

数据库课后习题复习 三

十、第十章

10.4

考虑从图 10-6 的文件中删除记录 5。比较下列实现删除的技术的相对优点：

- a. 移动记录 6 到记录 5 所占用的空间，然后移动记录 7 到记录 6 所占用的空间。
- b. 移动记录 7 到记录 5 所占用的空间。
- c. 标记记录 5 被删除，不移动任何记录。

记录0	10101	Srinivasan	Comp. Sci.	65000
记录1	12121	Wu	Finance	90000
记录2	15151	Mozart	Music	40000
记录11	98345	Kim	Elec. Eng.	80000
记录4	32343	El Said	History	60000
记录5	33456	Gold	Physics	87000
记录6	45565	Katz	Comp. Sci.	75000
记录7	58383	Calmeri	History	62000
记录8	76543	Singh	Finance	80000
记录9	76766	Crick	Biology	72000
记录10	83821	Brandt	Comp. Sci.	92000

图 10-6 图 10-5 中的文件，删除了记录 3
并且移动最后一条记录

- a.是最直接的方式，但是需要移动最多的记录并且需要做最多的访问
- b.可以移动更少的，但是会破坏原本文件的顺序
- c.不用移动记录，但是需要额外的开销来跟踪记录文件中的可用空间，这样会造成文件中由太多的孔，如果不合并压缩，会很影响效率。

给出经过下面每一步后图 10-7 中文件的结构：

- 插入(24556, Turnamian, Finance, 9800)。
- 删除记录 2。
- 插入(34556, Thompson, Music, 67000)。

头文件				
记录0	10101	Srinivasan	Comp. Sci.	65000
记录1				
记录2	15151	Mozart	Music	40000
记录3	22222	Einstein	Physics	95000
记录4				
记录5	33456	Gold	Physics	87000
记录6				
记录7	58583	Califieri	History	62000
记录8	76543	Singh	Finance	80000
记录9	76766	Crick	Biology	72000
记录10	83821	Brandt	Comp. Sci.	92000
记录11	98345	Kim	Elec. Eng.	80000




图 10-7 删除了第 1、4 和 6 条记录的图 10-4 中的文件

- a. The file after **insert** (24556, Turnamian, Finance, 98000).

header				↑ 4
record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	24556	Turnamian	Finance	98000
record 2	15151	Mozart	Music	40000
record 3	22222	Einstein	Physics	95000
record 4				↑ 6
record 5	33456	Gold	Physics	87000
record 6				
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

- b.

header				↑ 2
record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	24556	Turnamian	Finance	98000
record 2				↑ 4
record 3	22222	Einstein	Physics	95000
record 4				↑ 6
record 5	33456	Gold	Physics	87000
record 6				
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

The free record chain could have alternatively been from the header to 4, from 4 to 2, and finally from 2 to 6.

c.

header	↑ 4			
record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	24556	Turnamian	Finance	98000
record 2	34556	Thompson	Music	67000
record 3	22222	Einstein	Physics	95000
record 4				↑ 6
record 5	33456	Gold	Physics	87000
record 6				
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

10.6

考虑关系 *section* 和 *takes*。给出这两个关系的一个实例，包括 3 次开课，每次开课有 5 个学生选课。给出一个使用多表聚簇的这些关系的文件结构。

三次开课后的 *section* 中的元组是这样的：

course_id	sec_id	semester	year	building	room_number	time_slot_id
BIO-301	1	Summer	2010	Painter	514	A
CS-101	1	Fall	2009	Packard	101	H
CS-347	1	Fall	2009	Taylor	3128	C

每次开课有5个学生选课后的元组是这样的：

ID	course_id	sec_id	semester	year	grade
00128	CS-101	1	Fall	2009	A
00128	CS-347	1	Fall	2009	A-
12345	CS-347	1	Fall	2009	A
12345	CS-101	1	Fall	2009	C
17968	BIO-301	1	Summer	2010	null
23856	CS-347	1	Fall	2009	A
45678	CS-101	1	Fall	2009	F
54321	CS-101	1	Fall	2009	A-
54321	CS-347	1	Fall	2009	A
59762	BIO-301	1	Summer	2010	null
76543	CS-101	1	Fall	2009	A
76543	CS-347	1	Fall	2009	A
78546	BIO-301	1	Summer	2010	null
89729	BIO-301	1	Summer	2010	null
98988	BIO-301	1	Summer	2010	null

