Karlsruher Institut für Technologie

Algorithmen 1 – Sommersemester 2018
Zuverlässige Softwaresysteme im Kontext der Automobilindustrie
`http://verialg.iti.kit.edu`

Prof. Dr. Carsten Sinz – Markus Iser

Probeklausur **20.06.2018**

A Rekurrenzen (1 Punkt)

Bestimmen Sie die Lösungen der folgenden Rekurrenzen im Θ-Kalkül mit dem Master-Theorem:

$$T(1) = 2, T(n) = 23n + 8T(n/4), n = 4^k, k \in \mathbb{N} \tag{1}$$

$$S(1) = 5, S(n) = 4S(n/4) + 1002n, n = 4^k, k \in \mathbb{N} \tag{2}$$

Lösung

$T(n) = \Theta\left(n^{1.5}\right)$, $S(n) = \Theta(n \log n)$

B Algorithmendesign (3 Punkte)

Entwerfen Sie einen Algorithmus, der für eine natürliche Zahl $n > 0$ die größte Zweierpotenz kleiner oder gleich n berechnet, also die Zahl 2^k mit $k = \lfloor \log_2 n \rfloor$. Dabei dürfen Sie nur die üblichen Operationen auf Ganzzahlen verwenden (Arithmetik, Vergleiche, Bit-Operationen, etc.). Gleitkomma-Operationen (wie log) und Tabellierung sind nicht erlaubt. Schreiben Sie Pseudocode und analysieren Sie die Laufzeit Ihres Algorithmus.

Ein Algorithmus mit Laufzeit $\Theta(\log n)$ gibt 1 Punkt. Ein Algorithmus mit besserer Laufzeit gibt 2 Punkte. Eine sinnvolle Invariante im Pseudocode gibt einen weiteren Punkt.

Lösung

Folgender Algorithmus ist in $\Theta(\log n)$

Algorithm 1: procedure *leqPower2*$(n : \mathbb{N}) : \mathbb{N}$

```
1 int x = 1
2 int j = 0
3 while 2 · x ≤ n do
4 |   x = 2 · x
5 |   j = j + 1
6 |   invariant: x = 2^j ≤ n
7 return x
```

Folgender Algorithmus ist in $\Theta(\log w)$, wobei w die Länge eines Machinenworts ist

Algorithm 2: procedure *leqPower2bitshift*$(n : \mathbb{N}) : \mathbb{N}$

```
1 int s = 1
2 int x = n
3 while s < w do
4 |   x = x OR (x BitShiftRight by s)
5 |   s = 2 · s
6 |   invariant: Σ_{i<=w} x_i ≥ s (mit x_i ist i-tes Bit in x)
7 assert: x = 2^{k+1} − 1 mit k = ⌊log_2 n⌋.
8 x = x − (x BitShiftRight by 1)
9 return x
```

C Binäre Heaps (1 Punkt)

Ein Array A[1..h] stelle einen impliziten binären Heap dar. Geben Sie sowohl den Index des Parents als auch des linken Kindes für das Element j an, gegeben dass beide existieren.

Lösung

parent(j) = $\lfloor j/2 \rfloor$, linkes Kind(j) = $2 \cdot j$

D Wächter (0,5 Punkt)

Nennen Sie ein Beispiel für die vereinfachende Wirkung von Wächterelementen (= Sentinels).

Lösung

- Sortieren durch Einsetzen.
- Suchen in (verketteten) Listen.

E Matrixmultiplikation (2 Punkte)

Gegeben sind zwei Matrizen $A, B \in \mathbb{N}^{n \times n}$. Das Produkt $C = (c_{i,j})$ von A und B ist definiert als $C = A\,B$ mit $c_{i,j} = \sum_{k=1}^{n} a_{i,k} b_{k,j}$. Geben Sie Pseudocode an, der das Produkt von A und B berechnet, und geben Sie die Laufzeit als $\Theta(...)$ an.

Dafür können Sie eine Struktur Matrix als Blackbox benutzen. Diese bietet die Funktionen $M.get(int\ x, int\ y)$, um das Element $m_{x,y}$ zu erhalten, $M.set(int\ x, int\ y, int\ v)$, um das Element $m_{x,y}$ mit v zu überschreiben, und eine Funktion $M.size()$, um die Zeilen-/Spaltenzahl der Matrix abzufragen. Diese Operationen können in $\Theta(1)$ durchgeführt werden.

