Karlsruher Institut für Technologie

Maximilian Katzmann
Hans-Peter Lehmann
Stefan Walzer

Übungsblatt 15 – Aktivsession

Randomisierte Algorithmik – Wintersemester 2023/2024

Folgende Aufgaben werden in der Aktiv-Session am 13.2.2024 gemeinsam bearbeitet.
Eine Abgabe ist nicht erforderlich.

Aufgabe 1 – AMQ aus Retrieval

Sei $k \in \mathbb{N}$ und S eine Menge der Größe $n = |S|$. Verwende eine Retrieval Datenstruktur der Vorlesung als Black-Box um eine statische Approximate-Membership-Query Datenstruktur für S mit einer falsch-positiv Wahrscheinlichkeit von $\varepsilon = 2^{-k}$ und einem Speicherbedarf von $1.23nk$ Bits zu konstruieren. (Beachte: Ein Bloomfilter würde $1.44nk$ Bits benötigen.)
Hinweis: Verwende eine Hashfunktion $f \sim \mathcal{U}([2^k]^D)$. Gemäß der Simple Uniform Hashing Assumption kannst du dir f merken ohne Speicher dafür aufzuwenden. Wir nennen $f(x)$ den *Fingerprint* von $x \in D$.

Lösung 1

Sei $f \sim \mathcal{U}([2^k]^D)$ wie vorgeschlagen und R eine Retrieval Datenstruktur für f_S. Sie hat wie gefordert Speicherbedarf $1.23nk$, wenn wir die Konstruktion der Vorlesung mit dem Schälalgorithmus nutzen.

Wir fassen R als eine Filterdatenstruktur auf, die bei Anfrage $x \in D$ genau dann YES zurückgibt, wenn $\text{eval}(R, x) = f(x)$ gilt. Nach Konstruktion gibt es keine falsch-negativen Antworten. Nehmen wir nun an, dass $x \in D \setminus S$ gilt. Der Fingerprint $f(x)$ ist uniform zufällig in $[2^k]$ verteilt und unabhängig von allem was bisher eine Rolle gespielt hat. Was auch immer $\text{eval}(R, x)$ ist, die Wahrscheinlichkeit, dass $f(x)$ damit übereinstimmt, und somit x ein falsch-positives Element ist, ist somit $2^{-k} = \varepsilon$ wie gewünscht.

Aufgabe 2 – Learned Data Structures

Sei S eine Menge von $n = |S|$ Namen mit eindeutig zuordenbaren Geschlecht $f : S \to \{\text{F}, \text{M}\}$. Eine findige Studentin bemerkt, dass die meisten $x \in S$ mit $f(x) = \text{F}$ auf einen Vokal enden und die meisten $x \in S$ mit $f(x) = \text{M}$ auf einen Konsonanten enden. Diese simple Regel funktioniert für alle außer δn der Namen, für ein kleines $\delta > 0$.

Konstruiere eine Datenstruktur mit erwartetem Speicherbedarf $O(\delta n \log(1/\delta))$, die für jedes $x \in S$ das korrekte Geschlecht $f(x)$ liefert.
Hinweis: Stöpsle dazu einen Bloom Filter und eine Retrieval Datenstruktur geschickt zusammen.
Bemerkung: Unter *Learned Data Structures* versteht man eine Kombination aus klassischen Datenstrukturen und Machine Learning Techniken. Wie in dieser Aufgabe angedeutet, geht

es darum, die Mustererkennungsfähigkeiten von Machine Learning Techniken mit den Verlässlichkeitsgarantien klassischer Datenstrukturen nutzbringend zu verheiraten.

Lösung 2

Sei $F \subseteq S$ die Menge der δn Namen, für die die Heuristik versagt. sei B ein Bloomfilter für F mit falsch-positiv Wahrscheinlichkeit δ. Sei nun S^+ die Menge derjenigen Namen aus S, für die der Bloomfilter eine positive Antwort gibt. Neben F enthält S^+ auch alle $x \in S$, die falsch-positive Elemente von B sind. Die erwartete Größe von S^+ beträgt höchstens $|F| + |S \setminus F| \cdot \delta = \delta n + (1-\delta)n \cdot \delta \leq 2\delta n$. Sei $f_{S^+} : S^+ \to \{\text{F}, \text{M}\}$ die Einschränkung von f auf S^+. Wir konstruieren eine Retrievaldatenstruktur R für f_{S^+}.

