8.1 Suppose that we decompose the schema $r(A, B, C, D, E)$ into

$$r_1(A, B, C)$$
$$r_2(A, D, E)$$

Show that this decomposition is a lossless decomposition if the following set F of functional dependencies holds:

$$A \rightarrow BC$$
$$CD \rightarrow E$$
$$B \rightarrow D$$
$$E \rightarrow A$$

From answer 8.6 below, A is a candidate key and A -> ABC.

Let:
R_1 = (A, B, C)
R_2 = (A, D, E)

$R_1 \cap R_2$ = A
$R_1 \cap R_2$ -> R_1

Since $R_1 \cap R_2$ -> R_1, the decomposition is lossless.

8.6 Compute the closure of the following set F of functional dependencies for relation schema $r\ (A,\ B,\ C,\ D,\ E)$.

$$A \rightarrow BC$$
$$CD \rightarrow E$$
$$B \rightarrow D$$
$$E \rightarrow A$$

List the candidate keys for R.

1. A -> A — Trivial
2. B -> B — Trivial
3. C -> C — Trivial
4. D -> D — Trivial
5. E -> E — Trivial

6. A -> BC Given
7. CD -> E Given
8. B -> D Given
9. E -> A Given
10. A -> BC => A -> B
11. A -> BC => A -> C
12. A -> B, B -> D => A -> D
13. A -> CD, CD -> E => A -> E
14. **A -> ABCDE** Steps
15. E -> A => **E -> ABCDE**
16. CD -> E => **CD -> ABCDE**
17. B -> D, BC -> CD => **BC -> ABCDE**
18. BD -> BD
19. B -> BD

Let x be any set within the powerset of {A B C D E}

F^+ = {Ax -> x, BCx-> x, CDx -> x, Ex -> x, B -> B, B → B D, B -> D, C -> C, D -> D, B D -> B, B D -> D, B D -> B D}

Only A, E, CD, and BC are minimal superkeys; the candidate keys are therefore A, E, CD, and BC.

8.19 Give a lossless-join decomposition into BCNF of schema *R* of Practice Exercise 8.1.

The schema R of practice exercise is r(A, B, C, D, E).
With functional dependencies:
A -> BC
CD -> E
B -> D
E -> A

We know that B -> D is non-trivial, and that B is not a superkey, from exercise 8.6. Therefore, we can decompose the relationship to:
r_1(B, D)
r_2(A, B, C, E)

8.28 Show that the following decomposition of the schema R of Practice Exercise 8.1 is not a lossless decomposition:

$$(A, B, C)$$
$$(C, D, E)$$

Hint: Give an example of a relation r on schema R such that

$$\Pi_{A, B, C}(r) \bowtie \Pi_{C, D, E}(r) \neq r$$

Let r:

A	B	C	D	E
y	y	x	y	y
z	z	x	z	z

Let r_1:

A	B	C
y	y	x
z	z	x

Let r_2:

C	D	E
x	y	y
x	z	z

$r_1 \bowtie r_2$:

A	B	C	D	E
y	y	x	y	y
z	z	x	z	z

y	y	x	z	z
z	z	x	y	y

Since $r_1 \bowtie r_2 \neq r$, the decomposition is not lossless.

13.6 Suppose that a B^+-tree index on *building* is available on relation *department*, and that no other index is available. What would be the best way to handle the following selections that involve negation?

a. $\sigma_{\neg(building < \text{“Watson”})}(department)$

b. $\sigma_{\neg(building = \text{“Watson”})}(department)$

c. $\sigma_{\neg(building < \text{“Watson”} \vee budget < 50000)}(department)$

i

13.6

a. In this example, the best way to handle the selection would be to calculate the buildings whose name is greater than or equal to “Watson”. In relational algebra, this is:

$\sigma_{(building \geq "Watson")}(department)$

With a B^+ tree, this can be accomplished by searching for a node with value “Watson” in the B+ tree. The leaf node that should contain “Watson”, if it exists, should be searched for the value “Watson” or the first value that is greater than “Watson”. The tuples are then output in order, following the pointers between leaf nodes until there are no more tuples.

b. In this example, the index is of no use. The file is scanned sequentially, and every tuple where the *building* field is not “Watson” is output.

c. An equivalent relational algebra expression is:

$\sigma_{(building \geq "Watson" \wedge budget \geq 50000)}(department)$

We can retrieve all tuples where *building* is greater or equal to “Watson” as in part a. Before outputting each tuple, we check that *budget* is greater than 50,000.

