Lehrstuhl Algorithmen
und Komplexität
Prof. Dr. Fabian Kuhn
Sebastian Daum, Oleksii Saukh

Algorithmen und Datenstrukturen
(Informatik II) SS 2014

http://ac.informatik.uni-freiburg.de/teaching

UNI FREIBURG

Übungsblatt 5

Abgabe: Dienstag, 17. Juni, 16:00 Uhr

Laden Sie Ihre Lösungen für theoretische *Übungen im Format "loesung-xx-y.pdf" hoch, wobei "xx" die Nummer des aktuellen Übungsblattes ist (bei Bedarf mit führender Null) und "y" die Aufgabennummer, d.h., erzeugen Sie* pro (theoretische) Aufgabe *eine* .pdf*-Datei. Scans, Bilder von Scans etc. bitte zuerst ins pdf-Format umwandeln. Bei Programmieraufgaben müssen Sie – wann immer es relevant/durchführbar ist – Unit Tests sowie einen Style-check durchführen. Laden Sie außerdem eine Datei "erfahrungen.txt" hoch, in welcher Sie Ihre Erfahrungen und Meinungen zu dem Übungsblatt teilen.*

Aufgabe 1 (5 Punkte)

1. Zeigen Sie, wie man eine Familie von Bäumen $(T_n)_{n>0}$ konstruiert, für die folgendes gilt. Für ein beliebiges n sei T_n ein Baum mit $n' \in [n, 100n]$, dessen durchschnittliche *Knoten*tiefe $\bar{d} = \frac{1}{n'} \sum_{v \in T} d(v)$ in $\Theta(\log n)$ liegt, dessen Höhe allerdings $\omega(\log n)$ ist; $d(v)$ bezeichnet hierbei offenbar die Tiefe eines Knotens v.
2. Geben Sie die maximale *asymptotische* Höhe eines solchen Binärbaums an, für welchen $\bar{d} \in \Theta(\log n)$.

Aufgabe 2 (5 Punkte) Betrachten Sie einen binären Suchbaum, in welchem Elemente e mit gleichen Schlüssel $e.key$ abgespeichert werden können. Die asymptotische Laufzeit der *Insert(x)*-Prozedur, wenn n Elemente mit gleichen Schlüsseln im Suchbaum abgespeichert werden, ist $\Omega(n)$, denn das neue Element geht immer den gleichen Pfad entlang, und der Baum entartet zum Pfad. Um das zu verbessern, soll jetzt ein Test auf Schlüsselgleichheit eingebaut werden an jeder Stelle im Code, wo ein Schlüsselvergleich darüber entscheidet, ob links oder rechts entlanggegangen wird. Ist der Test positiv, so wird eine der folgenden Strategien implementiert.

1. In jedem Knoten y speichern wir einen Schalter *direction* ab, welcher die zwei Werte *left* und *right* annehmen kann und anfangs auf *left* steht. Wenn ein neues Element x in den Baum eingefügt wird, x an y vorbeikommt und der Test $x.key = y.key$ positiv ist, dann wird das Element x in die vom Schalter *direction* vorgegebene Richtung entlanggeschickt und der Schalter wird auf seine andere Stellung gekippt.
2. Wir hängen an jeden Knoten y eine Liste an und wenn x bei y vorbeikommt und $x.key = y.key$ gilt, dann wird x nicht als neuer Knoten in die Baumstruktur eingehängt, sondern stattdessen in die Liste bei y.
3. Wir speichern bei jedem Element ein Attribut *rank* ab, welches anfangs $\perp$ ist. Falls $x.key = y.key$, dann, wenn $x.rank = \perp$, dann würfelt x eine Zahl in $[0, 1)$ aus und speichert diese in $x.rank$ ab. Analog wird mit y verfahren, falls $y.rank = \perp$. Dann wird ein Vergleich durchgeführt und wenn $x.rank < y.rank$, dann wird x den linken Teilbaum entlanggeschickt, andernfalls den rechten.

Geben Sie für jede Strategie die asymptotische Laufzeit an, wenn n Elemente (nicht notwendigerweise identisch) mit gleichem Schlüssel in einen anfangs leeren Suchbaum eingefügt werden. Sie können auf Ergebnisse aus der Vorlesung verweisen, wo anwendbar.

Aufgabe 3 (10 Punkte) Implementieren Sie die Binary Search Tree (BSTree) Datenstruktur, in welcher jeder Knoten nur Integer Schlüssel und keine weiteren Daten enthält. Die Datenstruktur muss die folgenden Operationen unterstützen:

- insert(**int** key) - einen Schlüssel in den Baum einfügen. Existiert der Schlüssel bereits, so wird dieser überschrieben.
- **bool** find(**int** key) - nach einem Schlüssel suchen und TRUE oder FALSE wiedergeben.
- remove(**int** key) - einen Schlüssel aus dem Baum löschen.
- **int** size() - zurückgeben der Anzahl Knoten im Baum.
- **double** avgDepth() - zurückgeben der durchschnittlichen Tiefe aller Knoten im Baum.
- toArray() - zurückgeben eines Arrays, in welchem alle Knoten des Baums in der In-Order Reihenfolge abgespeichert sind.

Zu dieser Aufgabe stellen wir auch einige vorgefertigte Strukturdateien zur Verfügung, die Ihnen eventuell helfen können. Um Ihre Implementierung zu überprüfen, machen sie die folgenden drei Testläufe:

- Fügen Sie N sortierte Elemente in den Baum ein, d.h., in aufsteigender Reihenfolge. Wählen Sie dazu 100 verschiedene Werte für N und plotten Sie die durchschnittliche Knotentiefe in Abhängigkeit von N.
- Fügen Sie Elemente in den Baum ein, welche zufällig (gleichverteilt) aus dem Bereich $\{0, \ldots, \text{MAXINT}\}$ gewählt werden. Wiederholen Sie dies so lange bis der Baum N Knoten enthält. Wählen Sie erneut 100 aussagekräftige Werte für N, und machen Sie dann 5 Testläufe für jedes N und geben Sie von den resultierenden 5 durchschnittlichen Knotentiefen den Mittelwert zurück. Plotten Sie diese Mittelwerte in Abhängigkeit von N.
- Fügen Sie die folgende Liste von Elementen in den Baum ein: $a_i = ik \bmod N$, wobei $i = 0, \ldots, N-1$ und $k = \sqrt{N-1}$. Wählen Sie für diese Aufgabe wieder 100 Werte für N, lassen Sie die Werte allerdings eine um 1 inkrementierte Quadratzahl sein, so dass k eine natürliche Zahl ist. Also z.B. setzen Sie $k = 500, 510, 520, \ldots, 1490$ und entsprechend $N = 250001, 260101, 270401, \ldots, 2220101$. Plotten Sie wieder für jedes N (nicht k) die durchschnittliche Knotentiefe.