





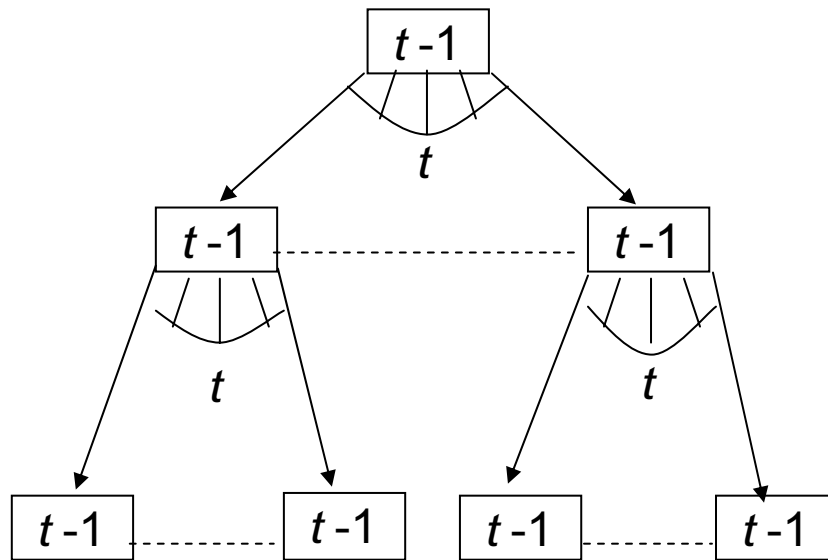
B-Tree Optimizations in free text search

Sarad AV,
Wireless Communication Research Group
AU-KBC Research Centre,
MIT Campus of Anna University.

- 
- B⁺ tree is a multi way decision tree under certain constraints and a variant of the B tree
 - Secondary disk storage provides cheaper mass storage support when compared to main memory devices
 - The requirement to storage large indices make main memory solutions economically infeasible
 - Secondary storage access time is usually at least 100,000 times slower than main memory
 - B⁺ tree is used to minimize I/O operations

- 
- Information is divided into equal sized pages stored are consecutive locations on disk
 - Page size is usually 2048 to 4096 bytes in length
 - Each disk access can read/ write one or more pages
 - B⁺ tree provides a robust solution to minimize disk access while manipulating the indices
 - Used in ReiserFS filesystem and XFS and other relational databases

- The minimum number of index element present in every node is restricted to k
- The order of the B⁺ tree is defined to be the maximum number of index elements to be present at every node of the tree with the exception of the root, represented as $2.k$, for an integer $k \geq 2$
- Branching Factor $t = 2.k + 1$ is the maximum number of branches a node can branch out
- The keys in an arbitrary node n are inserted in non-descending order



Depth ($h-1$)

No. of nodes (n)

0

1

1

t

2

t^2

For a complete B⁺ tree

$$n = 1 + t + t^2 + \dots + t^h = (t^{h+1} - 1) / (t - 1)$$

$$\text{Hence, } n \cdot (t - 1) + 1 = t^{h+1}.$$

Taking logarithms on both sides:

$$\log_t [n \cdot (t - 1) + 1] = h + 1$$

$$h = \log_t [n \cdot (t - 1) + 1] - 1$$

h = height

t = branching factor

n = No. of nodes

Max. number of elements = $(t-1).h$

Max. redundant nodes $n = 1 + t + t^2 + \dots + t^{h-1}$

$$= (t^h - 1) / (t - 1)$$

Therefore, max. number of non-redundant nodes

$$= [(t^{h+1} - 1) / (t - 1)] - [(t^h - 1) / (t - 1)]$$

= t^h . This is intuitively clear since all the leaf nodes are non-redundant.

Maximum number of non-redundant elements

$$= (t-1). t^h$$

A complete B-tree can store $(t^h - 1)$ more elements when compared to a complete B⁺ tree of the same size

A B-tree with index size l bits and satellite info. size l bits on a page size of p bits, gives a branching factor $t / (l+l)$.