Lösung

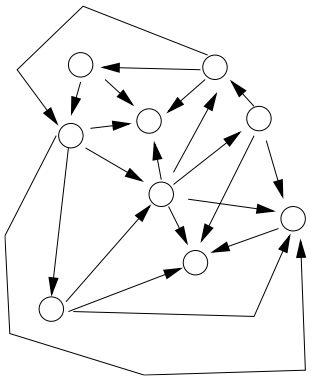
Algorithm 3: procedure *produktAB*$(A, B$: Matrix$)$: Matrix

```
1 Matrix M(A.size()), initialize Matrix M = 0                    // (NullMatrix)
2
3 for i = 1 to A.size() do
4     for j = 1 to A.size() do
5         int value = 0
6         for k = 1 to A.size() do
7             value = value + A.get(i,k) · B.get(k,j)
8         M.set(i,j, value)
9 return M
```

Die Laufzeit des Algorithmus ist $\Theta(n^3)$.

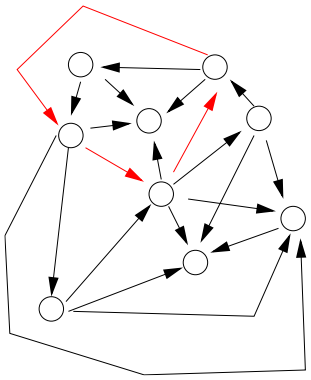
F Directed Acyclic Graphs (0,5 Punkt)

Ist dieser Graph ein DAG? Begründen Sie Ihre Antwort durch Markierungen in der Zeichnung oder durch Angabe einer topologischen Sortierung der Knoten!



Lösung

Der Graph ist kein DAG, da er einen gerichteten Kreis enthält:



Es reicht nicht nur hinzuschreiben, ob die Lösung richtig oder falsch ist. Eine richtige Begründung ist unbedingt erforderlich. Steht keine Begründung da oder ist die Begründung falsch gibt es 0 Punkte. Nur Markierungen im Graphen sind auch nicht ausreichend, es muß die richtige Antwort dabeistehen, sonst 0 Punkte.
Bei richtiger Antwort und ausreichender Begründung ist es natürlich auch ok, wenn im Graph nichts markiert ist.

G Hashtabelle (0,5 Punkt)

Gegeben sei eine Hashtabelle mit 10 Einträgen und die Hashfunktion $h(x) = x$ MOD 10. Fügen Sie die Elemente 18, 43, 78, 71, 8, 53, 2, 68, 81 in der gegebenen Reihenfolge in die Hashtabelle ein. Verwenden Sie Hashing mit linearer Suche (zyklisches Array). Geben Sie nur das endgültige Ergebnis an.

Lösung

0	1	2	3	4	5	6	7	8	9
8	71	2	43	53	68	81		18	78

H Binäre Heaps (0,5 Punkt)

Gegeben sei folgende Darstellung eines binären Heaps als Array mit den Indizes $1, \ldots, 5$:

1	2	3	4	5
2	7	3	14	12

Geben sie den Inhalt des Arrays an, nachdem *deleteMin* ausgeführt worden ist:

Lösung

1	2	3	4
3	7	12	14

I Algorithmendesign (3 Punkte)

Gegeben ist ein Array A : Array$[0..n-1]$ der Größe **n**. Es enthält Zahlen aus $\{\mathbf{0}, \ldots, \mathbf{n-1}\}$. Geben Sie einen *in-place* Algorithmus an, der eine Zahl ausgibt die doppelt vorkommt oder ausgibt, dass alle Zahlen verschieden sind. Die Laufzeit ihres Algorithmus darf $O(n)$ nicht übersteigen. Geben Sie auch die Best-Case Laufzeit ihres Algorithmus an.

Lösung

Algorithm 4: procedure *duplicates*(A : Array $[0..n-1]$ **of** $\mathbb{N}$, $x \in \mathbb{N}$)

```
1 for i = 0 to n − 1 do
2     while A[i] ≠ i do
3         if A[A[i]] == A[i] then
4             print "Duplikat A[i] gefunden"
5             return
6         else
7             swap(A[i], A[A[i]])
8 print "keine Duplikate"
9 return
```

Die Best-Case Laufzeit ist $O(1)$. Erreicht wird diese zum Beispiel bei einem Array das mit $0, 0, \ldots$ anfängt.

