ADBS-习题课
(HW 1, 4)

1、假设某磁盘具有以下特性：

（1）有 16 个盘面和 8192 个柱面

（2）盘面直径为 2.5 英寸

（3）每磁道平均有 512 个扇区，每个扇区 512 字节

（4）每个磁道 10%被用于间隙

（5）磁盘转速为 9600 RPM

（6）磁头启动到停止需要 1ms，每移动 400 个柱面另加 1ms

（1）磁盘容量是多少？

$$8192 * 16 * 512 * 512\ B = 32GB$$

1、假设某磁盘具有以下特性：

（1）有 16 个盘面和 8192 个柱面

（2）盘面直径为 2.5 英寸

（3）每磁道平均有 512 个扇区，每个扇区 512 字节

（4）每个磁道 10%被用于间隙

（5）磁盘转速为 9600 RPM

（6）磁头启动到停止需要 1ms，每移动 400 个柱面另加 1ms

（2）如果一个块是 16KB，那么一个块的传输时间是多少？

1）16**KB**/512**B**=32个扇区，因此磁头需要扫过 **32**个扇区和扇区之间的 **31** 个间隙，其在一个磁道占比为 (32/512) * 90% + (31/512) * 10% = 0.062305

2）磁盘旋转一圈需要 1 / (9600 RPM) = 6.25 ms

3）传输一个块耗时为 6.25ms * 0.062305 = 0.389 ms

1、假设某磁盘具有以下特性：

（1）有 16 个盘面和 8192 个柱面

（2）盘面直径为 2.5 英寸

（3）每磁道平均有 512 个扇区，每个扇区 512 字节

（4）每个磁道 10%被用于间隙

（5）磁盘转速为 9600 RPM

（6）磁头启动到停止需要 1ms，每移动 400 个柱面另加 1ms

（3）最大寻道时间是多少？

1+8192/400=21.48ms

1、假设某磁盘具有以下特性：

（1）有 16 个盘面和 8192 个柱面

（2）盘面直径为 2.5 英寸

（3）每磁道平均有 512 个扇区，每个扇区 512 字节

（4）每个磁道 10%被用于间隙

（5）磁盘转速为 9600 RPM

（6）磁头启动到停止需要 1ms，每移动 400 个柱面另加 1ms

（4）平均旋转等待时间是多少？

磁盘转一圈所需时间为 6.25 ms

平均旋转等待时间: 6.25ms/2 = 3.125 ms

2、假设某块磁盘的参数如下：容量为 256GB，传输速率为 200MB/s，磁盘转速为 7200RPM，最大寻道时间为 10ms，平均寻道时间为 4ms，最小寻道时间为 0.5ms（指磁头寻道到相邻磁道的时间），一个磁道大小为 2MB。如果磁盘块大小为 4KB，请回答下面问题（所有结果均四舍五入保留小数点后 1 位）：

（1）随机读取一个磁盘块需要多少时间（ms）？

寻道时间：4.0 ms
旋转延迟：(1/(7200RPM))/2= 4.17 ms
传输时间：4KB/(200MB/s)=0.02 ms
4.0+4.17+0.02=8.2 ms

- **磁盘块读取时间**
  - 从“发出块存取请求”到“块位于主存”的时间
  - ＝寻道时间**S**＋旋转延迟**R**＋传输时间**T**＋其它延迟

2、假设某块磁盘的参数如下：容量为 256GB，传输速率为 200MB/s，磁盘转速为 7200RPM，最大寻道时间为 10ms，平均寻道时间为 4ms，最小寻道时间为 0.5ms（指磁头寻道到相邻磁道的时间），一个磁道大小为 2MB。如果磁盘块大小为 4KB，请回答下面问题（所有结果均四舍五入保留小数点后 1 位）：

（2）如果要读取 200 个磁盘块，并且这些磁盘块在单个磁道上连续存储，此时读取这 200 个磁盘块最少需要多少时间、最坏情况下需要多少时间、平均情况下需要多少时间？（单位：ms）

寻道时间：max-10ms，　min-0ms
旋转延迟：max-8.33ms，min-0ms
传输时间：0.02*200=4ms
最少：0+0+4=4 ms
最多：10+8.33+4=22.3 ms
平均：4+4.17+4=12.2 ms

HW4

1、已知有关系模式 R(A, B, C)和 S(B, C, D)，每个属性都占 10 个字节，请估计下面的逻辑查询计划的 T(U), S(U)以及结果关系中每个属性的 V 值（假设满足"Containment of Value Sets"，并且选择条件中的值都在关系中存在）：

$$U=\pi_{AD}[(\sigma_{A=3\wedge B=5}R)\bowtie S)]$$

相应的统计量如下：

```
T(R) = 100000,    V(R,A) = 20,    V(R,B) = 50,    V(R,C) = 150
T(S) = 5000,      V(S,B) = 100,   V(S,C) = 200,   V(S,D) = 30
```

选择：$T(R')=\frac{T(R)}{V(R,A)\times V(R,B)}=\frac{100000}{20\times 50}=100$

连接：$T(U)=\frac{T(R')\times T(S)}{V(S,B)\times V(S,C)}=\frac{100\times 5000}{100\times 200}=25$

由于每个属性都占 10 个字节，而 U 含有两个属性，所以

S(U) = 20(B)