多表聚簇：

BIO-301	1	Summer	2010	Painter	514	A
17968	BIO-301	1	Summer	2010	null	
59762	BIO-301	1	Summer	2010	null	
78546	BIO-301	1	Summer	2010	null	
89729	BIO-301	1	Summer	2010	null	
98988	BIO-301	1	Summer	2010	null	
CS-101	1	Fall	2009	Packard	101	H
00128	CS-101	1	Fall	2009	A	
12345	CS-101	1	Fall	2009	C	
45678	CS-101	1	Fall	2009	F	
54321	CS-101	1	Fall	2009	A-	
76543	CS-101	1	Fall	2009	A	
CS-347	1	Fall	2009	Taylor	3128	C
00128	CS-347	1	Fall	2009	A-	
12345	CS-347	1	Fall	2009	A	
23856	CS-347	1	Fall	2009	A	
54321	CS-347	1	Fall	2009	A	
76543	CS-347	1	Fall	2009	A	

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. 一个文件中存储一个关系
- b. 一个文件中存储多个关系

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？

十一、第十一章

11.1

索引加速了查询处理，但是在作为潜在的搜索码的每一个属性上或者每一个属性组上创建索引，通常并不是一个很好的方法，请解释为什么。

- 在插入和删除期间，每个索引都需要额外的CPU时间和磁盘I/O开销
- 更新时可能必须更改非主键的索引，尽管主键上的索引可能不会（这是因为更新通常不会修改主键属性）。
- 每个额外的索引都需要额外的存储空间。
- 对于涉及多个搜索键条件的查询，有效即使只有一些键具有索引，科学性也可能不错在他们。因此，通过添加数据库，数据库性能的改善较少已经存在许多索引的索引。

11.2 在一个关系的不同搜索码上建立两个主索引一般来说是否可能？

通常，不可能有两个主要索引不同键的关系相同，因为关系中的元组会必须以不同的顺序存储以将相同的值存储在一起。我们可以通过将关系存储两次并复制全部来完成此操作值，但是对于集中式系统，这效率不高。

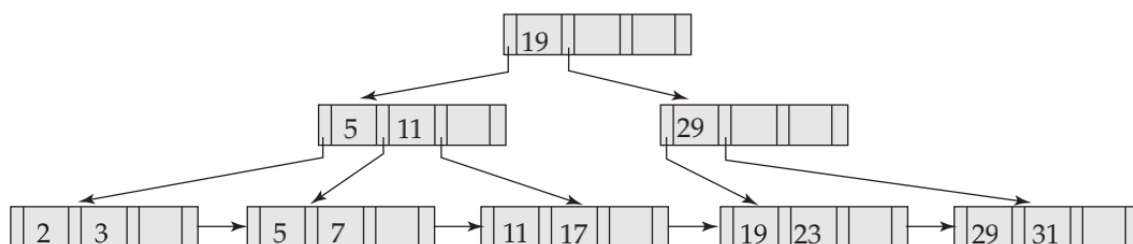
11.3

用下面的关键码值集合建立一个 B⁺ 树

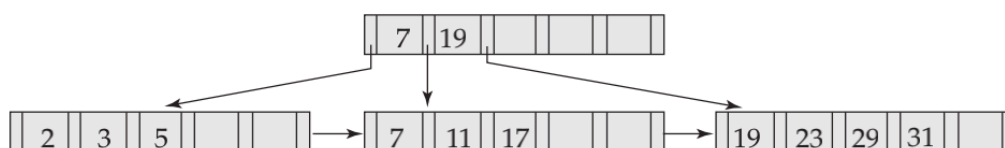
(2, 3, 5, 7, 11, 17, 19, 23, 29, 31)

假设树初始为空，值按上升顺序加入。根据一个结点所能容纳指针数的下列情况分别构造 B⁺ 树：

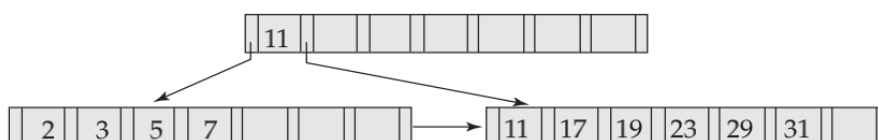
- a. 4
- b. 6
- c. 8



b.



c.



