1.8 列出文件处理系统和 DBMS 的四个主要区别。  
Answer:

文件处理系统和DBMS的四个主要区别如下:

• 存储在数据库中的数据，其逻辑结构可能不同于其物理结构，DBMS 不仅管理数据的物理结构还管理数据的逻辑结构，DBMS 的使用者只需知道数据的逻辑结构而不需要关心其物理结构。文件系统只有物理结构而没有逻辑结构。  
• DBMS以一种高效的方式访问数据，一条数据在物理结构上只需存储一次。文件系统往往不能做到让不同的程序高效地访问数据，一条数据常常有很多副本，容易出现数据冗余。  
• 使用DBMS时，数据的各种约束条件由DBMS检查，程序员只需在 DDL中声明所有约束。而使用文件系统时，每插入一条数据，都需要程序员的代码来检查是否满足约束条件。

• DBMS允许多个用户同时并发地访问同一个数据库并保证数据的一致性，文件系统不能很好地处理并发访问带来的问题。

1.9 解释物理数据独立性的概念， 以及它在数据库系统中的重要性。  
Answer:

物理独立性是指用户的应用程序与存储在磁盘上的数据库中数据是相互独立的。

物理独立性使应用程序与存储在磁盘上的数据相分离，应用程序不依赖于物理模式，因此物理模式改变了它们也无需重写。

2.9 考虑图 2-15 所示银行数据库。

a. 适当的主码是什么？

b. 给出你选择的主码， 确定适当的外码。

Answer:a. 主码：

branch: branch name

customer: customer name

loan: loan number

borrower: customer name, loan number

account: account number

depositor: customer name, account number

b. 外码：

branch: 无

customer: 无

loan: branch name

borrower: customer name, loan number

account: branch name

depositor: customer name, account number

2.10 考虑图2-8所示advisor关系，advisor的主码是s\_id。假设一个学生可以有多位指导老师。那么，s\_id还是advisor的主码吗？ 如果不是，advisor的主码会是什么呢？

Answer:

不是，若一位学生有多个指导老师，那么表中可能有多个元组有相同的s\_id取值，这违反了主码唯一的原则。此时主码应该为(s\_id, i\_id).

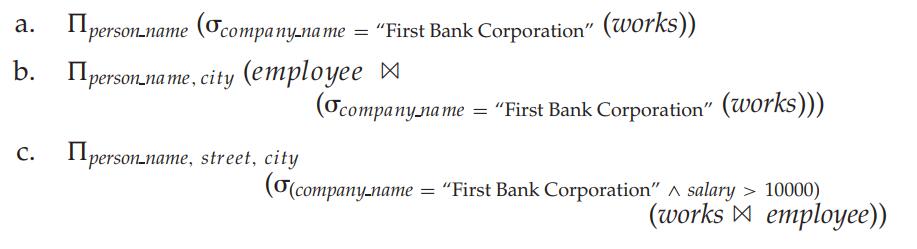
2.11 解释术语关系和关系模式在意义上的区别。  
Answer:

关系对应程序设计语言中变量的概念， 它是一个存在于物理存储空间中的实体， 是真正存放数据的地方。 关系模式对应于程序设计语言中类型的概念， 它仅仅是一种定义， 定义了某种类型的关系中应该存放什么属性。 关系可看作关系模式的实例化。

2.12 考虑图2-14所示关系数据库。给出关系代数表达式来表示下列每一个查询：

a. 找出为“FirstBank Corporation”工作的所有员工姓名。

b. 找出为“FirstBank Corporation”工作的所有员工的姓名和居住城市。  
c. 找出为“FirstBank Corporation”工作且挣钱超过10000美元的所有员工的姓名、街道地址和居住城市。

Answer:

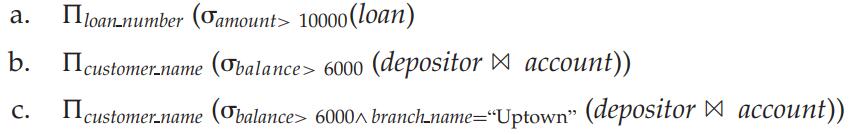
2.13 考虑图2-15所示银行数据库。对于下列每个查询，给出一个关系代数表达式：

a. 找出贷款额度超过 10000 美元的所有贷款号。

b. 找出所有这样的存款人姓名，他拥有一个存款额大于6000美元的账户。

c. 找出所有这样的存款人姓名，他在“Uptown”支行拥有一个存款额大于6000 美元的账户。

Answer:



2.14 列出在数据库中引入空值的两个原因。

Answer:

1、某个元组的某个属性未知，2、某个元组的某个属性不存在

2.15 讨论过程化和非过程化语言的相对优点。  
Answer:

非过程化语言简化了询问，用户可以不用关心操作的具体过程。但同时，需要花更多工作来优化询问器，来快速找到结果。过程化语言需要用户明确操作的具体过程，用户的灵活性更高，但也更容易出错，很难较快的表达目标结果，但设计好的操作过程，可能比询问其自动寻找，速度要快。

6.11 考虑图6-22所示关系数据库，主码加了下划线。给出关系代数表达式来表示下列每一个查询：

a. 找出 First Bank Corporation 的所有员工姓名。  
b. 找出 First Bank Corporation 所有员工的姓名和居住城市。

c. 找出 First Bank Corporation 所有年收入在 10000 美元以上的员工姓名和居住的街道、城市。

d. 找出所有居住地与工作的公司在同一城市的员工姓名。

e. 假设公司可以位于几个城市中。找出满足下面条件的所有公司，它位于Small Bank Corporation 所位于的每一个城市。  
Answer:

a.

b.

c.

d.

e.

6.13 考虑图 6-22 所示的关系数据库。分别给出下列查询的关系代数表达式：

a. 找出员工最多的公司。

b. 找出工资最少的员工所在公司。  
c. 找出人均工资比 First Bank Corporation 人均工资高的公司。

Answer:

a.

b.

c.

6.16 设R=(A, B)且S=(A, C), r(R)和s(S)是关系。分别给出与下列域关系演算表达式等价的关系代数表达式：

a. {< a > | ∃ b (< a, b > ∈ r ∧ b = 17)}

b. {< a, b, c > | < a, b > ∈ r ∧ < a, c > ∈ s}

c. {< a > | ∃ b (< a, b > ∈ r) ∨ ∀ c (∃ d (< d, c > ∈ s) ⇒ < a, c > ∈ s)}

d. {< a > | ∃ c (< a, c > ∈ s ∧ ∃ b1, b2 (< a, b1 > ∈ r ∧ < c, b2 >

∈ r ∧ b1 > b2))}

Answer:

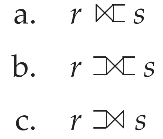
a.

b.

c.

d.

6.18 设R=(A, B)且S=(A, C), r(R)和s(S)是关系。使用特殊常量null，分别书写等价于下列表达式的元组关系演算表达式：



Answer:

a.

b.

c.

8.26 用Armstrong公理证明分解律的正确有效性。

Answer:

由自反律， βγ→β； 又由α→βγ 和βγ→β， 由传递率可知， α→β； 同理有α→γ。

8.27 用实践习题8.6中的函数依赖计算.

Answer:

第一次 repeat:result = {B}

• A → BC: 由于 A ⊈ result, result 不变

• CD → E: 由于 CD ⊈ result, result 不变

• B → D: 由于 B ⊆ result, result 变为 {B,D}  
• E → A: 由于 E ⊈ result, result 不变

第二次 repeat: result = {B,D}

• A → BC: 由于 A ⊈ result, result 不变

• CD → E: 由于 CD ⊈ result, result 不变

• B → D: 由于 B ⊆ result, result 不变

• E → A: 由于 E ⊈ result, result 不变

没有新属性加入 result, 算法终止。 = result = {B,D}*。*

8.28 证明实践习题 8.1 中的模式R的如下分解不是无损分解：

(A, B, C)

(C, D, E)

Answer:

根据提示，使用关系r如下表：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A | B | C | D | E |
| *a*1  *a*2 | *b*1  *b*2 | *c*1  *c*1 | *d*1  *d*2 | *e*1 *e*2 |

R1 = (A, B, C), R2 = (C, D, E) :  
a. R1 (r):

|  |  |  |
| --- | --- | --- |
| A | B | C |
| *a*1  *a*2 | *b*1  *b*2 | *c*1 *c*1 |

b. R2 (r) :

|  |  |  |
| --- | --- | --- |
| C | D | E |
| *c*1  *c*1 | *d*1  *d*2 | *e*1 *e*2 |

c. R1 (r) **⋈** R2 (r):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A | B | C | D | E |
| *a*1  *a*1  *a*2  *a*2 | *b*1  *b*1  *b*2  *b*2 | *c*1  *c*1  *c*1  *c*1 | *d*1  *d*2  *d*1  *d*2 | *e*1 *e*2 *e*1 *e*2 |

很明显, R1 (r) **⋈** R2 (r) ≠ r. 故该分解不是无损分解.

