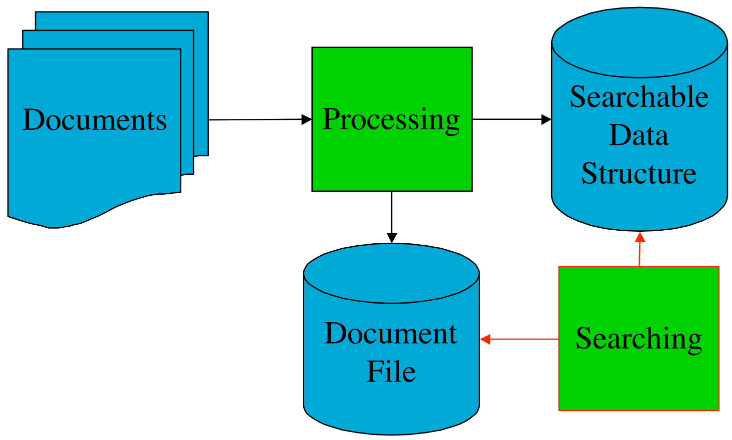
INFSCI 2140
Information Storage and Retrieval
Lecture 7: Data Structures and Algorithms

Peter Brusilovsky
http://www2.sis.pitt.edu/~peterb/2140-051/

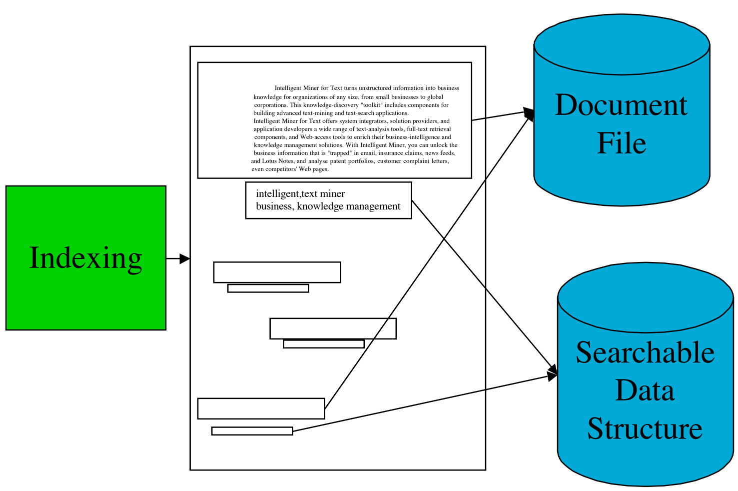
Overview

- Document processing, storage, search
- Document files
  - – the issue of record length
- Search problem
- Simple search solutions
- Algorithms and complexity
- Advanced searchable data structures

Document processing and search



Document processing and search



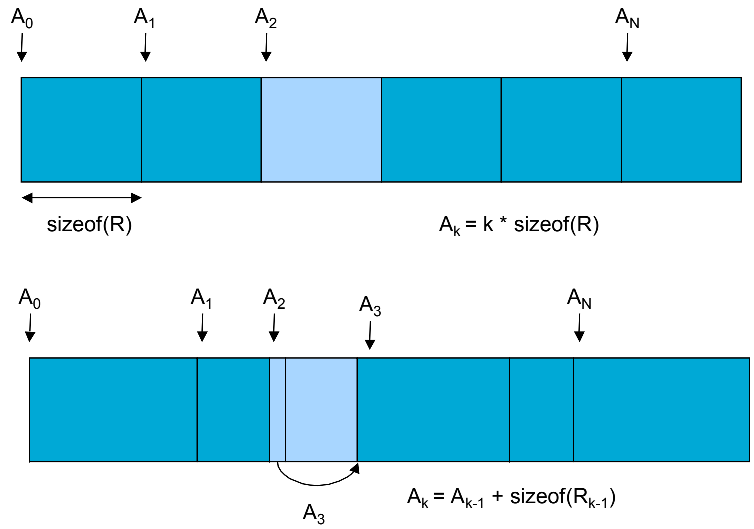
Document File

- A collection of documents is stored as a *document file*
- A representation of a document in a document file is called a *record*
- Type of document files
  - Sequential
  - Hashed
  - Hierarchical and netted

