CMSC 132:
Object-Oriented Programming II

Advanced Tree Structures

Department of Computer Science
University of Maryland, College Park
Overview

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

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

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

Degenerate binary tree

Balanced binary tree
Tree Balance

**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

Heights of children shown in red
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
Red-black Trees

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
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 $j^{th}$ child < $j^{th}$ key < keys in $(j+1)^{th}$ child

**Examples**

```
5     12
  
2     8     17

5  8  15  33
  
1  3  7  9  19  21  44
```
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**
  - \( T = \) 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 = some key then return node
3. Else if (x < key j) search child j
4. Else if (x > all keys) search child k+1

Example
Search(17)
Multi-way Search Trees

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 )

Before: 5 12
        /   |
       2   8 17

After: 5 12
       /   |
      2 4 8 17
Multi-way Search Trees

Insert Example (for 2-3 tree)

Insert( 1 )

Split node

Split parent
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
Indexed Search Tree (Trie)

- Special case of tree
- Applicable when
  - Key $C$ can be decomposed into a sequence of subkeys $C_1, C_2, \ldots, C_n$
  - Redundancy exists between subkeys
- Approach
  - Store subkey at each node
  - Path through trie yields full key
- Example
  - Huffman tree
Tries

- Useful for searching strings
  - String decomposes into sequence of letters
  - Example
    - “ART” ⇒ “A” “R” “T”

- Can be very fast
  - Less overhead than hashing

- May reduce memory
  - Exploiting redundancy

- May require more memory
  - Explicitly storing substrings
Standard Trie Example

For strings

{ bear, bell, bid, bull, buy, sell, stock, stop }
Word Matching Trie

- Insert words into trie
- Each leaf stores occurrences of word in the text
Tries and Web Search Engines

- Search engine index
  - Collection of all searchable words
  - Stored in compressed trie

- Each leaf of trie
  - Associated with a word
  - List of pages (URLs) containing that word
    - Called occurrence list

- Trie is kept in memory (fast)
- Occurrence lists kept in external memory
  - Ranked by relevance
Computational Biology

DNA
- Sequence of 4 different nucleotides (ATCG)
- Portions of DNA sequence produce proteins (genes)

Genome
- Master DNA sequence for organism
- For Human
  - 46 chromosomes
  - 3 billion nucleotides
DNA the molecule of life

Trillions of cells

Each cell:
- 46 human chromosomes
- 2 meters of DNA
- 3 billion DNA subunits (the bases: A, T, C, G)
- Approximately 30,000 genes code for proteins that perform most life functions
Tries and Computational Biology

- **ESTs**
  - Fragments of expressed DNA
  - Indicator for genes (& location)
  - 5.5 million sequences at NIH

- **ESTmapper**
  - Build suffix trie of genome
    - 8 hours, 60 Gbytes
  - Search for ESTs in suffix trie
    - 11 hours w/ 8 processor Sun

- Search genome w/ BLAST
  - 5+ years (predicted)