FernUniversität in Hagen

Lösungsvorschläge zur Hauptklausur „1661 Datenstrukturen I"

10.08.2013

Aufgabe 1 Reversi-Algebra

(a)

algebra *reversi*
sorts *Color, Board, Move, Boolean*
ops

rev	: *Color*	→ *Color*
create	:	→ *Board*
isValid	: *Board* × *Move*	→ *Boolean*
isFinished	: *Board*	→ *Boolean*
winner	: *Board*	→ *Color*
move	: *Board* × *Move*	→ *Board*

end *reversi.*

(b)

Wir definieren zunächst zwei Hilfsmengen *Index* und *Field*:

$Index = \{1, 2, 3, 4, 5, 6, 7, 8\}$

$Field = \{(row, column, col) \mid row, column \in Index, col \in Color\}$

Nun definieren wir die Trägermengen der Sorten:

$Boolean = \{\text{true}, \text{false}\}$

$Color = \{\text{Black}, \text{White}, \text{None}\}$

$Move = (Index \times Index) \cup \{\text{pass}\}$

$$
\begin{aligned}
Board = \{(F, C) \mid\ & C \in \{\text{Black}, \text{White}\}, F \subset Field \wedge \\
& |F| = 64 \wedge \\
& \forall (r_1, c_1, co_1), (r_2, c_2, co_2) \in F : (r_1 = r_2 \wedge c_1 = c_2) \Rightarrow co_1 = co_2 \wedge \\
& \forall (r_1, c_1, co_1) \in F : co_1 \neq \text{None} \Rightarrow \\
& \qquad \exists (r_2, c_2, co_2) \in F, d_1, d_2 \in \{-1, 0, 1\} : \\
& \qquad\quad co_2 \neq \text{None} \wedge \\
& \qquad\quad r_1 \neq r_2 \vee c_1 \neq c_2 \wedge \\
& \qquad\quad r_2 = r_1 + d_1 \wedge c_2 = c_1 + d_2 \\
\}&
\end{aligned}
$$

Das Board besteht somit aus zwei Komponenten, dem Spielfeld und der Farbe desjenigen Spielers, der als nächstes an der Reihe ist. Die Spielerfarbe kann hier nur schwarz oder weiß sein. Das Spielfeld selbst ist eine Menge von Feldern, die genau 64 Elemente enthält (Zeile 2 der Formel). Somit werden in dieser Lösung auch nicht besetzte Felder repräsentiert. Zeile 3 sichert, dass jede Zeilen/Spaltenkombination im Spielfeld eindeutig ist. Die Zeilen 4-8 sichern die Existenz mindestens eines Steins als direkten

Nachbarn eines Steins, d.h. es kann keine Steine geben, die vollständig von leeren Feldern umgeben sind.

(c)

$$rev(c) = \begin{cases} \text{Black} & \text{falls } c = \text{White} \\ \text{White} & \text{falls } c = \text{Black} \\ c & \text{sonst} \end{cases}$$

$create() = (F, \text{Black})$
mit:

$$\begin{aligned} F = \; & \{(i, j, \text{None}) \mid \; i, j \in Index \wedge \\ & (i = 4 \vee i = 5) \Rightarrow (j \neq 4 \wedge j \neq 5) \wedge \\ & (j = 4 \vee j = 5) \Rightarrow (i \neq 4 \wedge i \neq 5)\} \\ & \cup \{(4, 4, \text{White}), (4, 5, \text{Black}), (5, 4, \text{Black}), (5, 5, \text{White})\} \end{aligned}$$

Die Funktion *create* erzeugt ein neues Board bestehend aus einem Spielfeld sowie der aktuellen Spielfarbe schwarz. Das Spielfeld wird durch die Vereinigung zweier Mengen gebildet. Die erste Menge enthält alle bis auf die 4 mittleren Felder. Alle Felder dieser Menge sind leer. Die zweite Menge enthält genau die fehlenden 4 Felder mit den in der Aufgabe angegebenen Farben.

$$winner((F, C)) = \begin{cases} \text{None} & \text{falls } \neg isFinished((F, C)) \\ & \quad \vee |\{(i, j, \text{Black}) \in F\}| = |\{(i, j, \text{White}) \in F\}| \\ \text{Black} & \text{falls } isFinished((F, C)) \\ & \quad \wedge |\{(i, j, \text{Black}) \in F\}| > |\{(i, j, \text{White}) \in F\}| \\ \text{White} & \text{sonst} \end{cases}$$

