2020-06-27 - Handout – Data Structure Design

# Q1. Implement Trie (Prefix Tree)

Link:<https://leetcode.com/problems/implement-trie-prefix-tree/>

Implement a trie with insert, search, and startsWith methods.

**Example:**

Trie trie = new Trie();

trie.insert("apple");

trie.search("apple"); // returns true

trie.search("app"); // returns false

trie.startsWith("app"); // returns true

trie.insert("app");

trie.search("app"); // returns true

**Note:**

* You may assume that all inputs are consist of lowercase letters a-z.
* All inputs are guaranteed to be non-empty strings.

# Q2.Design In-Memory File System

Link: <https://leetcode.com/problems/design-in-memory-file-system/>

Design an in-memory file system to simulate the following functions:

ls: Given a path in string format. If it is a file path, return a list that only contains this file's name. If it is a directory path, return the list of file and directory names **in this directory**. Your output (file and directory names together) should in **lexicographic order**.

mkdir: Given a **directory path** that does not exist, you should make a new directory according to the path. If the middle directories in the path don't exist either, you should create them as well. This function has void return type.

addContentToFile: Given a **file path** and **file content** in string format. If the file doesn't exist, you need to create that file containing given content. If the file already exists, you need to **append** given content to original content. This function has void return type.

readContentFromFile: Given a **file path**, return its **content** in string format.

**Example:**

**Input:**

["FileSystem","ls","mkdir","addContentToFile","ls","readContentFromFile"]

[[],["/"],["/a/b/c"],["/a/b/c/d","hello"],["/"],["/a/b/c/d"]]

**Output:**

[null,[],null,null,["a"],"hello"]

# Q3. LRU Cache

Link:<https://leetcode.com/problems/lru-cache/>

Design and implement a data structure for [Least Recently Used (LRU) cache](https://en.wikipedia.org/wiki/Cache_replacement_policies#LRU). It should support the following operations: get and put.

get(key) - Get the value (will always be positive) of the key if the key exists in the cache, otherwise return -1.  
put(key, value) - Set or insert the value if the key is not already present. When the cache reached its capacity, it should invalidate the least recently used item before inserting a new item.

The cache is initialized with a **positive** capacity.

**Follow up:**  
Could you do both operations in **O(1)** time complexity?

**Example:**

LRUCache cache = new LRUCache( 2 /\* capacity \*/ );

cache.put(1, 1);

cache.put(2, 2);

cache.get(1); // returns 1

cache.put(3, 3); // evicts key 2

cache.get(2); // returns -1 (not found)

cache.put(4, 4); // evicts key 1

cache.get(1); // returns -1 (not found)

cache.get(3); // returns 3

cache.get(4); // returns 4