Michael Arbogast

Algorithm Sessions

Strings & Associative Arrays

10/31/16 REMOVE BLANKS

Create a `function` that, given a `string`, `return`s the `string` without blanks.

Video Link

11/1/16 GET DIGITS

Create a `function` that, given a `string` that includes numbers, `return`s the integer made from the string's digits. For example, given `"akjsdh45kjhas44489986kl"`, return `4544489986.`
Video Link

11/2/16 REVERSE A STRING & IS PALINDROME

Create a `function` that, given a `string`, `return` a string with the characters in reversed order (do not use the string's built-in reverse( )). For example, given `'creature'`, return `'erutaerc'`

Create a `function` that returns a boolean whether the string is a strict palindrome. Given 'racecar' return true. Given 'oho!', 'Dud', or 'race car' return false. **Bonus-level: DO** ignore whitespaces.
Video Link

11/3/16 PARENS VALID & BRACES VALID

Parens Valid: Create a `function` that, given an input `string` str, `return` a boolean whether the parentheses in `str` are valid. Valid sets of parntheses always open before they close, for example. For "Y(3(p)p(e)r)s", return `true`. Given "n(o(p)3", return `false` ; not every parenthesis is closed. Given ")()(", also return `false`, since there is nothing to open the first or close the last.

Braces valid: Given a sequence of parentheses, braces, and brackets, determine whether it is valid. Example: `"{W[a(t)s]o}n (i[s] [g]r)eat"` is valid.
Video Link

11/4/16 COIN CHANGE WITH OBJECT

Given a number of U.S. cents, return the optimal configuration of coins, in an object.

Video Link

11/7/16 BOOK INDEX

Martin is writing his opus: a book of algorithm challenges, set as lyrics for a suite of baroque fugues. He knows that some of these fugue challenges will be less popular than others - , so he needs an index. Given a sorted array of pages, produce a book index string. Consecutive pages should form ranges separated by a dash. For `[1,3,4,5,7,8,10]`, return the string `"1, 3-5, 7-8, 10"`.

11/8/16 STRING SPLIT & ARRAY JOIN

Split a given string into array of substrings, returning the new array. You'll also be given a separator, which specifies where to divide substrings and is not included in any substring. If `""` is specified, split string on every character.

Join a given array, returning a new string. You'll also be given a separator, which specifies what to place between the array values. Given `[hi,there,you]` and `""`, return `"hithereyou"`. Given `[hi,there,you]` and `"-"`, return `"hi-there-you"`.
Video Link

11/9/2016 PLAYER MOVEMENT TRACKING

Given a sequence of movements as a string (up: "U", down: "D", right: "R", left: "L") in a 2-D game, find me the current `(x,y)` position of the player AND how many times each move was performed. For our purposes, the output will be of the cartesian coordinates and each move will be one unit long. Example: Given `"UULDR"`, return a tuple (or list): `( [0, 2], { 'U' : 2, 'L' : 1, 'D' : 1, 'R': 1} )`

11/10/2016 COMMON SUFFIX

Lance is writing his opus: a tome of beat poetry. Always looking for a good rhyme, he seeks words ending with the same letters. Write a function that, given a word array, returns the largest common suffix(word-end). For the input strings `["deforestation", "placation", "action", "citation"]` return `"tion"` (not a great starting point). If it is `["nice", "ice", "baby"]`, return `""`.
Video Link

11/11/2016 ZIP ARRAYS TO MAP

Associative arrays are sometimes called *maps* because a key (string) maps to a value. Given two arrays, create an associative array (map) containing keys of the first, and values of teh second. For `arr1 = ['abc', 3, 'yo']` and `arr2 = [42, 'wassup', true]`, return `{'abc': 42, 3: 'wassup', 'yo': true}`.
Video Link

11/14/2016 LINKED LIST: ADDFRONT(VAL) && FRONT()

Given a value, create a new node, assign it to the list head, and return a pointer to the new head node.

Return the *value* (not the node) at the head of the list. If the list is empty, return `null`.

11/15/2016 LINKED LIST: REMOVEFRONT() && REMOVEBACK()

Implement a linked list method that removes the front node of a list. If the list is empty, return `null`

Implement a linked list method that removes the last node in a linked list. If the list is empty return `null`
Video Link

11/16/2016 LINKED LIST: CONTAINS(VAL) & LENGTH()

Implement a linked list method that accepts a value and returns `true` if that value is found in the list or `false` if that value is not found in the list.

Implement a linked list method that returns the total number of nodes in the list.

Video Link

11/17/2016 LINKED LIST: ADDBACK(VAL) && REMOVEVAL(VAL)

Implement `.addBack(val)` that adds a `new Node`, with the passed in value, to the back of the linked list.

Next, implement `.removeVal(val)`. This function will remove the first node containing the passed in `val` and `return "node removed"` without damaging the rest of the list. If the list does not contain a node with the passed in value, `return "no node removed"`.

11/18/2016 LINKED LIST: INSERT_AT(INDEX, VAL) AND REMOVE_AT(INDEX)

Given an index and a value, implement `.insert_at(index,val)` that inserts a node with that value at that index. You can treat the first node as being index 0. If the index is out of range, `return "index out of range"`, otherwise return the list.
Given an index, implement `.remove_at(index)` that removes a node at that index. Again, treat the first node as being index 0. If the index is out of range, `return "index out of range"`, otherwise return the list.
Video Link

11/21/2016 LINKED LIST: BUBBLESORT && INVERT HASH

Implement a linked list function that uses bubblesort to sort itself into ascending order. You can choose to implement this as a stand-alone function or as a singly linked list class method.

Invert Hash: Given an object, or associated array with keys and values, return a new object with each key:value pair a reverse of the original object. Example: `invertHash({'a':'b','c':'d'})` should `return {'b':'a','d':'c'}`. There are many edge-cases for this algorithm, ***YOU*** are responsible for documenting them and defending your handling of them. 🙂 . After you have finished implementing this, make another version that is ***in-place*** – meaning you should return the same object that was passed in. More edge cases!