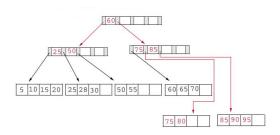
# Lesson 9: Sometimes It Pays Not to Be Binary



B-Trees, Tries, Suffix Trees

AICE1005

### **B-Trees**

- B-trees are balanced trees for fast search, finding successors and predecessors, insert, delete, maximum, minimum, etc.
- Not to be confused with binary trees
- They are designed to keep related data close to each other in (disk) memory to minimise retrieval time
- Important when working with large amount of data that is stored on secondary storage (e.g. disks)
- Used extensively in databases

AICE1005

### Accessing Data from Disk

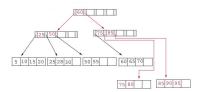
- When accessing data from disk minimising the number of disk accesses is critical for good performance
- In database applications we want to store data as large sets
- Storing data in binary trees is disastrous as we typically need around  $\log_2(n)$  disk accesses before we locate our data
- ullet It is not unusual in databases for  $n=10\,000\,000$  so that  $\log_2(n) \approx 24$
- Using binary trees it would often take several seconds to find a record

AICE1005

### B<sup>+</sup> Tree

- A pretty basic implementation would obey the following rules
  - 1. The data items are stored at leaves
- 2. The non-leaf nodes store up to M-1 keys to guide the search:  $\hbox{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
- 5. All leaves are at the same depth and have between  $\lceil L/2 \rceil$  and L data entries

- 1. B-Trees
- 2. Tries
- 3. Suffix Tree



AICE1005

## When Big-O Doesn't Work

- An underlying assumption of Big-O is that all elementary operations take roughly the same amount of time
- This just isn't true of disk look-up
- The typical time of an elementary operation on a modern processor is  $10^{-9}$  seconds
- But a typical hard disk might do 7 200 revolutions per minute or 120 revolutions per second
- ullet The typical time it takes to locate a record is around 10ms or  $10^7$ times slower than an elementary operation

AICE1005

Algorithms and Analysis

### Multiway-Trees

• To remedy this we can use M-way trees so that the access time is

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

- In practice we might use  $M \approx 200 \approx 2^8$  so we can reduce the depth of the tree by around a factor of 81
- The basic data structures for doing this is the B-tree!
- There are many variants of B-tree, all trying to squeeze a bit more performances from the basic structure

AICE1005

Algorithms and Analysis

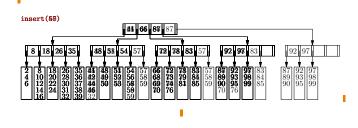
## Choosing M and L

- ullet The choice of M and L depends on the block size (the information read in one go from disk)
- It also depends on the type of data that is being stored (integer, reals, strings, etc.)
- ullet M and L might be in the hundreds or thousands
- $\bullet$  In the examples below we consider tiny M=L=5 which is unrealistic, but drawable

# **B-Tree Example**

# Other Changes





AICE1005 Algorithms and Analysis

## **B-Tree Summary**

- B-trees are an important data structure for databases where reducing the number of disk searches is vital.
- They tend to be much more complex than the other data structures we have seen!
- The problem of disk access can be improved by replacing disk memory with solid-state drives (still slow compared to memory)
- For massive databases new data structures have been developed to allow faster (although less flexible) information access (e.g. NOSQL, MongoDB, Neo4j)

AICE1005 Algorithms and Analysis 1

### Tries

- A **Trie** (pron. 'try') or **digital tree** is a multiway tree often used for storing large sets of words
- They are trees with a possible branch for every letter of an alphabet
- ullet Their names comes from the word retrieval
- Tries usually compactify the edges in the treel
- All words end with a special letter "\$"

AICE1005 Algorithms and Analysis

### **Uses of Tries**

- Tries are yet another way of implementing sets
- $\bullet$  They provide quick insertion, deletion and find  $\hspace{-0.1cm}\blacksquare$
- Typically considerably quicker than binary trees and hash tables
- They are particularly good for spell checkers, completion algorithms, longest-prefix matching, hyphenation
- Each search finds the longest match between the words in the set and the query

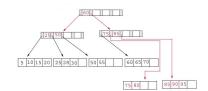
• If the root is full then it can be split into two and a new root

- B-trees also have to allow the removal of records without losing its structure!
- There are a number of variant strategies (e.g. neighbouring nodes can adopt a child if the current node cannot expand any more)
- The actual implementation of B-trees is tricky because there are many special cases