11.4 For each B^+-tree of Practice Exercise 11.3, show the form of the tree after each of the following series of operations:

a. Insert 9.

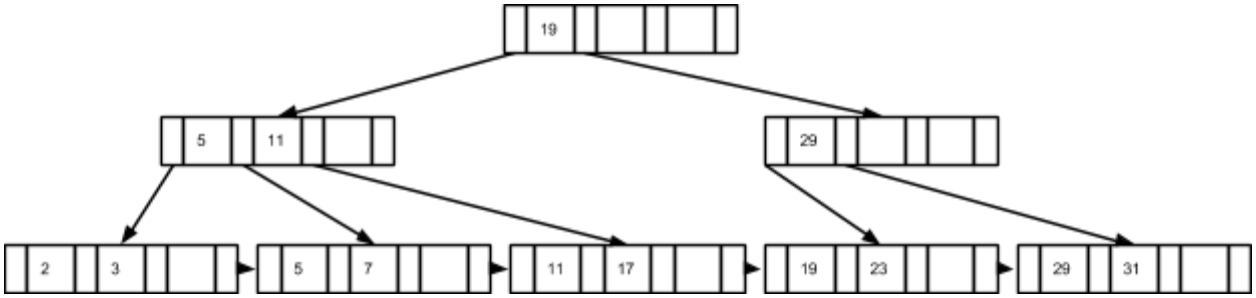
b. Insert 10.

c. Insert 8.

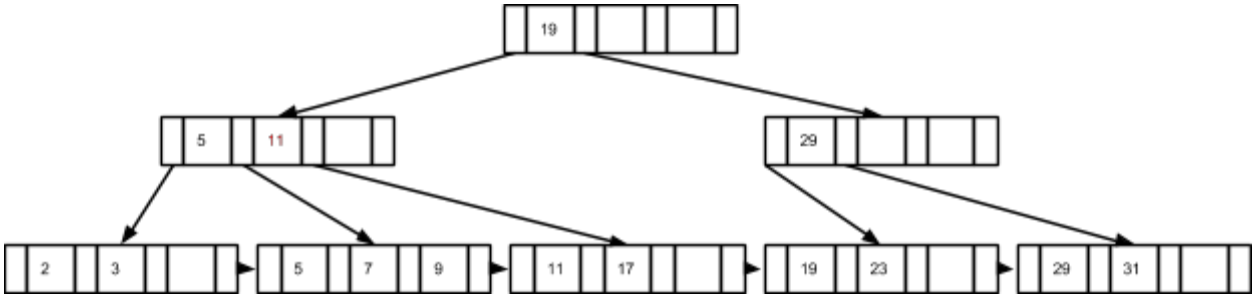
d. Delete 23.

e. Delete 19.

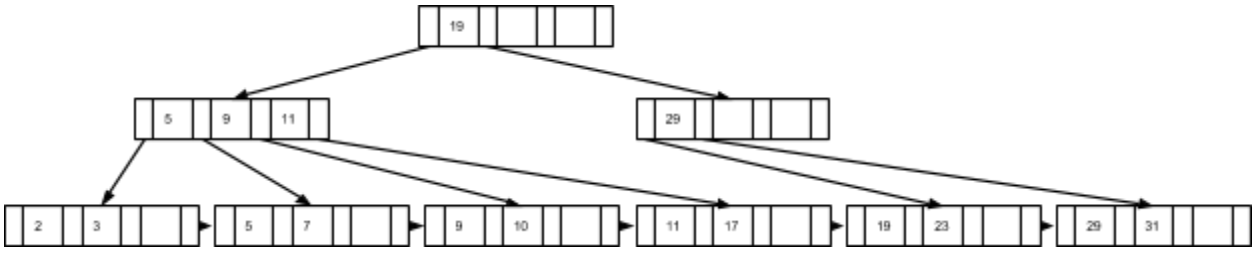
The initial B^+ tree with 4 pointers per node is:



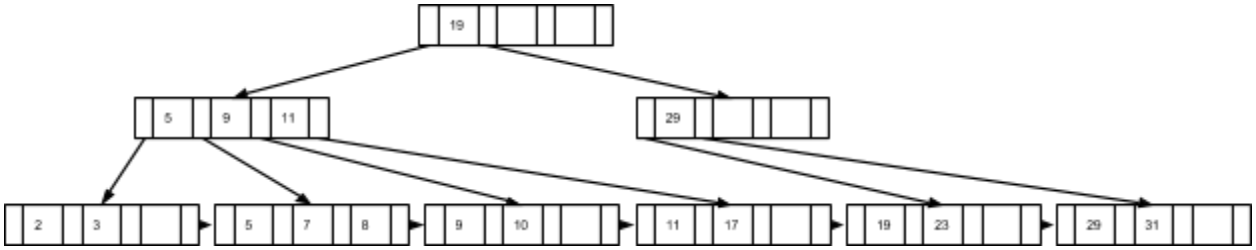
a. Insert 9



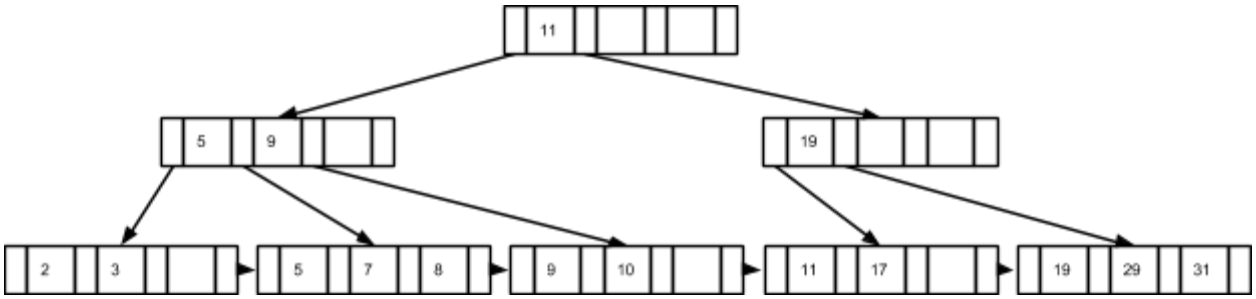
b. Insert 10



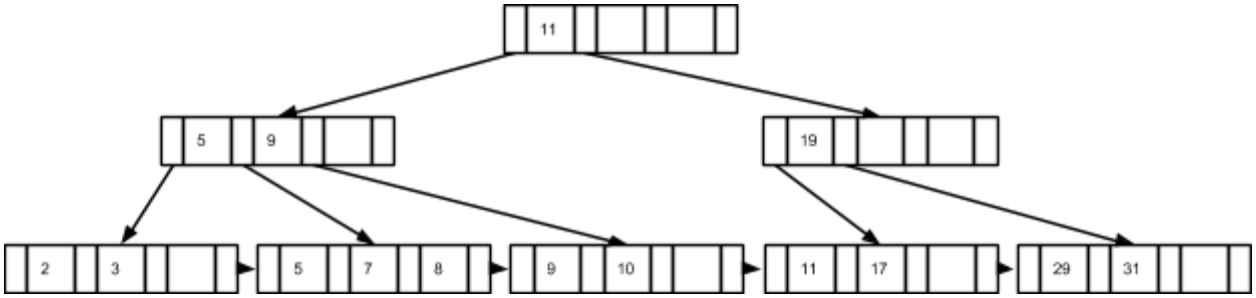
c. Insert 8



