Algorithmik

Funke/Bahrdt/Krumpe/Mendel/Seybold SS 2015

http://www.fmi.informatik.uni-stuttgart.de/alg

Institut für Formale
Methoden der Informatik
Universität Stuttgart

Übungsblatt 4

Punkte: **60**

Problem 1 - Happy Hour (10 P.)

Sie und Ihre $(n-1)$ Freunde wollen sich zur einstündigen Happy Hour in einer (Milch-)Bar treffen. Sie können sich allerdings nicht auf eine Uhrzeit festlegen und einigen sich daher darauf, dass jeder zu einer zufälligen Uhrzeit (innerhalb dieser Stunde) kommt. Sie haben dabei alle die gleiche Strategie: Wenn Sie die Bar betreten, warten Sie maximal 20 Minuten darauf, dass alle Ihre $(n-1)$ anderen Freunde auch eingetroffen sind. Ist dies nicht der Fall, so bestellen Sie einfach beleidigt selber etwas.

Wie hoch ist die Wahrscheinlichkeit im Fall $n = 2$ dafür, dass Sie sich beide rechtzeitig treffen?
Wie hoch ist die Wahrscheinlichkeit in Abhängigkeit von n wenn gilt $n > 2$?

Problem 2 - Universelles Hashing (8 P.)

Sei $\mathcal{U}$ mit $|\mathcal{U}| = u$ eine Menge zu hashender Elemente und $p \geq u$ eine Primzahl. Die Funktionenklasse $h_{a,b} : \mathcal{U} \to \mathbb{Z}_m$ hasht diese Elemente nach $\{0, \ldots, m-1\}$. Beweisen Sie, dass die Klasse von Hashfunktionen $h_{a,b}(x) = (((ax + b) \mod p) \mod m)$ unter der Vorraussetzung, dass $a \in \{1, \ldots, p-1\}$ zufällig gleichverteilt gewählt wird, *universell* ist.

Problem 3 - Vertex Cover (7 P.)

Zeigen Sie mittels Polynomzeitreduktion, dass die folgenden drei Vertex Cover Varianten gleich schwer sind (NP hart). Gegeben sei ein Graph $G = (V, E)$:

Entscheidungsproblem:	Gibt es ein VC der Größe k?
Optimierungsproblem:	Berechne die Größe k^* eines optimalen VC.
Berechnungsproblem:	Gebe ein optimales VC: $C \subseteq V$ an.

Problem 4 - Vertex Cover Greedy (15 P.)

Der folgende Greedy Algorithmus berechnet für einen Graphen $G = (V, E)$ ein Vertex Cover C:

function GreedyVC($G_0 = (V_0, E_0)$)
  $C \leftarrow \emptyset$
  $i \leftarrow 0$
  while $|E_i| > 0$ **do**
    $v_i \leftarrow$ Knoten in G_i mit maximalem Grad
    $C \leftarrow C \cup v_i$
    $G_{i+1} \leftarrow G_i \setminus v_i$
    $i \leftarrow i + 1$
  end while
  return C
end function

Beweisen Sie, dass `GreedyVC(G)` nicht besser als $\Omega(\log(n))$ approximiert.
Beweisen Sie, dass `GreedyVC(G)` nicht schlechter als $\mathcal{O}(\log(n))$ approximiert und damit eine $\Theta(\log(n))$ APX ist.

Problem 5 - Erwartete Turmhöhe in Skiplisten (5 P.)
Wir betrachten den diskreten Wahrscheinlichkeitsraum $(\mathbb{N}_0, \Pr)$ mit $\Pr : \mathbb{N}_0 \to [0,1]$ wobei $i \mapsto 1/2^{i+1}$. Sei $H : \mathbb{N}_0 \to \mathbb{R}$ die Zufallsvariable der Elementarereignisse ($H : i \mapsto i$). Zeigen Sie, dass

- Die Abbildung Pr macht $(\mathbb{N}_0, \Pr)$ tatsächlich zu einem Wahrscheinlichkeitsraum ($1 = \sum_{x \in \mathbb{N}_0} \Pr(x)$).
- Der Erwartungswert der Zufallsvariable H ist 1.

