UNIVERSITY OF CRETE SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER ENGINEERING
DATA AND FILE STRUCTURES

3 the exercise

Delivery date: May 22, 2020 **The exercise is individual**

Linear Hashing (4 units)

Build the Linear Hashing class in **main memory.** The class supports the operations:

• Insert random key • Search
random key • Delete random key

The hash table initially has size **M=100** positions and each position has a capacity of **10** keys. The capacity of overflow pages is 10 keys. The **page split criterion** for data entry operations is defined at the beginning of the method operation and depends on the value of the "fullness factor" **u** and can be **u > 50%** or **u > 80%.** The **criterion for merging pages** (page merging) in deletion operations is defined as **u < 50%.**

Performance of Linear Hashing (4 units)

We will examine the performance of the method for different values of the number of keys N. You are given a file with 104 keys which you will use to construct the hash table. The hash table grows by 100 keys (incrementally): initially do N=100 entries (with the first 100 keys) and take the following metrics. Continue with the next 100 intakes each time and take measurements again. Continue the same until N=10000. For each N = 100, 200, 300 ···· 10000 count:

1. The average number of comparisons per key input.
2. The average number of comparisons per search over 50 keys randomly selected from the above keys. 3. The
average number of comparisons per deletion over 50 keys randomly selected from the above keys (in the course of the measurements and after some keys have been deleted, the table has a smaller size than provided in the introductions).

Construct the X-Y diagram where Y is the average number of comparisons and X is N. On the same diagram draw 3 different curves showing the dependence of the average number of comparisons per insert/seek/delete respectively for u > 50% insert operations and search and u < 50% for delete operations. Repeat the same experiment (three additional curves) for u > 80% for insert and search cases and u < 50% for delete operations. **Note that when the split criterion is u > 80% page merges may not occur during deletion.**

In the same chart and for each N count the average number of comparisons per search (above to the same 50 keys you used above) into a Binary Search Tree (get a ready implementation from the Internet).

Construct the same diagram for the two cases of the splitting criterion ie for u > 50% and u > 80% for insert operations and u < 50% for delete operations.

Documentation of Results (2 credits)

Comment out (separate text file) the performance of all methods and try to justify the performance of each method. Suggest possible improvements (not requested implement). **Pay special attention to the interpretation of the results and do not show only the values.**

Recommendations:

- Do not proceed with the implementation before you have a good understanding of the Linear method Fragmentation. •
There are many implementations on the Internet that **have a misleading title "Linear Hashing" (Linear Hashing) but it is not the method that taught in class. Any such method will not be graded.**
- The "Linear Hashing" method is available o In the course notes (p. 34)
  - o https://link.springer.com/referenceworkentry/10.1007%2F978-0-387-39940-9_742
  - o http://delab.csd.auth.gr/papers/LinearHashing2017.pdf
  - o The method was originally proposed by Witold Litwin in the year 1980 at http://infolab.cs.unipi.gr/pre-eclass/courses/db/db-post/readings/Litwin-VLDB80.pdf
- There are codes online and on the course website. Exercise is worded in such a way that you will need to understand and change the codes in order to get the desired results.
Otherwise there is a risk that the exercise will not achieve what is requested and be simple code copying. In this case the exercise does not is graded.
- **Your program must work for entering keys into a file that will give us when correcting the exercise or for a random key value. • Your program should print to the screen the values per operation (search, insertion or deletion) for any key file to be given to input or for any random key value given in the input.**
- Indicative output of running your application (see the material from the 1st tutorial on how to produce sorted output):

Machine Translated by Google

```
java mypackage.MyClass path_to_input_file.bin
```

Input size (N)	LH u>50% avg # comparisons per insertion	LH u>50% avg # comparisons per search	LH u>50% avg # comparisons per deletion	LH u>80% avg # comparisons per insertion	LH u>80% avg # comparisons per search	LH u>80% avg # comparisons per deletion	BST avg # comparisons per insertion	BST avg # comparisons per search	BST avg # comparisons per deletion
100	5.8 1.1 3.1 1.2 1.2 1.3		...	...	...	...	...	...	...
200			...	...	...	...	...	...	...
300			...	...	...	...	...	...	...
.....									
10000	4.5	1.4	...	...	...	...	...	...	...

<u>Deliverables</u>:

A compressed zip file containing:

- A 2-3 page report with the results you are asked for, i.e. a table for N=104 and the X-Y diagram that shows how the execution time depends on N. **Pay special attention to the interpretation of the results** and do not show only the prices. **The grade of the 3rd part (2 credits) depends on the interpretation.**
- A report describing in 1-2 pages how the code was made (ie for each question what is the general idea of the solution in 3-4 sentences), there are clear instructions translation by compiler and execution, what errors it has (if any, cases where it does not the program works, or cases where it does more than what it asks you to do exercise, what you used from ready-made programs or sources of information. Even point to sources on the WWW like Wikipedia or Stackoverflow (full address of relevant pages).
- The code contains brief comments that explain the implementation. Add comments to javadoc format at the beginning of each class and each method. Also javadoc comments before each member variable of the classes. And where required within the code your.
- In addition to the above, the exercises are graded excellent if:
  - o The zip is complete.
  - o The codes pass through the compiler and run normally and correctly in windows or Linux environment (Caution: You should use relative directories paths rather than absolute ones, which only apply to your computers).
  - o Your code works for any parameter value.
- Exercises are submitted electronically on the course website and not by e-mail.
- Copies (even of part of the implementation) are zeroed out.