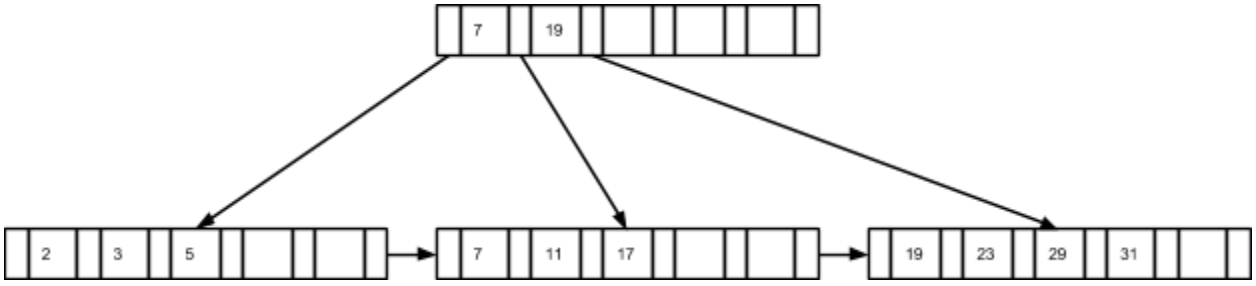
d. Delete 23



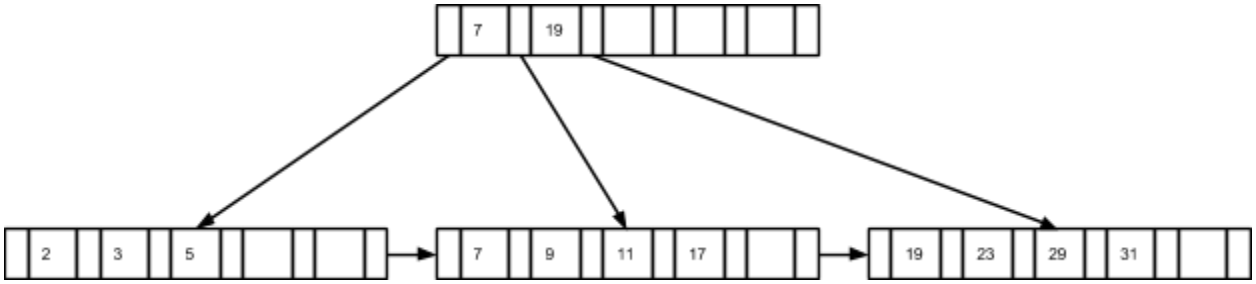
e. Delete 19



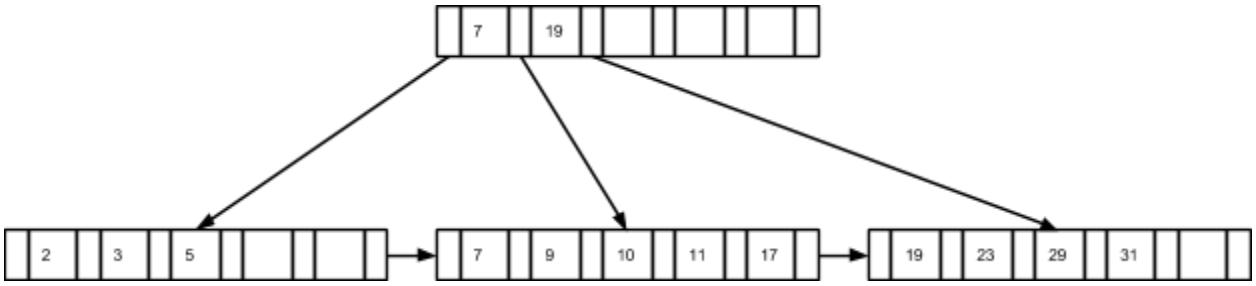
The initial B^+ tree with 6 pointers per node is:



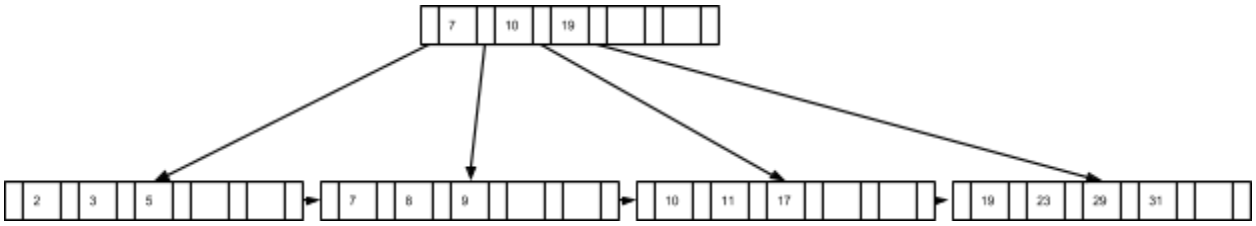
a. Insert 9



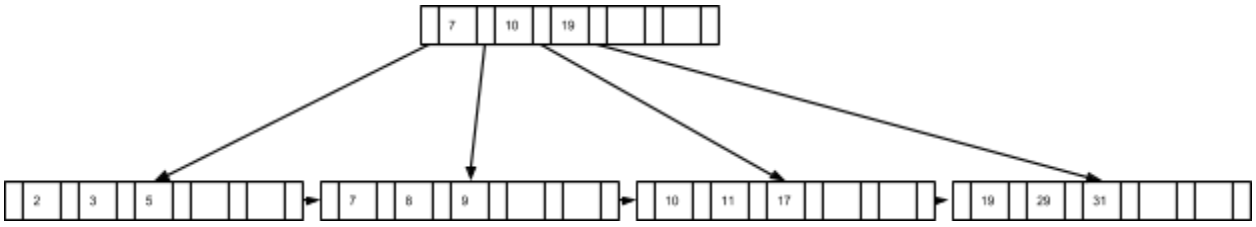
b. Insert 10



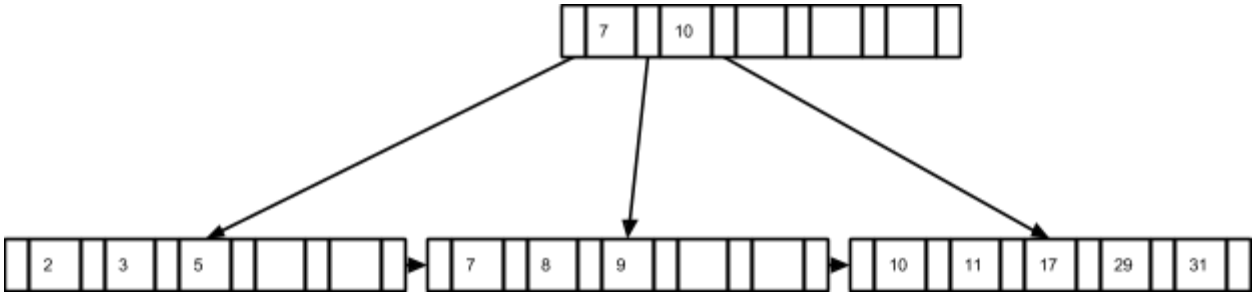
c. Insert 8



d. Delete 23

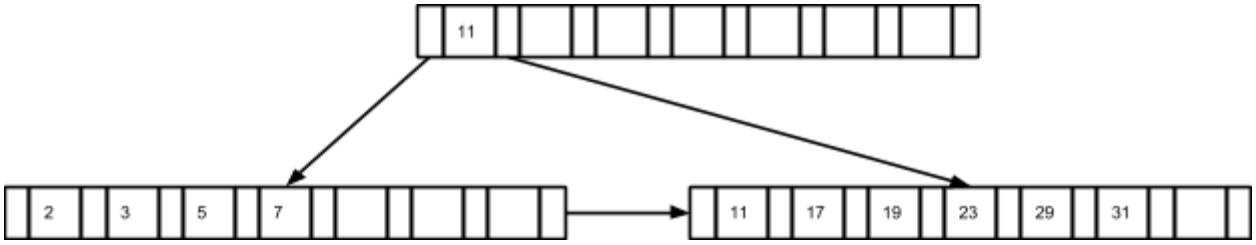