由于限定了 A = 3,所以

V(U, A) = 1,　V(U, D) = V(S, D) = 30

In General $T(W)=\frac{T(R1)\bullet T(R2)}{\max\{V(R1,A),V(R2,A)\}}$

2、查询处理器在回答涉及 **R(A, B)** 和 **S(B, C)** 的查询"**Select R.*, S.* From R, S Where R.B=S.B and R.B<>10**"时，生成了逻辑查询计划 $\sigma_{R.B\neq 10}(R) \bowtie \sigma_{R.B\neq 10}(S)$，已知有关参数如下：

> **R** 和 **S** 的元组都是定长的，长度均为 40 字节。页大小为 4000 字节。**T(R)** = 500000，**T(S)** =100000，**V(R, A)** = 100，**V(R, B)** = 25，**V(S, B)** = 25，**V(S, C)** = 30。**R** 和 **S** 都在磁盘上，且 **R** 为连续存储，**S** 为不连续存储。可用内存 **M** 的大小为 148 个页。

- ✧ 对于上述查询计划中的连接操作，我们采用一种改进的散列连接算法。设连接关系为 **R1** 和 **R2**，对应的桶数组分别为 **G** 和 **H**：
  （1）散列 **R1** 时将其中的 1 个桶（设 G_1），放在内存中不写到磁盘中，其余的桶写入磁盘；
  （2）散列 **R2** 时，将落在桶 H_1 中的元组直接与内存中的 G_1 桶中的元组进行连接，并输出连接结果，其余的桶写入磁盘；
  （3）对磁盘中除 G_1 和 H_1 桶外的其余桶执行桶对桶的连接并输出连接结果；
  （4）为了减少 I/O 代价，我们希望内存中的 G_1 能够尽可能地大。
- ✧ 所有中间结果均采用物化方式传递。

请估计此查询计划的 I/O 代价。

设R1=$\sigma_{R.B\neq 10}(R)$　R2=$\sigma_{R.B\neq 10}(S)$
T(R1)=T(R)-T(R) / V(R,B)=480000
一个块可以存储100个元组，所以B(R1)=4800
求解R1的I/O代价：c1=5000+4800=9800　（R为连续存储）

R1放到G1里

同理：T(R2)=96000, B(R2)=960
求解R2的I/O代价：c2=100000+960=100960　（S为不连续存储）

设R1平均一个桶中块数为x, 则G1中块数为x，内存中还剩（147-x）个块用于桶代理。（一个块用于输入缓冲）
则总的桶数为147-x+1=148-x。（1为G1的桶）
因此有(148-x)*x>=4800，解得 48 <= x <= 100
因此选择x=100，即总的桶数为148-100=48

所以，确定桶数为48，R1的一个桶的块数为4800/48=100
则R2的一个桶中块数=B(R2)/48=960/48=20。

因为G1和H1的每一块都节省了2次I/O，所以
散列连接代价：c3=3*[B(R1)+B(R2)]-(100+20)*2=17040

总代价：总的I/O代价=c1+c2+c3=127800

设R1=$\sigma_{R.B\neq 10}(R)$　R2=$\sigma_{R.B\neq 10}(S)$
T(R1)=T(R)-T(R) / V(R,B)=480000
一个块可以存储100个元组，所以B(R1)=4800
求解R1的I/O代价：c1=5000+4800=9800　（R为连续存储）

同理：T(R2)=96000, B(R2)=960
求解R2的I/O代价：c2=100000+960=100960　（S为不连续存储）

设R2平均一个桶中块数为x,
因此有(148-x)*x>=960，解得 7 <= x <= 141
因此选择x=141，即总的桶数为148-141=7

所以，确定桶数为7，R2的一个桶的块数为960/7=138
R1的一个桶中块数=B(R1)/7=4800/7=686。

因为G1和H1的每一块都节省了2次I/O，所以
散列连接代价：c3=3*[B(R1)+B(R2)]-(138+686)*2=15632

总代价：总的I/O代价=c1+c2+c3=126392

R2放到G1里

3、我们在课本上讨论的归并排序算法是一个两趟算法。设两个连接关系为 **R1** 和 **R2**，在基于两趟归并排序的排序连接算法中，我们要求内存 **M** 必须满足条件 $M \geq \max\left\{\sqrt{R1}, \sqrt{R2}\right\}$。现在我们考查关系 R 的两趟归并排序算法，我们发现当内存 **M** 不满足条件 $M \geq \sqrt{R}$ 时，我们仍可以采用一种多趟算法来完成归并排序操作。请用伪码给出这一多趟归并连接算法的简要描述，并估计当 **T(R1)**=500000，**T(R2)**=100000，**S(R1)= S(R2)**=1/10 **block**， **M** = 100 时该算法的 I/O 代价，这里我们假设 **R1** 和 **R2** 都不是连续存放的。

MergeSort：

(1)给定关系R，内存容量M。将关系R分成大小为M的N个组，最后一个组若大小不足M则正常保留其剩余部分。

(2)将N个组分别读入到内存中进行排序，排序的结果chunk 1，chunk 2，...，chunk N写入到磁盘中。

(3)对上述排好序的N个chunk再分成大小为M的x个组，重复上述(1)(2)操作得到排好序的X个更大的chunk

(4)重复以上操作，直至最后只剩下一块，即关系R中所有数据已归并排序到一个chunk中，排序完成。

R1第一趟：500000+50000=550000
分成500组，每组100个块有序
第二趟：50000*2=100000
分成5组，每组10000个块有序
第三趟：50000*2=100000
R1排序代价：550000+100000+100000=750000