A B⁺ tree stores the satellite information at the leaf nodes. Hence its branching factor is (t / l) , twice that of the B-tree.

- 
- The indexing and B⁺ tree data structure is explained in the following slides

F_0 = “Everyone likes Angelina_Jolie”

F_1 = “Angelina_Jolie is a Hollywood actress”

F_2 = “There is no one with the likes of Angelina_Jolie”

- Inverted index is used as a mapping from words to its corresponding file number(s)
- Fully Inverted index also holds the word’s position in the corresponding file number(s)

Inverted Index		Fully Inverted Index
“Everyone”	{0}	{(0,0)}
“likes”	{0,2}	{(0,1), (2,6)}
“Angelina_Jolie”	{0,1,2}	{(0,2), (1,0), (2,8)}
“is”	{1,2}	{(1,1), (2,1)}
“a”	{1}	{(1,2)}
“Hollywood”	{1}	{(1,3)}
“actress”	{1}	{(1,4)}
“There”	{2}	{(2,0)}
“no”	{2}	{(2,2)}
“one”	{2}	{(2,3)}
“with”	{2}	{(2,4)}
“the”	{2}	{(2,5)}

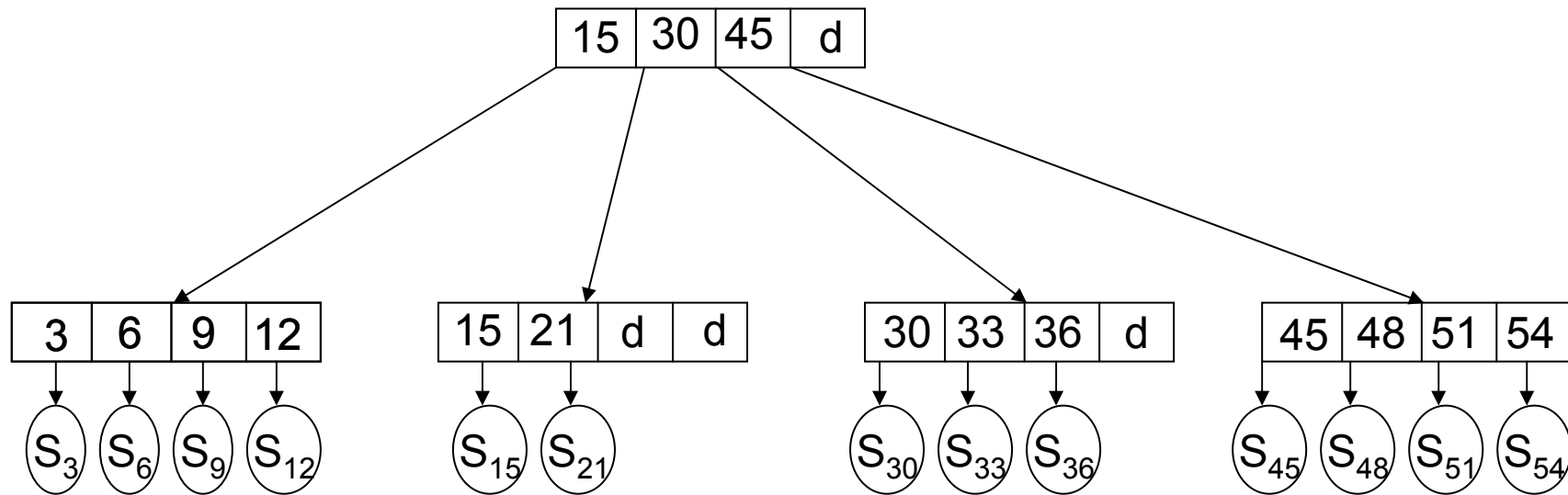
- A search for the words “likes” “Angelina_Jolie” from the inverted index and then computing $\{0,2\} \wedge \{0,1,2\} = \{0,2\}$ lists the files where the search string is found.

A search on the fully inverted index yields

“likes” $\rightarrow \{(0,1), (2,6)\}$

“Angelina_Jolie” $\rightarrow \{(0,2), (1,0), (2,8)\}$

- The search string appears consecutively in the correct order in (0,1) followed by (0,2) in F_0 . The occurrence of the search string in F_2 is (2,6) followed by (2,8) and do not appear consecutively.