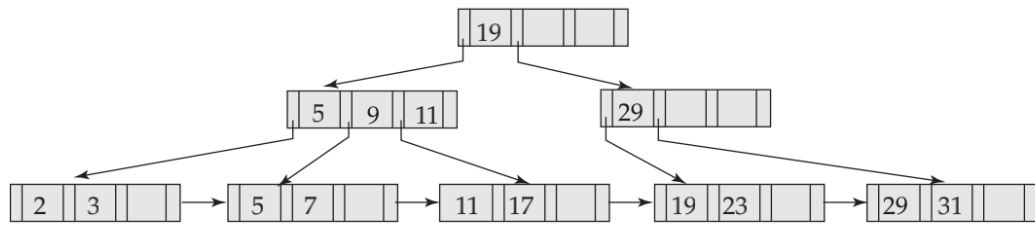
11.4

对习题 11.3 中的每一棵 B⁺ 树，给出下列各操作后树的形状：

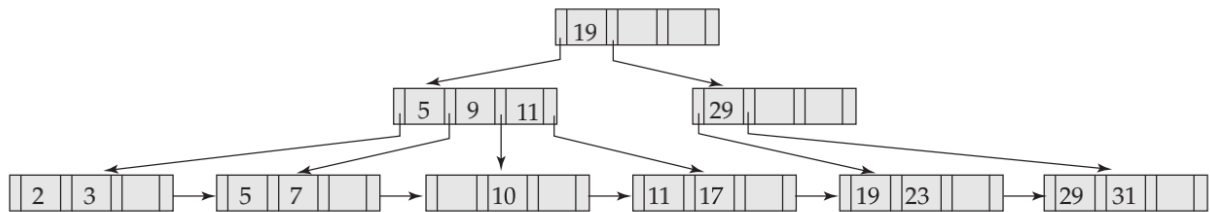
- a. 插入 9
- b. 插入 10
- c. 插入 8
- d. 删除 23
- e. 删除 19

- With structure 11.3.a:

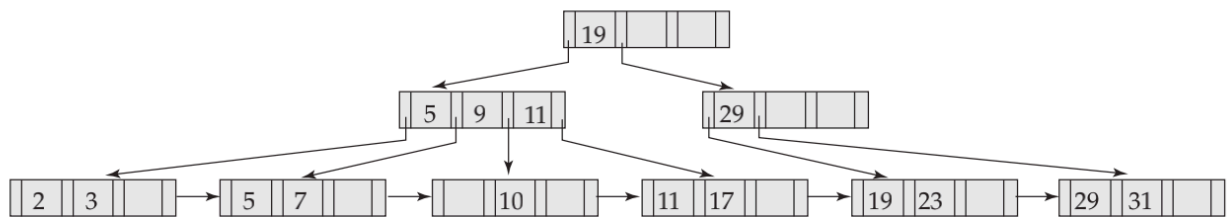
Insert 9:



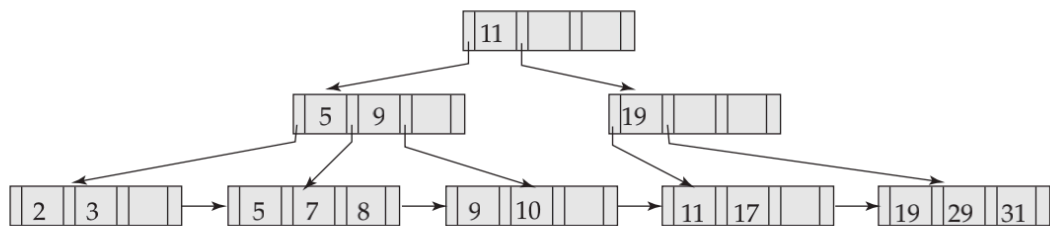
Insert 10:



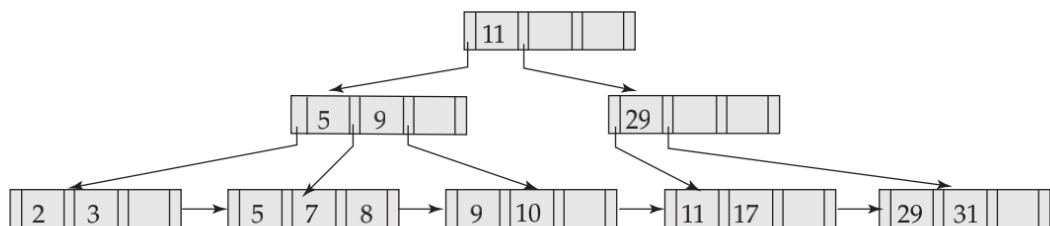
Insert 8:



Delete 23:

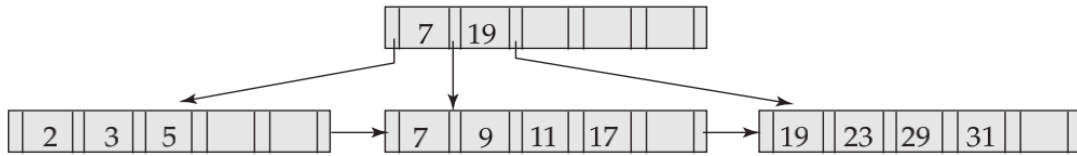


Delete 19:

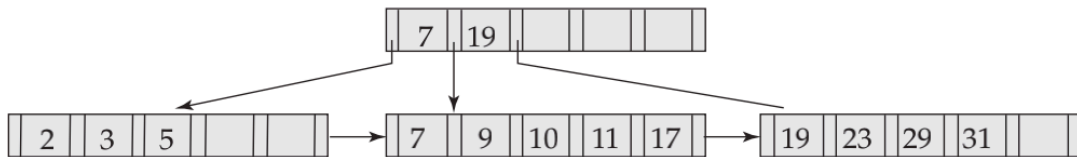


- With structure 11.3.b:

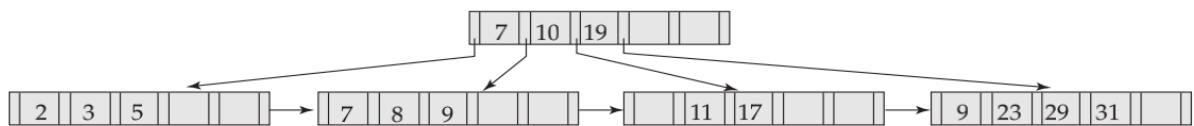
Insert 9:



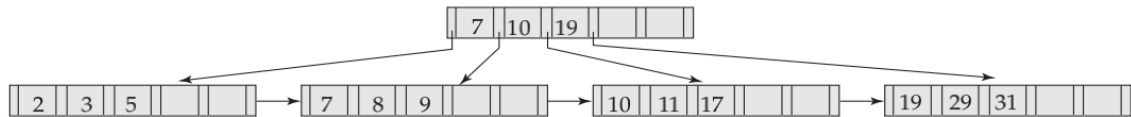
Insert 10:



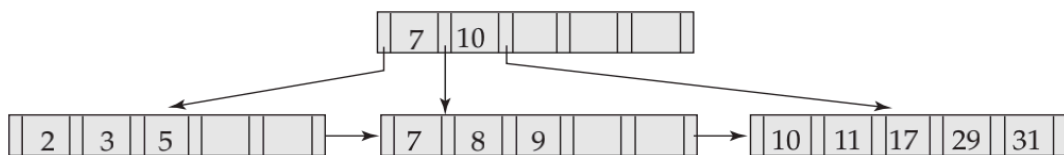
Insert 8:



Delete 23:

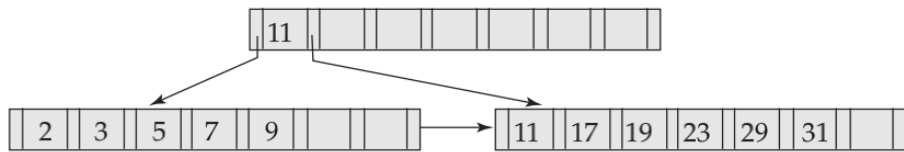


Delete 19:

