Problem 1  (Textbook Problem 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 \( h(x) = x \mod 8 \) and buckets can hold three records.

Answer:

11.6  Answer: Extendable hash structure

Given the above primary index of B+-tree of maximum fanout of 3, please answer the following questions.

1. List the nodes in the order of traversal for the following look up requests.

   (a) Look up key 13.

   (b) Look up key 8.

   (c) Look up keys in range \([6, 16]\) (inclusive range).
Answer:

(a) (9, null), (13, 17), (13, 15).
(b) (9, null), (5, null), (5, 7).
(c) (9, null), (5, null), (5, 7), (9, 11), (13, 15), (17, null).

2. Draw the tree after the following operations executed in the listed order. In case of leaf node split, assume two entries move to the new leaf.

(a) Insert 6.
(b) Insert 10.
(c) Delete 9.

Answer: Figure 1: Problem 5.2

![Tree Diagram]

Figure 1: Problem 5.2

3. Assume the original tree. How many keys at most can be inserted without any internal node split at the second level of the tree (i.e. nodes (5, null) and (13, 17))?  

Answer: At most 3 keys can be inserted before splitting any internal node at the second level. For example, insert 18, insert π and insert e. At this point the both the leaf level and the second level are saturated.

Problem 3

Consider the following hash buckets, each with a maximum size of 4 elements:

<table>
<thead>
<tr>
<th>catalog</th>
<th>buckets</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>8, 12, 24, 36</td>
</tr>
<tr>
<td>01</td>
<td>5, 17, 21</td>
</tr>
<tr>
<td>10</td>
<td>2, 18, 22</td>
</tr>
<tr>
<td>11</td>
<td>3, 7, 23, 31</td>
</tr>
</tbody>
</table>

Add the elements 11, 30, 28, 10 and 39 (in this order) using the following hashing strategies:

1. Use extendible hashing to add the elements. The buckets you start with have a global depth of 2, and each individual bucket also has a local depth of 2. If multiple entries in the directory point to the same bucket, be sure to indicate it in your solution. Please illustrate the full state of your hash buckets after each addition, including changes to the catalog and any change to global or local depth.
2. Use **linear hashing** to add the elements. Note that the "catalog" value is used purely for illustration, but you should maintain the values in your solution. The $Next$ pointer starts by pointing at the first bucket (labeled 00), and your value for $N = 4$. Please illustrate the full state of your hash buckets after each addition, including where the $Next$ pointer is pointing and additions to the "catalog" values.

1. Answer: The final state is

![Figure 2: Problem 6.1](image)

2. Answer: There can be many possible split criteria, but the one used here should be the same as stated in the lecture slides, unless the student stated otherwise. Assume an insertion causes a split when the corresponding bucket overflows after the insertion. The final state is

![Figure 3: Problem 6.2](image)
Problem 4  (Textbook Problem 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 required, using each of the following join strategies for \( r_1 \Join r_2 \):

Answer: \( r_1 \) needs 800 blocks, and \( r_2 \) needs 1500 blocks. Let us assume M pages of memory. If \( M > 800 \), the join can easily be done in 1500 + 800 disk accesses, using even plain nested-loop join. So we consider on the case where \( M \leq 800 \) pages.

1. Nested-loop join
   Using \( r_1 \) as the outer relation we need \( 20,000 \cdot 1,500 + 800 = 30,000,800 \) disk accesses, if \( r_2 \) is the outer relation we need \( 45,000 \cdot 800 + 1,500 = 36,001,500 \) disk accesses.

2. Block nested-loop join
   If \( r_1 \) is the outer relation, \( \lceil \frac{800}{M-2} \rceil \cdot 1500 + 800 \) disk accesses are needed. If \( r_2 \) is the outer relation, \( \lceil \frac{1500}{M-2} \rceil \cdot 800 + 1500 \) disk accesses are needed.

3. Merge join
   Assuming that \( r_1 \) and \( r_2 \) are not initially sorted on the join key, the total sorting cost inclusive of the output is \( B_s = 1500(2 \lceil \log_{M-1} 1500 \rceil + 2) + 800(2 \lceil \log_{M-1} 800 \rceil + 2) \) disk accesses. Assuming all tuples with the same value for the join attributes fit in memory, the total cost is \( B_s + 1500 + 800 \) disk accesses.

