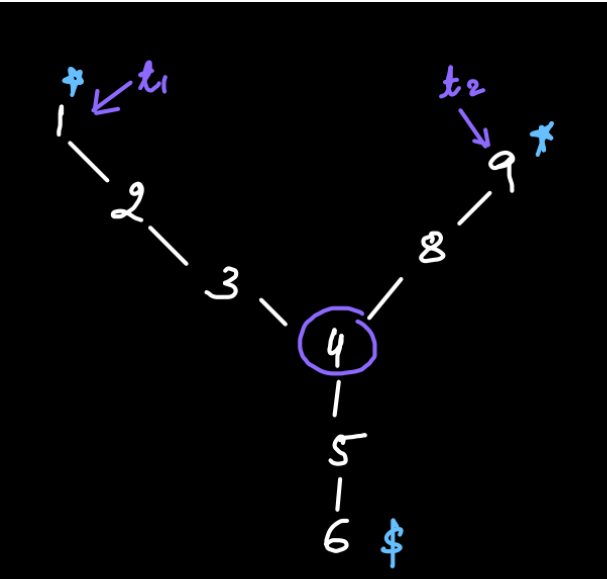
**1. Problem Discussion**

In the given problem, a function is passed two linked lists which start separately but merge at a node and become common thereafter. The function is expected to find the point where two linked lists merge. You are not allowed to use arrays to solve the problem.

CONSTRAINTS

1• Time complexity -> O(n) 2• Space complexity -> constant

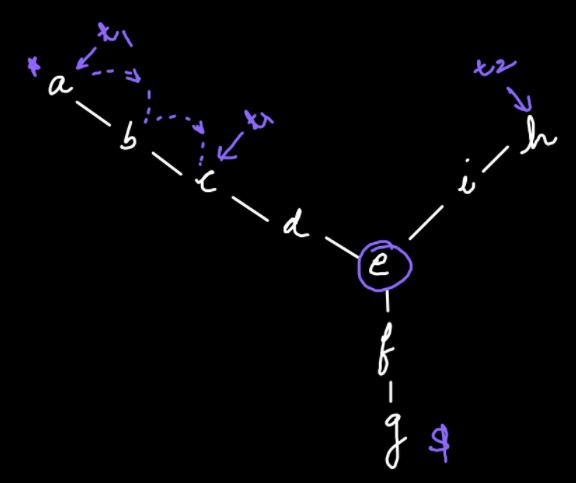
Let the 2 input linked lists be 1-2-3-4-5-6 and 9-8-4-5-6 as shown in the figure 1.

****

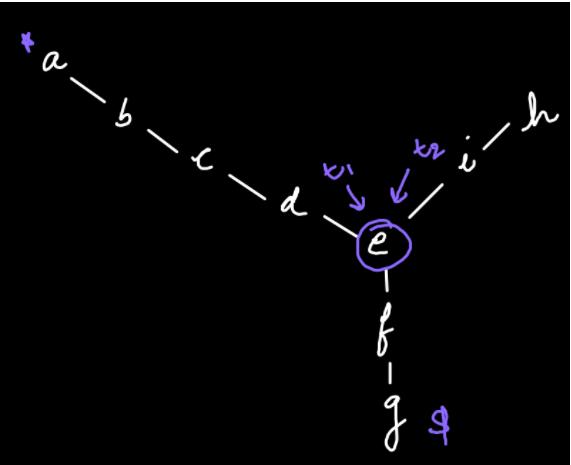
Then, these two linked lists merge with each other at an intersection point "4". The length of arms before the intersection point may not be the same for the linked lists. In our example, the length of the arm for the 1st linked list is 3 and for the 2nd linked list is 2.

**2. Approach**

Reader, let's discuss the approach for this problem. We say we have 2 linked lists a-b-c-d-e-f-g an h-i-e-f-g with t1 and t2 pointers at their heads respectively. Now we calculate an absolute value, "delta" which is the difference between the sizes of the two linked lists. In figure 2, size of the first linked list=7 and size of second linked list=5. Therefore, delta= |7-5|=2. We move the pointer to a larger sized linked list by "delta" number of steps. Hence t1 i.e. the pointer of the larger sized list is moved by "2" steps and reaches "c".

****

Next, we keep moving both the pointers simultaneously till they are not equal to each other. Whenever they get equal, we return the value at either of those pointers.

****

This approach works because after moving the pointer of the larger list by "delta" number of steps, the pointers t1 and t2 lie at equal distance from the point of intersection making it easy for us to find it. Now that we have seen the algorithm for this problem, we try to code it .

ConsoleJava

public static int findIntersection(LinkedList one, LinkedList two) {

Node t1 = one.head;

//1

Node t2 = two.head;

int delta = Math.abs(one.size - two.size); //2

if (one.size > two.size) { //3

for (int i = 0; i < delta; i++) {

t1 = t1.next;

}

} else if (two.size > one.size) {

for (int i = 0; i < delta; i++) {

t2 = t2.next;

}

}

while (t1 != t2) { //4

t1 = t1.next;

t2 = t2.next;

}

return t1.data; //5

}

}

**3. CODE DISCUSSION**

1• We initially point t1 and t2 at the heads of first and second linked lists respectively. 2• Then we find out the absolute value of difference b/w the two sizes of the lists. 3• We check which linked list has the largest size and move it's pointer by the "delta" number of steps. 4• Now that the size of both arms from the point of intersection is the same, we move both the pointer simultaneously till they reach the same position i.e. the point of intersection. 5• We return the data of that point of intersection.

**4. Analysis**

Time Complexity

O(n) Since a "while loop" is used in the program, therefore the time complexity is linear.

Space Complexity

O(1) Since no extra space is used in the program, therefore the space complexity is constant. We highly suggest you watch the solution video for a better understanding of the discussion.

[**Solution Video**](https://youtu.be/B4aqNarb0QQ)

With this we conclude our question "Intersection Point of Linked Lists" as well as the topic of "Linked Lists". From the next lecture we will learn about "Generic Trees". For now, Goodbye.