COSC-4411: Assignment #1

Due: 30 January 2003

---

1. **Buffer Pool.** *The Popularity Contest.*

   A page that is pinned multiple times at some point (that is, its *pincount* was greater than one) is intuitively more popular than a page that was only pinned singly (that is, its *pincount* was equal to one). This is because several transactions were using the page at once.

   However, none of the standard replacement policies—for instance, LRU nor Clock—account for this. In LRU, a page's popularity is measured by how recently it was last pinned (regardless of the *pincount*). Likewise in Clock, every unpinned page has an equal chance of replacement after having been passed over once.

   Design a new replacement policy that is an adaptation of Clock which does take into account pages that were multiply pinned.

---

2. **Buffer Pool.** *I'm sorry. You're being replaced.*

   A database is spread over two disks, **A** and **B**. An I/O read/write to **A** is ten times faster than an I/O to **B** . The same number of the database's pages reside on disk **A** as on **B** .

   Should the replacement policy of the buffer manager be changed or modified from a standard replacement policy (such as LRU or Clock) because of this?

   If so, how can the policy be changed to make the database system more efficient?

---

3. **Page Organization.** *Free the space!*

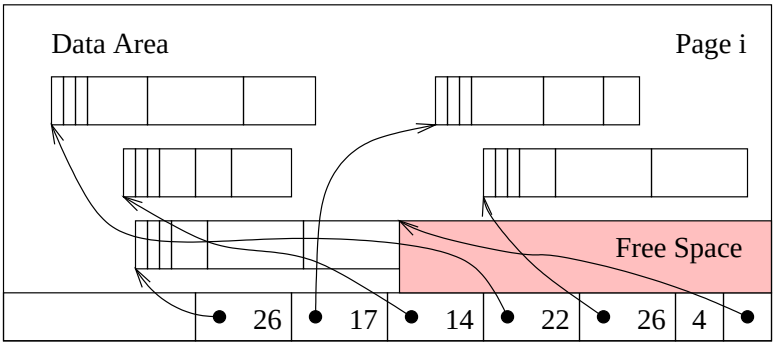

Figure 1: Page layout with variable length records.

Consider a page layout with variable length records and a slot directory as in Figure 1.

Say that we want to add a new record to page i, but there is not enough space for the new record in page i's *free space*. However, there is enough space overall on the page for the new record. That is, if the records already on page i were rearranged ideally, then the new record

would fit. Namely, we have to *pack* the records on page i: move them all so that they are sequentially adjacent starting from the beginning of the page. After this, all the empty space does belong to *free space*.

Write a pseudo-code algorithm for packing a page.

---

4. **Search Keys and Indexes.** *An index by any other name...*

   Consider the table **Student** with six attributes name, age, address, year, gpa, and major.

   Does it make sense to have both major + year and major + year + gpa as tree indexes? (Recall, the order of the attributes of an index's search key is important.) Briefly, why or why not?

   If both indexes are potentially useful for lots of the database's queries, but you can only keep one of them, which one would you keep? Why?

---

5. **Search Keys and Indexes.** *An index by any other name...*

   Consider the table **Student** with six attributes name, age, address, year, gpa, and major.

   Say that the following indexes exist on **Student**:

   A. name + gpa

   B. gpa + name

   C. major + year

   D. name + age + year

   Assume each index is good for both range and equality queries.

   For each of the following queries, say which of the indexes possibly could be used to evaluate the query, and for each, briefly in what way the index could be used to advantage. State *all* that could apply. If none is applicable, say so. If more than one is applicable, which would most likely be the best choice and why? Or maybe none is.

   a. select * from Student
      where gpa $\geq$ 8.0;

   b. select * from Student
      where name = 'George P. Burdell';

   c. select * from Student
      where age = 24;

   d. select * from Student
      where major = 'computer science' and age $\geq$ 19;

6. **I/O.** *I/O, I/O, it's off to work we go.*

   Consider file **R** which has 10,000,000 records, and that we have a B+ tree index on this file with the search key as **R**'s primary key. Assume that the order (d) of the B+ tree is 60, and that the index pages are 80% full, on average.

   a. Assume that the B+ tree index above for **R** is of alternative *one*; that is, the leaf pages of the B+ tree contain the data records themselves (at 20 data records per page).

      You want to find all records with search key $\geq x$ and search key $\leq y$. Say that 1,000 data records qualify. How many I/O's does it cost you to retrieve these?

   b. Assume that the B+ tree index above for **R** is of alternative *two* and is unclustered; that is, the leaf pages of the B+ tree contain data entries that are $\langle \mathsf{key}, \mathsf{rid} \rangle$ pairs and not the data *records* themselves. Say that 50 data entries fit per page, and say that your buffer pool is absurdly small with just five frames.

      Again, you want to find all records with search key $\geq x$ and search key $\leq y$, and 1,000 data records qualify. How many I/O's does it cost you to retrieve these?

   c. Now assume that we have an index on **R**'s primary key instead that is an extendible hash index. Assume that the index is of alternative *two* and is unclustered. We want to find the record with search key value x. How many I/O's does it cost you to retrieve the record?