

Computer Science 321
Database Systems
Fall 2003

HW 6 Due: November 24, 2003 Total weight: 140 pts.

1. [40pts] Exercise 12.2 from the textbook (latest edition). Justify your answers.
2. [100pts] Implement the bulk loading algorithm for which you have already written the pseudocode in a previous homework. Your program may be written in C/C++ or Java (whichever you are more comfortable with). Your program must compile and run under the compilers installed on the departmental machines (i.e., if you develop your program on a different machine or environment, make sure to “port” it so that the grader can compile it on the departmental machines). When you are ready to submit, send an email to the grader with the file(s) you need, including a README file stating precise instructions for saving (i.e, filenames), compiling, and running your program. In class, hand out to me a hardcopy of what you emailed.

The input file to your program contains sorted data pages. Each line of the input file corresponds to a page. For simplicity, you may assume that every line of the file contains exactly two integers, thus two records (in practice, a record would contain several more fields). The input file has the form:

```
5 11\n
12 99\n
...
900 971\n
```

Your program must read from standard input and bulk load the data. For this project, assume that d is set to the constant 3. Every node in your tree must have a natural number id . Once you have built your B+ tree, you must output it (to standard output) to show that your algorithm is in fact working correctly, printing the id 's to reference nodes, instead of pointers.

The following shows a basic recursive pseudocode algorithm for printing out your tree, including a struct/class pseudocode definition of a node. This node is intended to be used as both an index node and a leaf node (again, this is to simplify your code with respect to pointers). Feel free to deviate from the node structure below. However, printing must be logically implemented as shown.

```
struct/class node{
    int type;//leaf or index node
    int entries[];
    int childid[];
    node* childptrs[];
};

Printindex(node* currentnode) {
    if (currentnode.type == leaf) {
        print "L id : ";
        print "every entry\n";
    } else { //currentnode is an index node
        print "I id : ";
```

```

        print "childid[0]";
        for (i=1; childptrs[i]!=NULL; i++) {
            print "entry[i-1] ";
            print "childid[i] ";
        }
        print "\n";
        for (i=1; childptrs[i]!=NULL; i++) {
            Printindex(childptrs[i]);
        }
    }
}

```

For each index node, the output should be

I id : childid[0] entry[0] childid[1] ... entry[n-1] childid[n]

For each leaf node, the output should be

L id : entry[0] entry[1]