Falls das Spiel noch nicht beendet wurde oder die gleiche Anzahl schwarzer wie weißer Steine auf dem Spielfeld liegt, gibt es keinen Gewinnner (Fall 1 der Formel). Wurde das Spiel beendet und es gibt mehr schwarze als weiße Steine, so hat der Spieler mit der Farbe schwarz gewonnen (Fall 2). In allen anderen Fällen (Spiel ist beendet und es gibt mehr weiße als schwarze Steine) ist Spieler „weiß" der Gewinner (Fall 3).

$$\begin{aligned} isValid((F, C), (r, c)) = \; & (r, c, \text{None}) \in F \wedge \\ & \exists i, j \in \{-1, 0, 1\}, a \in Index, a > 1 : \\ & \quad (r + i * a, c + j * a, C) \in F, \\ & \quad \forall b \in Index, b < a : (r + i * b, c + j * b, rev(C)) \in F \end{aligned}$$

Diese Formel prüft die Gültigkeit eines normalen Zugs, der durch die Zielkoordinaten des Steins gegeben ist. In Zeile 1 der Formel wird gesichert, das die Zielposition nicht bereits belegt ist. Die restlichen Zeilen sichern ab, dass mindestens ein gegnerischer Stein umgefärbt wird. Dazu muß es in eine beliebige Richtung (dargestellt durch i, j) einen Stein geben, der mindestens 2 Felder entfernt ist ($a > 1$) und die gleiche Farbe hat wie der gesetzte Stein (Zeile 3). Weiterhin müssen alle Felder zwischen dem neu gesetzten und diesem Feld ($1 \leq b < a$) die gegnerische Farbe aufweisen (Zeile 4).

$$isValid(B, \text{pass}) = \forall r, c \in Index : \neg isValid(B, (r, c))$$

Ein *pass*-Zug ist nur dann gültig, wenn es keinen anderen gültigen Zug gibt, d.h. keine Zielposition zu einem gültigen Zug führt.

$$isFinished((F, C)) = isValid((F, C), \text{pass}) \wedge isValid((F, rev(C)), \text{pass})$$

Das Spiel gilt als beendet, wenn beide Spieler nur noch passen können.

$$move((F, C), \text{pass}) = \begin{cases} (F, C) & \text{falls } \neg isValid((F, C), \text{pass}) \\ (F, rev(C)) & \text{sonst} \end{cases}$$

Hier wird der *pass*-Zug durchgeführt. Ist dieser ungültig, so ändert sich nichts. Ansonsten wird die Farbe des aktuellen Spielers geändert.

$$move((F, C), (r, c)) = \begin{cases} (F, C) & \text{falls } \neg isValid((F, C), (r, c)) \\ (F', rev(C)) & \text{sonst} \end{cases}$$

mit:

$F' = \{(a, b, c_1) | (a, b, c_2) \in F\}$

und:

$$c_1 = \begin{cases} C & \text{falls} \quad a = r, b = c \\ C & \text{falls} \quad \exists i, j \in \{-1, 0, 1\}, d \in Index : \\ & \qquad a = r + i * d, b = c + j * d \wedge \\ & \qquad \exists e \in Index, e > d : (r + i * e, c + j * e, C) \in F \wedge \\ & \qquad\quad \forall f \in Index, f < e : (r + i * f, c + j * f, rev(C)) \in F \\ c_1 & \text{sonst} \end{cases}$$

Ein Zug mit Zielfeldangabe wird ähnlich zu einem *pass*-Zug durchgeführt (1. Formel). Ist der Zug ungültig, wird das unveränderte Board zurückgeliefert. Im anderen Fall wird die Farbe des Gegners gewechselt. Zusätzlich zum *pass*-Zug wird auch noch die Spielfeldkonfiguration geändert. Dabei werden lediglich Farben geändert. Dabei wird die aktuelle Spielfarbe verwendet, wenn es sich um das Zielfeld des Zugs handelt (Fall 1) oder sich der Stein in einer umzufärbenden Kette befindet (Fall 2). Bei allen anderen Feldern gibt es keine Änderung der Farbe (Fall 3). In Fall 2 wird geprüft, dass das zu prüfende Feld (a, b) vom Zielfeld des Zugs (r, c) erreicht werden kann. Geht man nun von der Zielposition in Richtung des zu prüfenden Steins über diesen hinaus (e Schritte), muss es einen Stein geben, der der gesetzten Farbe entspricht (Zeile 3 von Fall 2). Zwischen diesem Stein und dem neu gesetzten Stein müssen alle Steine die gegnerische Farbe aufweisen (Zeile 4 von Fall 2). Das schließt auch das zu prüfende Feld mit ein.

Aufgabe 2 Hashing

(a)

Siehe Spalte k in der Tabelle unter Teilaufgabe (b).

