Karlsruher Institut für Technologie

Institut für Theoretische Informatik
T.T.-Prof. Dr. Thomas Bläsius
https://scale.iti.kit.edu

Übungsblatt 09

Algorithmen I – Sommersemester 2022

Abgabe im ILIAS bis 29.06.2022, 14:00 Uhr

Bitte beschrifte Deine Abgabe gut sichtbar mit Deinem Namen und Deiner Matrikelnummer. Achte insbesondere bei handschriftlichen Abgaben auf Lesbarkeit und genügend Platz für Korrektur-Anmerkungen. Die Abgabe erfolgt über das Übungsmodul in der Gruppe Deines Tutoriums im ILIAS. Gib Deine Ausarbeitungen in *einer* PDF-Datei ab. Achte darauf, effiziente Algorithmen zu formulieren, also solche mit möglichst geringer asymptotischer Laufzeit!
Wenn du die Korrektheit eines Algorithmus begründen oder dessen Laufzeit analysieren sollst, tue dies getrennt von der Beschreibung des Algorithmus.
Wenn nicht anders spezifiziert oder aus dem Kontext ersichtlich, bezeichnen wir mit Graph einen einfachen ungerichteten Graphen.

Aufgabe 1 - Nicht ganz Corona-konform (6 Punkte)

Dr. Meta schmuggelt sich undercover in die Mitarbeiterversammlung, um herauszufinden, wie stark die Miterarbeiterschaft nach der Ausdünnung noch vernetzt ist. Dazu zählt er, wie viele Mitarbeiter sich gegenseitig die Hand geben und macht eine überraschende Feststellung: Die Anzahl an Mitarbeitern, die einer ungeraden Anzahl ihrer Kollegen die Hand geben, scheint stets gerade zu sein. Dr. Meta braucht nun deine Hilfe, um diese Aussage zu beweisen.

1. Beschreibe, wie das Handgeben auf der Mitarbeiterversammlung als Graph modelliert werden kann. (1 Punkt)
   Hinweis: Was sind die Knoten? Zwischen welchen Knoten gibt es eine Kante? Ist der Graph gerichtet oder nicht? Brauchst du Kantengewichte?

2. Übersetze Dr. Metas Hypothese in eine Aussage über den Graph aus der ersten Teilaufgabe. (2 Punkte)

Nachdem wir das Problem als Graphproblem formuliert haben, geht es nun an den Beweis. Dieser soll sich aus folgenden Schritten zusammensetzen:

3. Sei $G = (V, E)$ ein Graph. Wir betrachten einen Algorithmus, der die Summe der Knotengrade aller Knoten berechnet. Dabei iteriert er alle Knoten und macht für jeden Knoten $v \in V$ folgendes. Alle zu v inzidenten Kanten werden iteriert und für jede gesehene Kante wird die Summe um 1 erhöht.
   Sei $e \in E$ eine beliebige Kante. Wie oft wird e vom Algorithmus gesehen? (1 Punkt)

4. Stelle eine Gleichung auf, die die Summe der Knotengrade in einem beliebigen Graphen G in Beziehung zur Anzahl der Kanten in G darstellt. (1 Punkt)

5. Benutze die Erkenntnisse aus den vorherigen Teilaufgaben, um die Aussage aus Teilaufgabe 2 zu beweisen. (1 Punkt)
   Hinweis: Überlege, wie man von der Summe aller Knotengrade zur Summe über Knoten mit ungeradem Grad kommt.

Lösung 1

1. Wir modellieren das Handgeben mt Hilfe eines ungerichteten ungewichteten Graphen. Jeder Mitarbeiter wird durch genau einen Knoten repräsentiert. Zwischen zwei Knoten existiert genau dann eine Kante, wenn sich die beiden entsprechenden Mitarbeiter gegenseitig die Hand gegeben haben.

2. Die Anzahl an Knoten mit ungeradem Grad ist stets gerade.

3. Genau zwei Mal, denn der Algorithmus sieht jede Kante von ihren beiden Endpunkten aus.

4. Sei $G = (V, E)$ ein Graph. Dann:

$$\sum_{v \in V} \deg(v) = 2|E|$$

5. Sei $G = (V, E)$ ein Graph. Wir betrachten die Menge $V' \subseteq V$ der Knoten mit ungeradem Grad. Wir wissen, dass

$$\sum_{v \in V} \deg(v) = \sum_{v \in V'} \deg(v) + \sum_{v \in V \setminus V'} \deg(v) = 2|E|$$

   Da $2|E|$ und $\sum_{v \in V \setminus V'} \deg(v)$ gerade sind, muss auch $\sum_{v \in V'} \deg(v)$ gerade sein. Das ist nur möglich, wenn $|V'|$ gerade ist.