Die Intuition ist nun: Der Bloomfilter identifiziert die Namen, für die die Heuristik fehlschlägt, sowie ein paar falsch-positive. Die Retrievaldatenstruktur hat dann zur Aufgabe echt-positive von falsch-positiven Elementen zu trennen. Formal können wir für unsere Heuristik-gestützte Retrievaldatenstruktur R_H definieren:

Algorithm eval(R_H, x)**:**
  if query$(B, x) =$ NO **then**
    return Heuristik(x)
  else
    return eval(R, x)

Der Gesamtspeicherbedarf beträgt $1.44\delta n \log(1/\delta)$ Bits für B sowie erwartet höchstens $2\delta n$ Bits für R.

Aufgabe 3 – Retrieval mit Variabler Bitlänge

Gemäß Vorlesung können wir für jedes Universum D, jede Menge $S \subseteq D$ und jede Funktion $f : S \to \{0,1\}$ eine Retrieval-Datenstruktur für f mit Speicherbedarf $1.23|S|$ konstruieren. Dies soll hier als Black-Box verwendet werden.

Konstruiere eine Retrieval Datenstruktur für den Fall in dem der Wertebereich der Funktion f Bitstrings variabler Länge enthält.

Genauergesagt, sei $C \subseteq \{0,1\}^*$ ein präfixfreier Code, D' ein Universum, $T \subseteq D'$ und $g : T \to C$ eine Funktion. Konstruiere eine Datenstruktur R mit Speicherbedarf $1.23 \cdot \sum_{x \in T} |g(x)|$ und einen zugehörigen Algorithmus eval sodass für jedes $x \in T$ gilt eval$(R, x) = g(x)$.
Hinweis: Führe für jedes $x \in T$ soviele Schlüssel ein, wie die Länge $|g(x)|$ von $g(x)$ beträgt.

Lösung 3

Die Idee ist es, für jedes $x \in T$ mit $g(x) = (b_1, \ldots, b_k)$ die Schlüssel $\{(x,1), (x,2), \ldots, (x,k)\} \subseteq D' \times \mathbb{N}$ einzuführen, wobei (x,i) der Wert b_i zugeordnet ist. Es ergibt sich somit eine Funktion $f : S \to \{0,1\}$ wobei $S \subseteq D' \times \mathbb{N}$ eine Menge von Paaren ist mit $|S| = \sum_{x \in T} |g(x)|$.

Wir können also gemäß Vorlesung für das Universum $D = D' \times \mathbb{N}$ sowie S und f wie beschrieben eine Retrieval Datenstruktur R mit Speicherbedarf $1.23|S|$ konstruieren. Um dieser Datenstruktur dann für $x \in T$ den Wert $g(x)$ zu entlocken, betrachten wir der Reihe nach

$\text{eval}(R, (x, 1)), \text{eval}(R, (x, 2)), \text{eval}(R, (x, 3)), \ldots$ bis die gesehene Folge einen Code aus C ergibt. Weil C präfixfrei ist, wissen wir, dass keine Fortsetzung der Folge in C liegt, also haben wir $g(x)$ vollständig bestimmt.

Bei einer Anfrage für $x \in D' \setminus T$ ist egal, welches Element von C zurückgegeben wird, wir müssen lediglich sicherstellen, dass eval nicht abstürzt oder in eine Endlosschleife gerät. Das ist leicht (und im Grunde fast automatisch der Fall).

