2. Gyakorlat

Algoritmusok és adatszerkezetek 1.

Aszimptotikus függvények

- $O(g(n)) = \{f(n) : \exists\ c > 0$ és $n_0 \geq 0$, hogy $\forall\ n > n_0$ esetén $f(n) \leq c*g(n)\}$
  - aszimptotikus felső korlát
- $\Omega(g(n)) = \{f(n) : \exists\ c > 0$ és $n_0 \geq 0$, hogy $\forall\ n > n_0$ esetén $f(n) \geq c*g(n)\}$
  - aszimptotikus alsó korlát
- $\Theta(g(n)) = \{f(n) : \exists\ c_1, c_2 > 0$ és $n_0 \geq 0$, hogy $\forall\ n > n_0$ esetén $c_1*g(n) \geq f(n) \geq c_2*g(n)\}$
  - aszimptotikusan éles korlát

Mit jelent ez?

- O(g): minden olyan függvény, amely nem nő jobban g-nél
  - "Maximum konstansszorosa"
    - egy idő után
  - $O(n^2) = \{n, \log n, 2n, 3, n^2, 10n^2, \ldots\}$
  - Végtelen halmaz
  - Jelölés helyesen $f \in O(g)$, de gyakran úgy írjuk, hogy $f = O(g)$

Továbbiak

- Ω: legalább konstansszorosa
- Θ: pontosan konstansszorosa
    - $\Theta(n^k) = \{\Sigma_{i=1..k} a^i n^i\}$ //az állítás nem az, hogy ezeken kívül nincs más elem a $\Theta(n^k)$-ban, hanem hogy minden k-ad fokú polinom benne van.
- Ω-t nem gyakran használjuk
- O a legelterjedtebb, de igen gyakran Θ-t értünk alatta
    - Pl. Dijkstra algoritmus műveletigénye $O(n\log n + e)$ prioritásos sor használata esetén, az valójában $\Theta(n\log n + e)$

Igaz-e?

$n^2-1 = O(n^2)$

$n^2+1 = \Omega(n^2)$

$200n = O(n^2)$

$n^2+1 = O(n^2)$

$2^{n+1} = O(2^n)$

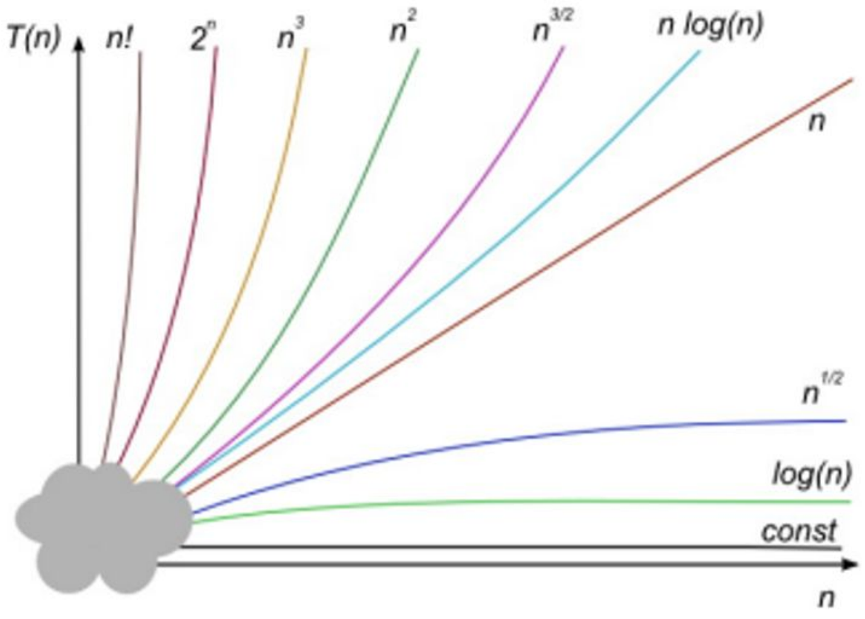
$2n^3+5n^2+n+10 = O(n^3)$

Igaz-e?

