The PH-Tree: A Space-Efficient Storage Structure and Multi-Dimensional Index

Christofer Fabian Chavez Carazas

National University of Saint Augustine

January 2, 2017

Introduction

PATRICIA-hypercube-tree

Efficient data access.

Space efficiency

Compare to kd-tree

Bit-stream

- The bit stream is the representation of the data in each dimension using bits.

Example

1D Example:

10 to binary: 1010

Bit-stream: 1 0 1 0

2D Example:

(10,5) to binary: (1010,0101)

Bit-stream: 10 01 10 01

3D Example:

(4,8,1) to binary: (0100,1000,0001)

Bit-stream: 010 100 000 001

- w is the size of the bit-stream and the depth of the PH-Tree.

Prefix Sharing (PATRICIA-Trie)

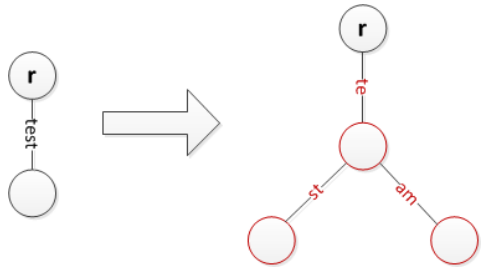


Figure: Insert "test" and later insert "team"

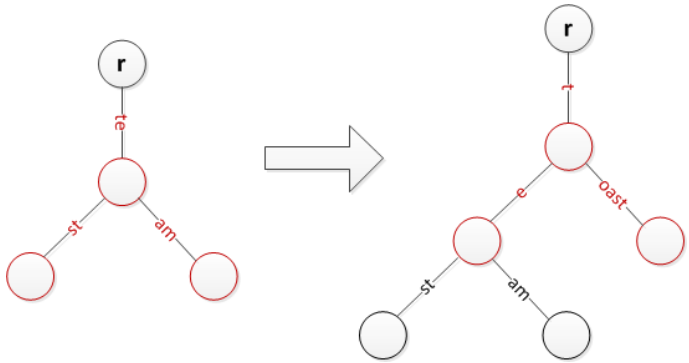


Figure: Insert "toast"

PH-Tree - Structure

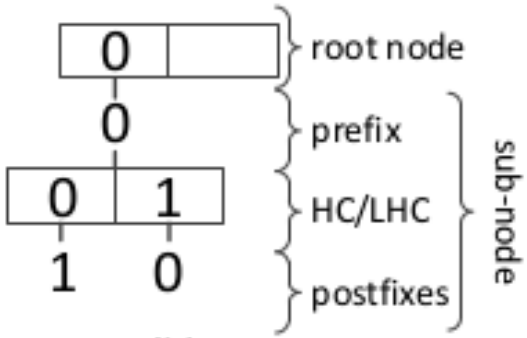


Figure: PH-Tree Structure

One node have:

- Prefix
- Hypercube
- Postfixes or sub-nodes

Hypercube

- The hypercubes allows navigation to subnodes and postfixes.

Figure: 1D and 2D Hypercubes

- The size of the hypercube is 2^k.

Insert

- We insert the point (5,9) : (0101,1001)