Problem 6 - Skiplisten (5+10 P.)
Beschreiben sie in Pseudocode wie man die Zeiger in einer Skipliste bei den Operationen `insert(key)` und `erase(key)` modifiziert, sodass die Skipliste weiterhin korrekt für `find(key)` verwendbar bleibt.

Implementieren Sie eine Skipliste, welche Integer Werte als Keys verwalten kann und die Operationen `insert(key)`, `find(key)` und `erase(key)` unterstützt. Alle diese Operationen sollen erwartet in $\mathcal{O}(\log n)$ Zeit realisiert werden. Die Skipliste soll randomisiert arbeiten und die Element Höhen wie unten definiert geometrisch verteilt bestimmen. Die Skipliste soll doppelte Keys verwerfen, d.h. ein bestimmter Key kann maximal einmal in der Liste vorkommen.

Um das Ergebnis der Aufgabe deterministisch zu machen, besteht die Eingabe sowohl aus den Keys selber als auch den Zufallszahlen für die Bestimmung der Elementhöhen. Die Eingabeinstanz ist eine Textdatei aus $2m$ Integer Werten mit einem Wert pro Zeile, wobei sich sich Zeile $2k$ der k'te einzufügende Wert V_k befindet. In Zeile $2k+1$ befindet sich eine zufällig, gleich verteilte Zahl N_k aus dem Intervall $[0, 2^{30})$, aus der Sie wie folgt die Höhe des einzufügenden Elementes bestimmen:

$$\text{ElementHeight}(N_k) = \max\left(1,\ \min_p : \sum_{i=1}^{p} \frac{1}{2^p} \geq \frac{N_k}{2^{30}}\right).$$

D.h. die Höhe von Element k ist 1 wenn $N_k \in [0, 2^{29}]$, 2 wenn $N_k \in (2^{29}, 2^{29} + 2^{28}]$ usw.
Eine C Implementierung könnte wie folgt aussehen:

```
unsigned int getGeomHeight(const unsigned int N) {
  double DN = (double)N/(1<<30);
  double C = .5;
  unsigned int height = 0;
  while (DN>0) {
    DN -= C;
    C /= 2;
    height++;
  }
  return height;
}
```

Fügen Sie alle m Werte der Eingabe in Ihre Skipliste ein. Löschen Sie danach die Werte V_i mit $i \in [\frac{5}{10}m, \frac{6}{10}m)$.
Geben Sie an wie viel Speicher Ihre Datenstruktur insgesamt verbraucht und wie viel Byte Sie durchschnittlich pro Element brauchen. Geben Sie die Gesamtlaufzeit für die Einfüge- und Löschoperationen an. (Es reicht wenn Sie das gesamte Programm extern einmessen.)
Optional: Geben Sie die Datenstruktur für die `small.txt` Instanz am Ende auf geeignete Weise aus.

Auf der Algorithmik Webseite finden Sie zwei Dateien, von denen Sie **eine** benötigen:

(a) `instances.tgz`

(b) `generator.tgz`

`instances.tgz` ist groß (4.5MB) und enthält direkt `small.txt` und `large.txt`.
`generator.tgz` ist sehr klein und enthält ein C++ Programm sowie ein Makefile, welches diese beiden Instanzen direkt generiert.

Die Instanz `small.txt` enthält $m = 10$ und `large.txt` enthält $m = 500000$ Werte (und doppelt so viele Zeilen).

Zu dem Quellcode gehört ein Shellscript/Makefile, welches Ihr Programm compiliert und auf den beiden Instanzen ausführt. Geben Sie die Instanzen selber **nicht** mit ab. Gehen Sie stattdessen davon aus, dass diese im gleichen Ordner wie Ihre Programm liegen.