CSC 220 Data Structures

Maps and Hash Tables I - Intro

Parkland College Fall 2016

20161019
Maps

- key/value pairs
- keys
  - strict weak ordering (irreflexive & transitive)
  - unique
- values
  - anything
- Python dictionaries, Perl hashes, std::map, Interface Map
Applications

• Everywhere
  • two most important STL containers are `std::vector` and `std::map`
  • simple “database”
    • indexing by word instead of number
• `word_frequency.py`
Five Essential Behaviors API

• get a value given a key
  • value = myMap[‘key’]

• set a key’s value
  • myMap[‘key’] = value

• delete a key/value pair
  • del myMap[‘key’]

• get the size of a map
  • len(myMap)

• iterate over a map
  • for k,v in myMap
Additional Useful API

• checking if key in map
• setting default values
• popping by key
• clearing all items
• getting all the keys
• getting all the values
• getting all the items as key, value pairs
• checking equality/inequality
• etc.
Class Hierarchy

```
MutableMapping
   
MapBase
   
UnsortedTableMap  |  SortedTableMap  |  HashMapBase  |  TreeMap
   
   |  
   |  
ChainHashMap  |  ProbeHashMap
   ```
MutableMapping

- defined in Python `collections` module
- abstract base class
- *template method design pattern*
  - declares but does not define 5 essential behaviors
  - other methods call those 5 undefined behaviors
  - concrete child classes define the 5 essential behaviors, automatically get a fully featured API!
MapBase

- `map_base.py`
- also abstract
- derives from `MutableMapping`
- defines `_Item` for use in concrete classes
Map on Python List

- `unsorted_table_map.py`
- `store_Items in a Python list`
- define 5 essential behaviors
- Pro: easy
- Con: getting/setting is O(n)
  - When we see `[ ]` in code, we expect O(1)
Hashing and Hash Tables

• *hash function* transforms arbitrary key to index
  • get back O(1) get/set at cost of “hashing”

• a good hash function minimizes *collisions*
  • (when different keys that hash to the same index)

• a good hash table handles collisions
  • many possible solutions
  • *bucket array*
Hash Functions

1. hash code
   • converts key to integer

2. compression function
   • fits hashed key to bucket array indices
   • Separate because compression depends on actual map implementation
Hash Codes

- want to
  - minimize collisions
  - distribute hashes throughout bucket array
- techniques
  - reinterpret bits
  - polynomial hashing
  - cyclic shifts
  - Python `hash()`

*Never roll your own for security/encryption apps!*
Compression Functions

- **division method**
  - $\text{hashed\_key} \mod \text{len}\left(\text{bucket\_array}\right)$

- **Multiply, Add, and Divide (MAD)**
  - $\left(\left(\text{a}\times\text{hashed\_key}+\text{b}\right)\mod\text{p}\right)\mod\text{len}\left(\text{bucket\_array}\right)$
  - $\text{p}$ is prime
  - $\text{a, b}$ random ints in $[0, \text{p}-1]$
  - $\text{a} \neq 0$