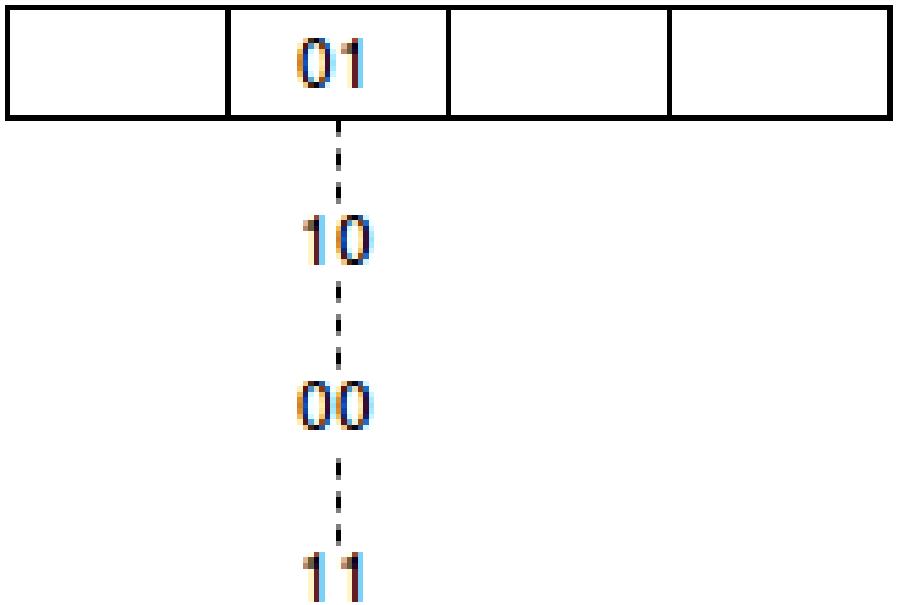


Figure: First insertion

Insert

- We insert the point (6,10) : (0110,1010)

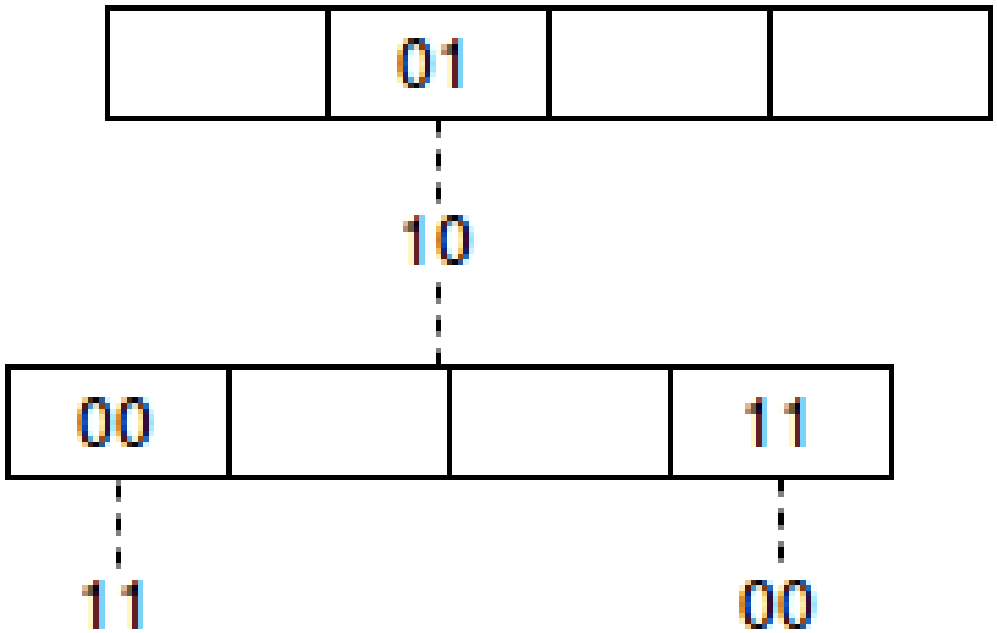


Figure: Second insertion

Insert

- We insert the point (6,8) : (0110,1000)

