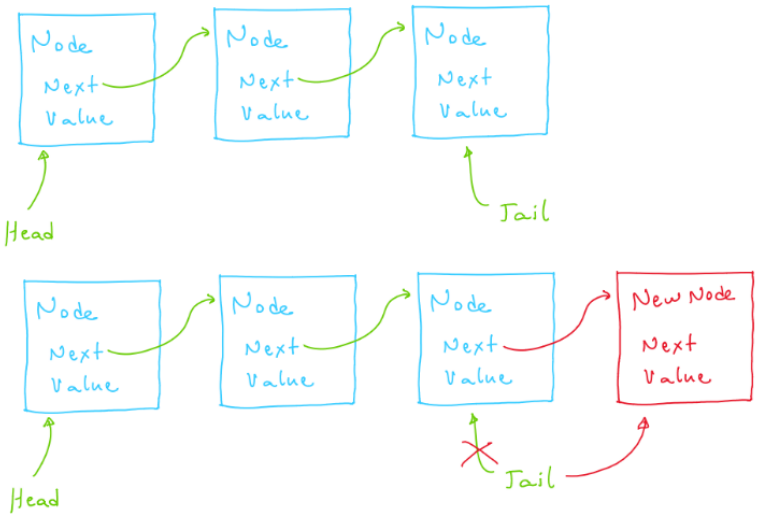
Problem 6: Union and Intersection

The problem was addressed by using the code template given in the problem statement with one change into the Linked List ***append()*** method implementation. It was included a reference to the list tail end to make it possible to insert new nodes with a ***O(1)*** time complexity cost.



The ***union()*** and ***intersection()*** functions were implemented with the same 3-step strategy:

- The algorithm firstly traverses both linked lists and store their content into Python sets.
- Then it uses the union/intersection operations over the Python sets generating a resulting one.
- At last, the algorithm traverses the resulting Python set to generate a corresponding linked list to be returned.

The algorithm input ***n*** will correspond to the sum of ***s*** and ***t*** items stored in the two linked lists used as argument.

The first algorithm step has a time complexity of ***O(n)*** since it needs to traverse both linked lists. The second step has ***O(len(s)+len(t))*** complexity for union and ***O(min(len(s), len(t))*** complexity for intersection based on the Python implementation ***[1]***. The third part has a time complexity of ***O(n)*** for union operation in the worst-case scenario of two disjoint sets and ***O(min(len(s), len(t))*** complexity for intersection operation in the worst-case scenario of one of the sets being a subset of the other. The complete algorithm will result in a time complexity of ***O(n)***.

Regarding to space, the algorithm uses an auxiliary data structure of two Python sets with the same number of elements of the input plus some other minor data structures which will result in a ***O(n)*** space complexity.

References

[1] Python time complexity - https://wiki.python.org/moin/TimeComplexity, accessed in 06/06/2021.