

Assignment #2

Due: 2:00pm CST on September 20, 2006

Out: September 8, 2006

NOTE: Please submit a hard copy of your homework. Bring it to the lecture table at the beginning of the lecture on 20th September, 2006.

The hard copy should be as clearly readable as possible. You may be subtracted points for unreadability and ugly presentation.

I2CS Students: You should email your solutions to the TA(ylee11@uiuc.edu) in the **pdf** or the **word** format. Please send the file as attachment with your email by 2PM UIUC time (CST). I2CS students in other time zones should note that the deadline is according to CST.

Problem 1 (20 pts)

Suppose that you want to index fixed-length records, each 256 bytes long. The key of each record is 4 bytes long. Records are stored on disk so that they do not cross page boundaries. If you use a three-level B+ tree to index this file, with the records themselves occupying the third level of the tree, then what is the maximum number of records that you can index if the B+ tree pointers are 4 bytes long and the B+ tree nodes are 4096 bytes long?

Problem 2 (30 pts)

In the development of access methods, for tree-based indexing techniques, we have seen B-tree, R-tree, and then GiST, among others. This series of trees represents a sequence of generalization.

Describe what generalizations exist in this series of trees. That is, how does R-tree generalize B-tree? How does GiST generalize R-tree?

Problem 3 (50 pts)

This problem asks you to build a GiST search tree for the **TIME** data type. Each **TIME** object can be either an *exact moment* or a *period*. For simplicity, we assume each **time** is in *military* format. The following are some example **TIME** objects.

```
t1 = 1145           // an exact moment example
t2 = (1130, 1300)  // a period example
t3 = (0400, 0530)
t4 = (1400, 1330)
```

2

We want to use GiST to build a search tree for supporting the following *query predicates*, in which x and y are two **TIME** objects. That is, given a predicate below, we want to search the index tree, for some given y , to find all the x that satisfy the predicate.

- *Overlap*(x, y): *True* if x and y overlap, and *False* otherwise; e.g., *Overlap*(t_1, t_2) = *True*, *Overlap*(t_1, t_3) = *False*.
- *Before*(x, y): *True* if x ends before y starts, and *False* otherwise; e.g., *Before*(t_3, t_2) = *True*, *Before*(t_1, t_2) = *False*.

Describe how *Consistent* can be implemented for each of the above query predicates.