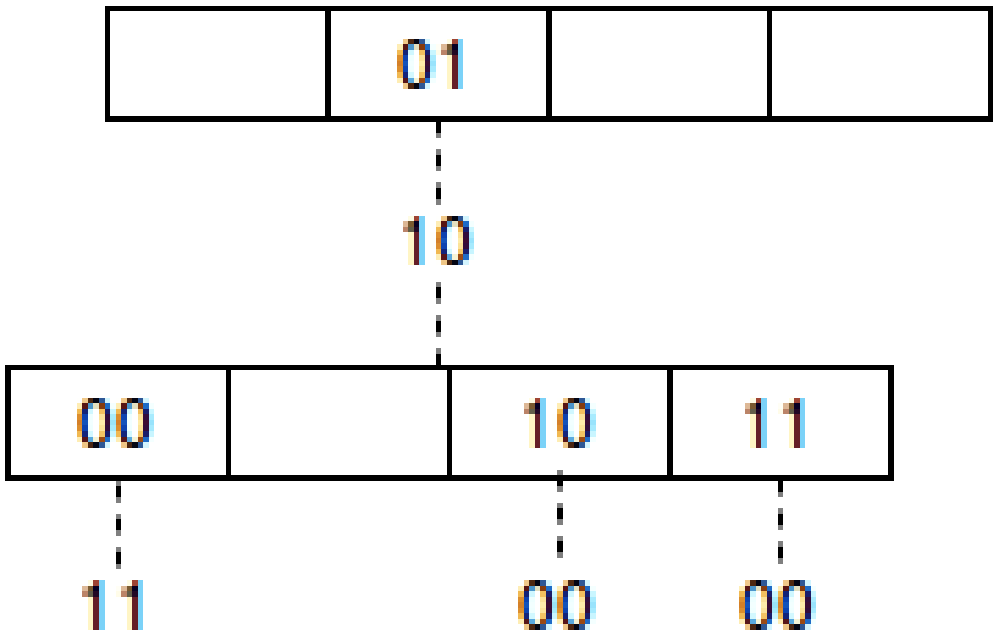


Figure: Third insertion

HC/LHC

- **HC (HiperCube):** An Array of references.
- **LHC (Linear HiperCube:** An a list of pairs that map <address in HC> -> <postfix/sub-node>

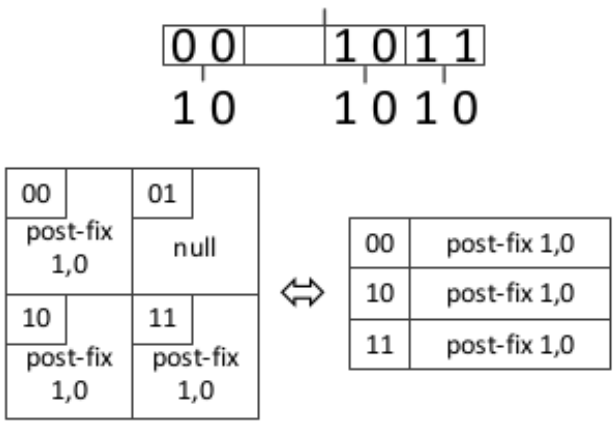


Figure: HC (left) and LHC (right) representation of hypercube (top) in a node

HC/LHC

Advantages and Disadvantages:

- **HC:** It requires only constant time operation to navigate to the sub-node or stored entry, but it wastes space.
- **LHC:** It saves up space, but it requires a binary search to navigate to the sub-node or stored entry.

Calculating the Space

- **HC:** $O(2^k)$ for the sub-nodes and $O(l_p * 2^k)$ for the postfixes
- **LHC:** $O(n_s * k)$ for the sub-nodes and $O(n_p * k * l_p)$ for the postfixes.

Where:

- k is the number of dimensions.
- l_p is the length of the postfixes in a node.
- n_s is the number of sub-nodes.
- n_p is the number of posfixes.

