COMP9315 DBMS Implementation 21T1 Final Exam

[Instructions] [PostgreSQL] [C]
[Q1] [Q2] [Q3] [Q4] [Q5] [Q6] [Q7] [Q8]

Question 1 (20 marks)

Consider the problem of determining which pages to visit for a partial-match retrieval query on a file using multi-attribute hashing. *Partial-match queries* have known values for some attributes, while the values for the other attributes are unknown. *Multi-attribute hashed files* have the following critical parameters:

- *n* = *number of attributes* in each tuple (indexed from 1..*n*)
- *d* = *depth* of file (file has 2^d pages, indexed *0..*2^d*-1*)
- *cv* = *choice vector* (which bit from which attribute is used in the combined hash value)

You are to complete a C program that behaves as follows:

- takes values for *n*, *d*, and *cv* from its command line
- reads and parses queries, one per line
- for each query, prints a line containing a list of data pages which need to be examined
  (these pages are selected because they may contain matching tuples according to the hash values in the query)

Some comments on the input formats for choice vectors and queries:

- attributes are indexed from *1 .. n*; pages are indexed from *0 ..* 2^d*-1*;
  bits are indexed from 31..0, with bit 0 being the least significant (rightmost) bit

- there must be no spaces embedded in either choice vectors or queries
  (the parsing functions are not particularly robust and may crash if you type invalid choice vectors and queries)

- a *choice vector* is written as a comma-separated list of attribute indexes

  Each element in the list determines the source of the next bit in the combined hash (starting from bit 0).
  Each time an attribute appears in the list, the next most significant bit from that attribute is used.

  Example: `1,1,2,2,3` represents the choice vector where

  - bit 0 of the combined hash value uses bit 0 from attribute 1
  - bit 1 of the combined hash value uses bit 1 from attribute 1
  - bit 2 of the combined hash value uses bit 0 from attribute 2
  - bit 3 of the combined hash value uses bit 1 from attribute 2
  - bit 4 of the combined hash value uses bit 0 from attribute 3

  The program displays this choice vector as: `< (1,0) (1,1) (2,0) (2,1) (3,0) >`

- a *query* is written as a comma-separated list of attribute values; unknown values are indicated by "?"

  Examples: for relation *R(a,b,c)*

```
SQL query:   select * from R where a = 1 and c = 'xyz'
Our format:  1,?,xyz

SQL query    select * from R where b = 5
Our format:  ?,5,?
```

The code for this question is in the `q1` directory, which contains:

- `pages.c` ... main program for determing pages to be read
- `hash.c` ... implementation of a hash function

- `hash.h` ... interface definitions for hash function
- `bits.c` ... implementation of bit-string ADT
- `bits.h` ... interface definitions for bit-string ADT
- `Makefile` ...for building the program
- `tests/` ... directory containing test cases

This contains a `Makefile` and several C program files (`*.c` and `*.h`). The program that you need to modify is contained in the file `pages.c`. You can compile the program using the `make` command. This will give you an executable file called `pages`. Examples of running the `pages` command are given below.

You should complete the section of code marked with `XXX`. You may write all of the code in the main program, or you can define as many functions as you want. It requires around 40 lines of code to solve this problem; partial marks are available if you complete some of the code.

Some hints on how to approach this problem:

- take a quick look at the `bits.c`, `hash.c` files to see what functions they provide
- you don't need to understand the details of the functions in these files; use them like library functions
- read the data structure definitions and main program in `pages.c` in some detail
- the `makeChoiceVector()` and `parseQuery()` functions show how the data structures are built
- read the block of comments in the main program near XXX, which give you some hints on what to do

Examples of how the program ought to behave when working correctly:

```
$ ./pages  3  4  1,1,2,2   -- 3 attributes, 16 pages, attr 3 unused in hash
CV = < (1,0) (1,1) (2,0) (2,1) >
query> ?,?,?
q[1] = ??
q[2] = ??
q[3] = ??
Pages: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
query> a,b,c
q[1] = 00101111 11100001 00011010 01110001
q[2] = 11101011 01100101 01010001 10111010
q[3] = 01101100 01000111 11011011 00100111
Pages: 9
query> a,b,?
q[1] = 00101111 11100001 00011010 01110001
q[2] = 11101011 01100101 01010001 10111010
q[3] = ??
Pages: 9
query> a,?,c
q[1] = 00101111 11100001 00011010 01110001
q[2] = ??
q[3] = 01101100 01000111 11011011 00100111
Pages: 1 5 9 13
query> quit

$ ./pages  2  4  1,2,1,2   -- 2 attributes, 16 pages, both attrs used in hash
CV = < (1,0) (2,0) (1,1) (2,1) >
query> 2,8
q[1] = 11100101 11000111 10110001 00101000
q[2] = 00100101 11001000 11110001 10110001
Pages: 2
query> 2,?
q[1] = 11100101 11000111 10110001 00101000
q[2] = ??
Pages: 0 2 8 10
query> ?,8
q[1] = ??
q[2] = 00100101 11001000 11110001 10110001
```

```
Pages: 2 3 6 7
query> ?,?
q[1] = ??
q[2] = ??
Pages: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
query> quit
```

You should also be able to devise your own test cases.

To help you check whether your program is working correctly, there is a script called `run_tests.sh` which will run the program against all of the tests and report the results. It will also add the output from your program into the `tests` directory; comparing your output against the expected output might help you to debug your code. You can run the testing script as:

```
$ sh run_tests.sh
```

Note that the output from the tests looks a little strange because it is not showing the query. It shows the `query>` prompt, but then immediately starts showing the output from determining the pages read.

Once your function is working (passes all tests), follow the submission instructions below. Even if it fails some tests, you should submit because you can get *some* marks. If your program does not compile, or if you simply submit the supplied code (even with trivial changes), then your "answer" is worth zero marks.

Submission Instructions:

- Type your answer to this question into the file called `pages.c`
- Submit via: **`give cs9315 exam_q1 pages.c`**
  or via: Webcms3 > exams > Final Exam > Q1 submission > Make Submission

End of Question