Searching

- Searching is for the purpose of retrieving a record to be updated or to be used in computation
- Search strategies range in complexity from **O**(1) to **O**(n).

1

Basic Terminology

- Same basic terminology as sorting:
  - record
  - file (of size n)
  - key
    - external key
    - external vs internal searching
    - primary vs. secondary

2

Searching: Points to consider

- size of data
- distribution of data
- reuse of existing code
- programmer time
- frequency of searching
- number of search types

3

Searching: Points to consider

- Search strategies exploit the file organization to <u>efficiently</u> find item.
- Common to search for items not in file - Usually done to prevent duplicates

4

Search strategies:

- Often rely on the data to be sorted
- Sometimes use nonsorted data
- Nonsorted does not necesserily mean unorganized.

5

Search Strategies Using Sorted Data

- Sequential Search
  - Naïve or brute-force
  - **O**(n/2) on average
- Binary Search
- Interpolation Search
- Indexed Sequential Search
- Search Trees
- Must maintain sorted order when doing inserts or deletes

6

Sequential Search

(on sorted data)

- A brute force method
- Naive and simple
- **O**(n/2) on average
- Stop when get to correct location
- Works even if item not present **O**(n/2)

7

Sequential Search Example

123456789	Larry	17 Elm St.	Burbank, CA
178940312	Moe	. . .	. . .
256789201	Curly	. . .	...
342765119	. . .	. . .	. . .

- Deleting “Curly” requires shifting records 4 through n
- OR could use scheme with marked deletes
- This does not affect retrieval time, but must maintain in sorted order.
- Sequential Search also works with linked lists

8

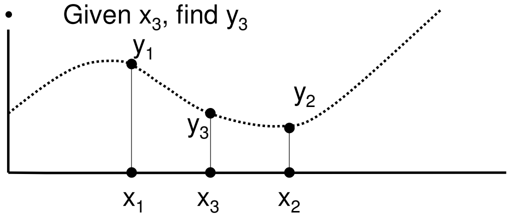
Binary Search

- Mechanics previously covered
- Only suitable for static allocation
- Need to do random access (no links)
- Only suitable for "small" files
- **O**(log n) on average

9

Interpolation Search

- (cousin to Binary Search)
- Uses linear interpolation on key
- Given x_3, find y_3



- A line is reasonable approx.. of curve between y_1 and y_2

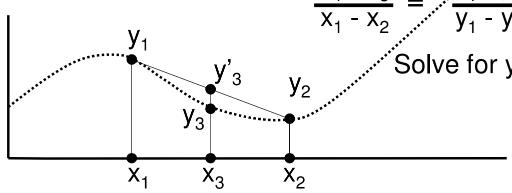
10

Interpolation Search

- Uses linear interpolation on key
- y'_3 is an estimate for y_3

$$\frac{x_1 - x_3}{x_1 - x_2} = \frac{y_1 - y'_3}{y_1 - y_2}$$

Solve for y'_3



11

Interpolation Search

- Uses key (x's) to find array index (y's)
- Answer y'_3 is an estimate of y_3
- Needs keys uniformly distributed.
- **O**(log log n)
- If not, can deteriorate to **O**(n).

12

Indexed Sequential Search

45	3
261	7
411	11
652	16

7	rest of record
23	
30	
45	
110	
145	
202	
261	
270	
300	
369	
411	

Space is **O**(n+k)

Find 369

Indexed Sequential Search

- Needs extra space for index: **O**(k)
- Uses total space of **O**(n+k)
- Creates environment to optimize simple strategy: Sequential Search
- Index identifies a small portion of main file in which to do Sequential Search
- Index is relatively small - can do Sequential or Binary Search

Indexed Sequential Search

- Main file suitable for static or dynamic alloc.
- Pick indices to get file pieces the same size
- Efficiency depends
  - on size of index
  - on sizes of pieces of file
- After lots of inserts/deletes, file may be inefficient so rebuild.

