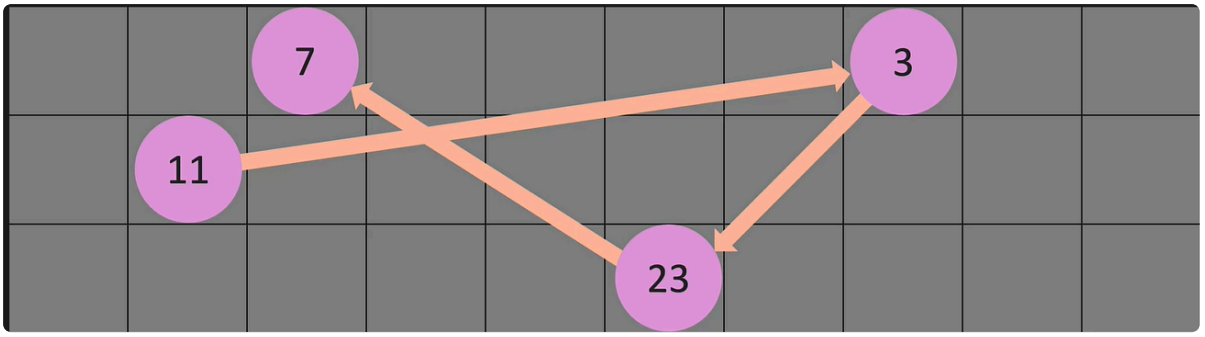
- An array (or a list) is a contiguous place in memory, that is, all items are next to each other in memory. They have indexes that you can access in O(1), because each of them have a specific address in memory.
- But the same doesn't applies to Linked Lists. They are not contiguous and they don't have indexes.
- Instead, they have several nodes spread all over the place.
- Each node point to the next one, and the last node points to None.
- They also have a variable called head, which points to the first node, and another one called tail, which points to the last node.
- 

Big O

- When prepending (append a variable as the first none) or popping the first node is O(1)
- But normal pop (last item) is O(n), differently from lists
- And lookup by index is O(n), while lists are O(1), because if you have to go over every node in order to discover their indexes

Linked Lists under the hood

- They kind of work like a set of nested dictionaries, with keys value and next

```
head : {
    "value": 21,
    "next": {
        "value": 14,
        "next": {
            "value": 7,
            "next": None
        }
    }
}
```

- It's not exactly like this, but you can think this way to help you understand