e. Delete 19

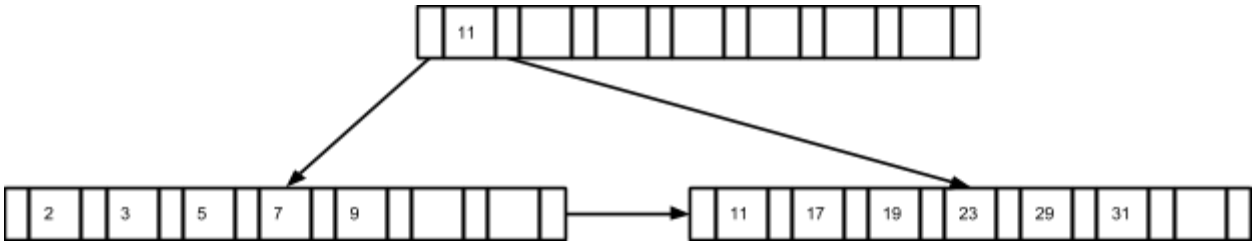


The initial B^+ tree with 8 pointers per node is:

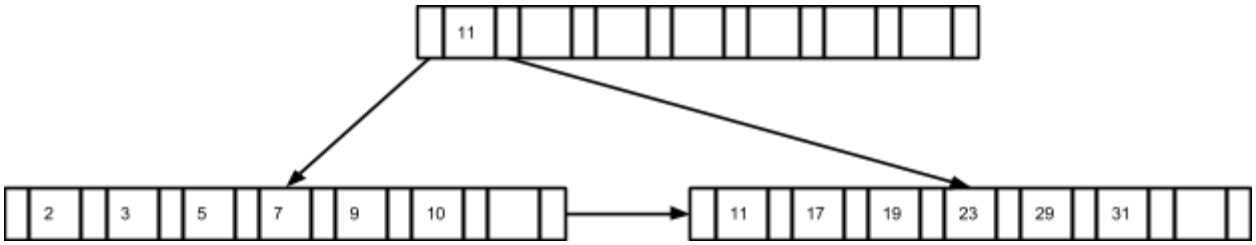
William Montgomery
CS 480 - Spring 2014
Homework 5



