Stack – tidskomplexitet

Skemaer – til sammenligning

Udfyld et skema som det herunder med Big O estimater for tidskompleksiteter for alle de datastrukturer du støder på. Skriv også noter til dig selv om nogle af de antagelser du gør dig – for eksempel hvis en operation nogle gange tager $O(1)$, men andre gange $O(n)$ …

Fjern denne generelle tekst og erstat den med en mere sigende forklaring af den pågældende datastruktur, og gem en pdf udgave af skemaet i mappen med din implementation i din portfolio.

Bemærk at skemaerne altid anvender *n* som det samlede antal af elementer i strukturen, og *i* som et enkelt, vilkårligt, element. Og vi kan bruge O(i) for at vise at en operation tager det antal iterationer som det index vi beder om – men i princippet burde det stadig være O(n) fordi vi altid angiver worst case.

Data i datastrukturen omtales altid som elementer, mens de ”databærende enheder” kaldes nodes. Et array og en array list har for eksempel ikke nodes, men en linked list eller et tree har altid, og nogle operationer er markant hurtigere hvis man allerede har adgang til en nabo-node.

Datastrukturnavn

Læs et element[1]	første	sidste	midterste	i'te	tidligere[2]
	$0(1)$	$O(n)$	$O(n)$	$O(n)$	$0(1)$
Find element [3]	eksisterer *usorteret liste*	eksisterer *sorteret liste*	eksisterer ikke *usorteret liste*	eksisterer ikke *sorteret liste*	
	$O(n)$	x	$O(n)$	$O(n)$	
Indsæt nyt element	i starten	i slutningen	i midten	efter node	før node
	$O(1)$ - *push*	$O(n)$	$O(n)$	$O(n)$	$O(n)$
Fjern element	første	sidste	i'te	efter node	før node
	$O(1)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
Byt om på to elementer	første og sidste	første og i’te	sidste og i’te	i’te og j’te	nodes
	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$

[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.

Noter:
Denne stack er implementeret som en singly linked list.

Det betyder, at vi altid har hurtig adgang til toppen (head), men ingen direkte adgang til elementer længere nede i listen.

Derfor er push() og pop() altid O(1), mens alle operationer der kræver traversal ned ad listen bliver O(n).

Fordi en stack kun rigtig understøtter push og pop på toppen, er alle andre indsætninger, sletninger eller læsninger af i’te element O(n).