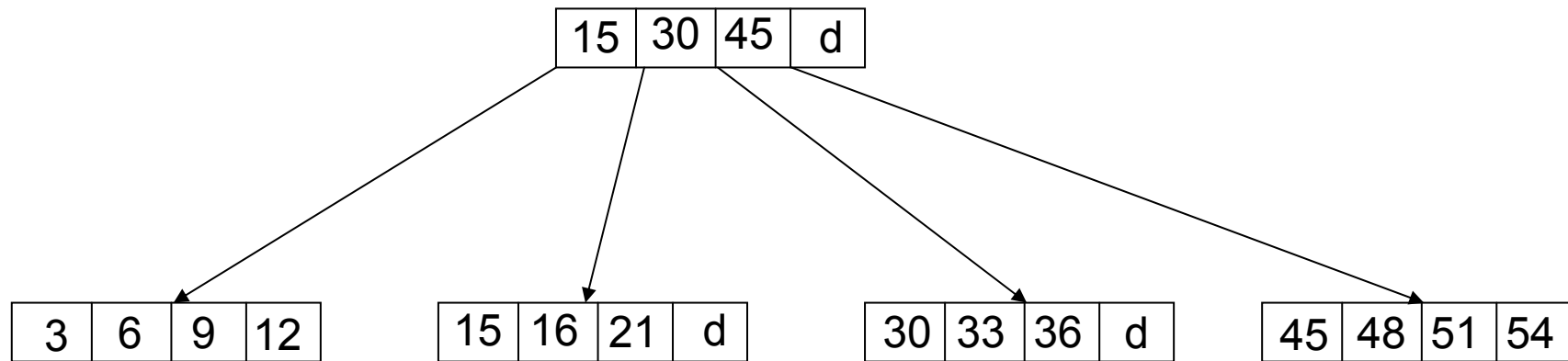


A B+ tree inverted index with branching factor $t = 5$

S_i = Satellite information/inverted index

d = Dummy.

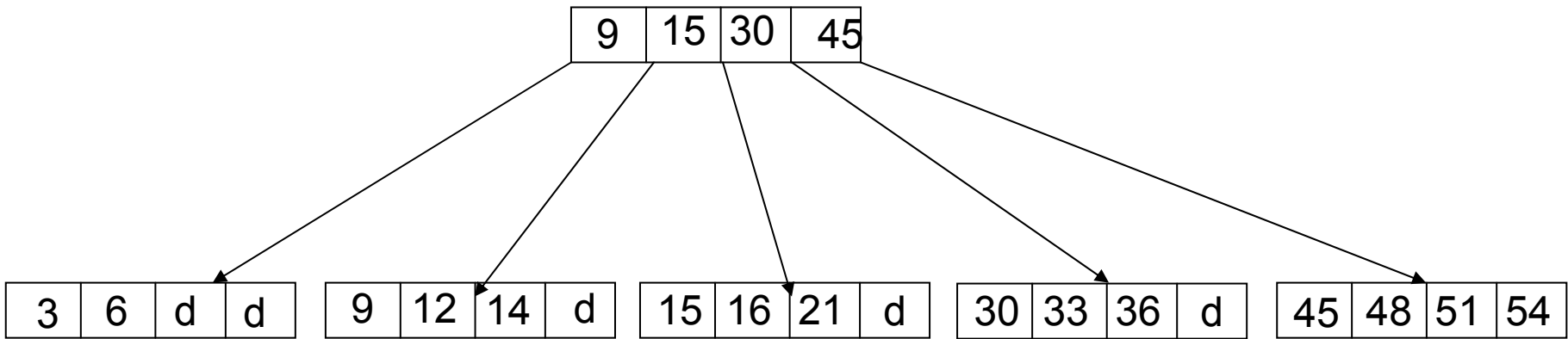
The index can be numbered using integers or words so long as the lexicographical ordering is maintained.