4. Hash join
   We assume no overflow occurs. Since \( r_1 \) is smaller, we use it as the build relation and \( r_2 \) as the probe relation. If \( M > 800/M \), i.e. no need for recursive partitioning, then the cost is \( 3(1500 + 800) = 6900 \) disk accesses, else the cost is \( 2(1500 + 800) \lceil \log_{M-1} 800 - 1 \rceil + 1500 + 800 \).

Problem 5  (Textbook Problem 12.6)

Consider the following bank database, where the primary keys are underlined:

\[
\begin{align*}
\text{branch} & : \text{(branch\_name, branch\_city, assets)} \\
\text{customer} & : \text{(customer\_name, customer\_street, customer\_city)} \\
\text{loan} & : \text{(loan\_number, branch\_name, amount)} \\
\text{borrower} & : \text{(customer\_name, loan\_number)} \\
\text{account} & : \text{(account\_number, branch\_name, balance)} \\
\text{depositor} & : \text{(customer\_name, account\_number)}
\end{align*}
\]

Suppose that a B+Tree index on \( \text{branch\_city} \) is available on relation \( \text{branch} \), and that no other index is available. List different ways to handle the following selections that involve negation:

Answer:

1. \( \sigma_{\text{\text{branch\_city} < "Brooklyn"}}(\text{branch}) \)
   Use the index to locate the first tuple whose \( \text{branch\_city} \) field has value “Brooklyn.” From this tuple, follow the pointer chains till the end, retrieving all the tuples.

2. \( \sigma_{\text{\text{branch\_city} = "Brooklyn"}}(\text{branch}) \)
   For this query, the index serves no purpose. We can scan the file sequentially and select all tuples whose \( \text{branch\_city} \) field is anything other than “Brooklyn.”

3. \( \sigma_{\text{\text{branch\_city} < "Brooklyn"}, \text{\text{assets} < 5000}}(\text{branch}) \)
This query is equivalent to the query
\[ \sigma_{\text{branch} \text{city} \geq "Brooklyn" \lor \text{assets} < 5000} (\text{branch}) \]
Using the index on \text{branch} \text{city}, we can retrieve all tuples with \text{branch-city} value greater than or equal to "Brooklyn" by following the pointer chains from the first "Brooklyn" tuple. We also apply the additional criteria of \text{assets} < 5000 on every tuple.

**Problem 6**  (Textbook Problem 13.4)

Consider the relations \( r_1(A, B, C) \), \( r_2(C, D, E) \), and \( r_3(E, F) \), with primary keys \( A \), \( C \), and \( E \), respectively. Assume that \( r_1 \) has 1000 tuples, \( r_2 \) has 1500 tuples, and \( r_3 \) has 750 tuples. Estimate the size of \( r_1 \bowtie r_2 \bowtie r_3 \) and give an efficient strategy for computing the join.

**Answer:**
1. The relation resulting form the join of \( r_1 \), \( r_2 \), and \( r_3 \) will be the same no matter which way we join them, due to the associative and commutative properties of joins. So we will consider the size based on the strategy of \((r_1 \bowtie r_2) \bowtie r_3\). Joining \( r_1 \) with \( r_2 \) will yield a relation of at most 1000 tuples, since \( C \) is a key of \( r_2 \). Likewise, joining that result with \( r_3 \) will yield a relation of at most 1000 tuples because \( E \) is a key of \( r_3 \). Therefore the final relation will have at most 1000 tuples.

2. An efficient strategy for computing this join would be to create an index on attribute \( C \) for relation \( r_2 \) and on \( E \) for \( r_3 \). Then for each tuple in \( r_1 \), we do the following:
   
   (a) Use the index for \( r_2 \) to look up at most one tuple which matches the \( C \) value of \( r_1 \).
   
   (b) Use the created index on \( E \) to look up in \( r_3 \) at most one tuple which matches the unique value of \( E \) in \( r_2 \).

**Problem 7**

Consider the following query on the \( \text{account}(\text{aID}, \text{name}) \) and \( \text{deposit}(\text{aID}, \text{date}, \text{amount}) \) relations:

\[
\text{SELECT a.name, d.date, d.amount}
\text{FROM deposit AS d}
\text{INNER JOIN account AS a}
\text{ON a.aID = d.aID}
\text{WHERE amount >= 400}
\]

Assume that \( \text{account} \) contains 10,000 accounts, and every account has made 50 deposits on average (the \( \text{deposit} \) table contains 500,000 deposits total). Both relations are not sorted in any particular order, and there are no indices on the relations. The in-memory buffer can hold up to 12 blocks and there is 100 tuples on average in every block (for both tables). Deposits range from $100 (inclusive) to $500 (exclusive), and you may assume an even distribution.

