برای این سؤال ۲ Stack نیاز داریم تا با یک بار پیمایش این list ها تمام اعضا در Stack بریزیم - پس برای این کار اگر عملیات Stack.push() را با cost=c در نظر بگیریم تا به اینجا nc+mc زمان صرف کرده ایم.

حال ما دو Stack داریم که هرکدام مجموعه ای از پوینترها وارد به Node های هر لیست هستند - حال شروع به pop کردن می کنیم و مقادیر را با هم مقایسه می کنیم. در اولین باری که نابرابری رخ بدهد، Node قبلی شروع اشتراک دو لیست بوده -

پس پیاده سازی این روش به صورت الگوریتم زیر است:

```
FindFirs Common (list1, list2):
    stack1, stack2 = initialize(list1, list2)
    tmp1, tmp2 = stack1.pop(), stack2.pop()
    while( stack1.isEmpty() != true and stack2.isEmpty() != true)
        if tmp1 == tmp2:
            holder = tmp1
            tmp1 = stack1.pop()
            tmp2 = stack2.pop()
        else:
            retur holder
```

| | |
|---|---|
| $C_1$ | |
| $C_2$ | |
| $mC_3$ or $nC_3$ | |
| $m-1 \times C_4$ | |
| $4C_5$ | |
| $4C_6$ | |
| $4C_7$ | |
| $C_8$ | |

فرض کرده ایم که Stack.pop() علاوه بر حذف Key آن را به ما برمی گرداند.

در این بخش نیز در بدترین حالت m یا n بار پیمایش صورت گرفته

پس برای کل الگوریتم پیچیدگی زمانی ما برابر است با

$nc + mc + mc' + c''$ یا $nc + mc + nc' + c''$

که در هر دو حالت از $O(m+n)$ هستند.

الگوریتم Initialize نیز به صورت زیر است:

```
Initialize (list1, list2):
    tmp = list1.head                          c
    while (tmp.next != null)                  n×c
        stack1.push (tmp)                     (n-1)c2
        tmp = tmp.next                        (n-1)c3
    stack1.push(tmp)                          c4
    tmp = list2.head                          c5
    while (tmp.next != null)                  m×c6
        stack2.push (tmp)                     (m-1)c7
        tmp = tmp.next                        (m-1)c8
    stack2.push (tmp)                         c9
    return stack1, stack2                     c10
```

در این الگوریتم تمام Node های هر link_list را درون یک Stack ریخته تا در ادامه با مقایسه هر دو به مرحله، اولین Node مشترک را بیابیم.

پیچیدگی زمانی برای این قسمت به طور خلاصه داریم:

$mc + nc + c_1$

```go
func initialize(list1, list2 *LinkedList) (*Stack, *Stack) {
  stack1 := CreateStack()
  stack2 := CreateStack()
  tmp := list1.head
  for tmp.next != nil {
    stack1.Push(tmp)
    tmp = tmp.next
  }
  stack1.Push(tmp)
  tmp = list2.head
  for tmp.next != nil {
    stack2.Push(tmp)
    tmp = tmp.next
  }
  stack2.Push(tmp)
  return stack1, stack2
}
func firstCommon(list1, list2 *LinkedList) *Node {
  stack1, stack2 := initialize(list1, list2)
  tmp1 :=stack1.Pop()
  tmp2 :=stack2.Pop()
  var holder *Node
  for ;!stack1.IsEmpty() && !stack2.IsEmpty();{
    if tmp1.key == tmp2.key{
      holder = tmp1
      tmp1=stack1.Pop()
      tmp2=stack2.Pop()
    }else{
      return holder
    }
  }
  return nil
}
func main() {
  list1:=CreateLinkedList();list2:=CreateLinkedList();
  list1.Add(1);list1.Add(2);list1.Add(3);list1.Add(4);list1.Add(5);
  list2.Add(9);list2.Add(6);list2.Add(3);list2.Add(4);list2.Add(5);
  fmt.Println("linked-list 1 :");list1.Display();
  fmt.Println("linked-list 2 :");list2.Display();
  fmt.Println("first common node is : ");fmt.Println(firstCommon(list1,list2).key);
}
```

```
armin@armin-Aspire-A715-71G: ~/Desktop/c++
(base) armin@armin-Aspire-A715-71G:~/Desktop/c++$ go run f
linked-list 1 :
1 2 3 4 5
linked-list 2 :
9 6 3 4 5
first common node is :
3
(base) armin@armin-Aspire-A715-71G:~/Desktop/c++$
```