R2第一趟：100000+10000=110000
分成100组，每组100个块有序
第二趟：10000*2=20000
一共10000个块有序
R2排序代价：110000+20000=130000

总代价：排序+连接=750000+130000+50000+10000=940000

习题课

hw2、hw5

1. 假设磁盘块大小为 8 KB，块中存储 200 字节的定长记录，块首部只包括一个 8 字节的模式指针和一个偏移量表。对于插入块内的每条记录，在偏移量表中都增加一个 2 字节的指针指向该记录。假设每天向块内插入 4 条记录（空间不足时允许插入部分记录后结束全部操作），删除 2 条记录。假设每天的删除记录操作总是发生在插入记录之前，删除记录使用一个“删除标记”代替记录在偏移量表中的指针。给定一个磁盘块，如果刚开始块是空的，则几天后不能再向该块内插入记录？此时，该块内一共有多少条记录？

解：第一天，插入4条记录，占的空间大小为(200+2)×2 = 808 B

此后每天删除2条记录，释放的空间大小为 2×200 = 400 B

所以每天增加所占的空间大小为 -400+808 = 408 B

设第x天，令808+408(x-1) <= 8192，可求，x <= 19.1

取整数x = 19，故当第19天插入记录后，所占磁盘空间为8+808+408×18 = 8160 B

第20天开始时，还剩下32B空间，删除2条记录后，余下432B空间

然后插入2条记录，剩余28B空间，插入本日第3条记录时，只插入了部分数据然后结束全部操作

因此，第20天后不能插入记录，此时磁盘块内一共有4+2×18 = 40条有效记录（此外还有1条无效记录）

2、假设我们采用 LRU 作为缓冲区置换策略，当我们向 Buffer Manager 发出一个读页请求时，请讨论一下：

（1）如果页不在缓冲区中，我们需要从磁盘中读入该页。请问如何才能在缓冲区不满的时候快速地返回一个 free 的 frame？请给出至少两种策略，并分析一下各自的时间复杂度。

（2）如何才能快速地判断所请求的页是否在缓冲区中？如果请求的页在缓冲区中，如何快速返回该页对应的 frame 地址？请给出至少两种策略，并分析一下各自的时间复杂度。

解：(1) a.将所有的空闲的frame id插入到一个链表中，每次从链表头部返回一个空闲frame id，时间复杂度为O(1)

b.使用位图，位图的每一位表示一个frame的空闲/占用情况（例如1表示被占用，0表示空闲），需要寻找一个free的frame时，直接扫描位图，找到位图中为0的位置，进而得出其frame id。时间复杂度为O(n)

(2) a.将在缓冲区的的页的page id以及所在的frame的id以键值对的形式(page id为key，frame id为value)存储在一个哈希表中。可以在O(1)的时间复杂度下判断所请求的页是否在缓冲区中，若在缓冲区中可以找到键值对，进而返回该页对应的frame地址

b.将在缓冲区的的页的page id以及所在的frame的id以键值对的形式(page id为key，frame id为value)存储在一棵B+-tree中，可以在O(log n)的时间复杂度下判断所请求的页是否在缓冲区中，若在缓冲区中可以找到键值对，进而返回该页对应的frame地址

3、我们在讲义上介绍了 SSD 感知的 CF-LRU 算法，即 Clean-First LRU 算法。该算法虽然看起来可以减少对 SSD 的写操作，但依然存在一些问题。请分析一下该算法的主要缺点有哪些？给出三点，并简要解释你的理由。

解：CF-LRU算法的基本思想是：把LRU链表分为工作区和替换区，工作区负责维护最近访问的数据页，替换区则负责维护替换候选队列，替换时总是优先替换替换区中的干净页，若替换区没有干净页，则选择LRU链表尾部的第一个脏页作为置换页。在CF-LRU算法中替换区的大小是由窗口大小决定的。CF-LRU通过优先替换出替换区的干净页，在一定程度上可以有效地减少对闪存的写和擦除操作，提升了性能，但还存在一些不足：

(1)很难确定一个合适窗口大小的值来适应不同类型的负载。

(2)当链表较长时，查找干净页作为置换页的代价会较高。由于算法在选择置换页时需要沿着链表反向查找干净页，当链表较长时查找代价会增加。

(3)没有考虑缓冲区页的访问频率，在进行替换操作时，容易保留较老的脏页，而替换热的干净页，这会导致缓冲区命中率的降低。

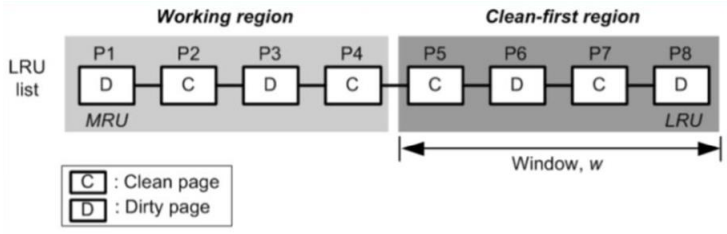


Figure 1. Example of CFLRU algorithm

1. 什么是事务的 ACID 性质？请给出违背事务 ACID 性质的具体例子，每个性质举一个例子。

解:

- **原子性Atomicity**
  转账 A=A+100; ~~B=B-100;~~

- **一致性Consistency**
  转账时如果只执行了A+100，则一致性约束A＋B不变被破坏。事务一致性被破坏

- **隔离性Isolation**
  在取款机上查询余额，第一次查询和第二次查询显示余额不一样，说明在两次查询中数据受到了其它事务的影响，此时事务没有完全隔离

- **持久性 Durability**
  事务已commit，而改变的数据仍然还在磁盘缓冲区中排队等待写入磁盘中，
  导致系统重启后commit的结果无效
  （注意：回滚是正常操作，不是违背durable的例子。重点是回答到未保存至磁盘等持久化介质，或仅保存至易失内存上。）

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（至少写出 2 点以上并且要给出自己的分析）？

解：(假设A事务调用了B事务)

(1) 此时如果B事务提交了，但A调用B返回后事务A选择Rollback，此时由于B已经提交，持久性已经生效，A的Rollback已经不能将事务B撤销从而导致事务A的原子性和一致性被破坏。

(2) 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。

(3) 可能出现互相调用陷入死循环。

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。

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

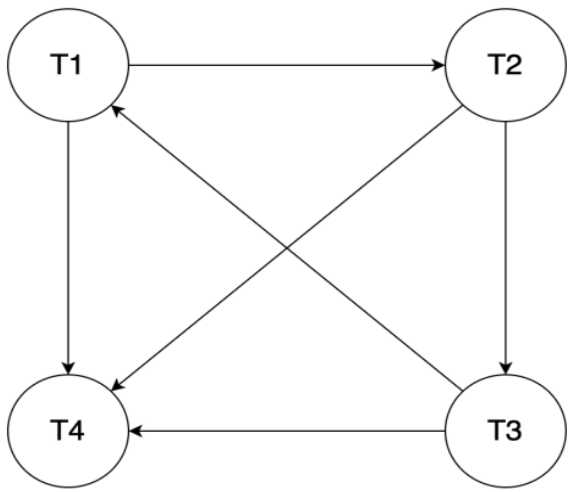
undo：F=100, G=20, F=20

所以，A=75, B=250, C=65, D=46, E=55, F=20, G=20

4. 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)

根据并发调度作出其所对应的优先图如下：



可以看到该优先图中存在环路 T1 → T2 → T3 → T1，因此该并发调度不是冲突可串。

习题课
(HW 3, 6)

HW3

1. 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）？

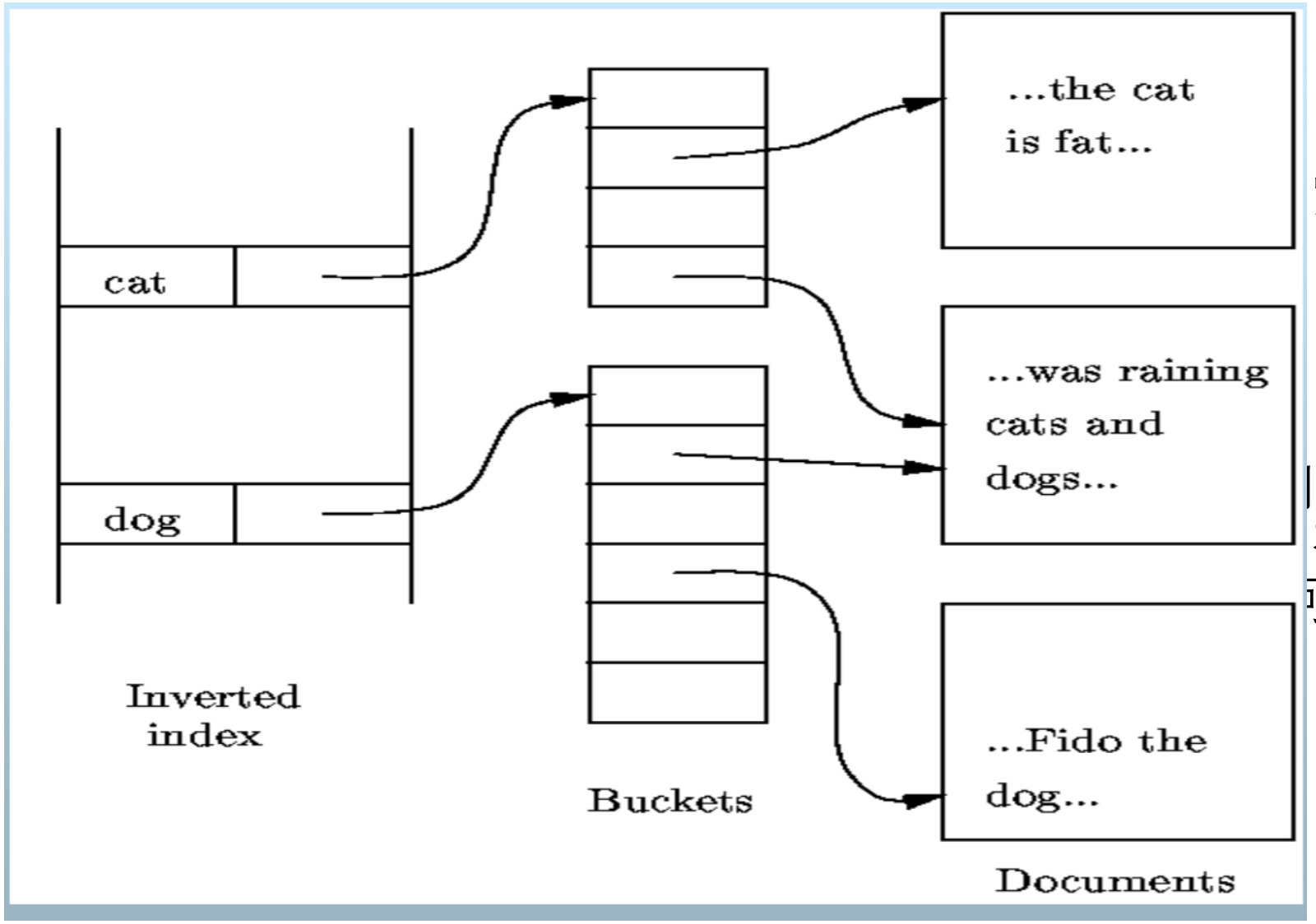
不能。文献查询并不是精确(whole-key)点查询，而是范围(partial-key)查询，查询的单词可能出现在title的任意位置。由于B+树是根据键的字典序排序的，仅支持前缀查询，不支持中缀和后缀范围查询。因此使用B+树结构仍要扫描所有数据项。