**Answer:** The table \( \text{account} \) has \( M = \frac{10,000}{100} = 100 \) blocks. The table \( \text{deposit} \) has \( N = \frac{500,000}{100} = 5,000 \) blocks. Let \( B \) be the number of blocks in the in-memory buffer, and \( B = 12 \). The expected number of blocks The expected number of tuples in \( \text{deposit} \) match the selection predicate is \( \frac{500,000 \times (500 - 400)}{500 - 100} = 125,000 \). The number of blocks needed to store these tuples is \( N_b = \lceil \frac{125,000}{100} \rceil = 1,250 \).

1. Using a **Block Nested-Loop Join**, compute the number of block accesses that will be required to perform the operation.

   Assume the \text{WHERE} clause is evaluated when tuples are joined. Assume the outer relation is \text{account}.

   The number of block accesses is \( M + \left\lceil \frac{M}{N} \right\rceil \times N = 100 + \left\lceil \frac{100}{1250} \right\rceil \times 50,000 = 50,100 \).
Alternatively, assume WHERE clause is evaluated before tuples are joined. To find \( N_s \) blocks it costs \( N + N_s \) disk transfers. The total cost is \( (N + N_s) + M + \left\lceil \frac{M}{s} \right\rceil \cdot N_s = (5,000 + 1,250) + 100 + 10 \cdot 1,250 = 18,850. \)

2. Compute the block accesses again using a Merge-Join instead. Both the deposit and account relations remain unordered.

Assume the WHERE clause is evaluated when tuples are joined. The cost of sorting account and deposit is \( (2M + 2N \cdot \left\lfloor \log_{B-1} M/B \right\rfloor) + (2N + 2N \cdot \left\lfloor \log_{B-1} N/B \right\rfloor) = 400 + 40,000 = 40,400. \) The cost of joining the sorted account and deposit is \( M + N = 100 + 5,000 = 5,100. \) The total cost is 40,400 + 5,100 = 45,500.

Alternatively, assume WHERE clause is evaluated before tuples are joined. To find \( N_s \) blocks it costs \( N + N_s \) disk transfers. The total cost is the sum of the costs of selection, merge-sort and join, or \( (N + N_s) + (2M + 2M \cdot \left\lfloor \log_{B-1} M/B \right\rfloor) + (2N_s + 2N_s \cdot \left\lfloor \log_{B-1} N_s/B \right\rfloor) + (M + N_s) = (5,000 + 1,250) + (2 \cdot 100 + 2 \cdot 100 \cdot \left\lfloor \log_{12-1} 100/12 \right\rfloor) + (100 + 1,250) = 15,500. \)

3. Now let’s add some indices to these tables. First let’s add a primary index on deposit.amount. The account relation remains unordered. Again, using a Block Nested-Loop Join, compute the number of block accesses that will be required to perform the operation. Assume that the WHERE clause is evaluated before the JOIN, with an index fan out of 100.

The data entry in the tree index contains a key and an actual tuple. The height of the tree is \( H = \left\lfloor \log_{\text{fanout}} 500,000/\text{num.entries/leaf} \right\rfloor = \left\lfloor \log_{10} 500,000/100 \right\rfloor = 2. \) To evaluate WHERE clause, all \( N_s \) blocks in deposit can be retrieved through a leaf-scan. The cost is \( N_i = H + N_s = 1,252. \)

The WHERE clause is evaluated before tuples are joined. The first way is to use account as the outer relation. We need to materialize \( N_s \) blocks first. To find \( N_s \) blocks it costs \( N_i + N_s \) disk transfers. The total cost is \( (N_i + N_s) + 3 = (1,252 + 1,250) + (100 + 1,250) = 15,102. \)

An alternative way is to use deposit as the outer relation, which does not require materialization. The total cost is \( N_i + \left\lfloor N_s/(B - 2) \right\rfloor \cdot M = 1,252 + 1,250/12 = 13,752. \)

(4. Assume that the primary index is now changed to deposit.aID, rather than deposit.amount. The account relation remains unordered. Using a Indexed Nested-Loop Join, compute the number of block accesses that will be required to perform the operation.

With the same tree index format assumption, the tree height is the same as above, \( H = 2. \) For deposit, the expected cost of looking up a tuple through the tree index is \( C_s = H + 1 = 3. \) Assume account is the outer relation. The expected total cost of the join is \( M + \left\lfloor \log_{2} N + N_s \right\rfloor \cdot C_s = 100 + 10 \cdot 3 = 100. \) This is not the assumption of this problem, though.)

