Assignment 2

Testing Resources

Last updated: **Sunday 24th March 3:50am**

Most recent changes are shown in red ... older changes are shown in brown.

Introduction

The following gives some ideas for testing your Assignment 2 code. It does not give you a simple scripted collection of tests. We expect **you** to think more about the process of developing methods for verifying the correctness of your code, not simply running it through a testing harness like `run_test.py`.

Warning: There are places in this system where you may observe different *but still correct* behaviour from your code compared to the examples below. Sometimes you should expect to see the exact output as shown. Sometimes, it acceptable to have variations on the shown output.

For example, the hash value for a given set of attribute values will always be the same. Query results should contain exactly the same set of tuples as shown, although the order may be different. The numbers of buckets and pages in the `stats` output may vary depending on precisely how you implement splitting.

Anywhere in the code that employs random number generation can lead to different results on different machines. In the supplied code, the only place that random numbers are used is in `gendata`. This means that if you run `gendata` on two different machines with the same set of command-line parameters, you may get a different output. If you want to make sure that you're loading the same set of data each time, run `gendata` and store the result in a file, then use that file to populate your databases.

To give a stable set of data for you to play with, there are two files produced with `gendata` that you can download and use:

```
/web/cs9315/24T1/assignments/ass2/tests/data0.txt
/web/cs9315/24T1/assignments/ass2/tests/data1.txt
```

You can use your browser's "Save link as ..." function to get a local copy of each file. In order to test your code more completely, you should also generate some data of your own. Especially generate very large data files, to ensure that your code continues to function properly at scale.

Task 1: Multi-attribute Hashing

After implementing multi-attribute hashing, you can test it as follows:

```
$ rm  -f  R.*
$ ./create  R  5  2  "0,1:1,1:2,1:3,1:4,1"
-- info about the choice vector
$ ./insert R < data0.txt
hash(1,kaleidoscope,hieroglyph,navy,hieroglyph) = 10010011 11111010 10100101 01111101
hash(2,floodlight,fork,drill,sunglasses) = 01001000 11100111 10010111 10101110
hash(3,bridge,torch,yellow,festival) = 01011000 10101011 11101011 11101110
hash(4,chief,carrot,gasp,rainbow) = 10110011 00111011 00110010 01101111
hash(5,rope,air,crystal,treadmill) = 10011100 11001000 10010000 01001001
hash(6,solid,sandpaper,sandpaper,sword) = 11011111 11011111 11000010 01010000
hash(7,surveyor,apple,leg,vampire) = 11001101 11000101 01100111 10010001
hash(8,sword,carpet,television,post) = 11101000 01100101 10101011 11010110
hash(9,surveyor,bank,spotlight,maze) = 10000110 10000011 11110010 10010101
hash(10,woman,eraser,planet,planet) = 01111010 01010001 10010000 01011101
$
```

You should see exactly the hash values above. Unless you change the print statements at the end of the `tupleHash()` function, you will not see all of the attribute values.

The `stats` command should produce the following output:

```
$ ./stats  R
Global Info:
#attrs:5  #pages:2  #tuples:10  d:1  sp:0
Choice vector
0,1:1,1:2,1:3,1:4,1:0,31:1,31:2,31:3,31:4,31:0,30:1,30:2,30:3,30:4,30:0,29:1,29:2,29:3,29:4,29:0,28:1,28:2,28:3,28:4,28:0,27:1,27:2,27:3,27:4,27:0,26:1,26
Bucket Info:
#    Info on pages in bucket
     (pageID,#tuples,freebytes,ovflow)
[ 0]  (d0,4,881,-1)
[ 1]  (d1,6,823,-1)
```

If it shows extra pages, then your splitting is happening too soon.

Task 2: Selection (Querying)

After implementing the `Query` ADT, you can test whether your queries work as follows:

```
$ ./create R 3 2 "0,0:1,0:2,0:0,1:1,1:2,1"
-- info about the choice vector
$ ./insert R < data1.txt
-- loads 2000 (id,v,w) tuples
-- id values range from 1000 .. 2999
```

You can use your browser's "Save link as ..." function to get a local copy of `data1.txt`.

You can then test your `Query` ADT by using the `select` command to ask queries of different types. Note that since the results come out effectively in random order (we're using hashing), we pipe the output through the `sort` command. (** The order is not "random", but depends on how you do your splitting).

```
$ ./select R '1042,?,?' | sort
1042,child,compact-disc
$ ./select R '?,horoscope,?' | sort
1172,horoscope,slave
1494,horoscope,water
2534,horoscope,cycle
2538,horoscope,dress
2650,horoscope,surveyor
2997,horoscope,famine
$ ./select R '?,?,shoes' | sort
1169,leg,shoes
1225,chair,shoes
1266,rope,shoes
1350,finger,shoes
1624,school,shoes
1770,bee,shoes
1978,woman,shoes
2550,chair,shoes
2982,bed,shoes
```

```
$ ./select R '?,shoes,?' | sort
1098,shoes,bee
1112,shoes,apple
1578,shoes,triangle
1671,shoes,butterfly
2523,shoes,barbecue
2749,shoes,chair
2992,shoes,clown
$ ./select R '?,chair,shoes' | sort
1225,chair,shoes
2550,chair,shoes
$ ./select R '?,shoes,chair' | sort
2749,shoes,chair
$ $ ./select R '?,?,?' | sort
--- gives all 2000 tuples in order
1000,kaleidoscope,hieroglyph
1001,navy,hieroglyph
1002,floodlight,fork
1003,drill,sunglasses
1004,bridge,torch
...
2995,perfume,button
2996,robot,sandwich
2997,horoscope,famine
2998,boss,carrot
2999,robot,parachute
```

