Technische Universität
Kaiserslautern

Fachbereich Informatik
AG Algorithmen und Komplexität

Ausgabedatum: 14.11.2013
Version: 2013-11-29 20:06

5. Übungsblatt zur Vorlesung Entwurf und Analyse von Algorithmen, WS 13/14

Abgabe: *Bis* ***Freitag****, 22.11.2013, 12:00 Uhr, Kasten im Treppenhaus 48-6.*

Der Fehler der Woche

Wir untersuchen $f(x) = \sin(x) + x$ und $g(x) = -\cos(x) + x$. Wir bestimmen

$$\lim_{x\to\infty} \frac{\sin(x)+x}{-\cos(x)+x} \overset{\text{Regel von L'Hôpital}}{=} \lim_{x\to\infty} \frac{\cos(x)+1}{\sin(x)+1} = ?\ ,$$

also kann Lemma 1.4 nicht angewendet werden. Folglich stehen f und g asymptotisch in keinem (sinnvollen) Zusammenhang.

Was ist falsch?

Basisaufgaben

B1: Queue in zyklischem Array

3 Punkte

Implementieren Sie die Prozeduren `DEQUEUE` und `ENQUEUE` für die Realisierung einer Queue mittels ringförmig geschlossenem Feld (siehe Buch).

Begründen Sie die Korrektheit Ihrer Implementierung.

Hinweis: Anders als bei *Entwurfs*aufgaben ist bei *Implementierungs*aufgaben tatsächlich detaillierter Code gefragt. Nutzen Sie bitte eine Sprache, die Ihr Übungsleiter versteht, also etwa eine der in der Vorlesung verwendeten. Hierbei ist in der Regel darauf zu verzichten, etwaige Bibliotheken die Arbeit machen zu lassen.

B2: Topologisches Sortieren

3 Punkte

Wir betrachten Sätze von Prozeduren, die von einem Compiler übersetzt werden sollen. Der Compiler ist schon etwas älter; um einen Prozeduraufruf $\texttt{P}(\texttt{x}_1, \ldots, \texttt{x}_n)$ zu übersetzen muss `P` schon übersetzt worden sein.

Wenden Sie jeweils beide Implementierungen zum topologischen Sortieren an, um eine funktionierende Übersetzungsreihenfolge zu finden oder festzustellen, dass es keine solche gibt.

Stellen Sie dazu als erstes die Prioritätenlisten auf, geben Sie dann geeignete Schnappschüsse der Variablenbelegungen an, aus denen der Verlauf des Algorithmus rekonstruiert werden kann, und bestimmen Sie schlussendlich das Ergebnis.

```
1  Prozedur a() {
2    Aufruf von b()
3    Aufruf von c()
4  }

6  Prozedur b() {
7    Aufruf von d()
8  }

10 Prozedur c() {
11   Aufruf von b()
12   Aufruf von d()
13 }

15 Prozedur d() {
16   Print("Fertig")
17 }
```

B3: Mengen als sortierte Listen

3 Punkte

Implementieren Sie die folgenden Mengenoperationen für Mengen in verketteter Listendarstellung analog zu der Implementierung des Durchschnitts aus der Vorlesung. Insbesondere soll auch hier die resultierende Liste wieder eine sortierte Liste sein.

Begründen Sie die Korrektheit Ihrer Implementierung. Geben Sie hierfür insbesondere an, welche Spezifikation Sie jeweils zugrundelegen; diese sollten konsistent mit dem gegebenen Beispiel (Schnitt) sein.

- Differenz ($A \setminus B$)

B4: Abzählen von (Di)Graphen

3 Punkte

Wir betrachten Graphen mit der Knotenmenge $\{1, 2, \ldots, n\}$. Bestimmen Sie

a) die Anzahl der Digraphen in Abhängigkeit von n.

b) die Anzahl der Graphen in Abhängigkeit von n.

Hinweis: Beachten Sie die genauen Definitionen von Graph und Digraph aus der Vorlesung!

Hinweis: Wir gehen hier gemäß Definition davon aus, dass Knoten eine Identität haben. Anders gesprochen, wir wollen isomorphe Graphen einzeln zählen und nicht zusammenfassen.

Aufbauaufgaben

12. Aufgabe

1 + 1 + 1 + 3 Punkte

Wir definieren zwei Alternativen zu $\mathcal{O}$ aus Definition 1.8:

- $\mathcal{O}'(f) = \{g : \mathbb{N}_0 \to \mathbb{R}_0^+ \mid \exists c > 0\, \forall n \geq 0 : g(n) \leq cf(n)\}$
- $\mathcal{O}''(f) = \{g : \mathbb{N}_0 \to \mathbb{R}_0^+ \mid \exists c, d > 0\, \forall n \geq 0 : g(n) \leq cf(n) + d\}$

Beweisen oder widerlegen Sie die folgenden Aussagen:

a) Für alle $f : \mathbb{N}_0 \to \mathbb{R}_0^+$ gilt $\mathcal{O}(f) = \mathcal{O}'(f)$.

b) Für alle $f : \mathbb{N}_0 \to \mathbb{R}_0^+$ gilt $\mathcal{O}(f) = \mathcal{O}''(f)$.

c) Für alle $f : \mathbb{N}_0 \to \mathbb{R}_0^+$ gilt $\mathcal{O}'(f) = \mathcal{O}''(f)$.

d) Benennen Sie eine reichhaltige und im Kontext der Laufzeitanalyse bedeutsame Menge von Funktionen, für die alle oben behaupteten Aussagen gelten. Skizzieren Sie Beweise für die fehlenden Identitäten.

