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

实体完整性，**参照完整性**，用户自定义完整性

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

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, 读者.姓名 Having COUNT(借阅.图书号) > 3;

7.

```sql
SELECT 读者号, 姓名
FROM 读者
WHERE NOT EXISTS (
    SELECT 1
    FROM 借阅
    JOIN 读者 AS 李林 ON 借阅.读者号 = 李林.读者号
    WHERE 李林.姓名 = '李林'
    AND 借阅.图书号 IN (
        SELECT 图书号
        FROM 借阅
        WHERE 借阅.读者号 = 读者.读者号
    )
);
```

```sql
SELECT 读者号, 姓名
FROM 读者
WHERE 读者号 NOT IN (
    SELECT DISTINCT 借阅1.读者号
    FROM 借阅 AS 借阅1
    WHERE 借阅1.图书号 IN (
        SELECT 借阅2.图书号
        FROM 借阅 AS 借阅2
        JOIN 读者 ON 借阅2.读者号 = 读者.读者号
        WHERE 读者.姓名 = '李林'
    )
);
```

Note

- 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

Note

- 分组操作后存在的属性包括：聚集函数用到的属性以及 Group by 后出现的属性

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→BC，B→CE,A→B，AB→C，AC→DE,E→A }

(1)求出 F 的最小函数依赖集（要求写出求解过程）

- 将右边写成单属性并去除重复FD
- F={A→B，A→C，B→C，B→E，AB→C，AC→D，AC→E，E→A}
- 消去左部冗余属性
- A→C,可推出AB→BC,从而推出AB→C,所以AB→ C中的B是冗余属性
- A→C,AC→D,AC→E可推出A→AC,A→D,A→E因此可去除AC→D中的C和AC→E中的C
- F={A→B，A→C，B→C，B→E，A→D，A→E，E→A }
- 消去冗余函数依赖
- A→C，A→E 冗余
- 得最小函数依赖集：F={A→B，B→C，B→E，A→D，E→A}，答案不唯一

3.1.2

(2)求 R 的候选码，并给出证明

- X → U ∈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 → U成立，则X是R的一个候选码。
- 因为A → A ，A → B ，A → C (A → B ,B → C) ，A → D ，A → E (A → B ,B → E) ，并且不存在A的真子集Y，使得Y→ U成立，所以A为候选码。
- 因为E → A,从而E → B ，E → C，E → D，E → E,并且不存在E的真子集Y，使得Y→ U成立，所以E也为候选码。
- 因为B → E，从而B → A(B → E，E → A)，B → B，B → C，B → D(B → E，E → A，A → D),并且不存在B的真子集Y，使得Y→ U成立，所以B也为候选码。
- 因此候选码为{A},{B},{E}

3.2.1

- 2.已知有关系模式 R(A，B，C，D，E)，R 上的一个函数依赖集：
  F={A→BD，BC→D，DCE→A，D→B，E→D }

(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→B，A→D，BC→D，DCE→A，D→B，E→D}

- 消去左部冗余属性
- E→D可推出E→DE，进一步推出CE→CDE，再由DCE→A传递律得到CE→A，即左部的D属性冗余
- A→D，D→B,可推出 A→B，因此A→B冗余函数依赖，删去
- 故最小函数依赖集 F={A→D，BC→D，CE→A，D→B，E→D}

3.2.2

(2)求 R 的候选码

- X → U ∈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 → U成立，则X是R的一个候选码。
- 因为 CE → A ，CE → B (CE → A，A → D，D → B)，CE → C，CE → D (CE → A，A → D) ，CE → E ，并且不存在A的真子集Y，使得Y→ U成立，所以CE为候选码。
- CE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→D，BC→D，CE→A，D→B，E→D}
- 主属性为 C,E，非主属性为A,B,D
- 存在非主属性局部函数依赖于主码： E→D，D局部依赖于E
- 所以是 1NF.

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。

- 最小FD集：F={A→D，BC→D，CE→A，D→B，E→D}
- 把所有不在F中出现的属性组成一个关系模式R’=Ø
- 对F按相同左部分组，得到
  q={R1(A,D),R2(B,C,D),R3(A,C,E),R4(B,D),R5(D,E)}
- R4为R2的子集，删去
- 主码为{C,E}，但为R3的子集
- 因此p=q ={R1(A,D),R2(B,C,D),R3(A,C,E),R5(D,E)}

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→E，A→B，B→C，C→D}

(1)该关系模式满足第几范式？为什么？

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

3.3.2

(2)如果将关系模式R分解为：R1(A,B,E)，R2(B,C,D)，R3(A,F,G),该数据库模式最高满足第几范式？

- R1(A,B,E)：F1={A→B，A→E} 主码A，所有不平凡、完全的函数依赖的决定因素都是A主码，故R1满足BCNF.
- R2(B,C,D)：F2={B→C，C→D} 主码B，存在D对于B的传递依赖，因此不满足3NF，但是不存在局部依赖R2满足2NF。
- R3(A,F,G)：无任何函数依赖，满足BCNF

- 整个关系模式最高满足2NF（如果说子模式最高就是 BCNF）

3.3.3

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

- 最小函数依赖集为F={A→B，B→C，C→D，A→E}
- F、G属性不在上述依赖集中出现
- 对F集按相同的左部分组，q={R1(A,B,E),R2(B,C),R3(C,D)}
- R 的主码为{A,F,G}，p=q ∪ R’{A,F,G}，无子集冗余
- p={R1(A,B,E),R2(B,C),R3(C,D)，R’{A,F,G}}

