Exercises 06

Implementing Selection on One Attribute (1-d)

[Show with no answers] [Show with all answers]

1. Consider the relations:

```
Student(id#, name, age, course)
Subject(code, title, description)
```

Make the following assumptions:

- $r_{Student} = 50{,}000$, $r_{Subject} = 5{,}000$
- the data file for `Student` is sorted on `id#`, the data file for `Subject` is sorted on `code`
- simple one-level indexes are used (e.g. because the files are reasonably static)
- record IDs (RIDs) are 4-bytes long
- all data pages and index pages are 4KB long

Based on these assumptions, answer the following:

a. You can build a dense index on any field. Why?

[show answer]

b. On which field(s) could you build a non-dense index?

[show answer]

c. If the student id# is a 4-byte quantity, how large would a dense index on `Student.id#` be?

[show answer]

d. If the subject code is an 8-byte quantity, how large would a dense index on `Subject.code` be?

[show answer]

e. If the $c_{Student} = 100$ and $c_{Subject} = 20$, and other values are as above, how large would non-dense indexes on `Student.id#` and `Subject.code` be?

[show answer]

f. If you had just dense indexes on `Student.id#` and `Subject.code`, briefly describe efficient processing strategies for the following queries:

i. `select name from Student where id=2233445`
ii. `select title from Subject where code >= 'COMP1000' and code < 'COMP2000'`
iii. `select id#,name from Student where age=18`
iv. `select code,title from Subject where title like '%Comput%'`

[show answer]

g. What techniques could you use to improve the performance of each of the above queries? And how would it impact the other queries?

[show answer]

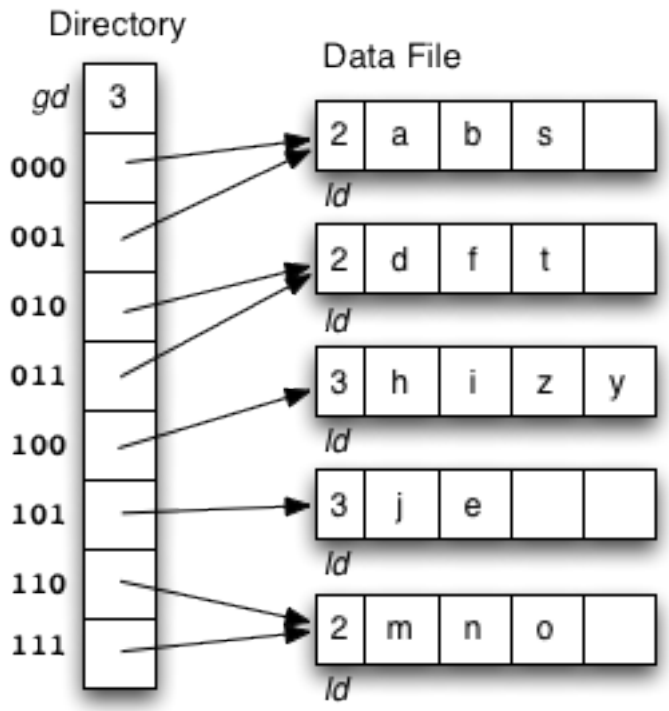
2. Consider a relation R(a,b,c,d,e) containing 5,000,000 records, where each data page of the relation holds 10 records. R is organized as a sorted file with the search key R.a. Assume that R.a is a candidate key of R, with values lying in the range 0 to 4,999,999.

   Determine which of the following approaches to evaluating the relational algebra expression $\pi_{a,b}(\sigma_{a>50000}(R))$ is likely to be the cheapest (minimum disk I/O):

   a. Use a clustered B+ tree index on attribute R.a.
   b. Use a linear hashed index on attribute R.a.
   c. Use a clustered B+ tree index on attributes (R.a,R.b).
   d. Use a linear hashed index on attributes (R.a,R.b).
   e. Use an unclustered B+ tree index on attribute R.b.
   f. Access the sorted file for R directly.

   [show answer]

3. Consider the following example file organisation using extendible hashing. Records are inserted into the file based on single letter keys (only the keys are shown in the diagram).



The dictionary contains a "global depth" value (*gd*), which indicates how many hash bits are being considered in locating data pages via the dictionary. In this example, the depth *gd=3* and so the dictionary is size $2^{gd}=2^{d}=8$.

Each data page is marked with a "local depth" value (*ld*), which indicates the effective number of bits that have been used to place records in that page. The first data page, for example, has *ld=2*, which tells us that only the first two bits of the hash value were used to place records there (and these two bits were `00`). The third data page, on the other hand, has *ld=3*, so we know that all records in that page have their first three hash bits as `100`.

Using the above example to clarify, answer the following questions about extendible hashing:

   a. Under what circumstances must we double the size of the directory, when we add a new data page?

      [show answer]

b. Under what circumstances can we add a new data page without doubling the size of the directory?

[show answer]

c. After an insertion that causes the directory size to double, how many data pages have exactly one directory entry pointing to them?