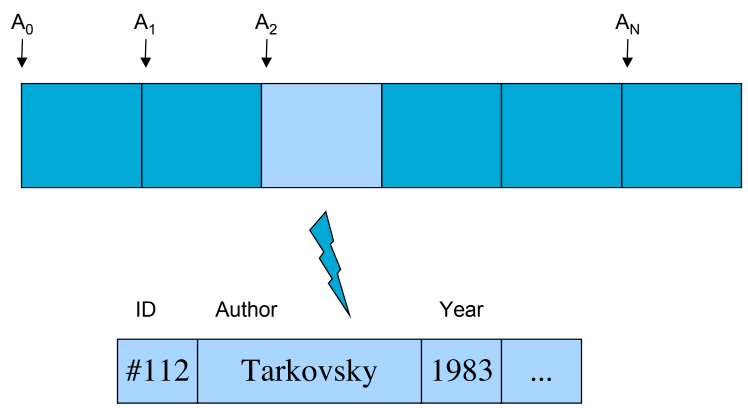
Sequential files

- Documents are arranged in a sequence
  - Usually sorted by some criteria so similar documents are close to each other
- Records in a document file can have fixed of variable length
- Each record in a file has an address and can be retrieved given this address
  - Need random access device for efficiency

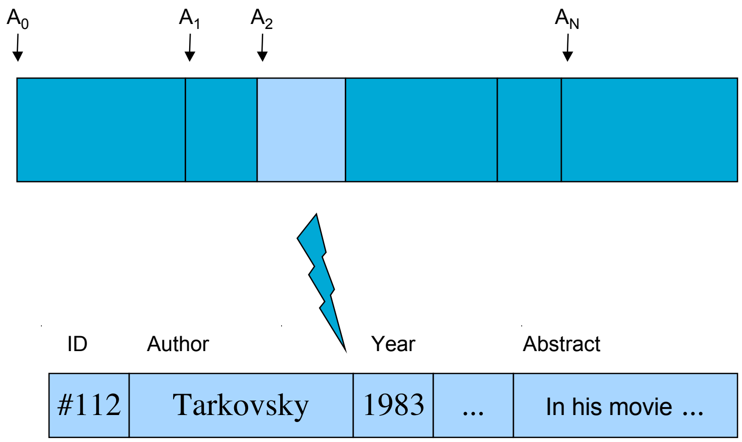
Fixed or variable record length



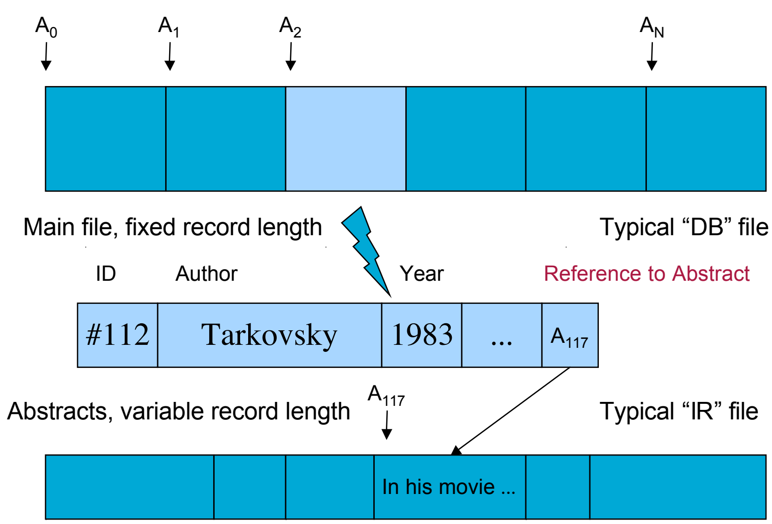
Fixed length example



Variable length example



From variable to fixed length



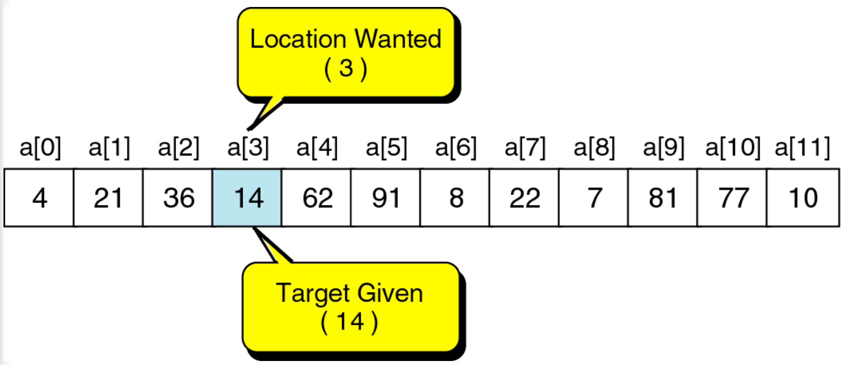