8.29 考虑如下关系模式r(A, B, C, D, E, F )上的函数依赖集F：

A → BCD

BC → DE

B → D

D → A

a. 计算 .

b.（使用 Armstrong 公理）证明AF是超码。

c. 计算上述函数依赖集F的正则覆盖；给出你的推导的步骤并解释。

d. 基于正则覆盖，给出r的一个3NF分解。

e. 利用原始的函数依赖集，给出r的一个BCNF分解。

f. 你能否利用正则覆盖得到与上面的r相同的BCNF分解？

Answer:

a. 第一次 repeat: result = {B}  
• A → BCD: 由于 A ⊈ result, result 不变.  
• BC → DE: 由于 BC ⊈ result, result 不变.  
• B → D: 由于 B ⊆ result, result 变为 {B; D}.  
• D → A: 由于 D ⊆ result, result 变为 {A; B; D}  
第二次 repeat: result = {A; B; D}  
• A → BCD: 由于 A ⊆ result, result 变为 {A; B; C; D}

BC → DE: 由于 BC ⊆ result, result 变为 {A; B; C; D; E}.  
• B → D: 由于 B ⊆ result, result 不变.  
• D → A: 由于 D ⊆ result, result 不变.

第三次 repeat 时，没有新属性加入 result, 算法终止。 B+ = {A; B; C; D; E}

b. • A → BCD 条件

• BCD → BC 自反律

• A → BC 传递律

• BC → DE 条件

• A → DE 传递律

• ABCD → BCDE 增补律

• A → ABCD 增补律

• A → BCDE 传递律

• AF → ABCDEF 增补律

• AF 是超码 超码定义

c. 第一次 repeat: F = Fc  
• 对于依赖A→BCD,D是无关属性，因为{A→BC; B→D}可推出A→D，将D去掉

• 对于依赖BC→DE,D是无关属性，因为B→D可推出BC→D，将D去掉。C是无关属性，因为由F可推出B→DE,将C去掉

• 对于依赖B→D,没有无关属性

• 对于依赖D→A,没有无关属性

第二次repeat: Fc={A→BC;B→E;B→D;D→A}，将B→E; B→D合并为B→DE。其他不变  
第三次repeat:Fc不变，Fc={A→BC;B→DE;D→A},算法结束。

d. 过程如下：

对 Fc中的每一个依赖，生成如下模式：R1(A,B,C),R2(B,D,E),R3(A,D)

Ri 中没有包含候选码的，又已知 AF 为候选码，增加一个 R4(A,F)

没有 R 包含于另一个 R，故得到一个 3NF 分解为：R1(A,B,C),R2(B,D,E),R3(A,D),R4(A,F)

e. result={r(A，B，C，D，E，F)}

A→BCD,但A不是r的超码，result={r1(A，B，C，D)，r2(A，E，F)}

A→E,但A不是r2的超码，result={r1(A，B，C，D)，r2(A，E)，r3(A，F)}

所有关系都已经是BCNF，故算法终止。

f. 可以。一组函数依赖和它的正则覆盖有相同的闭包， 而 BCNF 分解使用的也是闭包，所以得出的分解结果是相同的。

8.30 列出关系数据库设计的三个目标， 并解释为什么要达到每个目标。

Answer:

**•** 无损分解，因为要保证信息不丢失。 **•** 保持依赖，在检查依赖关系时避免做连接运算。 **•** 最小信息冗余，节省存储空间，易于保持数据的一致性。

14.12列出 ACID 特性，解释每一特性的用途。

Answer:

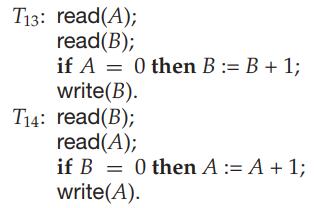
• 原子性 (atomicity): 事物是不可分割的。事务的所有操作要么全部执行，要么全部不执行。事务执行的过程中数据库可能会出现暂时性地不一致地现象，原子性避免了部分执行的事务导致的不一致。

• 一致性 (consistency): 隔离执行事务时，事务的操作能保持数据库的一致性。一致性保证了数据库在执行事务后仍然能真实反映现实世界。

• 隔离性 (isolation): 多个事务并发执行时，对于任意一对事务，系统保证在其中一个事务看来，另一个事务都执行完毕或未开始，即每个事务都不知道系统中还有其他事务并发执行。隔离性避免了多个事务交叉执行可能带来的不一致性。

