

# HW1 . 1

---

(1)在关系数据模型中，实体、实体间的联系以及语义约束都是如何表达的？

- 实体由元组表示；实体间联系用外码表示；语义约束由三类完整性约束表示

# HW1 . 2

---

- 根据以下关系模式，写出查询的关系表达式

图书(图书号: char, 书名: char, 作者: char, 单价: float, 库存量: float)

读者(读者号: char, 姓名: char, 工作单位: char, 地址: char)

借阅(图书号: char, 读者号: char, 借期: datetime, 还期: datetime, 备注: char)

- (1) 检索工作单位为中国科学技术大学的读者的读者号和姓名；
- (2) 检索 Ullman 所写的书的书名和单价；
- (3) 检索借阅了 Ullman 所写的书 (包括借阅未还和已还) 的读者姓名和借期；
- (4) 检索未借阅图书《数据库基础》的读者姓名；
- (5) 检索读者李林借阅过并且已还的图书的作者；
- (6) 检索借阅图书数目超过 3 本的读者姓名；
- (7) 检索没有借阅读者李林所借的任何一本书的读者姓名和读者号。

# HW1.2

1.  $\pi_{\text{读者号, 姓名}}(\sigma_{\text{工作单位}='中国科学技术大学'}(\text{读者}))$
2.  $\pi_{\text{书名, 单价}}(\sigma_{\text{作者}='Ullman'}(\text{图书}))$
3.  $\pi_{\text{姓名, 借期}}((\sigma_{\text{作者}='Ullman'}(\text{图书}) \bowtie \text{借阅}) \bowtie \text{读者})$
4.  $\pi_{\text{姓名}}(\pi_{\text{读者号, 姓名}}(\text{读者}) - \pi_{\text{读者号, 姓名}}((\sigma_{\text{书名}='数据库基础'}(\text{图书}) \bowtie \text{借阅}) \bowtie \text{读者}))$
5.  $\pi_{\text{作者}}((\sigma_{\text{姓名}='李林'}(\text{读者}) \bowtie (\sigma_{\text{还期} \neq \text{NULL}}(\text{借阅}))) \bowtie \text{图书})$
6.  $\pi_{\text{姓名}}(\sigma_{\text{借书数} > 3}(\gamma_{\text{读者号}, \text{COUNT}(\text{图书号}) \rightarrow \text{借书数}}(\text{借阅}) \bowtie \text{读者}))$
7.  $\pi_{\text{读者号, 姓名}}(\text{读者}) - \pi_{\text{读者号, 姓名}}((\sigma_{\text{姓名}='李林'}(\text{读者}) \bowtie \text{借阅}) \bowtie \text{借阅} \bowtie \text{读者})$

# HW2 . 1

---

- 根据以下关系模式，写出查询的SQL语句

图书(图书号: char, 书名: char, 作者: char, 单价: float, 库存量: float)

读者(读者号: char, 姓名: char, 工作单位: char, 地址: char)

借阅(图书号: char, 读者号: char, 借期: datetime, 还期: datetime, 备注: char)

- (1) 检索工作单位为中国科学技术大学的读者的读者号和姓名；
- (2) 检索 Ullman 所写的书的书名和单价；
- (3) 检索借阅了 Ullman 所写的书(包括借阅未还和已还)的读者姓名和借期；
- (4) 检索未借阅图书《数据库基础》的读者姓名；
- (5) 检索读者李林借阅过并且已还的图书的作者；
- (6) 检索借阅图书数目超过 3 本的读者姓名；
- (7) 检索没有借阅读者李林所借的任何一本书的读者姓名和读者号。

# HW2 . 2

---

1. Select 读者号,姓名 from 读者 where 工作单位='中国科学技术大学';
2. Select 书名,单价 from 图书 where 作者='Ullman';
3. Select 读者.姓名,借阅.借期 from 图书,读者,借阅 where 图书.图书号=借阅.图书号 and 读者.读者号=借阅.读者号 and 图书.作者='Ullman';
4. Select 姓名 from 读者 where 读者号 not in (Select 读者.读者号 from 图书,读者,借阅 where 图书.图书号=借阅.图书号 and 读者.读者号=借阅.读者号 and 图书.作者='Ullman');

# HW2 . 2

---

5. `Select 图书.作者 from 图书,读者,借阅 where 图书.图书号=借阅.图书号  
and 读者.读者号=借阅.读者号 and 读者.姓名='李林' and 还期 is not  
NULL;`
6. `Select 读者.姓名 from 读者,借阅 where 读者.读者号=借阅.读者号  
Group By 读者.读者号 Having COUNT(*) > 3;`
7. `Select 读者.读者号,读者.姓名 from 读者,借阅 where 读者.读者号=借阅.  
读者号 and 借阅.图书号 not in (Select 借阅.图书号 from 读者,借阅  
where 读者.读者号=借阅.读者号 and 读者.姓名='李林');`

# Problem

---

- 多重关系代数表达式尽量加括号凸显优先级，若不愿这样做请参考优先级
- 投影/选择的谓词一定是属性而非关系
- 注意主键的唯一性，非主键属性可能会出现重复导致结果有误（例如读者姓名）
- Exists前无属性
- 定义新的关系时需要说明
- 除法操作的除关系属性应为被除关系的子集

# 3.1.1

- 1. 已知有关系模式  $R(A, B, C, D, E)$ ,  $R$  上的一个函数依赖集如下:  $F=\{A\rightarrow BC, B\rightarrow CD, A\rightarrow E, AB\rightarrow C, AC\rightarrow DE, BE\rightarrow AD\}$

## (1) 求出 $F$ 的最小函数依赖集

- 将右边写成单属性并去除重复FD
- $F=\{A\rightarrow B, A\rightarrow C, B\rightarrow C, B\rightarrow D, A\rightarrow E, AB\rightarrow C, AC\rightarrow D, AC\rightarrow E, BE\rightarrow A, BE\rightarrow D\}$
- 消去左部冗余属性
- $A\rightarrow C$ , 可推出  $AB\rightarrow BC$ , 从而推出  $AB\rightarrow C$ , 所以  $AB\rightarrow C$  中的  $B$  是冗余属性
- $A\rightarrow C, AC\rightarrow D, AC\rightarrow E$  可推出  $A\rightarrow AC, A\rightarrow D, A\rightarrow E$  因此可去除  $AC\rightarrow D$  中的  $C$  和  $AC\rightarrow E$  中的  $C$
- $B\rightarrow D$ , 可推出  $BE\rightarrow DE$ , 从而推出  $BE\rightarrow D$ , 所以  $BE\rightarrow D$  中的  $E$  是冗余属性
- $F=\{A\rightarrow B, A\rightarrow C, B\rightarrow C, B\rightarrow D, A\rightarrow E, A\rightarrow D, BE\rightarrow A\}$
- 消去冗余函数依赖
- $A\rightarrow C, A\rightarrow D$  冗余
- $F=\{A\rightarrow B, B\rightarrow C, B\rightarrow D, A\rightarrow E, BE\rightarrow A\}$



