Coders - The Making of a New Tribe Chapter 3...

What exactly is a bug? A bug is an error in your code, something mistyped or miscreated, that throws a wrench into the flow of a program. They're often incredibly tiny, picky details. One sunny day in Brooklyn, I met in a cafe with Rob Spectre, a lightly grizzled coder who whipped out his laptop to show me a snippet of code written in the language Python. It contained, he said, a single fatal bug. This was the code:

```
stringo = [rsa,rsa1,lorem,text]
_output_ = "backdoor.py"
_byte_ = (_output_) + "c"

if (sys.platform.startswith("linux"))
    if (commands.getoutput("whoami")) != "root":
       print("run it as root")
       sys.exit() #exit
```

The bug? It's on the fourth line. The fourth line is an "if" statement-if the program detects that `(sys.platform.startswith ("linux"))` is "true," it'll continue on with executing the commands on line five and onward.

The thing is, in the language Python, any "if" line has to end with a colon. So line 4 should have been written like this . . .

`if (sys.platform.startswith("linux")):`

That one tiny missing colon breaks the program.

"See, this is what I'm talking about," Spectre says, slapping his laptop shut with a grimace. "The distance between looking like a genius and looking like an idiot in programming? It's one character wide."

Other Trees

1 Other Trees

- Heaps (Priority Trees)
- AVL Trees
- 2–3 and 2–3–4 Trees
- Red-Black Trees
- Optimal Search Trees
- B-Trees
- $B^{\star}$-Trees and B^{+}-Trees
- Quadtrees and Octrees
- Finger Trees
- Splay Trees
- More....

NOTE: Tree programming assignment will be sent out soon.

1.1 Heaps (Priority Trees)

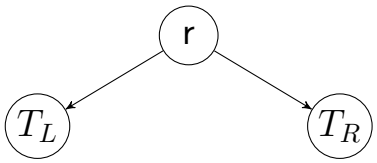
We will examine these in detail soon.

1.2 2–3 Trees

A 2–3 tree is a tree in which each internal node has either two or three children, and all leaves are at the same level.

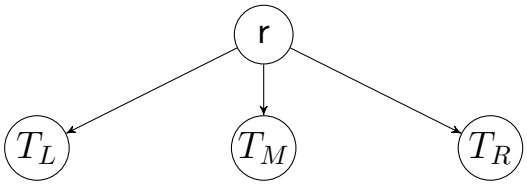
A 2–3 tree is not a binary tree, because a node can have three children. 2–3 trees resemble full binary trees.

1. T is empty (a 2–3 tree of height 0).
2. T is of the form



   where r is a node that contains one data item and T_L and T_R are both 2–3 trees, each of height $h - 1$. In this case, the search key in r must be greater than each search key in the left subtree T_L and smaller than each search key in the right subtree T_R.

3. T is of the form



   where r is a node that contains two data items and T_L, T_M, and T_R are 2–3 trees, each of height $h - 1$. In this case, the smaller search key in r must be greater than each search key in the left subtree T_L and smaller than each search key

in the middle subtree T_M. The larger key in r must be greater than each search key in the middle subtree T_M and smaller than each search key in the right subtree T_R.

1.3 2–3–4 Trees

A 2–3–4 tree is a tree in which each internal node has either two, three, or four children, and all leaves are at the same level.

1.4 Red–Black Trees

A 2–3–4 tree can be represented by a special binary search tree—a red–black tree.

Red–Black trees are in the STL (Standard Template Library).

We may examine them in more detail later.

1.5 AVL Trees

An AVL tree (after Adel'son-Vel'skii and Landis, its inventors) is a balanced binary tree. Since the heights of the left and right subtrees of any node in a balanced binary tree differ by no more than one (1), you can search an AVL tree almost as efficiently as a minimum-height binary search tree.

- AVL trees maintain a height close to the minimum.
- Rotations restore the balance.

We may examine them in more detail later.

AVL Tree Balancing

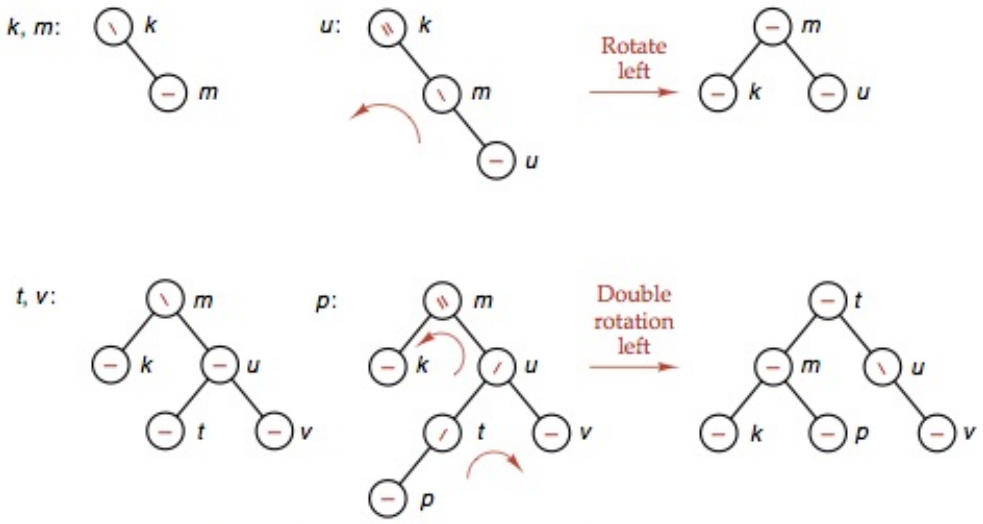


Figure 10.21. AVL insertions requiring rotations

We may examine them in more detail later.