Search Strategies Using Sorted Data

- Sequential Search
  - Naïve or brute-force
  - **O**(n/2) on average
- Binary Search
- Interpolation Search
- Indexed Sequential Search
- Search Trees
- Must maintain sorted order when doing inserts or deletes

16

Search Trees

- Start with a tree having the **SearchTree Characteristic**
- Search efficiency is the height of the tree. If tree is short, bushy and well balanced then this is **O**(log n).
- How to we get short, bushy trees?
- What do we mean by balanced?

17

Simple Search Trees

- Starts with sorted file
- Intuitive definition of balance
- Relaxed in maintenance.
- Makes no real effort to keep balance.
- Over time, with inserts & deletes, tree
  - is unbalanced,
  - is less bushy,
  - is more general
  - is less efficient to search

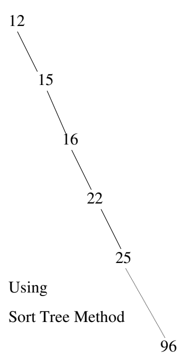
18

Simple Search Trees

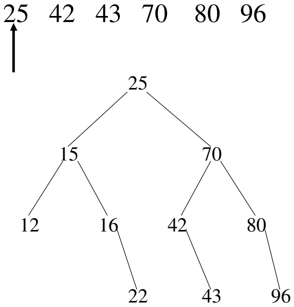
Not a Binary Sort Tree - built differently.
- Start with a sorted file
- Use the middle item as the root.
- The left half becomes the left subtree
- The right half becomes right subtree
- This is recursive.

19

Recall: 12 15 16 22 25 42 43 70 80 96



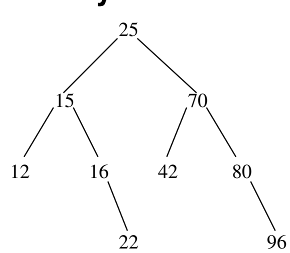
Using
Sort Tree Method



Using Search Tree Method

20

Binary Search Tree: Deletion



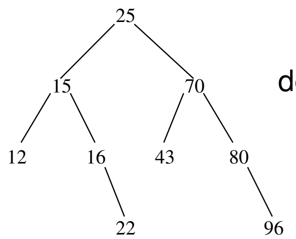
Using Search Tree Method

Tree with 43 deleted

Case I - no children

21

Binary Search Tree: Deletion



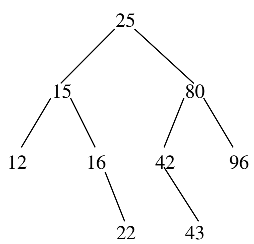
Tree with 42 deleted - replace with child

Case II - one child

Using Search Tree Method

22

Binary Search Tree: Deletion



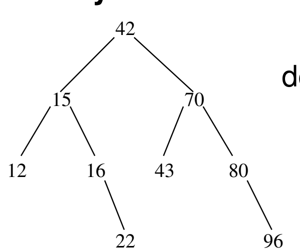
Tree with 70 deleted - replace with 80

Case III - two children

Using Search Tree Method

23

Binary Search Tree: Deletion



Tree with 25 deleted -replace with 42

Case III - two children

Using Search Tree Method

24

Simple Search Trees - Updates

- Insertions same as Binary Sort Tree
- To delete there are three cases:
  - (1) No children - just delete node
  - (2) 1 child - replace value with child and delete child
  - (3) 2 children - replace with inorder successor and delete IOS
- When performance is poor (or at designated times) the tree is rebuilt

25

Search Trees

- **Alternatively** use formulaic definition of balance - maintain aggressively
- AVL trees (Height Balanced) trees)

$$|H_{LST} - H_{RST}| <= 1$$

- B-Trees
  - Red-Black Trees
  - Splay Trees
  - 2-3 Trees
  - 2-3-4 Trees

26

Search Trees

- Different tree types use different strategies to maintain a short, bushy tree. A more rigid rule usually means more work to maintain the tree in that conformation.
- AVL Trees covered in Horowitz & Sahni
- B-Trees: See Files and Databases: an Introduction, P. D. Smith and G. M. Barnes, Addison Wesley