• 持久性 (durability): 一个事务成功执行后，即使系统出现了故障，该事务对数据库的影响也应该是永久的。持久性保证成功执行的事务在任何情况下都能够起作用。

14.15 考虑以下两个事务：



设一致性需求为 A = 0 ⋁ B = 0， 初值是 A = B = 0。

a. 说明包括这两个事务的每一个串行执行都保持数据库的一致性。  
b. 给出 T13 和 T14 的一次并发执行，执行产生不可串行化调度。  
c. 存在产生可串行化调度的 T13 和 T14 的并发执行吗？

Answer:a.两个事务共有两个串行执行: T13 T14 and T14T13.  
Case 1:

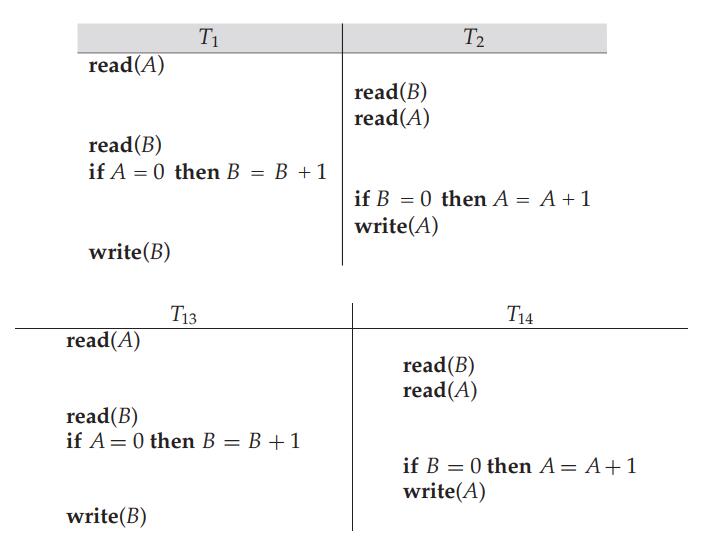
|  |  |  |
| --- | --- | --- |
|  | A | B |
| initially | 0 | 0 |
| after T13 | 0 | 1 |
| after T14 | 0 | 1 |

满足一致性: A *=* 0∨ B *=* 0≡ T ∨ F *=* TCase 2:

|  |  |  |
| --- | --- | --- |
|  | A | B |
| initially | 0 | 0 |
| after T13 | 1 | 0 |
| after T14 | 1 | 0 |

满足一致性: A = 0 ∨ B = 0 ≡ F ∨ T = T

b.



c. 不行。

假设并发执行的第一条语句是T13的read(A)，那么T14的最后一条指令write(A)肯定在其之下。虽然可以通过不断的调度可以把第一条语句read(A)与T14的其他指令交换，但由于T14的最后一条指令与其冲突, 所以无法把T14的所有指令向上串行化，也无法把T13的所有指令向上串行化。同理，若并发执行的第一条语句是 T14 的 read(B), 也无法实现非冲突可串行化。因为对于T13，T14的任意并发执行，任意调度都无法与串行调度冲突等价，也即使任意调度都是非冲突串行化的。所以不存在可以产生可串行化调度的T13和T14的并发执行。

15.1 证明两阶段封锁协议保证冲突可串行化，并且事务可以根据其封锁点串行化。

Answer:

给定任意两个事务Ti, Tj，设其封锁点为ai, aj，先证明当优先级图中存在有向边Ti→Tj 时，必有ai < aj：当有向边 Ti, Tj存在时，表明Ti, Tj先后读/写同一个数据项，且 Ti先于Tj，在封锁协议中，读取数据需要获得锁，且同一个数据项的数据锁在同一时间只能由一个事务持有，则Ti先获得锁，Ti在其封锁点ai后释放锁，之后Tj才能获得锁，aj在Tj获得锁之后，所以有ai < aj。对于任意 N个事务，不存在两个封锁点相同的事务，不妨令 a1 <… < an，则对于优先级图中任意边Ti→Tj，都有i < j，即优先图中无环，所以是冲突可串行化的，且 T1,…, Tn是一个优先图的拓扑排序。

15.2考虑下面两个事务：

T34：read(A)；read(B)；if A = 0 then B := B + 1； write(B).

T35：read(B)；read(A)；if B = 0 then A := A + 1； write(A).

