COS 226 Lecture 11: External sort/search

Huge file:

- 1967: 1 million bytes
- 1997: 1 trillion bytes

Use high-level abstraction

- local memory
  - random access
  - faster sequential access
- external memory
  - far slower than local memory
  - read/write in large blocks
  - faster sequential access
  - restrictions on access

Huge file:

- thousands of times the size of local memory

SORTING

- N: number of records
- M: size of memory
- 2P: number of external devices

SEARCHING

- N: number of records
- M: page size

Algorithms

SORTING

- balanced merge
- polyphase merge

SEARCHING

- indexed sequential
- B-trees
- skip lists
- extendible hashing

Many ancient algorithms still relevant

Balanced multiway merge

Make multiple passes over the file

* can MERGE huge sorted blocks in memory
* use half the devices for "output"

pass 0: sort the file into sorted blocks of size M
on devices 0, 1, 2, ... P−1

pass 1: make blocks of size M*P
P−way merge out to devices P, P+1, P+2, ... 2*P

pass 2: make blocks of size M*P^2
P−way merge out to devices 0, 1, 2, ... P−1

...

pass t: make blocks of size M*P^t

File is sorted when M*P^t > N
t = log_P (N/M) passes through the data

	PC sort	**workstation**
file	**1 billion**	**1 trillion**
memory	**1 million**	**1 billion**
devices	**2**	**10**
passes	**lg 1000 = 10**	**log_10 1000 = 3**

Costs

no. of passes (no. times each datum touched)
no. of devices

Bottom line:

can sort a file in 3−10 times the amount of time it takes to read or write it

Balanced merge example

input

T H E Q U I C K B R O W N F X J M P S V L A Z Y D G

sorted runs of size 3

E H T I Q U B C K O R W F N X J M P L S V A Y Z D G *

sorted runs of size 9

B C E H I K Q T U F J M N O P R W X A D G L S V Y Z *

output

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z *

sorted runs (lengths) on devices

0	1	2	3	4	5
3(3)	3(3)	3(3)	0	0	0
0	0	0	1(9)	1(9)	1(9) 27
1(27)	0	0	0	0	0 27

Polyphase Merge

Concept: cut the number of devices needed in half

MERGE-UNTIL-EMPTY

one device empty

different number of runs

on other devices

merge to empty device

until another device is empty

iterate

Strategically place runs on devices

to make it so that last merge

makes all devices empty at once

Work backwards to figure placement

Ex: 3 devices

.	**1**	**0**	**0**	**0**
.	**0**	**1**	**1**	**1**
.	**1**	**0**	**2**	**2**
.	**3**	**2**	**0**	**4**
.	**7**	**6**	**4**	**0**

distribution for 17 runs

General pattern is complicated!

Polyphase merge example

input

T H E Q U I C K B R O W N F X J M P S V L A Z Y D G

sorted runs of size 3

0 0 0 1 1 3 3 3 3

E H T I Q U B C K O R W F N X J M P L S V A Y Z D G *

result of phase 1

0 3 3 2 2

B C K A Y Z D G * E H J M O P R T W F I L N S Q U V X

result of phase 2

3 2 1

D G * F I L N S Q U V X A B C E H J K M O P R T W Y Z

output

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z *

sorted runs (lengths) on devices

.	0	1	2	3	
.	---	---	---	---	
.	3(3)	2(3)	0	4(3)	
.	1(3)	0	2(9)	2(3)	18
.	0	1(15)	1(9)	1(3)	15
.	1(27)	0	0	0	27

Polyphase merge analysis

Ex. 34 initial sorted runs, 3 devices

.	**0**	**21**	**13**
.	**13**	**8**	**0**
.	**5**	**0**	**8**
.	**0**	**5**	**3**
.	**3**	**2**	**0**
.	**1**	**0**	**2**
.	**1**	**0**	**1**
.	**1**	**0**	**0**

Fibonacci numbers

phi = 1.618... (golden ratio)

phi^N runs in phi phases

each touching 1/phi of the data

THM: Total number of passes is log_phi(N/M)/phi

.	**PC sort**	**workstation**
. **file**	**1 billion**	**1 trillion**
. **memory**	**1 million**	**1 billion**
. **devices**	**3**	**3**
. **passes**	**8**	**8**