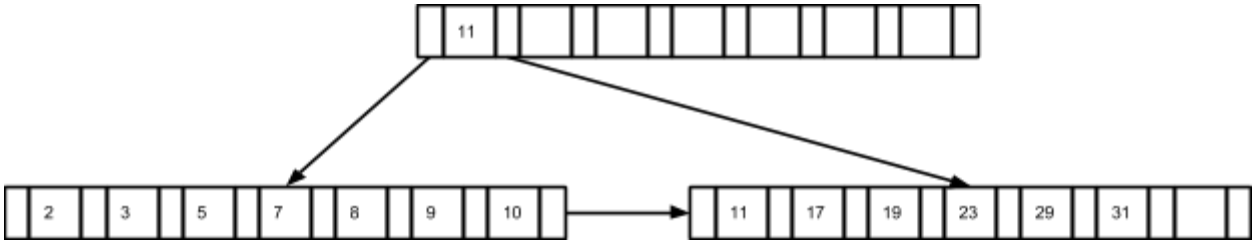
a. Insert 9



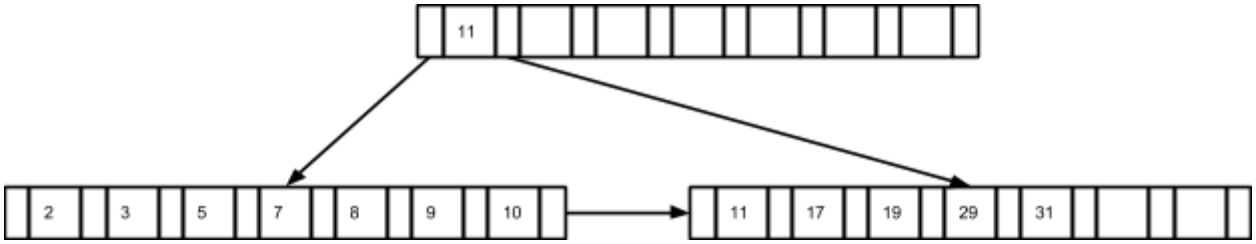
b. Insert 10



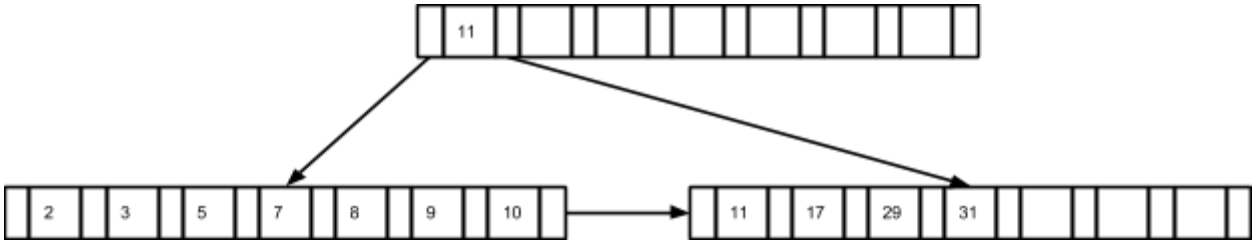
c. Insert 8



d. Insert 23



e. Delete 19



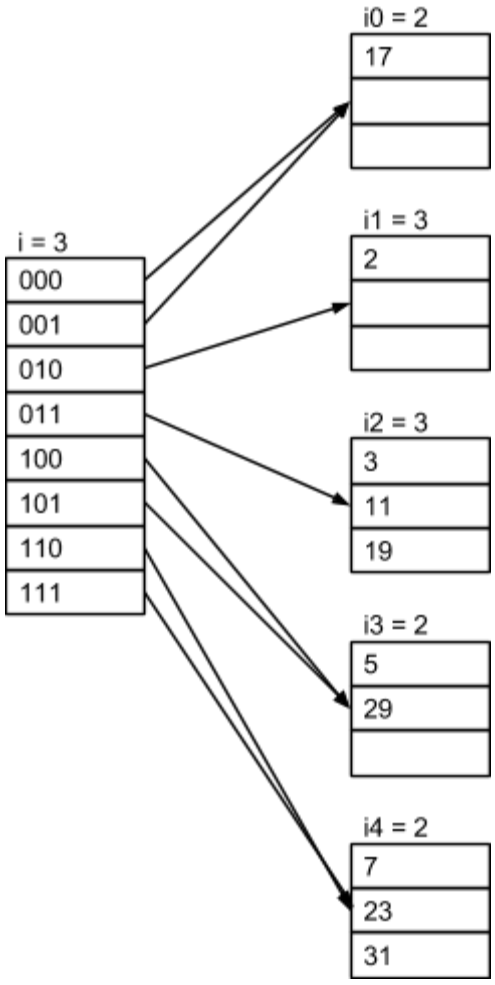
11.6 Suppose that we are using extendable hashing on a file that contains records with the following search-key values:

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

Show the extendable hash structure for this file if the hash function is $h(x) = x \bmod 8$ and buckets can hold three records.

First, I compute the hash function for each value:

h(2) = 2 mod 8 = 2 = 010_2
h(3) = 3 mod 8 = 3 = 011_2
h(5) = 5 mod 8 = 5 = 101_2
h(7) = 7 mod 8 = 7 = 111_2
h(11) = 11 mod 8 = 3 = 011_2
h(17) = 17 mod 8 = 1 = 001_2
h(19) = 19 mod 8 = 3 = 011_2
h(23) = 23 mod 8 = 7 = 111_2
h(29) = 29 mod 8 = 5 = 101_2
h(31) = 31 mod 8 = 7 = 111_2



William Montgomery
CS 480 - Spring 2014
Homework 5

12.3 Let relations $r_1(A, B, C)$ and $r_2(C, D, E)$ have the following properties: r_1 has 20,000 tuples, r_2 has 45,000 tuples, 25 tuples of r_1 fit on one block, and 30 tuples of r_2 fit on one block. Estimate the number of block transfers and seeks required, using each of the following join strategies for $r_1 \bowtie r_2$:

a. Nested-loop join.

b. Block nested-loop join.

c. Merge join.

d. Hash join.

