

1 Dateien mit festen Datensatzgrößen

Liste freier Plätze
zum Einfügen

Gegeben sei die folgende Datei mit Datensätzen fester Größe und einer Free-List:

header				→ 3
record 0	10101	Sromovasam	Comp. Sci.	65000
record 1	15151	Mozart	Music	40000
record 2	22222	Einstein	Physics	95000
record 3				→ 4
record 4				→ 7
record 5	33456	Gold	Physics	87000
record 6	58583	Califieri	History	62000
record 7				→ ⊥
record 8	87543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

↖ zeigt auf den
ersten freien
Datensatz

zeigt zur Ende der Liste

Notation:

→ 1 bezeichnet einen Zeiger auf Datensatz 1.

Zeigen Sie die Struktur der Datei nach jedem der folgenden Schritte (wobei jeder Schritt auf dem vorherigen aufbaut):

1. Füge Datensatz (24556, Turnamian, Finance, 98000) ein.
2. Lösche Datensatz 2.
3. Füge Datensatz (34556, Thompson, Music, 67000) ein.

Def: Alle Tupel haben die gleiche Länge,
auch wenn sie nicht den gesamten Speicherplatz benötigen

header				→ 3
record 0	10101	Sromovasam	Comp. Sci.	65000
record 1	15151	Mozart	Music	40000
record 2	22222	Einstein	Physics	95000
record 3				→ 4
record 4				→ 7
record 5	33456	Gold	Physics	87000
record 6	58583	Califieri	History	62000
record 7				→ ⊥
record 8	87543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

header				→ 4
record 0	10101	Sromovasam	Comp. Sci.	65000
record 1	15151	Mozart	Music	40000
record 2	22222	Einstein	Physics	95000
record 3	24556	Turnamian	Finance	88000
record 4				→ 7
record 5	33456	Gold	Physics	87000
record 6	58583	Califieri	History	62000
record 7				→ ⊥
record 8	87543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

Füge Datensatz
(24556, Turnamian, Finance, 88000)

header				→ 2
record 0	10101	Sromovasam	Comp. Sci.	65000
record 1	15151	Mozart	Music	40000
record 2				→ 4
record 3	24556	Turnamian	Finance	88000
record 4				→ 7
record 5	33456	Gold	Physics	87000
record 6	58583	Califieri	History	62000
record 7				→ ⊥
record 8	87543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

Lösche Datensatz 2

header				→ 4
record 0	10101	Sromovasam	Comp. Sci.	65000
record 1	15151	Mozart	Music	40000
record 2	34556	Thompson	Music	67000
record 3	24556	Turnamian	Finance	88000
record 4				→ 7
record 5	33456	Gold	Physics	87000
record 6	58583	Califieri	History	62000
record 7				→ ⊥
record 8	87543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

Füge Datensatz
(34556, Thompson, Music, 67000)

2 Sequential File Organization

Warum wird im Rahmen der Sequential File Organization ein *Overflow-Block* verwendet, auch wenn es im Moment nur einen einzigen *Overflow-Record* gibt?

3 Indizes

1. Indizes beschleunigen die Abfrageverarbeitung. Dennoch ist es selten eine gute Idee, Indizes auf jedem Attribut und jeder Kombination von Attributen zu erstellen, die möglicherweise für eine Abfrage nützlich sein könnten. Warum ist das so?
2. Kann eine Relation mehr als einen Clustering-Index haben, jeweils basierend auf einem unterschiedlichen Suchschlüssel? Erklären Sie Ihre Antwort.

4 B⁺-Baum

1. Konstruieren Sie einen B⁺-Baum für die folgenden Schlüsselwerte:

$[A, B, C, D, H, I, J, K, L, M]$

Angenommen, der Baum ist zunächst leer und die Werte werden nacheinander in aufsteigender Reihenfolge hinzugefügt. Konstruieren Sie B⁺-Bäume für die Fälle, wo der Verzweigungsgrad n (=Fanout =Anzahl Pointer pro Knoten) wie folgt gegeben ist:

- (a) $n = 3$
 - (b) $n = 5$
 - (c) $n = 7$
2. Zeigen Sie für die B⁺-trees (a) und (c), die Sie konstruiert haben, die Form des Baumes nach jeder der folgenden Operationen (wobei jede Operation auf dem vorherigen Ergebnis ausgeführt wird):
 - (a) Insert F
 - (b) Insert G
 - (c) Insert E
 - (d) Delete K
 - (e) Delete J

5 Nützlicher Index

Betrachten Sie die folgende Abfragen, die auf einer Bibliotheksdatenbank ausgeführt werden. Bewerten Sie, ob die Erstellung eines Index deren Ausführungsgeschwindigkeit verbessern könnte.