Point Query

Search the point (10,8) : (1010,1000).
Bit-stream: 11 00 10 00

Search the point (1,1) : (0001,0001)
Bit-stream: 00 00 00 11

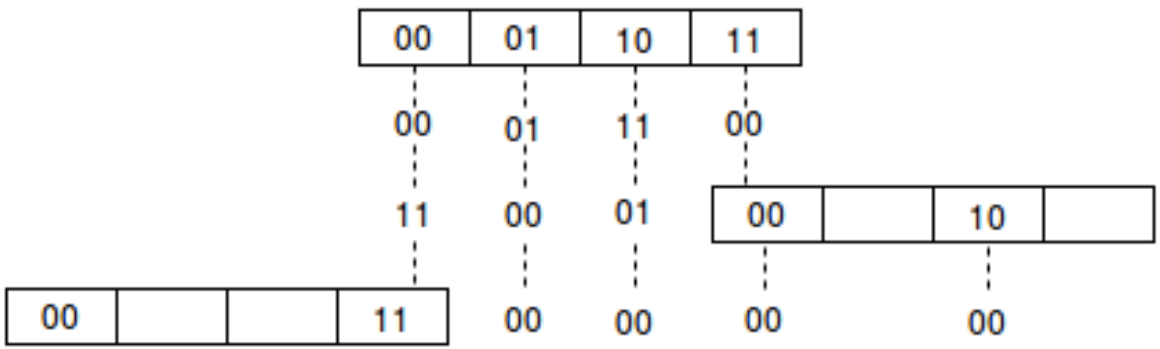


Figure: PH-tree example

Range Query

- Input: lower left point and uper rigth point.
- Output: Iterator.

1. Start with a point query that locates the starting node, defined as the lower left corner of the query range. Even if the point query fails, the last visited node is the starting point for the query.
2. Calculate two bit masks m_L and m_U
    1. m_L is set to 0 if the query range is less or equal to the lower left corner of the node in the dimension i.
    2. Inversely, any bit b i in m_U is set to 1 if the query range is equal to or larger than the top right corner of the node.
3. An HC address h fits if $(h|m_L) == h \;\&\&\; (h\&m_U) == h$
4. Continue iterating until there are not candidates.
5. If the candidate is a node, then repeat the step two.

Space Efficiency

Worst Cases

- Lack of prefix sharing. **HC:**$O(w * 2^k)$ **LHC:**$O(w * k * n)$
- Bad entry to node ratio ($r_{e/n} = n/n_{node}$)

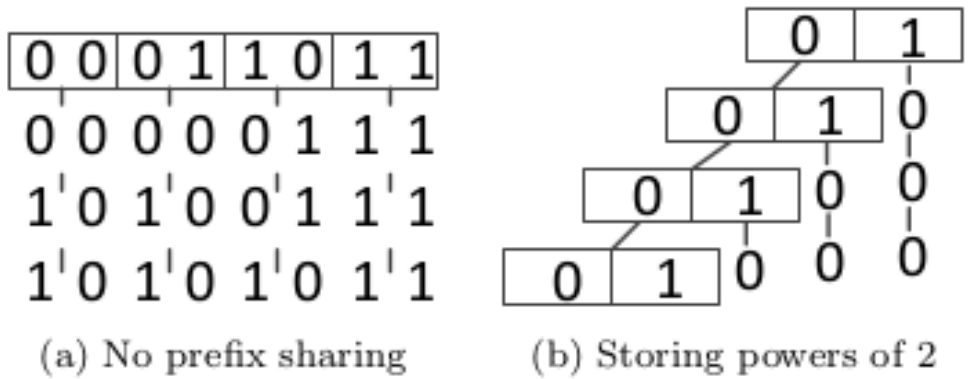


Figure: The two space worst cases of the PH-Tree

- The average case cannot be established as a simple function of n because it not only depends on the type of distribution of the data but also on the value range in each dimension.

Query Efficiency

Point Query

- Algorithm for interleaving: $O(w * k)$
- Query internal search: **HC:** $O(1)$ **LHC:** $O(log_2(2^k)) = O(k)$
- Prefix/postfix extraction: $O(w * k)$
- Putting everything together: **HC:** $O(w * k + w * 1 + w * k)$ **LHC:** $O(w * k + w * logk + w * k)$
- Resulting $O(w * k)$ for both approaches.

