{2} \Big) = \frac{2n+1}{6(n+2)} + \frac{1}{2(n+1)} \\
&= \frac{2n+1+3}{6(n+2)} = \frac{2(n+2)}{6(n+2)} = \frac{1}{3}
\end{aligned}$$

(g) Was ist die erwartete Anzahl von Klausuren in den Semestern 1 bis n, die als leicht empfunden werden?

Lösung: Nun können wir alle Ergebnisse zusammenfassen. Wir wissen, dass $E[K] = \sum_{i=1}^{n} E[K_i]$. Da wir $E[K_i]$ kennen, setzen wir das in die Formel ein und lösen auf:

$$E[K] = \sum_{i=1}^{n} E[K_i] = \sum_{i=1}^{n} \frac{1}{3} = \frac{n}{3}$$

Wir können erwarten, dass ein Drittel der Klausuren als leicht empfunden werden.

Aufgabe 7: Wahrscheinlichkeit und Erwartungswert

Auf dem Frühjahrs-Volksfest gibt es ein faires Glücksrad, welches mit den fünf Farben rot, grün, blau, gelb und glitzer angemalt ist. Der rote und der grüne Sektor nehmen jeweils 25 % der Fläche des Glücksrads ein, die anderen drei Farben nehmen jeweils ein Sechstel des Glücksrads ein.

(a) Hängt die Wahrscheinlichkeit, dass der Zeiger am Ende einer Drehung auf eine bestimmte Farbe zeigt, davon ab, ob die Farben zusammenhängen oder das Glücksrad in nach außen gehenden Streifen bemalt ist?

Lösung: Die Wahrscheinlichkeiten sind unabhängig davon, wie das Glücksrad bemalt ist, solange der insgesamte Flächenanteil einer Farbe dem insgesamten Anteil am Umfang des Glücksrads gleicht. Damit wird sichergestellt, dass keine Farbe zur Zierde im Inneren der Glücksrads angebracht ist und der Zeiger, der ja auf den Umfang zeigt, diese Farbe nicht erreichen kann.

(b) Sie bekommen fünf Freifahrten, falls Sie „glitzer“ gedreht haben (Hauptpreis), drei Freifahrten, wenn Sie „gelb“ gedreht haben. Zwei Freifahrten gibt es, wenn der Zeiger „blau“ zeigt und eine Freifahrt ist der Gewinn bei „grün“. Die Farbe „rot“ geht leer aus. Wie viele Freifahrten gibt der Standbesitzer im Durchschnitt pro Besucher aus, wenn jeder Besucher einmal drehen darf?

Lösung: Wir definieren zunächst die Zufallsvariable F, die dem Ergebnis ω eines Versuchs die Anzahl der Freifahrten zuweist:

$$F(\omega) = \begin{cases} 0 & \text{falls } \omega = \text{„rot“} \\ 1 & \text{falls } \omega = \text{„grün“} \\ 2 & \text{falls } \omega = \text{„blau“} \\ 3 & \text{falls } \omega = \text{„gelb“} \\ 5 & \text{falls } \omega = \text{„glitzer“} \end{cases}$$

Der Erwartungswert $E[F]$ ist dann die Summe aus den numerischen Werten multipliziert mit den Wahrscheinlichkeiten, dass diese Werte auftreten:

$$E(F) = 0,25 \cdot 0 + 0,25 \cdot 1 + 0,1\overline{6} \cdot 2 + 0,1\overline{6} \cdot 3 + 0,1\overline{6} \cdot 5 = 1,91\overline{6}$$

Pro Drehung werden im Schnitt $1,91\overline{6}$ Freifahrten verschenkt.

(c) Die letzten neun Drehungen hatten alle das Ergebnis „glitzer“. Wie hoch ist die Wahrscheinlichkeit, dass die nächste, zehnte Drehung, erneut das Ergebnis „glitzer“ hat?

Lösung: Alle Drehungen sind voneinander unabhängig, daher ändert sich die Wahrscheinlichkeit nie. Die Antwort ist daher $P(\text{„glitzer“}) = 0,1\overline{6}$.