2 Sequential File Organization

Warum wird im Rahmen der Sequential File Organization ein *Overflow-Block* verwendet, auch wenn es im Moment nur einen einzigen *Overflow-Record* gibt?

Sequential

nach Search Key geordnet

Search Key

1	Aaen	E111
2	Dolog	E116
3	Larsen	E117
4	Ravn	E161
5	Srba	E166
6	Rose	E167
7	Torp	E171
8	Lazy	E176

Tupel werden sequentiell in Reihenfolge des Search Keys abgelegt

- Example table is stored in order of the ID column
- The ordering column does not need to be the primary key
- Search: binary search when search on the ID column
- Insert: reorganization of the file

man müsste alles verschieben \Leftrightarrow teuer
 \Rightarrow Overflow-Record vermeidet das

10100

Alice

15100

Shane

20200

Tina

15500 Dana
aber keine freie Stelle

Hauptdatei

10100	Alice
15100	Shane
20200	Tina
Pointer: A	

Overflow A

15500 Dana

1. Suche sequentiell in der Hauptdatei
2. Wenn Search Key nicht gefunden
 \Rightarrow Durchsuche Overflow-Block (falls einer existiert)

Vorteile

- schnelles Einfügen

Nachteil

- verlangsamtes Suchen, wenn häufig verwendet

Ergänzung

Man kann eine Reorganisation der Datei durchführen, um den Overflow in die Hauptdatei zu packen.

Man kann Lücken mit Absicht lassen, damit spätere

Einträge dort Platz finden, anstatt gleich im Overflow-Block geschrieben zu werden

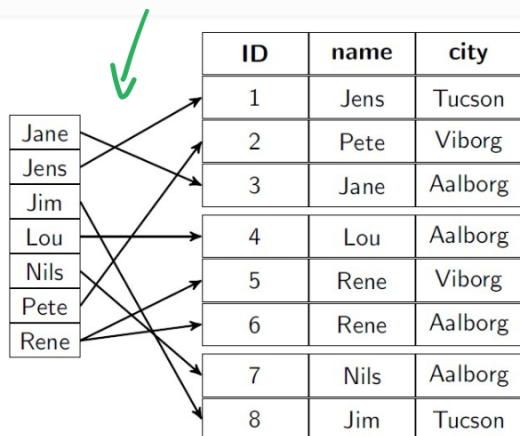
3 Indizes

1. Indizes beschleunigen die Abfrageverarbeitung. Dennoch ist es selten eine gute Idee, Indizes auf jedem Attribut und jeder Kombination von Attributen zu erstellen, die möglicherweise für eine Abfrage nützlich sein könnten. Warum ist das so?
2. Kann eine Relation mehr als einen Clustering-Index haben, jeweils basierend auf einem unterschiedlichen Suchschlüssel? Erklären Sie Ihre Antwort.

1) Indizes dienen v.a. dem Routing von Zugriffen
=> dadurch verkürzt man den Durchsuchungsaufwand

Dense Index: Pointer zu jedem Datensatz
+ sehr schnelle Zugriffe
- hoher Speicheraufwand

besser wäre es eine Balance zu finden
(nur so viele Pointer wie nötig)



Man muss alle Suchschlüsselwerte & Pointer speichern

- Wartungsaufwand: Man muss im WorstCase von Abfragen alle Indizes aktualisieren, Je mehr Indizes, desto mehr Aufwand

2) Clustering: Die Datei ist nach dem Search Key sortiert
(lineare Speicherung auf Festplatte)

Nein, eine Relation kann nur ein Cluster-Index haben, da auf der Festplatte die Position der Daten nur einem Sortierkriterium folgen kann

4 B+-Baum

1. Konstruieren Sie einen B+-Baum für die folgenden Schlüsselwerte:

[A, B, C, D, H, I, J, K, L, M]