Using half as many devices as balanced merge,

still completes sort in fewer passes

Precise analysis extremely complex

Virtual memory

Another idea: use quicksort

Paging system "automatically" takes care
of reading data in blocks

Requires
ability to address huge files
locality of reference in program

Partitioning on such a system:
read a block from the left
read a block from the right
write small elements on left
write small elements on right
Paging systems takes care of details

Estimate of costs
two devices working at a time
2 ln (N/M) passes

Can customize to partition-based sort
distribute files a la MSD sort
utilize multiple devices

Basic external searching

N records
stored in pages of size M

Goal:
- find item with given key
- read as few pages as possible

INDEXED SEQUENTIAL ACCESS

Store file sequentially on devices
- build index for each device
- build master index for devices

Like encyclopedia

Problem:
- index is hard to maintain
  - under insertion- deletion
- real goal: *dynamic* index

B-trees

Generalize 2-3-4 trees: up to M links per node
Split full nodes on the way down

Used for external search
- node size = page size
- typical: M=1000
  - search and insert 1 trillion records in four or five page acceses

Key point: flexibility to do fast insert

Red-black abstraction still works
- BUT

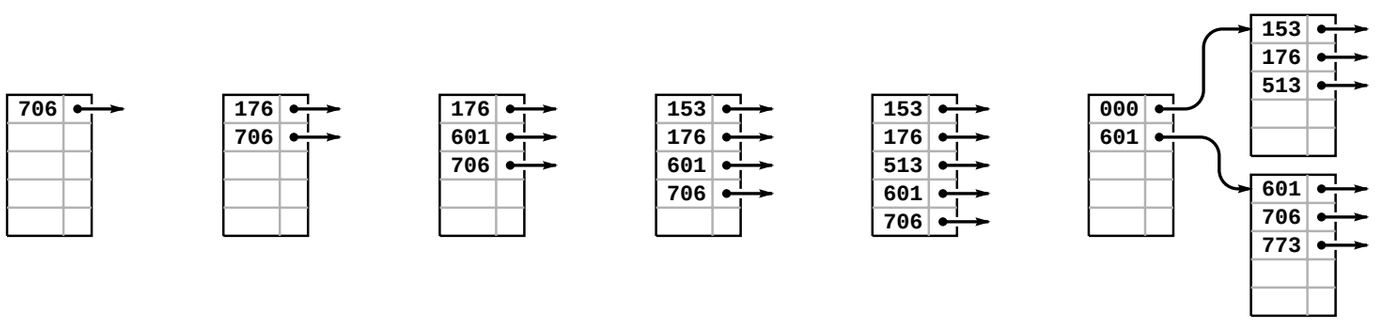
might not use internal links
(use binary search instead)

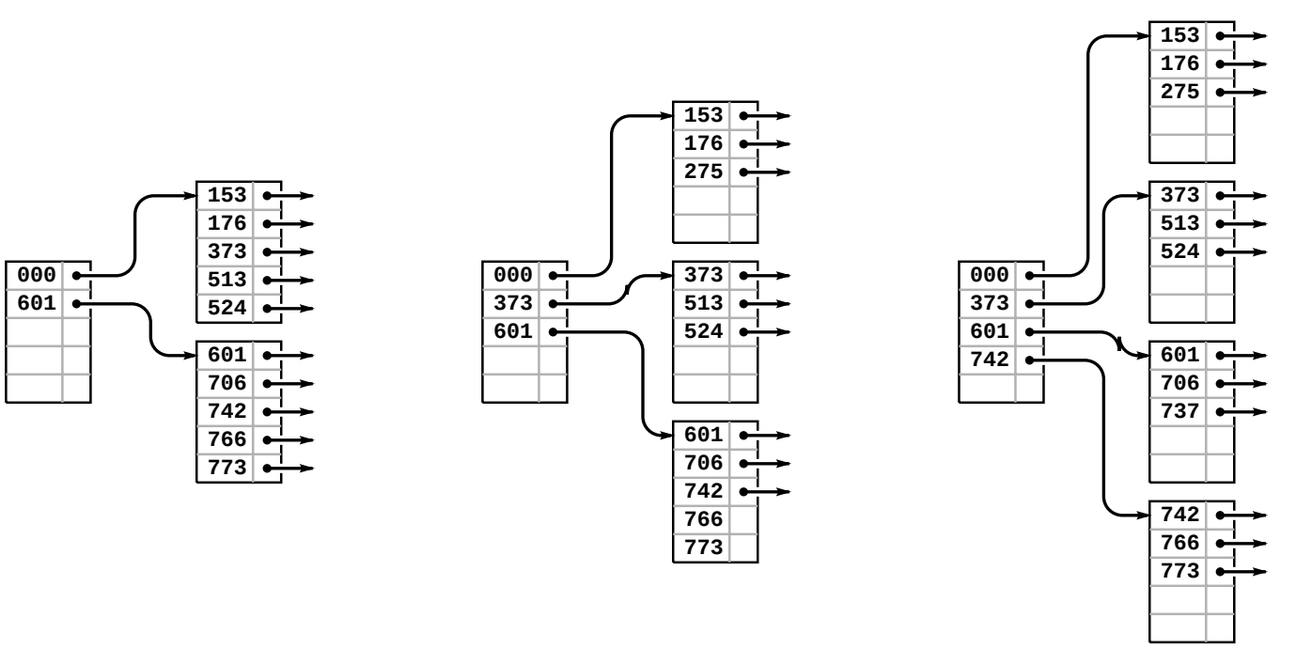
Space--time tradeoff
- M large: only a few levels in tree
- M small: less wasted space