[show answer]

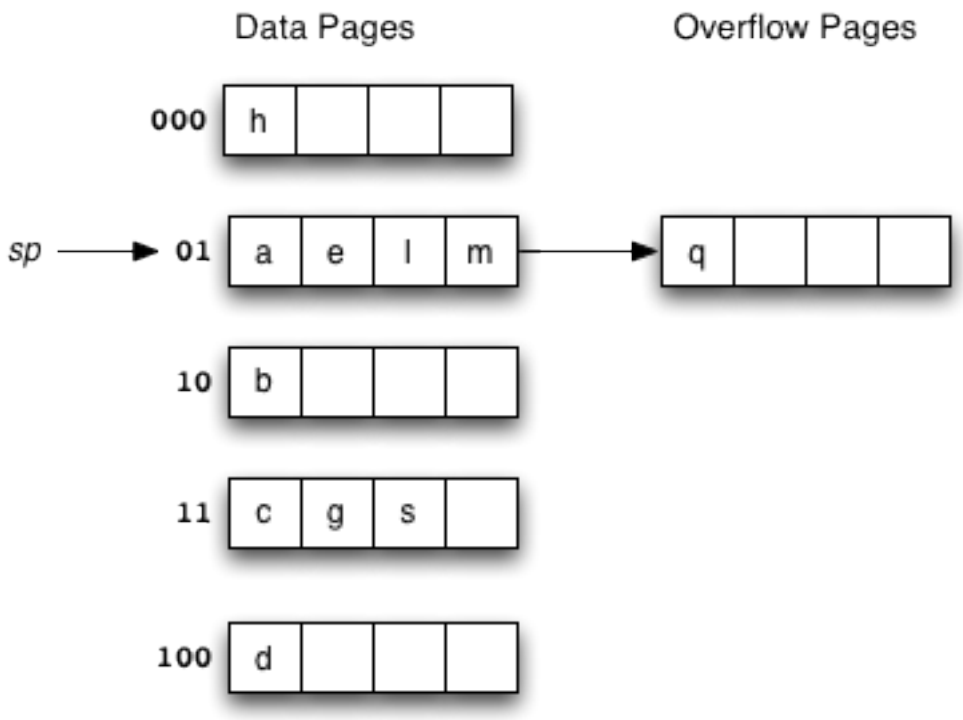
d. Under what circumstances would we need overflow pages in an extendible hashing file?

[show answer]

e. What are the best-case and worst-case scenarios for space utilisation in the dictionary and the data pages (assuming that there are no overflow pages)? Under what circumstances do these scenarios occur?

[show answer]

4. Consider the following example file organisation using linear hashing. Records are inserted into the file based on single letter keys (only the keys are shown in the diagram).



Using the above example to clarify, answer the following questions about linear hashing:

a. How is it that linear hashing provides an average case search cost of only slightly more than one page read, given that overflow pages are part of its data structure?

[show answer]

b. Under what circumstances will the average overflow chain length (*Ov*) not be reduced during page splitting?

[show answer]

c. If a linear hashing file holds *r* records, with *C* records per page and with splitting after every *k* inserts, what is the worst-case cost for an equality search, and under what conditions does this occur?

[show answer]

5. Consider the following employee relation:

```
Employee(id#:integer, name:string, dept:string, pay:real)
```

and some records from this relation for a small company:

```
(11, 'I. M. Rich', 'Admin',    99000.00)
(12, 'John Smith', 'Sales',    55000.00)
(15, 'John Brown', 'Accounts', 35000.00)
(17, 'Jane Dawes', 'Sales',    50000.00)
(22, 'James Gray', 'Sales',    65000.00)
(23, 'Bill Gates', 'Sales',    35000.00)
(25, 'Jim Morris', 'Admin',    35000.00)
(33, 'A. D. Mine', 'Admin',    90000.00)
(36, 'Peter Pipe', 'R+D',      30000.00)
(44, 'Jane Brown', 'R+D',      30000.00)
(48, 'Alex Brown', 'Plant',    40000.00)
(55, 'Mario Reid', 'Accounts', 27000.00)
(57, 'Jack Smith', 'Plant',    35000.00)
(60, 'Jack Brown', 'Plant',    35000.00)
(72, 'Mario Buzo', 'Accounts', 30000.00)
(77, 'Bill Young', 'Accounts', 31000.00)
(81, 'Jane Gates', 'R+D',      25000.00)
(88, 'Tim Menzie', 'R+D',      45000.00)
(97, 'Jane Brown', 'R+D',      30000.00)
(98, 'Fred Brown', 'Admin',    60000.00)
(99, 'Jane Smith', 'Accounts', 30000.00)
```

Assume that we are using the following hash function:

```
hash(11) = 01001010
hash(12) = 10110101
hash(15) = 11010111
hash(17) = 00101000
hash(22) = 10000001
hash(23) = 01110111
hash(25) = 10101001
hash(33) = 11001100
hash(36) = 01111000
hash(44) = 10010001
hash(48) = 00001111
hash(55) = 10010001
hash(57) = 11100011
hash(60) = 11000101
hash(72) = 10010010
hash(77) = 01010000
hash(81) = 00010111
hash(88) = 10101011
hash(97) = 00011100
hash(98) = 01010111
hash(99) = 11101011
```

Assume also that we are dealing with a file organisation where we can insert four records on each page (data page or overflow page) and still have room for overflow pointers, intra-page

directories, and so on.

Show the insertion of these records in the order given above into an initially empty extendible hashing file. How many pages does the relation occupy (including the pages to hold the directory)?

[show answer]

6. Using the same data as for the previous question, show the insertion of these records in the order given above into an initially empty linear hashed file. How many pages does the relation occupy (including any overflow pages)?

[show answer]