ADT & Algorithms
Lists.

Dr. NOURI Nabil

November 2, 2024

1. Linear List Exercises

1. **Initialization:** Define a linear list `L` and initialize it with 5 elements.
2. **Insert at End:** Write a function `insert_end(L,x)` to insert element `x` at the end of `L`.
3. **Delete by Value:** Implement `delete(L,x)` to remove the first occurrence of `x` from `L`.
4. **Search:** Define `search(L, x)` to return the index of the first occurrence of `x` in `L` or `-1` if not found.
5. **Count Occurrences:** Implement `count(L,x)` to count how many times `x` appears in `L`.
6. **Reverse List:** Write a function `reverse(L)` to reverse the elements of `L`.
7. **Sorting:** Define `sort(L)` to sort the elements of `L` in ascending order using any sorting algorithm.
8. **Merge Two Lists:** Let `L1` and `L2` be two sorted linear lists. Define `merge(L1,L2)` to produce a new sorted list containing elements from both lists.
9. **Remove Duplicates:** Implement `remove_duplicates(L)` to remove duplicate elements from `L`.
10. 1 **Kth Largest Element:** Write a function `kth_largest(L,k)` that returns the `k`-th largest element in `L` without sorting.

---

2. Doubly Linked List Exercises

1. **Initialization:** Define a doubly linked list `D` and add `3` nodes with arbitrary values.
2. **Insert at Beginning:** Write a function `insert_begin(D,x)` to insert element `x` at the beginning of `D`.
3. **Delete Node by Position:** Implement `delete(D,p)` to remove the node at position `p`.
4. **Traverse Forward:** Define `traverse_forward(D)` to print elements of `D` from head to tail.
5. **Traverse Backward:** Define `traverse_backward(D)` to print elements of `D` from tail to head.
6. **Find Middle Node:** Write a function `find_middle(D)` to return the middle node of `D`.
7. **Detect Cycle:** Implement `has_cycle(D)` to return truetruetrue if `D` contains a cycle, otherwise `false`.
8. **Copy List:** Define `copy(D)` to create and return a new doubly linked list that is a copy of `D`.
9. **Kth Node from End:** Write a function `kth_from_end(D,k)` to find the `k`-th node from the end of `D`.