Hinweis: Verwenden Sie genau die in Definition 1.8 gegebene Formulierung! Zusätzlich können Sie gerne weitere Definitionen aus anderen Quellen auf Äquivalenz untersuchen.

„$g : A \to B$“ ist hierbei eine (auch international) gängige Alternative zu unserer Notation „$g \in \mathrm{Abb}(A, B)$“; beides ist als „g ist Funktion, die Werte aus A auf Werte in B abbildet“ zu lesen.

13. Aufgabe

6 Punkte

Betrachten Sie eine Listenimplementierung, die ein Array für die Speicherung der Elemente nutzt. Dabei wird die Größe des Arrays immer dann verdoppelt, wenn das Array voll ist und ein weiteres Element eingefügt werden soll. Weil in typischen Speicherverwaltungen ein Array nicht mehr vergrößert werden kann, müssen wir ein neues Array doppelter Länge anlegen und die vorhandenen Elemente umkopieren. Zu Beginn – also im Falle der leeren Liste – starten wir mit einem (leeren) Array der Länge 1.

Berechnen Sie die amortisierten Kosten dafür, ein Element ans Ende der Liste anzuhängen, wenn n Anhängeoperationen (direkt) nacheinander ausgeführt werden. Dabei betrachten wir als Kosten die Schreibzugriffe, die beim erstmaligen Speichern eines Elements sowie bei Umkopieren anfallen, d. h. das Anhängen eines Elements in ein ausreichend großes Array verursacht Kosten von 1 und das Verlängern des Arrays von Länge i auf $i+j$ kostet i Schreibzugriffe. (Wir gehen davon aus, dass das Array nicht initialisiert wird; dann verbleibt als Aufwand das Kopieren der alten Elemente.)

Hinweis: Die hier analysierte Datenstruktur ist Basis für viele Implementierungen in populären Bibliotheken, zum Beispiel `java.util.ArrayList` in Java Runtime Library oder `std::vector` in C++ STL.

14. Aufgabe

8 + 2 Punkte

In der Vorlesung haben wir gesehen, wie man Mengen mit Bitvektoren – Arrays aus Booleans – implementieren kann, was schnelle Operationen auf Kosten von Speicher erlaubt. Ein unangenehmer Nachteil entsteht bei sehr großen Universen und kleinen Mengen, auf denen nur wenige Operationen ausgeführt werden; in diesem Fall ist die Initialisierungszeit – jeder Eintrag im Array muss explizit auf `0` gesetzt werden – dominant, nicht die eigentlichen Operationen.

a) Seien eine beliebige Menge $\mathcal{U}$ mit $|\mathcal{U}| = N \in \mathbb{N}$ und eine Bijektion $\varphi : \mathcal{U} \to [0..N-1]$ gegeben. Geben Sie eine Implementierung für Teilmengen von $\mathcal{U}$ auf Basis von Arrays an, die

   - die Basisoperationen – also Prüfung, ob ein Element in der Menge enthalten ist, sowie Entfernen und Hinzufügen von Elementen – ebenso in Zeit $\mathcal{O}(T_\varphi)$ ermöglicht, $T_\varphi : \mathcal{U} \to \mathbb{R}^+$ die Laufzeit von φ,
   - dabei $\mathcal{O}(mN)$ Speicher benötigt, aber
   - in Zeit $\mathcal{O}(1)$ initialisiert werden kann – was das eigentliche Ziel ist.

   m ist hierbei die *Wortbreite*, also die Anzahl Bits, die zur Speicherung eines Integers benötigt wird. Sie können annehmen, dass $m \in \Theta(\log N)$.

   Wir nehmen an, dass Speicherallokation in konstanter Zeit vom Betriebssystem durchgeführt wird. Sie können jedoch keine Annahmen darüber machen, welchen Inhalt eine nicht initialisierte Speicherstelle direkt nach der Allokation hat.

   Begründen Sie, warum die gewünschten Kriterien – also Korrektheit, Speicherbedarf und Laufzeiten – von Ihrer Datenstruktur erfüllt werden.

   Hinweis: Die Landausymbole sind hier für $N \to \infty$ zu lesen.

b) Skizzieren Sie, wie man mit Ihrer Datenstruktur aus a) die Vereinigungsoperation, also $C = A \cup B$, in Zeit $\mathcal{O}(|A| + |B|)$ durchführen kann. Sollte das nicht gehen, begründen Sie, warum nicht.