CSI33 Data Structures

Sharon Persinger

Fall 2019

Day 5

Finishing the chapter on container classes

Deck class

- A Deck object is a collection of cards – another container class.
- Methods: constructor, size, deal, shuffle
- shuffle method is interesting.

Storing and retrieving data

- Computer memory is a sequence of storage locations.
- Each storage location has an address.
- A data item is stored in a storage location or in several neighboring storage locations.
- To retrieve data, we need to know its address.

How can we store a collection of data so we can operate on it efficiently?

- A data structure called an array is used to store a collection of data all of the same data type.
- An array is a collection of neighboring or contiguous memory locations. Array elements are usually retrieved by indexing as with Python's lists: A[0], A[1], A[2], and so on. It is common to start the indexing at 0.
- To store a collection of 100 4-byte integers, allocate an array A of 400 bytes. Suppose the address of the beginning of this array is 1024. The initial element A[0] of the array takes up bytes 1024, 1025, 1026, and 1027, and nest element starts at address 1028 = 1024 + 4. Generally, the ith element is 1024 + 4*i.

Good and Bad about Arrays

- Good: Arrays allow for efficient random access. The address of A[i] is address of A[0] + i*(size of A's data type). This takes constant time.
- Bad: To do this, the data elements must all be of the same size. Usually this is enforced by requiring that elements are all of the same type.
- Bad: The size of an array is determined when the array is allocated. Arrays are static. Programmers write classes for dynamic arrays that resize when needed.

Python lists are

- Heterogeneous – They can store objects of a mix of types.
- Dynamic - They can grow and shrink using the built-in methods.
- So Python lists have advantages over arrays.

How are Python lists implemented?

- A Python list is an array of references to Python objects. A reference is a memory address, and all references are the same length – 32 bits or 64 bits depending on the operating system.
- Suppose we have a 32-bit operating system and suppose A is a Python list that begins at memory address 1024. To find the list element A[0], go to memory locations 1024 through 1024+31 and get the reference (address) stored there. Look at that memory location and you will find A[0]. To find the list element A[i], get the reference that is stored in locations 1024 + 32*i through 1024 +32*i +31, and go to that memory location to find A[i].
- The memory for the array will be increased when needed. The underlying array of references has dynamic memory allocation.

Time analysis of list operations

- Python lists are dynamically allocated arrays of references
- how are Python list operations carried out?
  - retrieving an element based on its index Time?
  - changing the element at an index Time?
  - appending an element? Time?
    - uses dynamic memory allocation when needed
  - inserting a new element into the middle of the list? Time?
    - Lots of recopying here
  - deleting an element from the middle of a list? Time?
    - Recopying here too

Python dictionaries

- Python dictionaries are mappings or functions
- A list is a mapping of a certain type:
  - domain is set of indices {0, 1, 2, 3, ..., n}
- A dictionary is a collection of key- value pairs
- Do example suits = {'c':"Clubs", 'd':"Diamonds", "h":"Hearts"}
- Add the spades pair
- Look for values associated with keys- either indexing or get
- Look for entry not present
- Change some values
- List the keys, values, items
- Loop over keys
- in operator

Dictionary ADT

- dictionary is a function table with methods - a useful ADT, built-in in Python
- Create - returns an empty dictionary
- put(key, value) - post: value is associated with key in the function
- get(key): pre: there is X with (key, X) in dictionary, post: returns X
- delete(key): pre: There is X with)key, X) in dictionary, post (key, X) is removed from dictionary

How is the Python dictionary implemented?

- Ideas? Could you do it as a list of key-value pairs? That's what we do with a function table in a math class. How would the operations be implemented? Time analysis?
- Better implementation: Python uses a hashing function to create a hash table for the dictionary.
- A hashing function takes a key - a piece of data - and computes from it a number for the memory location to store the data.
- hash examples:
- can hash only immutable data. hash is based on the underlying representation of the object, can't change that
- Hash table is stored in an indexed list. (key, value) pairs are stored in the list. hash(key) determines the index.
- Actually hash(key)%length of list so we get a valid index. Good hash functions distribute the keys uniformly among the indices.
- As long as there are no collisions, retrieval is efficient. There are ways to deal with collisions.

How is the Python dictionary implemented?

- Look up a value for a key: Compute hash(key). If something is stored at the location, grab it. If nothing is there, report that the pair doesn't exist.
- Insert a (key, value) pair: Compute hash(key). Record the (key, value) pair at that location in the list.
- Change the pair (key, value1) to (key, value2).