Karlsruher Institut für Technologie — Prof. Dr. Jörn Müller-Quade
Institut für Theoretische Informatik — Björn Kaidel, Sebastian Schlag, Sascha Witt

5. Tutorenblatt zu Algorithmen I im SoSe 2017

http://crypto.iti.kit.edu/index.php?id=799
{bjoern.kaidel,sascha.witt}@kit.edu

Im Folgenden findet ihr einige unverbindliche Vorschläge zur Gestaltung des fünften Tutoriums.
In den Vorlesung wird/wurde am mit 22.05 und 24.05 das Kapitel Sortieren *angefangen*. Bisher wurden voraussichtlich `Mergesort`, `Insertionsort`, `Quicksort` besprochen, die $\Omega(n \log n)$-Schranke für vergleichsbasiertes Sortieren bewiesen und über randomisierten Quicksort gesprochen.
Ihr könnt Euch daher diese Woche auf Sortieren konzentrieren und z.B. weitere Sortieralgorithmen vorstellen und analysieren.
Eine Möglichkeit wäre beispielsweise `Binary Insertion Sort`:
`https://github.com/bingmann/sound-of-sorting/commit/0836dbfc19f1acf1c3841eebf25b31d37be397fb`.
„Sound of Sorting“ wurde von einem ehemaligen Übungsleiter geschrieben. Das compilierte Programm sowie die Erklärung, was die verschiedenen Töne bedeuten, findet ihr unter `http://panthema.net/2013/sound-of-sorting/` - oder eben die Sourcen auf Github. Das kommt bei den Studenten immer sehr gut an, spielt damit ruhig im Tutorium mal rum - in der Übung werden wir das vermutlich nicht oder nur am Rande verwenden.

Aufgabe 1

Rechnet ein paar Sortieralgorithmen vor, beschreibt wir sie gehen und macht Beispiele. Nehmt irgendeine beliebige Liste und lasst sie mit Insertion-, Merge- und Quicksort sortieren. Haltet euch in der Darstellung der Beispiele an die Vorlesung (siehe Vorlesungsfolien). Insbesondere sollte Mergesort gut verstanden werden. Quicksort kommt erst auf dem nächsten Übungsblatt dran, kann aber trotzdem schonmal gezeigt werden. Nehmt bitte nicht die Matrikelnummer von einem Studenten, da die ja in einer Übungsaufgabe von Blatt 5 sortiert werden soll.

Aufgabe 2

Sie sind der Manager eines Autoverleihs. Ihre Firma besitzt $k \in \mathbb{N}$ unterschiedliche Fahrzeugtypen, die alle verliehen werden können. Von jedem Fahrzeugtyp $t \in \{1 \dots k\}$ besitzt Ihre Firma $c_t \in \mathbb{N}$ Fahrzeuge. Ihnen liegen nun die n nächsten Buchungen vor, wobei jede Buchung aus einem Abholzeitpunkt, einem Rückgabezeitpunkt und einem gewünschten Fahrzeugtypen besteht. Ihre Aufgabe ist es, zu überprüfen, ob mit den vorhandenen Fahrzeugen alle Buchungen erfüllt werden können. Zum aktuellen Zeitpunkt sind keine Fahrzeuge verliehen. Gehen Sie davon aus, dass ein Fahrzeug, ab dem Moment, in dem es zurückgegeben wird, sofort wieder verliehen werden kann.

a) Entwerfen Sie einen Algorithmus, der dieses Problem in höchstens $\mathcal{O}(n \log n + k)$ Zeit löst.

b) Begründen Sie kurz, warum Ihr Algorithmus das gewünschte Laufzeitverhalten aufweist.

Musterlösung

a) Der Algorithmus geht wie folgt vor: Zunächst wird aus der Liste der Buchungen eine Liste von Ereignissen erzeugt. Ein Ereignis besteht aus einem Zeitpunkt, einem Fahrzeugtypen und einem Vorgang, der entweder *Ausleihe* oder *Rückgabe* sein kann. Somit werden für jede Buchung genau zwei Ereignisse angelegt. Die Liste der Ereignisse wird dann aufsteigend nach Zeitpunkt sortiert, wobei bei Ereignissen mit gleichen Zeitpunkten Rückgaben vor Ausleihen sortiert werden müssen.