27

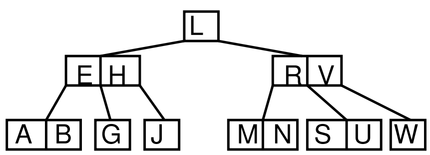
B-Trees

- A B-Tree is an m-ary tree
- Root has 2 or more children (root may be a leaf)
- Other nodes have $\lceil M/2 \rceil$ children
- All leaves are at the same level
- A node with k+1 children 0,1,…,k has k records 1, 2, …, k

Left	Data	Right

p_0	r_1	p_1	r_2	p_2	r_3	p_3

28



M = 3

$\lceil M/2 \rceil = \lceil 3/2 \rceil = 2$

Min children 2

Min records 1

Max children 3

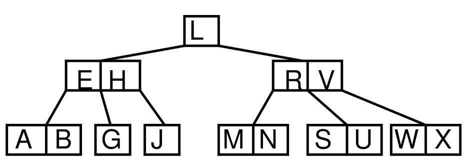
Max records 2

Height = 2

Insert X:

Goes in Node with W

29



M = 3

$\lceil M/2 \rceil = \lceil 3/2 \rceil = 2$

Min children 2

Min records 1

Max children 3

Max records 2

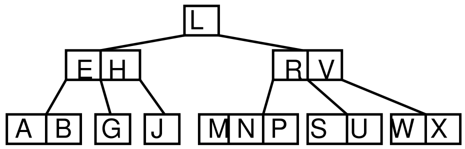
Height = 2

Insertion of X complete

Insert P:

Goes in Node with MN

30



M = 3

$\lceil M/2 \rceil = \lceil 3/2 \rceil = 2$

Min children 2

Min records 1

Max children 3

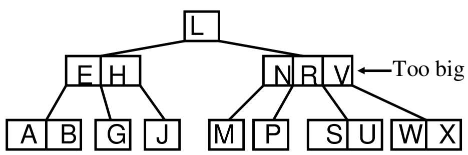
Max records 2

Height = 2

P goes in Node with MN - Full

M N P $\Rightarrow$ Split Node

31



M = 3

$\lceil M/2 \rceil = \lceil 3/2 \rceil = 2$

Min children 2

Min records 1

Max children 3

Max records 2

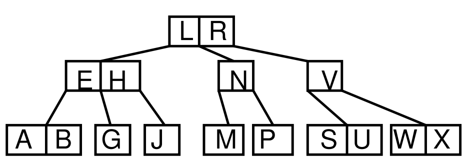
Height = 2

Move N up to parent

Goes in Node with RV - Full

N R V $\Rightarrow$ Split Node

32



M = 3

$\lceil M/2 \rceil = \lceil 3/2 \rceil = 2$

Min children 2

Min records 1

Max children 3

Max records 2

Height = 2

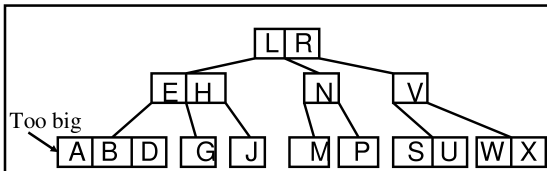
Move R up to Parent

Goes in Node with L

Insertion of P complete

Next: Insert D

33



M = 3

$\lceil M/2 \rceil = \lceil 3/2 \rceil = 2$

Min children 2

Min records 1

Max children 3

Max records 2

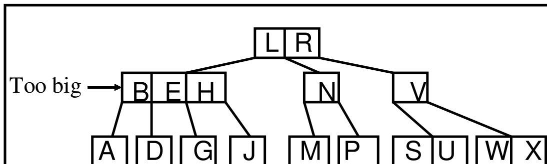
Height = 2

Insert D:

Goes in Node with AB - Full

A B D ⟹ Split Node

34



M = 3

$\lceil M/2 \rceil = \lceil 3/2 \rceil = 2$

Min children 2

Min records 1

Max children 3

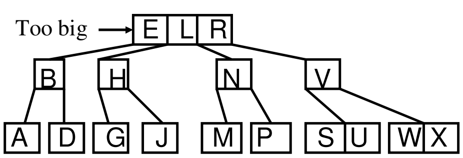
Max records 2

Height = 2

B moves up to parent - EH -Full

B E H ⟹ Split Node

35



M = 3

$\lceil M/2 \rceil = \lceil 3/2 \rceil = 2$

Min children 2

Min records 1

Max children 3

Max records 2

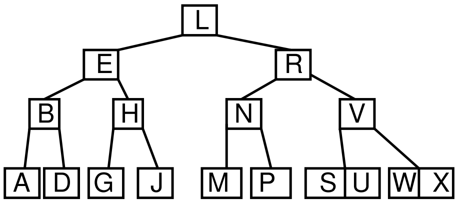
Height = 2

Move E up to Parent

Goes in Node with LR - Full

E L R ⟹ Split Node

36



M = 3

$\lceil M/2 \rceil = \lceil 3/2 \rceil = 2$

Min children 2

Min records 1

Max children 3

Max records 2

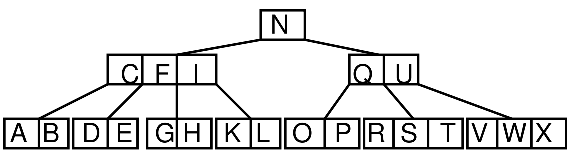
Height = 3

Move L up to Parent

Goes in New Node – adds level

Insertion of D complete

37



M = 5

$\lceil M/2 \rceil = \lceil 5/2 \rceil = 3$

Min children 3

Min records 2

Max children 5

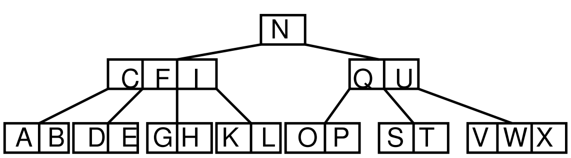
Max records 4

Height = 2b

Delete R:

R is easy to delete from Node RST

38



M = 5

$\lceil M/2 \rceil = \lceil 5/2 \rceil = 3$

Min children 3

Min records 2

Max children 5

Max records 4

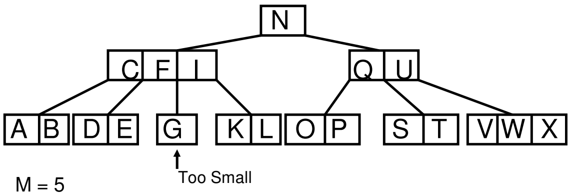
Height = 2

Deletion of R complete

Next Delete H:

Delete H from Node GH - Too Small

39



M = 5

$\lceil M/2 \rceil = \lceil 5/2 \rceil = 3$

Min children 3

Min records 2

Max children 5

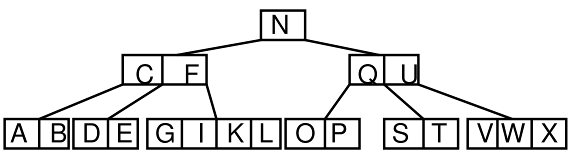
Max records 4

Height = 2

G combines with sibling KL

Demote I from Parent Node

40



M = 5

$\lceil M/2 \rceil = \lceil 5/2 \rceil = 3$

Min children 3

Min records 2

Max children 5

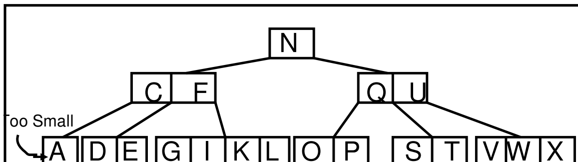
Max records 4

Height = 2

Deletion of H complete

Next delete B:

41



M = 5

$\lceil M/2 \rceil = \lceil 5/2 \rceil = 3$

Min children 3

Min records 2

Max children 5

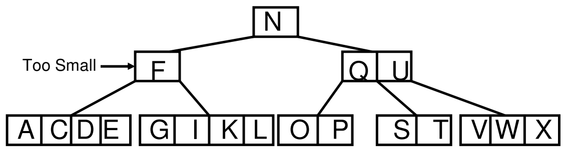
Max records 4

Height = 2

Consolidate with sibling DE

Demote C from parent

42



M = 5

$\lceil M/2 \rceil = \lceil 5/2 \rceil = 3$

Min children 3

Min records 2

Max children 5

Max records 4

Height = 2

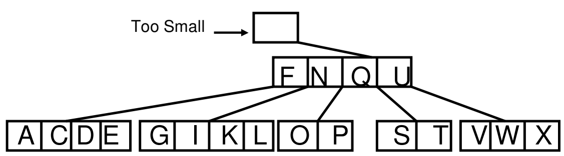
C demoted

Parent F too small

Combine with sibling QU

Demote N from parent

43



M = 5

$\lceil M/2 \rceil = \lceil 5/2 \rceil = 3$

Min children 3

Min records 2

Max children 5

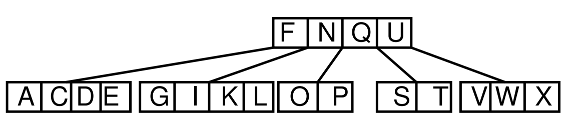
Max records 4

Height = 2

Parent (root) too small

Delete root

44



M = 5

$\lceil M/2 \rceil = \lceil 5/2 \rceil = 3$

Min children 3

Min records 2

Max children 5

Max records 4

Height = 1

Root deleted

Deletion of B complete

45

Search Strategies Using NonSorted Data

- Sequential Search
- Transposition
  - Move-To-the-Front
- Hashing

46

Sequential Search

- A brute force method
- Naive and simple
- **O**(n/2) if item in file
- **O**(n) if not

47

Transposition

- The data is not sorted.
- Data ordered by frequency of access.
- Most frequently used records at front.
- Uses Sequential Search efficiently.
- Much better than **O**(n/2)

48

Transposition

Able	A	A	(4)S	S	S
Thomas	S	(3)S	A	A	A
(1)Smith	T	J	J	J	J
Jones	(2)J	T	T	T	B
Brown	B	B	B	(5)B	T

Thomas never accessed - moves to back
Smith accessed most - moves to front

49

Transposition

- Best insertion point depends on how frequently item will be accessed
- Safe place in file midpoint
- If using array, inserting and deleting both require shifting
- Linked list - no shifting

50

Transposition

- May be able to guess initial ordering
- What if can't guess initial ordering?
- Also, in many applications, which records are freq. accessed changes
- So, whenever a record is accessed it is swapped with one in front of it
- This is a simple linked list problem involving 3 ptrs

51

Transposition

- Creates order by frequency of access:
- Frequently accessed records move closer to front of file
- Infrequently used records move to back
- Responsive to changes in frequency of access patterns.
- Not suitable for applications needing for sorted data.
- How would you restore the ordering?

52

Move-To-the-Front

- (cousin to Transposition)
- When data is accessed,
  - It is moved right to front of file.
  - Great, if it will be frequently accessed from now on.
  - Terrible, if infrequently used record.
- Tenenbaum likes this better
- Responds faster to pattern changes

53

Hashing

- Attempts to be an **O**(1) Search Strategy.
- Does single calculation on key to get address: address = $\mathbf{h}_f$(key)
- Buckets
- Address is area in hash table:
  - Array
  - Hard disk
  - Other linear,contiguous space

54

Hashing: h_f

4 main strategies for hash functions

(M is table size)

(1) Multiplication - like a random number generator

addr = (A*key + B)/M

(2) Division - most popular and effective - Works best if divisor is prime

addr = key % M

55

Hashing: h_f

(3) Folding - break key into pieces which are combined using logical operations: **and, or, exor**, etc

ABC	character string
31 32 33	ASCII equivalent (HEX)
0011 0001 0011 0010 0011 0011	
001 100 010 011 001 000 110 011	

(4) Mid-Square - Calculate key*key and use mid portion of result as address. Not a good method.