(d) Dem Standbetreiber ist es wichtig, dass die Kunden in der Warteschlange sehen, dass der Hauptpreis gewinnbar ist. Es sieht jeweils der nächste Kunde der Warteschlange und der vorherige Kunde, der gerade sein Gewinn abholt, das Ergebnis der aktuellen Drehung. Wenn an einem Tag n Kunden das Glücksrad drehen, wie viele sind dann im Durchschnitt Zeuge eines Hauptgewinns eines anderen Kunden?

Lösung: Sei Z die Anzahl der Kunden, die einen Hauptgewinn sehen. Gesucht ist dann $E[Z]$. Wir lösen die Aufgabe mit der Indikatorzufallsvariable Z_i für $0 \leq i \leq n-1$:

$$Z_i = \begin{cases} 1 & \text{falls } w_{i-1} \text{ oder } w_{i+1} \text{ die Farbe „glitzer“ gezeigt hat} \\ 0 & \text{sonst} \end{cases}$$

Die Wahrscheinlichkeit, dass Z_i gleich 1 ist, müssen wir durch eine Fallunterscheidung berechnen, da der erste und letzte Kunde jeweils nur einmal die Möglichkeit hat, einen Hauptgewinn zu sehen:

$$P(Z_i = 1) = E[Z_i] = \begin{cases} 0,1\overline{6}^2 + 2(1 - 0,1\overline{6}) \cdot 0,1\overline{6} & \text{falls } 0 < i < n-1 \\ 0,1\overline{6} & \text{sonst} \end{cases}$$

Nun gilt

$$\begin{aligned} E[Z] = E[\sum_{i=0}^{n-1} Z_i] = \sum_{i=0}^{n-1} E[Z_i] &= 0,1\overline{6} + \sum_{i=1}^{n-2} 0,1\overline{6}(2-0,1\overline{6}) + 0,1\overline{6} \\ &= 0,1\overline{6} + (n-2)0,1\overline{6}(2-0,1\overline{6}) + 0,1\overline{6} \\ &= 0,\overline{3} + (n-2)0,1\overline{6}(2-0,1\overline{6}) \\ &= 0,\overline{3} + (n-2)\frac{184}{625} \\ &= 0,\overline{3} + \frac{184}{625}n - \frac{368}{625} \\ &= \frac{184}{625}n - \frac{479}{1875} = \frac{552}{1875}n \approx 0,2944n - 0,25 \end{aligned}$$

Das heißt, etwa ein Drittel der Kunden sind Zeugen eines fremden Hauptgewinns.

Aufgabe 8: Zufall und Zufallsvariablen mit Würfeln

Gegeben seien zwei 6-seitige faire Würfel.

(a) Geben Sie den Ereignisraum Ω und dessen Kardinalität an, wenn beide Würfel nacheinander einmal geworfen werden.

Lösung: Jeder Würfel zeigt eine Zahl zwischen 1 und 6 an. Somit ist $\Omega = \{(1,1),(1,2),\ldots,(6,5),(6,6)\} = \{1,2,3,4,5,6\}^2$ und dadurch ist die Kardinalität $|\Omega| = 6^2 = 36$.

(b) Mit welcher Wahrscheinlichkeit würfelt der erste Würfel eine bestimmte Augenzahl i und der zweite eine bestimmte Augenzahl j?

Lösung: Die Wahrscheinlichkeit, dass $(i,j) \in \Omega$ ist $\mathbf{Pr}[(i,j)] = 1/|\Omega| = 1/36$.

(c) Mit welcher Wahrscheinlichkeit haben beide Würfel die gleiche Augenzahl?

Lösung: Sei A das Ereignis, das beschreibt, dass beide Würfel die gleiche Augenzahlen haben. Dann gilt: $A = \{(1,1),(2,2),(3,3),(4,4),(5,5),(6,6)\}$ und somit ist $\mathbf{Pr}[(i,j) \in A] = |A|/|\Omega| = 6/36 = 1/6$.

(d) Was ist der Erwartungswert des Maximums der beiden Augenzahlen eines Wurfes mit diesen zwei Würfeln?