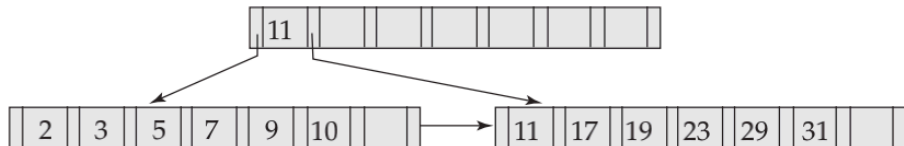


- With structure 11.3.c:

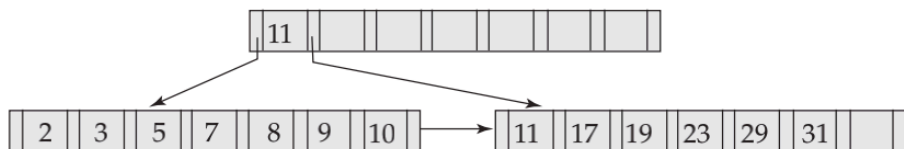
Insert 9:



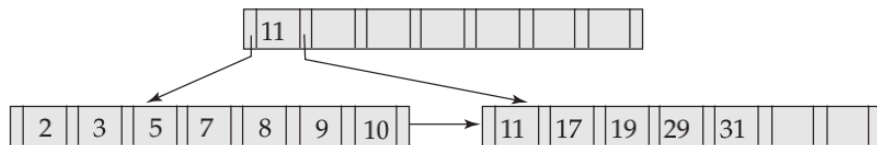
Insert 10:



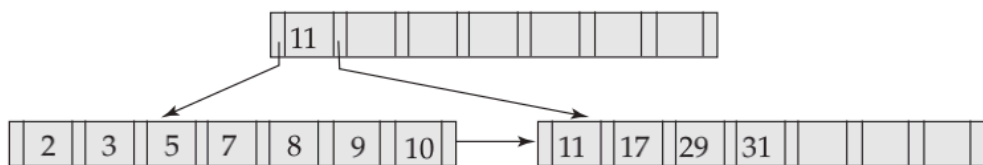
Insert 8:



Delete 23:



Delete 19:

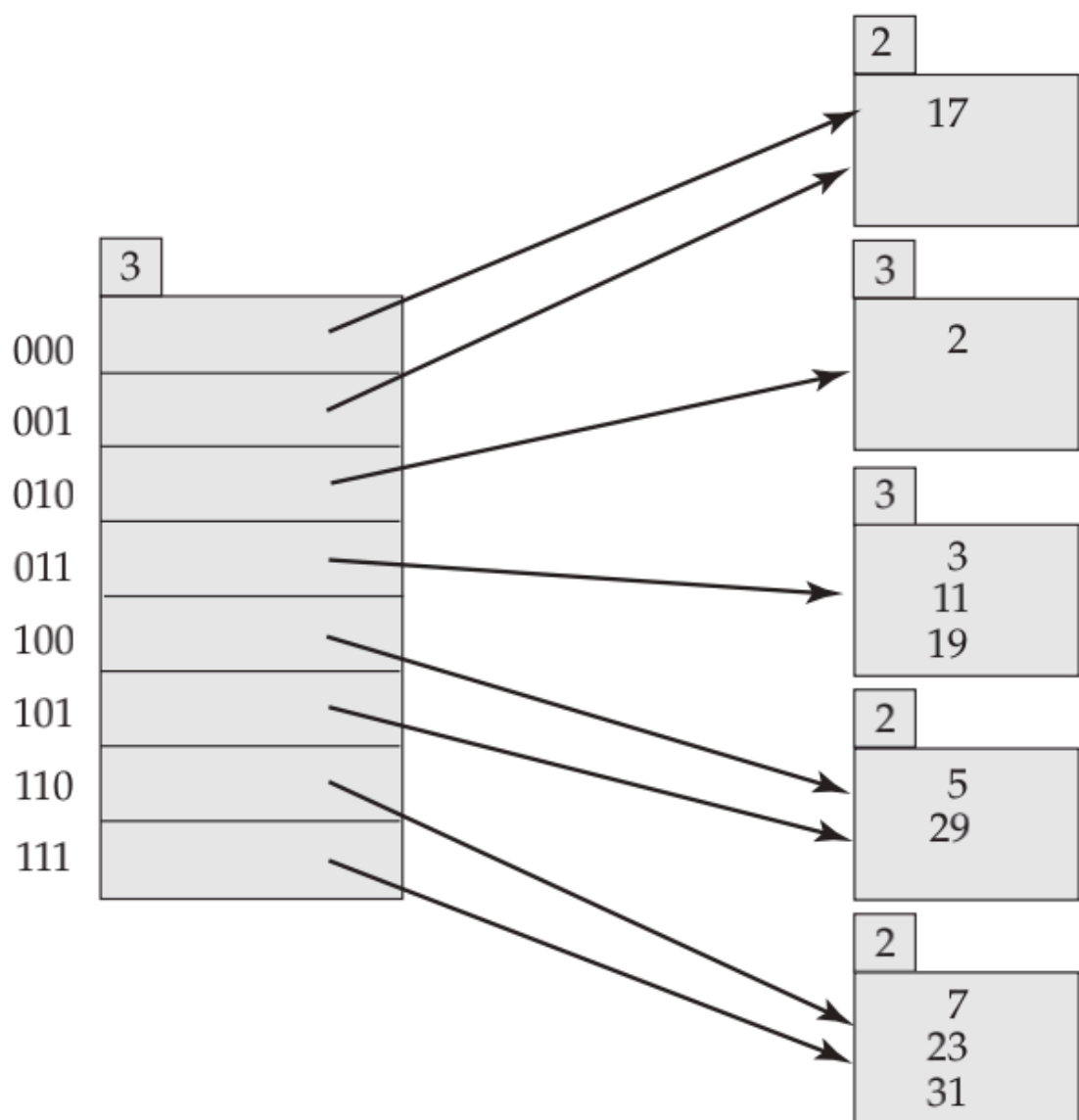


11.6

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 $h(x) = x \bmod 8$ ，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。

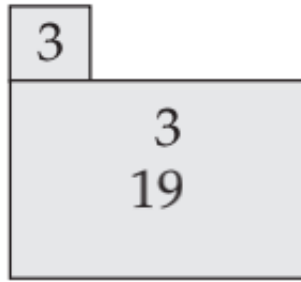


11.7

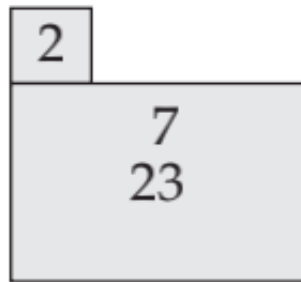
进行下列各步以后，习题 11.6 中的可扩充散列结构如何变化？

- a. 删除 11
- b. 删除 31
- c. 插入 1
- d. 插入 15

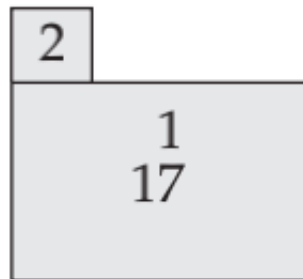
a. 第三个桶中的内容变成了：



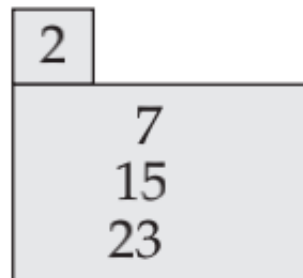
b.最后一个桶的内容变成了：



c.第一个通变成了：



d.最后一个变成了



考虑在图 11-1 中表示的关系 *instructor*。

- 在属性 *salary* 上构建一个位图索引，把 *salary* 的值分成 4 个区间：小于 50000，50000 ~ 60000 以下，60000 ~ 70000 以下，70000 及其以上。
- 考虑一个查询，即在金融系中所有工资为 80 000 美元或更多的教师。假设上述 *salary* 上的位图索引和 *dept_name* 上的位图索引可用概述回答该查询的步骤，并且给出为回答这个查询而构建的中间和最终位图。

ID	<i>name</i>	<i>dept_name</i>	<i>salary</i>
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

- a. Bitmap for *salary*, with S_1 , S_2 , S_3 and S_4 representing the given intervals in the same order

S_1	0	0	1	0	0	0	0	0	0	0	0	0
S_2	0	0	0	0	0	0	0	0	0	0	0	0
S_3	1	0	0	0	1	0	0	1	0	0	0	0
S_4	0	1	0	1	0	1	1	0	1	1	1	1

- b. The question is a bit trivial if there is no bitmap on the *dept_name* attribute. The bitmap for the *dept_name* attribute is:

Comp. Sci	1	0	0	0	0	0	1	0	0	0	1	0
Finance	0	1	0	0	0	0	0	0	1	0	0	0
Music	0	0	1	0	0	0	0	0	0	0	0	0
Physics	0	0	0	1	0	1	0	0	0	0	0	0
History	0	0	0	0	1	0	0	1	0	0	0	0
Biology	0	0	0	0	0	0	0	0	0	1	0	0
Elec. Eng.	0	0	0	0	0	0	0	0	0	0	0	1

如果是要找在经济系的所有工资>80000或者更多的所有老师，我们先找到经济系位图和工资 S_4 的位图的交集，再扫描所有工资超过80000的

交集：

S_4	0	1	0	1	0	1	1	0	1	1	1	1
Finance	0	1	0	0	0	0	0	0	1	0	0	0
$S_4 \cap \text{Finance}$	0	1	0	0	0	0	0	0	1	0	0	0

11.12 如果索引项按照已经拍好的顺序插入，B+树的每个叶结点的充满情况如何？解释为什么？

如果索引条目以升序插入，则新条目将定向到最后一个叶子节点。当这个叶子节点被填满时，它分为两部分。在拆分生成的两个节点中，左节点保持不变，插入发生在右侧节点上。这个使叶子节点的占用率达到50%（最后一个除外）叶。如果插入的键以降序排列，则出现上述情况仍然会发生，但是对称地，分割的正确节点永远不会再次受到感动，所有客房的入住率将再次达到50%第一个叶子以外的节点。

11.13

假设你有一个包括 n_r 条元组的关系 r ，需要在上面建立一个辅助 B^+ 树索引。

- 通过一次插入一条记录来建立 B^+ 树索引的代价是多少？给出计算公式。假设平均每个页面存储 f 条索引项，并且所有高于叶级的结点都在内存中。
- 假设一次随机磁盘访问的时间是 10 毫秒，在一个包含 1 000 万条记录的关系上建立索引的代价是多少？
- 请写出如 11.4.4 节描述的自底向上构建 B^+ 树的伪代码。你可以假设存在一个可以有效排序大文件的函数。

a. 由于非叶节点在内存中，所以为了插入而查找所需叶子页的页码的成本可以忽略。在叶子级别上，需要一个随机磁盘访问才能读取以及另一个随机访问磁盘以更新它以及写一个磁盘的成本页。导致叶节点分裂的插入需要一个

附加页面写入。

因此，用 nr 项构建 B^+ 树，最多需要 $2 * nr$ 个随机磁盘访问，以及 $nr + 2 * (nr / f)$ 次页面写入。成本的第二部分来自以下事实：

最坏的情况是每片叶子都充满了一半，因此分割的数量发生两次是 nr / f 。

上式忽略了编写非叶节点的开销，因为我们假设它们在记忆中，但实际上它们也将最终写。该成本近似为 $2 * (nr / f) / f$ ，这是叶子正上方的内部节点数；我们可以添加更多术语以说明更高级别的节点，但这是比叶数小得多，可以忽略。

b. 将上述公式中的值代入而忽略成本对于页面写入，大约需要 $10,000,000 * 20$ 毫秒或 56 小时，因为每次插入要花费 20 毫秒

11.14

为什么 B^+ 树文件组织的叶结点可能会丢失顺序性？

- 请建议文件组织该如何重新组织来恢复顺序性。
- 对于一个相当大的 n ，另一种重新组织的方法是以 n 块为单位重新分配叶子页。当分配 B^+ 树的第一个叶子时， n 块单元中只有一块被使用，剩下的页是空闲的。如果有一页分裂，并且它的 n 块单元有空闲页，则新的页可以使用该空间。如果 n 块单元已经存满了，分配另一个 n 块单元，并且前 $n/2$ 叶子页被放入第一个 n 块单元，剩余的放入第二个 n 块单元中。简单起见，假设没有删除操作。
 - 假设没有删除操作，当第一个 n 块单元存满时，已分配空间的充满度的最坏情况是什么？
 - 有没有可能出现这样的情况：分配给一个 n 结点块单元的叶子结点是不连贯的，也就是说有可能有两个叶子结点分配到同一个 n 结点块中，但是在它们之间的另一个叶子结点分配到了另一个不同的 n 结点块中？
 - 在缓冲区对于存储一个 n 页的块来说是足够的这一合理假设下， B^+ 树的页级扫描在最坏情况下需要多少次寻道？请把该数字和如果每次只为叶子页分配一个块的最坏情况进行比较。
 - 为了提高空间利用率而将值重新分配给兄弟结点的技术如果与上述用于叶子结点的分配策略结合使用可能会更有效。请解释为什么。

