Admin

- Today's topics
  - Hashing
- Reading
  - Ch 11
- Terman café today after class
  - Last chance!

Lecture #24

Compare Map implementations

<table>
<thead>
<tr>
<th>Vector</th>
<th>Sorted Vector</th>
<th>BST</th>
</tr>
</thead>
<tbody>
<tr>
<td>getValue</td>
<td>O(N)</td>
<td>O(logN)</td>
</tr>
<tr>
<td>add</td>
<td>O(N)</td>
<td>O(N)</td>
</tr>
</tbody>
</table>

- Space used, code complexity?
  - Vector is just key+value, no overhead
    - Fairly simple to implement (hardest part is binary search)
  - BST adds 8 bytes of pointers to each entry
    - Pointers, dynamic memory, recursion
    - Plus code/space for tree-balancing to guarantee O(logN)

A completely different tactic

- How do you look up word in dictionary?
  - Linear search?
  - Binary search?
  - A-Z tabs...?
- Hashable idea
  - Table maintains B different "buckets"
  - Buckets are numbered 0 to B-1
  - Hash function maps a key to value in range 0 to B-1
    - add/getValue hash key to determine which bucket it belongs in
      - only search/modify this one bucket

Hash functions

- Hash function maps key to a number
  - Result constrained to some range
  - Result is stable
    - Same key in -> same number out
- Goal to distribute keys over range
  - Bad if many keys map to 17 and none to 22
- Possible hash functions
  - First letter?
  - Length of word?
  - Sum of ASCII values for letters?
Hash collisions

- What happens if several keys hash to same code?
  - Called a collision
- Good hash function tries to avoid, but no guarantee
- One strategy is "chaining"
  - Keys within bucket are stored in a linked list
  - Each list expected to be small, so easy to traverse

Hashtable performance

- Time required for getValue & add?
  - Hash to bucket, search chain = $O(N/B)$
  - Use basically same steps for both operations
- How to determine number of buckets?
  - If same as num entries, operations are $O(1)$!
- How to store each bucket?
  - Array vs linked list vs vector?
  - Should entries be sorted?
- Rehashing
  - Track "load factor" ($n_{entries}/n_{buckets}$), when too high, resize table, and rehash everything

Compare Map implementations

<table>
<thead>
<tr>
<th></th>
<th>Vector</th>
<th>Sorted Vector</th>
<th>BST</th>
<th>Hash</th>
</tr>
</thead>
<tbody>
<tr>
<td>getValue</td>
<td>$O(N)$</td>
<td>$O(\log N)$</td>
<td>$O(\log N)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>add</td>
<td>$O(N)$</td>
<td>$O(N)$</td>
<td>$O(\log N)$</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>

Space used, code complexity?

- Vector is just key+value, no overhead
  - Fairly simple to implement (hardest part is binary search)
- BST adds 8 bytes of pointers to each entry
  - Pointers, dynamic memory, recursion
  - Plus code/pace for tree-balancing to guarantee $O(\log N)$
- Hash uses 4 bytes per entry + 4 bytes per bucket, total 8 bytes per entry
  - Does hash have degenerate cases?

Hashing generic types

- Map requires key to be string type
- What about a 2-type template?
  ```cpp
template <typename KeyType, typename ValType>
class Map {
  public:
    Map();
    void add(KeyType k, ValType v);
  ...}
};
```

Client usage:
```
Map<string, int> s;
Map<int, Vector<string>> t;
```

What would this require from client?
Implementing Set

◊ Last ADT in the 106 class library
  • Goal: fast search (contains), fast update (add/remove), hopefully efficient high-level ops (browse in order will help)

◊ What strategies might work?
  • Vector/array (sorted?)
  • Linked list
  • Trees
  • Hashing

◊ Our set build on BST template
  • BST is balanced binary search tree abstraction

Class library

◊ Last ADT in the 106 class library

◊ What strategies might work?
  • Vector/array (sorted?)
  • Linked list
  • Trees
  • Hashing

◊ Goals
  • Fast search, fast update (add/remove)