(b)

Übersicht der berechneten Hashwerte und Behälternummern für die Städte:

Stadt	k	$h(k)$	$h_1(k)$	$h_2(k)$	$h_3(k)$
Wolfsburg	50	0			
Essen	43	3			
Neuenahr	40	0	3	6	
Frankfurt	25	5			
Leverkusen	39	9			
Potsdam	51	1			
Sindelfingen	42	2			
Jena	29	9	2	5	8
Duisburg	34	4			

(c)

Es folgt die Hashtabelle:

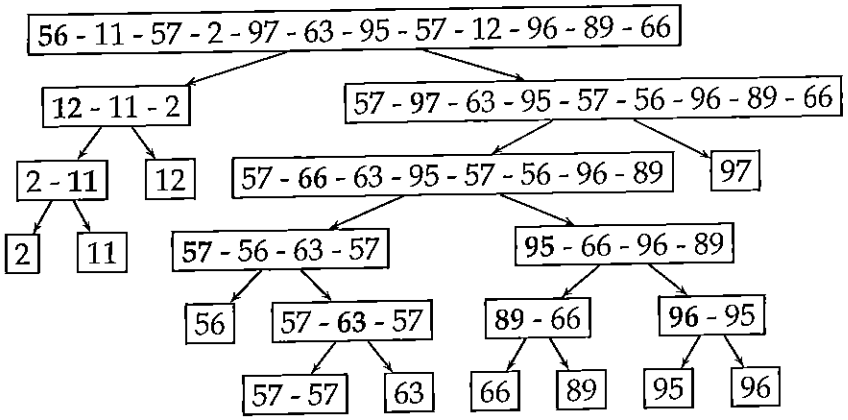
0	Wolfsburg	3	Essen	6	Neuenahr	9	Leverkusen
1	Potsdam	4	Duisburg	7			
2	Sindelfingen	5	Frankfurt	8	Jena		

(d)

Die Verwendung von geschlossenem Hashing in der Form der Aufgabenstellung ist nicht sinnvoll. Durch die hohe Auslastung von 90% ergeben sich lange Suchpfade, z.B. müssen bei der Suche nach Jena 4 Behälter durchsucht werden.

Aufgabe 3 Quicksort

(a)



(b)

Quicksort ist nicht stabil, da die Reihenfolge von Elementen mit gleichem Schlüssel durch das Vertauschen von Datensätzen über große Entfernungen i.A. verändert wird.

(c)

In diesem Fall beträgt die Laufzeit von Quicksort $O(n^2)$, da der Aufrufbaum zu einer Liste entartet.

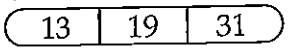
(d)

Bubble Sort, Insertion Sort.

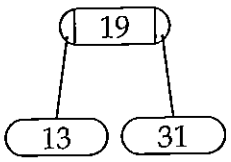
Aufgabe 4 B-Baum

(a)

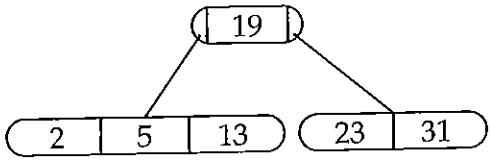
Beim Einfügen des Elements 19 kommt es zum Überlauf, der durch einen Split beseitigt wird:



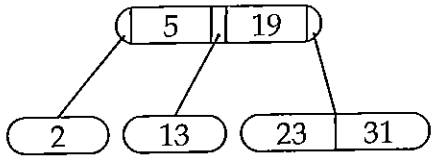
Baum nach dem Split:



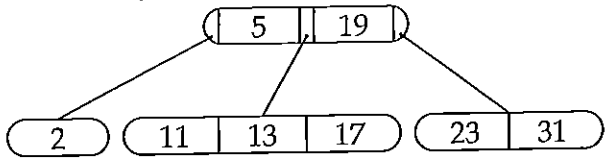
Nach dem Einfügen der 5 ist ein weiterer Split notwendig:



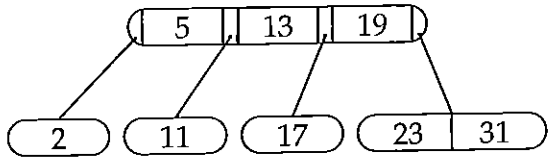
Baum nach dem Split:



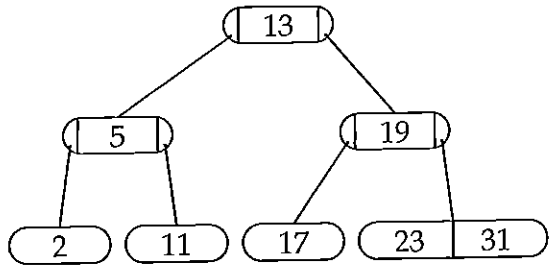
Das Einfügen des Elements 17 erfordert den nächsten Split:



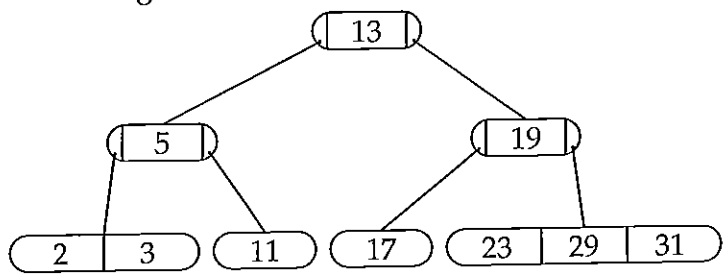
Baum nach dem Split:



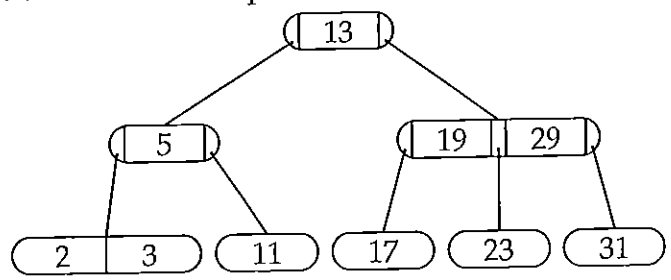
Es ist ein weiterer Split notwendig, da es zum Überlauf in der Wurzel kommt:



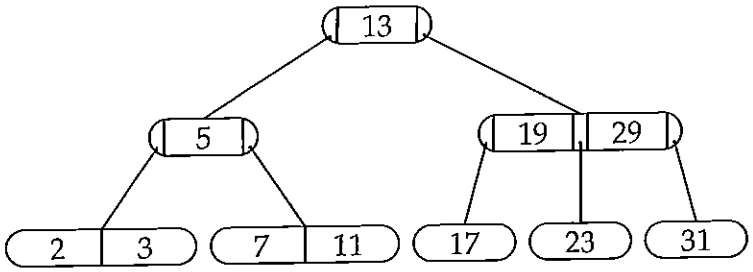
Das Einfügen des Elements 29 erfordert erneut einen Split:



Baum nach dem Split:

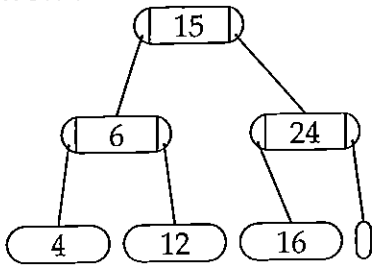


Das Einfügen der 7 verläuft ohne Split-Operation. Der Baum nach Einfügen aller Elemente ist der folgende:

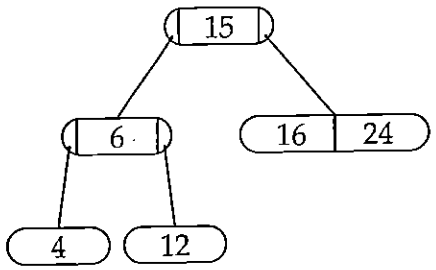


(b)

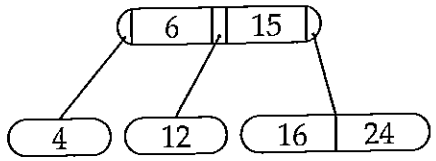
Das Löschen des Elements 30 führt zu einem Unterlauf, der durch Merge beseitigt werden muss:



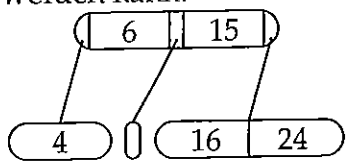
Baum nach Merge:



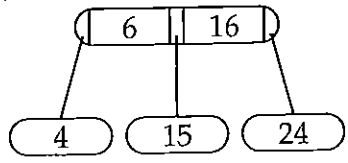
Es ist eine weitere Mergeoperation notwendig, die zu folgendem Baum führt:



Das Löschen der 12 führt wieder zu einem Unterlauf, der mittels Balance beseitigt werden kann:



Baum nach Balance:



Aufgabe 5 Deckblatt

Hier erhalten Sie den Punkt, wenn Sie beide Klausurdeckblätter korrekt und vollständig ausgefüllt haben, also Namen, Matrikelnummer und Adresse korrekt eingetragen und genau diejenigen Aufgaben markiert haben, die Sie auch bearbeitet haben.