3.3.4

(4)请将关系模式 R 无损连接地分解到 BCNF，要求给出步骤

- A->E是独立的，先后消除不影响，先消除它，得到R1(A,E),R2(A,B,C,D,F,G)
- A->B,B->C,C->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R2中的A->B:Rx(A,C,D,F,G) 依赖变成A->C,C->D
- 再消除A->C:R1(A,E),R2(A,B),R3(A,C),R4(A,D),R5(A,F,G)
- 再消除C->D:R1(A,E),R2(A,B),R3(C,D),R4(A,C),R5(A,F,G)

3.3.4

(4)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->C:Rx(A,B,D,F,G) 依赖变成A->B,B->D.
- 再消除A->B:R1(A,E),R2(B,C),R3(A,B),R4(A,D),R5(A,F,G)
- 再消除B->D:R1(A,E),R2(B,C),R3(B,D),R4(A,B),R5(A,F,G)

- 先消除R2中的C->D:Rx(A,B,C,F,G) 依赖变成A->B,B->C
- 再消除A->B:R1(A,E),R2(C,D),R3(A,B),R4(A,C),R5(A,F,G)
- 再消除B->C:R1(A,E),R2(C,D),R3(B,C),R4(A,B),R5(A,F,G)
- BCNF分解答案不唯一（答出一种即可）

Note

- 由于选择去除的冗余依赖及其先后去除顺序不同，最小函数依赖集可能不唯一
- 模式分解时，若存在无函数依赖的属性，其一定在主码中，按照算法模式分解后，添加主码集再去除子集即可