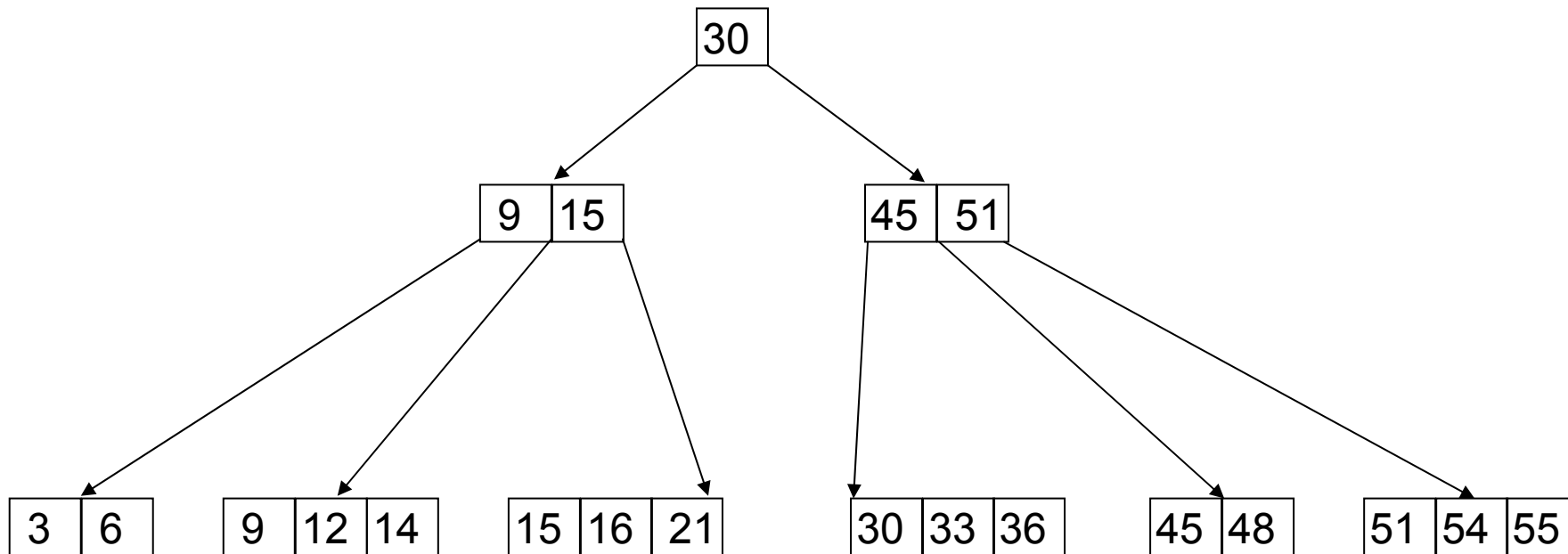
Insert 16

The Satellite nodes are omitted for clarity here after.

