Databases II
2019-10-10

1. Decode the following bitvector which was encoded with run-length-encoding!
11101101110110111010010001111010001

First, we count how many digits does it take to find the first 0. The answer is 4, so after the first 0, we consider the first 4 digits:

*1110*110111011011101001000111111010001

As a next step, we count the number of digits to find the first zero after our 4-digit number. It takes 3 digits, so we consider the next 3 digits after that zero:

*1110*1101*1110*1101110100100011111010001

We continue to process the bitvector the same way until we reach its end.

*1110*1101*1110*110*1110*1001*0*0*0*1*11110*10001

The base-2 numbers we get are:

1101, 110, 1001, 0, 1, 10001

After converting them to base-10:

13, 6, 9, 0, 1, 17

We get the original bitmap vector by writing down 13 zeros, then a one, 6 zeros, then a 1, 9 zeros, a one, 0 zeros, so we just put a 1 next to the previous one, and so on. The original bitmap is:

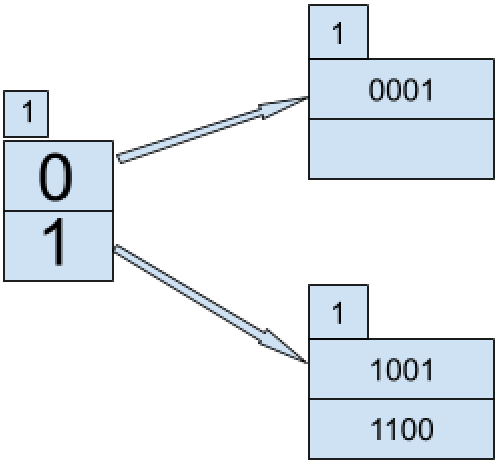
0000000000000100000010000000001101000000000000000001

2. Starting from the initial hashing table, insert the following keys: 0011, 0110, 1011, 0111, 1110, 1111, 0100

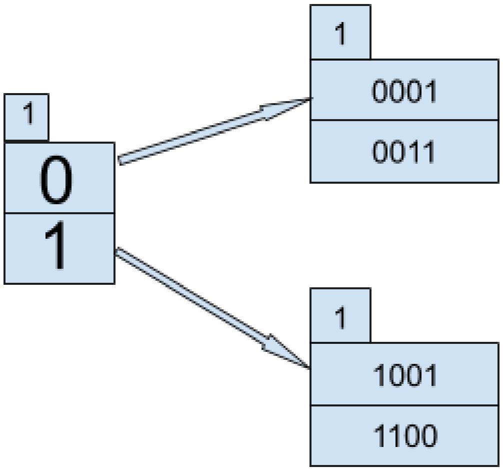
The size of the bucket array is always a power of 2.
Let's suppose we can put two records into a block, k=4 (the hash function computes k bits), i=1 (the bucket numbers use the first i bits); j (in the nub of the blocks) indicates how many bits of the hash function's sequence is used in this block.

To insert a record with search key K , we compute h(K), take the first i bits of this bit sequence, and go to the entry of the bucket array indexed by these i bits, then follow the pointer in this entry and arrive at a block B. If there is no room in the block B we do the following:
1. If j < i we split B into two; distribute records in B to the two blocks based on the (j+1)st bit; put j+1 into the block's nub; adjust the pointers in the bucket array
2. If j = i we increment i by 1; double the length of the bucket array; put new pointers into the array;
finally we proceed to split block B as in case 1.