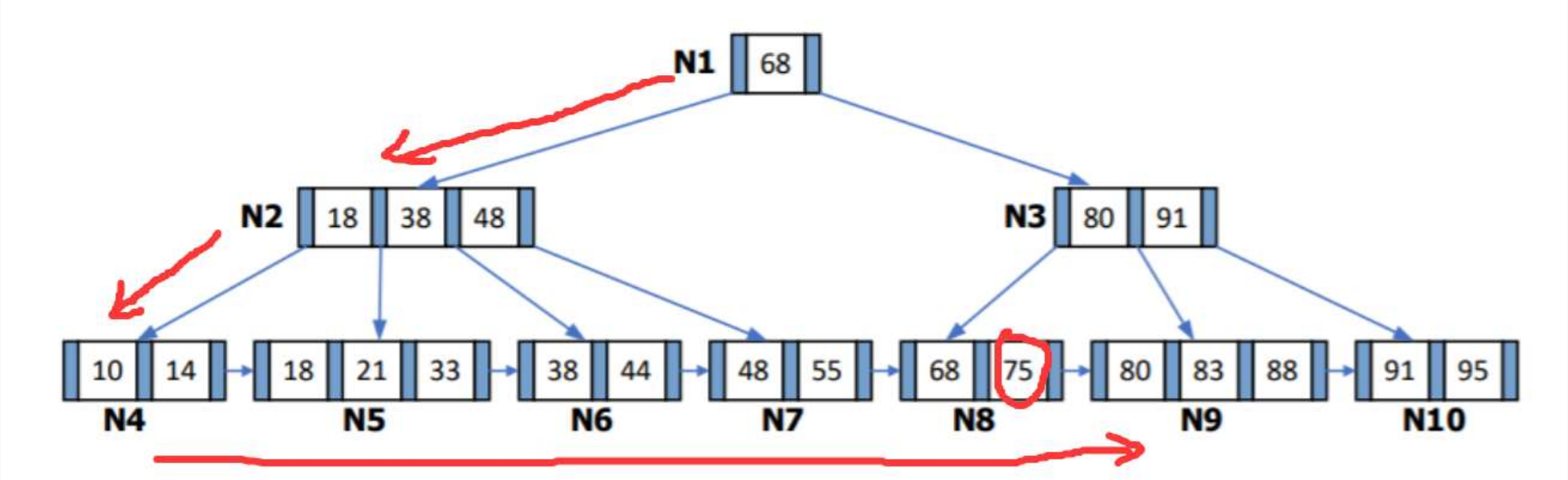
4.1

1.执行范围查询[17，76]时依次访问的节点序列。

B+树范围查询:首先查询17所在节点，再从叶子节点扫描到76所在节点

N1->N2->N4->N5->…->N9

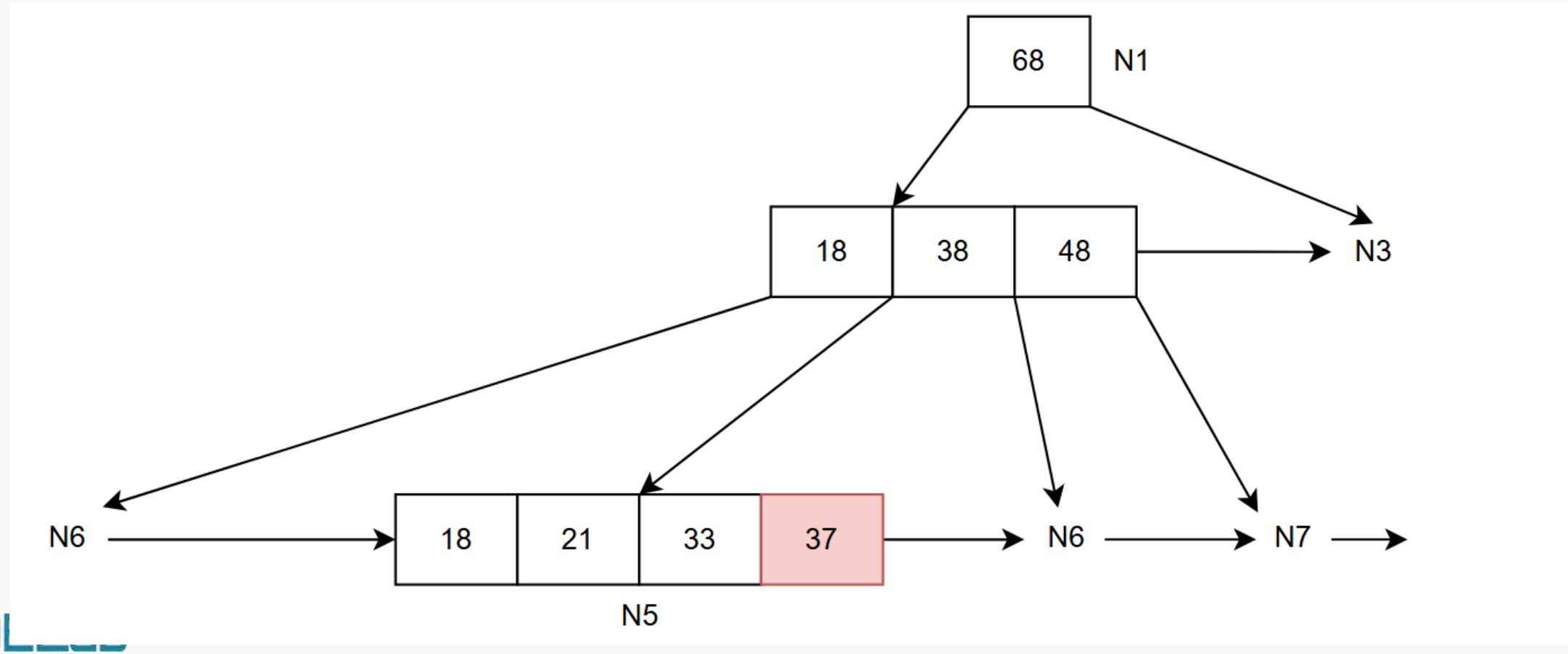
(扫描到N8时，N8最大值为75，不知道N9会不会包含76，需要访问N9)



