Implementing dictionaries

Madhavan Mukund

https://www.cmi.ac.in/~madhavan

Programming, Data Structures and Algorithms using Python

Week 3

Dictionary

- An array/list allows access through positional indices

Dictionary

- An array/list allows access through positional indices
- A dictionary allows access through arbitrary keys
  - A collection of key-value pairs
  - Random access — access time is the same for all keys

Dictionary

- An array/list allows access through positional indices
- A dictionary allows access through arbitrary keys
  - A collection of key-value pairs
  - Random access — access time is the same for all keys
- How is a dictionary implemented?

Implementing a dictionary

- The underlying storage is an array
  - Given an offset `i`, find `A[i]` in constant time

Implementing a dictionary

- The underlying storage is an array
  - Given an offset `i`, find `A[i]` in constant time
- Keys have to be mapped to $\{0, 1, \ldots, n-1\}$
  - Given an key `k`, convert it to an offset `i`

Implementing a dictionary

- The underlying storage is an array
  - Given an offset `i`, find `A[i]` in constant time
- Keys have to be mapped to $\{0, 1, \ldots, n-1\}$
  - Given an key `k`, convert it to an offset `i`
- Hash function
  - $h : S \to X$ maps a set of values S to a small range of integers $X = \{0, 1, \ldots, n-1\}$

Implementing a dictionary

- The underlying storage is an array
  - Given an offset `i`, find `A[i]` in constant time
- Keys have to be mapped to $\{0, 1, \ldots, n-1\}$
  - Given an key `k`, convert it to an offset `i`
- Hash function
  - $h : S \rightarrow X$ maps a set of values S to a small range of integers $X = \{0, 1, \ldots, n-1\}$
  - Typically $|X| \ll |S|$, so there will be collisions, $h(s) = h(s'), s \neq s'$
  - A good hash function will minimize collisions

Implementing a dictionary

- The underlying storage is an array
  - Given an offset `i`, find `A[i]` in constant time
- Keys have to be mapped to $\{0, 1, \ldots, n-1\}$
  - Given an key `k`, convert it to an offset `i`
- Hash function
  - $h : S \rightarrow X$ maps a set of values S to a small range of integers $X = \{0, 1, \ldots, n-1\}$
  - Typically $|X| \ll |S|$, so there will be collisions, $h(s) = h(s'), s \neq s'$
  - A good hash function will minimize collisions
  - SHA-256 is an industry standard hashing function whose range is 256 bits
    - Use to hash large files — avoid uploading duplicates to cloud storage

Hash table

- An array A of size n combined with a hash function h

Hash table

- An array A of size n combined with a hash function h
- h maps keys to $\{0, 1, \ldots, n-1\}$

Hash table

- An array `A` of size n combined with a hash function h
- h maps keys to $\{0, 1, \ldots, n-1\}$
- Ideally, when we create an entry for key `k`, `A[h(k)]` will be unused

Hash table

- An array `A` of size n combined with a hash function h
- h maps keys to $\{0, 1, \ldots, n-1\}$
- Ideally, when we create an entry for key `k`, `A[h(k)]` will be unused
  - What if there is already a value at that location?
- Dealing with collisions

Hash table

- An array `A` of size n combined with a hash function h
- h maps keys to $\{0, 1, \ldots, n-1\}$
- Ideally, when we create an entry for key `k`, `A[h(k)]` will be unused
  - What if there is already a value at that location?
- Dealing with collisions
  - Open addressing (closed hashing)
    - Probe a sequence of alternate slots in the same array

Hash table

- An array `A` of size n combined with a hash function h
- h maps keys to $\{0, 1, \ldots, n-1\}$
- Ideally, when we create an entry for key `k`, `A[h(k)]` will be unused
  - What if there is already a value at that location?
- Dealing with collisions
  - Open addressing (closed hashing)
    - Probe a sequence of alternate slots in the same array
  - Open hashing
    - Each slot in the array points to a list of values
    - Insert into the list for the given slot

Hash table

- An array `A` of size n combined with a hash function h
- h maps keys to $\{0, 1, \ldots, n-1\}$
- Ideally, when we create an entry for key `k`, `A[h(k)]` will be unused
  - What if there is already a value at that location?
- Dealing with collisions
  - Open addressing (closed hashing)
    - Probe a sequence of alternate slots in the same array
  - Open hashing
    - Each slot in the array points to a list of values
    - Insert into the list for the given slot
- Dictionary keys in Python must be immutable
  - If value changes, hash also changes!

Summary

- A dictionary is implemented as a hash table
  - An array plus a hash function
- Creating a good hash function is important (and hard!)
- Need a strategy to deal with collisions
  - Open addressing/closed hashing — probe for free space in the array
  - Open hashing — each slot in the hash table points to a list of key-value pairs
  - Many heuristics/optimizations possible for dea