给事务T34与T35增加加锁、 解锁命令， 使它们遵从两阶段封锁协议。这两个事务会引起死锁吗？

Answer:

a.T34:

lock-S(A)

read(A)

lock-X(B)

read(B)

if A = 0

then B := B + 1

write(B)

unlock(A)

unlock(B)

T35:

lock-S(B)

read(B)

lock-X(A)

read(A)

if B = 0

then A := A + 1

write(A)

unlock(B)

unlock(A)

b. 当调度如下时，这两个事务会引起死锁。

|  |  |
| --- | --- |
| T34 | T35 |
| lock-S(A) |  |
|  | lock-S(B) |
|  | Read(B) |
| read(A) |  |
| lock-X(B) |  |
|  | lock-X(A) |

15.3 强两阶段封锁协议带来什么好处？ 它与其他形式的两阶段封锁协议相比有何异同？

Answer:

强两阶段封锁协议要求事务提交之前不得释放任何锁。 ⽽普通的两阶段封锁协议只要求事务在增长阶段可以获得锁但不能释放锁，在释放阶段可以释放锁但不能获得锁，严格两阶段封锁协议则只要求排他锁必须在事务提交之后才能释放。

强两阶段封锁协议的好处在于事务若写⼊⼀个数据，则在它提交之前没有其他事务可以读它写⼊的数据，避免了级联回滚。 在强两阶段封锁条件下，事务可以按其提交的顺序串⾏化。

15.20 严格两阶段封锁协议带来什么好处？ 会产生哪些弊端？

Answer:

严格两阶段封锁协议避免了级联回滚，减小了回滚带来的性能影响。但由于两阶段封锁协议中事务释放排他锁的时间推迟到事务提交之后，则可获得的调度集是普通两阶段封锁协议可获得的调度集的⼦集，即会导致并发度的下降。

15.22考虑树形协议的一个变种，它称为森林协议。 数据库按有根树的森林的方式组织。每个事务必须遵从以下规则：

•每棵树上的首次封锁可以在任何数据项上进行。

•树上的第二次以及以后的封锁申请仅当被申请结点的父结点上有锁时才能

发出。

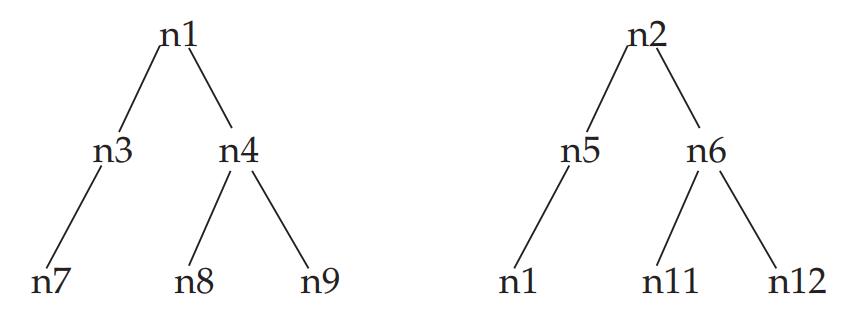
•数据项解锁可在任何时候进行。

•事务释放数据项后不能再次封锁该数据项。

证明森林协议不保证可串行性。

Answer:

对于两棵树:

****

执行如下调度：

|  |  |
| --- | --- |
| T1 | T2 |
| Lock(n1) |  |
| Lock(n3) |  |
| Write(n3) |  |
| Unlock(n3) |  |
|  | Lock(n2) |
|  | Lock(n5) |
|  | Write(n5) |
|  | Unlock(n5) |
| Lock(n5) |  |
| Read(n5) |  |
| Unlock(n5) |  |
| Unlock(n1) |  |
|  | Lock(n3) |
|  | Read(n3) |
|  | Unlock(n3) |
|  | Unlock(n2) |

显然这样的调度满足森林协议但是不是可串行化的。

15.24 如果通过死锁避免机制避免了死锁后，饿死仍有可能吗？解释你的答案。

Answer:

仍然有可能会饿死，通过死锁恢复，在回滚的时候需要选择牺牲者，这就可能导致饿死。而死锁预防也可能导致饿死，原因是一样的。

16.18 考虑图 16-5 中的日志。假设恰好在<T0 abort>日志记录写出之前系统崩溃。请解释在系统恢复时会发生什么。

Answer:

假定在 <T0 abort> 之前发生崩溃，按照课本 16.4 的恢复算法:

在重做阶段:

• 找到最后一个 checkpoint，初始化 undo-list={T0; T1}.

• 从上到下扫描日志:<T1; C; 700; 600>，重做，C=600.

