RAČ 2 VAJE 1

15.2.2023

gremo na repozitorij, damo fork

ko bo treba oddat damo pull request

↗

damo v glavni repozitorij

---

v navodilu točno napišemo kako je treba pognati datoteko, če bo treba dati input ali ne

---

VAJE 1

NALOGA 1

$f = O(g)$

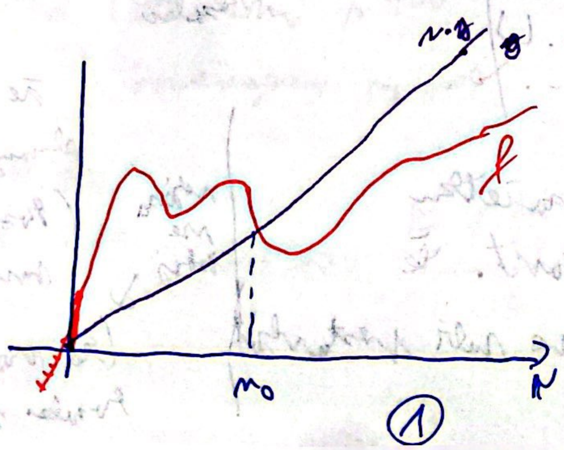
Kaj pomeni $f \in O(g)$ za neki funkciji $f, g : \mathbb{N} \to \mathbb{R}^+$?

$f \in O(g) \iff \exists c\ \exists n_0 : \forall n > n_0\ \ f(n) \le c \cdot g(n)$

"$f \le g$"

$n \in O(n)$

velikost vhodnih podatkov



①

$n \in O(n^2)$

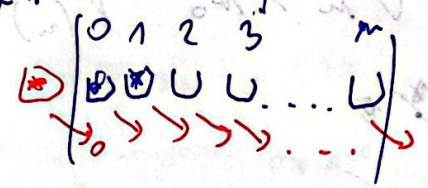
$\log(n) \in O(n)$

$n^2 \notin O(n)$

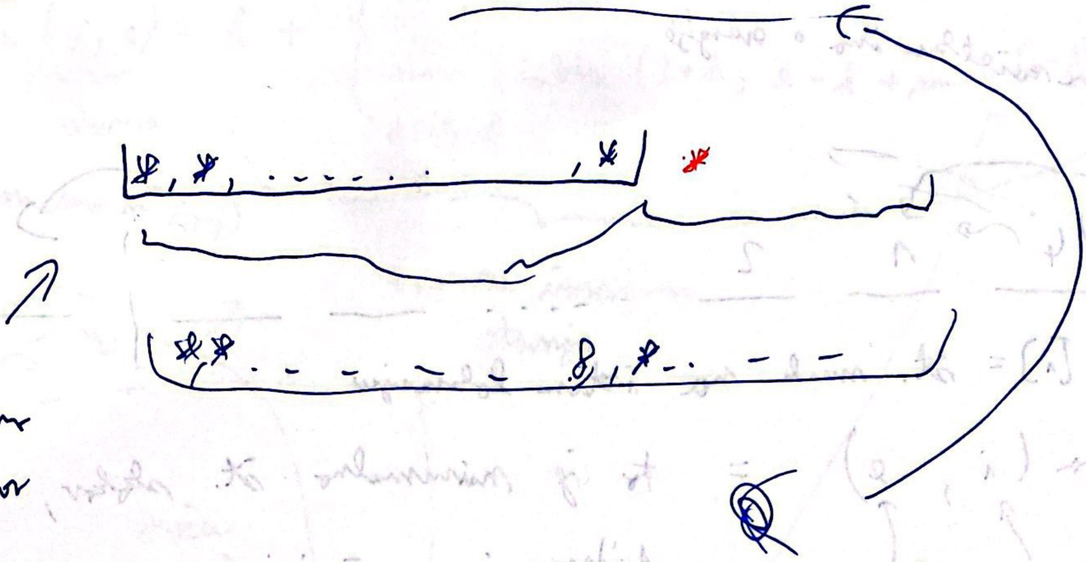
$f \in \Omega(g)$ "$f \geq g$"

$f \in \Theta(g)$ "$f = g$"

	dodaj(n)	dodaj(i)	dodaj(0)	dostop(i)	"x in"	briši(0)	briši(i)	briši(n)
seznam	$O(1)$	$O(n)$	$O(n)$	$O(1)$	$O(n)$	$O(n)$	$O(n)$	$O(1)$
slovar / množ.	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$
verižni seznam	$O(1)$	$O(n)$	$O(1)$ $O(n)$	$O(1)$ $O(n)$	$O(n)$	$O(n)$	$O(1)$ $O(n)$	$O(1)$ $O(n)$



Python v tem seznamu "rezervira" nekaj prostora



če želimo dodati nov element, moramo ga prekopirat, in da poveča prostor

v tem primeru v seznamu dodaj(n) ni O(1) časovna zahtevnost ni O(1) ampak O(n)

Ampak se splača se samo povečati za 1, saj vsakič dodaj, ker vse kopira elemente ni v redu

Dodali bi N elementov v seznam

(začnemo s 100 povečujemo za 100)