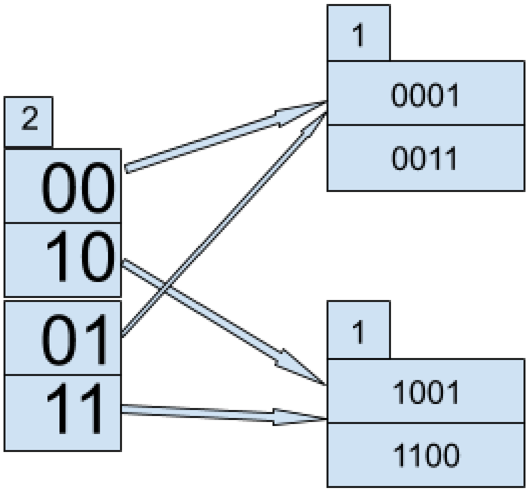
Initial state:



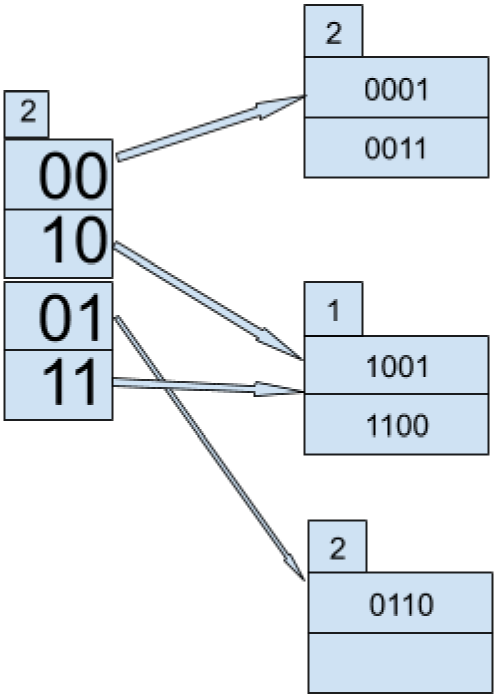
After inserting 0011:



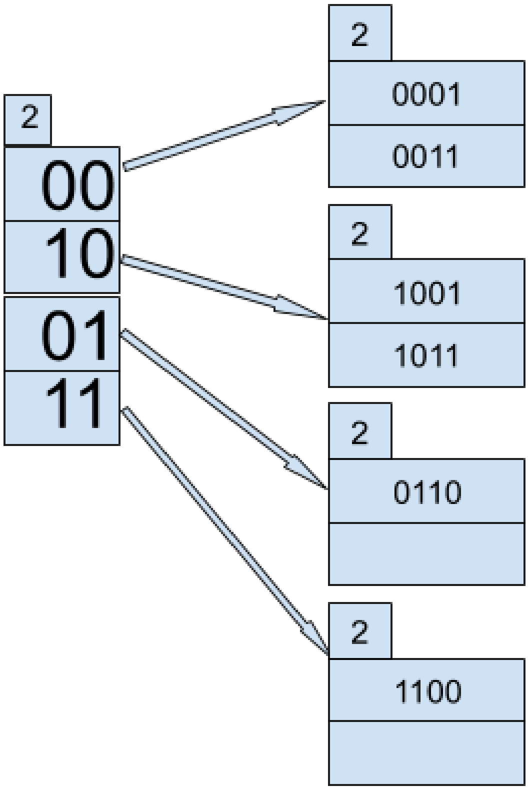
When trying to insert 0110, we find that the block pointed by [0] is full. The local depth of the block is 1, the global depth is also 1, so we need to double the number of buckets, increase the global depth, and set the new pointers.



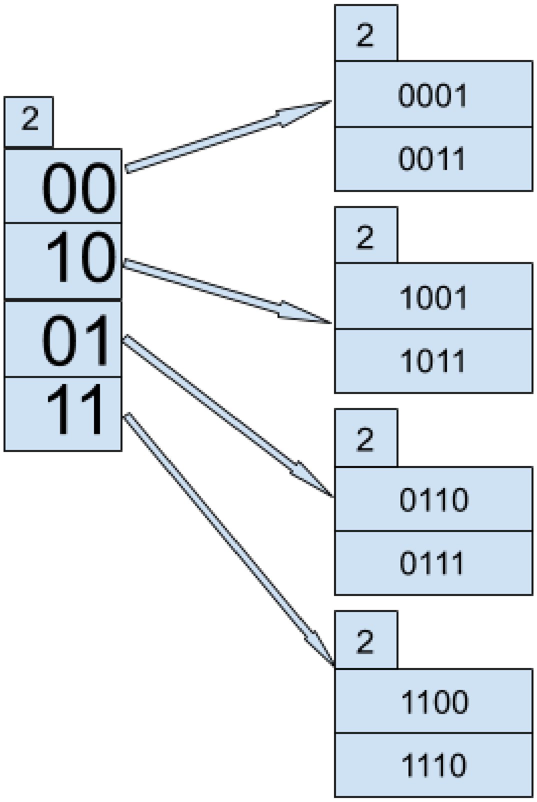
Block B is still full, but its local depth is now smaller than the global depth, so we split the block, increase the resulting block's local depth, redistribute the keys, and insert the new key.