## 3.1.2

---

### (2)求 R 的候选码, 并给出证明

- $X \rightarrow U \in F^+$ , 则X是R的一个超码, 如果同时不存在X的真子集Y, 使得 $Y \rightarrow U$ 成立, 则X是R的一个候选码。
- 因为 $A \rightarrow A$ ,  $A \rightarrow B$ ,  $A \rightarrow C$  ( $A \rightarrow B, B \rightarrow C$ ),  $A \rightarrow D$  ( $A \rightarrow B, B \rightarrow D$ ),  $A \rightarrow E$ , 并且不存在A的真子集Y, 使得 $Y \rightarrow U$ 成立, 所以A为候选码。
- 因为 $B \rightarrow B$ ,  $B \rightarrow C$ ,  $B \rightarrow D$ ,  $E \rightarrow E$ ,  $BE \rightarrow A$ , 从而 $BE \rightarrow B$ ,  $BE \rightarrow C$ ,  $BE \rightarrow D$ ,  $BE \rightarrow E$ ,  $BE \rightarrow A$ , 并且不存在BE的真子集Y, 使得 $Y \rightarrow U$ 成立, 所以BE也为候选码。
- 因此候选码为{A}, {B, E}

# 3.1.3

---

(3)判断 R 属于第几范式？为什么？

- 最小函数依赖集为 $F=\{A \rightarrow B, B \rightarrow C, B \rightarrow D, A \rightarrow E, BE \rightarrow A\}$
- 候选码为 $\{A\}, \{B, E\}$ ，主属性为 $A, B, E$ ，非主属性为 $C, D$
- 存非主属性局部函数依赖于主码，例如 $B \rightarrow C$ (多个候选码时，要求非主属性完全函数依赖于所有的候选码)
- 所以是1NF.

## 3.2.1

- 2. 已知有关系模式  $R(A, B, C, D, E, F, G)$ ,  $R$  上的一个函数依赖集:  
 $F = \{A \rightarrow B, B \rightarrow C, AC \rightarrow DE, E \rightarrow F, AB \rightarrow E, AC \rightarrow G\}$

(1) 求出  $F$  的最小函数依赖集

- 将右边写成单属性并去除重复FD
- $F = \{A \rightarrow B, B \rightarrow C, AC \rightarrow D, AC \rightarrow E, E \rightarrow F, AB \rightarrow E, AC \rightarrow G\}$
- 消去左部冗余属性
- $A \rightarrow B, B \rightarrow C$  可推出  $A \rightarrow C$ , 从而  $A \rightarrow AC$
- 又由  $AC \rightarrow D, AC \rightarrow E, AC \rightarrow G$  可推出  $A \rightarrow D, A \rightarrow E, A \rightarrow G$  因此可去除  $AC \rightarrow D, AC \rightarrow E, AC \rightarrow G$  中的  $C$
- $A \rightarrow B$ , 可推出  $A \rightarrow AB$ , 又由于  $AB \rightarrow E$ , 可推出  $A \rightarrow E$ , 因此可去除  $AB \rightarrow E$  中的  $B$
- 无冗余函数依赖, 故  $F = \{A \rightarrow B, B \rightarrow C, A \rightarrow D, A \rightarrow E, E \rightarrow F, A \rightarrow G\}$

## 3.2.2

---

(2)求 R 的候选码, 并给出证明

- $X \rightarrow U \in F^+$ , 则X是R的一个超码, 如果同时不存在X的真子集Y, 使得 $Y \rightarrow U$ 成立, 则X是R的一个候选码。
- 因为 $A \rightarrow A$ ,  $A \rightarrow B$ ,  $A \rightarrow C$  ( $A \rightarrow B, B \rightarrow C$ ),  $A \rightarrow D$ ,  $A \rightarrow E$ ,  $A \rightarrow F$  ( $A \rightarrow E, E \rightarrow G$ ),  $A \rightarrow G$ , 并且不存在A的真子集Y, 使得 $Y \rightarrow U$ 成立, 所以A为候选码。
- 所以A为候选码。

## 3.2.3

---

(3)判断 R 属于第几范式？为什么？

- 最小函数依赖集为 $F=\{A \rightarrow B, B \rightarrow C, A \rightarrow D, A \rightarrow E, E \rightarrow F, A \rightarrow G\}$
- 主属性为A，非主属性为B,C,D,E,F,G
- 不存在非主属性局部函数依赖于主码
- 存在非主属性传递依赖于主码，例如 $A \rightarrow B, B \rightarrow C$
- 所以是2NF.

## 3.2.4

---

(4) 请将关系模式  $R$  无损连接并且保持函数依赖地分解到 3NF，要求给出具体步骤。

- 最小FD集:  $F = \{A \rightarrow B, B \rightarrow C, A \rightarrow D, A \rightarrow E, E \rightarrow F, A \rightarrow G\}$
- 把所有不在  $F$  中出现的属性组成一个关系模式  $R' = \emptyset$
- 对  $F$  按相同左部分组，得到  $q = \{R1(A, B, D, E, G), R2(B, C), R3(E, F)\}$
- 主码为  $\{A\}$ ，但为  $R1$  的子集
- 因此  $p = q = \{R1(A, B, D, E, G), R2(B, C), R3(E, F)\}$

## 3.3.1

---

- 3. 现有关系模式：  $R(A, B, C, D, E, F, G)$ ，  $R$  上的一个函数依赖集：  
 $F = \{AB \rightarrow E, A \rightarrow B, B \rightarrow C, C \rightarrow D\}$

(1) 该关系模式满足第几范式？为什么？

- $A \rightarrow B$  可得  $A \rightarrow AB$ ，又由于  $AB \rightarrow E$  可得到  $A \rightarrow E$ ，可消去  $AB \rightarrow E$  中的  $B$
- 最小函数依赖集为  $F = \{A \rightarrow B, B \rightarrow C, C \rightarrow D, A \rightarrow E\}$
- 候选码为  $\{A, F, G\}$ ，存非主属性局部依赖于主码，例如  $A \rightarrow B$
- 因此为 1NF.

## 3.3.2

---

(2) 请将关系模式  $R$  无损连接地分解到 BCNF，要求给出步骤

- $A \rightarrow E$  是独立的，先后消除不影响，先消除它，得到  $R_1(A, E), R_2(A, B, C, D, F, G)$
- $A \rightarrow B, B \rightarrow C, C \rightarrow D$  是连续的传递依赖，消除的先后顺序会影响最终结果。
  - 先消除  $R_2$  中的  $A \rightarrow B$ :  $R_x(A, C, D, F, G)$  依赖变成  $A \rightarrow C, C \rightarrow D$
  - 再消除  $A \rightarrow C$ :  $R_1(A, E), R_2(A, B), R_3(A, C), R_4(A, D), R_5(A, F, G)$
  - 再消除  $C \rightarrow D$ :  $R_1(A, E), R_2(A, B), R_3(C, D), R_4(A, C), R_5(A, F, G)$