4.1.2

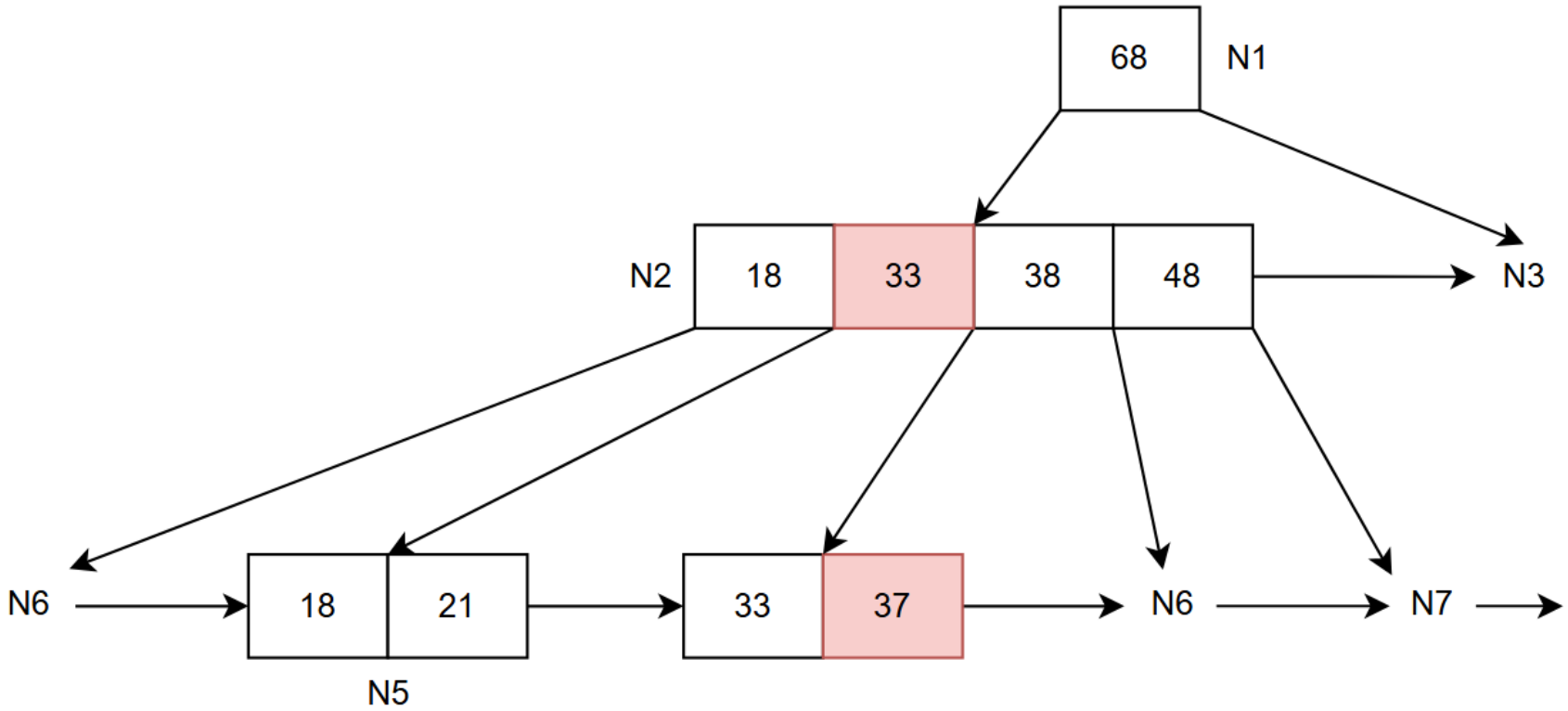
2．插入键值37后那些子树节点受到影响？画出受影响子树结构(插入后)。

N1 N2 N5 受影响



4.1.2

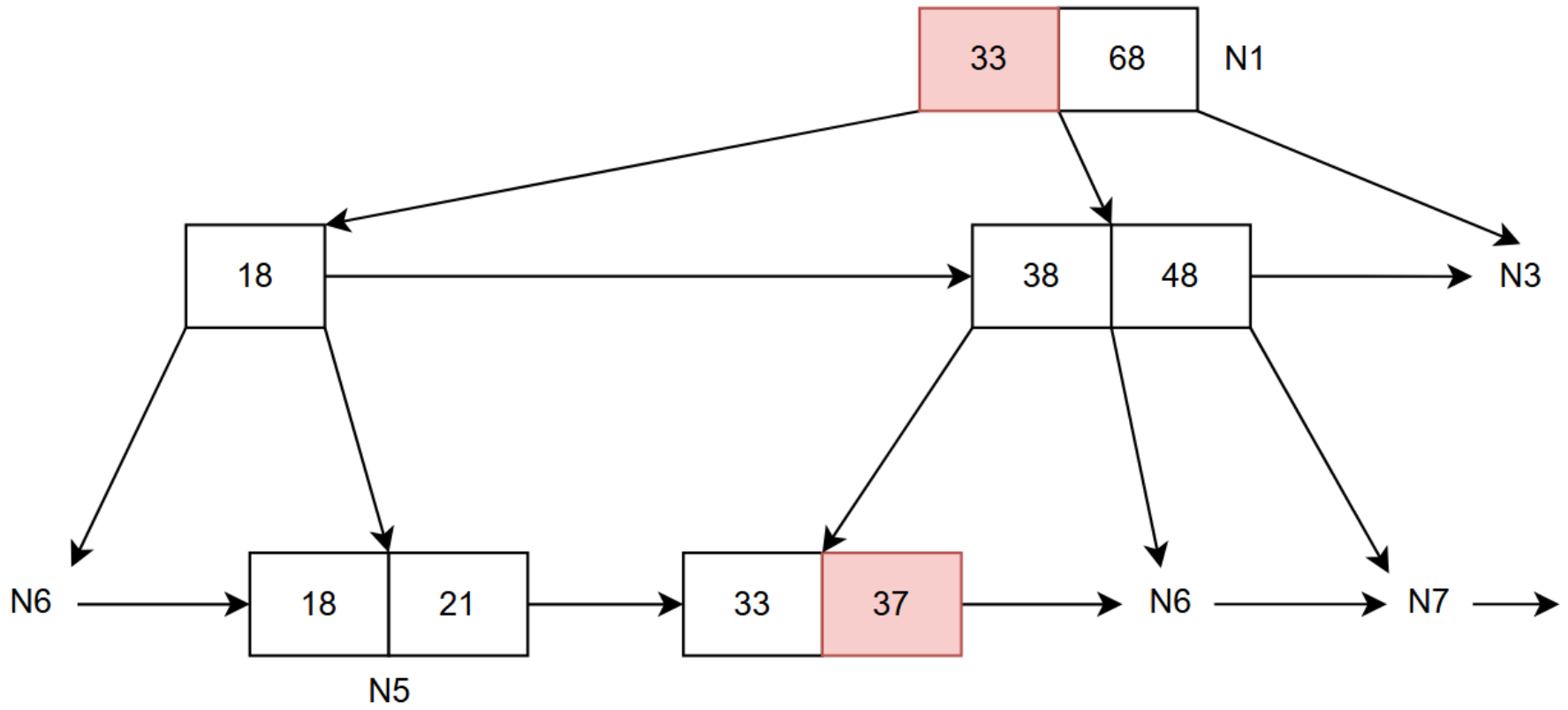
2.插入键值37后那些子树节点受到影响？画出受影响子树结构(插入后)。



4.1.2

2.插入键值37后那些子树节点受到影响？画出受影响子树结构(插入后)。

非叶节点分裂后，本层会少一个键，递归插入到上层（而叶节点不会少）