5. A primary index is added on account.aID. Now that both relations are sorted, recompute the number of block accesses using a Merge-Join.

Assume the WHERE clause is evaluated when tuples are joined. Because we have primary indices on both tables, a merge join costs \( M + N = 5,100. \)

Alternatively, assume WHERE clause is evaluated before tuples are joined. To find \( N_s \) blocks it costs \( log_2 N + N_s \) disk transfers. The total cost is \( (log_2 N + N_s) + M = (\left\lceil log_2 5,000 \right\rceil + 1,250) + 100 = 1,363. \)

Problem 8  (Textbook Problem 10.11)

How does the remapping of bad sectors by disk controllers affect data-retrieval rates? Answer: Remapping of bad sectors by disk controllers reduces data retrieval rates because of the loss of sequentiality amongst the sectors.
Problem 9  (Textbook Problem 10.15)

Explain why the allocation of records to blocks affects database-system performance significantly. Answer: If we allocate related records to blocks, we can often retrieve most, or all, of the requested records by a query with one disk access. Disk accesses tend to be the bottlenecks in databases; since this allocation strategy reduces the number of disk accesses for a given operation, it significantly improves performance.

Problem 10

Imagine that you have the following table:

<table>
<thead>
<tr>
<th>bandId</th>
<th>bandName</th>
<th>ranking</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Bill Monroe</td>
<td>14</td>
</tr>
<tr>
<td>2</td>
<td>Grascals</td>
<td>19</td>
</tr>
<tr>
<td>3</td>
<td>Coon Creek Girls</td>
<td>53</td>
</tr>
<tr>
<td>4</td>
<td>Earl Scruggs</td>
<td>47</td>
</tr>
<tr>
<td>5</td>
<td>Hot Rize</td>
<td>22</td>
</tr>
<tr>
<td>6</td>
<td>Old Crow Medicine Show</td>
<td>87</td>
</tr>
<tr>
<td>7</td>
<td>Nashville Bluegrass Band</td>
<td>62</td>
</tr>
<tr>
<td>8</td>
<td>Infamous Stringdusters</td>
<td>55</td>
</tr>
<tr>
<td>9</td>
<td>Jim and Jesse</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>Doc Watson</td>
<td>6</td>
</tr>
<tr>
<td>11</td>
<td>Kathy Mattea</td>
<td>42</td>
</tr>
<tr>
<td>12</td>
<td>Avett Brothers</td>
<td>33</td>
</tr>
<tr>
<td>13</td>
<td>Mac Wiseman</td>
<td>92</td>
</tr>
<tr>
<td>14</td>
<td>Farmers Market</td>
<td>4</td>
</tr>
<tr>
<td>15</td>
<td>Psychograss</td>
<td>77</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>1024</td>
<td>Tangleweed</td>
<td>63</td>
</tr>
</tbody>
</table>

The table contains 1024 rows in total. Answer the following questions about this table:

1. Suppose the database administrator decides to store bandId as an 8-byte integer, bandName as a 48-byte character array, and ranking as a 8-byte integer. If an instance of bandName is shorter than 48 bytes, the empty space will be filled with null characters.
   • All attributes of a tuple are stored in contiguous space within the same disk block\(^1\).
   • A disk block size is 512 bytes.
   • The disk on average performs the sequential read at 1\(ms\) per disk block and the random read at 10\(ms\) per disk block.

   1. What is the maximum number of tuples that can be stored in a disk block? Answer: The maximum number of tuples can be stored in a disk block is
   \[
   \frac{512B}{8B + 48B + 8B} = 8
   \]

   2. What is the minimum number of disk blocks that need to be allocated for all tuples in the table? Answer: The minimum number of disk blocks that need to be allocated for all tuples in the table is
   \[
   \frac{1024 \text{ tuples}}{8 \text{ tuples per block}} = 128 \text{ blocks}
   \]

\(^1\)The minimum storage unit on a disk is called a disk block or a disk page.
3. What is the minimum time to read all tuples (in no particular order), assuming that the minimum number of disk blocks are allocated? Answer: The minimum time to read all tuples is when disk blocks are allocated next to each other. One random read incurs at the first block, and all sequential reads for the rest blocks thereafter.

\[10ms \times 1 + 1ms \times (128 - 1) = 137ms\]