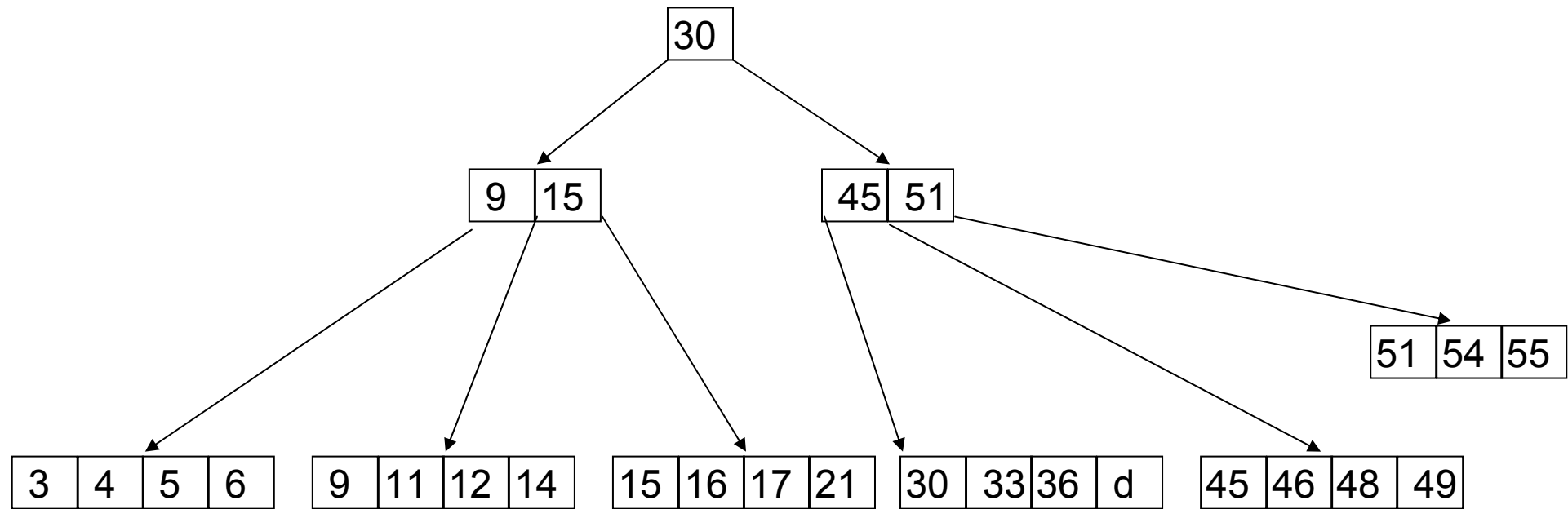
The worst case time complexity for insertion in a B⁺ tree with m elements is $O(\log_t m)$