56

Hashing Example

15429	Screwdriver
38416	Claw Hammer
27154	Mallet
87216	Ballpeen Hammer
95071	Phillips Screwdriver
66782	Vise Grips
66729	Needlenose Pliers
52347	Allen Wrench
48917	Air Chisel
73584	Torque Wrench
18499	Diagonal Cutters
38060	Socket Set
42066	Hack Saw
78816	File
60284	Drill Bits

0	
16	Claw Hammer
17	Air Chisel
29	Screwdriver
47	Allen Wrench
54	Mallet
60	Socket Set
66	Hack Saw
71	Phillips SD
82	Vise Grips
84	Torque Wrench
99	Diagonal Cutters

These items have NOT been stored in table:

☹ Ballpeen Hammer, Needlenose Pliers, File, Drill Bits

Hashing

- What happens if two different keys generate the same address - called a clash or **collision**. Must resolve.

- A **secondary collision** is one that is strictly due to a collision handling method.

58

Hashing

- First: tweak the hash function to reduce collisions

- Handle collisions by two primary methods
  (1) Rehashing
  (2) Chaining
    With or without overflow area

59

Hashing: Rehashing

- Recalculate address
  - Use same hash function
  - Use simple variation.
- Often can only apply a limited number of times.
- Works well with Multiplication type hash functions

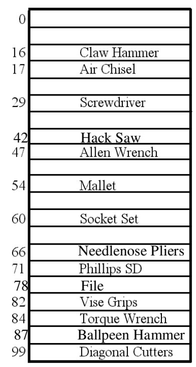
60

Hashing Example - Rehashing

15429 Screwdriver
38416 Claw Hammer
27154 Mallet
87216 Ballpeen Hammer
95071 Phillips Screwdriver
66782 Vise Grips
66729 Needlenose Pliers
52347 Allen Wrench
48917 Air Chisel
73584 Torque Wrench
18499 Diagonal Cutters
38060 Socket Set
42066 Hack Saw
78816 File
60284 Drill Bits

This item has NOT been stored in table:

☹Drill Bits
....
☺Hack Saw



Hashing: Chaining

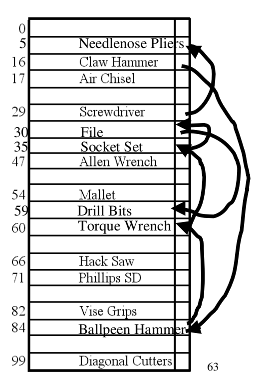
- Create a linked list of colliding records.
- Choice of list structure affects efficiency and depends on application.
- If separate overflow area:
  - Need more space **OR**
  - Hash table smaller but no secondary collisions.
- If no separate overflow area
  - Risk of secondary collisions.

62

Hashing Example - Chaining

15429 Screwdriver
38416 Claw Hammer
27154 Mallet
87216 Ballpeen Hammer
95071 Phillips Screwdriver
66782 Vise Grips
66729 Needlenose Pliers
52347 Allen Wrench
48917 Air Chisel
73584 Torque Wrench
18499 Diagonal Cutters
38060 Socket Set
42066 Hack Saw
78816 File
60284 Drill Bits

☹ Socket Set,
Torque Wrench



63

Hashing: Linear Probing

- Linear probing is a degenerate form of chaining.
- Move forward through file until find available space.
- Move by one or larger, fixed amount.
- In very full table, deteriorates to **O**(n)

64

Hashing Example - Linear Probing

15429	Screwdriver
38416	Claw Hammer
27154	Mallet
87216	Ballpeen Hammer
95071	Phillips Screwdriver
66782	Vise Grips
66729	Needlenose Pliers
52347	Allen Wrench
48917	Air Chisel
73584	Torque Wrench
18499	Diagonal Cutters
38060	Socket Set
42066	Hack Saw
78816	File
60284	Drill Bits

☺ Air Chisel

0	
16	Claw Hammer
17	Ballpeen Hammer
18	Air Chisel
19	File
29	Screwdriver
30	Needlenose Pliers
47	Allen Wrench
54	Mallet
60	Socket Set
66	Hack Saw
71	Phillips SD
82	Vise Grips
84	Torque Wrench
85	Drill Bits
99	Diagonal Cutters

Hashing

- Hashing tends to waste space.
- As utilization increases so do collisions.
- > aprox. 65% causes problems
- One way to reduce collisions is to enlarge table - wasting more space.
- Not suitable for applications needing sorted data.
- Division with Linear Probing - common

66