Aufgabe 4 – Minimal-Perfekte Hash Funktion mit Brute Force

Betrachte folgendes Algorithmenpaar zur Konstruktion und Auswertung von Minimal-Perfekten Hashfunktionen[1]. Dabei sei $S \subseteq D$ und $n = |S|$. Wir gehen gemäß der SUHA davon aus, dass $h_1, h_2, h_3, \ldots \sim \mathcal{U}([n]^D)$ unabhängige, voll zufällige Hashfunktionen sind, die selbst keinen Speicherplatz benötigen.[2]

Algorithm construct(S)**:**
  for seed = 1 **to** ∞ **do**
    if $|\{h_{\text{seed}}(x) \mid x \in S\}| = n$ **then**
      return seed

Algorithm eval(seed, x)**:**
  return $h_{\text{seed}}(x)$

(a) Argumentiere: Jeder Schleifendurchlauf in construct führt mit Wahrscheinlichkeit $\frac{n!}{n^n}$ zum Erfolg.

(b) Argumentiere: Der Rückgabewert seed von construct erfüllt $\mathbb{E}[\text{seed}] = \frac{n^n}{n!}$.

(c) Der Speicherbedarf beträgt space $= \lceil \log_2(\text{seed}) \rceil$ Bits. Zeige: $\mathbb{E}[\text{space}] \leq n \log_2(e) + 1$. **Hinweis:** Verwende die Jensen-Ungleichung (Aufgabenblatt 10) sowie die Stirling Approximation der Fakultätsfunktion. (siehe en.wikipedia.org/wiki/Stirling's_approximation).

(d) Bewerte den erwarteten Platzbedarf und die erwartete Konstruktionszeit des vorgestellten Verfahrens auf einer Skala von großartig bis grottenschlecht.

Lösung 4

(a) Die Bedingung $|\{h_{\text{seed}}(x) \mid x \in S\}| = n$ ist äquivalent dazu, dass die n Hashwerte der n Schlüssel in S unter h_{seed} paarweise verschieden sind. Abstrakt gesprochen müssen wir beim n-maligen ziehen aus einer Urne mit n Bällen mit Zurücklegen jedes mal einen Ball ziehen, den wir zuvor noch nicht gezogen haben. Die Wahrscheinlichkeit dafür ist

$$\frac{n}{n} \cdot \frac{n-1}{n} \cdot \frac{n-2}{n} \cdot \ldots \cdot \frac{2}{n} \cdot \frac{1}{n} = \frac{n!}{n^n}.$$

[1] Beachte: Die „Datenstruktur“ besteht hier lediglich aus der Zahl seed.

[2] Alternativ kann man sich vorstellen, dass eine einzige Hashfunktion $h : \mathbb{N} \times D \to [n]$ voll zufällig ist, die neben dem eigentlichen Schlüssel eine natürliche Zahl als Seed akzeptiert.

(b) Wie oft musst du im Mittel mit einem sechsseitigen Würfel würfeln, bevor du das erste mal eine Sechs würfelst? Naja, sechs mal. Dieselbe Überlegung liegt hier zugrunde:

Die Zufallsvariable seed gibt an, wie oft wir ein Zufallsexperiment mit Erfolgswahrscheinlichkeit $p = n!/n^n$ durchgeführt haben bis der erste Erfolg eingetreten ist. Damit gilt seed $\sim \text{Geo}(p)$. Wir können einfach Nachrechnen oder auf Wikipedia nachlesen, dass damit $\mathbb{E}[\text{seed}] = 1/p = \frac{n^n}{n!}$ gilt.

(c) Aus der Stirling Approximation ziehen wir $n! \geq \left(\frac{n}{e}\right)^n$. Damit gilt:

$$\begin{aligned}\mathbb{E}[\text{space}] = \mathbb{E}[\lceil \log_2(\text{seed}) \rceil] \leq \mathbb{E}[\log_2(\text{seed})] + 1 &\overset{\text{Jensen}}{\leq} \log_2(\mathbb{E}[\text{seed}]) + 1 \\ = \log_2\left(\frac{n^n}{n!}\right) + 1 \leq \log_2\left(\frac{n^n}{(n/e)^n}\right) + 1 &= \log_2(e^n) + 1 = n \cdot \log_2(e^n) + 1.\end{aligned}$$

(d) Der erwartete Platzbedarf ist angesichts der unteren Schranke, die wir in der nächsten Aufgabe kennenlernen (nahezu) optimal. Die erwartete Konstruktionszeit ist mindestens $n^n/n! \approx e^n$ und somit grottenschlecht.