Range Query

- The worst case is a full scan of the nodes with $O(n)$

- The average complexity cannot be established as a simple function of n, because it depends on different characteristics of the data.
- The query complexity tends to vary between $O(logn)$ and $O(1)$ for low w

Update Efficiency

- Point query: $O(w * k)$
- Creation of sub-node: $O(1)$
- Encoding the new entry and copying an existing postfix from the parent node: $O(w * k)$
- Insert in the LHC: $O(w * 2^k)$
- Insert in the HC: $O(1)$
- Total cost of the LHC: $O(w * (2^k + k))$
- Total cost of the HC: $O(w * k)$

Experiment

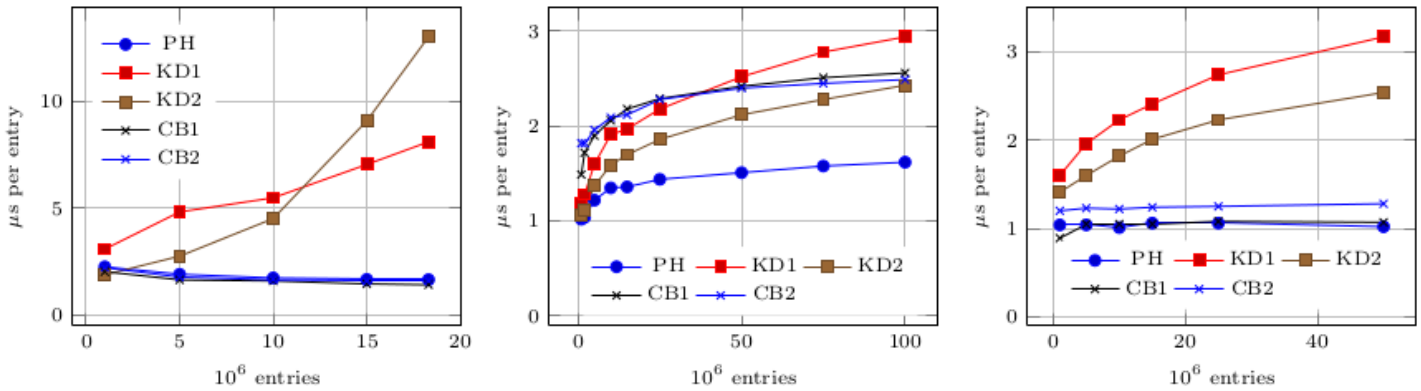
Set-Up

- PC with 32GB RAM and and Intel i7-3770K 3.50GHz GPU.
- They used two kD-tree implementations called KD1 and KD2.
- They used two critical-bit-trees(bit-wise PATRICIA-tries) called CB1 and CB2.

Datasets

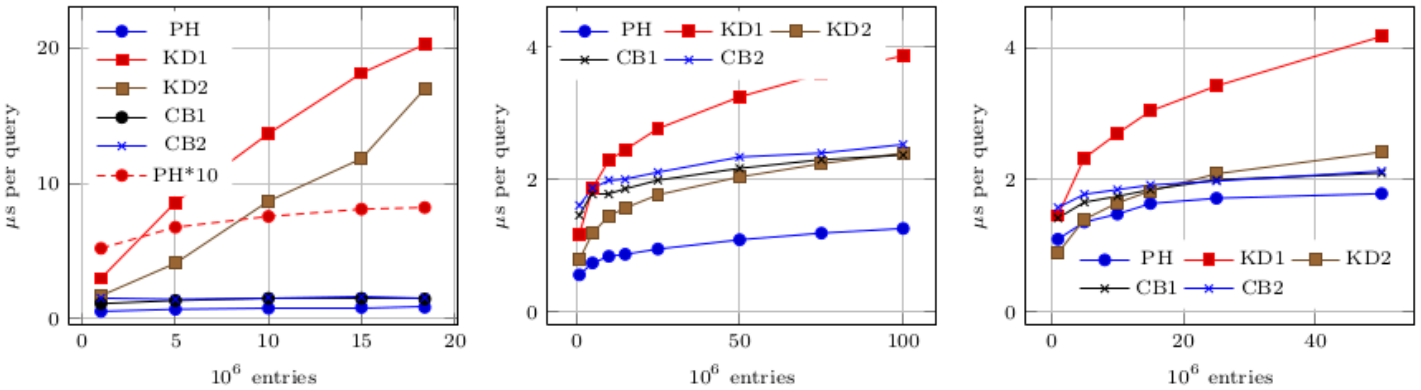
- **2D TIGER/Line:** It consists of poly-lines that describe map features of the United States of America. They extracted all points from counties belonging to mainland USA. $18.4 * 10^6$
- **3D CUBE:** is a set of up to 100,000,000 points distributed uni-formly at random between 0.0 and 1.0 and independently in every dimension.
- **3D CLUSTER:** The dataset consists of a line of 10000 evenly spaced clusters of points. In total, the CLUSTER dataset contains up to 50,000,000 points.

