Containers

hannes@ru.is

Kinds of Standard Containers

- Array
- Dynamic Array
- Linked List
- Stack
- Queue
- Priority Queue
- Tree
- Binary Search Tree
- Dictionary (map)
- Set
- Graph
- Directed Acyclic Graph

Container Algorithms

- Insert
- Remove
- Iterate
- Random Access
- Find
- Sort

Iterators

- The benefits of using iterators:
  - Don‘t break the class encapsulation
  - Simple to use, even for complex containers
    (make them feel more like arrays)

Algorithmic Complexity

- The time spent on running the algorithm may be a function of the number of elements in the container: T = f(n)
- Enough to consider an approximation for f(n) that gives us the „on the order of“ complexity

O(1) – Not dependent on n

O(n) – Loops over elements

O(n2) – Nested loops

O(logn) – Eliminate ½ of elements every step

Pick a container that provides the best of these for the algorithm you will use the most commonly

Container Characteristics

- What is the memory overhead required?
  - E.g. pointers that need to be stored
- Is the data stored contiguously in memory?
  - This has great impact on cache performance

Build Your Own?

- STL
  - **+** Robost, rich, included
  - **-** Slower (generic), memory hog, dynamic allocation, not same on all compilers (use STLPort)
  - → Use when memory is not at premium
- Boost
  - **+** Works around STL problems, solves complex problems (e.g. smart pointers), well documented
  - **-** Huge Lib files, not guaranteed code, lack of backward compatability, particular license

Build Your Own? (cont.)

- Loki

  + Tricks compilers to do things they were not designed to do through „template metaprogramming“

  - Hard to understand

Dynamic Arrays

- Fixed size c-style arrays are good!
  - No new memory allocation
  - Contiguous memory (good for caching)
- Dynamic arrays sometimes needed
  - Buffer can be grown as needed (by n or by doubling)
  - Actually new buffer allocated, then data copied
    - This is costly!
  - Maybe use this at development time and then change to fixed size when we know the biggest size

Linked Lists

- Extrusive List
  - Link structure is separate from the elements
  - + Elements can be in many lists
  - - Link objects dynamically allocated
- Intrusive List
  - Link structure allocated as part of elements
  - + You get links „for free“
  - - Elements stuck in one list at a time

Linked Lists (cont.)

- Circular Lists store the root also as a regular link, where the Next and Prev pointers serve as the Beginning and End pointers
  - Simplifies the algorithms

Dictionaries and Hash Tables

- Dictionaries implemented as:
  - Binary Tree
    - Key-value pair in each node (key sorted)
    - O(logn)
  - Hash Table
    - Fixed size key slot table
    - O(1) in the absence of Collision

Collisions Resolved

- Open Table
  - + Linked list at each index can grow indefinitely
  - - Dynamic memory allocation
- Closed Table
  - Use Probing for empty slots (while they exist)
  - + Fixed amount of memory (good on consoles!)
  - - Upper limit

Hashing

Hash value: $h = H(k)$

Table index: $i = h \bmod N$

Hash function H:

- is crucial for distributing the keys well
- has to be quick
- has to be deterministic

Probing

- Different ways
  - Linear
  - Quadradic (to avoid clustering of keys)

Strings

- Tricky!
  - Not an atomic type
  - Often require localization
  - Often used as internal names

- How you handle them → Major ramifications

String Class

- Dangerous
  - Expensive copying if not careful
  - May rely on dynamic memory allocation
  - May or may not be optimized (find out)

- Justifiable when you have special kinds of strings, e.g. Filepaths

Unique Identifiers

- Objects and Assets need identification
  - Often done with Strings
  - GUIDs are too cryptic
- **BUT** we need fast comparisons!
  - Compare hash-codes (integers) instead
  - Hashed strings called „Names“ in UE
- Replace Strings with IDs
  - At Preprocessing stage
  - Definitely not every time you use the string

Localization

- What might you need to localize?
  - Visible strings
  - Textures / Signs
  - Sounds
  - GUI layout
  - Ratings
  - Sensored contents
  - Cultural aspects
- Requires an internationalized software development approach

Unicode

- Itself not a mapping to bit patterns in memory!
  - Instead maps letters to code points
- However UTF-16 encodes code points in two bytes for each letter (note endedness!)
  - Waste of space a lot of the time
- UTF-8 brilliantly encodes code points in both 1 and 2 bytes, with 1 byte characters sharing the encoding scheme with ANSI

String Databases

- For localization, visible strings are kept in a table with multiple translations
  - Simple (e.g. CSV text files)
  - Complex (e.g. Oracle databases)
- Looked up via IDs
- Sometimes distributed
  - Work distributed across various countries

Engine Configuration

- Location
  - Text configuration file (INI, XML, ...)
  - Compressed binary
  - Windows registry
  - Command line options
  - Environment variables
  - Online user profile
- Per-User locations
  - Disk slots/folders, „Application Data“, „HKEY-CURRENT_USER“, $HOME

Example Configurations

- Quake CVARs
  - Global linked list of strings or floats
  - Manipulated in console, dumped to config.cfg
- Ogre 3D
  - Collection of INI files e.g. ogre.cfg, resources.cfg
  - Manipulated through Ogre::ConfigFile
- Uncharted
  - INI style text file manipulated through flexible menu
  - Also Scheme data definitions for complex data