Array-Based Data Structures, Searching, and Sorting

Salaar Liaqat

Data Sciences Institute, UofT

Outline

- Array Based Data Structures
  - Stack, queue, Python List
- Searching
  - Linear, Binary Search
- Sorting
  - Selection, Insertion Sort
- Hash map, hash table (Python dictionary), hash functions

Section 1

Array Based Data Structures

Abstract Data Types Versus Data Structure

- Some concepts are generally useful and transcend any programming language
- An **abstract data type** (ADT) defines some kind of data and operations that can be performed on it
  - Abstract because there is no mention of *how* data is stored or *how* the operations work
  - Concerned about "what"
- A **data structure** is a concrete method of storing data (and therefore its operations).
  - For instance, Python List is a data structure because it has a specific implementation.
- ADTs form a common vocabulary for computer scientists to discuss problems. It allows us to focus on the design and worry about implementation later.

Important ADTs

- Set
  - Data: a collection of unique elements
  - Operations: get size, insert a value (without introducing duplicates), remove a specified value, check membership
- List
  - Data: an ordered sequence of elements
  - Operations: access element by index, insert a value at a given index, remove a value at a given index

Important ADTs [1]

- Map
  - Data: a collection of key-value pairs, where each key is unique and associated with a single value
  - Operations: look-up a value for a given key, insert a new key-value pair, remove a key-value pair, update the value associated with a given key
- Iterable
  - Data: a collection of values (may or may not be unique)
  - Operations: iterate through the elements of the collection one at a time.

[1] From https://www.teach.cs.toronto.edu/~csc148h/winter/notes/

Relation between ADTs and Data Structures

- A Python `list` is not a ADT. But it is a natural implementation of the List ADT.
  - The designers of Python implemented `list` operations
- A single ADT can be implemented by many data structures
  - You could implement List ADT using a Python `dict`
  - We can store the list `["DS", 4, "Life"]` like this: `{0: "DS", 1: 4, 2: "Life"}`
- A data structure can implement many ADTs
  - Practice: how can you implement a set with a Python `list`?

Python Lists

- Each element has an address in memory. The addresses are ordered by index number and adjacent to each other.
- Run time for `append` method
  - A new address is created and placed at the end of the list
  - $O(1)$ time because it doesn't matter how long the list is
- Run time for `insert` method
  - The worst case occurs when you insert at the beginning of the list because each element in the list has to be shifted down by 1.
  - $O(n)$ time
- Run time for `delete` method
  - If you remove the first element, all other elements must be shifted up by one.
  - $O(n)$ time

Stack

- A stack contains zero or more items
  - Items are added at the top of the stack, called *pushing*
  - Items are removed from the top of the stack, called *popping*
- The first item added to the stack is the last item removed
  - We call this "first-in-last-out" (LIFO) behavior
- 2 minutes: is it faster to use the front or back of a Python list to implement a stack? What is the Big-O for stack operations under each choice?

Queue

- A queue contains zero or more items
  - Items are added at the rear of the queue, called *enqueue*
  - Items are removed from the front of the queue, called *dequeing*
- Items come out of the queue in the order they were added
  - We call this "first-in-first-out" (FIFO) behavior
- 2 minutes: is it faster to use the front or back of a Python list to implement a queue? What is the Big-O for stack operations under each choice?

Section 2

Searching

Motivating Example

- You want to develop a ML method to search through a video to figure out when an bike is stolen.
- You could start from the beginning of your video feed and run your ML method on each frame until you the bike is not in the frame.
  - This would take $O(n)$, probably a long time since you're using ML
- What if we started halfway through? If the bike was there, then break the remaining video in half and check again. If the bike wasn't there, then break the previous part of the video in half and check again.
  - This is *binary search*

Binary Versus Linear Search

- How many steps does binary searching through 100 numbers take? 10,000?
  - We can generalize this as $O(\log n)$
- What is the big-O of linear searching through 100 numbers? 10,000?
  - $O(n)$
- Notice binary search requires the list to be sorted in advance.
  - We implicitly assumed this in the bike theft example (time is "sorted")