HW3

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。

HW3



因此上述文献查
索引技术来进一步
简要说明该索引

的所有文档。相比于
存储 (b).对于文献查
可完成搜索。

HW3

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。

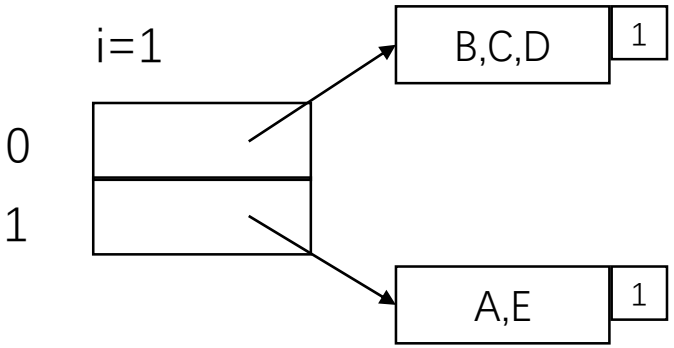
- 插入：对文档的title进行分词，然后将各个关键词插入到索引中
  - 查询关键词是否已经存在，若存在则直接将指针插入对应的桶中
  - 若不存在，则插入关键词，增加对应的桶以及指向文档的指针
- 查询：
  - 单个关键词；返回得到的文档集合即可
  - 多个关键词；返回各个关键词查询结果的文档集合的交集

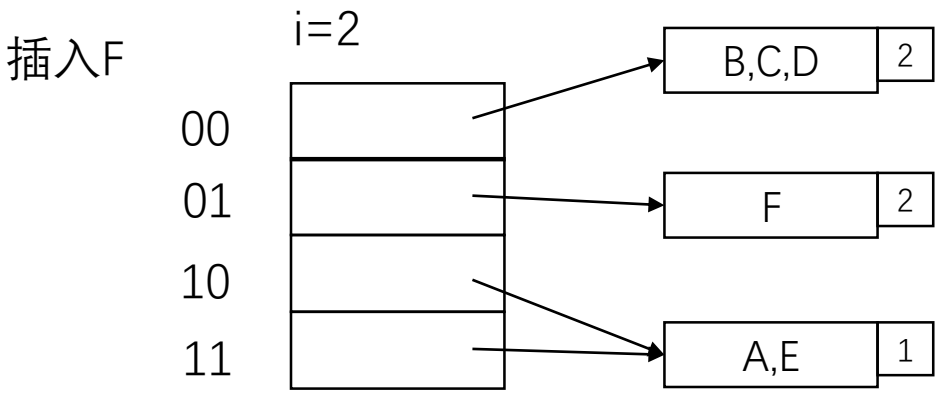
2. 假设有如下的键值，现用 5 位二进制序列来表示每个键值的 hash 值。回答问题：

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。

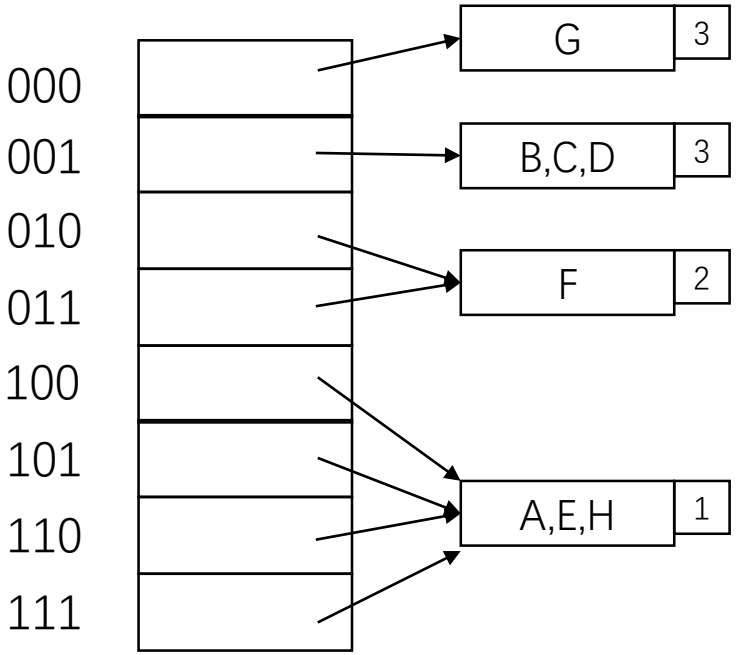




A [11001] B [00111] C [00101] D [00110] E [10100] F [01000] G [00011]