Normally build as separate index
Main issue is to minimize wasted space

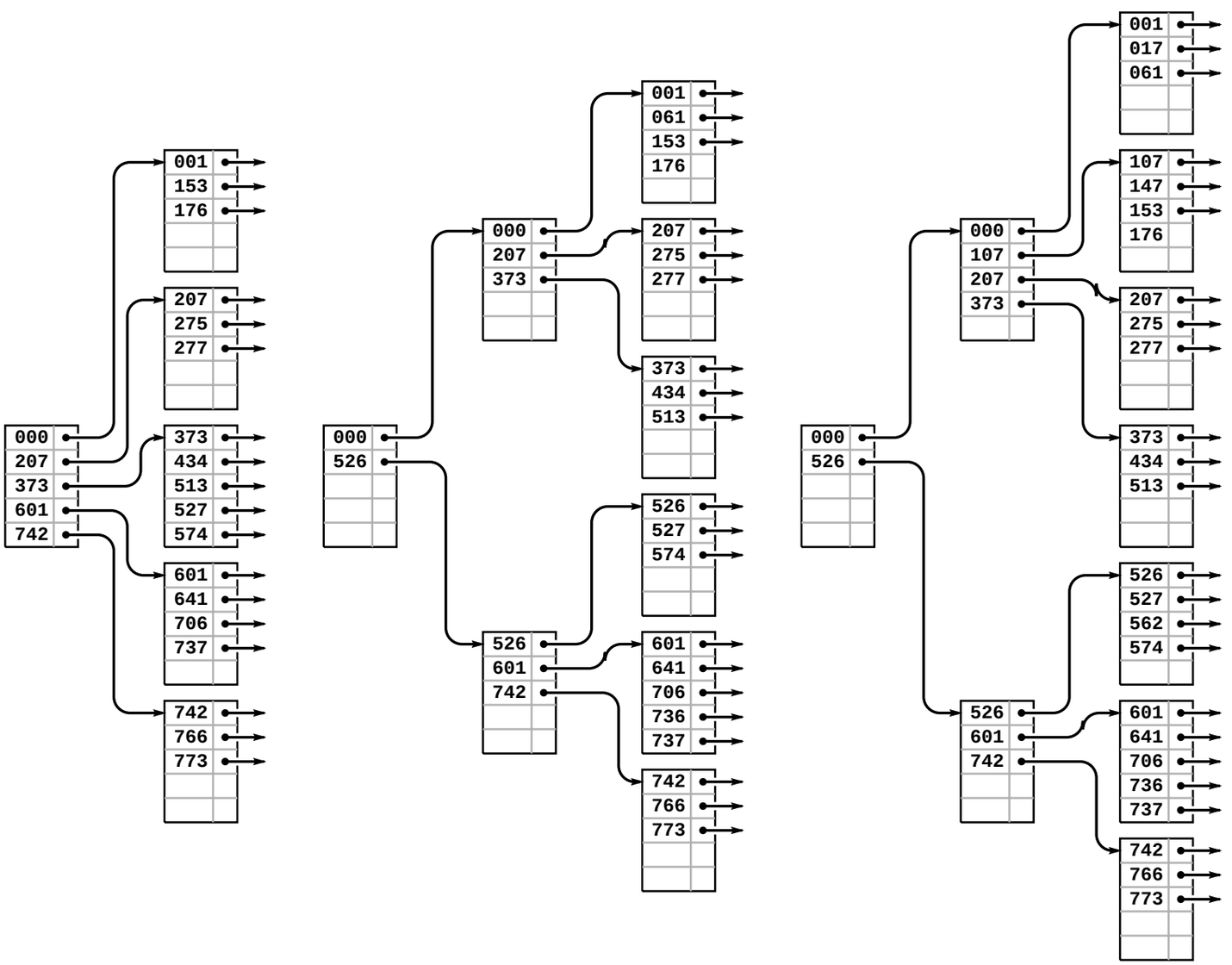
Bottom line
- roughly log_M N page accesses
- 3 or 4 in practice

