Lesson 23

PA 18

```
def rotations(n):
    s = str(n)
    result = []
    result.append(int(s))
    for i in range(len(s)-1):
        s = s[1:] + s[0]
        if int(s) not in result:
            result.append(int(s))
    return result
```

```
def is_prime(n):
    if n <= 1:
        return False
    for i in range(2,int(sqrt(n))+1):
        if n % i == 0: # i is a factor of n
            return False
    return True
```

Credit: Ryan

```
def is_prime(n):
    # Write your function to check for prime number here
    if n <= 3:
        return n > 1
    elif n % 2 == 0 or n % 3 == 0:
        return False
    i = 5
    while i * i <= n:
        if n % i == 0 or n % (i + 2) == 0:
            return False
        i += 6
    return True
```

Credit: Yiming

```
def iscircularprime(n):
    for i in rotations(n):
        if not is_prime(i):
            return 0
    return 1

def count_circular_primes(n):
    res = 0
    for i in range(n+1):
        res += iscircularprime(i)
    return res
```

Credit: Yiwen

```
def get_num_flights(src, dst, filename):
    f = import_csv(filename)
    ans = 0
    for i in f:
        if (i[1] == src and i[2] == dst):
            ans += 1
    return ans
```

reverse dictionary

```
def reverse_dict(d):
    result = {}
    for key,value in d.items():
        if value not in result.keys():
            result[value] = [key]
        else:
            result[value].append(key)
    return result
```

```
def get_top_k_hubs(k, filename):
    f = import_csv(filename)
    ap = {}
    for line in f:
        if line[1] not in ap:
            ap[line[1]] = 1
        else:
            ap[line[1]] = ap.get(line[1]) + 1

        if line[2] not in ap:
            ap[line[2]] = 1
        else:
            ap[line[2]] = ap.get(line[2]) + 1
    r_ap = reverse_dict(ap)
    k_keys = sorted(r_ap.keys())[-k:][::-1]
    result = []
    for key in k_keys:
        result.append(tuple([str(r_ap[key])[2:-2],key]))
    return result
```

Credit: Nathan

```
def search_routes(src, dest, filename, n):
    ans = []
    lis = []
    newlis = []

    for i in range(n):
        for l in import_csv(filename):
            if i == 0:
                if l[1] == src:
                    newlis.append([src,l[2]])
            else:
                for v in lis:
                    if l[1] == v[-1]:
                        newlis.append(v + [ l[2] ])
        for v in newlis:
            if v[-1] == dest and len(v) == len(set(v)) and v not in ans:
                ans.append(v)
        lis = newlis
        newlis = []

    return ans
```

Using Breadth First Search
Credit: Yiming

```
from collections import deque
def search_routes(src, dest, filename, n):
    adj = dict()
    data = import_csv(filename)
    for i in data:
        if i[1] not in adj:
            adj[i[1]] = []
        adj[i[1]].append(i[2])
        # adj[x] stores every node that x can reach in 1 edge/1 "hop"
    '''
    Breadth-first search
    visualize at https://visualgo.net/en/dfsbfs !!!
    '''
    dq = deque() # deque is just a list, but it's faster if you want to access and remove the first element
    res = []
    dq.append([src]) # at the start, the only possible route so far is just only the source node
    while len(dq):
        cur = dq[0][:] # copy the first element in the deque to cur.
        # cur is the current route that we are considering.
        if len(cur) > n+1:
            # this route is not good and there is no way it can be a valid route. furthermore, everything
            # behind it is also the same length or longer. So no point continuing.
            break
```

```
        break
    dq.popleft() # remove the current element from the deque
    if cur[-1] == dest and len(cur) <= n+1:
        # cur[-1] is the last element inside the current route that we are considering. so if the last
        # element is the destination, then it is a possible route
        # just len(cur) <= n+1 is just another check that it's not too long
        if cur not in res: # so that no route is repeated. This may be possible? I don't think so.
                           # Just to be safe :)
            res.append(cur)


    if cur[-1] not in adj:
        # adj is the adjacency list. If it's not inside, that means no flights are going out of the last
        # node so far. So this route is not good.
        continue
    for i in adj[cur[-1]]: # we try to extend the current route.
        if i in cur:
            continue # we cannot go to the same node more than one time
        tmp = cur[:] # copy this. [:] is so that it copies the array instead of the memory address.
        tmp.append(i) # we append this node to the route we are considering
        dq.append(tmp) # we append this route to every route, so that this can be used again later.
return res
```

Credit: Nathan

```
def __init__(self,name):
    self.name = name
    self.tree = {}
    self.lockstree = {}
```

```
def get_ancestors(self,tech):
    ans = []
    if tech not in self.tree:
        return False
    elif self.get_parents(tech) == []:
        return []
    else:
        for i in self.get_parents(tech):
            if i not in ans:
                ans.append(i)
            ans += self.get_ancestors(i)
        return ans
```

```
def has_loop(self):
    for i in self.tree:
        try:
            if i in self.get_ancestors(i):
                return True
        except RecursionError:
            return True
    return False
```

Credit: Ryan

```
from collections import deque

class TechTree:
    # Write your solution here
    def __init__(self, name):
        self.name = name
        self.adjlist = {}
        self.backedges = {}
        self.unlocked = {}
    def add_tech(self, name):
        if name in self.adjlist:
            return False
        self.adjlist[name] = []
        self.backedges[name] = []
        self.unlocked[name] = False
        return True

    def add_dependency(self, parent, child):
        if parent not in self.adjlist or child not in self.adjlist:
            return False
        self.adjlist[parent].append(child)
        self.backedges[child].append(parent)
    def get_ancestors(self, tech):
        if tech not in self.adjlist:
            return False
        bfs_queue = deque()
        bfs_queue.append(tech)
        all_ancestors = []
        while len(bfs_queue) > 0:
            last_tech = bfs_queue.pop()
            for parent in self.get_parents(last_tech):
                if parent not in all_ancestors:
                    all_ancestors.append(parent)
                    bfs_queue.append(parent)
        return all_ancestors


    def has_loop(self):
        for tech in self.adjlist:
            if tech in self.get_ancestors(tech):
                return True
        return False
```