Main problem of search

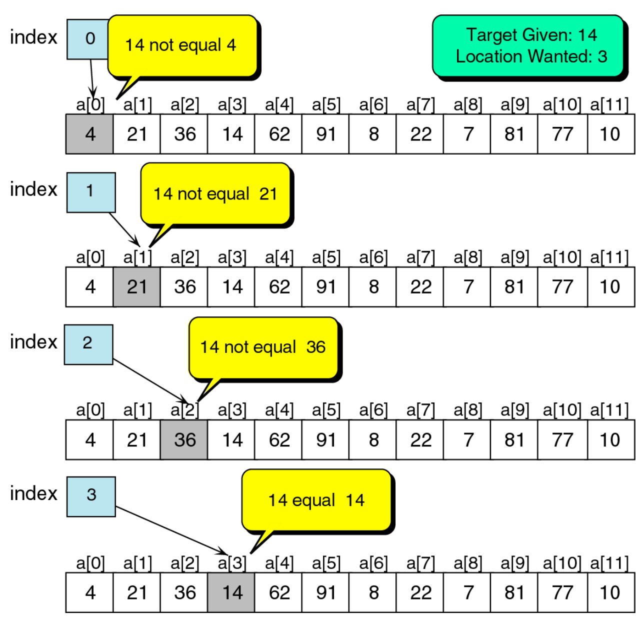
- Find all records (documents) with the given value of the key
  - *Year* = 1998
  - Fellini in *Director*
  - Brilliant in *Abstract*
- The search techniques for fixed and variable length record files are technically different but conceptually similar

Search Techniques

- Sequential search, unordered records
  - Linked lists for variable size records
- Sequential search, ordered records
  - Linked lists for variable size records
- Binary search
  - Binary trees for variable size records
- Direct search (hashing)

