FUSION TREES

Data structures II

Members:
Ali Muhammad Asad
Iqra Ahmed
Mohit Rai
Saira Junaid
Anoosha Hasan

PROJECT INTRODUCTION

The implementation of Fusion Trees

Background on Fusion trees

- The name Fusion Tree, is believed to be derived from from the "Cold Fusion Debacle", which was popular in the 90s, essentially a form of Fusion at Room temperature
- The fusion trees were developed, under the assumption that "We assume throughout this paper that the number of data items that are present never exceeds 2^w, the universe size. (To cope with a larger number of items, we could proceed by storing bucket pointers in our data structure, with equal items placed in common buckets.) ", as

Fusion Trees

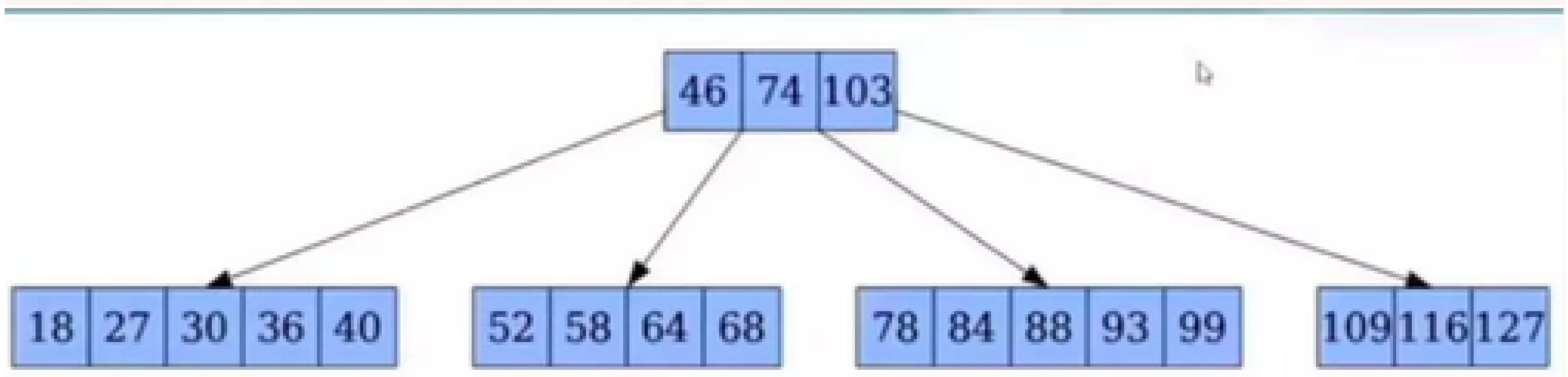
- A Fusion Tree is a modification of a B-Tree that implements an associative array on w-bit integers in a known universe size.
- An associative array is an abstract data type that stores a collection of pairs such as key-value pair.
- Fusion Trees are mostly used when our universe size is large, while providing linear - O(n) - space complexity, and O(logn) search time complexity making it faster than a traditional self-balancing tree.

Fusion Trees Complexities

- Height: O(logw(n))
- Search, Predecessor, Successor: O(logw(n))
- Insert: O(w^(2/3)logw(n))
- Delete: O(w^(2/3)logw(n))
- Space: O(n)

- Where n is the count of values, w is word size ie, the size of the data type used to store the values.

Data structure of Fusion Tree



- Complexity of going through one fusion tree node becomes O(1) thanks to "Parallel Comparison" of x with the keys in a node.
- Hence the search complexity of Fusion tree = O(log(t) n)

HOW DOES IT WORK?

How does it work?

- Essentially a B-Tree with a branching factor of w^1/5 giving it a height of O(logw(n)) where 'w' is the machine word
- Our assumption is that universe size <= 2^w
- Uses "Sketching" - fits keys all into one machine word allowing "Parallel Comparison" to be done.
- Helps achieve desired runtimes for updates and queries