Aufgabe 2 - Déjà vu (6 Punkte)

In dieser Aufgabe möchten wir eine Datenstruktur entwickeln, mit der wir auf einer dynamischen Menge von geordneten Elementen für gegebene k effizient das k-kleinste Element finden können. Analog zu balancierten Suchbäumen soll es möglich sein in $O(\log n)$ Zeit Elemente einzufügen, zu finden und zu löschen.

Um dies Umzusetzen möchten wir (a, b)-Bäume erweitern. Gegeben sei dazu ein (a, b)-Baum, der die Elemente aus einer Menge $S \subset \mathbb{N}$ der Größe n hält.

1. Beschreibe, welche zusätzlichen Informationen in den Knoten des (a, b)-Baumes gespeichert werden können, um in $O(\log(n))$ das k-kleinste enthaltene Element in S zu finden. (1 Punkt)

2. Begründe wieso das asymptotische Laufzeitverhalten von INSERT, FIND und REMOVE trotz der Ergänzungen aus Teilaufgabe 1 gleich bleibt. (3 Punkte)
   Hinweis: Überlege, wie sich die zusätzlichen Informationen in den Knoten durch Operationen auf dem (a, b)-Baum ändern können und wie sie aktuell gehalten werden können.

3. Beschreibe einen Algorithmus, der auf einem (a, b)-Baum mit deinen Erweiterungen aus Teilaufgabe 1 das k-kleinste Element in Zeit $\Theta(\log n)$ findet. Begründe die Korrektheit deines Algorithmus und dass er das geforderte Laufzeitverhalten hat. (2 Punkte)

Lösung 2

1. Jeder Knoten des (a, b)-Baums speichert, wie viele Blatt-Knoten bzw. Elemente der sortierten Folge über ihn erreichbar sind.

2. Der neue gespeicherte Wert kann sich ändern, wenn Elemente in den (a, b)-Baum eingefügt oder aus ihm entfernt werden. Das bedeutet, dass FIND durch unsere Ergänzung nicht beeinflusst wird, bei INSERT und REMOVE müssen jedoch Informationen in Knoten aktualisiert werden.
   Wird durch das Einfügen oder Löschen von Elementen in S die Invariante über die Knotengrade im Baum nicht verletzt, muss in jeder Lage des Baumes ein Knoten aktualisiert werden. Die resultierende Laufzeit ist weiterhin logarithmisch. Ansonsten müssen Knoten aufgespalten oder vereinigt werden. In diesem Fall lässt sich die Anzahl der erreichbaren Blätter eines Knoten jedoch in $\Theta(1)$ anhand seiner Kind-Knoten bestimmen, also ändert sich auch hier die asymptotische Laufzeit nicht.
   Also ändert sich die asymptotische Laufzeit von keiner der drei Operationen.

3. Wir traversieren den (a, b)-Baum. Sind über den Wurzel-Knoten weniger als k Blätter erreichbar, brechen wir ab. Ansonsten betrachten wir für jeden der Kind-Knoten die Anzahl l der über ihn erreichbaren Blätter und die Anzahl $l_<$ der Blätter, die über seine Geschwister-Knoten mit kleinerem Schlüssel erreichbar sind. Wir wählen den Knoten, für welchen $l_< < k$ und $l_< + l \geq k$ gilt. Dann setzen wir k auf $k - l_<$ und führen das Verfahren fort, bis wir bei einem Blatt-Knoten angelangt sind. Dieser enthält das gesuchte Element.
   Dieser Algorithmus benötigt Zeit in $\Theta(\log(n))$, da ein (a, b)-Baum mit logarithmischer Höhe traversiert wird und pro Ebene nur konstant (in Abhängigkeit von a

and b) viel Aufwand nötig ist, um den richtigen Kind-Knoten zu identifizieren. Der Algorithmus liefert das korrekte Ergebnis, da wir auf jeder Lage des (a, b)-Baumes den Knoten auswählen, über den das k-te Element erreichbar ist.

Aufgabe 3 - Baumschule (5 Punkte)

In dieser Aufgabe wollen wir uns mit der Konstruktion von $(2, 3)$-Bäumen beschäftigen.