## 3.3.2

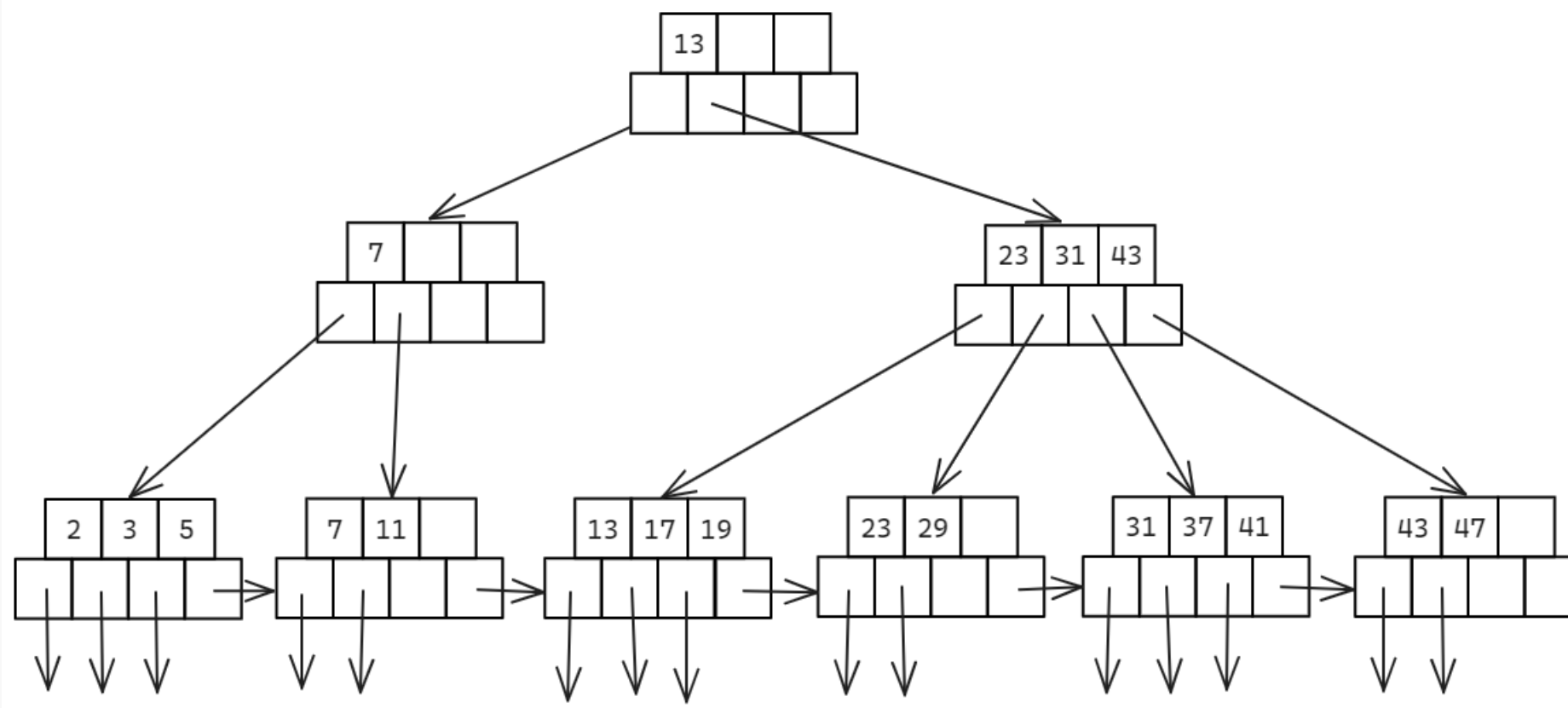
---

(2) 请将关系模式 R 无损连接地分解到 BCNF，要求给出步骤

- 先消除R2中的 $B \rightarrow C$ :  $R_x(A, B, D, F, G)$  依赖变成 $A \rightarrow B, B \rightarrow D$ .
- 再消除 $A \rightarrow B$ :  $R_1(A, E), R_2(B, C), R_3(A, B), R_4(A, D), R_5(A, F, G)$
- 再消除 $B \rightarrow D$ :  $R_1(A, E), R_2(B, C), R_3(B, D), R_4(A, B), R_5(A, F, G)$
  
- 先消除R2中的 $C \rightarrow D$ :  $R_x(A, B, C, F, G)$  依赖变成 $A \rightarrow B, B \rightarrow C$
- 再消除 $A \rightarrow B$ :  $R_1(A, E), R_2(C, D), R_3(A, B), R_4(A, C), R_5(A, F, G)$
- 再消除 $B \rightarrow C$ :  $R_1(A, E), R_2(C, D), R_3(B, C), R_4(A, B), R_5(A, F, G)$
- BCNF分解总共有6种不同的答案（答出一种即可）