n^2-1 = O(n^2) - igen c = 1, n_0= 1
n^2+1 = Ω(n^2) - igen c = 1, n_0 = 1
200n = O(n^2) - igen c = 1, n_0 = 200 vagy c = 200, n_0 = 1
n^2+1 = O(n^2) - igen c = 2, n_0= 2, hiszen $2n^2 > n^2+1$
2^{n+1} = O(2^n) - igen, c = 2, n_0 = 1, hiszen $2^{n+1} \geq 2*2^n$
$2n^3+5n^2+n+10$ = O(n^3) - igen, c = 18 (együtthatók: $10n^3 \geq 10$, n3 ≥ n, stb.)

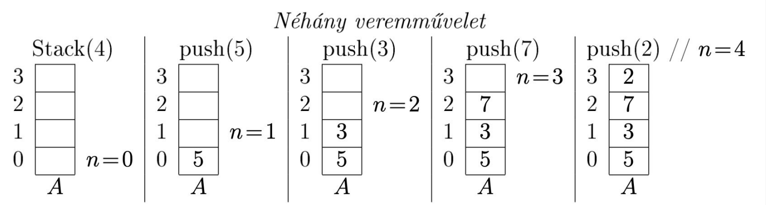
Mire használjuk ezeket?

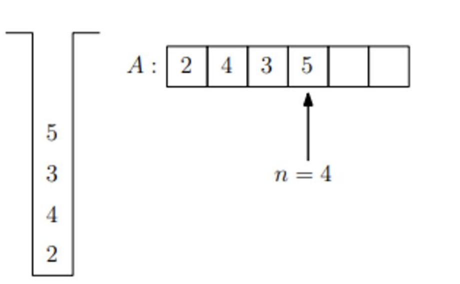
- Lépésszám komplexitás meghatározáshoz / osztályozáshoz
  - Szakirodalom: “Time complexity”
  - Tipikus osztályok:
    - c: két érték megcserélése
    - logn: logaritmikus keresés - nomen est omen
    - n: linker, maxker
    - nlogn: rendezési algoritmusok, Dijksta
    - n^2 : Bellmann-Ford algoritmus
- Nem csak lépésszám komplexitást figyelhetünk, hanem memória foglalást is.
  - Memória, Kiegészítő memória (“Space complexity”, “Auxiliary space complexity”)


