CSI33 Data Structures

Sharon Persinger

Fall 2019

Day 26 December 4

Topics

Hash Table

Hashing

Managing Collisions

- chaining
- open addressing

Hash Table

A hash table or associative array is a data structure for storing (key, value) pairs.

Python's dictionary data structure is a hash table.

Underlying a hash table is an array that stores each (key, value) pair at a specific index.

We want to find the index for a key in $\Theta(1)$ time.

Hashing Function

A hashing function(or hash function) is a function that computes the index in the hash table array for a key.

It is difficult to write one hashing function that works with a large collection of possible keys, and easier to develop a good hashing function when you know ahead of time the number and characteristics of the possible keys.

Simple example of a Hash Table with set of keys the single capital letters ‘A’ through ‘Z’

HashLetter.py

This hashing function actually stores the values in the same order as the keys.

The function converts each key to a number and then does a mathematical computation with the number to get an index.

Hashing Functions for more complicated keys

Keys can be data of different types – numerical types, string data, even complex objects.

How do we define a hashing function that works for a larger set of keys?

Usually a hashing function is created based on knowledge of the characteristics of the expected set of keys. Each key is converted to a number, and then some mathematical function is applied to the number to create an index.

Suppose the keys are strings of characters. One way to create a hashing function for these keys is to convert each character to a number using its ASCII value, do some computations with the ASCII values to get another integer numbers, and then convert that integer number into an index in the array.

Example of simple hash function for strings

```
def hash(w, array_size):

#pre: w is an ASCII string

#post: returns a value between 0 and
array_size – 1

    v = 0

    for i in range(min(len(w), 4)):
        v = 128*v + ord(w[i])

    return v%array_size
```

Before the modulus, this function produces a different number v for each different 4 character string.

How many ASCII strings of length 4? There are 128 different single characters, so there are 128^4= 268,435,456 ASCII strings of length 4.

We need an array of that size to be able to have a different index for each possible 4 character string.

If we use a smaller array, we will possibly have collisions - two different keys that give the same hash function value

Dealing with collisions

Chaining:

- Store multiple (key, value) pairs in a list at the same index
- Easy to do on Python
- Worst case – all key value pairs are stored at the same index location
- Must store both key and value at the index

Open addressing:

- Create some second function for finding a new index to use for the (key, value) pair if the index created by the hash function is already used for another pair
- Must store both key and value at the index

Details next class