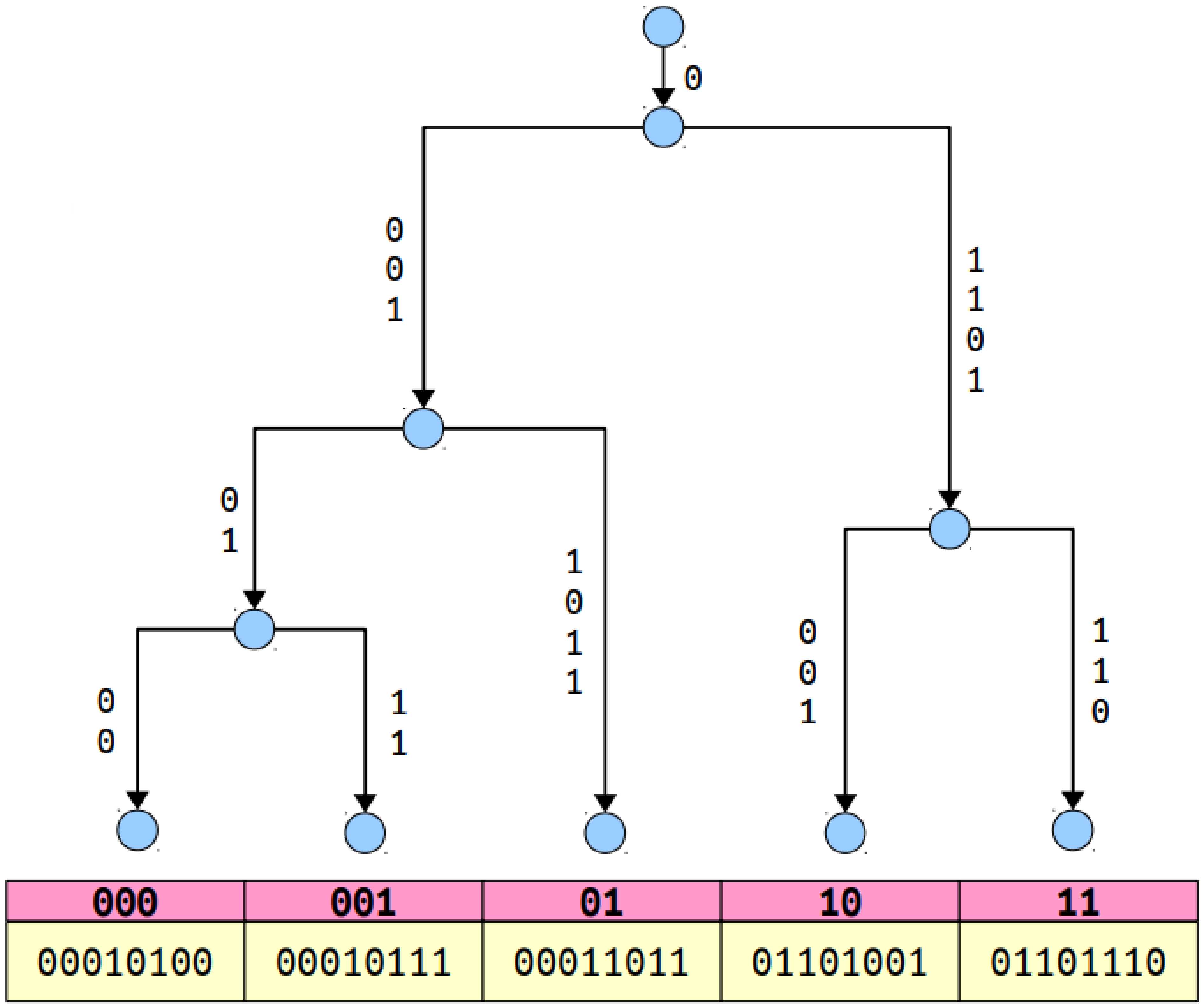
SKETCHING

Sketching

- Each w-bit key at a node containing k-keys is compressed
- Creates a path-like structure in a full binary tree of height w
- Max k-keys, so maximum k - 1 branching points (where two keys differ)
- Preserves order of the keys, that is sketch(x) < sketch(y) for any two keys in a node where x < y.