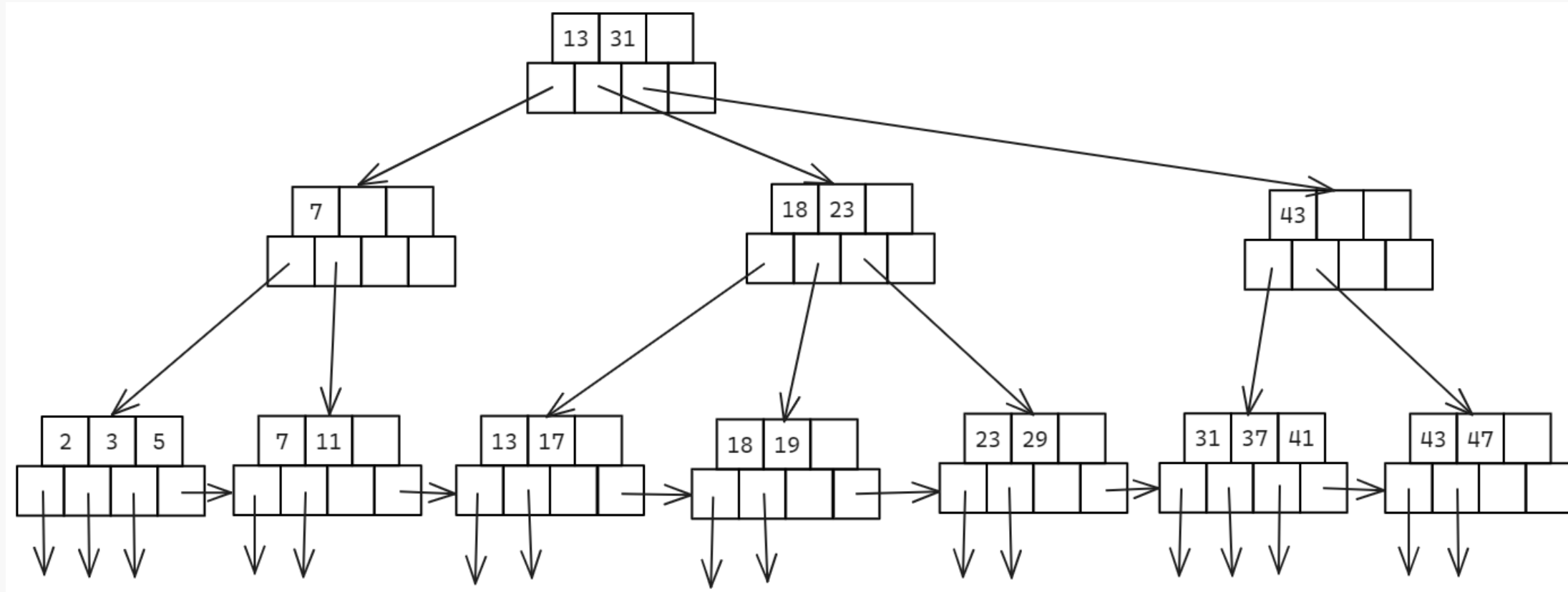
# 4.1

1. 考虑下面的 B+树（假设每个节点最多只能容纳 3 个键值），请画出依次插入了 18, 32, 30, 38 之后的 B+树。



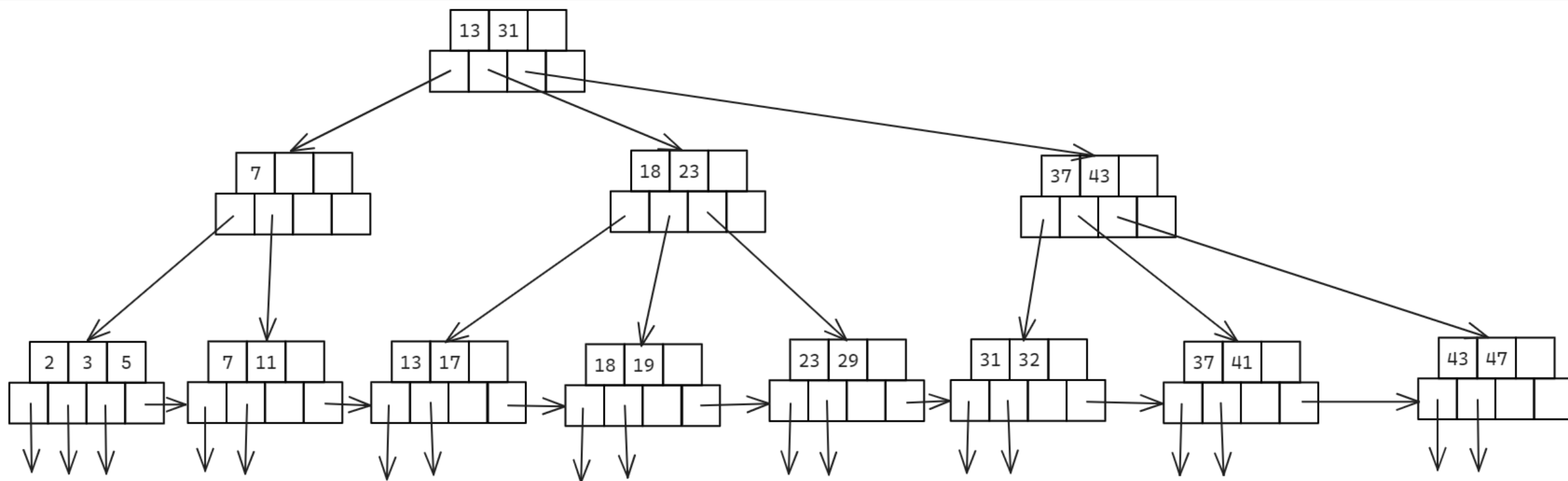
# 4.1

插入18后:



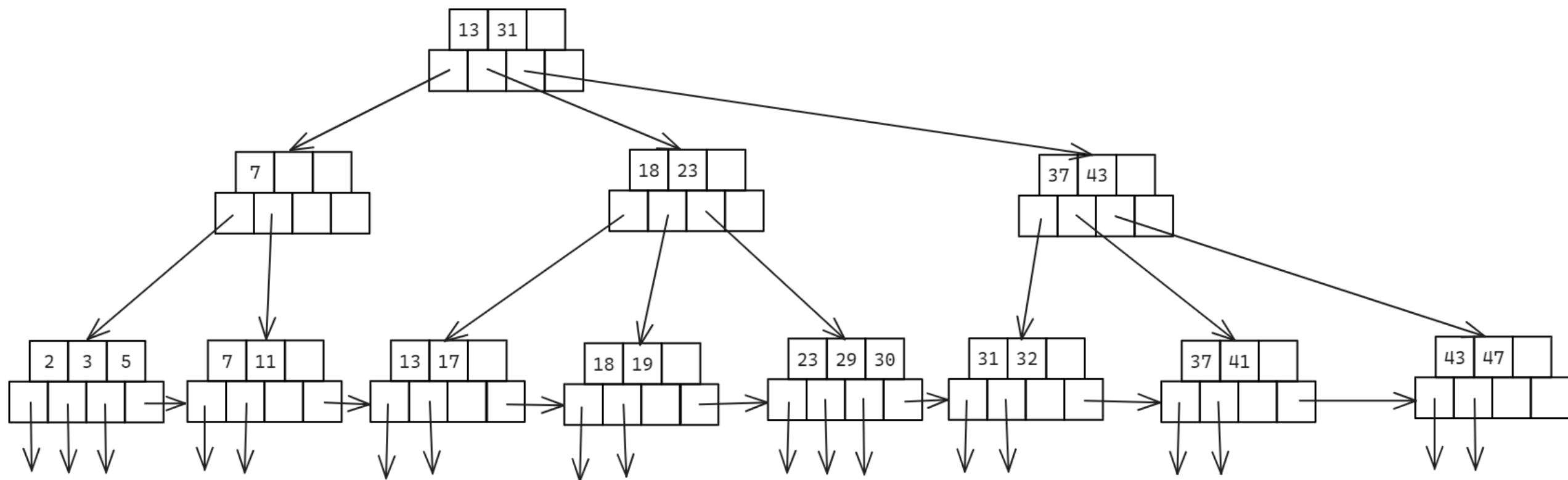
# 4.1

插入32后:



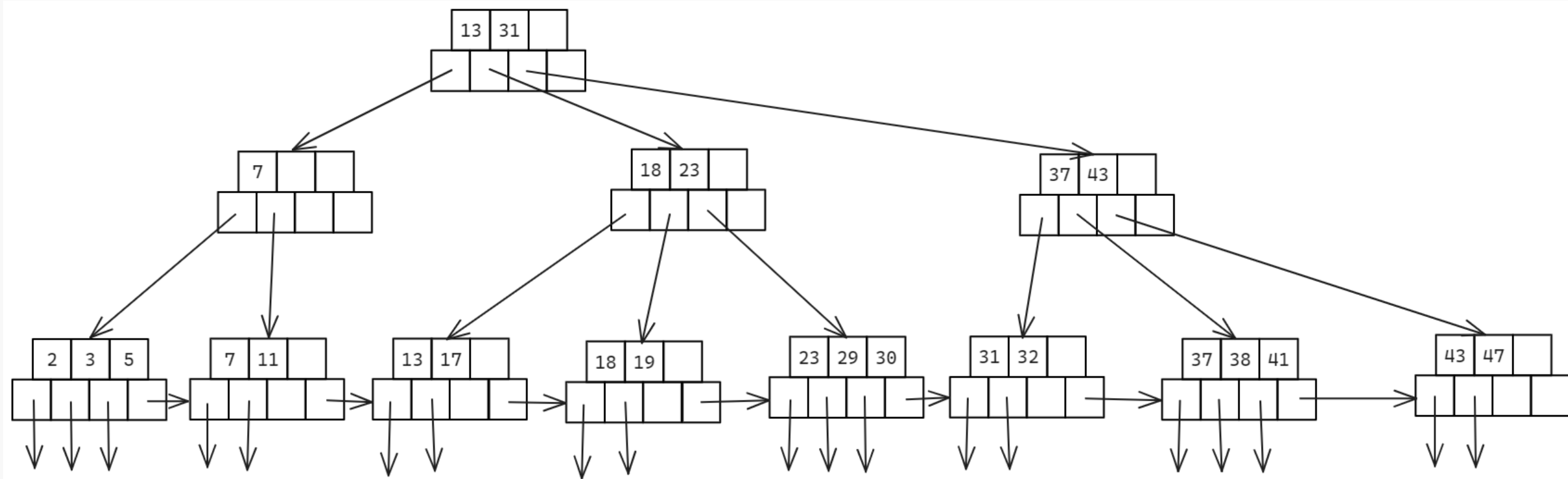
# 4.1

插入30后:



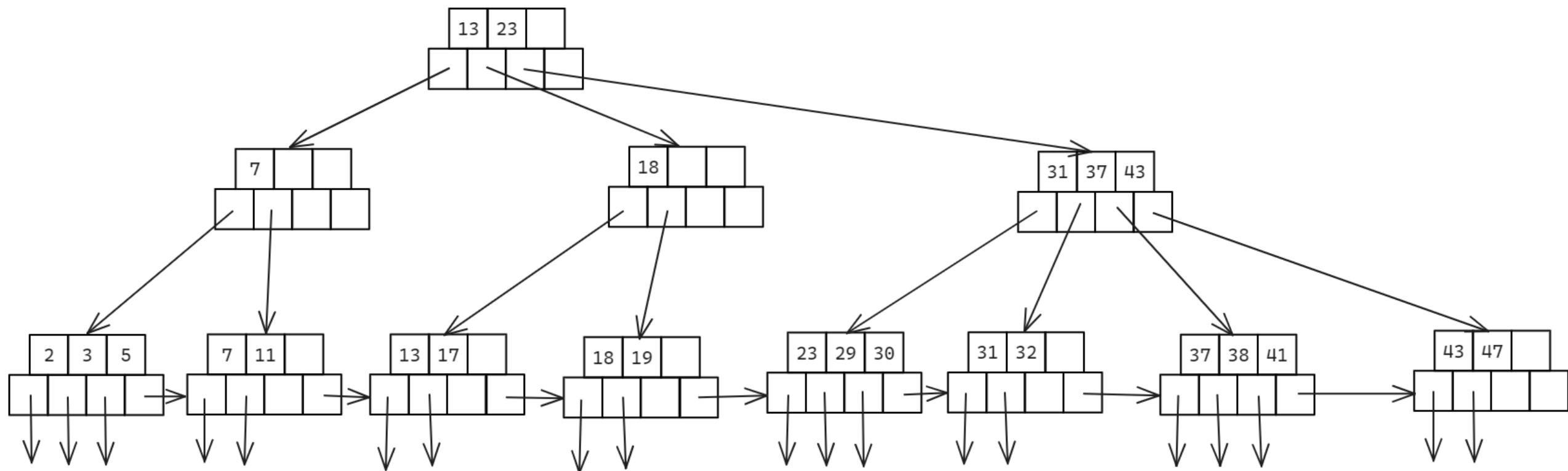
# 4.1

插入38后:



# 4.1

插入38后：（答案2）



## 4.2.1

---

2. 假设我们在数据库中设计了如下基本表来存储文献: `paper(id: int, title: varchar(200), abstract: varchar(1000))`。最常见的文献查询可以描述为“查询 `title` 中同时包含给定关键词的文献”，关键词可以是一个，也可以是多个。请回答下面问题（假设所有文献都是英文文献）：

(1) 假如在 `title` 上创建了 `B+-tree` 索引，能不能提高此查询的效率（须解释理由）？

- 不能。文献查询并不是精确点查询，而是范围查询，查询的单词可能出现在`title`的任意位置。由于`B+`树是根据键的字典序排序的，仅支持前缀查询，不支持中缀和后缀范围查询。因此使用`B+`树结构仍要扫描所有数据项。



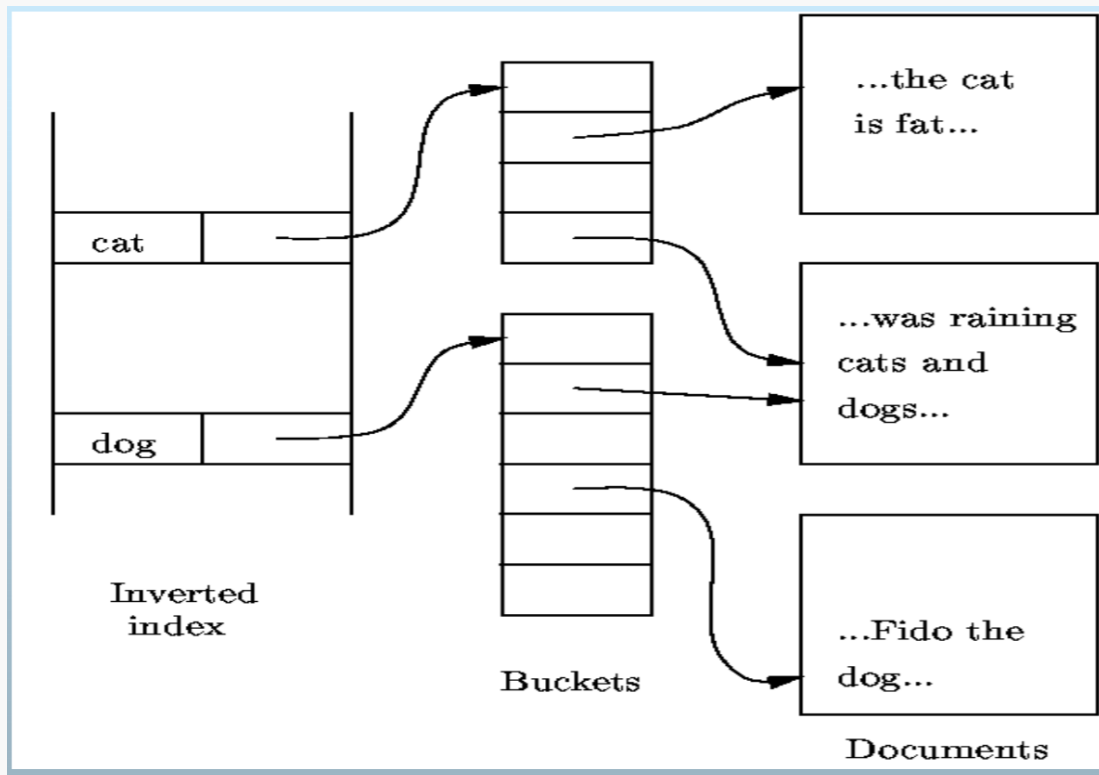
## 4.2.2

---

(2) 由于文献 **title** 的关键词中存在很多重复词语，因此上述文献查询可以借鉴我们课上讲述的支持重复键值的辅助索引技术来进一步优化。请基于此思想画出一种优化的索引结构，简要说明该索引上的记录插入过程以及文献查询过程。

- 使用倒排索引。倒排索引根据单词索引到**title**中包含该单词的所有文档。相比于B+树结构，采用倒排索引优点如下：(a). 避免重复单词的存储 (b). 对于文献查询不需要扫描所有数据项，仅通过少数几次索引查询即可完成搜索。

## 4.2.2



- 插入:

- 对文档的**title**进行分词，然后将各个关键词插入到索引中
- 查询关键词是否已经存在，若存在则直接将指针插入对应的桶中
- 若不存在，则插入关键词，增加对应的桶以及指向文档的指针

- 查询:

- 单个关键词；返回得到的文档集合即可
- 多个关键词；返回各个关键词查询结果的文档集合的交集

## 4.3.1

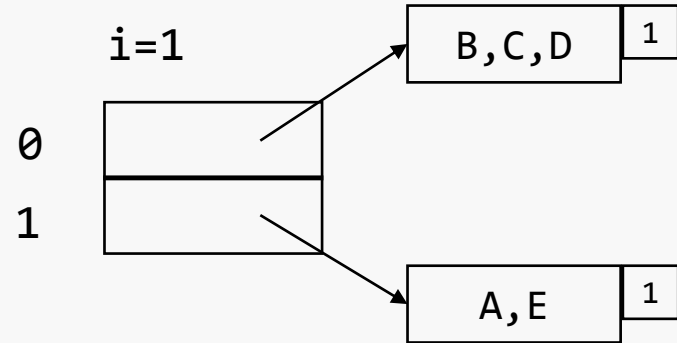
3. 假设有如下的键值，现用 5 位二进制序列来表示每个键值的 hash 值。回答问题：