1. Beschreibe, wie in Linearzeit aus einer gegebenen sortierten Folge ein $(2, 3)$-Baum konstruiert werden kann. Begründe, warum dein Algorithmus das geforderte Laufzeitverhalten hat. (3 Punkte)

2. Beschreibe, wie in Linearzeit zwei gegebene $(2, 3)$-Bäume zu einem $(2, 3)$-Baum zusammengefügt werden können. Begründe, warum dein Algorithmus das geforderte Laufzeitverhalten hat. (2 Punkte)

Lösung 3

1. Sei S die gegebene sortierte Folge. Zunächst fügen wir das Dummy-Element mit Schlüssel ∞ an das Ende von S ein. Nun betrachten wir die Elemente von S in aufeinander folgenden Paaren, d.h. zunächst das erste und das zweite Element, darauf das dritte und das vierte etc. Zu jedem dieser Paare legen wir einen neuen Knoten an, der die beiden Elemente als Kinder erhält. Als Schlüssel erhält der neue Knoten den Wert des größeren der beiden Elemente. Sollte S eine ungerade Anzahl an Elementen enthalten, setzt sich das letzten „Paar“ aus drei Elementen, darunter das Dummy-Element, zusammen.
   Nun wird die Kontruktion des Baumes rekursiv weitergeführt, bis wir bei der Wurzel angelangt sind. Dazu wird stets zu zwei aufeinander folgenden Knoten einer Lage ein Elter-Knoten auf der darüberliegenden Lage erstellt.
   Dieser Algorithmus hat eine Laufzeit in $\Theta(n)$, wobei n die Anzahl der Elemente in S ist. Das Anlegen eines Knoten benötigt konstanten Zeitaufwand. Auf der i-ten Lage des Baumes, wobei der Wuzelknoten auf Lage 0 liegt, müssen maximal $\frac{n}{2^{\lceil \log n \rceil - i}}$ Knoten angelegt werden, d.h. es ergibt sich ein Gesamtaufwand von

$$\sum_{i=0}^{\lceil \log n \rceil} \frac{n}{2^i} \cdot \Theta(1) \in \Theta(n)$$

   Der Wert der Summanden nimmt exponentiell ab, weswegen die Summe vom ersten Summanden, also n, dominiert wird. Damit ergibt sich diese Laufzeitabschätzung.

2. Seien n und m die Anzahl der Elemente, die in den beiden (a, b)-Bäumen enthalten sind. Wir nutzen aus, dass beide (a, b)-Bäume bereits sortierte Folgen darstellen. Diese können wir mit Hilfe von MERGE in $\Theta(n + m)$ zu einer sortierten Folge

zusammenfügen. Anschließend nutzen wir den Algorithmus aus der ersten Teilaufgabe, um auf der entstandenen Folge einen (a, b)-Baum zu konstruieren. Insgesamt benötigen wir dazu Zeit in $\Theta(n + m)$.

Aufgabe 4 - Geburtstag (3 Punkte)

Um immer sicherzugehen, dass du nie mehr vergisst, einem Freund zum Geburtstag zu gratulieren, hast du beschlossen, dich zu Beginn jedes Monats an alle anstehenden Geburtstage erinnern zu lassen. Da du dich im Rahmen der letzten Vorlesungen ausgiebig mit (a, b)-Bäumen befasst hast, beschließt du, sie zu diesem Zwecke zu verwenden. Die Geburtstage deiner n Freunde hast du bereits in einer sortierten Folge gesammelt, wobei die Elemente Tupel der Form (Geburtstag, Name) sind, welche nach der ersten Komponente (Geburtstag) und im Falle von gleichen Geburtstagen nach der zweiten Komponente (Name) sortiert sind. Aufgrund der hohen Verwechslungsgefahr können wir annehmen, dass die Folge keine Duplikate enthält. Die Tage des Jahres sind in aufsteigender Reihenfolge durchnummeriert, d.h. jeder Geburtstag wird durch eine natürliche Zahl im Intervall $[0, 365]$ repräsentiert.
Unter der Annahme, dass im angefragten Monat k Geburtstage liegen, beschreibe, wie du diese mit Hilfe eines (a, b)-Baumes, in $O(k + \log(n))$ bestimmen kannst. Begründe, warum dein Algorithmus das geforderte Laufzeitverhalten hat.

Lösung 4

Gegeben seien zwei Tage, dargestellt als die Zahlen $x, y \in \{0, \ldots, 365\}$ mit $x \leq y$. Um alle Einträge im (a, b)-Baum zwischen x und y zu bestimmen, rufen wir zunächst für beide FIND auf und erhalten die Elemente $x' \geq x$ und $y' \geq y$. Wir wissen, dass x' der erste von uns gesuchte Tag ist. Nun iterieren wir bei x' beginnend über die sortierte Folge bis zum letzten Element kleiner / gleich y. Ist $y' = y$, so ist y' dieses Element, ansonsten ist es der direkte Vorgänger von y' in der sortierten Folge.
Dieser Algorithmus hat eine Gesamtlaufzeit in $\Theta(k+\log(n))$, denn es wird zwei Mal FIND mit einer Laufzeit in $\Theta(\log(n))$ aufgerufen und das Auslesen der k gesuchten Werte in der sortierten Folge benötigt Zeit in $\Theta(k)$.