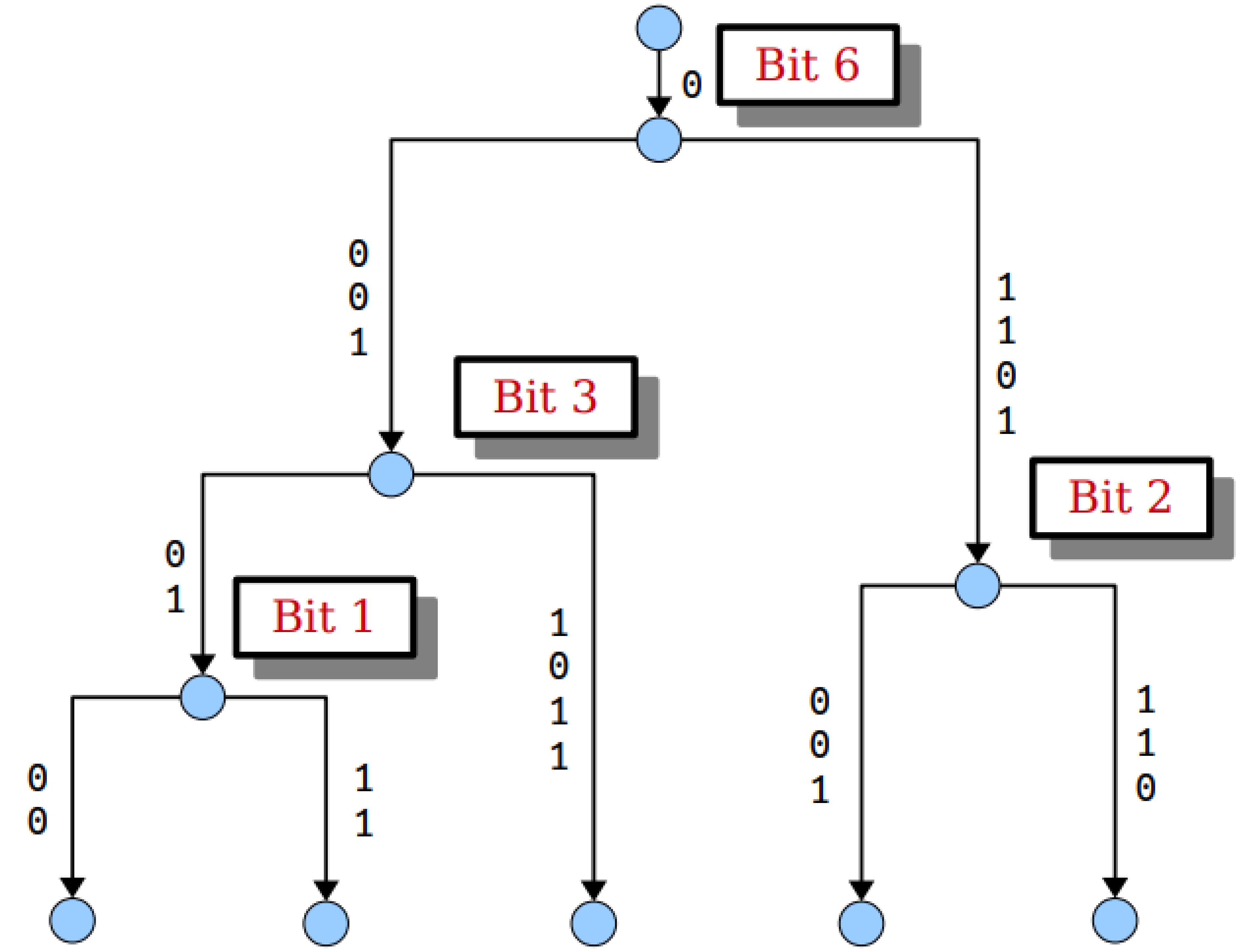
Sketching - Aims

- Ultimately we are interested in compressing our numbers so that they fit in a machine word.
- There at most w^e (epsilon) bits in each of these new numbers - that's really promising!



Sketching - Patricia Code

- A bit index *i* is interesting if there is a branching node in the trie at that bit index
- The *Patricia Code* of an integer is the bitstring consisting of just the interesting bits in that number