J Hashfunktionen

Die Grundmenge M bestehe aus positiven ganzen Zahlen. Jede Zahl $x \in M$ kann als $x = \sum_{i=0}^{\lceil \log_8 x \rceil} a_i \cdot 8^i$ mit Koeffizienten $a_i \in \{0, \ldots, 7\}$ dargestellt werden.

Wir betrachten folgende Hashfunktion: für $x \in M$ sei $h(x) = \left(\sum_{i=0}^{\lceil \log_8 x \rceil} a_i \right) \bmod m$.

J.1 Spielen (0,5 Punkt)

Finden Sie für $m = 13$ drei Zahlen, die alle den Hashwert 3 haben. Geben Sie diese als Dezimalzahlen an, wobei als solche eine Zahl zwei, eine drei und eine vier Stellen haben sollen und führende Nullen nicht erlaubt sind.

Lösung

Beispielsweise $24 = [30]_8$, $129 = [201]_8$ und $1025 = [20001]_8$

J.2 Bewerten (0,5 Punkt)

Nennen Sie einen stichhaltigen Grund, warum diese Hashfunktion nicht gut ist.

Lösung

Kollisionen sind (sehr) leicht zu finden. Potenzielle Angreifer können also leicht Zahlenfolgen generieren, für die die erwartete Laufzeiten der Vorlesung nicht gelten.

K Listen und Felder (1 Punkt)

Nennen Sie zwei Vorteile von beschränkten Feldern gegenüber einfach verketteten Listen.

Lösung

- keine Zeiger
- Cache-effizientes Iterieren
- effizienter beliebiger/wahlfreier Zugriff
- einfach

L Heap Eigenschaft (0,5 Punkt)

Wie lautet die Heap-Eigenschaft?

Lösung

Bäume mit $\forall v, \mathsf{parent}(v) \leq v$

M Hashing (1 Punkt)

Nennen Sie einen Vorteil und einen Nachteil von Hashing mit linearer Suche gegenüber Hashing mit verketteten Listen.

Lösung

- *Vorteile:* platz-effizient, cache-effizient, einfach
- *Nachteile:* insert nicht in O(1), für dichte Hashtabellen schlechter, keine Referentielle Integrität

N Sortieren (1 Punkt)

Nennen Sie einen Vorteil von Radix-Sort gegenüber Mergesort und von Mergesort gegenüber Quicksort.

Lösung

- *Vorteil Radix-Sort gegenüber Mergesort:* Zeit in $O(n)$ statt $\Theta(n \log n)$ (bei konstanter Stelligkeit)
- *Vorteile Mergesort gegenüber Quicksort:* stabil, $O(n \log n)$ bzw. $n \log n + O(n)$ Vergleiche im worst-case

O Analyse (1 Punkt)

Welche der folgenden Operationen eines unbeschränkten Feldes liegen im schlimmsten Fall in $\Theta(n)$ (ohne Amortisierung): *pushBack*, Folge von n *pushBack*, $[\cdot]$ (Elementzugriff), $|\cdot|$ (Größe)?

Lösung

pushBack, Folge von n *pushBack*

Insbesondere sind $[\cdot]$ *(Elementzugriff) und* $|\cdot|$ *(Größe) falsche Antworten.*

P *recMult*-Algorithmus (0,5 Punkt)

Erläutern Sie kurz die Hauptidee des rekursiven *recMult*-Algorithmus der Vorlesung zur Multiplikation von zwei n-Bit Ziffern a, b.

Lösung

Im Wesentlichen handelt es sich um einen Darstellungstrick:
Es sei $a = a_1 B^k + a_0$, $b = b_1 B^k + b_0$. Dann $a \cdot b = a_1 b_1 B^{2k} + (a_1 b_0 + b_1 a_0) B^k + a_0 b_0$. Daraus lässt sich dann der Algorithmus ableiten.

Hier sind vermutlich auch andere Antworten denkbar, z.B. etwas wie: Aufteilen in Zahlen der Länge $n/2$ Bit, diese dann Mutiplizieren, usw. . .

Q Verkettete Hashtabellen (2 Punkte)

Wurden bei den verketteten Hashtabellen in der Vorlesung einfach- oder doppelt-verkettete Listen verwenden? Geben Sie ~~zwei~~ einen *stichhaltigen Grund* an, warum diese Wahl besser als die Alternative ist.

Lösung

Es wurden immer einfach-verkettete Listen verwendet. Doppelt-verkettete sind natürlich auch denkbar, aber einfach-verkettete Listen brauchen nur einen Pointer in den Zellen einer sowieso schon großen Hashtable.