Database Systems, WS 08/09
Information Systems Group
Saarland University

Prof. Dr. Jens Dittrich
Teaching Assistant: Josiane Xavier Parreira
Assignment 5

1 Linear Hashing

Linear Hashing technique is very flexible in the choice of what triggers a split. An example was given in the lecture, where a split depends on space utilization. An alternative is to perform a split whenever a new overflow page is added. Consider the snapshot of the Linear Hashing index shown in Figure 1. Assume that a bucket split occurs whenever an overflow page is created.

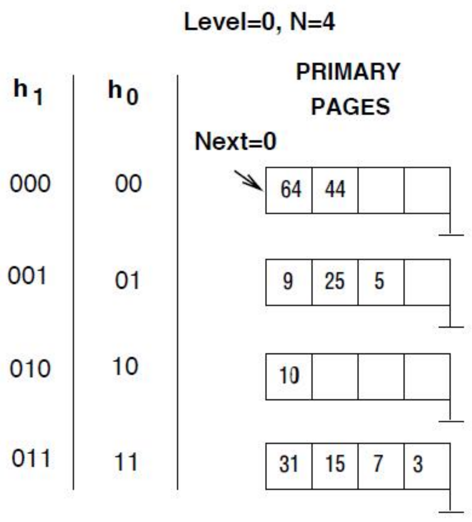

Figure 1: Linear Hashing index.

1. What is the maximum number of data entries that can be inserted (given the best possible distribution of keys) before you have to split a bucket? Explain very briefly.
2. Show the file after inserting a single record whose insertion causes a bucket split.
3. (a) What is the minimum number of record insertions that will cause a split of all four buckets? Explain very briefly.

   (b) What is the value of Next after making these insertions?

   (c) What can you say about the number of pages in the fourth bucket shown after this series of record insertions?

Solution Sketch

1. The maximum number of entries that can be inserted without causing a split is 6 because there is space for a total of 6 records in all the pages. A split is caused whenever an entry is inserted into a full page.
2. See Figure 2.
3. (a) Consider the list of insertions 0, 8, 24, 17, 33, 11, 16. The insertion of 24 leads to split of bucket 1. The insertion of 33 leads to split of bucket 2. The insertions of 11 and 16 cause the third and fourth split, respectively.

   (b) Since all four buckets would have been split, that particular round comes to an end and the next round begins. So Next = 0 again.

(c) There is always one data page in bucket 4 after these insertions. Depending on the last inserted value, this bucket can contain 3 or 4 elements.

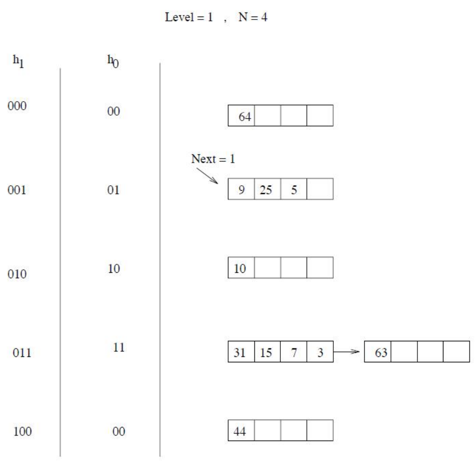

Figure 2: Linear Hashing index.

2 Grid Files, Linearized B+-trees and Extendible Hashing

Compare grid files, linearized B+-trees, and extendible hashing w.r.t.:

1. Point queries,
2. Range and window queries,
3. NN-queries.

Solution Sketch

	Point queries	**Range and window queries**	**NN-queries**
Grid files	Very efficient - usually only one page is from external memory.	Performance depends on the data distribution and grid cell size (granularity). Postfiltering is usually required. Note that higher number of dimensions leads to more pages being fetched, as well as more random I/O operations.	Efficient. Postfiltering is usually required. Note that higher number of dimensions leads to more pages being fetched, as well as more random I/O operations.
Linearized B+-trees	Efficient.	Efficient; postfiltering is usually required.	Efficient; postfiltering is usually required.
Extendible hashing	Very efficient - usually only one page is read from external memory.	Inapplicable - locality is not preserved.	Inapplicable - locality is not preserved.

3 Project: Main-Memory Efficient Tree-index

Keep working on your project. Deadline is Dec 1st, 23:59.

How to hand-in your code:

Please provide a single zip file. The zip-file should have the name of your group and contain the following:

(a) the entire source code of your implementation including the original B+-tree framework,

(b) a pre-compiled jar with your map implementation,

(c) a working shell script that executes the WorkloadGenerator using the map implementation contained in your precompiled jar,

(d) a README.txt or README.pdf briefly explaining what you implemented: core ideas, major modifications you did to the original B+-tree; if possible some comments on which change triggered how much improvement, do not provide more than a single A4 page.

You may either upload your zip to the repository or send an email with your zip to jens.dittrich@cs.