Scheduling: 2-3$^+$ TreeProject

Assignment:

For this project, you will implement a seminar database and search system. The idea is that organizers of training seminars will be able to post training seminar descriptions in the database, and users will be able to locate seminars based on keyword, location, time, etc. In some ways, this project is similar to your first project, except that you will use a 2-3$^+$ Tree instead of a BST to provide access to the seminar records.

The 2-3$^+$ Tree:

For information about the 2-3 Tree, read Section 10.4 in the textbook. However, the data structure you actually will implement is the 2-3$^+$ Tree. A 2-3$^+$ Tree is a modification to the 2-3 Tree that stores pointers to records in the leaf nodes, and stores key values only as direction finders in the internal nodes. Thus, internal nodes are quite different from leaf nodes in their structure. Section 10.5 in the textbook discusses B-trees in general (B-trees are the generalization of the 2-3 Tree) and the B$^+$-tree in particular.

Seminar Records and Searches

A seminar record contains the following fields:

- Title: A string
- Date/Time: A string in the format YYMMDDHHmm where YY is the last two digits of the year, MM is the month, DD is the date, HH is the hour (24-hour clock) and mm is the minutes.
- Length: An integer (minutes).
- Keywords: A list of keywords. Keywords are strings.
- Location: Two unsigned short integers representing the X and Y coordinates (short integers are two bytes each).
- Description: A string.
- Cost: An integer (whole dollar amount).
- ID: An integer that uniquely identifies the seminar.

Users can search by a range of dates, by keyword, by location (within a given range) and by a range of costs. To support these possible search queries, you will index the database of seminar entries using one k-d tree and four 2-3$^+$ Trees. The k-d tree will index records by location. One of the 2-3$^+$ Trees will index records by cost. One of the 2-3$^+$ Trees will index records by date. One of the 2-3$^+$ Trees will index records by keyword. One of the 2-3$^+$ Trees will index records by ID. A record may contain multiple keywords. It will appear in the tree once for each keyword. Thus, if a given seminar has the three keywords "Database" "Woodworking" and "Accounting" then nodes with pointers to that seminar will appear three times in the keyword 2-3$^+$ Tree, once for each keyword. Any given keyword search might (and typically will) generate multiple matching seminars.

Input and Output

There will be no input parameters to the program. Your program will read from standard input (`stdin`) and write to standard output (`stdout`). The input for this project will consist of a series of commands (some with associated parameters, separated by spaces), one command for each line. No command line will require more than 80 characters. Commands are free format in that an arbitrary number of additional spaces may be interspersed between parameters, and blank lines may appear between commands. You do not need to check for syntax errors in the command lines (although you **do** need to check for logical errors such as illegal duplicate insertions or deletions of records with non-existent keys).

Each input command should result in meaningful feedback in terms of an output message. Each input command should be echo'ed to the output. In addition, some indication of success or error should be reported. Some of the command specifications below indicate particular additional information that is to be output.

Commands and their syntax are as follows.

insert *ID*
<title>
<date/time> <length> <x> <y> <cost>
<keyword list>
<description>

An insert command spans multiple lines. There will be no blank lines within an insert command. No line will require more than 80 characters. There can be multiple keywords on the keyword line, but there is only one line of keyword (thus, the keyword list is terminated by the newline symbol). Keywords may contain upper and lower case letters and underscores. The description itself can be an arbitrary number of lines long, and will be terminated by a blank line (which might contain spaces that should be ignored). Be sure to concatenate the description together, preserving the spacing and line breaks. It is an error to attempt to insert a record whose ID duplicates that of an existing record in the database. Such inserts should be ignored.

search

There are multiple forms of the search command, as follows:

search date *<date/time> <date/time>*

Print all records that fall within a range of date/times (inclusive).

search keyword *<keyword>*

Print all records that match the keyword.

search location <*x*> <*y*> <*radius*>

Print all records that fall within *radius* distance of the search point (inclusive).

search cost <*low*> <*high*>

Print all records that fall within the range (inclusive).

delete <*ID*>

Delete the record with the given *ID* from the database and all of the indexing trees.

dump date
dump keyword
dump location
dump cost
dump ID

Depending on the modifier to the **dump** command, print out a traversal of the corresponding tree. Use indentation and an appropriate traversal so as to be able to see the structure of the tree. For each seminar record, print out the relevent key (for the tree being dumped) and the seminar ID.

Implementation:

You should store only a single copy of each seminar record, with pointers to the record from the various tree indices. You should **not** use parent pointers in your 2-3$^+$ Tree impelementation. 2-3$^+$ Tree nodes should be implemented with an abstract base node class and two subclasses, one for leaf nodes and one for internal nodes. You will probably want to store the keyword list for each record as a linked list.