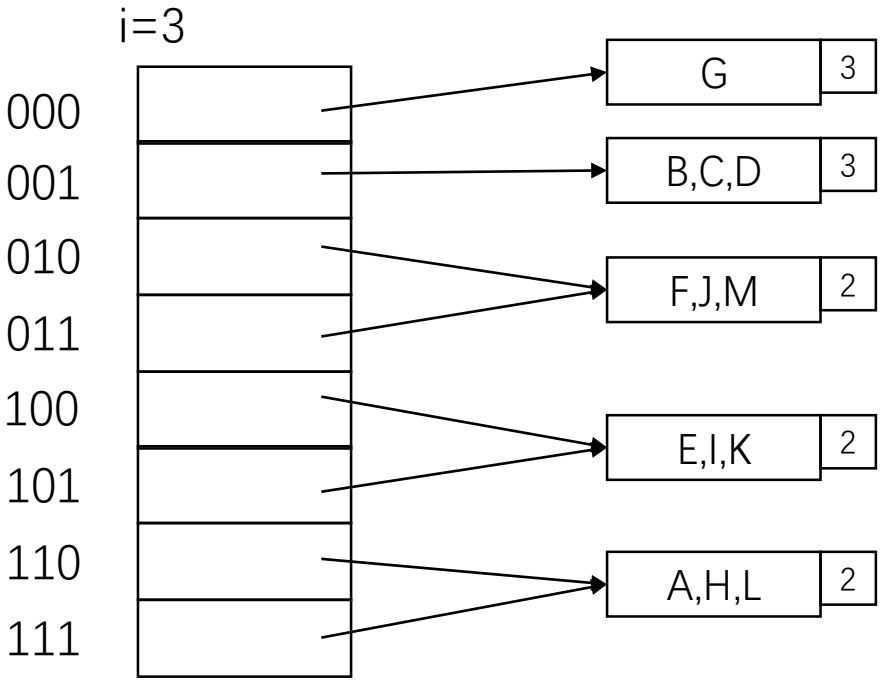
H [11110] I [10001] J [01101] K [10101] L [11100] M [01100] N [11111]

插入G i=3

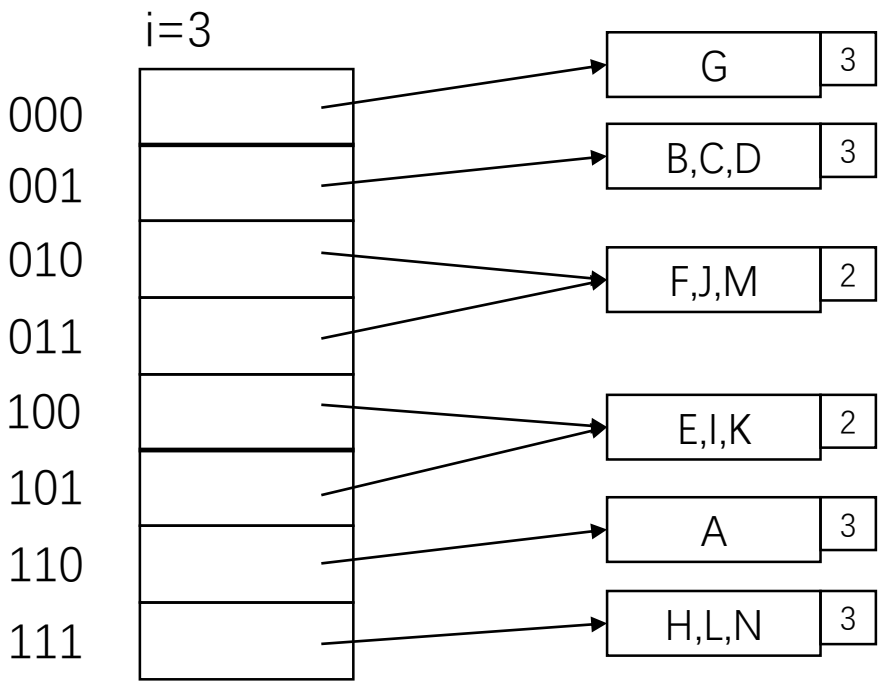


6个桶，E、I、K

插入I i=3



插入N i=3



A [11001]　B [00111]　C [00101]　D [00110]　E [10100]　F [01000]　G [00011]　值

后

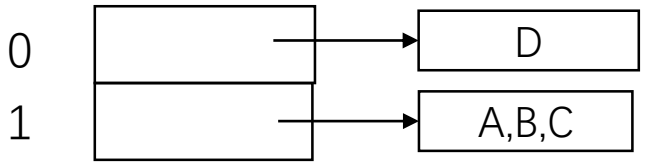
桶　H [11110]　I [10001]　J [01101]　K [10101]　L [11100]　M [01100]　N [11111]　l个

