Advanced Tree Data Structures

Nelson Padua-Perez
Chau-Wen Tseng

Department of Computer Science
University of Maryland, College Park

Overview

- Binary trees
  - Balance
  - Rotation
- Multi-way trees
  - Search
  - Insert
Tree Balance

- **Degenerate**
  - Worst case
  - Search in $O(n)$ time

- **Balanced**
  - Average case
  - Search in $O(\log(n))$ time

**Question**
- Can we keep tree (mostly) balanced?

**Self-balancing binary search trees**
- AVL trees
- Red-black trees

**Approach**
- Select invariant (that keeps tree balanced)
- Fix tree after each insertion / deletion
  - Maintain invariant using rotations
- Provides operations with $O(\log(n))$ worst case
AVL Trees

Properties
- Binary search tree
- Heights of children for node differ by at most 1

Example

AVL Trees

History
- Discovered in 1962 by two Russian mathematicians, Adelson-Velskii & Landis

Algorithm
1. Find / insert / delete as a binary search tree
2. After each insertion / deletion
   a) If height of children differ by more than 1
   b) Rotate children until subtrees are balanced
   c) Repeat check for parent (until root reached)
Red-black Trees

Properties
- Binary search tree
- Every node is red or black
- The root is black
- Every leaf is black
- All children of red nodes are black
- For each leaf, same # of black nodes on path to root

Characteristics
- Properties ensures no leaf is twice as far from root as another leaf

Example
Red-black Trees

- **History**
  - Discovered in 1972 by Rudolf Bayer

- **Algorithm**
  - Insert / delete may require complicated bookkeeping & rotations

- **Java collections**
  - TreeMap, TreeSet use red-black trees

Tree Rotations

- **Changes shape of tree**
  - Move nodes
  - Change edges

- **Types**
  - Single rotation
    - Left
    - Right
  - Double rotation
    - Left-right
    - Right-left
Tree Rotation Example

Single right rotation

Node 4 attached to new parent
Example – Single Rotations

Example – Double Rotations
Multi-way Search Trees

**Properties**
- Generalization of binary search tree
- Node contains 1…k keys (in sorted order)
- Node contains 2…k+1 children
- Keys in jth child < jth key < keys in (j+1)th child

**Examples**

```
5   12
  2   8
```

```
5  8  15  33
  1  3  7  9
      19 21
```

**Types of Multi-way Search Trees**

- **2-3 tree**
  - Internal nodes have 2 or 3 children

- **Index search trie**
  - Internal nodes have up to 26 children (for strings)

- **B-tree**
  - $T = \text{minimum degree}$
  - Non-root internal nodes have $T-1$ to $2T-1$ children
  - All leaves have same depth
**Multi-way Search Trees**

**Search algorithm**
1. Compare key $x$ to 1…$k$ keys in node
2. If $x = \text{some key}$ then return node
3. Else if ($x < \text{key } j$) search child $j$
4. Else if ($x > \text{all keys}$) search child $k+1$

**Example**
- Search($17$)

```
  5   12
 /     \
|       |
1  2  8  17
```

**Insert algorithm**
1. Search key $x$ to find node $n$
2. If ( $n$ not full ) insert $x$ in $n$
3. Else if ( $n$ is full )
   a) Split $n$ into two nodes
   b) Move middle key from $n$ to $n$’s parent
   c) Insert $x$ in $n$
   d) Recursively split $n$’s parent(s) if necessary
Multi-way Search Trees

Insert Example (for 2-3 tree)

- Insert(4)

![Insert Example Diagram](image)

Multi-way Search Trees

Insert Example (for 2-3 tree)

- Insert(1)

![Insert Example Diagram](image)
B-Trees

Characteristics
- Height of tree is $O(\log_T(n))$
- Reduces number of nodes accessed
- Wasted space for non-full nodes

Popular for large databases
- 1 node = 1 disk block
- Reduces number of disk blocks read