PYTHON

Floydov algotitem

Definicija problema

Python urejevalnik se nahaja na strani:
`https://www.w3resource.com/python-exercises/python-basic-exercises.php#EDITOR`

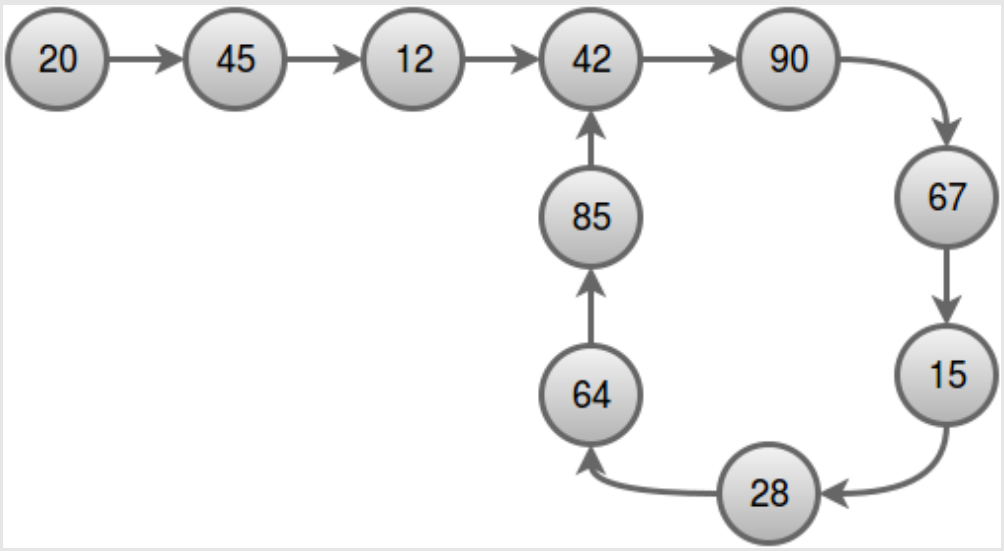
Floydov algoritem, imenovan tudi algoritem *želve in zajca* oziroma *Floydov algoritem za zaznavanje ciklov*. To je algoritem za zaznavanje ciklov v povezanih seznamih ali zaporedjih. Deluje tako, da uporablja dva kazalca, ki se pomikata skozi zaporedje oziroma seznam. Eden imenujmo ga *zajec* se premika skozi zaporedje hitreje od drugega *želve*. Če obstaja cikel v zaporedju, se bosta oba kazalca nekje srečala.

Algoritem

Kako deluje algoritem?

1. Kazalec *želvo* postavimo na začetek zaporedja, medtem ko drugega *zajca* pomaknemo nekoliko naprej.
2. Hkrati pomikamo oba kazalca naprej, pri čemer se želva premika po eno mesto v zaporedju, medtem ko se *zajec* pomakne za dve mesti v zaporedju.
3. Če zajec pride do konca zaporedja (tj. kazalec ne kaže nikamor `null`), potem v zaporedju ni cikla.
4. Če se zajec sreča s želvo (tj. kazalca kažeta na isti vozel), potem v zaporedju obstaja cikel.

Če želimo ugotoviti, kje je začetek cikla, potem postavimo *želvo* na začetek zaporedja in oba pomikamo za eno mesto naprej, dokler se znova ne srečata. Vozlišče v katerem se srečata ponovno je začetek cikla.
Floydov algoritem ima časovno zahtevnost O(n), kjer je n dolžina zaporedja.


20
45
12
42
90
67
85
15
64
28

Rešitev

```python
#!/usr/bin/env python3
# Definition of a singly-linked list node
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next

# Function to detect a cycle in a linked list using Floyd's algorithm
def detect_cycle(head: ListNode) -> ListNode:
    slow = head
    fast = head

    # Move the fast pointer two nodes at a time and the slow pointer one node at a
    ↪  time
    while fast and fast.next:
        slow = slow.next
        fast = fast.next.next

        # If the two pointers meet, there is a cycle
        if slow == fast:
            break

    # If the fast pointer reaches the end of the list, there is no cycle
    if not fast or not fast.next:
        return None

    # Move one of the pointers back to the head of the list
    slow = head

    # Move both pointers one node at a time until they meet at the start of the
    ↪  cycle
    while slow != fast:
        slow = slow.next
        fast = fast.next

    return slow

# Example usage
if __name__ == '__main__':
    # Create a linked list with a cycle
    node1 = ListNode(1)
    node2 = ListNode(2)
    node3 = ListNode(3)
    node4 = ListNode(4)
    node5 = ListNode(5)
    node1.next = node2
    node2.next = node3
    node3.next = node4
    node4.next = node5
    node5.next = node2  # Cycle

    # Detect the cycle and print the value of the node where the cycle starts
    cycle_start = detect_cycle(node1)
    if cycle_start:
        print(f"Cycle detected. Start node has value: {cycle_start.val}")
    else:
        print("No cycle detected.")
```