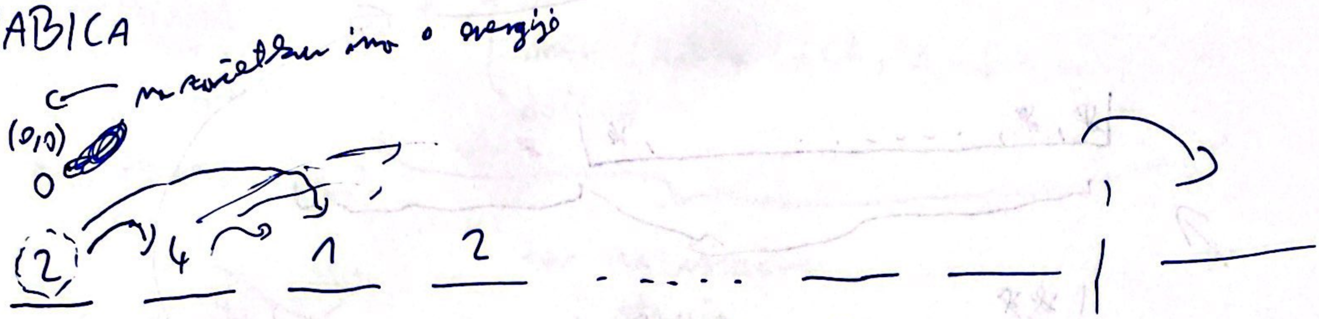
toda je več splača samo za 100

$$N + \sum_{i=1}^{N/100} i \cdot 100 = N + 100\,\frac{\frac{N}{100}\left(\frac{N}{100}+1\right)}{2} = O(N^2)$$

NALOGA 2:

TOMO – DINAMIČNO PROGRAMIRANJE

1) ŽABICA



muhe[i] = št. muh na i-tem lokvanju

žabica(i, e) = to je minimalno št. skokov, da pridemo iz močvirja, če se nahajamo na i-tem lokvanju in imamo e energije

(i: indeks na lokvanju kjer smo; e: količina energije)

Sedaj napišemo neko rekurzivno zvezo, upoštevamo pa nov indeks in novo količino energije (kam vse moramo skočiti / koliko velika mora biti):

$i < n$

$$\text{žabica}(i, e) = 1 + \begin{cases} 0 & ; i + e > n \\ \text{žabica}(i+j,\ e - j + \text{muhe}[i+j]) \end{cases}$$

$$j = \arg\max_{d \in (1, \dots, e)} (\text{muhe}[i+d])$$

(d: dolžina skoka)

(0: če v naslednjem skoku skočimo na breg; 1: če smo s tem skokom že skočili ven)

(skočimo na lokvanj z največjo energijo)

2. NAČIN

↓

BOLJŠI

$(n \times n)$

$$\text{čokolada}(i, e) = 1 + \begin{cases} 0 & ; \; i+e > n \\ \min_{d \in (1,..,e)} (\text{čokolada}(i+d, e-d + \text{mrvice}[i+d])) \end{cases}$$

stanja

(torej imamo n^2 stanj)

čas za izračun stanja

možen

pogledamo vsak skok in vidimo kateri se nam najbolj izplača

ROBNI POGOJI:

$$\text{čokolada}(i, e) = 0 \quad i > n$$

$$\text{čokolada}(i, e) = 1 \quad e > n - i$$

ta bi n rezultat največ

Rešitev:

$$\text{čokolada}(0, \text{mrvice}[0])$$

toliko smo dobili tako da smo rekli, da smo na začetku že pojedli mrvo

rodi rekurzivne računamo

$(1, *)$ $(2, *)$ ... $(\text{mrvice}[0], *)$

ČASOVNA ZAHTEVNOST:

Izračunati moramo vsa stanja $\Rightarrow O(n^2) \cdot O(n) = O(n^3)$

imamo min $\in O(e)$, torej dvema kar $O(n)$

koliko časa rabimo da izračunamo eno stanje

št. stanj

(stanja so vsi možni argumenti; ki jih sprejme funkcija,
$i \in$ indeks od 1 do n, vsak indeks ima še ↓

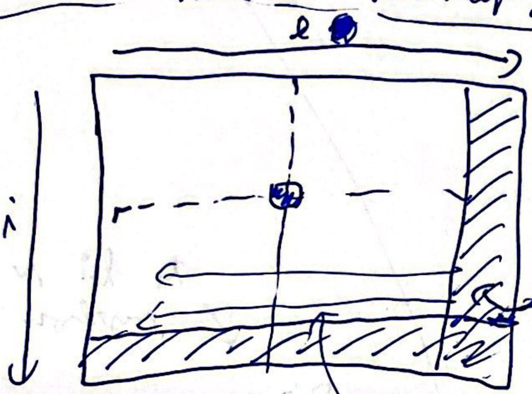
rez energije (torej v n × n redi)

Izbiro največjo vrednost up in rekurzivno

pripravimo si novo tabelo velikosti n × n

(vsebina vrstic velikosti n)

potem se na pravilen način napolnimo vso tabelo



zadnji stolpec vedno izračunat

tako napolnimo vso tabelo

od spodaj

gor postavimo

MEMOIZACIJA:

```
from functools import lru_cache

@lru_cache(maxsize=None)
def funkcija(i, e):
```