AICE1005 Algorithms and Analysis 10

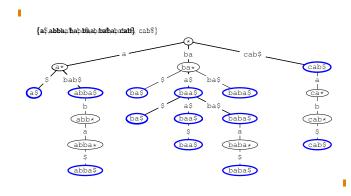
# Outline

- 1. B-Trees
- 2. Tries
- 3. Suffix Tree



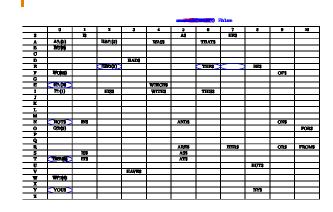
AICE1005 Algorithms and Analysis 12

# Trie



AICE1005 Algorithms and Analysis 14

## Trie for 31 Most Common English Words



CE1005 Algorithms and Analysis 15 AICE1005 Algorithms and Analysis

# Disadvantage of Tries

- Table-based tries typically waste large amounts of memory
- Often table-based tries are used for the first few layers, while lower levels use a less memory intensive data structure!
- These days memory is less of a problem so table-based tries are acceptable for some applications
- There are many implementations of tries each suited to a particular task

AICE1005 Algorithms and Analysis 1

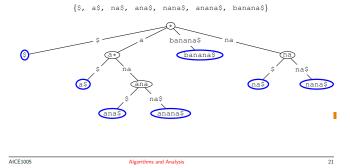
# Why Tries?

- Tries are a classic example of a trade-off between memory and computational complexity
- Tries are slightly specialist and tend to get used in very particular applications
  - ★ Finding longest matches
  - ⋆ Completion, spell checking, etc.
- A basic trie is not too complicated, however, . . . I
- There are many implementation which try to overcome the difficulty of wasting too much memory!

AICE1005 Algorithms and Analysis 19

# **Suffix Tree**

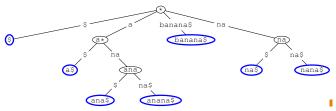
- Suffix tree is a trie of all suffixes of a string
- E.g. banana



## **String Matching**

 To find a match of a query string, Q, in a text, T, we can first construct the suffix tree of the string T we then simple look up the query, Q, using the trie

 $\{\$$ , a\$, na\$, ana\$, nana\$, anana\$, banana\$



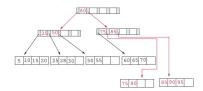
# **Binary Tries**

- One extreme (though not uncommon) solution to address memory issues is to build a bit-level trie so the data-structure is a binary tree!
- It differs from a binary tree in that the decisions to go left or right depends on the current bit
- Although you lose the advantage of a multiway tree (of reducing the depth) it does find the longest match and it speeds up finds which fail

AICE1005 Algorithms and Analysis 18

## **Outline**

- 1. B-Trees
- 2. Tries
- 3. Suffix Tree



AICE1005 Algorithms and Analysis 20

## Importance of Suffix Tree

- The first linear-time algorithm for computing suffix trees was proposed by Peter Weiner in 1973, a more space efficient algorithm was proposed by Edward M. McCreight in 1976
- Esko Ukkonen in 1995 proposed a variant of McCreight's algorithm, but in a way that was much easier to understand
- It really only got implemented after this
- They are very important for string-based algorithms
- $\bullet$  The classic application is in finding a match for a query string, Q, in a text, T

AICE1005 Algorithms and Analysis 22

## Complexity of Suffix Trees

- Using a regular trie for a suffix tree would typically use far too much memory to be useful
- $\bullet$  However, by using pointers to the original text it is possible to build a suffix tree using O(n) memory where n is the length of the text
- Furthermore (and rather incredibly) there is a linear time  $\big(O(n)\big)$  algorithm to construct the triel
- The algorithm is not however trivial to understand

ICE1005 Algorithms and Analysis

### Lessons

- Suffix trees are efficient whenever it is likely that you will do multiple searches!
- Exact word matching is in itself a very important application
- Suffix trees in combination with dynamic programming (which we will eventually get to) can be used to do inexact matching (finding the match with the smallest edit distance)
- Suffix trees get used in bioinformatics, advanced machine learning algorithms, . . . ■

- Multiway trees can considerable speed up search over binary trees
- They are very important in some specialised applications (e.g. databases, spell-checking, completion, suffix trees)
- They are not as general purpose as binary search trees and are more complicated to implement
- But they can give the best performance—sometimes performance matters enough to make it worthwhile implementing multiway trees

AICE1005 AIgorithms and Analysis 25 AICE1005 AIgorithms and Analysis :