CS203 Programming with Data Structures
Hashing

Chengyu Sun
California State University, Los Angeles

Search Complexity

Data structure	Complexity	N = 1,000,000
Unordered list	O(N)	500,000
Balanced BST	O(logN)	~20
??	**O(1)**	

Hashing

- Given function **H** and a table **HT**
- For an object `o`, the address of the object in the table is determined by `H(o)`

Hashing Example

Hash table (HT): `String table[];`

Hash function (H):

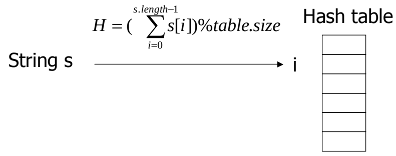
```
int hash( String s )
{
    int value = 0;
    for( int i=0 ; i < s.length ; ++i )
        value += s.charAt(i);
    return value%table.length;
}
```

Two Key Issues in Hashing

- Hash function
- Collision resolution

Hash Function 1

$$H = \left(\sum_{i=0}^{s.length-1} s[i] \right) \% table.size$$



Hash Function 2

$$H = \left(\sum_{i=0}^{s.length-1} s[s.length - i - 1] \bullet 37^{i} \right) \% table.size$$

```
int hash( String s )
{
    int value = 0;
    for( int i=0 ; i < s.length ; ++i )
        value = 37 * value + s.charAt(i);
    return value%table.length;
}
```

Other Hash Functions

- Mid-square
- Folding
- `hashCode()` in `Object` class
  - Return the memory address of the object by default
  - Overridden in many classes such as `String` and `Integer` so that if `o1.equals(o2)`, `o1.hashCode()==o2.hashCode()`

Characteristics of Good Hash Functions

- Efficient to compute
- Distribute the objects evenly in the table (minimize collision)

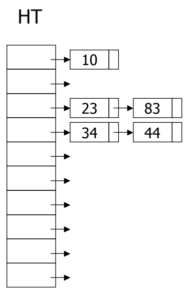
Collision Resolution

- Chaining
- Open addressing
  - Linear probing
  - Quadratic probing

Chaining

H(o) = o%10

10, 44, 34, 23, 83



About Chaining

- Easy to implement
- Require dynamic memory allocation
- Require implementation of another data structure

Load factor $\lambda = \frac{\text{\# of elements in the table}}{\text{table size}}$

Search complexity: $1 + \lambda/2$

Linear Probing

- If position `i` is taken, try `i+1`, `i+2`, `i+3` … until an available position is found

Empty table

After inserting 83,23,34,44,10

??

About Linear Probing

- Primary clustering
- How do we *search* elements??
- How do we *delete* elements??
- Complexities

Successful search: $(1+1/(1-\lambda))/2$

Unsuccessful search: $(1+1/(1-\lambda)^2)/2$

Quadratic Probing

- If position `i` is taken, try `i+1*1`, `i+2*2`, `i+3*3` … until an available position is found

Empty table

After inserting 83,23,34,44,10

??

About Quadratic Probing

- Theorem: if quadratic probing is used, and the table size is prime, then a new element can always be inserted if the table is at least half empty.

`HashTable` Class

- void insert(Object o)
- Object remove(Object o)
- boolean contains(Object o)
- int size()
- void clear()

`HashMap` Class

- void insert(Object key, Object value)
- Object remove(Object key)
- Object get(Object key)
- int size()
- void clear()