• <T1 commit>，将T1从undo-list中删除，undo-list={T0}.

• <T2 start>，将T2加入undo-list，undo-list={T0; T2}.

• <T2; A; 500; 400>，重做，A=400.

• <T0; B; 2000>，重做，B=2000.

在撤销阶段:

• 从后往前扫描日志: <T0; B; 2000>为redo-only，忽略.

• <T2; A; 500; 400>，撤销，A=500，写入日志:<T2; A; 500>.

• <T2 start>，将T2从undo-list中删除，undo-list={T0}，写入日志<T2 abort>.

• <T0; B; 2000; 2050>，撤销，B=2000，写入日志:<T0; B; 2000>.

• <T0 start>，将T0从undo-list删除，写入日志<T0 abort>， undo-list为空，结束.

恢复完毕后:A=500; B=2000; C=600.

10.17列出下列存储关系数据库的每个策略的两个优点和两个缺点：  
a. 在一个文件中存储一个关系。  
b. 在一个文件中存储多个关系（甚至是整个数据库）。  
Answer:a 优点：直接使用 OS 提供的文件系统，使数据库的设计变得容易。同时使得数据库的结构简单明了。

缺点：限制了数据库使用更复杂但性能好的结构提高性能。此外，因为 OS 的文件系统本身读取存储也很耗时，更限制了性能

b 优点：可以使用复杂的存储结构来提高数据库的性能。

缺点：增加了数据库的复杂性。增加数据库的大小。

10.18 在顺序文件组织中，为什么即使当前只有一条溢出记录，也要使用一个溢出块？  
Answer:

因为块已经是可以从磁盘读取的最小空间，无法使用更小的单位。而若不使用溢出块，采用别的方法将溢出块的数据放入顺序文件组织中去，会导致性能的降低，一点点的空间成本无疑比性能的下降更划算。

11.3用下面的关键码值集合建立一个 树

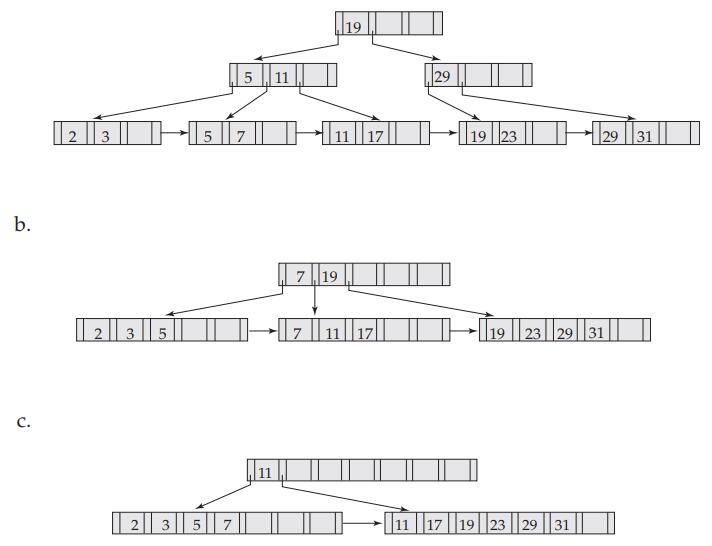
(2, 3, 5, 7, 11, 17, 19, 23, 29, 31)

假设树初始值为空，值按上升顺序加入。根据一个节点所能容纳指针数的下列情况分别构造 树：

a. 4 b. 6 c. 8

Answer:

a.

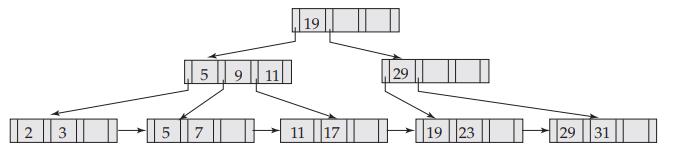


11.4 对习题11.3中的每一棵树，给出下列各操作后树的形状：

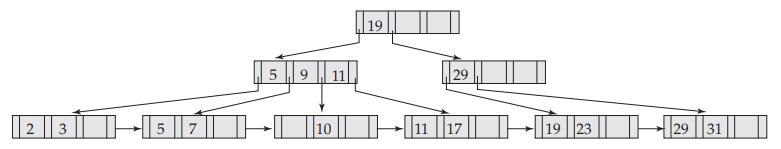
Answer:

对于11.3.a

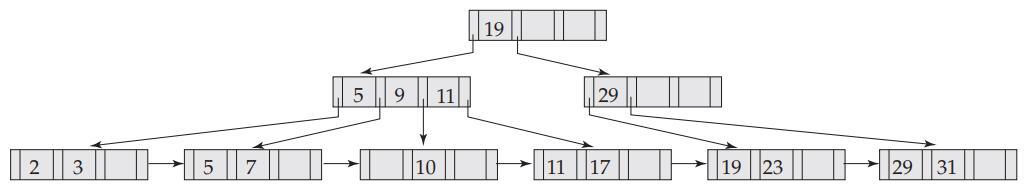
插入 9:

****

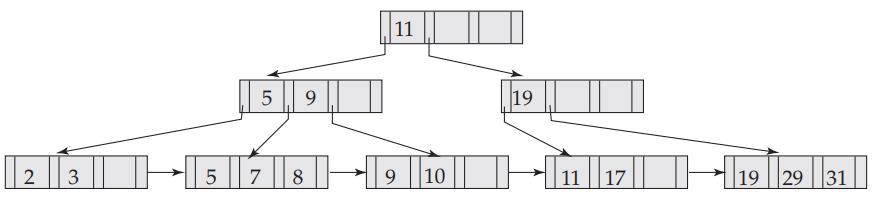
插入 10:

****

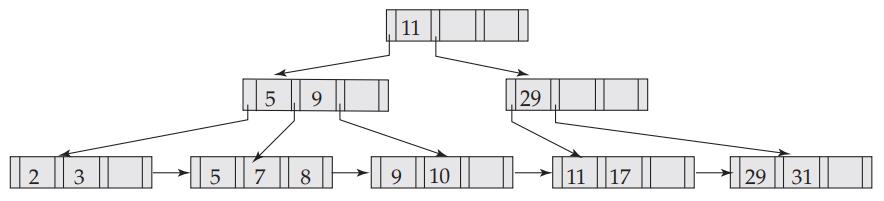
插入 8:

****

删除 23:

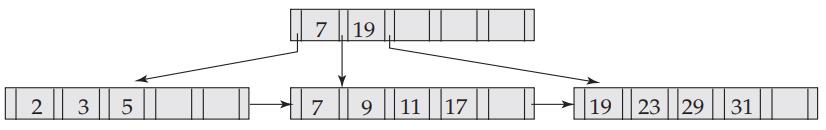
****

删除 19:

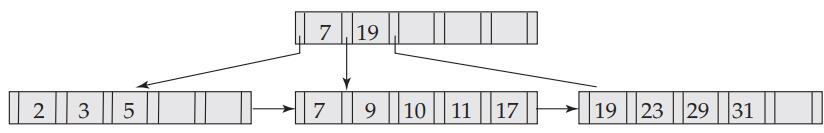
****

对于11.3.b

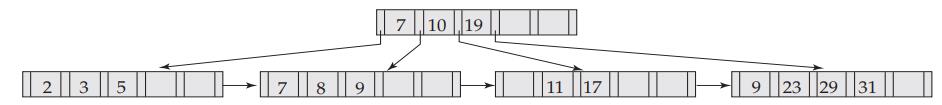
插入 9:

****

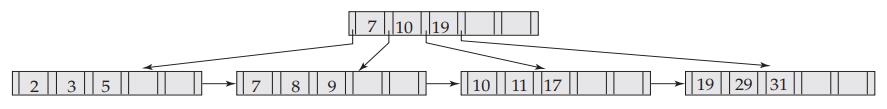
插入 10:

****

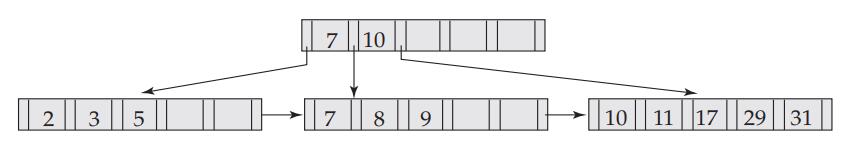
插入 8:

****

删除 23:

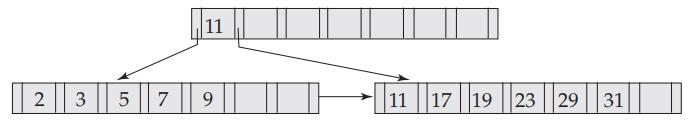
****

删除 19:

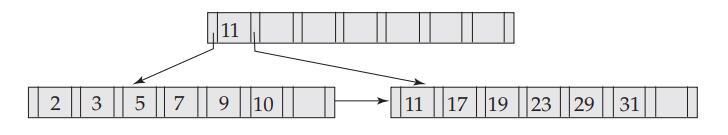
****

对于11.3.c

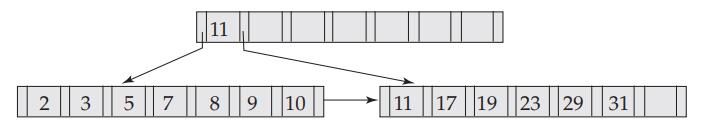
插入 9:

****

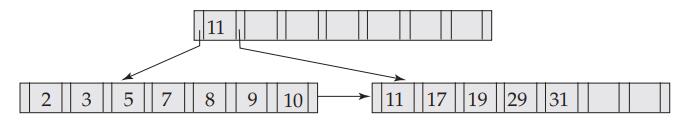
插入 10:

****

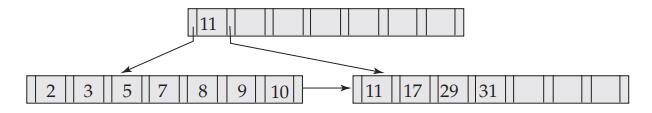
插入 8:

****

删除 23:

****

删除 19:

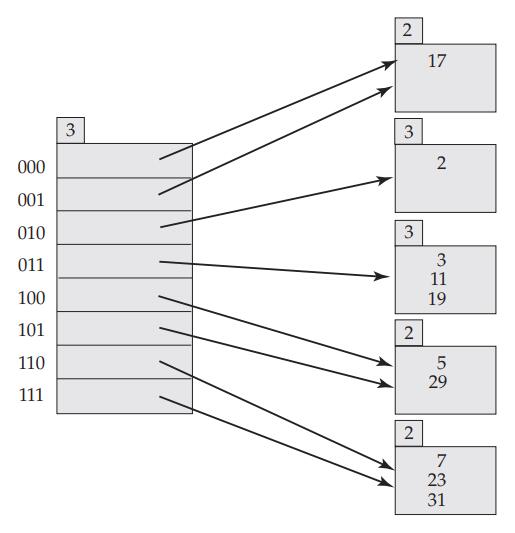
****

11.6 假设我们在一个文件上使用可扩充散列，该文件所含记录的搜索码值如下：

2, 3, 5, 7, 11, 17, 19, 23, 29, 31

如果散列函数为 h(x) = x mod 8，且每个桶可以容纳 3 条记录。给出此文件的可扩充散列结构。

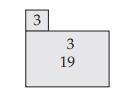
Answer:



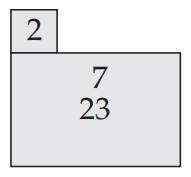
11.7 进行下列各步以后，习题11.6中的可扩充散列结构如何变化？

Answer:

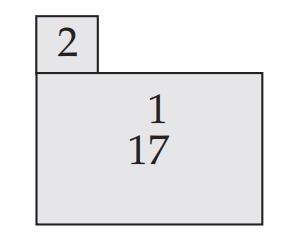
a. 删除11后，第三个桶变成：



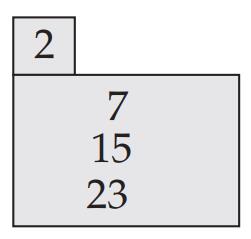
b. 删除31后，第五个桶变成：



c. 插入1后，第一个桶变成：



d. 插入15后，最后一个桶变成：



12.3 设关系r1(A, B,C)，r2(C, D, E)有如下特性：r1有20000个元组，r2有45000个元组，一块中可容纳25个r1元组或30个r2元组。估计使用以下连接策略计算r1 ⋈ r2需要几次块传输和磁盘搜索：

a. 嵌套循环连接.

b. 块嵌套循环连接.

c. 归并连接.

Answer:

可知r1有800块，r2有1500块。设有M个内存块。如果M > 800，连接可以在1500 + 800次磁盘访问中轻松完成，甚至可以使用纯嵌套循环连接。因此我们只讨论M ≤ 800 的情况。

a. 嵌套循环连接:

r1 作为外层循环需传输： 20000\*1500+800 = 30000800

r2 作为外层循环需传输： 45000\*800 + 1500 = 36001500 个块

b. 块嵌套循环连接:

如果将 r1 作为外层循环：

如果将 r2 作为外层循环：

c. 归并连接：

假设 r1 和 r2 最初并没有在连接键上排序，排序的总成本是：

然后写回内存总共需要：Bs + 1500 + 800 disk accesses次块传输和磁盘搜索