course_content

Data Structures and Algorithms

Arrays and Hash Tables

Week 3

Class Quiz!

Time:
5 min

[bit.ly/DSA1920Quiz3]

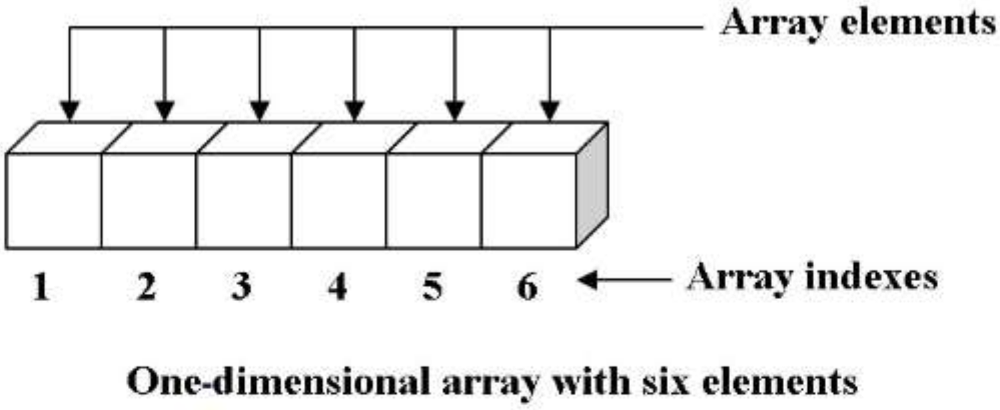
Bonus qn at the end to help you make up for other questions or missed quizzes..

Time:
5 min

Arrays

- Arrays are an **Abstract Data Type(ADT)**
- They hold a collection of data
- Each element can be accessed by its index
- The two primary operations in arrays are:
  - Store value in a specific index - *set(val,index)*
  - Retrieve value at a specific index - *get(index)*

Arrays - Visualisation



Source: [Lucas Magnum Medium]

Time:
5 min

Arrays - Exercise

- Groups of 3 - discuss the following
- What are the more advanced operations we can do on arrays starting from just set and get?
- Write the pseudocode for those using the basic operations
- This should be independent of the programming language that you use
- You can use operations on atomic data types

Time:
5 min

Arrays - Extra Operations

- **Traversing** - as we can access each element, we can traverse the array
- **Searching** - as we can traverse the array, we can look for a specific element
- **Sorting** - if we have defined a definition of >, we can traverse through the elements and re-order our elements to produce a sorted array (assuming 1D)
- **Size** - we can know the number of elements in the array by checking each index until we can't access an element

Time:
5 min

Arrays - Implementation Decisions

- How many **dimensions?**
- Is it **fixed-size** or **dynamic-size?** (tuples vs lists)
- What **types** can we have within the array?
- Are the elements **mutable?**
- How much time does **set** and **get** take?
- Are operations between arrays **vectorized?** (not in Python, super fast if you can!)
- Find a comparison of implementations here

Time:
5 min

Arrays - Implementation

- In simple implementations, it is enough to simply store the memory address of the first index of the list.
- Some also store the size to prevent overflowing into another piece of allocated memory
- Accessing and setting the ***i****'th* index is easy. You just take the first point in memory **M** and then multiply the number of bytes **B** per index by the index to get **M+iB** to get the location of the index you want to get/set. This causes the two basic operations to be O(1)

Time:
5 min

Arrays - Dynamic Arrays

- Dynamic Arrays give you more operations
    - Adding - append to end or insert in the middle
    - Deleting - removing the last element or something in the middle
- Since only the memory address of the first item is stored, adding or deleting requires other elements to change memory location to continue to be able to set/get in O(1) time.
- Most dynamic arrays are implemented to have a fixed size (with some elements unused) and then doubling at a new memory space when necessary. This is because leaving room for infinite space is impossible.

