ETH Zürich
Institute of Theoretical Computer Science
Prof. Angelika Steger, Prof. Emo Welzl
Dr. Anders Martinsson, Charlotte Knierim

FS 2022

Algorithmen und Wahrscheinlichkeit
Theorie-Aufgaben 8

Abgabe in Moodle (https://moodle-app2.let.ethz.ch/mod/assign/view.php?id=743710) bis zum 28.04.2022 um 16:15 Uhr.

Aufgabe 1 – *Bunte Kreise*

In der Vorlesung haben Sie einen Algorithmus kennengelernt, welcher entscheidet ob ein Graph $G = (V, E)$ einen "langen Pfad" besitzt. Eine Variante dieses Algorithmus' kann auch zur Berechnung langer Kreise verwendet werden. Betrachten Sie dazu einen mit der Menge $[k]$ gefärbten Graphen $G = (V, E)$ und definieren Sie für $u, v \in V$, $u \neq v$ und $i \in \{1, 2, \ldots, k-1\}$ die Menge

$$P_i(u,v) := \left\{ S \in \binom{[k]}{i+1} \;\middle|\; \exists \text{ einen bunten } u\text{-}v\text{-Pfad, der genau mit den Farben in } S \text{ gefärbt ist} \right\}$$

(a) Beschreiben Sie die Mengen $P_1(u, v)$, $u, v \in V$, $u \neq v$.

(b) Beschreiben Sie wie für $i \geq 2$ die Mengen $P_{i-1}(u, v)$, $u, v \in V$, $u \neq v$, die Mengen $P_i(u, v)$, $u, v \in V$, $u \neq v$, bestimmen und zwar als Mengendefinition (Formel) und als Algorithmus.

(c) Analysieren Sie den Zeitbedarf des Schritts (b) um $P_i(u, v)$ für alle $u, v \in V$, $u \neq v$ zu berechnen.

(d) Zeigen Sie, wie Sie mit Hilfe dieser Mengen entscheiden können, ob ein mit $[k]$ gefärbter Graph G einen bunten Kreis der Länge k hat.

(e) Analysieren Sie den Zeitbedarf des Verfahrens in (d).

ETH Zürich
Institute of Theoretical Computer Science
Dr. Johannes Lengler, Prof. Emo Welzl
Dr. Anders Martinsson, Charlotte Knierim

FS 2021

Algorithmen und Wahrscheinlichkeit
Extra-Aufgaben Woche 9

Die folgenden Aufgaben dienen der Selbstüberprüfung. Sie werden nicht korrigiert und geben keine Bonuspunkte.

Old Exercise, not relevant

Aufgabe 2 – *Primzahlgenerator*

In der Vorlesung haben Sie gesehen, wie ein Monte-Carlo-Algorithmus benutzen werden kann um zu testen ob eine Zahl eine Primzahl ist. In dieser Aufgabe zeigen wir, wie man diese Idee zu einem Algorithmus der Primzahlen generiert erweitern kann.

Nehmen Sie an wir haben Zugang zu den folgenden zwei Black-Box Algorithmen[1].

- `getMaybePrime()` gibt eine zufällige natürliche Zahl N aus, für die gilt, dass N eine Primzahl ist mit einer (kleinen aber positiven) Wahrscheinlichkeit $\varepsilon > 0$.
- `testPrime(`N`)` nimmt als Eingabe eine natürliche Zahl N. Falls N eine Primzahl ist, so gibt die Funktion immer "Primzahl" aus. Falls N keine Primzahl ist, gibt die Funktion mit Wahrscheinlichkeit $\frac{1}{2}$ "Primzahl" aus und mit Wahrscheinlichkeit $\frac{1}{2}$ "keine Primzahl".

Für eine natürliche Zahl $k \geq 1$ betrachten Sie den folgenden Algorithmus.

1. Sei N die Ausgabe von `getMaybePrime()`,
2. rufe `testPrime(`N`)` genau k mal auf,
3. falls `testPrime(`N`)` genau k mal "Primzahl" zurück gibt, gib N aus,

[1] Diese Funktionen könnten unter anderem durch einen Zufallszahlengenerator und den Miller-Rabin Test realisiert werden, aber für diese Aufgabe ist die Implementation der Funktionen nicht wichtig.

4. sonst, gehe zu Schritt 1.

In den folgenden Teilaufgaben wollen wir diesen Algorithmus analysieren.

(a) Betrachten Sie einen Durchlauf der Schleife (also einen Durchlauf der Schritte 1–3) des oberen Algorithmus.

(i) Was ist die Wahrscheinlichkeit, dass eine Primzahl ausgegeben wird in Schritt 3?

(ii) Was ist die Wahrscheinlichkeit, dass eine Zahl, die keine Primzahl ist, ausgegeben wird in Schritt 3?

(iii) Was ist die Wahrscheinlichkeit, dass nichts ausgegeben wird in Schritt 3? Also die Wahrscheinlichkeit, dass die Schleife erneut durchlaufen wird?

(b) Sei P_{fail} die Wahrscheinlichkeit, dass der Algorithmus eine Zahl ausgibt, die keine Primzahl ist. Benutzen Sie ihre Ergebnisse aus (a) um eine Rekursionsformel für P_{fail} aufzustellen, lösen Sie diese und geben Sie die Wahrscheinlichkeit in geschlossener Form an.

Hinweis: Es kann hilfreich sein zu beachten, dass der Algorithmus in jeder Iteration neu startet und daher gilt, dass $\Pr[\text{Fail}] = \Pr[\text{Fail} \mid \text{Nothing returned first iteration}]$.

(c) Sei $\delta > 0$. Zeigen Sie, dass unser Algorithmus für $k \geq \log_2(\varepsilon^{-1} - 1) + \log_2(\delta^{-1} - 1)$ mit Wahrscheinlichkeit mindestens $1 - \delta$ eine Primzahl ausgibt.

(d) Nehmen Sie an, wir wollen mit unserem Algorithmus B-bit Primzahlen generieren. In diesem Fall laufen sinnvolle Implementationen von `getMaybePrime()` und `testPrim(`N`)` in $O(B)$ beziehungsweise $O(B^3)$ Zeit, mit $\varepsilon = \Theta(1/B)$.

Zeigen Sie, dass unser Algorithmus eine Primzahl mit Erfolgswahrscheinlichkeit $1 - \delta$ für jedes $\delta \in (0, 1)$ in einer erwarteten Laufzeit von $O\left(B^4 \left(\log B + \log \delta^{-1}\right)\right)$ generieren kann.