P.L.B. Sampath – 100476M

CSE 2010

Programming Project - Data structures and Algorithms

References

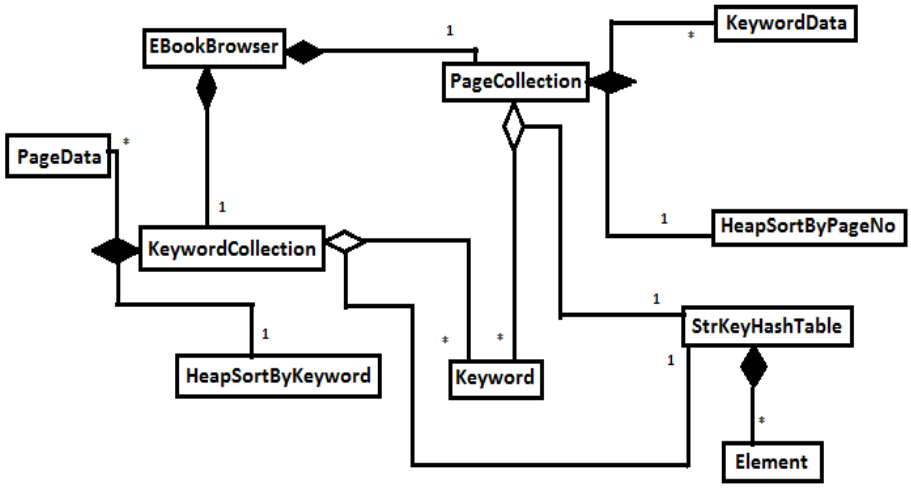
Instruction to Algorithms Book

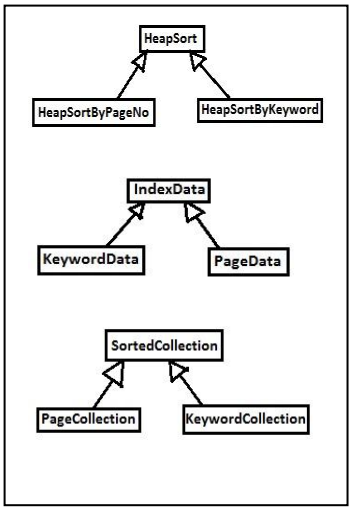
Eclipse Java IDE

Project Report

a) Design of the solution

Solution was implemented using Java. The class relationships of the solution are given below.





Steps of the program

- Reading the input file.
- Counting number of keywords in the input file (n).
- Creating two similar arrays of Keyword objects with a size of n+1 and adding a Keyword object to array for each keyword-pageNo pair of input file to the array. (types of both keyword and pageNo are String)
- Sorting keywords of one array using heap sort taking 'keyword' as the primary key (ignoring case) and 'pageNo' as the secondary key. Index i to i+4 of the array will look like below (5< i+4 <=n)

index	i-1	i	i+1	i+2	i+3	i+4	…
keyword	keyword0	keyword1	keyword1	keyword1	keyword2	keyword2	…
pageNo	2	1	2	3	1	2	…

Figure 1

- Counting no of distinct keywords (m). (from i to i+4 above, no of distinct keys=2)
- Creating KeywordData objects to store starting index and ending index of each distinct keyword with size of m. from i to i+4 for above array objects will look like below, where (i+3 <= k <= n+1)

KeywordData object	kwData1	kwData2
keyword	keyword1	keyword2
starting index	i	i+3
ending index	i+2	k

Figure 2

- Storing KeywordData objects in a hash table that uses linked list for collisions.
- Creating another hash table for PageData objects Using the same way for the other array taking 'pageNo' as the primary key and 'keyword' as the secondary key.
- Getting queries from the input file.
- Executing queries by getting data from hash table and printing output.

  Ex:- if Query is "List keyword1" program search for key "keyword1" of the hash table and gets a KeywordData object(kwData1 in this case) from the hash table. As i is the starting index and i+2 is the ending index, program prints pageNos of the array shown in 'Figure 1' from indexes i to i+2. (1 2 3).

  Other operations are executed in the same way with little differences.

b) Algorithms Data Structures used

1. Heap Sort

As this program needs a sorted output, sorting is an essential part of the program. Sorting can be implemented either when taking input from the text file or when giving output. As this is an EBook most of times when a user opens the EBook once, the user gets multiple outputs. Because of that the best choice is at the beginning.

There are many sorting algorithms. Among those algorithms Merge sort and Heap sort are two of algorithms that give better asymptotic running times o(nlog(n)) for worst case and average cases. Heap sort sorts in place and Merge sort doesn't. Because of that considering memory efficiency Heap sort was chosen. However I have implemented a difference algorithm changing comparison procedure of the original one to consider two keys, 'pageNo' and 'keyWord', as asymptotic running times don't change.

2. Array

Firstly after reading the input data is stored in two arrays because heap sort is to be used to sort and an array is the best for it.

3. Hash Table

Storing data in memory is another thing that we should implement. As there can be lots of keyword as the input we should store data in a data structure that gives us a better asymptotic running time for searching. A linked list or a binary search tree are not suitable because there is already a sorted array and searching in the array can be implemented with o(logn) running time. A hash table is used for storing keywords that gives θ(1) running time in average case. Hash function was chosen considering no of characters, difference of character and deference of the positions in a string used as a key. As number of slots in the hash table is equal to number of distinct keys here, α=1 for the hash table. Because of that I chose chaining for collisions believing on my hash function not to give the worst case ever practically.

4. Other algorithms

Some other algorithms were used to do general things like reading input. Those algorithms were implemented with θ(n) asymptotic running time or less in worst case.

C. Assumptions

1. The format of the input text file will be given 100% accurately as given in instructions. (if not nothing will be the output or run time errors may occur)

2. Sufficient amount of memory is available for the program, that is θ(n)+θ(m) memory where n=number of keyword-pageNo pairs in input file and m=number of distinct pairs)

D. Problems faced

Thought keywords can be asymptotic, number of pages cannot be practically. So for storing page numbers can be done using only a single array or linked list. But the method used for storing keywords can be easily abstract to page numbers; a hash table was used for storing page numbers too.

E. How search performance can be improved in operation 1 and 2

If a universal hash function is used for mapping, as random key mapping comes to map keys, the performance of the hash table will improve. Also we can use red black binary trees for collisions that reduce the asymptotic running time for worst case to o(log(n)).