T(n)
n!
2^n
n^3
n^2
n^3/2
n log(n)
n
n^1/2
log(n)
const
n

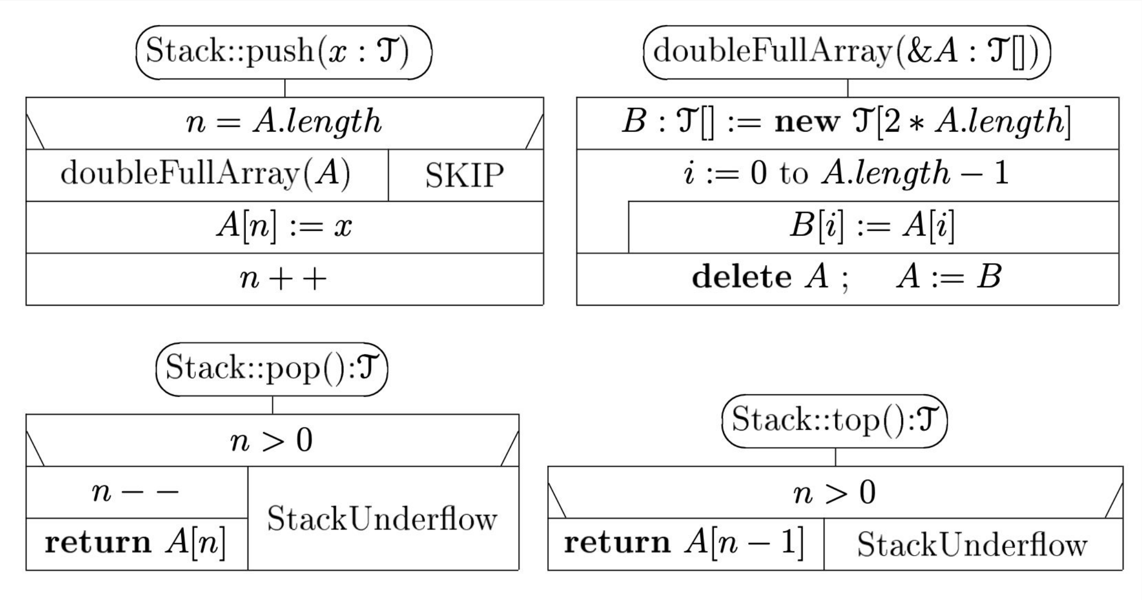
Verem adattípus - tömbös ábrázolás

Stack
$-\ A : \mathcal{T}[]$ // $\mathcal{T}$ is some known type ; $A.length$ is the physical $-$ constant $m0 : \mathbb{N}_+ := 16$ // size of the stack, its default is m0. $-\ n : \mathbb{N}$ // $n \in 0..A.length$ is the actual size of the stack
$+$ Stack($m : \mathbb{N}_+ := m0$) $\{A :=$ **new** $\mathcal{T}[m]$; $n := 0\}$ // create empty stack $+ \sim$ Stack() { **delete** A } $+$ push($x : \mathcal{T}$) // push x onto the top of the stack $+$ pop() $: \mathcal{T}$ // remove and return the top element of the stack $+$ top() $: \mathcal{T}$ // return the top element of the stack $+$ isEmpty() $: \mathbb{B}$ $\{$**return** $n = 0\}$ $+$ setEmpty() $\{n := 0\}$ // reinitialize the stack