Section 3

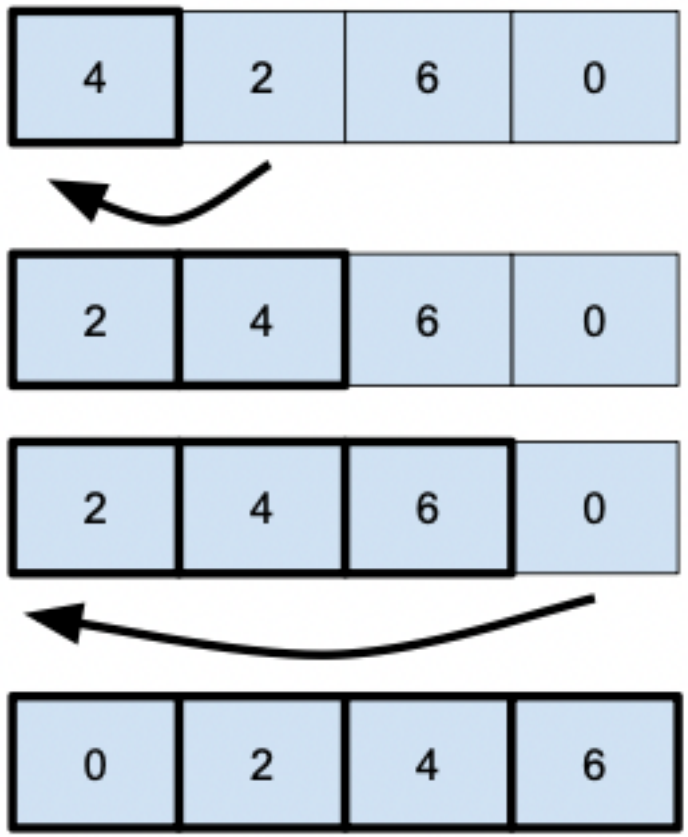
Sorting

Selection Sort

- Suppose you want to sort prices of all fruits at a supermarket from lowest to highest
- You go through the list, find the item with the lowest price then place it on top, then find the second lowest price and place it second, etc.
  - You will end up with a sorted list!
- To find the lowest price, you need to traverse the entire list. You must do this n times until there are no more items.
  - This takes $O(n^2)$ time

Insertion Sort

- Compare the current item to its predecessor. If the item is smaller than its predecessor, compare it to the items before. Move the greater items one position up to make space for the swapped item.
- You need to traverse the list once for each item in the list, so the Big-O is $O(n^2)$.



Section 4

Hash map, hash table (Python dictionary), hash functions

Motivating Example

- Recall from the first lecture that searching in a Python set took (basically) 0 seconds
  - How was this achieved?
  - Binary search only has $O(\log n)$ time, so there must be something else
- To achieve $O(1)$ time, we need something that immediately knows the where/what the item is.
  - This is the purpose of *hash functions*

Hash Functions

- A hash function is a function where you enter a string and it returns an integer
  - Python objects have hash

```
hash("DS 4 Life")
```

```
-8050649017317958 21
```

- There are two requirements for a hash function
  - It needs to be consistent. For instance, if you enter "UofT" and get "1827", then every time you enter "UofT" you should get "1827"
  - It maps different words to different numbers. Each string has a unique hash.

Using Hash Functions: Example

- Suppose you have a grocery store catalog with prices and barcodes. When you scan an item at checkout, you want it to instantly return the price.
- You can put each barcode into a hash function.
  - Let's say barcode "1234" *hashes* to "1" and "2" hashes to "9876"
  - We store the price of item "1234" at address "1". Store the price of item "4321" at address "2"
  - We say the price at "1" is the *hash value* of "1"
- If there are 8 items sold at the store, then the hash function will only return integers from 1 to 8
  - The size of the hash table is often referred to as its number of bins or slots.
  - Thus, the hash function depends on the array
- This implementation is called a *hash table*

Python's Hash Tables: `dict`

- You will likely never implement a hash table yourself, most languages have an implementation for has tables.
  - In Python, this is the `dict` class
- Dictionaries have keys and values (barcodes and prices)
- Dictionaries have really good performance. Search, insert, or delete item are all are $O(1)$ in the average case.
  - Average case assumes you have a "good" hash function that avoids *collisions*. You can read more about collisions in the textbooks.
  - The worst case of Python dictionaries for search, insert, and delete is $O(n)$.
- Recall Python dictionaries don't allow duplicate keys, that is because has hashes must be unique!

Python set

- Recall during the first lecture, we showcased that Python's set search was much faster than list search
- This is because Python's set implements a hash function to store its values

Section 5

Recommended Problems and References

Recommended Problems

- Bhargava: Chapter 5
  - 5.1 to 5.4
  - Read pages 79 to 86 on the use cases of hash functions
- Additional
  - Give examples of 2 situations to use a queue and 2 situations to use a stack
  - In Python, code a `stack` class with `is_empty`, `push`, and `pop` methods using the end of a Python list as the top of the stack. Bonus: Compare the run time of using the start of the list versus the end of the list as the top of the stack using the `timeit` library!
  - In Python, code a `binary_search` function.
  - In Python, code a `hash_table` that can hash 4 values.

References

- Bhargava, A. Y. (2016). *Grokking algorithms: An illustrated guide for programmers and other curious people.* Manning. Chapter 5.
- Cormen, T. H. (Ed.). (2009). *Introduction to algorithms* (3rd ed). MIT Press. Chapter 2, 10, 11.
- Horton, D., & Liu, D. (2023, November 19). *CSC148 Lecture Notes*. https://www.teach.cs.toronto.edu/~csc148h/winter/notes/