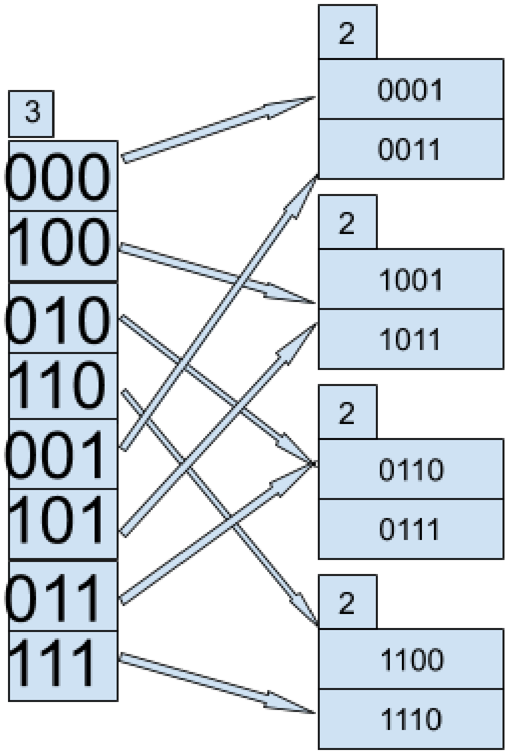
The next key is 1011. The block pointed by [10] is full, its local depth is smaller than the global depth, so we split the block, increase the resulting block's local depth, redistribute the keys, and insert the new key.



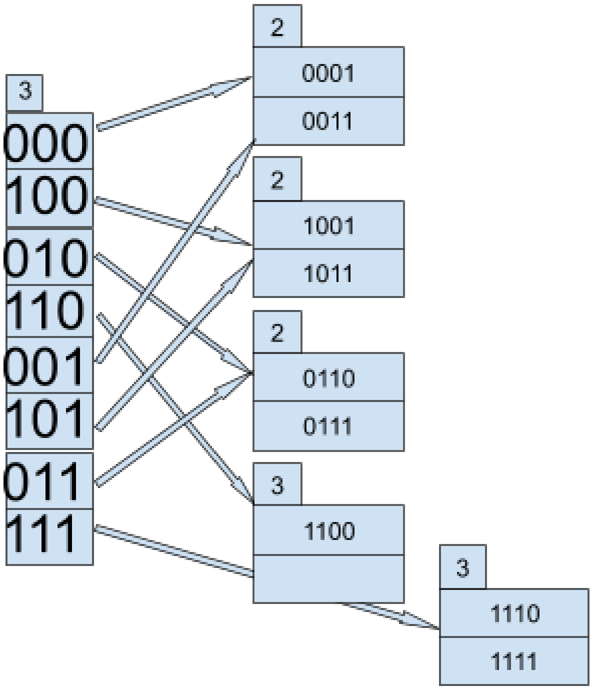
The next key is 0111, and there is a free space in the block pointed by [01]. Same with 1110.