桶数＝n，h(k)的右i位＝m

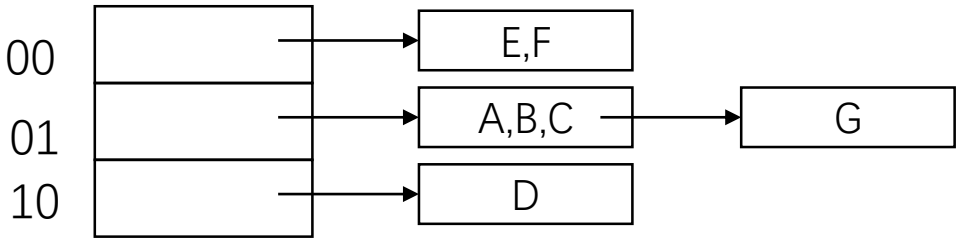
若m<n，则记录位于第m个桶

若$n \le m < 2^i$,则记录位于第 $m-2^{i-1}$ 个桶

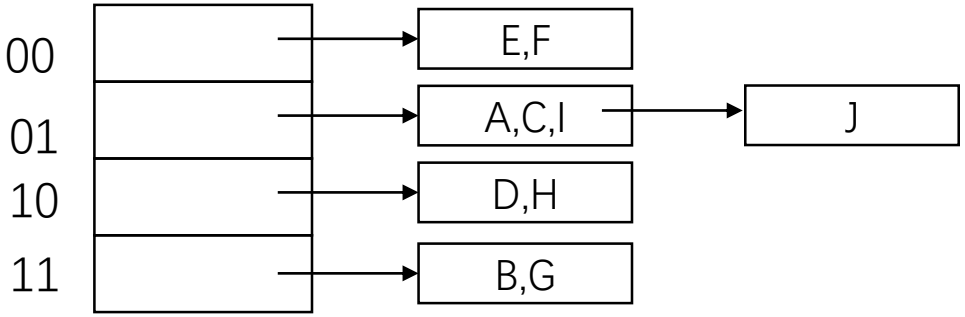
i=1



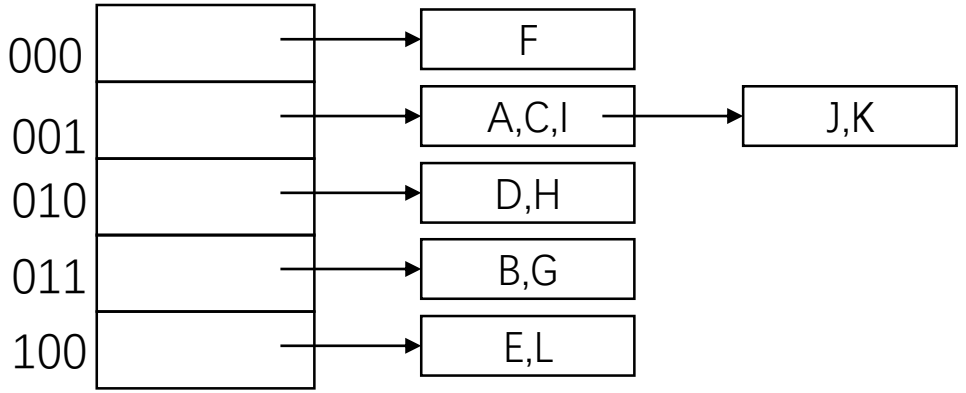
插入E　i=2　5/6>0.8



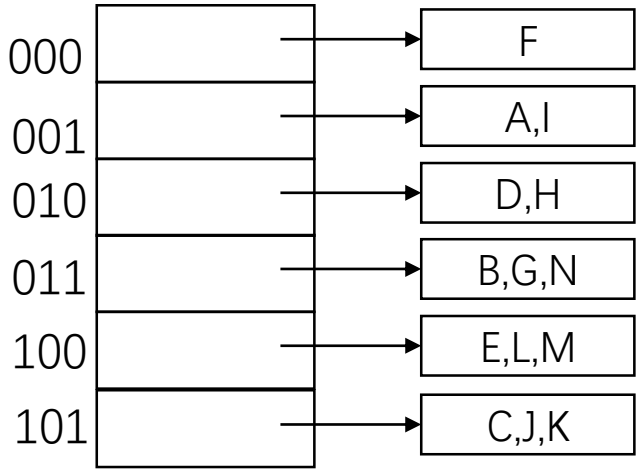
插入H　i=2　8/9>0.8



插入K　i=3　10/12>0.8



插入M　i=3　13/15>0.8



6个桶，B、G、N

3、对于 B+树，假设有以下的参数：

参数	含义	参数	含义
N	记录数	*S*	读取一个磁盘块时的寻道时间
n	B+树的阶，即节点能容纳的键数	*T*	读取一个磁盘块时的传输时间
R	读取一个磁盘块时的旋转延迟	*m*	在内存的 *m* 条记录中查找 1 条记录的时间（线性查找）

