

Homework 4

March 23, 2006

Due Date: April 4

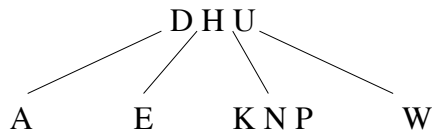
The material needed for 4(c) and 4(d) will be covered on Tuesday March 28. The material for problem 6 will not be covered until Thursday March 30. However, problem 6 will be an easy problem and can wait until then. You will be implementing a binary heap for Lab 4 and I want to be sure everyone understands how they work before you start the lab.

- (10 pts) Give a high-level description of the most efficient algorithm you can to take the union of skiplists S_1 and S_2 . (S_1 and S_2 can be destroyed by this method.) Analyze the expected time complexity of your algorithm when there are n_1 items in S_1 and n_2 items in S_2 . Recall that the expected height of a tower is $1/(1-p)$.

- (5 points) Suppose that you have an application in which you want to use B-trees. The computer you will be using has disk pages holding 2048 bytes. Each key is 4 bytes long, each child pointer (which is a disk page id) is 4 bytes, each data record reference (which is a disk page id along with an offset within the page) is 8 bytes. Finally, each B-tree node also includes the number of keys held in it (an integer which is 4 bytes). What value would you select for t ? Show how you derived it.

You have an application in which you want to store 1,000,000,000 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.

- (5 pts) Show the B-tree of order 3 (i.e. $t = 3$) that results when inserting the letters “N,E,W,A,L,G,O,R,I,T,H,M,S” in that order. You are required to show the B-tree just before each split occurs.
- (20 pts) Consider the following B-tree with a minimum branching factor of $t = 2$. For each part, you must show and clearly mark the B-tree that results after each requested insert or delete method has completed. Feel free to show intermediate steps so that we can give partial credit if you’ve made an error.



- (4 pts) Show the B-tree that results from inserting J and then M into the above B-tree.
 - (8 pts) Starting with the ORIGINAL B-tree shown above, show the result from deleting A, and then deleting H, and then deleting W.
 - (3 pts) Draw a legal red-black tree that corresponds to the ORIGINAL B-tree shown above.
 - (5 pts) Insert C and then L into the red-black tree from part (c).
- (50 points) The following components should be clearly given in your solutions to parts (a) and (b). *Keep your solution to at most one page..* It is important that you learn how to concisely describe your solution to this type of a problem. Either write very legibly or type your solution.
 - 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) (25 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.
- (c) (5 points) How would you change your design for the application described in part (b) if $n = 1,000,000,000,000$?
6. (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. You must show the binary heap (in tree form) after each insertion. You can show intermediate steps for partial credit if you make an error.
- (c) (3 pts) Show the binary heap (in tree form) that results when performing an `extractMin`.

Challenge Problem: (5 points)

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