4.1.3

3．若节点内部无序，节点间保持有序，这一修改能否提升B+树的插入性能？

不能

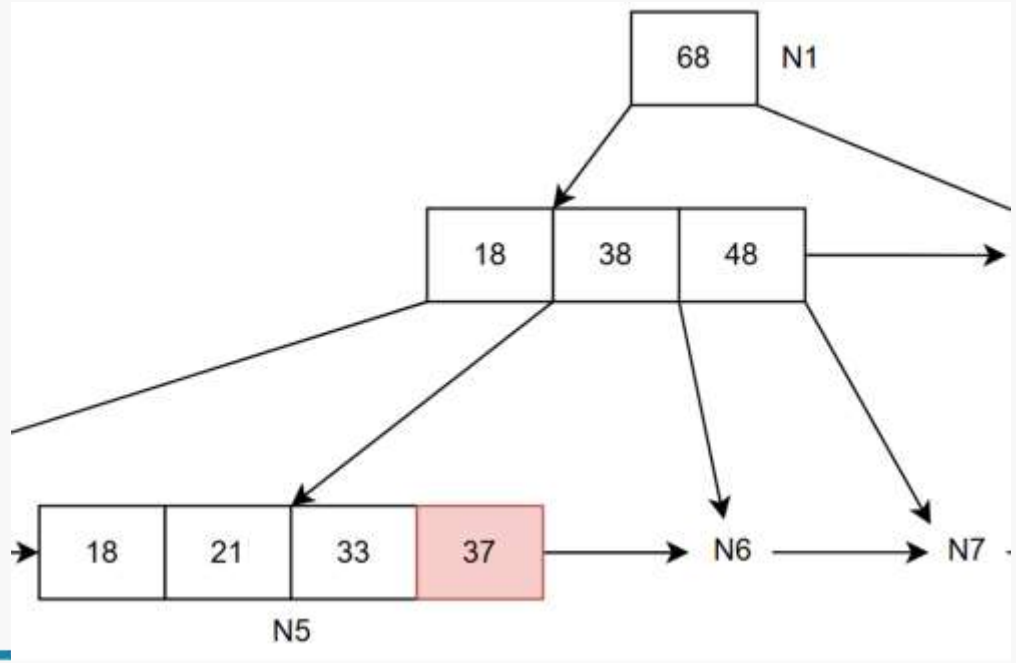
插入时的操作过程与修改前的B+树相比只减少了内存排序操作，I/O代价并没有减少。由于I/O代价决定了插入性能，因此此修改并不能优化 B+树的插入性能。

4.1.4

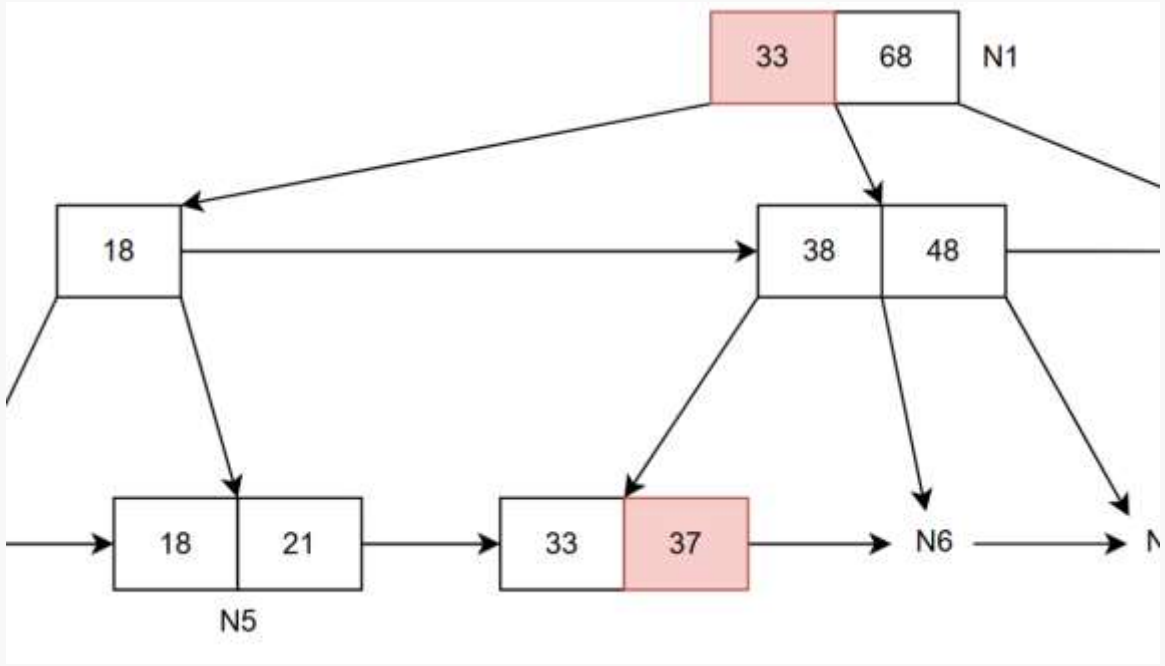
4．给每个叶子节点增加一个溢出节点，这一修改能否提升B+树的性能？以插入37为例，分别计算优化前后的I/O代价。