假设所有磁盘块都不在内存中。现在我们考虑一种压缩 B+树，即对 B+树的节点键值进行压缩存储。假设每个节点中的键值压缩 1 倍，即每个节点在满的情况下可压缩存储 2*n* 个压缩前的键值和 2*n*+1 个指针。额外代价是记录读入内存后必须解压，设每个压缩键值的内存解压时间为 *c*。给定 N 条记录，现使用压缩 B+树进行索引，请问在一棵满的 *n* 阶压缩 B+树中查找给定记录地址的时间是多少？（使用表格中的参数表示，*n*+1 或 *n*–1 可近似表示为 *n*）？

只有叶子节点存放数据，假设层数为k，故有$(2n+1)^{k-1} \times 2n = N$，可得$k \approx \log_{2n} N$

时间：$(R + S + T + 2nc + 2n) \times \log_{2n} N$

HW6

1. 下表给出了 3 个事务的一个调度序列，请回答问题：

time	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}
T_1			R(A)	W(A)			R(C)	W(C)				
T_2					R(B)	W(B)						
T_3	R(A)	W(A)							R(C)	W(C)	R(B)	W(B)

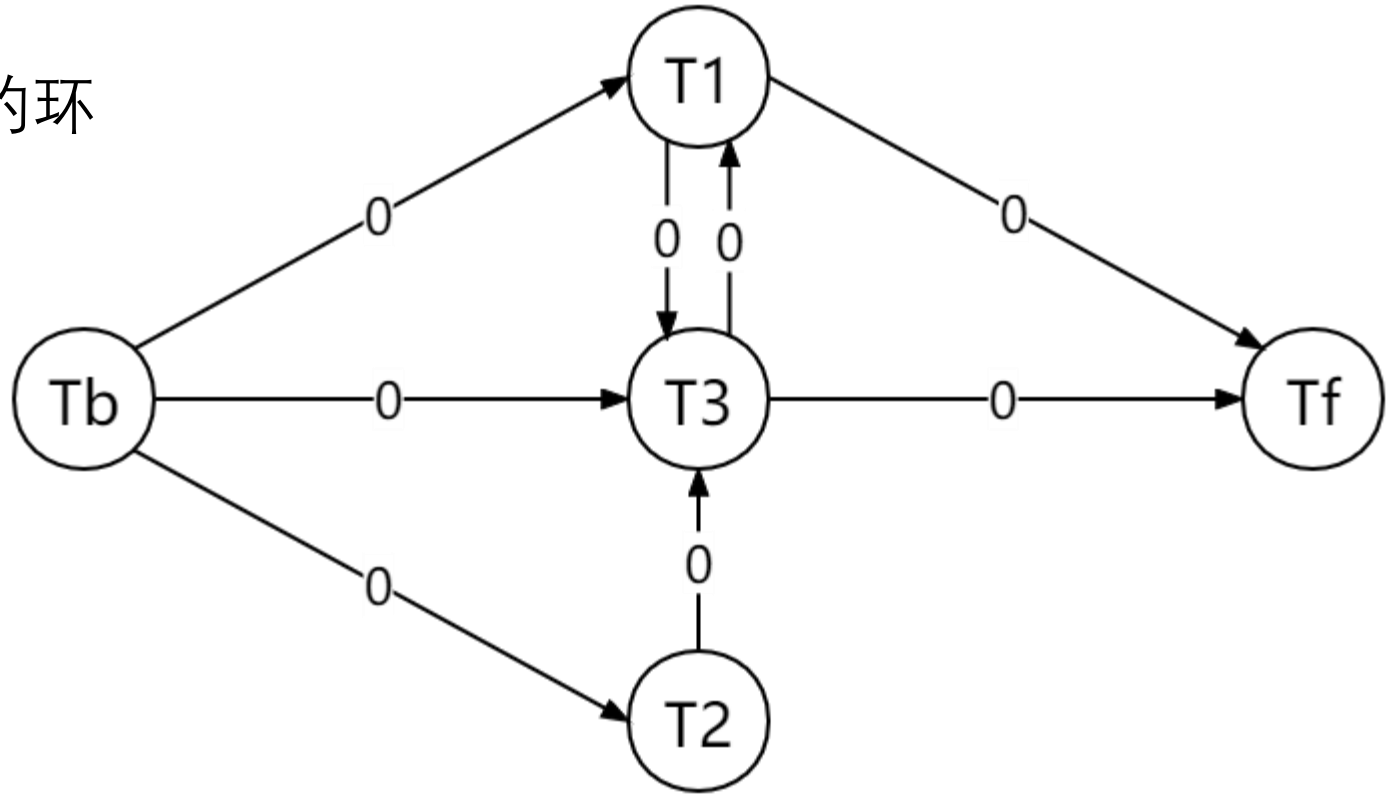
画出该调度的多重图，并判断该调度是否视图可串。如果回答“是”，请给出视图等价的串行调度序列；如果回答“不是”，请解释理由。

HW6

T1和T3之间有由“0”线构成的环

无法通过删除“p”线消除环

➔不是视图可串



HW6

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)

1）T1，T2并发执行能保证数据库的一致性吗？为什么?

能。因为这是两阶段锁，而采用两阶段锁的调度一定是可串化调度（为什么？），因此一定能保证一致性。

***死锁由系统处理，不在事务的考虑范围内!

HW6

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)

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？

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。

言之有理即可

HW6

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)

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?

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。

Thanks !