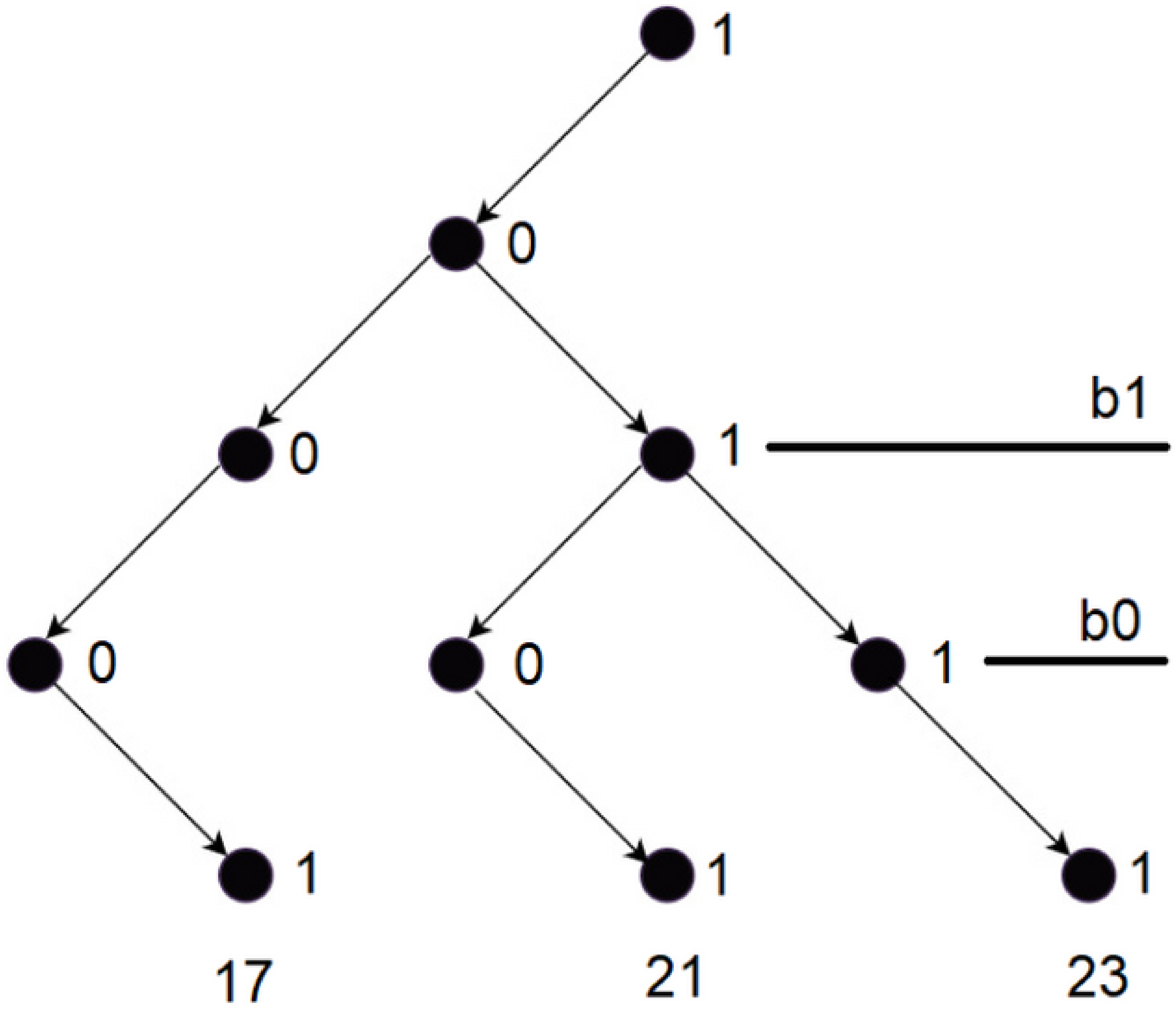


0010	0011	0101	1100	1111
00010100	00010111	00011011	01101001	01101110

Bringing it Together

sketch(21) = sketch(10101) = 10101 = 01
sketch(17) = sketch(10001) = 10001 = 00
sketch(23) = sketch(10111) = 10111 = 11

- Not possible to compare O(k) keys of a node in constant time
- Sketch compresses the keys for faster comparison of keys within a node
- Allows constant time for predecessor queries



PARALLEL COMPARISON

Parallel Comparison

- Finds the position of a value within a set of keys in a node in constant time
- Provides efficient storage, and retrieval of large sets of items
- Multiple keys are compared simultaneously at each level
- Uses bitwise operations for comparison of the binary representation of keys given by Sketching

Parallel Comparison

```
  01101110 00101110 01111000 01001101 00101111 00001101 01110111 01100001
+ 00011010 01000101 00010100 00100000 01010000 00100010 01000100 00001000
-------------------------------------------------------------------------
  10001000 01110011 10001100 01101101 01111111 00101111 10111011 01101001
```

