1. Tooltip

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <link rel="stylesheet" type="text/css" href="./index.css">
    <title>Document</title>
</head>
<body>
    <div class="tooltip">
        Hover over me
        <span class="tooltiptext">Tooltip text</span>
    </div>
</body>
</html>
.tooltip {
    position: relative;
}

.tooltip .tooltiptext {
    visibility: hidden;
    /* Position the tooltip text */
    position: absolute;
    background-color: gray;
    z-index: 1;
}
.tooltip:hover .tooltiptext {
    visibility: visible;
}
```

2. Longest Common Prefix

```
class Solution:
    def longestCommonPrefix(self, strs: List[str]) -> str:
        if not strs:
```

```
        return ''
    min_length = min(map(len, strs))
    cur = strs[0][:min_length]
    for s in strs[1:]:
        for i in range(len(cur)):
            if cur[i] != s[i]:
                cur = cur[:i]
                break
    return cur
```

3. Given a list of folder names, findout their top most common parent folder

```
# ["abc/def/opq", "abc/def", "xyz"]
def longest_parent_path(strs):
    paths = {}
    result = []

    def dfs(cur_node, cur_path):
        if "is_end" in cur_node:
            return cur_path
        else:
            if len(cur_node.keys()) == 1 and "is_end" not in cur_node:
                key = list(cur_node.keys())[0]
                return dfs(cur_node[key], cur_path + [key])

    for s in strs:
        node = paths
        for dire in s.split('/'):
            if dire not in node:
                node[dire] = {}
            node = node[dire]
        node["is_end"] = True
    for s in paths.keys():
        result.append('/'.join(dfs(paths[s], [s])))
    return result


print(longest_parent_path(["abc/def/opq", "abc/def", "xyz"]))
```

4. LRU Cache

```
class LinkedNode:
    def __init__(self, val, key):
        self.val = val
        self.next = None
        self.prev = None
        self.key = key

class LRUCache:

    def __init__(self, capacity: int):
        self.nodes = {}
        self.capacity = capacity
        self.memo = LinkedNode(0, "#")
        self.memo.next = self.memo
        self.memo.prev = self.memo

    def get(self, key: int) -> int:
        if key not in self.nodes:
            return -1
        node = self.nodes[key]
        self.remove_node(node)
        self.insert_to_head(node)
        return node.val

    def put(self, key: int, value: int) -> None:
        if key not in self.nodes:
            if self.capacity == 0:
                del_key = self.memo.prev.key
                self.remove_node(self.memo.prev)
                del self.nodes[del_key]
            else:
                self.capacity -= 1
            node = LinkedNode(value, key)
            self.nodes[key] = node
        else:
            node = self.nodes[key]
            node.val = value
            self.remove_node(node)
        self.insert_to_head(node)

    def insert_to_head(self, node):
        node.next = self.memo.next
        self.memo.next.prev = node
```

```
        self.memo.next = node
        node.prev = self.memo

    def remove_node(self, node):
        node.prev.next = node.next
        node.next.prev = node.prev
```

207. Course Schedule
O(E)

```
from collections import defaultdict
class Solution:
    def canFinish(self, numCourses: int, prerequisites: List[List[int]]) -> bool:
        memo = defaultdict(set)
        degrees = defaultdict(int)
        result = 0
        for s, e in prerequisites:
            memo[e].add(s)
            degrees[s] += 1

        cur = set(range(numCourses)) -  set(degrees.keys())
        result = len(cur)
        while cur:
            next_ite = set()
            for i in cur:
                if i in memo:
                    for j in memo[i]:
                        degrees[j] -= 1
                        if degrees[j] == 0:
                            next_ite.add(j)
                            result += 1
                del memo[i]
            cur = next_ite
        return result == numCourses
```