Python useful functions

[base.py] Base change

Binary(n) : returns base-2 conversion

Inputs: n is an integer
Output: converts n to base 2
Examples: Binary(8) = “1000”

Hexa(n) : returns base-16 conversion

Inputs: n is an integer
Output: converts n to base 16
Examples: Hexa(255) = “FF”

Bin2Dec(n) : returns base-10 conversion from base-2

Inputs: n is a string containing a base-2 integer
Output: converts n to base 10
Examples: Bin2Dec(“1110”) = 13

Hex2Dec(n) : returns base-10 conversion from base 16

Inputs: n is a string containing a base-16 integer
Output: converts n to base 10
Examples: Hex2Dec(“777”) = 125

IsBin(n) : returns whether an integer is a binary or not

Inputs: n is an integer
Output: True if n is a binary, False if it is not
Examples: IsBin(“20”) = False
IsBin(“1010”) = True

IsHex(n) : returns whether an integer is hexadecimal or not

Inputs: n is an integer
Output: True if n is hexa, False if it is not
Examples: IsHex(“FFA755”) = True

IsDec(n) : returns whether an integer is decimal or not

Inputs: n is an integer
Output: True if n is decimal, False if it is not
Examples: IsDec(2005) = True

[chars.py] Characters types

IsNumber(s) : returns whether a string is a number or not

Inputs: s is a string

Output: True if it's a number, False otherwise

Examples: IsNumber("1950") = True
IsNumber("five") = False

IsUpper(s) : returns whether a string is in all caps or not

Inputs: s is a string

Output: True if it's allcaps, False otherwise

Examples: IsUpper("MONDAY") = True
IsUpper("YOLOSwag") = False

IsLower(s) : returns whether a string in lowercase or not

Inputs: s is a string

Output: True if it's in lowercase, False otherwise

Examples: IsLower("monday") = True

IsLetter(s) : returns whether a string is only letters or not

Inputs: s is a string

Output: True if it's only letters, False otherwise

Examples: IsLetter("MoonDay") = True

IsAlphanum(s) : returns whether a string is alphanumeric or not

Inputs: s is a string

Output: True if it's alphanumeric, False otherwise

Examples: IsAlphanum("Swag4Ever") = True

IsSpecial(s) : returns whether a string is only special characters or not

Inputs: s is a string

Output: True if it's only special characters, False otherwise

Examples: IsSpecial("-_.!?") = True
IsSpecial("@gmail") = False

StringIs(s) : returns the types of characters composing a string

Inputs: s is a string

Output: an array of values between 0 and 3
0 - Number, 1 -Uppercase, 2 - Lowercase, 3 - Special

Examples: StringIs("Y0l0-Sw4g") = [1,0,2,0,3,1,2,0,2]

[comb.py] Combinations, sets & permutations

A) Without repetition

allPerm(l, n) : returns all permutations of all subsets using n elements of l

Inputs: l is a sorted list, in ascending order
n is the size of the generated subsets
Output: an array with all permutations using n elements ofl
Examples: allPerm([1,2,3], 2)
[[1, 2], [1, 3], [2, 1], [2, 3], [3, 1], [3, 2]]

allSets(l, n) : returns all ordered subsets generated using n elements of l

Inputs: l is a sorted list, in ascending order
n is the size of the generated subsets
Output: an array with all ordered subsets using n elements of l
Examples: allSets([1,2,3], 2)
[[1, 2], [1, 3], [2, 3]]

B) Without repetition

allGen(l, n) : returns all subsets generated using n elements from l

Inputs: l is a sorted list, in ascending order
n is the size of the generated subsets
Output: an array with all subsets using n elements from l
Examples: allGen([1,2,3], 2)
[[1, 1], [1, 2], [1, 3], [2, 1], [2, 2], [2, 3], [3, 1], [3, 2], [3, 3]]

allSetsR(l, n) : returns all ordered subsets generated using n elements from l

Inputs: l is a sorted list, in ascending order
n is the size of the generated subsets
Output: an array with all ordered subsets using n elements from l
Examples: allSetsR([1,2,3], 2)
[[1, 1], [1, 2], [1, 3], [2, 2], [2, 3], [3, 3]]