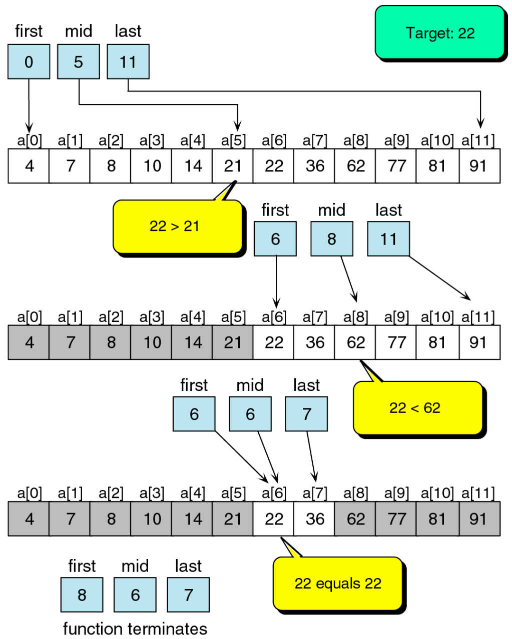
Sequential search





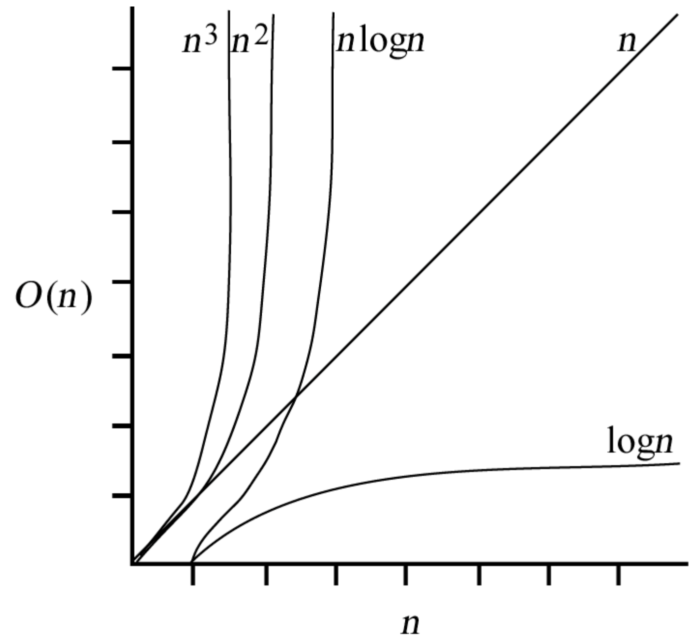
Sequential search

Binary search

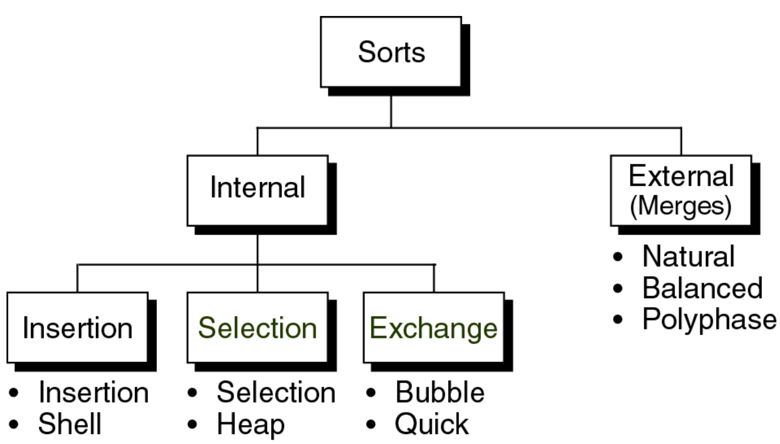


Complexity

- Big-O notation
- Sequential search: n/2 (n if not found)
- Sequential ordered search: n/2 ➔ O(n)
- Binary search ➔ $O(\log_2 n)$



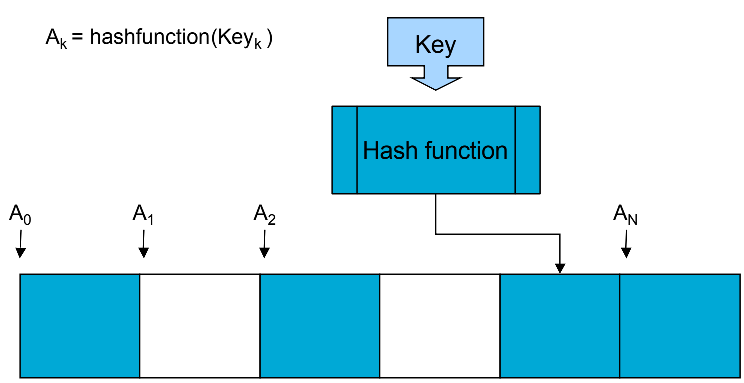
Sort Classification



Complexity for various sorts

- Complexity for insertion sorts:
  - straight insertion: $O(n^2)$
  - Shell sort $O(n^{1.25})$
- Complexity for selection sorts
  - straight selection: $O(n^2)$
  - heap sort: $O(n\log_2 n)$
- Complexity for exchange sorts
  - bubble sort: $O(n^2)$
  - quick sort: $O(n\log_2 n)$

Hashing and hashed files



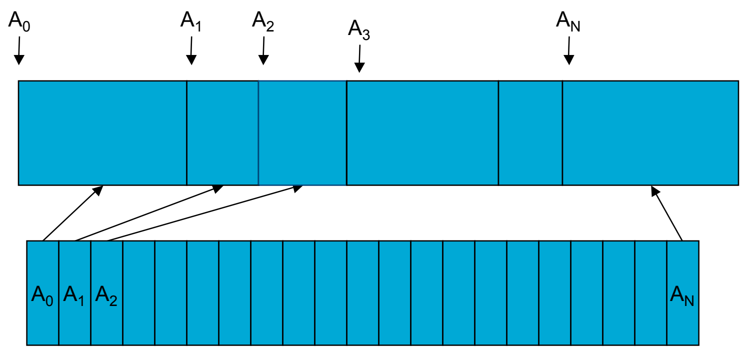
Problems of hashing

- Hard to define good hash function
  - Wasted space vs. *collisions*
- No sequential processing
- Similar documents are scattered all over the file
- As a result, hashing has relatively little use in IR