B tree example

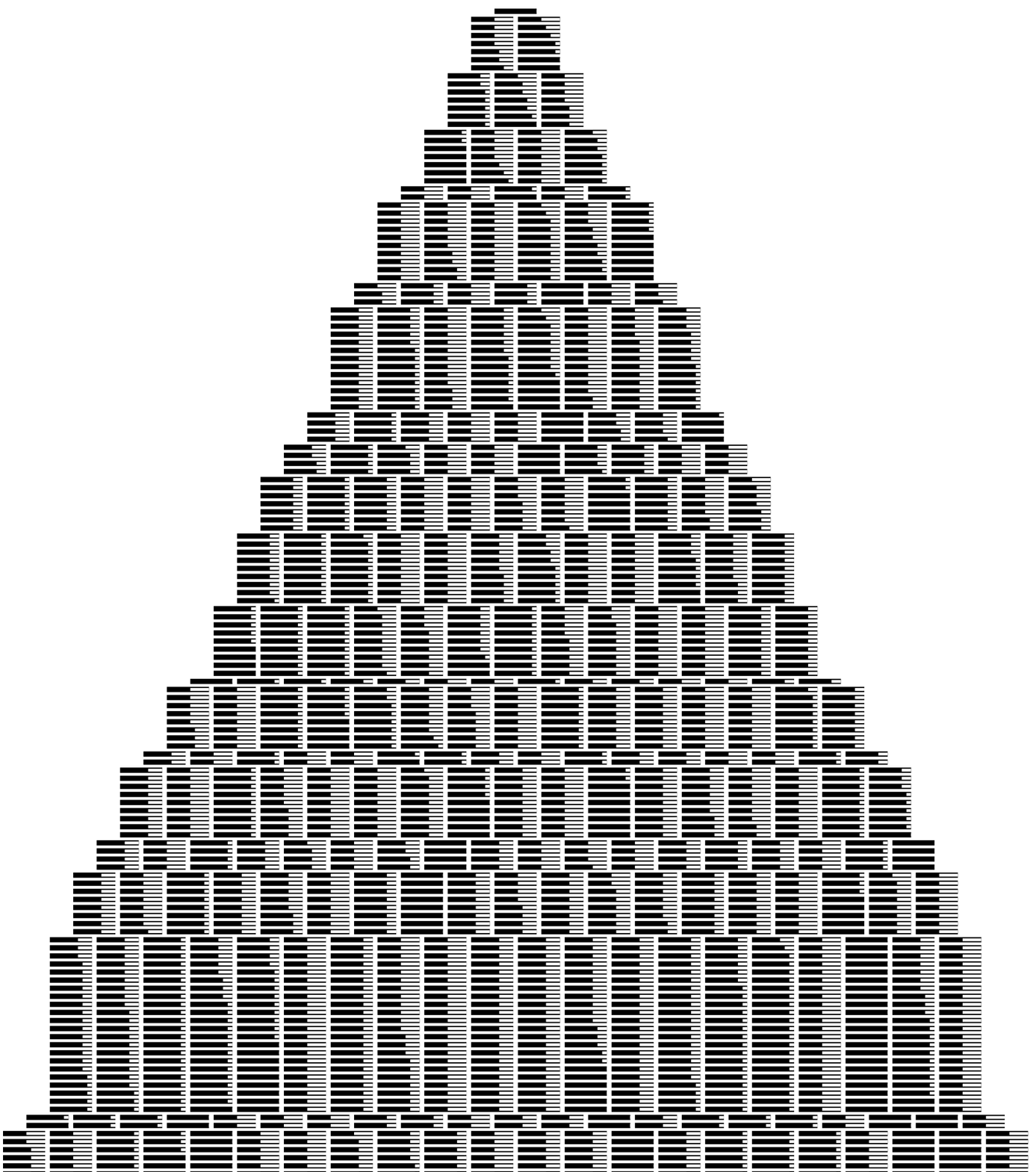




B tree example (continued)



B tree growth



Skip Lists

Generalize skip lists

- nodes hold M records
- use sequential search within nodes

INSERT algorithm

- node not full: add record to node