[common.py] String and list intersection

Common(l1,l2) : returns all element two lists/strings have in common

Inputs: l1, l2 are both lists/strings
Output: an array of all element both in l1 and l2 / a string
Examples: Common([1,2,3,4,5], [1,3,5,7,9]) = [1,3,5]
Common(“Monday”, “YoloSwag”) = “oay”

LargestSublist(l1,l2) : returns the largest sublist l1 and l2 have in common

Inputs: l1, l2 are both lists
Output: an array of prime numbers which composed the integer n
Examples: LargestSublist([1,2,3,4,5,6], [2,5,3,4,1]) = [3,4]

LargestSubstring(w1,w2) : return the largest string s1 and s2 have in common

Inputs: w1, w2 are both strings
Output: the largest consecutive substring
Examples: LargestSubstring(“baccalaureat”, “laurealparis”) = “laurea”

[dich.py] Dichotomy

dich(l, t) : returns if an element is in a list and where it should be

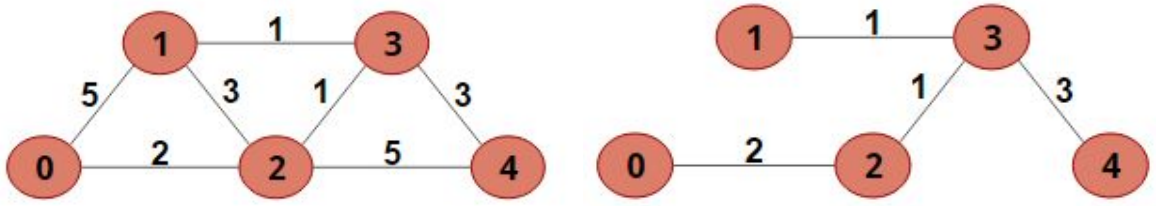
Inputs: l is a sorted list, in ascending order
t is an element that might be in l
Output: [bool, index]
bool indicates whether t is in l
index where t should be inserted.
Examples: dich([1,2,3,5,6], 2) = [True,1] *# 2 is already in l at index 1*
dich([1,2,3,5,6], 4.5) = [False,3] *# 4.5 isn’t in l and should be at index 3*

[graph.py] Graph Theory

A) Minimal Spanning Tree

MST(links) : returns the MST of a linked graph

Inputs: links represent all links of a given graph
Output: all necessary links to the MST
Examples: for this graph, it outputs:
[[[2, 2]], [[3, 1]], [[0, 2], [3, 1]], [[2, 1], [1, 1], [4, 3]], [[3, 3]]]

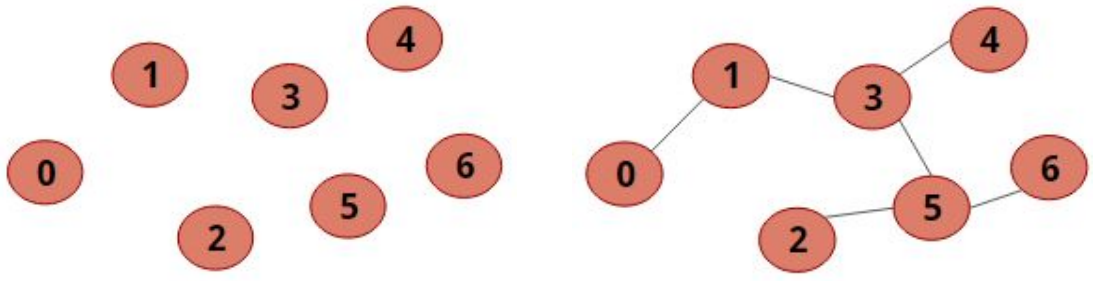


MSTUnitary(links) : returns the MST of a unitary linked graph

Inputs: points is an array of coordinates [x,y]
Output: all necessary links to the MST
Examples: MSTUnitary([[1,2], [0,2], [0,1]])
[[[1,1],[2,1]], [[0,1], [3,1]], [[0,1]], [[2, 1]]]