优化前:8次I/O

N1,N2,N5被读入内存共3个读I/O



N5,N2分裂共4个写I/O，N1插入1个写I/O

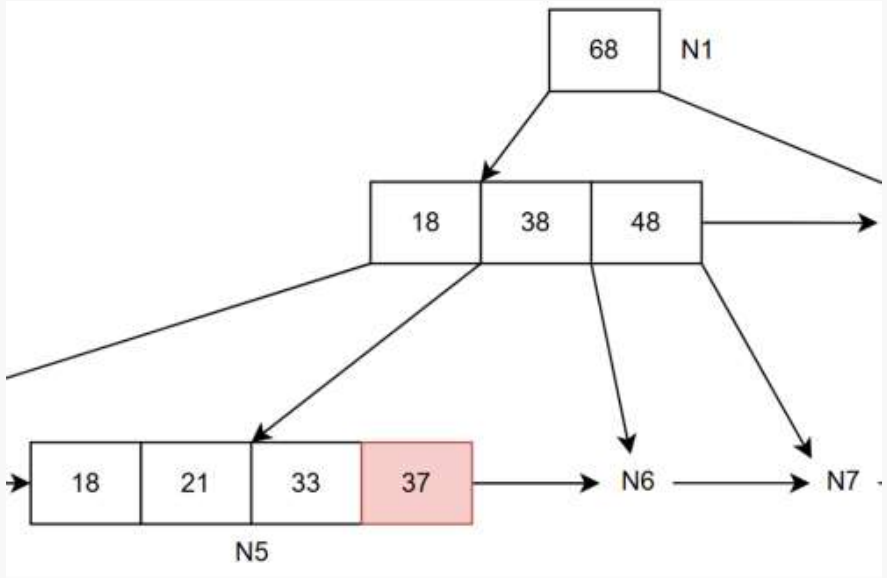


4.1.4

4．给每个叶子节点增加一个溢出节点，这一修改能否提升B+树的性能？以插入37为例，分别计算优化前后的I/O代价

优化后:5个I/O

N1,N2,N5被读入内存共3个读I/O。插入溢出节点一个写I/O



每个叶子节点预先分配好了一个溢出节点，由一个指针指向。因此要写入溢出节点就必须需要一次额外的读I/O。

4.2.1

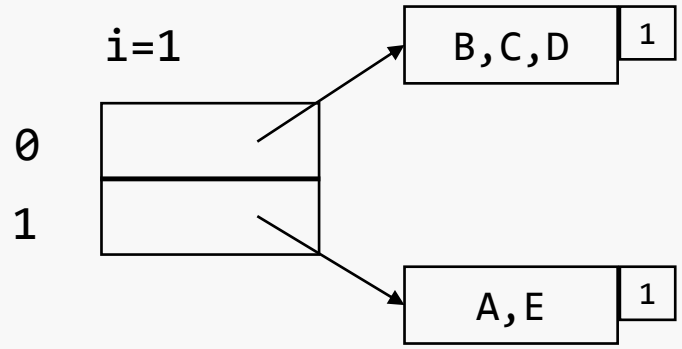
2．假设有如下的键值，现用 5 位二进制序列来表示每个键值的 hash 值。回答问题：

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]

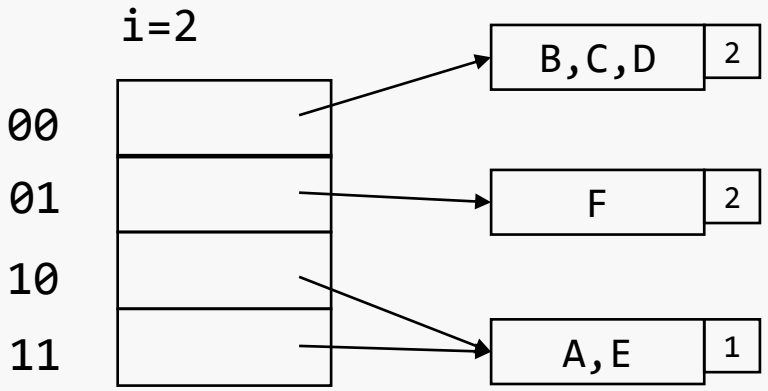
(1)如果将上述键值按 A 到 N 的顺序插入到可扩展散列索引中，若每个桶大小为一个磁 盘块，每个磁盘块最多可容纳 3 个键值，且初始时散列索引为空，则全部键值插入完成后该 散列索引中共有几个桶？并请写出键值 E 所在的桶中的全部键值。