Experiment



(a) 2D TIGER/Line (b) 3D CUBE (c) 3D CLUSTER

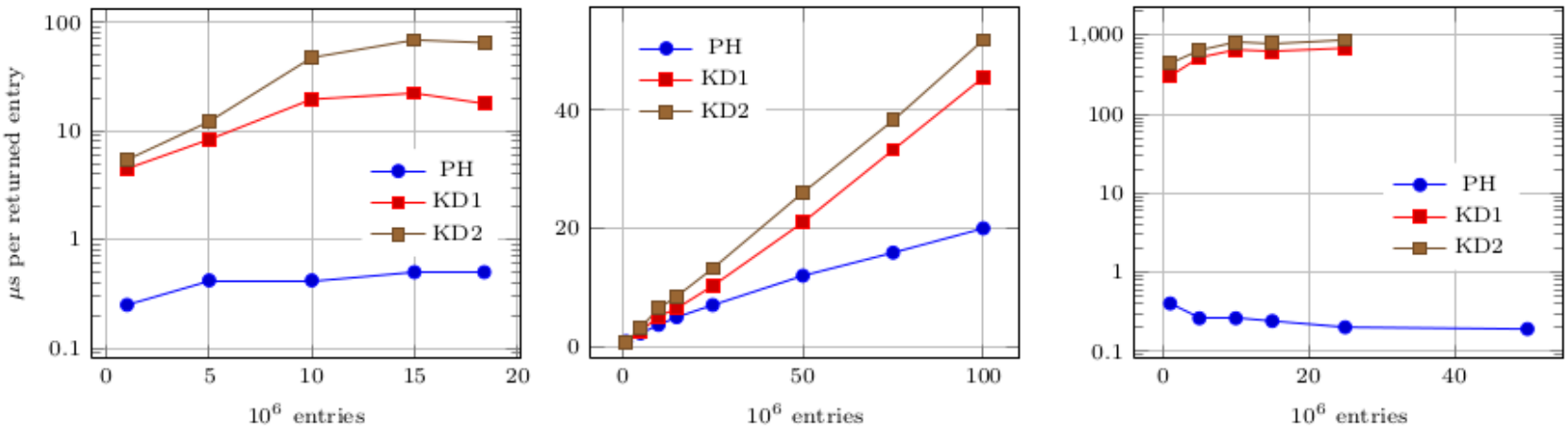
Figure 7: Insertion times per entry for the TIGER/Line, CUBE and CLUSTER datasets



(a) 2D TIGER/Line (b) 3D CUBE (c) 3D CLUSTER

Figure 8: Point query times for the TIGER/Line, CUBE and CLUSTER datasets

Experiment



(a) 2D TIGER/Line (b) 3D CUBE (c) 3D CLUSTER

Figure 9: Range query time for the TIGER/Line, CUBE and CLUSTER datasets

Bibliografa

Tilmann Zschke, Christoph Zimmerli, and Moira C. Norrie. 2014.

The PH-tree: a space-efficient storage structure and multi-dimensional index.

In Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data (SIGMOD '14).

ACM, New York, NY, USA, 397-408.

DOI: http://dx.doi.org/10.1145/2588555.2588564