MSTComplete(points) : returns the MST of a complete graph

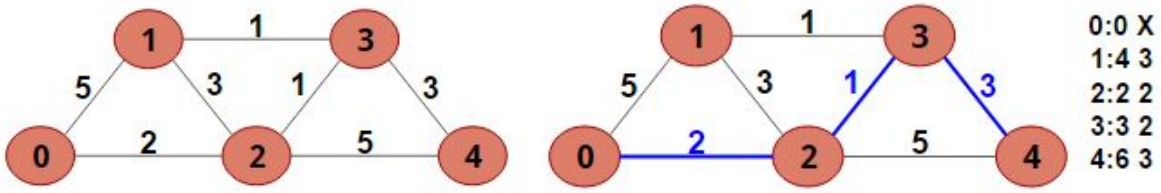
Inputs: points is an array of coordinates [x,y]
Output: all necessary links to the MST
Examples: MSTComplete([[0,0], [-1,1.5], [2,1], [3,2.5], [2.5,-0.5], [0,3.5]])
[[[1,1],[2,1]], [[0,1], [3,1]], [[0,1]], [[2, 1]]]



B) Graph Pathfinding

Pathfind(s, links) : pathfind from s through a linked graph

Inputs: s is the starting index, links is an array of valued links
Output: [D, A] minimal distance D and its direct antecedent A for each point
Examples: for this graph, it outputs:
[[0, -1], [4, 3], [2, 0], [3, 2], [6, 3]]

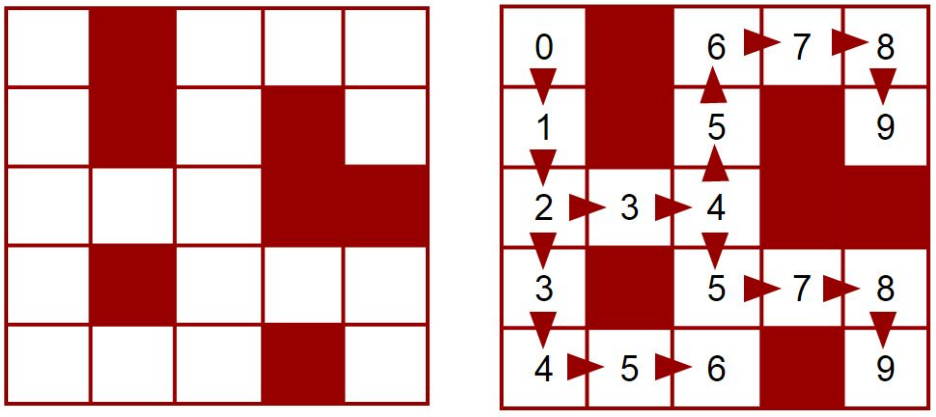


PathfindUnitary(s, links) : pathfind from s through a linked unitary graph

Inputs: s is the starting index, links is an array of unitary links
Output: [D, A] minimal distance D and its direct antecedent for each point
Examples: MSTUnitary([[1,2], [0,2], [0,1]])
[[0, -1], [1, 0], [1, 0]]

PathfindGrid(s, grid) : pathfind from s through a grid

Inputs: s is the starting index, grid is an array filled with 1 (struct) and 0.
Output: [D, A] minimal distance D and its direct antecedent for each point
Examples: For this grid starting from 0:



[knapsack.py] Knapsack problem

dynExist(l, t) : returns whether there is a solution to a knapsack problem

Inputs: l is an array of numbers
t is a number, stands for target
Output: True if you can add elements of l to obtain t, False otherwise
Examples: dynExist([1,2,3,4,5,6], 10) = True
dynExist([2,4,6,8,10], 13) = False

dynSmallest(l, t): returns the smallest solution to a knapsack problem

Inputs: l is an array of numbers
t is a number, stands for target
Output: the smallest sublist of l that can add up to t, False if there's none
Examples: dynSmallest([1,2,3,4,5,6], 10) = [4,6]

[primes.py] Prime numbers and factorisation

Prime(n) : returns a list of all prime numbers lower than n

Inputs: n is an integer
Output: an array of all primes < n
Examples: Prime(8) = [1,2,3,5,7]

NthPrime(n) : returns the nth prime number

Inputs: n is an integer
Output: the nth prime number
Examples: NthPrime(6) = 11

IsPrime(n) : returns whether an integer is a prime or not

Inputs: n is an integer
Output: True if n is prime, False otherwise
Examples: IsPrime(8) = False
IsPrime(23) = True

ClosestPrime(n) : returns the closest prime number of n

Inputs: n is an integer
Output: the closest prime number of n
Examples: ClosestPrime(100) = 101

DecompFactor(n) : returns the decomposition of an integer n

Inputs: n is an integer
Output: an array of prime numbers that compose the integer n
Examples: DecompFactor(20) = [2,2,5]