A [11001]	B [00111]	C [00101]	D [00110]	E [10100]	F [01000]	G [00011]
H [11110]	I [10001]	J [01101]	K [10101]	L [11100]	M [01100]	N [11111]

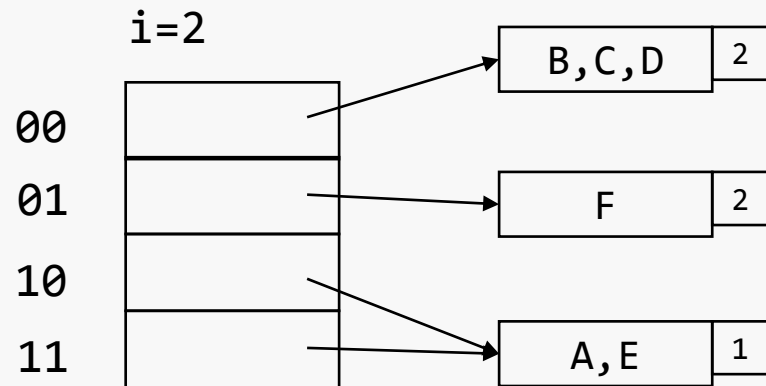
(1) 如果将上述键值按 A 到 N 的顺序插入到可扩展散列索引中，若每个桶大小为一个磁盘块，每个磁盘块最多可容纳 3 个键值，且初始时散列索引为空，则全部键值插入完成后该散列索引中共有几个桶？并请写出键值 E 所在的桶中的全部键值。

# 4.3.1

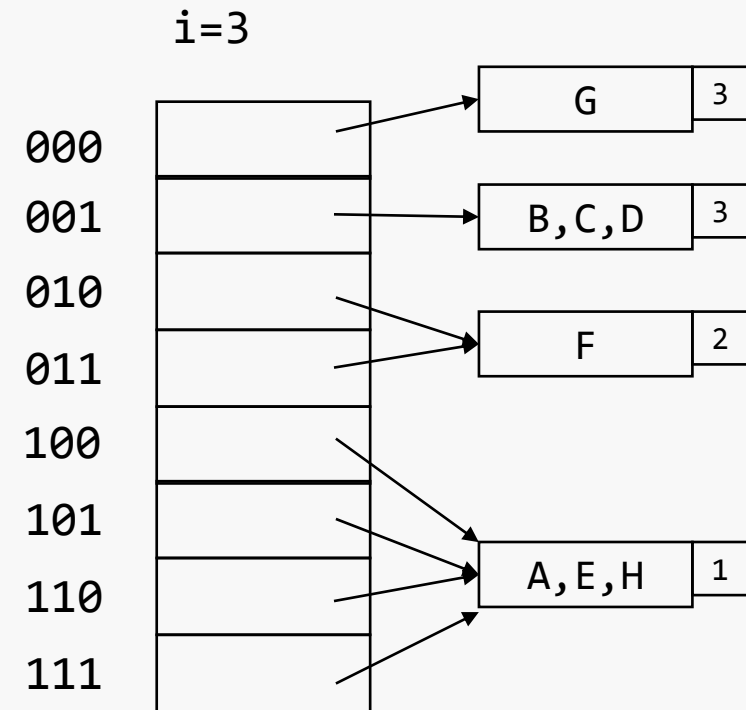
插入ABCDE



插入F

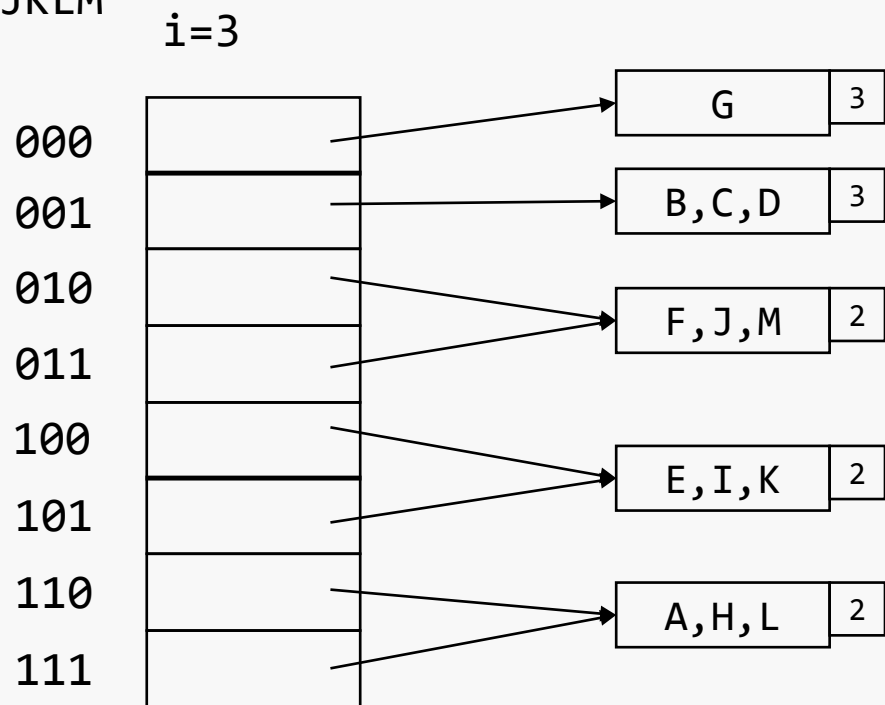


插入GH

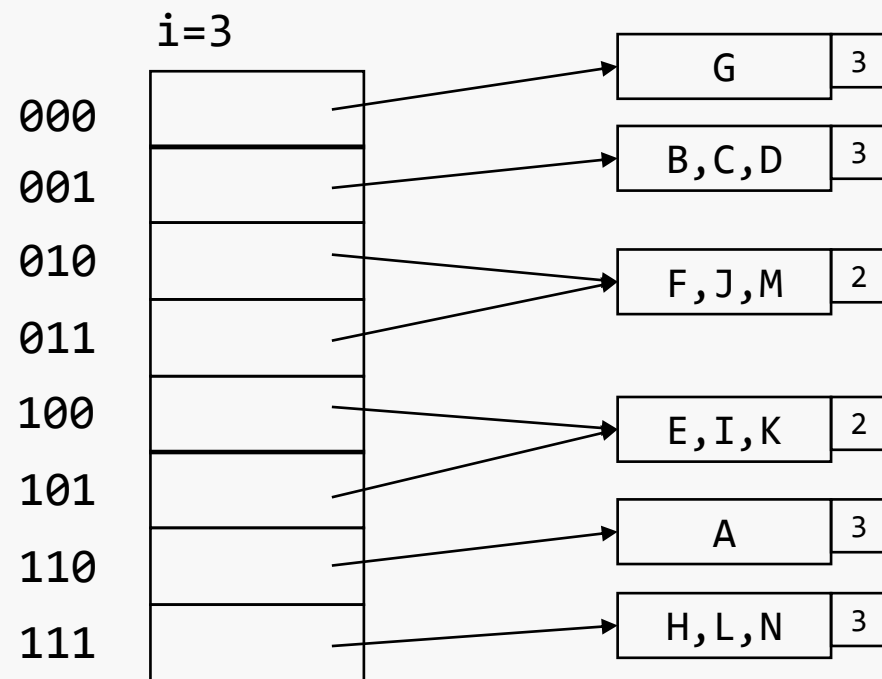


# 4.3.1

插入IJKLM



插入N



6个桶, E、I、K

## 4.3.2

- (2) 前一问题中, 如果换成线性散列索引, 其余假设不变, 同时假设只有当插入新键值后 空间利用率大于 80%时才增加新的桶, 则全部键值按序插入完成后该散列索引中共有几个桶? 并请写出键值 B 所在的桶中的全部键值 (包括溢出块中的键值)。