   Nun legt der Algorithmus ein Array von A von k Zahlen an, eins pro Fahrzeugtyp. Das Array wird mit der Anzahl verfügbarer Fahrzeuge initialisiert: $A[t] = c_t$ für alle $t \in \{1 \dots k\}$. Dann

iteriert der Algorithmus über die sortierten Ereignisse. Bei einem Ereignis vom Typ *Rückgabe* und Fahrzeugtyp t wird $A[t]$ um eins inkrementiert, bei einem Ereignis vom Typ *Ausleihe* wird $A[t]$ um eins dekrementiert. Unterschreitet $A[t]$ dabei zu einem beliebigen Zeitpunkt den Wert 0, so können die vorliegenden Buchungen mit den vorhandenen Fahrzeugen nicht erfüllt werden.

b) Initial müssen genau $2n$ Ereignisse erstellt werden. Da dies im Voraus bekannt ist, kann ein Array mit $2n$ Einträgen für die Ereignisse angelegt und gefüllt werden. Dies läuft in $\mathcal{O}(n)$. Das Sortieren dieses Arrays läuft sodann z.B. durch Verwendung von Mergesort in $\mathcal{O}(n \log n)$. Das Anlegen und Initialisieren von A mit k Einträgen funktioniert klar in $\mathcal{O}(k)$. Abschließend muss über $2n$ Ereignisse iteriert werden, wobei die Abarbeitung jedes einzelnen Ereignisses in konstanter Zeit erfolgt. Insgesamt ergibt sich für diesen Schritt erneut Zeitaufwand in $\mathcal{O}(n)$.
In der Summe ergibt sich ein Zeitaufwand in $\mathcal{O}(n \log n + k)$.

Kreativaufgabe

Gegeben sei ein Array mit n verschiedenen Elementen (unsortiert, aber mit Ordnung) und eine Medianfunktion, die für ein (Teil-)Array mit m Elementen den Median deterministisch in $O(m)$ berechnet.

1. Schwierigkeitsstufe:
Finde einen Algorithmus, der das $\frac{1}{3}$-Perzentil deterministisch in $O(n)$ berechnet.
2. Schwierigkeitsstufe:
Finde einen Algorithmus, der die $\frac{1}{3^{k-1}}, \frac{1}{3^{k-2}}, \ldots, \frac{1}{3}$-Perzentile deterministisch in $O(n)$ berechnet. (Nicht in $O(nk)$!)

Zusatzschwierigkeit: Der Algorithmmus funktioniert in-place.

Lösung:
1. Schwierigkeitsstufe:
Einengen des $\frac{1}{3}$-Perzentils mit „Intervallhalbierung“ und Master-Theorem.

2. Schwierigkeitsstufe:
Halte in jedem Schritt i das Intervall der $\frac{n}{2^i}$ kleinsten Elemente. In jedem Schritt (ab dem 2.) kommt maximal ein zusätzliches Perzentil hinzu, dass nichtmehr im Intervall der kleinsten Elemente liegt. Alle Perzentile, die nicht mehr im Intervall der kleinsten Elemente liegen, liegen im i-ten Schritt in einem Intervall der Größe $\frac{n}{2^i}$. Die Arbeit in jedem Schritt ist proportional zu der Anzahl der Elemente in allen solchen Intervallen. Damit ist die Arbeit im Schritt i gleich $\frac{i \cdot n}{2^i}$. Und die Gesamtarbeit ist somit kleiner gleich $c \cdot \sum_{i=1}^{\infty} \frac{i \cdot n}{2^i} = cn \cdot \sum_{i=1}^{\infty} \sum_{j=i}^{\infty} \frac{1}{2^j} = cn \cdot \sum_{i=1}^{\infty} \frac{1}{2^{i-1}} = 2cn$.

Zusatzschwierigkeit: Wie bei Quicksort.