- node full:
  - split into two half-full nodes
  - give new node t links
    - with probability 1/M^t

Bottom line

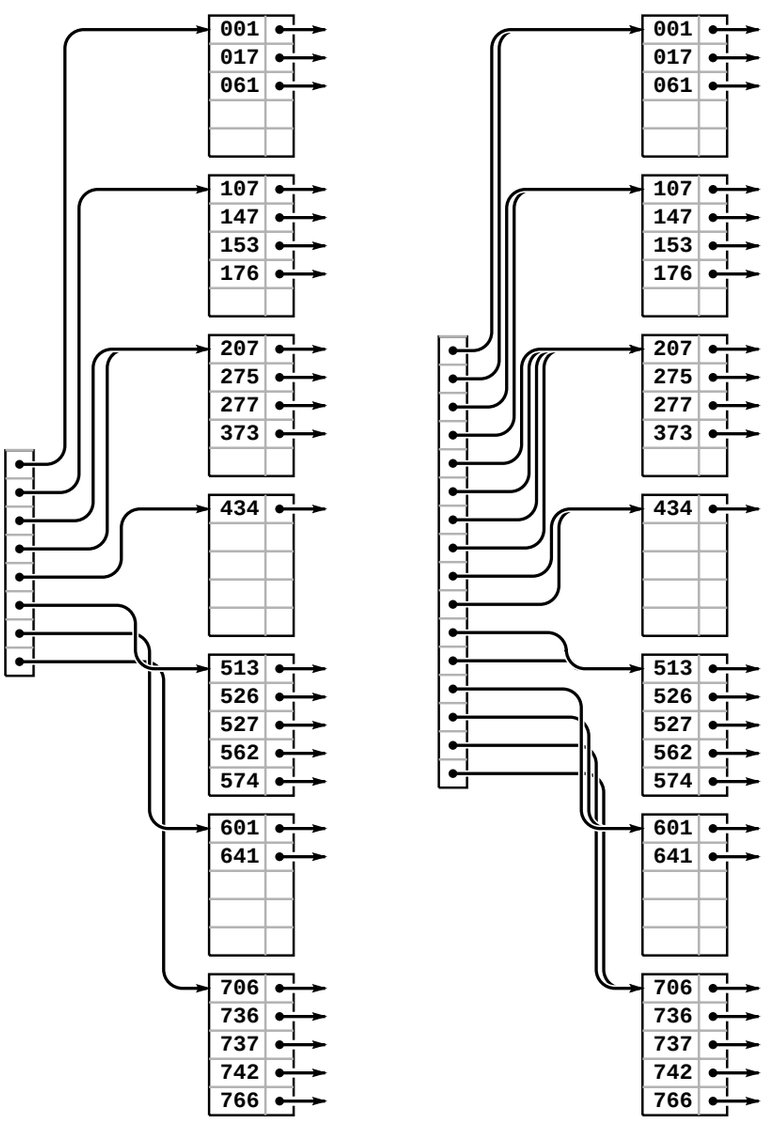
- roughly log_M N page accesses
- 3 or 4 in practice
- randomized algorithm OK?