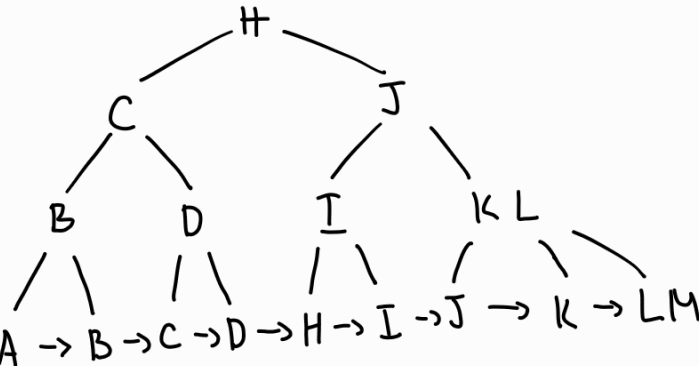
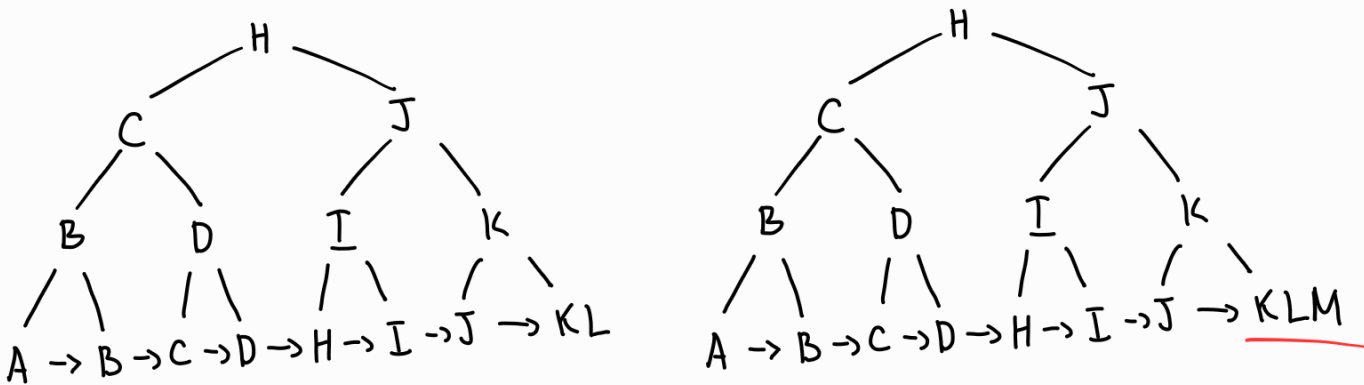
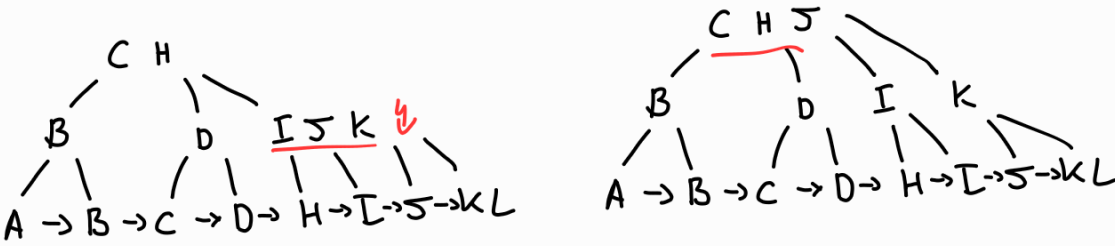
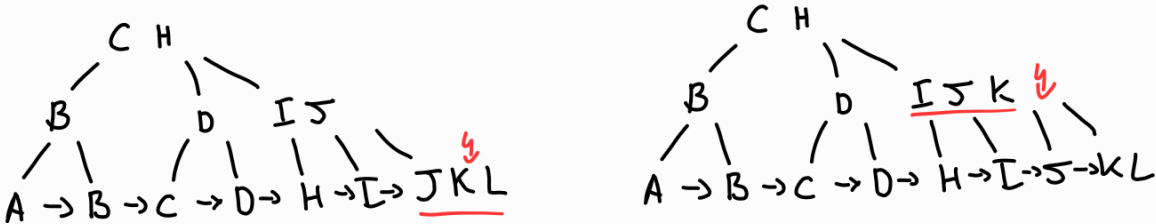
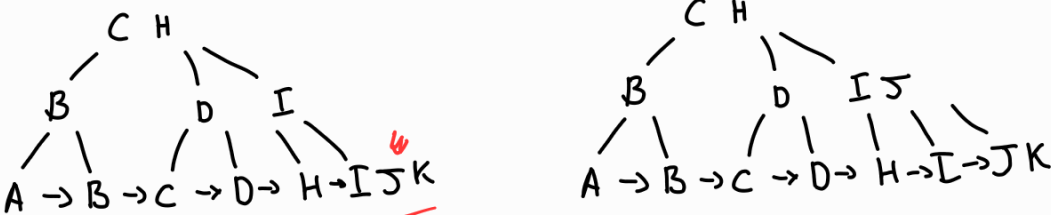
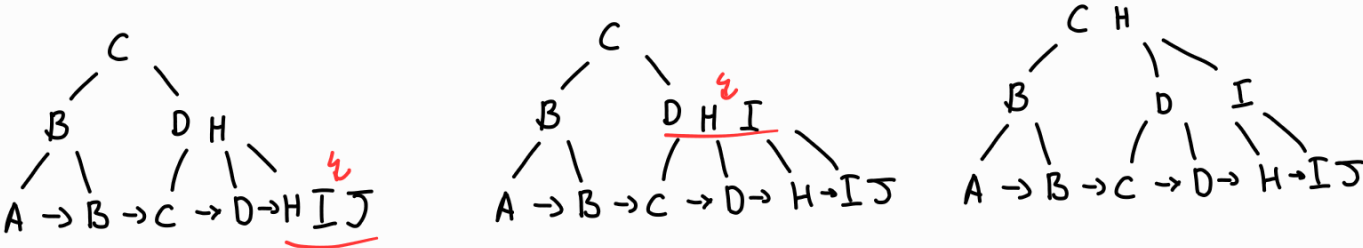
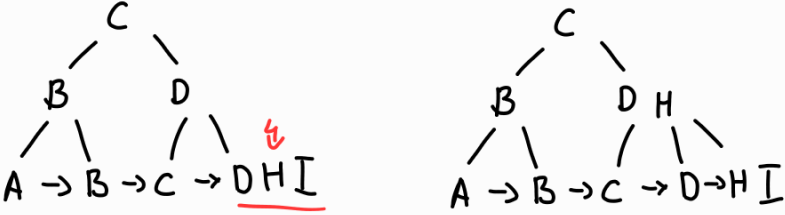