4.2.1

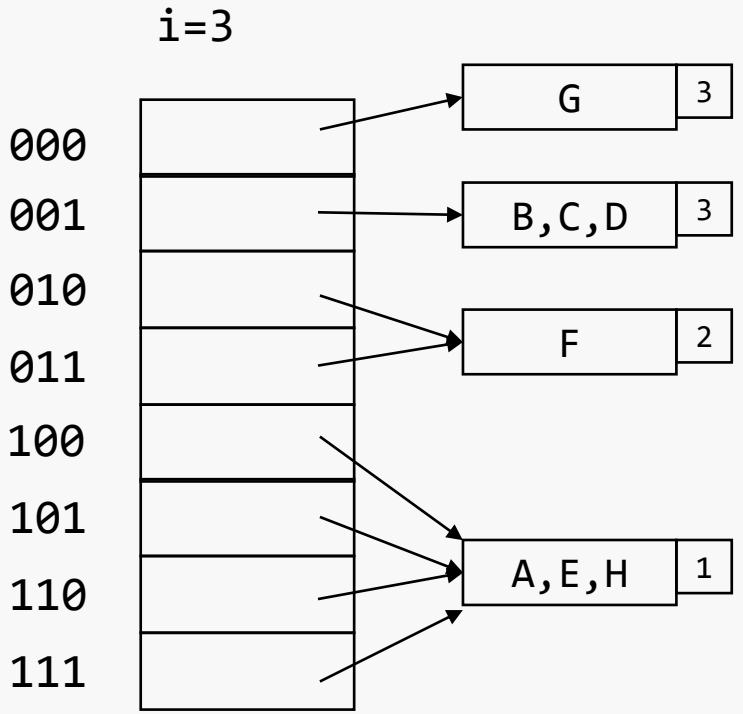
插入ABCDE



插入F

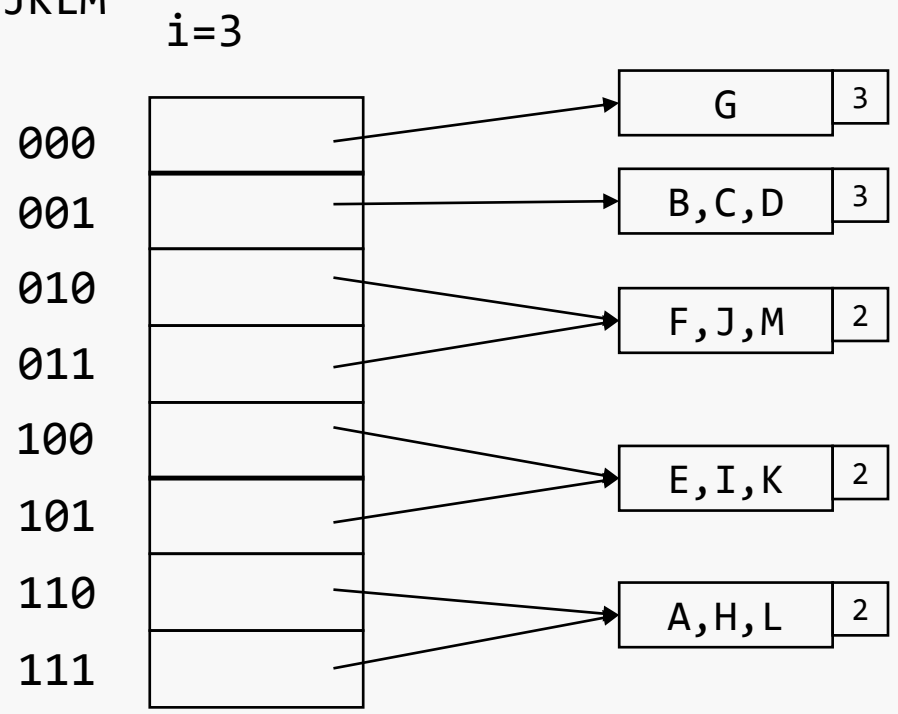


插入GH

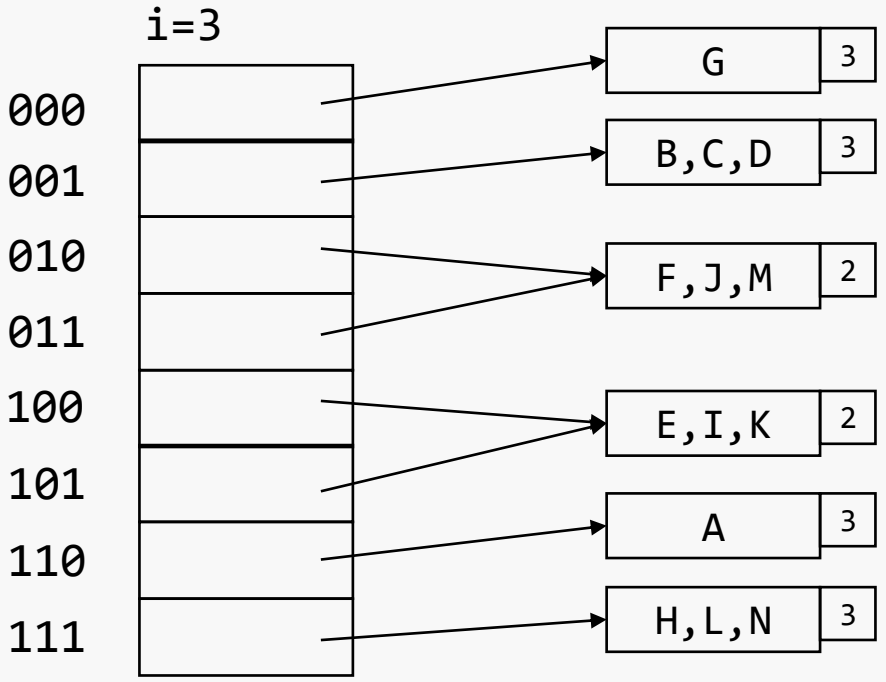


4.2.1

插入IJKLM



插入N



6个桶，E、I、K

4.2.2

- （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

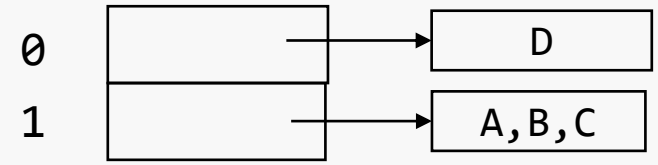
- 桶数＝n，h(k)的右i位＝m
- 若m<n，则记录位于第m个桶
- 若$n \le m < 2^i$,则记录位于第 $m-2^{i-1}$ 个桶
- n的选择：总是使n与当前记录总数r保持某个固定比例

