CS127 Homework 5

Due: October 29th, 2014 3:00PM

Warmup 1 (Textbook Problem 11.15)

When is it preferable to use a dense index rather than a sparse index? Explain your answer.

Warmup 2 (Textbook Problem 11.16)

What is the difference between a clustering index and a secondary index?

Warmup 3 (Textbook Problem 11.22)

Up until now, the indexes covered in class have all been single-field indexes. However, we can also have what's called a **compound index**, which allows a dataset to be sorted by multiple attributes. Suppose there is a relation $r(A,B,C)$, with a B^+-tree index with search key (A,B). This means that r is sorted first by A, then by B.

1. What is the worst-case cost of finding records satisfying $10 < A < 50$ using this index, in terms of the number of records retrieved n_1 and the height h of the tree?
2. What is the worst-case cost of finding records satisfying $10 < A < 50 \wedge 5 < B < 10$ using this index, in terms of the number of records n_2 that satisfy this selection, as well as n_1 and h defined above?
3. Under what conditions on n_1 and n_2 would the index be an efficient way of finding records satisfying $10 < A < 50 \wedge 5 < B < 10$?

Problem 4 (To Be Graded)

Consider again the simplified university registrar database from the previous homeworks:

Student		
name	**gradyear**	**gpa**
Amy	2016	3.95
Ben	2015	3.87
Carl	2016	3.29
Dan	2017	3.43
Eliza	2015	4.0

Course		
title	**semester**	**instructor**
CS33	2014F	Doeppner
CS127	2014F	Zdonik
CS195	2013F	Kraska
CS127	2012F	Zdonik
CS136	2012S	Fonseca

Enrollment			
name	**title**	**semester**	**grade**
Eliza	CS33	2014F	A
Eliza	CS127	2014F	A
Ben	CS127	2012F	A
Carl	CS195	2013F	C
Carl	CS127	2014F	B

The keys for each relation are as follows:

- *Student:* name (all student names are assumed to be unique)
- *Course:* title and semester
- *Enrollment:* name, title, and semester

Answer the following questions:

1. Suppose we create a B-tree index for the *name* attribute of the *Student* relation. Assuming that 3 index records and 4 pointers fit into a block, draw a picture of the B-tree that results from inserting records for students Ben, Amy, Meg, Jim, Sue, Carl, Mary, Bob, Dan, Alex, Tim, Joe, Jill, Eric, and Eliza. Names should be sorted lexicographically.

2. Suppose instead we create a hash index for the *name* attribute of the *Student* relation. Consider the following hash buckets, each with a maximum size of 3 elements:

catalog	**buckets**
00	Amy, Dan, Eliza
01	Jim
10	Ben, Carl
11	Gary, Max

   Use the following formula to convert a name with n characters to a numeric value as the basis for the hash function: $\sum_{i=1}^{n} name[i]$. The value of each character is its alphabetic offset, as given by the following table:

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

   For example, the numeric value for Amy would be: $0 + 12 + 24 = 36$.

   Add the students Tom, Mike, John, Kyle, and Jen (in this order) using the following strategies:

   a. Use **extendible hashing** to add the students. The buckets you start with have a global depth of 2, and each individual bucket also has a local depth of 2. If multiple entries in the directory point to the same bucket, be sure to indicate it in your solution. Please illustrate the full state of your hash buckets after each addition, including changes to the catalog and any change to global or local depth.

   b. Use **linear hashing** to add the students. Note that the catalog value is used purely for illustration, but you should maintain the values in your solution. The *next* pointer starts by pointing at the first bucket (labeled **00**) and $N = 4$. Please illustrate the full state of your hash buckets after each addition, including where the *next* pointer is pointing and additions to the catalog values.

3. Compare B-tree and hash indexes within the context of this database. For which types of queries on the *Student* relation would each be preferable?