Aufgabe 5 – Untere Platzschranken für MPHF

Wir wollen zeigen, dass eine minimal-perfekte Hashfunktion im Allgemeinen nicht mit weniger als $\log_2(e) \approx 1.44$ Bits pro Element auskommt.

Betrachte Vorlesungsfolie 9. Sei $\varepsilon = 0$, $n = m \in \mathbb{N}$, $d = |D|$. Sei $\mathcal{I} := \{S \subseteq D \mid |S| = n\}$ die Menge aller möglicher Eingaben der Größe n. Wir betrachten die Anzahl der Eingaben, für die eine gegebene Datenstruktur P gleichzeitig als perfekte Hashfunktion dienen kann. Sei hierzu:

$$\text{cov}(P) := \left\{ S \in \mathcal{I} \;\middle|\; \{\text{eval}(P, x) \mid x \in S\} = [n] \right\}.$$

(a) Zum Warmwerden: Angenommen $n = 3$, $D = \{\text{a}, \text{b}, \text{c}, \text{d}, \text{e}, \text{f}, \text{g}, \text{h}\}$ und P ist eine Datenstruktur auf der sich eval wie in folgender Tabelle dargestellt verhält. Begründe: P kann für $|\text{cov}(P)| = 12$ verschiedene $S \in \mathcal{I}$ als MPHF dienen.

$x \in D$	a	b	c	d	e	f	g	h
$\text{eval}(P, x)$	3	3	2	1	1	3	1	3

(b) Seien nun n und d sowie P beliebig. Zeige $\text{cov}(P) \leq (\frac{d}{n})^n$.
Hinweis: Verwende ohne Begründung, dass unter allen Quadern mit vorgebener Summe von Kantenlängen der Würfel das maximale Volumen besitzt. Im n-dimensionalen bedeutet das $\max_{\substack{0 \leq c_1, \ldots, c_n \leq d \\ c_1 + \cdots + c_n = d}} c_1 \cdot \ldots \cdot c_n = (d/n)^n$.

Sei nun $\mathcal{P} = \{\text{construct}(S) \mid S \in \mathcal{I}\}$ die Menge aller verschiedener Datenstrukturen, die construct produziert.

(c) Begründe $\bigcup_{P \in \mathcal{P}} \text{cov}(P) = \mathcal{I}$ und folgere $|\mathcal{P}| \geq \binom{d}{n} / (\frac{d}{n})^n$.

(d) Zeige $\binom{d}{n}/(\frac{d}{n})^n \geq \frac{n^n}{n!} \cdot (1 - o(1))$ falls $d = \omega(n^2)$.
Hinweis: Überlege dir in einem Zwischenschritt, dass $(1 - \frac{n}{d})^n \geq 1 - \frac{n^2}{d} = 1 - o(1)$ gilt.

(e) Begründe: Werden die Datenstrukturen, die construct liefert, jeweils als Bitstring einer Länge ℓ kodiert, dann gilt $\ell \geq \lceil \log_2(|\mathcal{P}|) \rceil$.

(f) Zeige dass $\ell \geq \log_2(e) \cdot n - o(n)$ gilt falls $d = \omega(n^2)$.
Hinweis: Setze die vorherigen Teilaufgaben zusammen und verwende Stirlings Approximation der Fakultätsfunktion.

Lösung 5

(a) Damit $S \in \text{cov}(P)$ gilt, muss S genau ein Element x_1 enthalten, für das $\text{eval}(P, x) = 1$ gilt, genau ein Element x_2 enthalten, für das $\text{eval}(P, x) = 2$ gilt und genau ein Element x_3 enthalten, für das $\text{eval}(P, x) = 3$ gilt. An der Tabelle sieht man, dass es 3 mögliche Wahlen für x_1, nur eine mögliche Wahl für x_2 und 4 mögliche Wahlen für x_3 gibt. Das Multipliziert sich zu $3 \cdot 1 \cdot 4 = 12$ Möglichkeiten.

