

CSI33 DATA STRUCTURES

Department of Mathematics and Computer Science
Bronx Community College

November 29, 2017



OUTLINE

- 1 CHAPTER 13: HEAPS, BALANCED TREES AND HASH TABLES
 - Hash Tables



OUTLINE

- 1 CHAPTER 13: HEAPS, BALANCES TREES AND HASH TABLES
 - Hash Tables



PYTHON DICTIONARIES

Various names are given to the abstract data type we know as a **dictionary** in Python:

- Hash (The languages Perl and Ruby use this terminology; implementation is a hash table).
- Map (Microsoft Foundation Classes C/C++ Library; because it maps keys to values).
- Dictionary (Python, Smalltalk; lets you "look up" a value for a key).
- Association List (LISP—everything in LISP is a list, but this type is implemented by hash table).
- Associative Array (This is the technical name for such a structure because it looks like an array whose 'indexes' in square brackets are key values).



PYTHON DICTIONARIES

In Python, a **dictionary** associates a **value** (item of data) with a unique **key** (to identify and access the data). The implementation strategy is the same in any language that uses associative arrays. Representing the relation between key and value is a **hash table**.



PYTHON DICTIONARIES

The Python implementation uses a **hash table** since it has the most efficient performance for insertion, deletion, and lookup operations. The running times of all these operations are better than any we have seen so far. For a hash table, these are all $\Theta(1)$, taking a **constant** time to run. (If the table gets larger, the running times do not increase.) This is done by calculating the **address** of any item, stored in an array, from its **key** value. The function used to calculate this address is called a **hash function**.



HASHING FUNCTIONS

- For keys which are already numeric (integers), they can be divided by the size of the array. The remainder becomes the index into the fixed array.
- Text keys must be transformed into integers. The characters' ASCII codes can be added together, for example. Then divide and take the remainder.
- Functions like this scatter the items through the array. If the function spreads the items 'randomly' over the array, there are few problems until the array starts to fill up (when the **load factor**—the filled fraction of the array—becomes one-half or greater).
- The problems begin when a key having some value get transformed into the same address as an existing key. This is called a **collision**. Where can the value for the new key go, so it can be found later?



COLLISION RESOLUTION

There are different strategies for resolving collisions.

- Open addressing–Linear Probe
- Open addressing–Quadratic Probe
- Double Hashing
- Separate Chaining



COLLISION RESOLUTION

OPEN ADDRESSING—LINEAR PROBE

- If a hash function produces a slot address that is already in use, a linear function is used to calculate an alternate location for the new key's data. This is repeated until a free slot is found.
- When the new key is used to access the data, the original address is inspected. If the data is not found, the linear function is used to calculate the next likely location for the data. This is repeated until the data is found.
- This policy can lead to a high collision rate as items cluster around a few locations.



COLLISION RESOLUTION

OPEN ADDRESSING—QUADRATIC PROBE

- If a hash function produces a slot address that is already in use, a quadratic function is used to calculate an alternate location for the new key's data. This is repeated until a free slot is found.
- When the new key is used to access the data, the original address is inspected. If the data is not found, the quadratic function is used to calculate the next likely location for the data. This is repeated until the data is found.
- This policy can lead to fewer collisions than a linear probe policy.



COLLISION RESOLUTION

DOUBLE HASHING

- If a hash function produces a slot address that is already in use, an alternate hashing function is used to calculate an alternate location for the new key's data. This is repeated until a free slot is found.
- When the new key is used to access the data, the original address is inspected. If the data is not found, the alternate hashing function is used to calculate the next likely location for the data. This is repeated until the data is found.
- This policy can lead to fewer collisions than a quadratic probe policy.



COLLISION RESOLUTION

SEPARATE CHAINING

- If a hash function produces a slot address that is already in use, a linked list is begun at that slot which contains all the data for colliding keys at that slot. Subsequent keys that hash to that same slot are appended to the linked list.
- When the new key is used to access the data, the original address is inspected. If there is a linked list at that slot, the list is searched for that key's data.
- This policy still leads to $\Theta(1)$ performance as long as the load factor for the table is not too high.

