DATA-Lab89

Word Problem

Larry is a strange child. He likes to put his vegetables on different plates and then place straws between each plate. Due to Larry's strangeness, he won't eat his dinner until all plates have only one type of vegetable and no two plates joined by a straw have the same type of vegetable.

⇒ **Conditions:**

(1) Each plate has only one type of vegetable

(2) No 2 plates joined by straw have the same type of vegetable

Problem summary

Input: An array that contains all plates with its attribute (vegetable and linked_plates), *P*

Output:

True P[n].vegetable.length == 1 for n from 1 to P.length) and (linked(plates[i], plates[j]) and (plates[i].vegetable ≠ plates[j].vegetable)) or (not linked(plates[i], plates[j]) for i,j from 1 to P.length)

False otherwise.

Pseudo-code

```
VEG-ON-PLATE-GREEDY(P):
if P.length == no_type_vegetable:
    return True
elif P.length < no_type_vegtable:
    return False
for i from 1 to P.length:
    if P[n].vegetable.length != 1:
        return False
    for p in P[i].linked_plates:
        if p.vegetable == P[i].vegetable:
            return False
return True
```

```
VEG-ON-PLATE(P):
if P.length == no_type_vegetable:
    return True
```

```
elif P.length < no_type_vegtable:
    return False
for i from 1 to P.length:
    if P[n].vegetable.length != 1:
        return False
    for p in P[i].linked_plates:
        if p.vegetable != P[i].vegetable:
            P[i].linked_plates.delete(p)
            p.linked_plates.delete(P[i])
           //They are no longer linked
        else:
            return False
return True
```

Correctness of algorithm

VEG-ON-PLATE-GREEDY(P): At the start of each iteration, the subarray P[1..j-1] contains only plates that have only one vegetable which is different from one on any plate they are linked to.

Initialization: When i=1, the subarray is completely empty ⇒ Obviously true

Maintenance: Assuming that the loop variant is true at iteration i:

- if P[i].vegetable.length ≠ 1, then False is returned, which means there is no more iteration
- else P[i].vegetable is compared with each plate P[i] linked to
    - if P[i] and that plate have the same vegetable, False is returned, which means there is no more iteration
    - else continue

  ⇒ At the start of next iteration i+1 , A[1..i] contains only plate that has only one vegetable which is different from one on any plate they are linked to.

Termination: There are three cases:

- The loop ends and return False because there exists a plate that is empty or has more than one type of vegetable. (Violating (1))
- The loop ends and return False because there exists a plate that has the same vegetable with another plate it is linked to. (Violating (2))
- The loop ends after i goes all from 1 to P.length, which means each plate is visited and its link are checked, both condition are passed, True is returned

VEG-ON-PLATE(P): At the start of each iteration, the subarray P[1..j-1] contains only plates that have only one vegetable which is different from one on any plate they are linked to. Moreover, those plates do not have any link to plates in the subarray P[j..P.length]

Initialization: When i=1, the subarray is completely empty ⇒ Obviously true

Maintenance: Assuming that the loop variant is true at iteration i, there is no link between P[1..i-1] and P[i..P.length]

- if P[i].vegetable.length ≠ 1, then False is returned, which means there is no more iteration
- else P[i].vegetable is compared with each plate P[i] linked to
  - if P[i] and that plate have the same vegetable, False is returned, which means there is no more iteration
  - else, P[i] is deleted from the linked_plates list of that plate, and vice versa.

⇒ P[i] no longer has link with P[i..P.Length]. At the start of next iteration i+1 , A[1..i] contains only plate that has only one vegetable which is different from one on any plate they are linked to, and there is no link between 2 subarray

Termination: There are three cases:

- The loop ends and return False because there exists a plate that is empty or has more than one type of vegetable. (Violating (1))
- The loop ends and return False because there exists a plate that has the same vegetable with another plate it is linked to. (Violating (2))
- The loop ends after i goes all from 1 to P.length, which means each plate is visited and its link are checked (ie. no link between P[i..P.length] and P[P.length..P.length], which is empty) both condition are passed, True is returned

Complexity

Space complexity:

For each plate in P, there are maximum (n_plate-1) links with other plate. Then linked_plates have at max (n_plate-1) elements.

Space complexity = Space_to_store_vegetable + Space_to_store_linked_plates

= O(n_plate) + n_plate * O(n_plate-1)

= O(n_plate^2)

Time complexity:

There are maximum n_plate*(n_plate-1)/2 links in total.

The linked_plates list is a stack implemented with a doubly linked list. The key of each element in the stack is the index of the linked plate in P. Then, delete() function runs in O(1). Stack is chosen because it is easier to understand visually instead of just deal only *head* attribute of the linked list

VEG-ON-PLATE-GREEDY(P): n_plate * Time_to_check_condition_with_other_linked_plates

= n_plate * O(n_plate*(n_plate-1)/2)

= O(n_plate^3)

VEG-ON-PLATE(P): Time_to_visit_each_plate + Time_to_check_each_link

= O(n_plate) + O(n_plate*(n_plate-1)/2)

= O(n_plate^2)

Implementation

1. **Input:** All numbers are integer

Variable	Definition	Condition
n_plate	Number of plate used	n_plate > 0
n_type_vegetable*	Number of types of vegetable used	0 < n_type_vegetable <= n_plate
n_straw	Number of straw used to connect plates	0 <= n_staw <= n_plate*(n_plate-1)/2
veg_order**	A 1D array show how vegetable are set on each plate in order from plate 0 to plate (n_plate-1).	len(veg_order) == n_plate; 0 <= veg_order[i] < n_type_vegetable for i in range(len(veg_order))
plate_links	A 2D array, each row contains a couple of plates linked with each other	len(plate_links) == n_straw, 0 <= plate_links[i][j] < n_plate for i,j in range(n_plate)
plates	*A 1D array contains all plates with its information (vegetable, linked_plates)*	

** n_type_vegetable > n_plate (Upper bound) means that there are at least a plate with more than 1 vegetable, violating (1)*

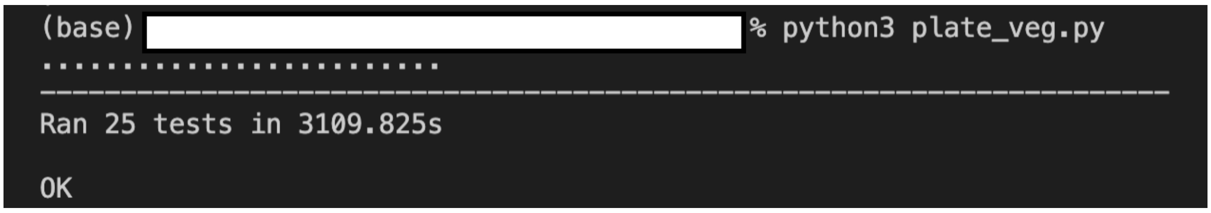
***len(veg_order) < n_plate means there is at least an empty plate, violating (1)*

Except for * and **, other case that does not meet the condition will raise an error in the implementation.

2. **Output:**

   TRUE: Both conditions are met
   FALSE: Either of condition (1) or (2) is not met

Testing



```
(base) [redacted] % python3 plate_veg.py
.........................
----------------------------------------------------------------------
Ran 25 tests in 3109.825s

OK
```