Lecture 28: Maps

Today

- Reading
  - JS Chapter 15.1-15.7
- Objectives
  - Finish deadlocks
  - Maps, HashMaps
  - Hash functions
  - (Collisions)

+ Deadlock

- A deadlock occurs when there are threads $T_1, \ldots, T_n$ such that:
  - T_i is waiting for a resource held by T_{i+1} for i=1,..,n-1
  - T_n is waiting for a resource held by T_1
- A cycle of waiting
  - Can formalize as a graph of dependencies with cycles bad
- Deadlock avoidance in programming amounts to techniques to ensure a cycle can never arise

+ Solving Deadlocks

Options for avoiding deadlocks:

- No thread ever holds more than one lock
- Define globally agreed upon order for locks
  - Dining Philosopher's Problem (Dijkstra)
  - Every bank account has unique number – acquire lock for lower ordered bank accounts first
- Sometimes can't guarantee no deadlock

A Last Example

From the Java standard library

```
class StringBuffer { // a mutable String
  private int count;
  private char[] value;
  …
  synchronized append(StringBuffer sb) {
    int len = sb.length();
    if(this.count + len > this.value.length)
      this.expand(…);
    sb.getChars(0,len,this.value,this.count);
  }
  synchronized getChars(int x, int, y,
                        char[] a, int z) {
    "copy this.value[x..y] into a starting at z"
  }
}
```

Two problems

- Problem #1: Lock for `sb` is not held between calls to `sb.length` and `sb.getChars`
  - The variable sb could get longer
  - Would cause append to throw an `ArrayBoundsException`
- Problem #2: Deadlock potential if two threads try to append in opposite directions
- Not easy to fix both problems:
  - Do not want unique ids on every `StringBuffer`
  - Do not want one lock for all `StringBuffer` objects
- Actual Java library fixed neither (left code as is; changed javadoc)
  - Up to clients to avoid such situations with own protocols

Concurrency summary

- Correctly and efficiently controlling access to shared resources
  - Benefits
  - Race conditions: bad interleavings, data races
  - Critical sections too small
  - Deadlocks
- Requires synchronization
  - Locks for mutual exclusion
- Guidelines for correct use help avoid common pitfalls
- Getting shared memory correct is hard!

Class to Date

- Data structures
- Complexity
- Sorting
- Parallelism and Concurrency
- Additional data structures (maps, hashmaps)
- C++
- Graph Algorithms

Map<K,V>

- A collection of associations between a key and an associated value
  - e.g. name and phone number
  - e.g. word and definition
  - (not Bailey's Association)
- Many possible implementations
- Also called "dictionary" since provides good implementation of a lookup table

The Java Map Interface

```
public interface Map<K,V> {
	public boolean containsKey(Object k);
	public boolean containsValue(Object v);
	public V get(Object k);
	public V put(K k, V v);
	public V remove(K k);
	public void putAll(Map<K,V> other);
	public Set<K> keySet();
	public Collection<V> values();
	public Set<Map.Entry<K,V>> entrySet();
	public boolean equals(Object o);
	public int hashCode();
	// Also has size(), clear(), isEmpty()
}
```

Map.Entry is the equivalent of Bailey's Association

Map Implementations

Data Structure	Search	Insert	Delete	Space
Linked List	O(n)	O(1)	O(n)	O(n)
Sorted Array	$O(\log_2 n)$	O(n)	O(n)	O(n)
Balanced BST	$O(\log_2 n)$	$O(\log_2 n)$	$O(\log_2 n)$	O(n)
array[KeyRange]	O(1)	O(1)	O(1)	O(KeyRange)

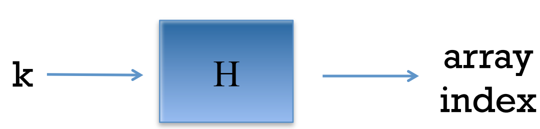
n = number of elements in map

Sorted array and balanced BST require comparable keys

Last implementation requires keys that can be used as array subscripts

Hash Table

- What are some of the drawbacks of using keys as subscripts?
  - Restricts types of keys
  - Keys often too sparse
  - e.g. use SSN for table of students
- Instead use a function that maps from keys to subscripts (control the range)



Hash Functions

- A function from the set of keys to array subscripts

$$H : K \longrightarrow \text{Subscripts}$$

- Ideally:
  - H(k) can be computed quickly
  - H is a one-to-one function, i.e. if $H(k_1) = H(k_2)$ then $k_1 = k_2$
  - Called a perfect hashing function (hard to find)
- `hashCode` function is built-in to Java classes – hashes the object and returns an integer
- Require that if `k1.equals(k2)` then H(k1) == H(k2)
- If override `equals` method, must override `hashCode`

Examples of Hash Functions

- For `String` Java uses:

  $H(s) = s[0]*31^{(n-1)} + s[1]*31^{(n-2)} + \ldots + s[n-2]*31 + s[n-1]$

- For integers, we could use:

  H(x) = x mod N where N is the size of the array

- For social security numbers, we could use (not the best):

  H(ssn) = (last 4 digits) mod N

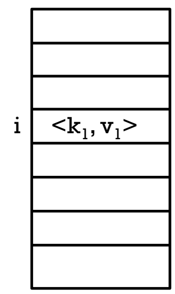
- Bad hash function for strings:

  H(s) = (length of s) mod N

  H(s) = sum of characters in s

+ Hashing Collisions

- A collision occurs when k1 ≠ k2 but H(k1) = H(k2)
- Two solutions:
  - Open addressing: rehash as needed to find empty slot
  - External chaining: keep all entries that hash to same subscript in list



If k_2 maps to i as well, where do we put the entry <k_2, v_2>?

+ Primary and secondary clustering

- Primary clustering
  - When an open addressing scheme tends to create long stretches of filled slots
  - "Two values that hash to same slot continue to compete during rehashing"
- Secondary clustering
  - Two values that hash to different slots eventually compete during rehashing
- Pertain only to open addressing schemes

+
Open addressing (Probing)

- Linear probing
  - Use (currentSlot + offset) % (array.length)
  - offeset should be relatively prime to array length to ensure we search every array slot (use array whose length is prime)
  - Easy to implement but prone to primary clustering
- Quadratic probing
  - Use (currentSlot + j^2) % (array.length) on j^{th} rehash
  - Helps with secondary clustering but not primary
  - Can result in case where we don't search every slot
    - e.g. array.length = 5 and H(k) = 1

+
Open addressing (Probing)

- Double hashing
  - Use second hash function to determine the offset
  - e.g. Suppose we use $H_1(x)$ = x mod N for the array subscript and $H_2(x)$ = x mod (N-2) + 1 for offset for N=5
  - Helps with primary and secondary collisions

Collisions!	Different offsets	Next subscripts to try
$H_1(1) = 1$	$H_2(1) = 2$	$H(1) = 1 + 2$
$H_1(6) = 1$	$H_2(6) = 1$	$H(6) = 1 + 1$
$H_1(11) = 1$	$H_2(11) = 3$	$H(11) = 1 + 3$

External Chaining

- Each slot in table (array) holds unlimited number of entries
  - Each slot contains a list data structure (e.g. array, linked list)
  - Each list should be short (balanced BST would be overkill)
  - Deleting is simple
  - No elements hashed can be greater than size of array
  - Avoids secondary clustering