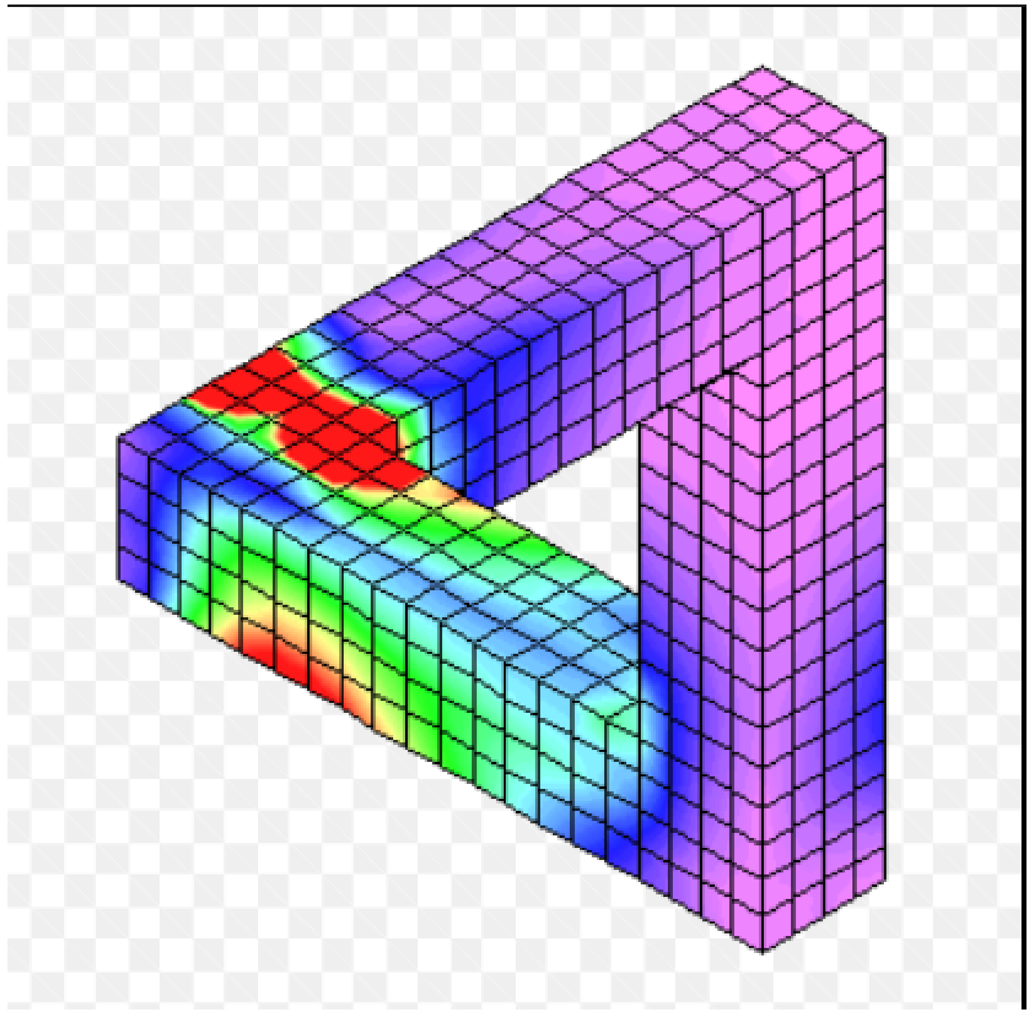
- Eight pairs of 7-bit numbers are compared using a single 64-bit addition
- The MSBs of each are summed to evaluate the appropriate positions

APPLICATION

A	B	C	D	E
		Database clients of Jolly Da		
№	Customer	Type	Country	City
1	Intersection	com.network	USA	New York
2	Magnet	com.network	USA	New York
3	Perspective korp.	warehouse	Belarus	Minsk
4	Driveway	enterprise	USA	New York
5	near	enterprise	USA	Los Angele
6	Nori	warehouse	Japan	Tokyo
7	Nevsky comp.	com.network	Russia	Moscow
8	Perspective korp.	enterprise	Belarus	Minsk
9	in touch	warehouse	USA	San Francis
10	Nardis	com.network	Japan	Tokyo

- **Databases Management Systems for faster insertion, deletion, and lookup**
- **Indexes for search engines involving numerical data**

- **Computational Geometry**
- **Machine Learning**
- **Geometric Problems like nearest neighbor queries in higher dimensions**



LIMITATIONS AND COMPLEXITIES

- Design: a unique combination of ideas and techniques from various mathematical fields - challenging implementation and understanding
- Bit manipulation and bitwise operations for lookup - predecessor and successor search
- Less efficient space complexity
- Complex insertion and deletion
- Non-intuitive design
- Limited practicality

REFERENCES

[1] Fredman, M. L.; Willard, D. E. (1990), "BLASTING Through the Information Theoretic Barrier with FUSION TREES", Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing (STOC '90), New York, NY, USA: ACM, pp. 1–7, doi:10.1145/100216.100217, ISBN 0-89791-361-2, S2CID 16367160.

[2] https://www.youtube.com/watch?v=xSGorVW8j6Q&ab_channel=MITOpenCourseWare

[3] https://youtu.be/QXASOE5scoM

[4] https://youtu.be/Jec59qLhkGU

[5] https://www.slideshare.net/RohithND/merkle-trees-and-fusion-trees

[6] https://stackoverflow.com/questions/3878320/understanding-fusion-trees

THANK YOU