Σ: 20/30

AlgoDat: 7.Hausaufgabe (07.06.23) - Cora Zeitler

Donnerstag, 1. Juni 2023 02:15

8/8

Aufgabe 3:
Fügen Sie die Zahlenfolge $42, 14, 83, 25, 66, 37, 33$ in eine Hash-Tabelle der Länge $m = 11$ mit offenem Hashing und der primären Hash-Funktion $h(k) = k \mod m$ ein. Verwenden Sie bei Kollisionen:

(a) Lineares Sondieren,

(b) Doppeltes Hashing mit $h_1(k) = h(k)$ und $h_2(k) = 1 + (k \mod (m-1))$.

(8 Punkte)

3a) geg: - Zahlenfolge (42, 14, 83, 25, 66, 37, 33)
- Tabellenlänge: $m = 11$
- Hashfkt: $h(k) = k \mod m$

Berechnung:
$h(42) = 42 \mod 11 = (33)\ 9$ (Index im Array)
$h(14) = 14 \mod 11 = (11)\ 3$
$h(83) = 83 \mod 11 = (77)\ 6$
$h(25) = 25 \mod 11 = (22)\ 3$ // wenn Zahl schon vorhanden, dann wird in Tabelle nur aufgerutscht
$h(66) = 66 \mod 11 = (66)\ 0$
$h(37) = 37 \mod 11 = (33)\ 4$
$h(33) = 33 \mod 11 = (33)\ 0$

Hashtabelle:

Index	0	1	2	3	4	5	6	7	8	9	10
Array	**66**	**33**		**14**	**25**	**37**	**83**			**42**	
Versuche	1	2		1	2	2	1			1	

√

3b) geg: - Zahlenfolge
- Tabellenlänge: $m = 11$
- Hashfkt: $h_1(k) = h(k)$ und
$h_2(k) = 1 + (k \mod (m-1))$ // Doppeltes Hashing

Berechnung: (immer 1. Versuch, wenn nicht ffetriert, dann 2. Versuch, usw.)
(Position schon vorhanden)

↳ Formel für Versuche: $h(k,i) = (h_1(k) + i \cdot h_2(k)) \mod 11$ // i = Versuche −1

↳ bei 1. Versuch: i=0 ↴ nur $h_1(k)$

$h_1(42) = 42 \mod 11 = (33)\ \underline{9}$ (Index im Array)
$h_1(14) = 14 \mod 11 = (11)\ \underline{3}$
$h_1(83) = 83 \mod 11 = (77)\ \underline{6}$
$h_1(25) = 25 \mod 11 = (22)\ 3$ ↴ Position schon belegt

↳ $h(25,1)$ (2.Vers.) $= (h_1(25) + 1 \cdot h_2(25)) \mod 11$ // $h_2(25) = 1 + (25 \mod 10) = 6$
$= (3 + 1 \cdot 6) \mod 11$
$= 9 \mod 11 = 9$ ↴ Position schon belegt

↳ $h(25,2) = (h_1(25) + 2 \cdot h_2(25)) \mod 11$
$= (3 + 2 \cdot 6) \mod 11$
$= 15 \mod 11 = (11)\ \underline{4}$

$h_1(66) = 66 \mod 11 = (66)\ \underline{0}$
$h_1(37) = 37 \mod 11 = (33)\ 4$ ↴ Position schon vorhanden

$h_1(66) = 66 \bmod 11 = (66)\ \underline{0}$

$h_1(37) = 37 \bmod 11 = (33)\ 4$ ↴ Position schon vorhanden

↳ $h(37,1) = (h_1(37) + 1 \cdot h_2(37)) \bmod 11$ // $h_2(37) = 1 + (37 \bmod 10) = 8$

$= (\ 4\ + 1 \cdot 8\ ) \bmod 11$

$= 12 \bmod 11 = (11)\ \underline{1}$

$h_1(33) = 33 \bmod 11 = (33)\ 0$ ↴ Position schon vorhanden

↳ $h(33,1) = (h_1(33) + 1 \cdot h_2(33)) \bmod 11$ // $h_2(33) = 1 + (33 \bmod 10) = 4$

$= (\ 0\ + 1 \cdot 4\ ) \bmod 11$

$= 4 \bmod 11 = 4$ ↴ Position schon vorhanden

↳ $h(33,2) = (h_1(33) + 2 \cdot h_2(33)) \bmod 11$

$= (\ 0\ + 2 \cdot 4\ ) \bmod 11$

$= 8 \bmod 11 = \underline{8}$

Hashtabelle:

Index	0	1	2	3	4	5	6	7	8	9	10
Array	**66**	**37**		**14**	**25**		**83**		**33**	**42**	
Versuche	1	2		1	3		1			1	

✓

Aufgabe 1:
In einem Rot-Schwarz-Baum soll zu jedem inneren Knoten x die Anzahl der inneren Knoten im Teilbaum mit Wurzel x verwaltet werden. Zeigen Sie, wie die Operationen `INSERT` und `DELETE` ausgeführt werden können, ohne die asymptotischen Laufzeiten im worst-case zu verschlechtern.

