

## Homework 4

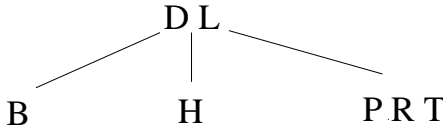
November 8, 2005

Due Date: November 15

1. (5 points) Suppose that you have an application in which you want to use B-trees. The computer you will be using has disk blocks holding 2048 bytes, each key is 4 bytes long, each child pointer (which is a disk block id) is 4 bytes, each data record reference (which is a disk block id along with a offset within the block) is 8 bytes. Finally, each node holds an integer (which takes 4 bytes) that holds the number of keys.

What value would you select for  $t$ ? (Show how you derived it.) You have an application in which you want to store 100,000,000 items in your B-tree. What is the maximum number of disk pages that will be brought into main memory during a search? Remember that the root is kept in main memory at all times.

2. (20 pts) Consider the following B-tree where  $t = 2$ . Parts (a), (b), and (c) will use this B-tree as its starting point.



- (a) (5 pts) Show the B-tree that results from inserting Y, F, X, Z (in that order) into the above B-tree. Circle (or in some way clearly mark) the B-tree obtained after each insertion is completed.
- (b) (10 pts) Show the B-tree that results from deleting B, then D, and then H from the B-tree shown above.
- (c) (2 pts) Show the legal way to represent the B-tree shown above as a red-black tree where “L” is in the root. You can indicate the color of each node by circling it with red or black or just by putting a “r” or “b” next to it.
- (d) (3 pts) Show the red-black that results when you insert “Z” into your answer from part (c). (If you did part (c) correctly then no rotations should occur here.)
3. (10 pts) Let array  $A = \langle 2, 5, 4, 8, 12, 7, 6, 10, 9, 14 \rangle$  be a “min-oriented” binary heap.
- (a) (1 pts) Show this binary heap in tree form.
- (b) (6 pts) Show the state of the binary heap after inserting 1, then 3, and then 2. Please show the state of the binary heap (in tree form) after each insertion. You can show an intermediate steps if you’d like, but it is not necessary.
- (c) (3 pts) Show the binary heap (in tree form) that results when performing an **extractMin**.
4. (15 pts) Describe an  $O(n \log k)$  algorithm to merge  $k$  sorted lists  $L_1, \dots, L_k$  (sorted in increasing order) into one sorted list, where  $n$  is the total number of items in all the lists combined.
- Big Hint:* Think about the merge step of mergesort and how you can use a binary heap. Be sure to clearly describe what you will use as the key and associated data.

5. (50 points) For the following two parts you are to pick a data structure that is best suited for the problem. The following components should be clearly given in each of your solutions. HOWEVER, YOUR SOLUTION FOR EACH PART MUST BE AT MOST 1 PAGE. It is important that you learn how to concisely describe your solution to this type of a problem. For this problem, either write very legibly or type your solutions (using at least an 11 point font and reasonable margins.)

- Describe your data structure choice(s) including all decisions such as what is used as the key, what is the associated data, ... This should only take 2-3 sentences.
- Briefly describe how to perform each requested operation. Do NOT give an code or even pseudo-code. You only need to describe any variations of standard methods that you need. Then just say what methods are used.
- Analyze the time efficiency for each operation. You need not repeat the analysis for any standard data structure method – just state what it is. You'd only need to analyze any modified methods (and even there you can assume the reader is familiar with the standard analysis).

(a) (20 pts) Consider a simple key frame animation system where  $n$  is the number of frames stored in your data structure. You are to support the following methods.

- `insertFrame(frameID, frame, timeInMovie, artist)` where `frameID` is a unique id for the frame, `frame` is an image, `timeInMovie` is the offset in seconds that the frame will be shown in the movie, and `artist` is the name of the artist who designed the frame. You should assume that the frames are not inserted in the order in which they appear in the movie. This method should run in worst-case or expected-case  $O(\log n)$  time.
- `produceMovieSegment(startTime, endTime)` produces a list of all frames, sorted by time, with times between `startTime` and `endTime`, inclusive. This should run in worst-case or expected-case  $O(k + \log n)$  time where  $k$  is the number of frames in the given time range.
- `getFrames(artist)` should return an iterator for a list of frameIDs (in any order) for all frames produced by the given artist. You can assume the artists have unique names. This method must run in worst-case or expected-case  $O(1)$  time.

(b) (30 pts) You are to maintain a multimedia document with a set of  $n$  video segments that are scheduled to play where  $n$  is about 100,000. Each segment  $s$  consists of a beginning time  $b$ , an ending time  $e$ , and a reference to a video object  $v$ . You should assume the the video segment is large and must be stored on disk. (So the video segment location will be stored as a disk location.) You must support the following methods.

- Given a new segment  $s = (b, e, v)$  determine if  $s$  overlaps any segment currently scheduled to play. This should run in worst-case or expected-case  $O(\log n)$  time.
- Insert a new segment  $s = (b, e, v)$  into the schedule if it does not overlap any currently scheduled segments. (If it does overlap any segment then just report that it cannot be inserted). This should run in worst-case or expected case  $O(\log n)$  time.
- Remove the segment with the earliest beginning time, returning the location (on disk) of the associated video segment. This method must run in worst-case or expected-case  $O(1)$  time.

**Challenge Problem: (5 points)**

A common implementation of sequential files on disk has each block point to its successor, which may be any block on the disk. This method requires a constant amount of time to write a block, to read the first block in the file, and to read the  $i$ th block, once you have read the  $i - 1$ st block. Reading the  $i$ th block from scratch, requires time proportional to  $i$ . Show how by adding just one additional pointer per node you can keep all the other properties, but allow the  $i$ th block to be read in time  $O(\log i)$ .