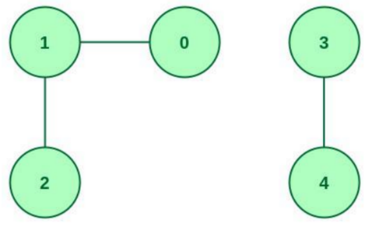
COL106: Assignment 5

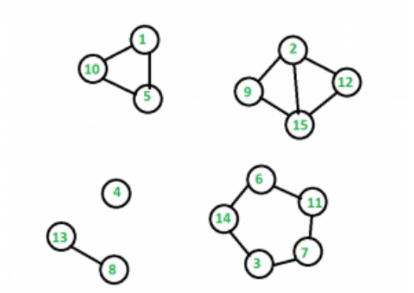
Version 2: Modified Q2, Corrected test case in Q4, added Q5, Q6.

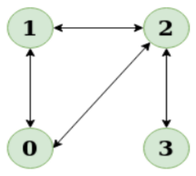
Question 1 - Use BFS/DFS

Find the number of connected components in a given graph. The graph input can be taken by first taking an integer input *n*, and a representation of the graph as either an adjacency matrix or an adjacency list.

Sample Test Cases

1. 

In this graph, the number of connected components is 2. According to the above input format, you can write code such that the above graph can be inputted as:
n = `5`
Adjacency list: `[[1], [0,2], [1], [4], [3]]`
2. 

In this graph, the number of connected components is 5. According to the above input format, you can write code such that the above graph can be inputted as: (above graph does not have a 0 node, so we can just consider all node values decremented by 1 (node 1 becomes node 0, etc))
n = `15`
Adjacency list: `[[4,9], [8,11,14], [6,13], [], [0,9], [10,13], [2,10], [12], [1,14], [0,4], [5,6], [1,14], [7], [2,5], [1, 8,11]]`
3. 

In this graph, the number of connected components is 1 (consider all edges as undirected). According to the above input format, you can write code such that the above graph can be inputted as:
n = 4
Adjacency list: `[[1,2], [0,2], [0,1,3], [2]]`

Question 2 - Use Trie

Build a search suggestion system: Given an array of product names build a trie containing all product names. Then, for every query string return a list of all products which could be possible autocomplete options after the given query string, that is, the query string should be the prefix of those product names.

Sample Test Cases

Product Names: products = ["mobile","mouse","moneypot","monitor","mousepad"]
Queries = [“m”, “mon”, “t”, "mouse"]
Once you build the trie, it should be easy to return the answers to each of the above queries.
Sample outputs:

1. For “m”: ["mobile","mouse","moneypot","monitor","mousepad"]
2. For “mon”: ["moneypot","monitor"]
3. For “t”: []
4. For “mouse”: ["mouse","mousepad"]

Question 3 - Use Binary Search

Given an array of strings, find the longest common prefix string.
This was earlier incorrectly tagged as a question where you could use KMP, however that would be very inefficient, and would not use the properties of the KMP algorithm to the fullest. Refer the two additional questions added in this document for practice of the KMP algorithm

Sample Test Cases

1. Input strings = [“abcdefgh”, “abcefgh”]
   Output = “abc”
2. Input strings = [“abcdefgh”, “aefghijk”, “abcefgh”]
   Output = “a”
3. Input strings = [“sadadadaghjt”, “sadadadaghjt”, “sadadaghjt”, “sadadadadadadaghjt”]
   Output = “sadada”

Question 4 - Stable Bucket Sort

Given a Vector of Vectors of length *n*, where each internal vector is of length 2 and contains integers, write a stable Bucket sort algorithm which sorts the outer vector according to the first element in the inner vectors only, in O(n) time. Consider m is the maximum integer value present in the vectors. You are guaranteed that m is at most *max(1000, 3n)*. (Therefore an O(m) solution will also be O(n))

Sample Test Cases

1. Input = `[[2,4], [2,6], [1,10], [1,5], [1,7]],`
   Output = `[[1,10], [1,5], [1,7], [2,4], [2,6]];`
2. Input = `[[50,20], [10,30], [20,40], [50,10], [10,20], [15,25]]`
   Output = `[[10,30], [10,20], [15,25], [20,40], [50,20], [50,10]]`

3. Input = `[[35, 45], [50, 10], [35, 30], [50, 20], [10, 40], [35, 25], [10, 30], [50, 40], [35, 35], [10, 20]],`
   Output = `[[10, 40], [10, 30], [10, 20], [35, 45], [35, 30], [35, 25], [35, 35]], [50, 10], [50, 20], [50, 40];`

Question 5 - Direct KMP

Given a string *s*, and a pattern *p*, find the number of times the pattern *p* occurs in the string *s*. This is a straightforward application of the KMP algorithm.

Sample Test Cases

1. s = “AABAACAADAABAABA”, p = “AABA”
   Output = 3 (The patterns are bolded here: **AABA**ACAAD**AABAABA**)
2. s = “AAAAAAAAAA”, p = “AAA”
   Output = 8 (Pattern can start at any index in [0,7] (0-indexed))
3. s = “heyhihey”, p = “heyi”
   Output = 0 (Pattern does not exist in string)

Question 6 - Indirect KMP, slightly challenging

Given a string *s*, find the minimum number of letters that must be added to it to convert it into a palindrome.
Hint: This doesn’t use KMP directly, but does use the way we build the LPS lookup table. The essential idea is to find the length of the longest palindrome already present in the string, which can be cleverly reduced to the problem of finding the longest prefix that is also a suffix.

Sample Test Cases

1. s = “aacecaaa”
   Output: 1. We can add 1 ‘a’ at the beginning to make it a palindrome: “aaacecaaa”
2. s = “abcd”
   Output: 3. We need to add three characters: “abcdcba”
3. s = “aaaaaaa”
   Output: 0. The string is already a palindrome.