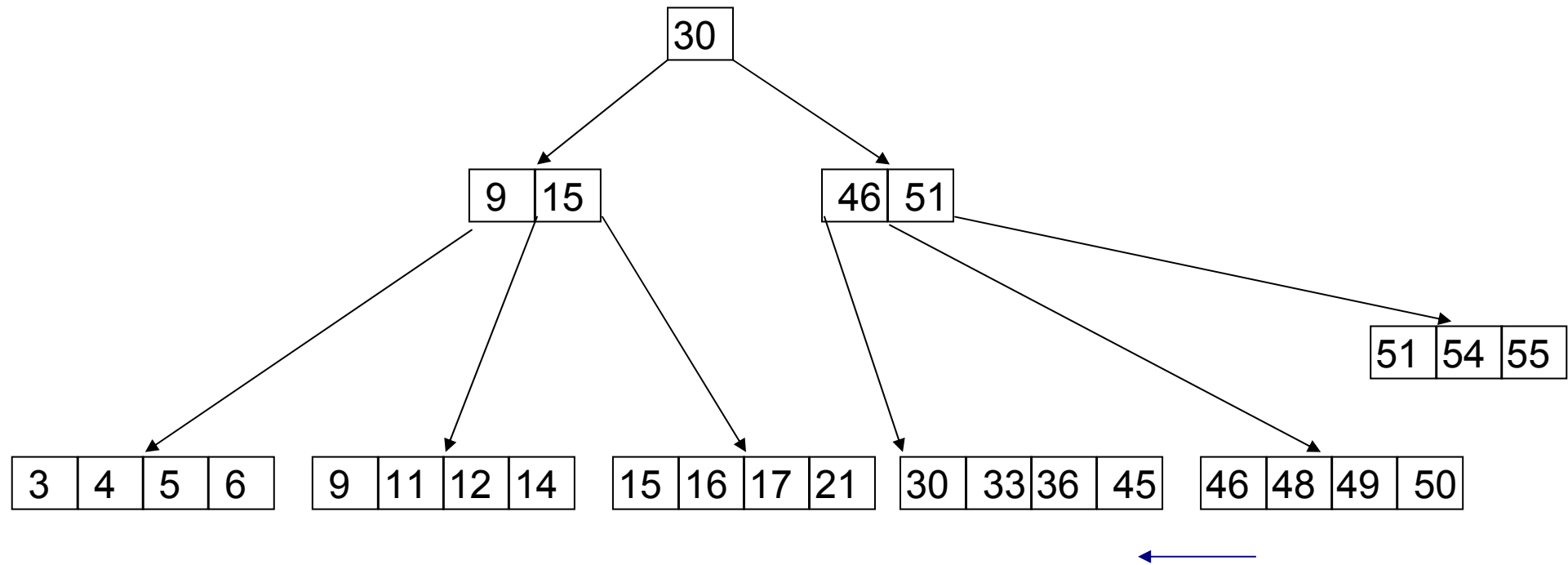
Insert 14



Insert 55

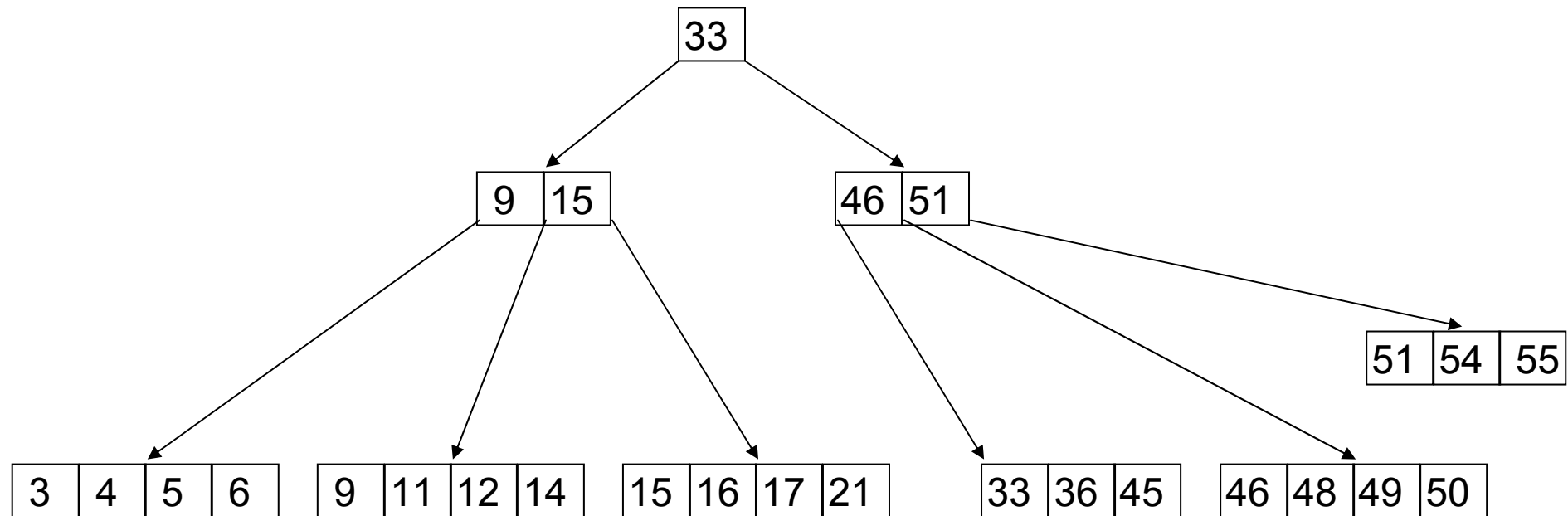


Insert 4, 5, 11, 17, 46, 49



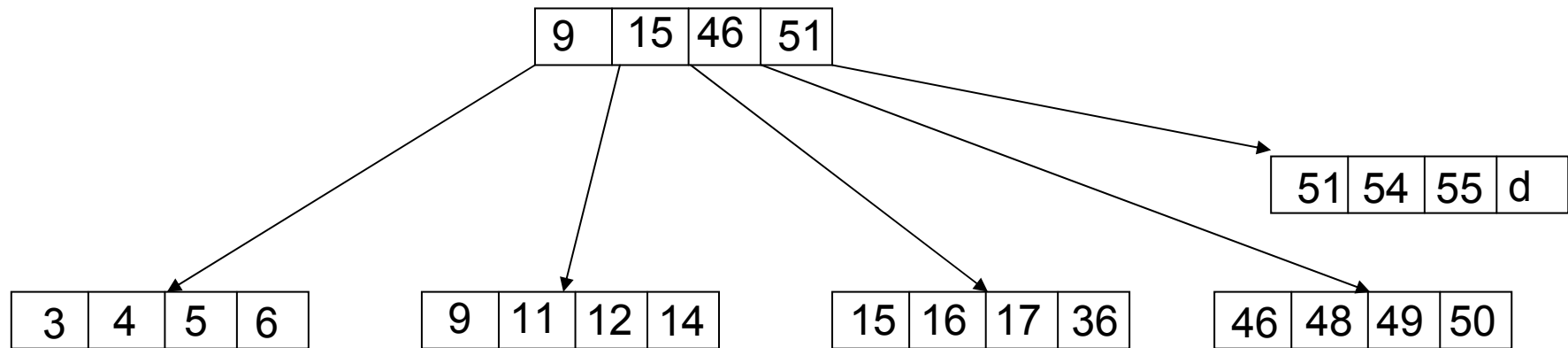
Insert 50 results in a rotation

Rotations minimize page splits by moving the records at the leaf and adjusting the indices. The movement of the record is usually from right to left whenever an empty slot is encountered.



Remove 30

The worst case time complexity for removal in a B+ tree with m elements is $O(\log_t m)$



Remove 21, 45 and 33


Speed Optimization

- Cache the root of the tree in main memory
- If possible, cache the depth=1 nodes of the B⁺ tree. Caching the root and the first level saves two disk operations for any manipulation on the B⁺ tree.
- Inserting r indices on a tree cached to height c and total height of the tree h , takes utmost $r * (h - c)$ disk accesses
- Instead of caching the depth=1 nodes, this buffer can be used to hold the r newly inserted indices. The r indices are sorted in main memory(sorting time is negligible since there is no disk access). Associate w instances with the r indices ($w < r$). I.e. each $w(1 \leq i \leq w)$ will have one or more lexicographically ordered index associated with it.

- If the w instances are uniformly distributed over the r indices, then one requires utmost $w * (h - c)$ disk accesses assuming that all the indices associated with each $w(1 \leq i \leq w)$ fits in one page/node.
- In practice the gains are observed not to be high and keeping $10 \leq w \leq 20$ seems to be a reasonable choice[3]. Certain practical timing observations are given in [10].