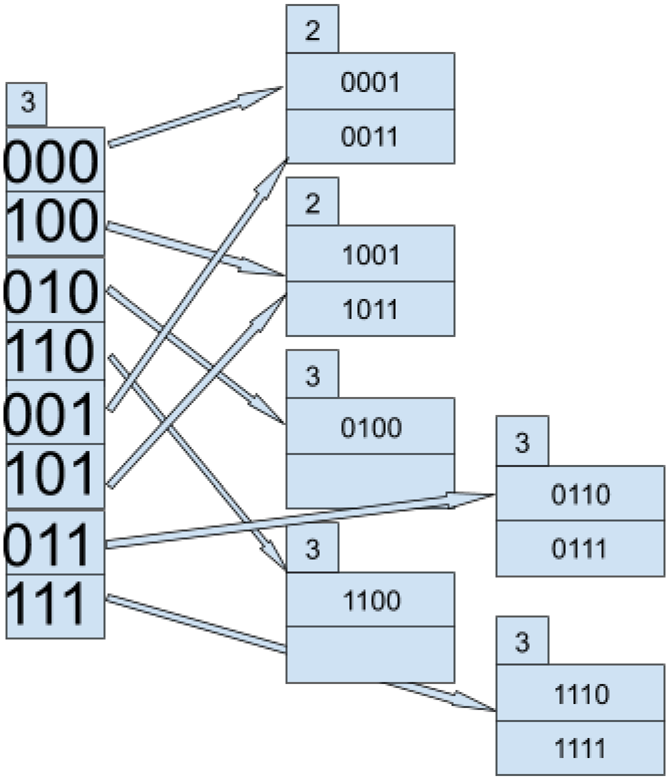
Unfortunately, 1111 can not be inserted in the block pointed by the [11] bucket. The local- and global depths are equal, so we need to increase the global depth and double the number of buckets.



Splitting block pointed by [110] and [111]:



Finally, inserting 0100 also requires splitting the block pointed by [010] and [011]:



That is the final form of the table, all keys have been inserted!

SQL

1. Give the tables (table_name) which has a column indexed in descending order. (In the solutions only objects of Nikovits are concerned.)

```
SELECT *
FROM dba_ind_columns
WHERE descend='DESC' AND index_owner='NIKOVITS';
```

2. Give the indexes (index name) which are composite and have at least 9 columns (expressions).

```
SELECT index_owner, index_name
FROM dba_ind_columns
GROUP BY index_owner, index_name
HAVING count(*) >=9;
```

Confirm one of them!

```
SELECT *
FROM dba_ind_columns
WHERE index_owner='SYS' AND index_name='I_OBJ2';
```

3. Give the name of bitmap indexes on table NIKOVITS.CUSTOMERS.

```
SELECT *
FROM dba_indexes
WHERE table_owner='NIKOVITS' AND table_name='CUSTOMERS' AND
index_type='BITMAP';
```

4. Give the indexes of owner NIKOVITS which has at least 2 columns and are function-based.

```
SELECT index_owner, index_name
FROM dba_ind_columns
GROUP BY index_owner, index_name
HAVING count(*) >=2
 INTERSECT
SELECT index_owner, index_name FROM dba_ind_expressions WHERE
index_owner='NIKOVITS';
```

5. Give for one of the above indexes the expression for which the index was created.

```
SELECT *
FROM dba_ind_expressions
WHERE index_owner='NIKOVITS';
```

6. Write a PL/SQL procedure which prints out the names and sizes (in bytes) of indexes created on the parameter table.

```
CREATE OR REPLACE PROCEDURE list_indexes(p_owner VARCHAR2, p_table
VARCHAR2) IS
   CURSOR cur1  IS SELECT i.index_name, s.BYTES
               FROM dba_indexes i, dba_segments s
               WHERE i.OWNER = upper(p_owner) and i.TABLE_NAME = upper(p_table) AND
               s.SEGMENT_TYPE='INDEX' AND s.SEGMENT_NAME=i.INDEX_NAME AND
i.OWNER=s.OWNER;
   rec cur1%ROWTYPE;
BEGIN
   OPEN cur1;
   LOOP
      FETCH cur1 into rec;
      EXIT WHEN cur1%NOTFOUND;
      dbms_output.put_line(rec.index_name || ' : ' || rec.bytes);
   END LOOP;
END;
```

Test:

```
-----
set serveroutput on
execute list_indexes('nikovits', 'emp');
```