Műveletek implementációja - tömbös ábrázolás



Verem adattípus - Láncolt ábrázolás

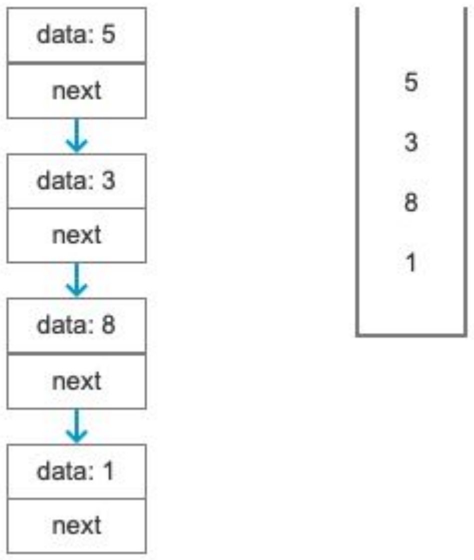
Stack

top: E1*

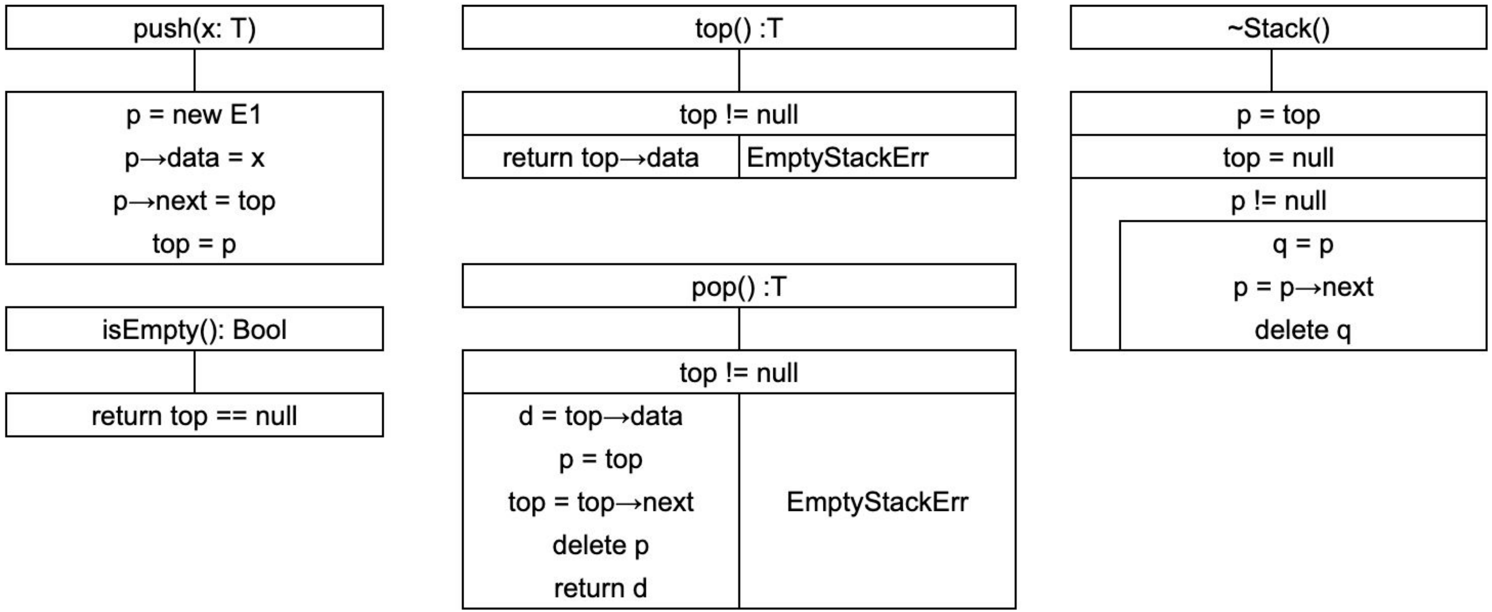
Stack() { top:=null }
~Stack()
push(x: T)
pop(): T
top(): T
isEmpty(): Bool
setEmpty()

E1

data: T
next: E1*



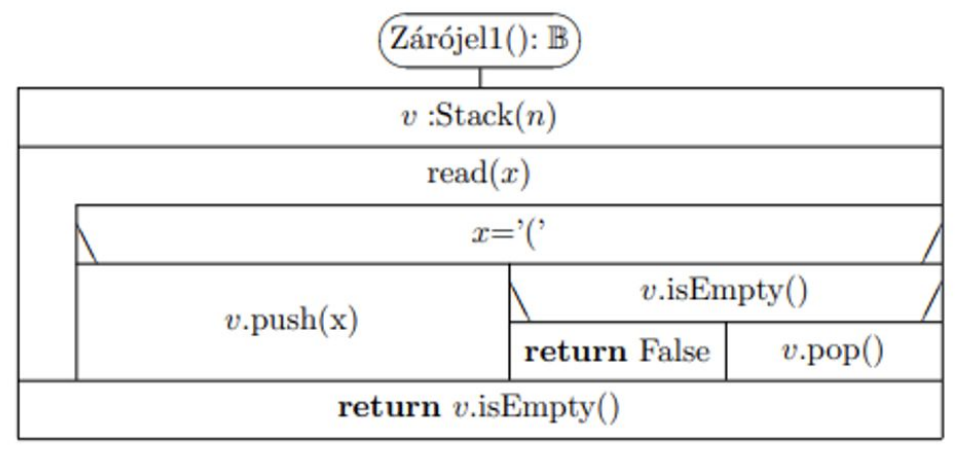
Műveletek implementációja - láncolt ábrázolás



Feladat - Helyes zárójelezés #1

Adott egy zárójelekből álló, legfeljebb n hosszú karaktersorozat a bemeneten. Olvassuk be és döntsük el róla, hogy helyes zárójelezést határoz-e meg.

Feladat - Helyes zárójelezés #1



Feladat - Helyes zárójelezés #2

Adott egy zárójelekből álló, legfeljebb n hosszú karaktersorozat a bemeneten. Olvassuk be és döntsük el róla, hogy helyes zárójelezést határoz-e meg. Írjuk ki az összetartozó zárójelpárok indexeit.

Feladat - Helyes zárójelezés #2