Lösung: Sei $X = \max\{i,j\} \in \{1,2,3,4,5,6\}$ die Zufallsvariable, die das Maximum eines Wurfes mit Augenzahlen i und j angibt. Durch Zählen erhalten wir die Wahrscheinlichkeiten $\mathbf{Pr}[X=1] = 1/36$, $\mathbf{Pr}[X=2] = 3/36$, $\mathbf{Pr}[X=3] = 5/36$, …, $\mathbf{Pr}[X=6] = 11/36$ und somit erhalten wir den Erwartungswert von X durch:

$$\mathbf{E}[X] = 1 \cdot \mathbf{Pr}[X=1] + 2 \cdot \mathbf{Pr}[X=2] + \cdots + 6 \cdot \mathbf{Pr}[X=6] \approx 4.5$$

Aufgabe 9: Bank-Schließfächer und Heaps

Bei einer Bank können Sie Schließfächer mieten. In jedes Schließfach passt genau ein Gegenstand. Der Service ist aber nicht kostenlos. Für das Einlagern des n. Gegenstands stellt die Bank $\log n$ Euro in Rechnung. Ebenfalls werden bei der Rückgabe des n. Gegenstands $\log n$ Euro fällig.

(a) Geben Sie die Gesamtkosten für das Ein- und Auslagern den n. Gegenstands in Θ-Notation an.

Lösung: Die Kosten für das Ein- und Auslagern des n. Gegenstands sind in $\Theta(\log n)$.

(b) Wie muss die Bank ihre Bezahlpolitik ändern, sodass das Abholen der Gegenstände kostenlos ist, die Bank aber trotzdem denselben Gewinn macht?

Lösung: Die Bank verlangt die Kosten für das Abholen des Gegenstands bereits bei der Einlagerung. Die Kosten für das Einlagern des n. Gegenstands sind dann $2 \log n$.

(c) Geben Sie die Kosten für das Ein- und Auslagern mit der neuen Bezahlpolitik in Θ-Notation an.

Lösung: Die Kosten für das Hinbringen bleiben in $\Theta(\log n)$, während die Kosten für das Abholen in $\Theta(1)$ liegen.

(d) Übertragen Sie Ihre Überlegungen aus den vorherigen Teilaufgaben auf einen MaxHeap. Zeigen Sie mit der Buchhaltermethode, dass die Insert-Methode amortisiert eine Laufzeit von $\mathcal{O}(\log n)$ und ExtractMax eine konstante Laufzeit hat.

Lösung: Wir zahlen für eine Einfügeoperation der i. Zahl $\hat{e}_i = 2 \log i$ und für das Löschen der i. Zahl $\hat{l}_i = 1$. Für das Einfügen von n Zahlen und das anschließende Extrahieren zahlen wir also $\hat{c} = \sum_{i=1}^{n} 2 \log i + \sum_{i=1}^{n} 1$. Vergleichen wir das mit den tatsächlichen Worst-Case-Kosten $c = \sum_{i=1}^{n} \log i + \sum_{i=1}^{n} \log i$, so erkennen wir, dass $\hat{c} > c$. Da $\hat{c} \in \mathcal{O}(n \log n)$ und auch die tatsächlichen Kosten $c \in \mathcal{O}(n \log n)$, schließen wir daraus, dass Insert eine amortisierte Laufzeit von $\mathcal{O}(\log n)$ hat und ExtractMax amortisiert in $\mathcal{O}(1)$ liegt.

(e) Lösen Sie die Aufgabe nun mit der Potentialmethode.

Lösung: Die Potentialmethode bei n Elementen im Heap definieren wir mit $\Phi(n) = \sum_{i=1}^{n} \log i$, also der Summe aller Höhen im Heap. Damit ist $\Phi(0) = 0$ und es gibt keinen Wert n, sodass $\Phi(k) < 0$. Nun berechnen wir die amortisierten Kosten der Operationen:
Insert: $\hat{c}_i = c_i + \Phi(n) - \Phi(n-1) = \log n + \sum_{i=1}^{n-1} \log i + \log n - \sum_{i=1}^{n-1} \log i = 2 \log n \in \mathcal{O}(\log n)$
ExtractMax: $\hat{c}_i = c_i + \Phi(n-1) - \Phi(n) = \log n + \sum_{i=1}^{n-1} \log i - (\sum_{i=1}^{n-1} \log i + \log n) = 0 \in \mathcal{O}(1)$

Aufgabe 10: Graham's Scan

Gegeben sei eine Menge $S \subseteq \mathbb{R}^2$ von Punkten auf der Ebene. Wir wollen ein Polygon finden, das alle Punkte von S enthält und nur Punkte aus S als Eckpunkte enthält. Abbildung 10 zeigt ein Beispiel für das gesuchte Polygon.