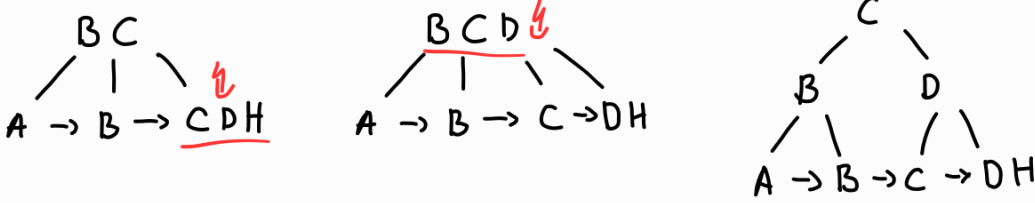
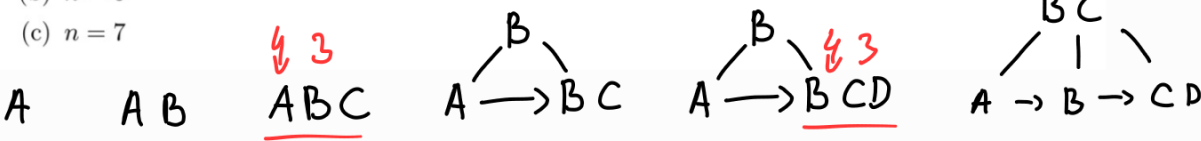
Angenommen, der Baum ist zunächst leer und die Werte werden nacheinander in aufsteigender Reihenfolge hinzugefügt. Konstruieren Sie B+-Bäume für die Fälle, wo der Verzweigungsgrad n (=Fanout =Anzahl Pointer pro Knoten) wie folgt gegeben ist:

(a) $n = 3$

(b) $n = 5$

(c) $n = 7$

a) $n=3$



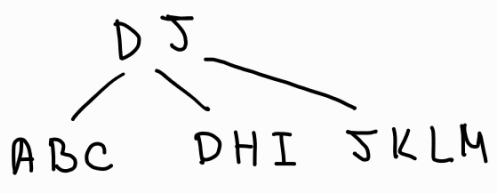
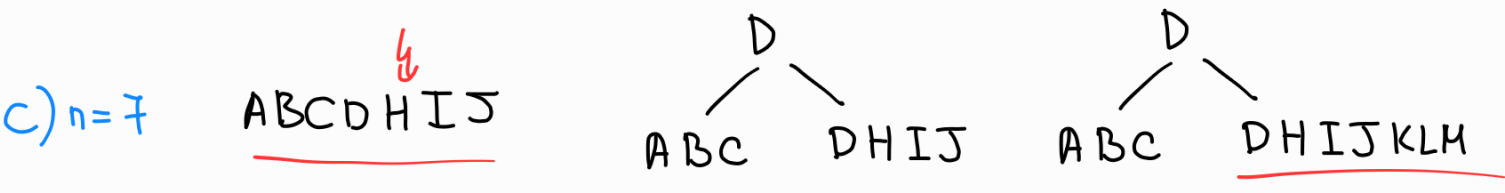
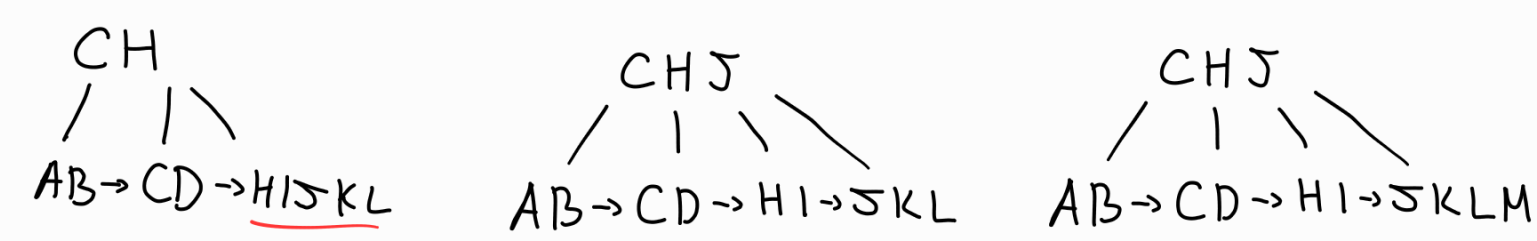
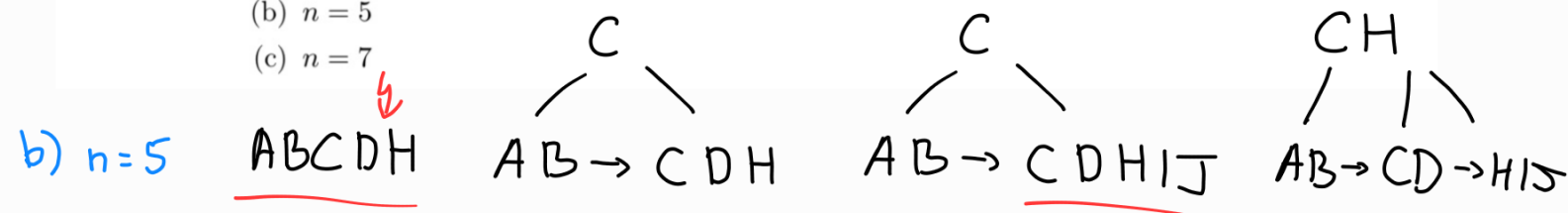
4 B+-Baum

1. Konstruieren Sie einen B+-Baum für die folgenden Schlüsselwerte:

[A, B, C, D, H, I, J, K, L, M]

Angenommen, der Baum ist zunächst leer und die Werte werden nacheinander in aufsteigender Reihenfolge hinzugefügt. Konstruieren Sie B+-Bäume für die Fälle, wo der Verzweigungsgrad n (=Fanout =Anzahl Pointer pro Knoten) wie folgt gegeben ist:

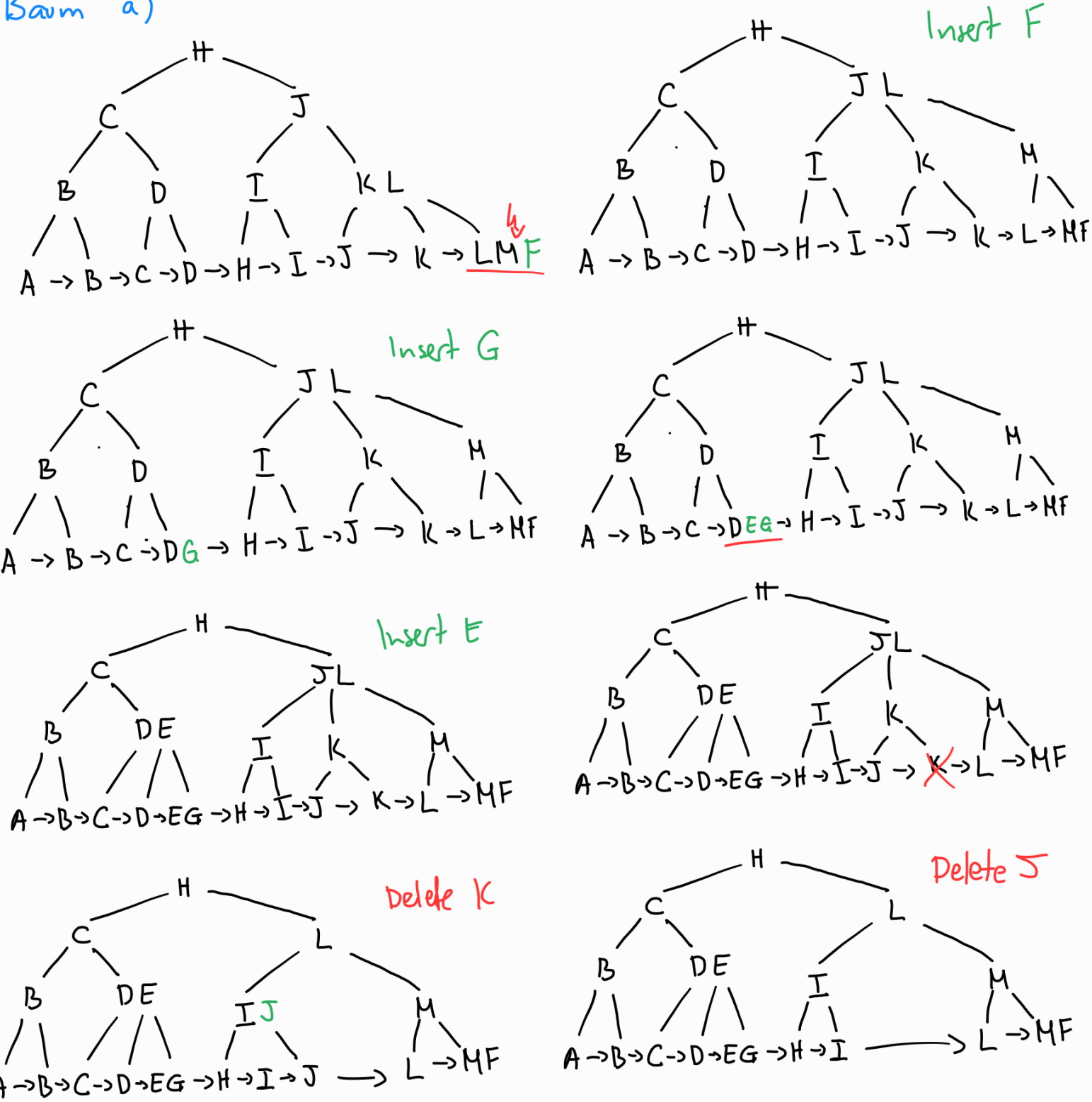
- (a) $n = 3$
- (b) $n = 5$
- (c) $n = 7$