Space Optimization

- If the indices appearing in lexicographical order are words, then the prefix of the word is separated out and only the respective suffix is stored. It is found to give a 50% saving in space.
- Delta encoding is a compression technique which works by storing the difference between the present entry and the next entry.

- 
- E.g. 5, 6, 12, 14, 20 is delta encoded as 5,1,6, 2, 6. The disadvantage is that the chain has to be unrolled to decompress the data.



Conclusion

- B⁺ trees are efficient ways to store the inverted indices for text search
- A number of optimizations on space and time can be achieved by the methods discussed

References

1. Doug Cutting and Jan Pedersen, *Optimizations for Dynamic Inverted Index maintenance*.
2. Rivest, Cormen, Leiserson and Stein, *Introduction to Algorithms*, Prentice Hall of India.
3. <http://blog.scalingweb.com/2007/11/03/full-text-search-for-database-using-lucene-search-engine/>
4. Study notes for B+ tree-insertion and deletion.
URL: <http://www-users.itlabs.umn.edu/classes/Spring-2006/csci4707/B+Trees.pdf>
5. Structure of B-tree Index pages. URL:
<http://publib.boulder.ibm.com/infocenter/idshelp/v10/index.jsp?topic=/com.ibm.adref.doc/adref235.htm>
6. B-tree, Wikipedia Entry
7. B⁺ tree, Wikipedia Entry
8. Inverted Indexing, Wikipedia Entry
9. Delta encoding, Wikipedia Entry
10. <http://www.tfd.co.uk/blogs/sakaiblog/2008/01/15/luciene-index-merge-and-optimisation/>



Thank you!

avsarad@gmail.com