Searching in IR document files

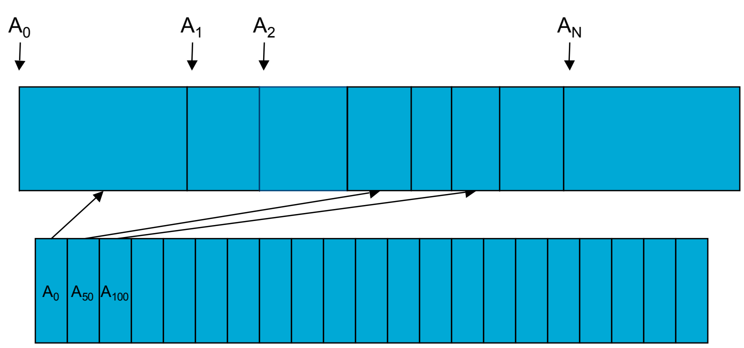
- Need additional data structures for fast search in large IR document files
  - Binary tree over sequential file
  - Indexed files
  - Inverted files
  - B-trees
  - Suffix trees
  - Signature files
  - Tries

Simple “full” index file



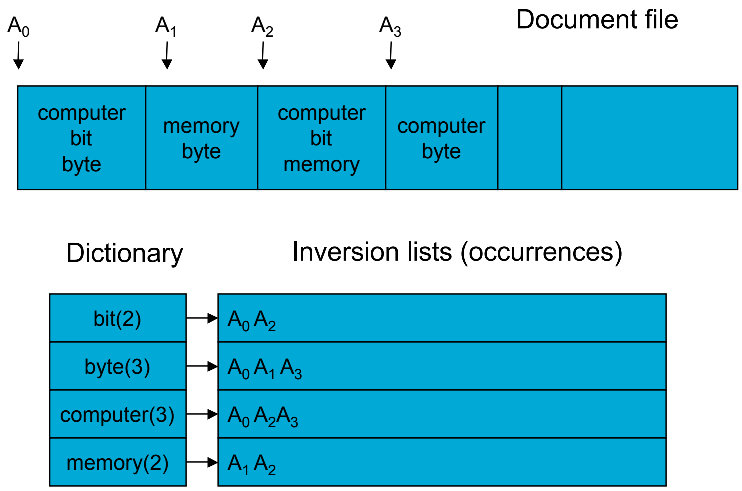
- Each document is indexed in the index file
- Can be used for binary search in an “IR” file
- Waste of space for large files

Typical index file



- File is split into sections. Each section is indexed in the index file
- Use combination of binary and sequential search
- Large files need hierarchical indexing (B-trees)

Inverted files



What is in inversion lists?

- Document reference
  - Address, record number
- Location inside the document
  - Where is the word: address, number, block
- Parameters
  - Weight of this term for the document
- B-tree is better than dictionary to point to inversion lists

Google

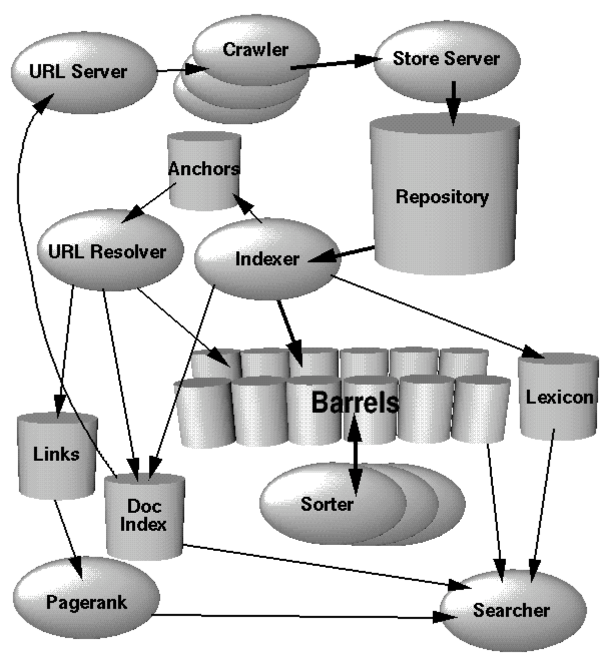
Not really searching the WWW, but querying an respresentation of a “pre-searched” WWW.