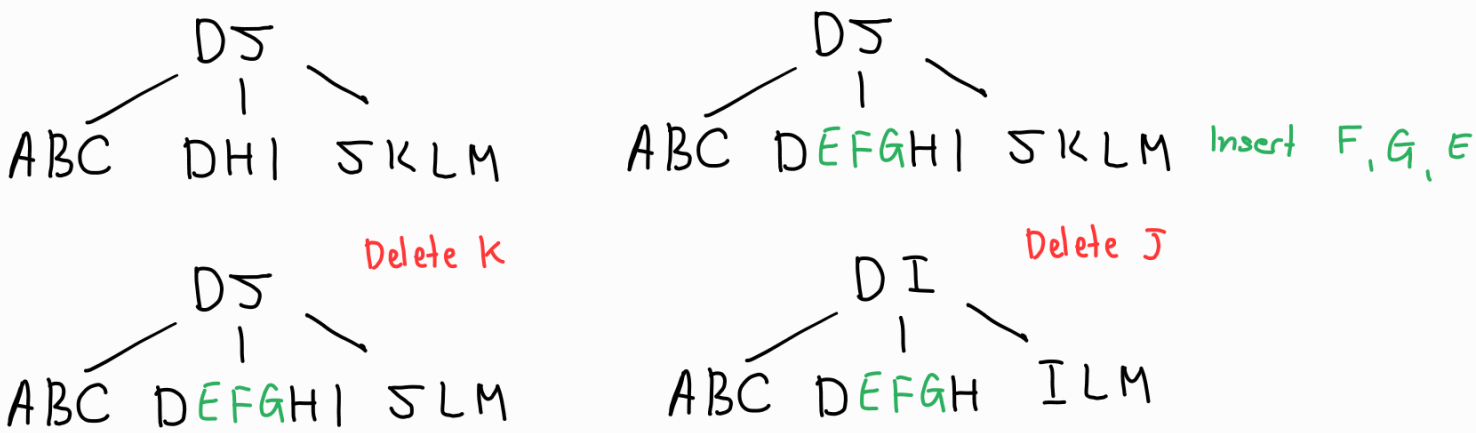
2. Zeigen Sie für die B⁺-trees (a) und (c), die Sie konstruiert haben, die Form des Baumes nach jeder der folgenden Operationen (wobei jede Operation auf dem vorherigen Ergebnis ausgeführt wird):

- (a) Insert F
- (b) Insert G
- (c) Insert E
- (d) Delete K
- (e) Delete J

Baum a)



Baum c) n=7



1. Identifizieren Sie die Attribute, bei denen ein Index den größten Nutzen für diese Abfrage bringen würde:

```
SELECT bk.Title, l.DueDate
FROM Book bk, Loan l
WHERE bk.BookID = l.BookID;
```



2. Betrachten Sie die folgende Abfrage, die bestimmte Buchtitel sowie deren zugehörige Ausleih-Fälligkeitsdaten abrufen. Analysieren Sie, welche Indizes zur Optimierung der Leistung dieser Abfrage am Vorteilhaftesten wären.

Wäre die alleinige Indizierung der Join-Attribute (bk.BookID, l.BookID) die effektivste Strategie, um diese spezielle Abfrage zu beschleunigen? Begründen Sie Ihre Antwort.

```
SELECT bk.Title, l.DueDate
FROM Book bk, Loan l
WHERE bk.BookID = l.BookID
  AND bk.Title = 'The Hitchhiker's Guide to the Galaxy'
  AND l.DueDate = '2025-07-31';
```

2)

Die Indizierung der Join-Attribute beschleunigt nur den Join
und nicht die Filtrierung nach Title und DueDate

Man muss trotzdem alle Titel durchsuchen und überall den DueDate überprüfen

D.h. es lohnt sich bk.Title und l.DueDate zu indexieren

Wenn das Join häufig auftritt, dann auch l.BookID

Aufschreibeweise für Join

Ziel: Join effizient gestalten

1. Identifizieren Sie die Attribute, bei denen ein Index den größten Nutzen für diese Abfrage bringen würde:

```
SELECT bk.Title, l.DueDate  
FROM Book bk, Loan l  
WHERE bk.BookID = l.BookID;
```

bk.BookID Primärschlüssel und somit automatisch indexiert

l.BookID Fremdschlüssel (kein Index automatisch)
=> muss manuell erzeugt werden

Es lohnt sich zu l.BookID ein Index zu erzeugen, wenn die Abfrage häufig passiert bzw. wenn andere Abfragen l.BookID häufig benötigen. **bessere performance**

Wenn nicht, dann ist es nur zusätzlicher Speicherbedarf und Verlangsamung bei INSERT/DELETE/UPDATE