(b) Im Geiste der Warmwerd-Aufgabe sei $S_i = \{x \in D \mid \text{eval}(P, x) = i\}$ für $i \in [n]$. Damit P minimal-perfekt auf S ist, muss S aus jedem der Mengen $S_1, \ldots, S_n$ genau ein Element enthalten. Definieren wir $c_i := |S_i|$ so gilt $\text{cov}(P) = c_1 \cdot \ldots \cdot c_n$. Weil die Mengen $(S_i)_{i \in [n]}$ eine Partition von S bilden gilt $c_1 + \cdots + c_n = d$. Nach dem Hinweis folgt somit $\text{cov}(P) \leq (d/n)^n$ wie behauptet.

(c) Die Korrektheit der Datenstruktur garantiert für jedes $S \in \mathcal{I}$, dass $S \in \text{cov}(\text{construct}(S))$. Also wird jedes $S \in \mathcal{I}$ von der Vereinigung abgedeckt. Mithilfe einer (nicht-probabilistischen) Vereinigungsschranke sowie dem vorherigen Aufgabenteil folgt:

$$\binom{d}{n} = |\mathcal{I}| = \left| \bigcup_{P \in \mathcal{P}} \text{cov}(P) \right| \leq \sum_{P \in \mathcal{P}} |\text{cov}(P)| \leq \sum_{P \in \mathcal{P}} \left(\frac{d}{n}\right)^n = |\mathcal{P}| \cdot \left(\frac{d}{n}\right)^n.$$

(d) Zunächst zeigen wir per Induktion, dass für jedes $\varepsilon > 0$ und jedes $i \in \mathbb{N}_0$ gilt, dass $(1 - \varepsilon)^i \geq 1 - \varepsilon i$. Der Basisfall $i = 0$ ist klar. Der Induktionsschritt geht so:

$$(1 - \varepsilon)^{i+1} = (1 - \varepsilon)^i \cdot (1 - \varepsilon) \overset{\text{Ind}}{\geq} (1 - \varepsilon i) \cdot (1 - \varepsilon) = 1 - \varepsilon i - \varepsilon + \varepsilon^2 i \geq 1 - \varepsilon(i + 1).$$

Angewendet für $\varepsilon = \frac{n}{d}$ und $i = n$ ergibt sich der Hinweis. Wir rechnen nun:

$$\frac{\binom{d}{n}}{\left(\frac{d}{n}\right)^n} = \frac{\frac{d \cdot (d-1) \cdot \ldots \cdot (d-n+1)}{n!}}{\left(\frac{d}{n}\right)^n} \geq \frac{\frac{(d-n)^n}{n!}}{\left(\frac{d}{n}\right)^n} = \frac{n^n}{n!} \cdot \left(\frac{d-n}{d}\right)^n = \frac{n^n}{n!} \cdot \left(1 - \frac{n}{d}\right)^n$$
$$\overset{\text{Hinweis}}{\geq} \quad \frac{n^n}{n!} \cdot \left(1 - \frac{n^2}{d}\right) = \frac{n^n}{n!} \cdot (1 - o(1)).$$

(e) Für jede Möglichkeit in $\mathcal{P}$ muss ein eigener Bitstring zur Verfügung stehen. Da es nur 2^ℓ Bitstrings der Länge ℓ gibt, muss also $|\mathcal{P}| \leq 2^\ell$ gelten. Umstellen und Ganzzahligkeit von ℓ ergibt die gewünschte Formel.

(f) Stirlings Approximation liefert $n! = \Theta((n/e)^n \cdot \sqrt{n})$. Somit gilt:

$$\begin{aligned}
\ell &\geq \log_2(|\mathcal{P}|) \geq \log_2\left(\binom{d}{n} \Big/ \left(\frac{d}{n}\right)^n\right) \geq \log_2\left(\frac{n^n}{n!} \cdot (1 - o(1))\right) \\
&= \log_2\left(\frac{n^n}{(n/e)^n \cdot \Theta(\sqrt{n})}\right) = \log_2(e^n) - \log_2(\Theta(\sqrt{n})) \\
&= n \cdot \log_2(e) - \tfrac{1}{2}\log_2(n) \pm \Theta(1) = n \cdot \log_2(e) - O(\log n).
\end{aligned}$$