(5 Punkte)

7/5

-> man muss sicherstellen, dass die Anzahl der inneren Knoten im Teilbaum mit Wurzel x korrekt verwaltet werden
-> dazu muss die Anzahl der inneren Knoten aktualisiert werden, wenn Knoten eingefügt oder gelöscht werden

Vorgehensweise

INSERT-Operation:

a. führe normale INSERT-Operation durch, um den neuen Knoten einzufügen

b. gehe den Pfad von der Wurzel zum neu eingefügten Knoten hinunter und aktualisiere die Anzahl der inneren Knoten in jedem Knoten entlang des Pfads Wie?
-> beginne dabei beim ersten Vorgänger des neu eingefügten Knotens, dessen Anzahl der inneren Knoten sich ändert

c. erreichbar, indem man einen Zeiger auf den Vorgänger des neu eingefügten Knotens verwendet und den Pfad nach oben geht, um die Anzahl der inneren Knoten in jedem Knoten zu aktualisieren
-> wenn der Zeiger auf einen Knoten zeigt, dessen Anzahl der inneren Knoten sich nicht ändert, kann der Prozess abgebrochen werden, da Anzahl der inneren Knoten in den darüberliegenden Knoten ebenfalls unverändert bleibt

d. durch Aktualisierung wird die Anzahl der inneren Knoten im Teilbaum mit Wurzel x korrekt gehalten

DELETE-Operation:

a. führe eine normale DELETE-Operation durch, um den zu löschenden Knoten zu entfernen

b. gehe den Pfad von der Wurzel zum gelöschten Knoten hinunter und aktualisiere die Anzahl der inneren Knoten in jedem Knoten entlang des Pfads
-> beginne dabei beim ersten Vorgänger des gelöschten Knotens, dessen Anzahl der inneren Knoten sich ändert

c. erreichbar, indem man einen Zeiger auf den Vorgänger des gelöschten Knotens verwendet und den Pfad nach oben geht, um die Anzahl der inneren Knoten in jedem Knoten zu aktualisieren
-> wenn der Zeiger auf einen Knoten zeigt, dessen Anzahl der inneren Knoten sich nicht ändert, kann der Prozess abgebrochen werden, da Anzahl der inneren Knoten in den darüberliegenden Knoten ebenfalls unverändert bleibt

d. durch Aktualisierung wird die Anzahl der inneren Knoten im Teilbaum mit Wurzel x korrekt gehalten

Was ist, wenn rotiert wurde?

Aufgabe 4:
Gegeben sei ein Schlüssel k, eine Hashtabelle $[0, 1, \ldots, m-1]$ und eine Hashfunktion h'.
Zum Einfügen eines Schlüssels wird das folgende Verfahren angewendet;

1: $\text{pos} = h'(k); i = 0$

2: Teste Position pos; falls leer $\rightarrow$ fertig

3: $i = (i+1) \bmod m; \text{pos} = (\text{pos} + i) \bmod m$; goto 2.

Zeigen Sie, dass das angegebene Verfahren eine Instanz des offenen Hashing mit quadratischen Sondieren darstellt und somit in das Schema $h(k,i) = [h'(k) + c_1 i + c_2 i^2] \bmod m$ passt. Geben Sie die Konstanten c_1 und c_2 an.

(4 Punkte)

0/4

4) Schlüssel h, Hashtabelle $[0,1,\ldots,m-1]$ (länge = m), Hashfkt k

→ Quadratisches Hashing: $h(k,i) = (h'(k) + c_1 \cdot i + c_2 \cdot i^2) \bmod m$; c_1, c_2 ... konst, $c_2 \neq 0$

$i = (i+1) \bmod m$

$\text{pos} = (\text{pos} + i) \bmod m$

↳ $\text{pos} = (\text{pos} + ((i+1) \bmod m)) \bmod m$

$= (\text{pos} + i + 1) \bmod m$

und weiter?

Aufgabe 2:
Wir wollen auf Rot-Schwarz-Bäumen die folgenden beiden Operationen ausführen:

- GreaterThan(a) zur Ausgabe aller Schlüssel x_i, die in der Datenstruktur gespeichert sind und für die $a \leq x_i$ gilt.
- Between(a, b) zur Ausgabe aller Schlüssel x_i, die in der Datenstruktur gespeichert sind und für die $a \leq x_i \leq b$ gilt.

(a) Beschreiben Sie eine Implementierung von GreaterThan, die es erlaubt, die Suchanfrage für Eingabe a in $O(m + \log n)$ Zeit zu beantworten. Dabei sind n die Anzahl der in der Datenstruktur gespeicherten Schlüssel und m die Anzahl der ausgegebenen Schlüssel.

(b) Wie kann die Operation Between(a, b) in $O(m + \log n)$ Zeit (n, m wie oben) ausgeführt werden?

(13 Punkte)

nach 7/7

GREATER-THAN(a)

-Man geht von der Wurzel startend nach unten durch den Baum
-> dabei prüft man die Kinder der Wurzel

Was ist mit der Wurzel selbst?

1. sind beide Kinder < a, dann fährt man mit dem rechten Kind fort

2. sind beide Kinder >= a, dann INORDER-TREEWALK für beide Teilbäume
-> Listen werden einfach aneinandergehangen, da nicht sortiert werden muss

3. rechtes Kind >= a, dann INORDER-TREEWALK für den Teilbaum des Kindes
-> rekursiv weiter durch den linken Teilbaum gehen mit den Fällen 1 bis 3, solange bis man unten angekommen ist (v)

=>Laufzeit: -O(log n) für Durchlauf des Baumes
-O(m) insgesamt für Treewalk
--> O(m + log n)

✓

BETWEEN(a,b): Ausgabe ist Liste A

-Man geht von der Wurzel aus nach unten durch den Baum

-jetzt rekursiv: BETWEEN(x,a,b):

1. if x > a, dann BETWEEN(left[x], a, b)
2. if a <= x <= b, dann A += x
3. if x < b, dann BETWEEN(right[x], a, b)

nächstes Mal lieber mit Worten beschreiben, die Notation könnte sonst sehr verwirren

4/6

✓

Laufzeit?