Data Structures

Michael J. Watts

http://mike.watts.net.nz

Lecture Outline

- Arrays
- Matrices
- Structures
- Stacks
- Queues
- Trees

Introduction

- Selection of an appropriate data structure is an important part of programming
  - Efficiency
  - Flexibility
- Choice depends on problem
  - Rate of new data acquisition / insertion
  - Type of data being stored
  - Desired method of data access

Arrays

- Basic data structure for most programming languages
- Collection of data values
- Usually a single data type
  - cf MATLAB cell arrays
- Contents accessed by index numbers
- Problems with searching for specific elements
- Good for fixed numbers of items

Matrices

- Array of arrays
- Basis of MATLAB
- One dimensional matrices are arrays
  - Row / column vectors
- Can be > 2D
- Accessed via row / column indices
- Same problems with searching as arrays
- Makes certain mathematical operations easier

Structures

- Collection of named pieces of data
- Multiple data types within a structure
- Elements within a structure are called fields
- Contents accessed by field name
- Good for grouping related items together

Stacks

- Like a stack of plates
- Oldest items are at the bottom
- Newest items are at the top
- New items are 'pushed' onto the top of the stack
- Retrieved items are 'popped' off of the top of the stack
- First in, last out data structure

Stacks

- Often used to provide temporary storage of data values
- Can't be searched
  - Have to pop each value out to find the one you're looking for
- Simple to implement
- Can be used for evaluating expressions

Queues

- Like a queue at the supermarket
- Sequential data structure
- Oldest items are at the front
- Newest items are at the back
- Elements are 'enqueued' at the end
- Elements are 'dequeued' at the front
- First in, first out data structure

Queues

- Used to control access to finite resources
  - Petrol pumps, checkouts, printers
- Sequential access only
- Unordered
- Problems with searching

Trees

- Way of storing data values in order
- Two dimensional structure
- Collection of nodes and edges
  - Nodes are data items
  - Edges connect nodes
- Position of an item in the structure depends on the value of a *key*
- Many types of tree in existence

Trees

- Navigate by the vales of the nodes
- Much faster than sequential search
  - Don't need to examine every item
  - Adding a level to the tree adds just one more comparison
    - A level can have many items
- Search speed scales as to the *log* of *N*
  - *N* is the number of items in the tree

B-Trees

- Binary trees
- Each node has zero or more subtrees
  - Left and right
  - Node without a subtree is a leaf
  - First node is the root
- Values in left subtree are smaller
- Values in right subtree are greater

B-Trees

- Allow for efficient searches
  - Search for value of key
- Often used in indexing
  - Databases, file systems
- Can degenerate
  - Sequential values
  - Becomes a list
    - Inefficient

AVL Trees

- Adel'son-Vel'skii and Landis trees
- Balanced binary trees
- Height between left and right subtree differ by at most one
  - Height measured between bottom-most nodes
- Height difference maintained by rotations
  - Single / double rotate left / right

AVL Trees

- Don't become degenerate
- Always efficient searching
  - Close to the theoretical maximum
- Rotations can be expensive
  - Frequent insertions / deletions
- Other optimisations for in-order iterations

Summary

- Selection of a data structure is problem dependent
- Arrays and structures are built into most programming languages
- Stacks are often used for temporary storage
- Queues control access to a resource
- Trees are efficient for retrieval
- B-Trees can degenerate
- AVL trees are balanced