4.2.2

插入ABCD

i=1

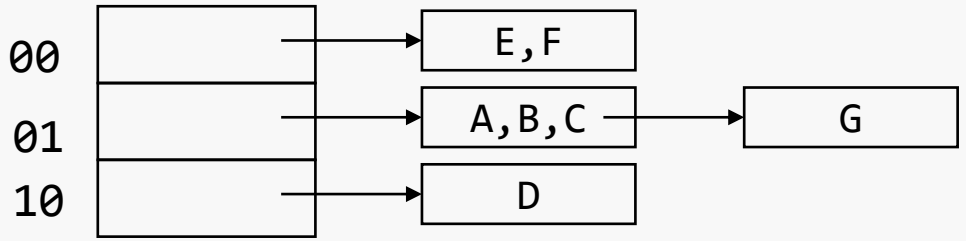
最多6*0.8=4.8



插入EFG

i=2

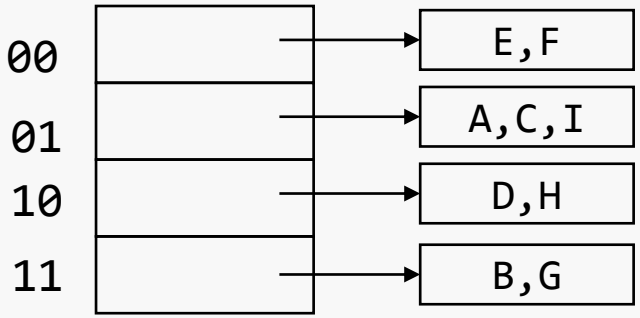
最多9*0.8=7.2



插入HI

i=2

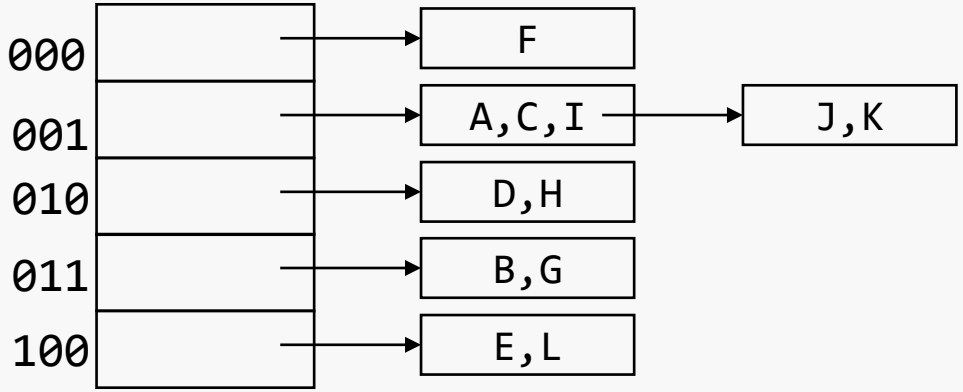
最多12*0.8=9.6



插入JKL

i=3

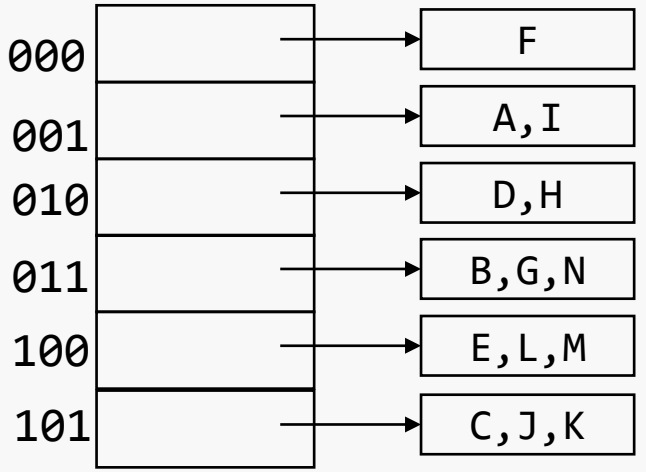
最多15*0.8=12



插入MN

i=3

最多18*0.8=14.4



6个桶，B、G、N

HW5.1

1. 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。

HW5.2

2. 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>
---------- ③
20) <T4, Commit Transaction>

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。

HW5.2

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

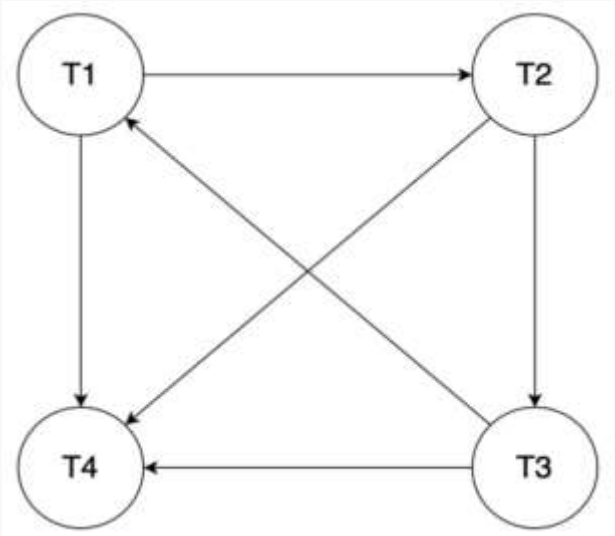
- 检查点后：T4：undo
- undo：F=100,G=20,F=20
- 所以，A=75,B=250,C=65,D=46,E=55,F=20,G=20

HW5.3

3. 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

HW5.4

4. 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->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。

HW5.5

5. 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