Last time

- What info we store:
  - inverted index, meta data
- Query processing based on merge-like operations on postings lists
- Use of classic linear-time list merge algorithm:
  - postings lists sorted by a doc (static) value

Today

- Accessing entries of inverted index
  - disk access costs
- Constructing index

Data structure for inverted index?

How access individual terms and each associated postings list?

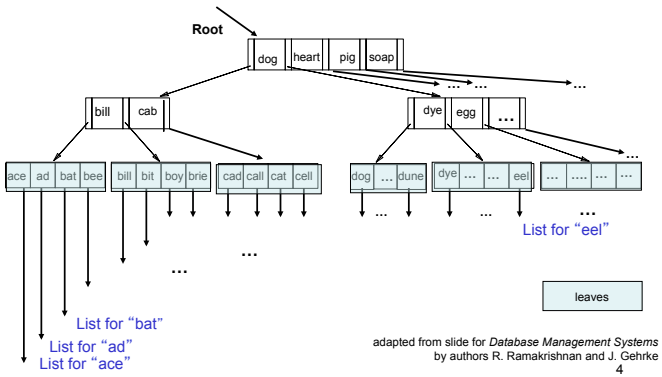
Assume an entry for each term points to its posting list

Data structure for inverted index?

- Sorted array:
  - binary search IF can keep in memory
  - High overhead for additions
- Hashing
  - Fast look-up
  - Collisions
- Search trees: B+-trees
  - Maintain balance - always log look-up time
  - Can insert and delete

Example B+ Tree

order = 2: 2 to 4 search keys per interior node



adapted from slide for *Database Management Systems* by authors R. Ramakrishnan and J. Gehrke

B+- trees

- All index entries are at leaves
- Order *m* B+ tree has *m+1* to *2m+1* children for each interior node
  - **except root** can have as few as 2 children
- Look up: follow root to leaf by keys in interior nodes
- Insert:
  - find leaf in which belongs
  - If leaf full, split
  - Split can propagate up tree
- Delete:
  - Merge or redistribute from too-empty leaf
  - Merge can propagate up tree

5

Disk-based B+ trees for large data sets

- Each leaf is file page (block) on disk
- Each interior node is file page on disk
- Keep top of tree in buffer (RAM)
- Typical sizes:
  - m ~ 200;
  - average fanout ~ 267
    - Height 4 gives ~ 5 billion entries

6

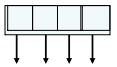
prefix key B+ trees

- Save space
- Each interior node key is shortest prefix of word needed to distinguish which child pointer to follow
  - Allows more keys per interior node
  - higher fanout
    - fanout determined by what can fit
    - keep at least 1/2 full

7

Revisit hashing - on disk

- hash of term gives address of bucket on disk
- bucket contains pairs
  (term, address of first page of postings list)
- bucket occupies one file page



8

Now

- How construct inverted index from "raw" document collection?
  - Don't worry about getting into final index data structure

9

Preliminary decisions

- Define "document": level of granularity?
  - Book versus Chapter of book
  - Individual html files versus combined files that composed one Web page
- Define "term"
  - Include phrases?
    - How determine which adjacent words -- or all?
  - Stop words?

10

Pre-processing text documents

- Give each document a unique ID: docID
- Tokenize text
  - Distinguish terms from punctuation, etc.
- Normalize tokens
  - Stemming
    - Remove endings: plurals, possessives, "ing",
      - cats -> cat; accessible -> access
    - Porter's algorithm (1980)
  - Lemmatization
    - Use knowledge of language forms
      - am, are, is -> be
    - More sophisticated than stemming

(See *Intro IR* Chapter 2)

11

Construction of posting lists

- Overview
  - "document" now means preprocessed document
  - One pass through collection of documents
  - Gather postings for each document
  - Reorganize for final set of lists: one for each term
- Look at algorithms when can't fit everything in memory
  - Main cost file page reads and writes
    - "file page" minimum unit can read from drive
      - May be multiple of "sector" device constraint

12

Memory- disk management

- Have buffer in main memory (RAM)
  - Size = B file pages
  - Read from disk to buffer, page at a time
    - Disk cost = 1 per page
  - Write from buffer to disk, page at at time
    - Disk cost = 1 per page

13

Sorting List on Disk - External Sorting General techique

- Divide list into size-B blocks of contiguous entries
- Read each block into buffer, sort, write out to disk
- Now have ⌈L/B⌉ sorted sub-lists
  where L is size of list in file pages
- Merge sorted sub-lists into one list
  - How?

14

Merging Lists on Disk: General technique

- K sorted lists on disk to merge into one
- If K+1 <= B:
  - Dedicate one buffer page for output
  - Dedicate one buffer page for each list to merge input from different lists
  - Algorithm:
    Fill 1 buffer page from each list on disk
    Repeat until merge complete:
      Merge buffer input pages to output buffer pg
      When output buffer pg full, write to disk
      When input buffer pg empty, refill from its list

15

- If K+1 > B:
  - Dedicate one buffer page for output
  - B-1 buffer page for input from different lists
  - Define "level-0 lists": lists need to merge

16

If K+1 > B: Algorithm

```
j=0
Repeat until one level-j list:
   { Group level-j lists into groups of B-1 lists
              // ⌈K/(B-1)⌉ groups for j=0
      For each group, merge into one level-(j+1) list by:
       { Fill 1 buffer page from each level-j list in group
         Repeat until level-j merge complete:
            Merge buffer input pages to output buffer pg
            When output buffer pg full,
                 write to group's level-(j+1) list on disk
            When input buffer pg empty, refill from its list
       }
    j++
   }
```

17

Number of file page read/writes?

- Merge lists?
- External sort?

18

So far

- Preprocessing the collection
- Sorting a list on disk (external sorting)
  - Cost as disk I/O

Now look at actually building

19

Index building Algorithm: "Block Sort-based"

1. Repeat until entire collection read:
   - Read documents, building (term, <attributes>, doc) tuples until buffer full
     - one tuple for each occurrence of a term
   - Sort tuples in buffer by term value as primary, doc as secondary
     - Tuples for one doc already together
     - Use sort algorithm that keeps appearance order for = keys: stable sorting
   - Build posting lists for each unique term in buffer
     - Re-writing of sorted info
   - Write partial index to disk

20

continuing "Blocked Sort-based"

2. Merge partial indexes on disk into full index

- Partial index lists of (term:postings list) entries must be merged
- Partial postings lists for one term must be merged
  - Concatenate
    - Keep documents sorted within posting list
- If postings for one document broken across partial lists, must merge

21

Remarks: Index Building

- As build index:
  - Build dictionary
  - Aggregate Information on terms, e.g. document frequency
    - store w/ dictionary
  - What happens if dictionary not fit in main memory as build inverted index?
- May not actually keep every term occurrence, maybe just first k.
  - Early Google did this for k=4095. Why?

22

What about anchor text?

- Complication
- Build separate anchor text index
  - strong relevance indicator
  - keeps index building less complicated

23

Other separate indexes?

Examples

- Other strong relevance indicators
  - abstracts of documents
    - compare listing abstract positions 1st in main index
  - tiered indexes based on term weights
- types of documents
  - volatility
    - news articles
    - blogs
    - etc.

24