1.6 Optimal Search Trees

We have been interested in trees that have a frequency of access that is equal for all nodes. All *keys* are equally likely. There are cases in which the access probabilities are known. For these trees, the tree remains the same—no insertions or deletions. An optimal search tree can be constructed for such cases.

Optimal search trees are used in some compilers to determine if a token is a keyword or not. For example, in C++, the keywords `for` and `if` show up much more frequently than `namespace` or `operator`, so would be located closer to the root.

1.7 B-Trees

Many databases are implemented using B-Trees and have the following characteristics; n is said to be the *order* of the B-Tree.

A tree can be subdivided into subtrees and the subtrees represented as units which are all accessed all together. These subtrees are called *pages*. The figure below shows a binary tree subdivided into pages, each page consisting of seven nodes.

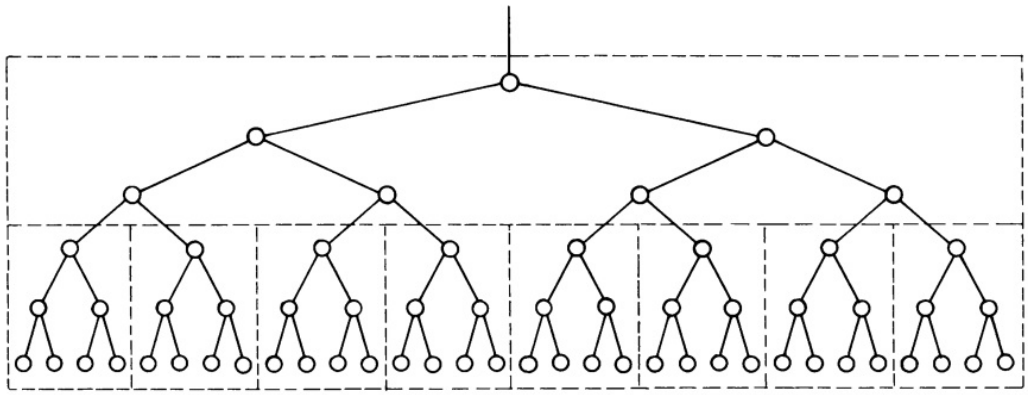


Figure 1: A binary tree subdivided into pages

1. Every page contains at most $2n$ items (keys).
2. Every page, except the root page, contains at least n items.
3. Every page is either a leaf page (no descendants), or it has $m + 1$ descendants, where m is its number of keys on the page.
4. All leaf pages appear at the same level.

B-tree of order 2 with 3 levels. All pages contain 2, 3, or 4 items; the exception is the root which is allowed to contain a single item only. All leaf pages appear at level 3.

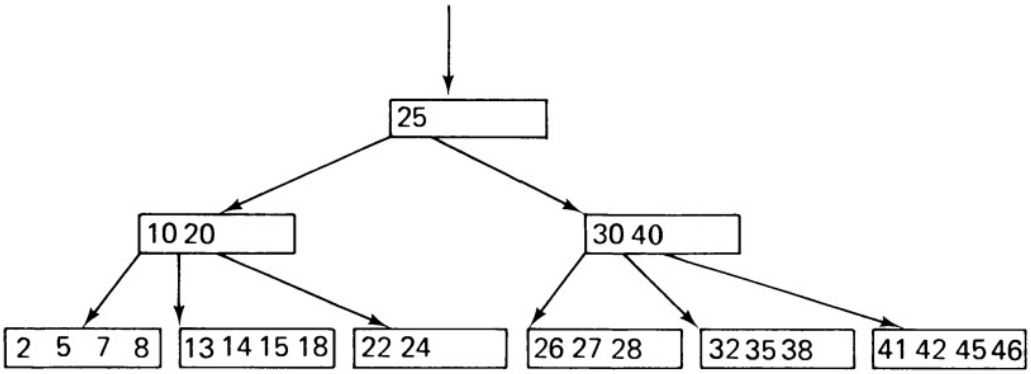


Figure 2: B-tree of order 2

1.8 B* and B$^+$-Trees

B* (B-star) and B$^+$-Trees are variations of B-trees.

1.9 Quadtrees and Octrees

Efficient representation of images (and other information) in two and three dimensions. Consider the following example:[1]

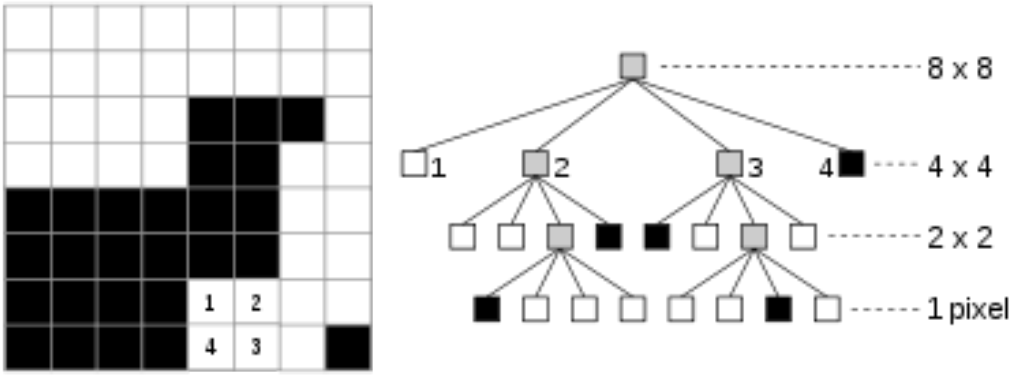


Figure 3: Quadtree

[1] `https://en.wikipedia.org/wiki/Quadtree`