Time:
5 min

Associative Arrays

- Associative Arrays are an **Abstract Data Type(ADT)**
- Often called **map, symbol table** or **dictionary**
- They store **key-value** pairs (with unique keys)
- The primary operations in associative arrays are:
  - Addition of a key-value pair - *add(key,value)*
  - Removal of a key-value pair - *remove(key)*
  - Modification of an existing key-value pair - *modify(key,newvalue)*
  - Lookup of a value - *lookup(key)*

Time:
5 min

Associative Arrays - Exercise

- Groups of 3 - discuss the following
- What are the more advanced operations we can do on associative arrays starting from just add, remove, modify and lookup?
- Write the pseudocode for those operations using the basic ones
- This should be independent of the programming language that you use
- You can use operations on atomic data types

Time:
5 min

Associative Arrays - Extra Operations

- **Searching** through keys is easy using lookup
- It is very difficult to do anything else without a **full set of keys!**
- Most implementations will allow you to find what available keys are!
- **Traverse, Searching** values, **Sizing** are now possible with the list of keys
- **Subsetting** or **duplicating** are also possible with the list of keys
- E.g. Finding all keys that have a certain value or counting the number of each value

Time:
5 min

Associative Arrays - Implementation

- 3 Primary ways to implement these
  - Association List (alist) - lists where each element contains a key-value pair
  - Binary Search Tree - a tree where the keys are sorted through the tree's relations
  - Hash Table - a hash function that computes an index for each key which looks up the desired value (**We will look at this on Thursday)**
- You should decide allowable **data types**

Time:
5 min

Associative Arrays - Comparison

Underlying data structure	Lookup		Insertion		Deletion		Ordered
	average	worst case	average	worst case	average	worst case	
Hash table	O(1)	O(*n*)	O(1)	O(*n*)	O(1)	O(*n*)	No
Self-balancing binary search tree	O(log *n*)	O(log *n*)	O(log *n*)	O(log *n*)	O(log *n*)	O(log *n*)	Yes
unbalanced binary search tree	O(log *n*)	O(*n*)	O(log *n*)	O(*n*)	O(log *n*)	O(*n*)	Yes
Sequential container of key-value pairs (e.g. association list)	O(*n*)	O(*n*)	O(1)	O(1)	O(*n*)	O(*n*)	No

Source: Wikipedia - Associative Array

Time:
10 min

Ranges

- Ways of specifying arithmetic sequences over integers
  - range(n) - (1,2,3,.....n-1)
  - range(a,b) - (a,a+1,a+2,......b-1)
  - range(a,b,k) - (a,a+k, a+2k, a+3k, a+4k, ...b-1) (Note: k can be negative)
- Very useful for looping as it is **significantly faster** than for loops and while loops. Doesn't have to create, delete and iterate variables as often.
- There is just one range object and you iterate through it until you've reached the max level.
- You can concatenate ranges and access elements individual elements.

Time:
10 min

Iterator

- Lists, Sets, Tuples, Tuples, Dictionaries, Strings and Range Objects are all iterables - you can convert them to iterators.
- Iter() will convert an iterable to an iterator
- You loop through an iterator using the next() function
- For-loops do this implicitly. They convert an iterable into an iterator type and then loop through them using the next() function.
- While loops are different - there is no clearly defined start and end point.

Time:
10 min

One-line Iteration

- [x+2 for x in [1,2,3]] - create list using one line for loop
- [x + 2 if x%2==1 else 0 for x in range(10)] - if else statement within for loop
- {x+3 for x in (1,2,3)} - create set instead from tuple
- tuple(x +3 for x in {1,2,3}) - create tuple from set
- print(key,value) for key,value in myDict.items() - using key,value in dictionary
- Module **itertools** is very good for more advanced iteration

Time:
1 min

Questions

Time:
3 min

Next Steps

1. Week 3 Readings
2. Week 2 Implementations