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)