DoublyLinkedList - tidskomplexitet

Skemaer – til sammenligning

En DoublyLinkedList består af noder, hvor hver node peger både på den forrige og den næste node. Det gør det muligt at bevæge sig i begge retninger og gør visse operationer mere fleksible end i en SinglyLinkedList.

Pros:

- Indsættelse/fjernelse i starten og slutningen: O(1) – Hurtigt, da referencer nemt kan opdateres.
- Dobbelt traversal: Mulighed for at bevæge sig både frem og tilbage i listen.
- Effektiv fjernelse: O(1), hvis man allerede har reference til noden (ingen traversal nødvendig).

Cons:

- Adgang via indeks: O(n) – Skal stadig traversere listen for at finde et element.
- Større hukommelsesforbrug: Hver node kræver to referencer (forrige og næste).
- Mere kompleks implementering: Flere referencer skal opdateres korrekt ved ændringer.

DoublyLinkedList

Læs et element[1]	første	sidste	midterste	i'te	næste[2]
	$O(1)$	$O(1)$	$O(n)$	$O(n)$	O(n)
Find element [3]	eksisterer *usorteret liste*	eksisterer *sorteret liste*	eksisterer ikke *usorteret liste*	eksisterer ikke *sorteret liste*	
	$O(n)/O(1)$	$O(n)/O(1)$	$O(n)/O(1)$	$O(n)/O(1)$	
Indsæt nyt element	i starten	i slutningen	i midten		
	$O(1)$	$O(1)$	$O(n)$		
Fjern element	første	Sidste	i'te		
	$O(1)$	$O(1)$	$O(n)$		
Byt om på to elementer	første og sidste	første og i'te	sidste og i'te	i'te og j'te	
	$O(1)$	$O(n)$	$O(n)$	$O(n)$	

[1] At læse et element er som regel det samme som at skrive nyt indhold i et eksisterende element

[2] Hvis vi allerede har fat i ét element i en datastruktur, kan vi måske læse det "næste" hurtigere end i+1'te

[3] Find et element med en bestemt værdi – alt efter om vi ved at listen er sorteret eller ej, og om elementet findes eller ej.