Key-indexed directory

Use leading t bits of keys to find page with key

capacity: M*2^t

To increase capacity, double directory



Problems when >M keys have same leading bits

Ex: M+1 equal keys

Extendible hashing

Generalize trie, hash search

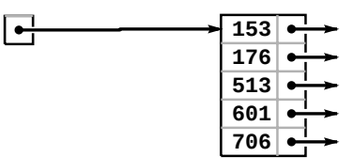
Hash keys to make them random bits
Build two-level trie on hashed keys
    directory size 2^t
    page size M

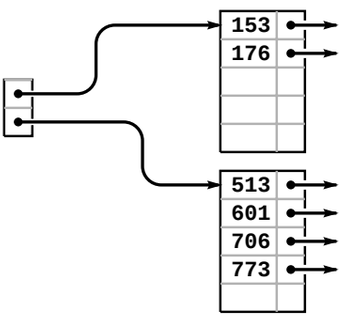
SEARCH
    i = first t bits of hashed key
    directory entry i gives page name
    search that page for the key

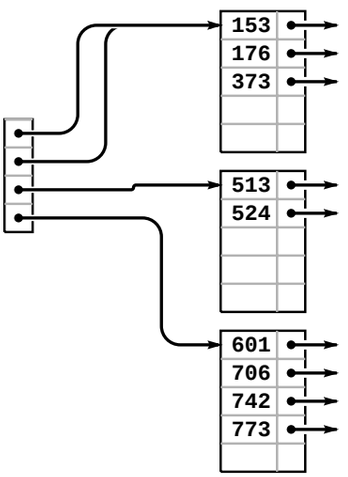
INSERT
    same as search
    put key on page accessed
    if page full
        split into two pages
    may need to split directory

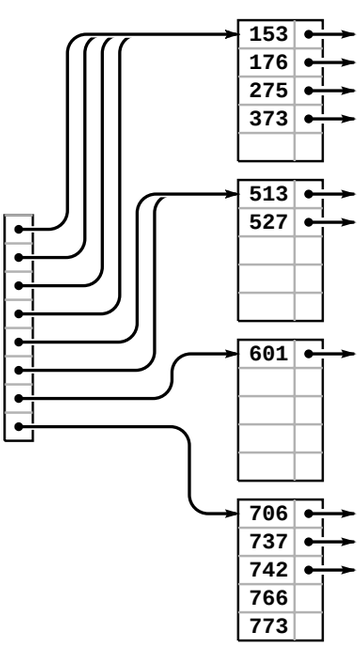
Bottom line
    2 page accesses per search
    44 percent extra space
    randomized algorithm OK?