在 B^+ 树索引或文件组织中，叶节点是树中彼此相邻的位置，但可能位于磁盘上的不同位置。当在一组记录上新创建文件组织时，可以将磁盘上最连续的块分配给树中连续的叶子节点。随着在树上发生插入和删除操作，顺序丢失越来越多，顺序访问不得不越来越频繁地等待磁盘搜索。

a. 解决这个问题一个方法是重建索引以恢复序列性。

b.

- 在最坏的情况下，每个 n 块单元和 B 树的每个节点都被填充一半。这使得最坏的情况下占用率为 25%。

2. 不。在分割 n 块单元时，第一个 $n/2$ 页被放置在一个 n 块单元中，其余的放置在第二个 n 块单元中。也就是说，每个 n 块分割都保持顺序。因此，那个 n 块单元中的节点是连续的。
3. 在常规的B树结构中，叶页可能不是顺序的，因此在最坏的情况下，每页需要一个搜索。使用块在一个时间方法，对于每个 n 节点块，我们在里面至少有 $n/2$ 叶节点。每个node块都可以使用一个search读取。因此，最坏的情况下寻求以系数 $n/2$ 下降。
4. 允许在同一块的节点之间重新分配不需要额外的搜寻，而在常规B+树中，需要与涉及的叶子页数一样多的搜索在重新分配。这使得叶块可以重新分配此方案非常有效。还有最坏的情况回到近50%。（最好分离叶节点当参与的叶子节点几乎已满时。因此几乎50%，而不是精确的50%）

11.15 什么情况下使用稠密索引比使用稀疏索引更可取？

11.16 聚集索引和辅助索引有什么区别？

11.17

对习题 11.3 中的每一棵 B⁺ 树，给出下列查询中涉及的步骤：

- a. 找出搜索码值为 11 的记录。**
- b. 找出搜索码值在 7 ~ 17 之间(包括 7 和 17)的记录。**

11.19 解释闭地址和开地址的区别，讨论相对优点？

11.20 在散列文件组织中导致桶溢出的原因是什么？如何减少桶溢出的发生？

11.21 为什么对于一个可能在其上进行范围查询的搜索码而言散列结构不是最佳选择？

11.22

假设有一个关系 $r(A, B, C)$ ，带有一个搜索码为 (A, B) 的 B⁺ 树索引。

- a. 用这个索引查找满足 $10 < A < 50$ 条件的记录，最坏情况下的代价是多少？用获取的记录数目 n_1 和树的高度 h 来度量。
- b. 用这个索引查找满足 $10 < A < 50 \wedge 5 < B < 10$ 条件的记录，最坏情况下的代价是多少？用满足条件的记录数目 n_2 和上面定义的 n_1 和 h 来度量。
- c. 当 n_1 和 n_2 满足什么条件的时候这个索引对于查找满足 $10 < A < 50 \wedge 5 < B < 10$ 条件的记录是一个高效的手段？

11.23

假设你必须在大量的名字上建立一个 B⁺ 树索引，这些名字的最大长度可能非常大(比如 40 个字符)，并且这些名字长度的平均值本身也很大(比如 10 个字符)。解释一下如何用前缀压缩来使内部结点的平均扇出尽可能大。

11.24

假设一个关系以 B⁺ 树文件组织的形式存储。假设辅助索引存储的记录标识符是指向磁盘中记录的指针。

部分 数据存储和查询

- a. 如果一个文件组织中发生磁盘页分裂，会对辅助索引产生什么影响？
- b. 更新辅助索引中所有影响到的记录，代价是多少？
- c. 使用一个逻辑记录标识符作为文件组织的搜索码如何能够解决这个问题？
- d. 使用这样的逻辑记录标识符会带来什么附加的代价？