

# Multiway Trees

Josh Wilcox (jw14g24@soton.ac.uk)

May 6, 2025

## Contents

<b>1</b>	<b>Why care about disk</b>	2
1.1	Drawbacks of Big O . . . . .	2
1.2	Accessing Data from Disk . . . . .	2
<b>2</b>	<b>Multiway-Trees</b>	2
<b>3</b>	<b>B-Trees</b>	2
3.1	Definition and Use . . . . .	2
<b>4</b>	<b>B+ Trees</b>	2
4.1	Definition . . . . .	2
4.2	Implementation . . . . .	2
4.3	Adding values to B+ Trees . . . . .	3
<b>5</b>	<b>Trie</b>	3

## 1 Why care about disk

### 1.1 Drawbacks of Big O

- An assumption of Big-O is that all elementary operations take roughly the same amount of time
- **This is not true**
  - Typical time it takes to locate a record on a disk is around 10ms
  - $10^7$  times slower than an elementary operation

### 1.2 Accessing Data from Disk

- Minimising the number of **disk accesses** is critical for good performance
- This means, in database applications, we should store data as large sets
  - Storing data in binary trees is terrible as we would need around  $\log_2(n)$  disk accesses before data is located

## 2 Multiway-Trees

- An  $M$ -way tree has  $M$  children for each non-leaf node.
- This means the access time is:

$$\log_M(n) = \frac{\log_2(n)}{\log_2(M)}$$

## 3 B-Trees

### 3.1 Definition and Use

- B-Trees are **Balanced multiway trees** used for:
  - Fast Search
  - Finding successors and predecessors
  - Inserting, deletion, max, min, etc
- Not to be confused with *binary trees*
- Designed to keep **related data close** to each other in **disk** memory to *minimise retrieval time*
- Important for working with large amounts of data stored on **secondary storage**
- Used extensively in databases

## 4 B+ Trees

### 4.1 Definition

- Only leaf nodes contain data pointers. Internal nodes contain keys only
- Sequential access is possible like a linked list - leaf nodes are linked together
- Searching, Insertion, and Deletion is faster than a B-Tree

### 4.2 Implementation

1. Data items are stored at leaves
2. Non-leaf nodes store up to  $M - 1$  keys to guide the search
  - Key  $i$  represents the smallest key in subtree  $i + 1$

3. The root is either a leaf or has between 2 and  $M$  children
4. All non-leaf nodes except the root have between  $\lceil M/2 \rceil$  and  $M$  children
  - $M$  is the **maximum number of children** each non-leaf node can have
  - This means that for order  $M = k$ , each parent node can have  $k - 1$  keys!!
5. All leaves except the root are at the same depth and have between  $\lceil L/2 \rceil$  and  $L$  data entries:
  - $L$  is the number of data entries available in each leaf

### 4.3 Adding values to B+ Trees

- To add the value  $x$
- Find the two values in the root that  $x$  lies between and go to the node it points to
- Then find the values in this node that  $x$  lies in between and continue
- Repeat this until finding a leaf,
- If the leaf is not full:
  - Add  $x$  to the leaf
- If the leaf is full:
  - Split the leaf in two, adding the start of the second chunk as a new value of the parent node

## 5 Trie

- A trie (pronounced 'try') is a multiway tree used for storing large sets of **words**
- They are trees with a possible branch for every letter of **an** alphabet
- All words end with a special character \$
- If you go down the \$ branch, the word has ended