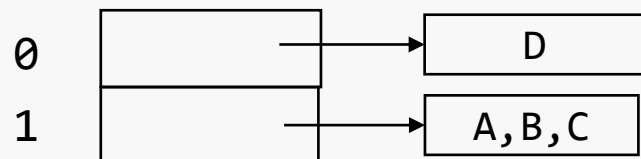
**$h(k)$ 仍是二进制位序列, 但使用右边(低) $i$ 位区分桶**

- 桶数 $=n$ ,  $h(k)$ 的右 $i$ 位 $=m$
- 若 $m < n$ , 则记录位于第 $m$ 个桶
- 若 $n \leq m < 2^i$ , 则记录位于第  $m - 2^{i-1}$  个桶
- $n$ 的选择: 总是使 $n$ 与当前记录总数 $r$ 保持某个固定比例

## 4.3.2

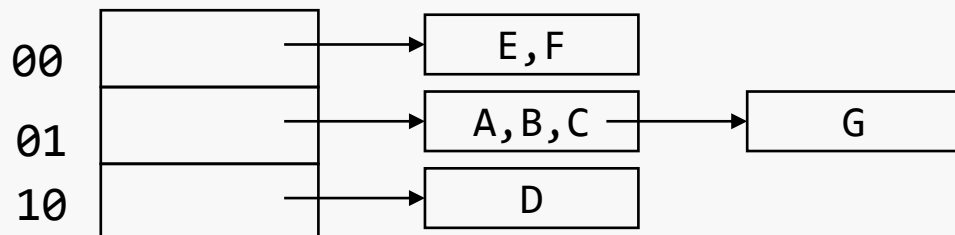
插入ABCD

$i=1$  最多  $6 \times 0.8 = 4.8$



插入EFG

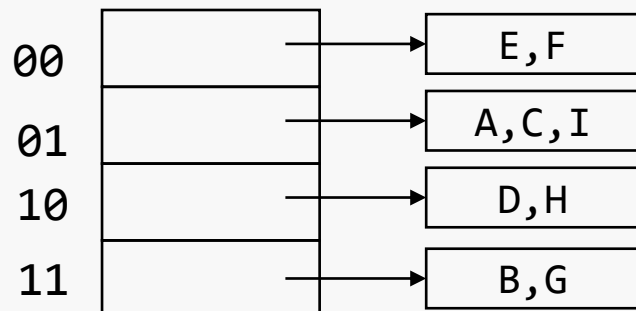
$i=2$  最多  $9 \times 0.8 = 7.2$



插入HI

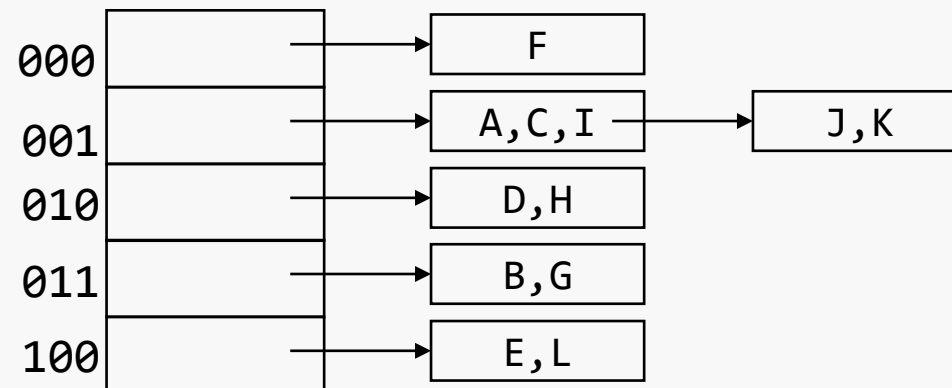
$i=2$

最多  $12 \times 0.8 = 9.6$



插入JKL

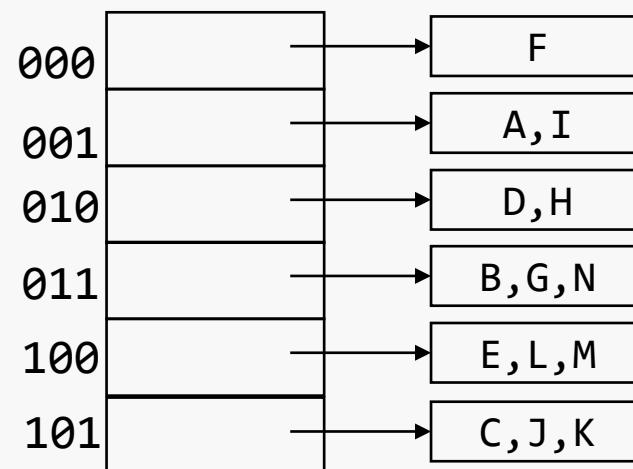
$i=3$  最多  $15 \times 0.8 = 12$



插入MN

$i=3$

最多  $18 \times 0.8 = 14.4$



6个桶, B、G、N

# HW5 . 1

---

1. 什么是事务的 ACID 性质？请给出一个违背性质 I 的一个例子。

事务的ACID性质是：

原子性(**Atomicity**):事务是不可分的原子，其中的操作要么都做，要么都不做。

一致性(**Consistency**):事务的执行保证数据库从一个一致的状态转移到另一个一致的状态。

隔离性(**Isolation**):多个事务一起执行时相互独立。

持久性(**Durability**):事务一旦成功提交，就在数据库中永久保存。



# HW5 . 1

- 假设A账户有200元，B账户有0元。A账户向B账户转账2次，金额均为50元，分别在两个事务中执行，如果不保证事务的隔离性，A虽然扣款2次但B只增加1次，B账户凭空丢失50元，违反事务的一致性。

t	T1	T2
1	Read(A,t), A=200,t=200	
2	$t \leftarrow t-50$ , A=200,t=150	
3	Write(A,t), A=200,t=150	
4	Output(A), A=150	
5	Read(B,t), B=0,t=0	
6	$t \leftarrow t+50$ , B=0,t=150	
7	Write(B,t), B=0,t=50	

t	T1	T2
8		Read(A,t), A=150,t=150
9		$t \leftarrow t-50$ , A=150,t=100
10		Write(A,t), A=150,t=100
11		Output(A), A=100
12		Read(B,t), B=0,t=0
13		$t \leftarrow t+50$ , B=0,t=150
14		Write(B,t), B=0,t=50
15	Output(B), B=50	
16		Output(B), B=50

# HW5 . 2

2. 目前许多 DBMS 例如 MySQL 都默认不支持嵌套事务（即在一个事务内部又开始另一个事务），请分析一下：如果 DBMS 支持嵌套事务，将面临哪些问题（给出自己的分析）？