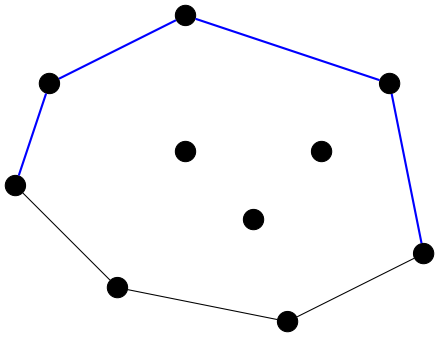


Abbildung 10: Beispiel für S und das dazugehörige Polygon in Aufgabe 10.

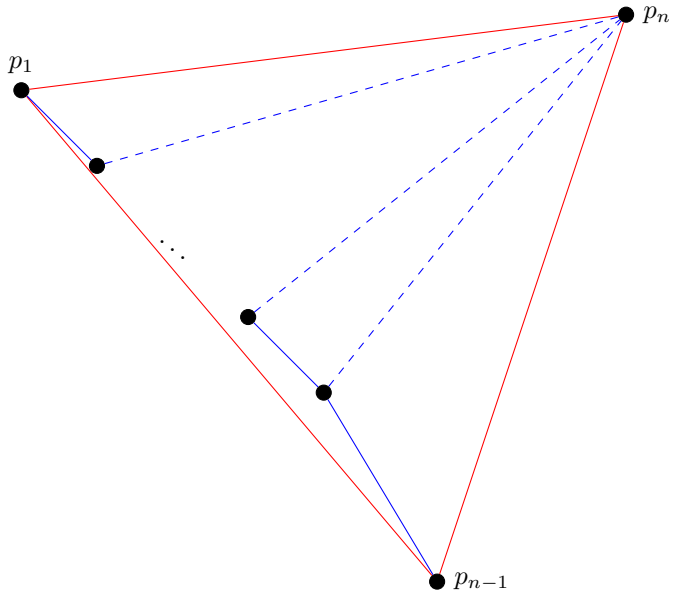
Algorithmus 6: Graham's Scan

```
1 Sortiere Punkte in S nach x-Koordinate,
    falls gefordert
2 L ← ⟨p_1, p_2⟩
3 for i = 3 to n do
4 |   L.Insert(p_i)
5 |   while |L| > 2 and die letzten 3 Punkte
  |     bilden einen links Knick do
6 |   |   entferne den vorletzten Punkt aus L
7 return L
```

Algorithmus 6 kann als Unterroutine benutzt werden, um ein solches Polygon zu finden. Dabei berechnet dieser Algorithmus die obere Hülle des Polygons. Das Beispiel in Abbildung 10 enthält die obere Hülle in blau eingezeichnet.

(a) Geben Sie eine worst-case Laufzeit für die Schleife in den Zeilen 5-6 an. Geben Sie weiterhin ein Beispiel an, für die dieser worst-case eintritt.

Lösung: Ein worst-case wäre folgendes Set S:



Hier sind $n-2$ Punkte in einer Geraden nach unten angereiht und ein weitere Punkt (p_{n-1}) ist unter beziehungweise über (p_n) dieser Geraden, sodass der Punkt p_1 ganz links auf der Geraden und die beiden Punkte über bzw. unter der Geraden das Polygon bilden. Der Algorithmus findet nun für die ersten $n-1$ Punkte keinen Linksknick, doch erst beim Einfügen des letzten Punkts (über der Gerade) entsteht ein Linksknick. Nun müssen wir die gesamte Liste L bis auf die Punkte p_1 und p_n entfernen. Folglich kann die Laufzeit der While-Schleife in den Zeilen 5-6 $\mathcal{O}(n)$ Zeit in Anspruch nehmen.

(b) Gehen Sie nun davon aus, dass S bereits nach x-Koordinate sortiert ist. Zeigen sie mittels amortisierter Analyse, dass die Laufzeit von Algorithmus 6 $\mathcal{O}(n)$ ist.