All of the above queries produce one or more results. The following set of queries (should) produce no results:

```
./select R '42,?,?'
./select R '?,wombat,?'
./select R '?,?,wombat'
./select R '?,shoes,finger'
./select R '1001,chair,shoes'
```

If you want more test cases, it's easy to use `grep` on the `data1.txt` file to find what the results should be, e.g.

```
$ grep '^1042' data1.txt
-- gives same result as:
$ ./select R '1042,?,?' | sort

$ grep ',shoes,' data1.txt
-- gives same result as:
$ ./select R '?,shoes,?' | sort

$ grep ',shoes$' data1.txt
-- gives same result as:
$ ./select R '?,?,shoes' | sort

$ grep ',pendulum,elephant' data1.txt
-- gives same result as:
```

```
$ ./select R '?,pendulum,elephant'
```

Task 3: Linear Hashing

A simple example of how splitting might work. Note that there are many equally-valid variations: insert *before* splitting, insert tuples from overflow pages first, etc.

We assume a very simple database that can hold c = 2 tuples in each page. We denote pages by:

```
(#tuples, [list-of-tuples], overflow-page)
```

We also show just the ID to identify tuples and use an extremely simple hash function. We also show just the lower-order 4 bits of hash values; higher-order bits are not relevant for such a small database. None of this affects the general principles being illustrated; they are equally applicable to the assignment.

```
ID     hash     ID     hash     ID     hash
001 ...0001     006 ...0110     011 ...1011
002 ...0010     007 ...0111     012 ...1100
003 ...0011     008 ...1000     013 ...1101
004 ...0100     009 ...1001     014 ...1110
005 ...0101     010 ...1010     015 ...1111
```

We assume that a split occurs whenver we try to insert, and (*r* % 5) == 0, where *r* is the total number of tuples. This split frequency is different to what's specified in the assignment, but, once again, it's the principles that matter, not the precise values.

```
Initial state of database

sp = 0, d = 1
Data Pages              Overflow Pages
[0]  (0,[],-)
[1]  (0,[],-)


After inserting first five tuples
using 1 bit from hash value (d=1)

sp = 0, d = 1
Data Pages              Overflow Pages
[0]  (2,[002,004],-)    [0]  (1,[005],-)
[1]  (3,[001,003],0)


While inserting 006, need to split

Split bucket [0]
Tuples to be redistributed: [002,004]

First, clear bucket [0]

Data Pages              Overflow Pages
[0]  (0,[],-)           [0]  (1,[005],-)
[1]  (3,[001,003],0)
```

```
Add new data page [2]
Then re-insert [002,004] and insert [006]
using 2 bits of hash value
Then increment sp

sp = 1, d = 1
Data Pages              Overflow Pages
[0]  (1,[004],-)        [0]  (1,[005],-)
[1]  (3,[001,003],0)
[2]  (2,[002,006],-)
```

Reminder: once `sp = 1`, pages [0] and [2] are indexed using 2 bits from the hash value, and page [1] is indexed using just 1 bit. I.e. if the lower-order bit of the hash is 1, then the tuple goes into bucket [1]; if the the lower-order bit is 0, then we consider 2 bits to determine whether the tuple goes into page [0] or page [2] (hash bits `00` or `10`)

```
Then insert tuples 007 .. 010
No splits

sp = 1, d = 1
Data Pages              Overflow Pages
[0]  (2,[004,008],-)    [0]  (2,[005,007],1)
[1]  (3,[001,003],0)    [1]  (1,[009],-)
[2]  (2,[002,006],2)    [2]  (1,[010],-)


While inserting 011, need to split

Split bucket [1]
Tuples to be redistributed: [001,003,005,007,009]

First, clear bucket [1]

sp = 1, d = 1
Data Pages              Overflow Pages
[0]  (2,[004,008],-)    [0]  (0,[],-)
[1]  (0,[],-)           [1]  (0,[],-)
[2]  (2,[002,006],2)    [2]  (1,[010],-)

Add new data page [3]
Then re-insert [001,003,005,007,009]
using 2 bits of hash value
Then insert 011
And increment sp, but has reached a new
power of two, so reset sp = 0 and increment d

sp = 0, d = 2
Data Pages              Overflow Pages
[0]  (2,[004,008],-)    [0]  (1,[009],-)
[1]  (2,[001,005],0)    [1]  (1,[011],-)
[2]  (2,[002,006],2)    [2]  (1,[010],-)
[3]  (2,[003,007],1)
```

```
Insert 012 .. 015
No splits

sp = 0, d = 2
Data Pages              Overflow Pages
[0]  (2,[004,008],3)    [0]  (2,[009,013],-)
[1]  (2,[001,005],0)    [1]  (2,[011,015],-)
[2]  (2,[002,006],2)    [2]  (1,[010,014],-)
[3]  (2,[003,007],1)    [3]  (1,[012],-)
```

Giving examples with hundreds or thousands of tuples, using the settings for the assignment is unlikely to be helpful, given the variety of valid in ways of doing splitting.

To illustrate the point, almost every correct solution will produce different `stats` output after inserting the same data.