（假设事务A嵌套调用事务B）

## 1) 原子性和持久性矛盾

如果B `commit`，但之后A `rollback`，此时A的原子性与B的持久性冲突。

## 2) 一致性影响

B开始的时候可能不处于一致性。

## 3) 隔离性

A事务需要先修改一条数据v再调用B，B事务也需要修改数据v。如果系统采用了锁机制，则会陷入死锁。否则如果B读的是事务A修改后的数据，则发生了脏读；如果读的是修改前的数据，则发生了脏写，两种情况均破坏了隔离性。

## 4) 死循环

AB相互调用，可能陷入死循环。

# HW5 . 3

3. 下面是一个数据库系统开始运行后的日志记录，该数据库系统支持简单检查点。

- 1) < T1, Begin Transaction>
- 2) <T1, A, 49, 20>
- 3) < T2, Begin Transaction>
- 4) <T1, B, 250, 20>
- 5) <T1, A, 75, 49>
- 6) <T2, C, 35, 20>
- 7) <T2, D, 45, 20>
- 8) <T1, Commit Transaction>
- 9) <T3, Begin Transaction>
- 10) <T3, E, 55, 20>
- 11) <T2, D, 46, 45> ----- ①
- 12) <T2, C, 65, 35>
- 13) <T2, Commit Transaction>
- 14) <T3, Commit Transaction> ----- ②
- 15) <CHECKPOINT>
- 16) < T4, Begin Transaction>
- 17) <T4, F, 100, 20>
- 18) <T4, G, 111, 20>
- 19) <T4, F, 150, 100>
- 20) <T4, Commit Transaction> ----- ③

设日志修改记录的格式为 <Tid, Variable, New value, Old value>，请给出对于题中所示①、②、③三种故障情形下，数据库系统恢复的过程以及数据元素 A, B, C, D, E, F 和 G 在执行了恢复过程后的值。

# HW5 . 3

---

① T1: redo

- T2、T3: undo
- 先undo:  $D=45, E=20, D=20, C=20$
- 再redo:  $A=49, B=250, A=75$
- 所以,  $A=75, B=250, C=20, D=20, E=20, F=20, G=20$

② T1、T2、T3: redo

- redo:  $A=49, B=250, A=75, C=35, D=45, E=55, D=46, C=65$
- 所以,  $A=75, B=250, C=65, D=46, E=55, F=20, G=20$

③ 检查点前:  $A=75, B=250, C=65, D=46, E=55, F=20, G=20$

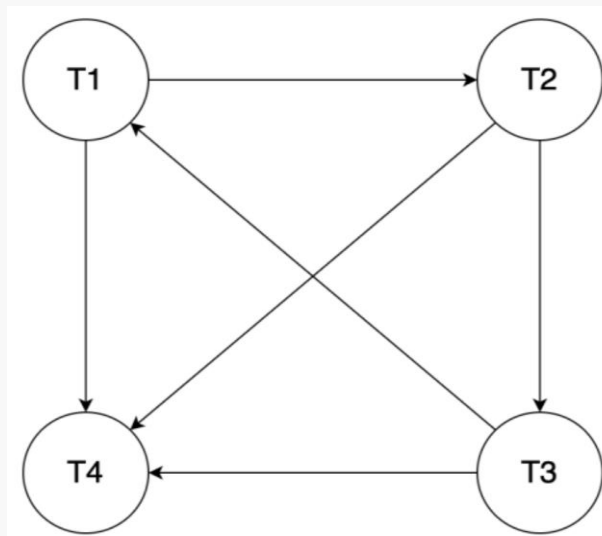
- 检查点后: T4: undo
- undo:  $F=100, G=20, F=20$
- 所以,  $A=75, B=250, C=65, D=46, E=55, F=20, G=20$

# HW6.1

1. 判断下面的并发调度是否冲突可串？如果是，请给出冲突等价的串行调度事务顺序；如果不是，请解释理由。

w3(D); r1(A); w2(A); r4(A); r1(C); w2(B); r3(B); r3(A); w1(D); w3(B); r4(B); r4(C); w4(C); w4(B)

• 根据并发调度做出其对应的优先图：



可以看到该优先图中存在环路，因此该并发调度不是冲突可串的

# HW6.2

2. 证明：如果一个并发调度 S 中的所有事务都遵循 2PL，则该调度必定是冲突可串调度。

- 反证法。如果发生了不可串调度的情况，则该调度不满足冲突可串行性，那么它的优先图中必存在环。设环上的事务为  $T_1, T_2, \dots, T_n$ ，操作的数据为  $D_1, D_2, \dots, D_n$ ，则有：
- (先发生  $\rightarrow$  后发生)
- $uL_1(D_1) \rightarrow L_2(D_1)$  (T1 解锁 D1 后，T2 才可使用 D1)
- $L_2(D_1) \rightarrow uL_2(D_2)$  (先加锁，再解锁)
- $uL_2(D_2) \rightarrow L_3(D_2)$
- ...
- $uL_n(D_n) \rightarrow L_1(D_n)$
- $L_1(D_n) \rightarrow uL_1(D_1)$
- 事务  $T_1$  在释放锁之后又申请了锁，违背了 2PL，故不成立。

# HW6.3

3. 如果一个并发调度中的所有事务都遵循两阶段锁协议，该并发调度还会出现脏读问题吗？如果不会，请解释理由；如果会，请给出一个例子。

t	T1	T2
1	xL1(A)	
2	A=A-100	
3	xL1(B)	
4	Write(A)	
5	Unlock(A)	
6		sL2(A)
7	B=B+100	Read(A)
8	Write(B)	
9	Unlock(B)	
10	Rollback	

两阶段锁(2PL): 不要求所有的锁保持到事务结束

严格2PL(S2PL): 2PL基础上要求所有X锁保持到事务结束

强2PL(SS2PL): 2PL基础上要求所有锁都保持到事务结束

脏读，事务T1在释放A锁后Rollback，此时T2所读取的A值是错误的

# HW6 . 4

---

4. 考虑下面两个事务：

T1 = r1(A) w1(A) r1(B) w1(B)

T2 = r2(B) r2(A) w2(A) w2(B)

假设调度器插入了 X 锁动作，如下所示：

T1 = xL1(A) r1(A) w1(A) xL1(B) r1(B) w1(B) , AFTER COMMIT: U1(A) U1(B)

T2 = xL2(B) r2(B) xL2(A) r2(A) w2(A) w2(B) , AFTER COMMIT: U2(A) U2(B)

现在 T1 和 T2 并发执行，问：

- 1) T1, T2 并发执行能保证数据库的一致性吗？为什么？
- 2) 如果不使用上面的加锁方式，而是约定事务按先 A 后 B 的顺序请求锁，T1、T2 并发执行会产生死锁吗？为什么？
- 3) 如果使用等待—死亡(wait-die)死锁预防方案，假设 T1 先开始，那么如果产生死锁时系统将采取什么样的动作？



# HW6 . 4

---

- (1) 能。事务T1、T2都等到commit之后才释放x锁，能保证一致性。(死锁由系统处理，不在事务的考虑范围内)
- (2) 不会。约定顺序之后，T1和T2都需要先申请A上的锁，无论哪个事务申请成功，另一个事务都只能等待A锁的释放，而不能申请其他锁。等申请成功的事务提交释放A锁后，另一个事务才能继续执行，因此不会产生死锁。
- (3) 如果采用wait-die方案，最初T1获得A上的锁，T2获得B上的锁，T1申请B上的锁发现被T2占用，因为T1先开始，所以T1进入等待状态，当T2申请A上的锁发现被T1占用，开始回滚，此时T1获得B上的锁顺利运行，T2在T1提交以后获得B上的锁继续执行。