Lösung: Die Gesamtanzahl der Aufrufe des Schleifenkopfes hängt von der Länge der Elemente in L ab, da wir immer stoppen, wenn $|L| \leq 2$ ist, und wir bei einer Iteration genau ein Element löschen. Wir fügen einen Punkt p_i nur einmal in L ein und löschen diesen Punkt auch nur maximal einmal. Somit ist die Gesamtanzahl der Aufrufe des Schleifenkopfes $\mathcal{O}(n)$.

Aufgabe 11: Von Monte Carlo nach Las Vegas

(a) Geben Sie eine scharfe obere Schranke für die Fehlerwahrscheinlichkeit von Algorithmus 7 an.

Algorithmus 7: `FindLarge`(**int**[] A, **int** k)

input : Array A ist ein Feld ganzzahliger paarweise verschiedener Zahlen
output: Gibt ein Element zurück, dass mindestens so groß ist wie der Median

```
1 m = 0
2 for i = 1 to k do
3 |   randomly choose r ∈ {1, ..., n}
4 |   if A[r] > m then
5 |   |   m = A[r]
6 return m
```

Lösung: Wir geben ein falsches Ergebnis zurück, wenn in allen k Iterationen ein Element gezogen wird, das kleiner ist als der Median. Dies sind insgesamt $\lfloor n/2 \rfloor - 1$ Elemente. Somit haben wir eine Fehlerwahrscheinlichkeit von $\leq 1/2^k$.

(b) Geben Sie die Erfolgswahrscheinlichkeit mit jedem Schritt von Algorithmus 8 an.

Algorithmus 8: `FindRepeated`(**int**[] A)

input : Array A der Länge n mit ganzzahligen Zahlen, sodass $\lceil n/2 \rceil$ Zahlen identisch und $\lfloor n/2 \rfloor$ paarweise unterschiedlich sind.
output: Das identische Element.

```
1 while true do
2 |   randomly choose i ∈ {1, 2, ..., n}
3 |   randomly choose j ∈ {1, 2, ..., n} \ {i}
4 |   if A[i] = A[j] then
5 |   |   return A[i]
```

Lösung: Es gibt mindestens $n/2$ identische Elmente, somit ist die Wahrscheinlichkeit, dass $A[i]$ ein identisches Element ist mindestens $1/2$. Die Wahrscheinlichkeit, dass $A[j]$ ein identisches Element ist, ist dann mindestens $(n/2 - 1)/(n - 1)$. Somit ist die Erfolgswahrscheinlichkeit $\approx 1/4$.

(c) Was sind die Vor- und Nachteile beider Algorithmen?

Lösung: `FindLarge` könnte ein falsches Ergebnis zurückgeben, aber falls wir k sinnvoll wählen, z.B. $k = c \log_2 n$, dann ist die Laufzeit dieses Algorithmus schneller als jeder deterministische Algorithmus, der immer das korrekte Ergebnis liefert und hat eine sehr geringe Fehlerwahrscheinlichkeit. `FindRepeated` gibt auf jeden Fall immer das korrekte Ergebnis zurück aber könnte potentiell lange laufen. Allerdings ist die erwartete Anzahl an Iterationen 4 und somit erwartet konstant.

Aufgabe 12: Bernoullis Spielzeuge

Eine Supermarktkette verschenkt ein zufälliges Spielzeug, wenn Sie einen Einkauf in Höhe von über 10 Euro tätigen. Insgesamt gibt es n unterschiedliche Spielzeuge, die Sie gerne sammeln möchten. Mit wie vielen Einkäufen müssen Sie rechnen, bis Sie alle Spielzeuge gesammelt haben?

Lösung: Angenommen wir haben bereits $i - 1$ unterschiedliche Spielzeuge. Dann ist die Wahrscheinlichkeit ein neues Spielzeug zuerhalten $p_i = (n - i + 1)/n$. Sei X_i eine ZV, die angibt, wie häufig wir einkaufen gehen müssen, um das i-te Spielzeug zuerhalten. Wir kaufen solange ein, bis wir das i-te Spielzeug erhalten haben. Damit folgt das Experiment einer geometrischen Verteilungsfunktion und der

Erwartungswert ist $\mathbf{E}[X_i] = 1/p_i$. Somit ist die erwartete Anzahl an Einkäufen $\sum_{i=1}^{n} \mathbf{E}[X_i] = \sum_{i=1}^{n} = p_i = n \cdot (1/n + 1/(n-1) + \cdots + 1) \in \mathcal{O}(n \log n)$.