Google’s indices map keys (search vocabulary elements) to web pages.

- Require ~ 50GB
- Proposal suggests that its Document Index is btree based (~10 GB)

The following image is from Brin and Page doc:

http://www7.scu.edu.au/programme/fullpapers/1921/com1921.htm



Google, cont.

- The proposal of Sergey Brin and Lawrence Page estimated originally that they would need about 100 Million pages.
- Now over 1 Billion pages – off by an order of magnitude.
- How big is a billion pages? 4 terabytes!

Btree - requirements

- Invented by R. Bayer in 1970
- A Btree is a generalization of a Multiway tree which in turn is a generalization of a binary tree.
- Requirements:
  - Maintain balance
  - Minimize Disk I/O - why?

Btree – requirements, cont.

- Disk access speed
  - – between 3ms and 10ms
- Compare this to CPU speeds
- So, although btrees are old technology, they remain useful!
- Common to trees:

  RANDOM access - not direct access

Btree - definition

- A multiway tree in which
  - All leaf nodes are on the same level
  - Every non-leaf node, except the root, has between **M/2** and **M** descendents (leaf nodes have zero descendents)
  - The root can have **0** - **M** descendents

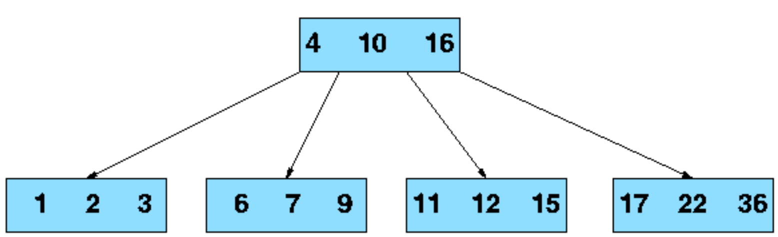
  (All descendents are non-empty)
- **M** is the ***order*** of the btree
- What determines **M**?

Btrees

- The order of a binary tree is trivially 2.
- The *order* (**M**) of a btree is set at creation. A function of
  - – Size of node - **How is this determined?**
  - – Size of keys (or partial keys)
- What does a node look like?

Btree

- A simple example of Btree
- What is its order?



Btree - height

- The maximum height of a btree index determines the path length or max number of accesses for a search. ***Remember, each node represents a potential disk access.***
- Assume that each internal node has the minimum number of descendents (M/2); this results in maximum depth of tree.
- For **N** elements,

  max. height **$< \log_{m/2}((N+1)/2)$**

  So search is $O(\log_{m/2} N)$

Some max. height examples

- For M = 200

  $\log_{M/2}(1M) \quad <= 3$

  $\log_{M/2}(1G) \quad <= 5$

  $\log_{M/2}(1T) \quad <= 7$

  $\log_{M/2}(1P) \quad <= 8$

  $\log_{M/2}(1E) \quad <= 9$

Btree Improvements

- Most Btrees today are really B+trees
  - Records (vs keys) are stored in leaf nodes
  - Leaf nodes are links to provide ***sequential*** as well as random access
- Can relax the constraint on number of elements for leaf nodes without affecting algorithms.
- Variable-length keys – can relax bounds (m/2, m) for number of descendents
- High Concurrency – multi-granular locking

Btree Access Methods

- Create a btree
- Destroy a btree
- Search for a specific record (query)
- Insert a record
- Delete a record
- Read a record
- Iterator operations
- Demo: http://sky.fit.qut.edu.au/~maire/baobab/baobab.html
- Tutorial: http://www.bluerwhite.org/btree/