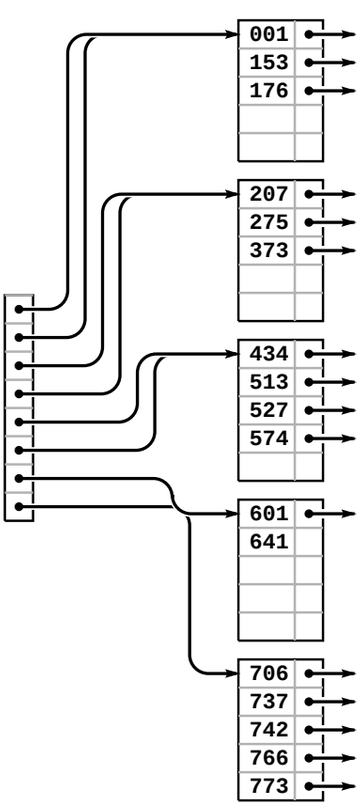
Extendible hashing example

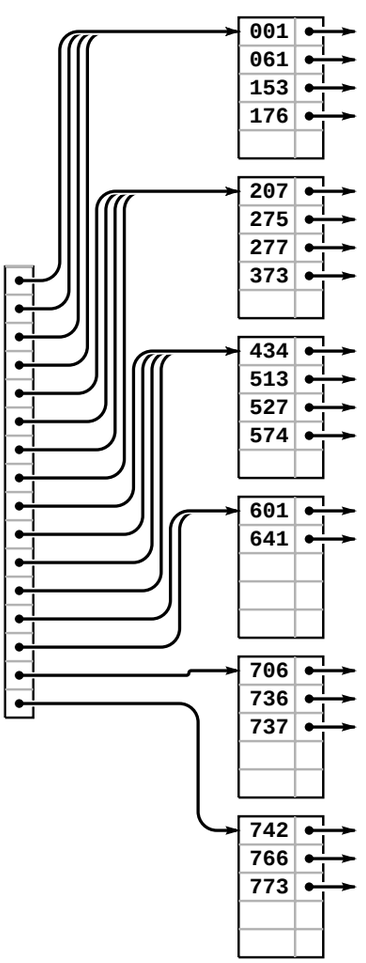
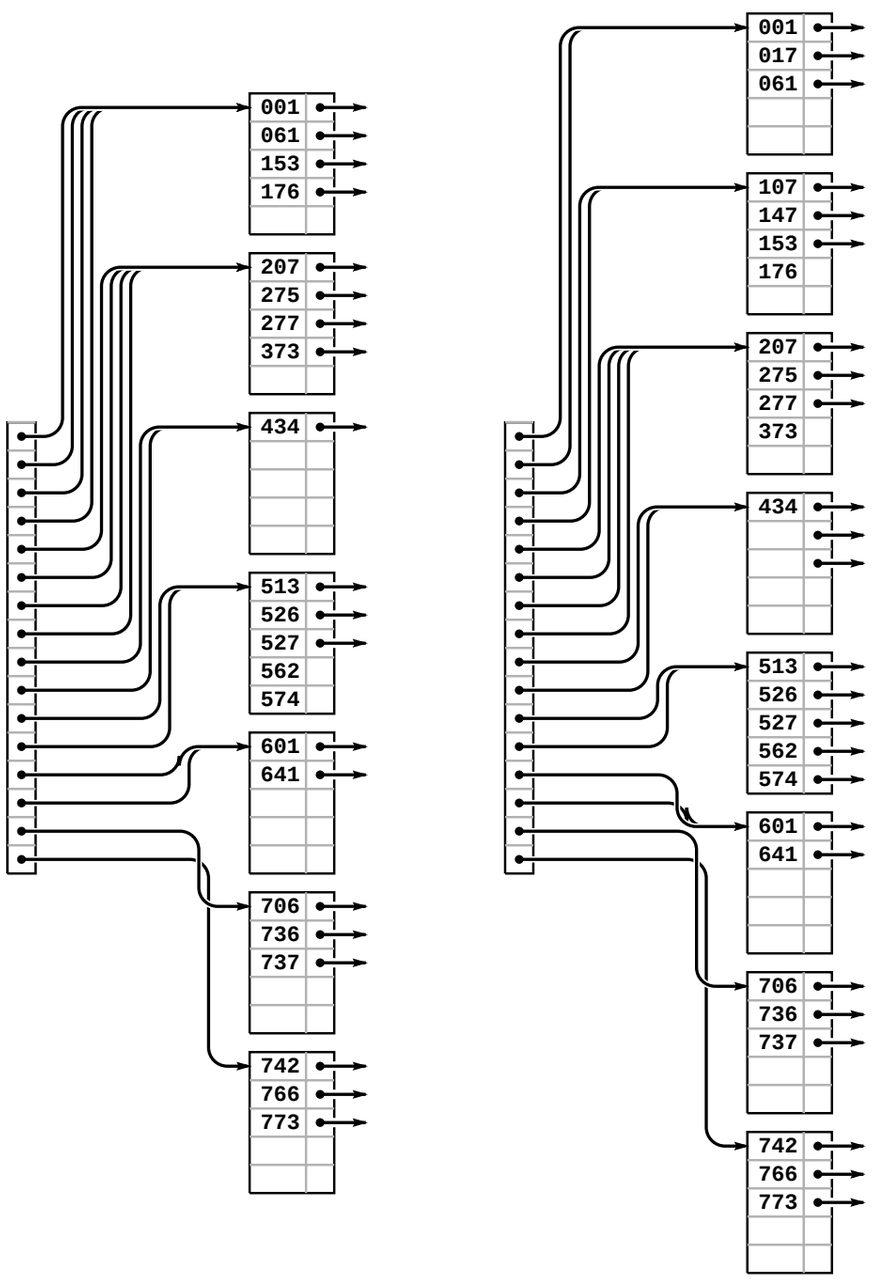












Extendible hashing example (continued)