Let x = r_1
Let y = r_2

x requires 20,000 tuples / 25 tuples per block = 800 blocks
y requires 45,000 tuples / 30 tuples per block = 1,500 blocks

Number of tuples in x: $n_x = 20,000$
Number of blocks in x: $b_x = 800$
Number of tuples in y: $n_y = 45,000$
Number of blocks in x: $b_y = 1,500$

a. Nested-loop join
I will use r_1 as the outer relation and assume that only one block of each relation will fit into memory at one time.
Block transfers: $n_x * b_y + b_x = 20,000 * 1,500 + 800 = 30,000,800$
Seeks: $n_x * b_x = 20,000 + 800 = 20,800$

b. Block nested-loop join
I will use r_1 as the outer relation. I will also assume that only one block of each relation fits in memory at the same time.
Block transfers: $b_x * b_y + b_x = 800 * 1,500 + 800 = 1,200,800$
Seeks: $b_x + b_x = 2 * b_x = 2 * 800 = 1,600$

c. Merge join
I will assume that only one block from each relation will fit into memory at one time.

If I assume that both of the relations are already sorted:
Block transfers: $b_x + b_y = 800 + 1,500 = 2,300$
Seeks: $b_x + b_y = 800 + 1,500 = 2,300$

In the case when the relations are not sorted, the cost of sorting much be factored in. For a single relation, this is:
Block transfers: ceil ($\log_{M-1}$ (b/M))
Seeks: 2 ceil (b_r/M) + ceil (b_r/b_b) (2 ceil ($\log_{M-1}$ (b_r /M)) − 1)

Where M is the number of blocks that are allocated in memory at one time, and b_b is the number of blocks read at a time. If we assume the worst case scenario where M = 3 (2 blocks being merged and 1 block for output) and b_b = 1, then we have:

Total Block transfers:

transfers sorting x + transfers sorting y + transfers merging x and y
= ceil ($\log_{3-1}$ (b_x/3)) + ceil ($\log_{3-1}$ (b_y/3)) + b_x + b_y
= ceil ($\log_2$ (800/3) + ceil ($\log_2$ (1,500/3)) + 800 + 1,500
= 2,318

Total Seeks:

seeks sorting x + seeks sorting y + seeks merging x and y
= 2 ceil (b_x/M) + ceil (b_x/b_b) (2 ceil ($\log_{M-1}$ (b_x /M)) − 1) +
2 ceil (b_y/M) + ceil (b_y/b_b) (2 ceil ($\log_{M-1}$ (b_y /M)) − 1) +
b_x + b_y
= 2 ceil (800/3) + ceil (800/1) (2 ceil ($\log_2$ (800 /3)) − 1) +
2 ceil (1500/3) + ceil (1500/1) (2 ceil ($\log_2$ (1500 /3)) − 1) +
800 + 1500
= 42,934

d. Hash-join

Since r_1 is the smaller relation, we use it as the build relation, and r_2 as the probe relation. I will also assume that there is no overflow. I will also assume that there is one block per relation available for the join.
Block transfers:

$3(b_x + b_y) + 4 * n_h = 3(800 + 1{,}500) + 4 * n_h = 3\ (2{,}300) + 4 * n_h = 6{,}900 + n_h$

Seeks:

$2(\text{ceil}(b_x/b_b) + \text{ceil}(b_s/b_b)) + 2n_h = 2(800 + 1{,}500) + 2n_h = 2(2{,}300) + 2n_h = 4{,}600 + 2n_h$