

CMPT 280

Tutorial: Hash Tables

Mark G. Eramian

University of Saskatchewan

Collision Resolution using Open Addressing

- Hash array size: N
- Hash function: $h(k)$
- Probe increment: $p(j)$
- Location of j -th alternative array offset: $(h(k) + p(j)) \bmod N$

Collision Resolution using Open Addressing

Linear Probing

- Hash function: item value mod 10
- j -th probe increment: $p(j) = j$
- What does the hash array look like after insertion of 59, 42, 92, 102, 29, 39, 62.

0	1	2	3	4	5	6	7	8	9

Collision Resolution using Open Addressing

Quadratic Probing

- Hash function: item value mod 10
- j -th probe increment: $p(j) = (-1)^{j-1} \left(\frac{j+1}{2}\right)^2$
- What does the hash array look like after insertion of 59, 42, 92, 102, 29, 39, 62.

0	1	2	3	4	5	6	7	8	9

Collision Resolution using Open Addressing

Double Hashing

- Hash function: item value mod 10
- j -th probe increment: $p(j) = (k \bmod 7 + 1)j$
- What does the hash array look like after insertion of 59, 42, 92, 102, 29, 39, 62.

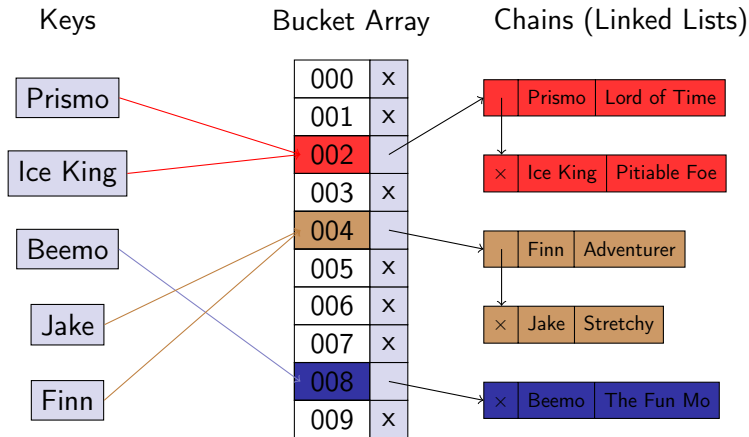
0	1	2	3	4	5	6	7	8	9

Collision Resolution using Open Addressing

- More examples of linear and quadratic probing and double hashing on chalkboard as needed...

Chained Hash Tables

Conceptual View



Hash function maps keys to bucket array offset.

Linked chain for each bucket contains full records for keys that map to the bucket.

Chained Hash Tables in lib280

In Assignment 4 you will use lib280's KeyedChainedHashTable280 class.

```
1 public class KeyedChainedHashTable280<K extends Comparable<? super K>,  
2     I extends Keyed280<K> >  
3     extends HashTable280<I>  
4     implements KeyedDict280<K, I>
```

- Two type paramters: K and I.
- Extends abstract class HashTable280.
- Implements KeyedDict280<K,I>.

Keyed Dictionaries

- Recall: Dictionaries allow adding, querying, and deleting of elements.
- `OrderedSimpleTree280`, is a dictionary, but not a keyed dictionary. If you want to query or delete from a non-keyed dictionary, you need to already have the entire item that you are querying or deleting.
- A keyed dictionary is one in which every item has a unique key, and items can be looked up in the dictionary using **only** their key to obtain the entire item with that key.

KeyedChainedHashTable280 is a keyed dictionary.

- Since KeyedChainedHashTable280 is a keyed dictionary (it implements the KeyedDict280 interface) supports operations like:
 - `obtain(k)` - get the entire item that has key `k`.
 - `delete(k)` - delete the item with key `k` from the hash table.
- But it also still supports the non-keyed dictionary operations like:
 - `obtain(i)` - get the item that matches item `i`
 - `delete(i)` - delete the item that matches item `i`
 - `insert(i)` - insert the item `i`(this is because it inherits Dict280 via HashTable280 – HashTable280 enforces a requirement that all hash tables be at least non-keyed dictionaries).

Type parameters for KeyedChainedHashTable280<K,I>

```
1 public class KeyedChainedHashTable280<K extends Comparable<? super K>,  
2     I extends Keyed280<K> >  
3     extends HashTable280<I>  
4     implements KeyedDict280<K, I>
```

- The type parameter K is the type of the keys of the items to be stored.
- Since instances of type K must be comparable other instances of type K, it is required that type K be one that implements the Comparable interface.

Type parameters for KeyedChainedHashTable280<K,I>

```
1 public class KeyedChainedHashTable280<K extends Comparable<? super K>,  
2     I extends Keyed280<K> >  
3     extends HashTable280<I>  
4     implements KeyedDict280<K, I>
```

- The type parameter I is the type of the items that may be stored in the hash table.
- Note that I must extend Keyed280<K>. This forces the type I to implement a method called key() which returns the item's key, which must be of type K.
- In summary: the hash table stores items of type I whose keys are of type K.

Internal Data Structure for KeyedChainedHashTable280<K,I>

```
1 // Array to store linked lists for separate chaining.  
2 protected LinkedList280<I>[] hashArray;
```

- The hash table implementation is an array of linked lists (the array of buckets on slide 3) containing items of type I.
- A function called `hashPos`, defined in `HashTable280` is used to take an item and map it to one of the offsets of the linked-list array.
- The `hashPos` function calls the `hashCode` method of the given item (which, if not overridden, is found in `Object`) and converts the integer result to a number between 0 and `hashArray.length-1` using the modulus operator.

Other features of `KeyedChainedHashTable280<K,I>`

- The internal array starts at a default length, and increases in size if the hash tables *load factor* gets too large. This is why you do not have to specify an array size in the hash table's constructor.
- The load factor is the number of items in the hash table divided by the length of the array.
- The hash table also has an internal cursor that can be used to iterate over all of the items in the table. It supports the usual cursor methods like `goFirst()`, `itemExists()`, `goForth()`, etc.
- You can look at the code for `KeyedChainedHashTable280<K,I>` if you want to see how all of this works.