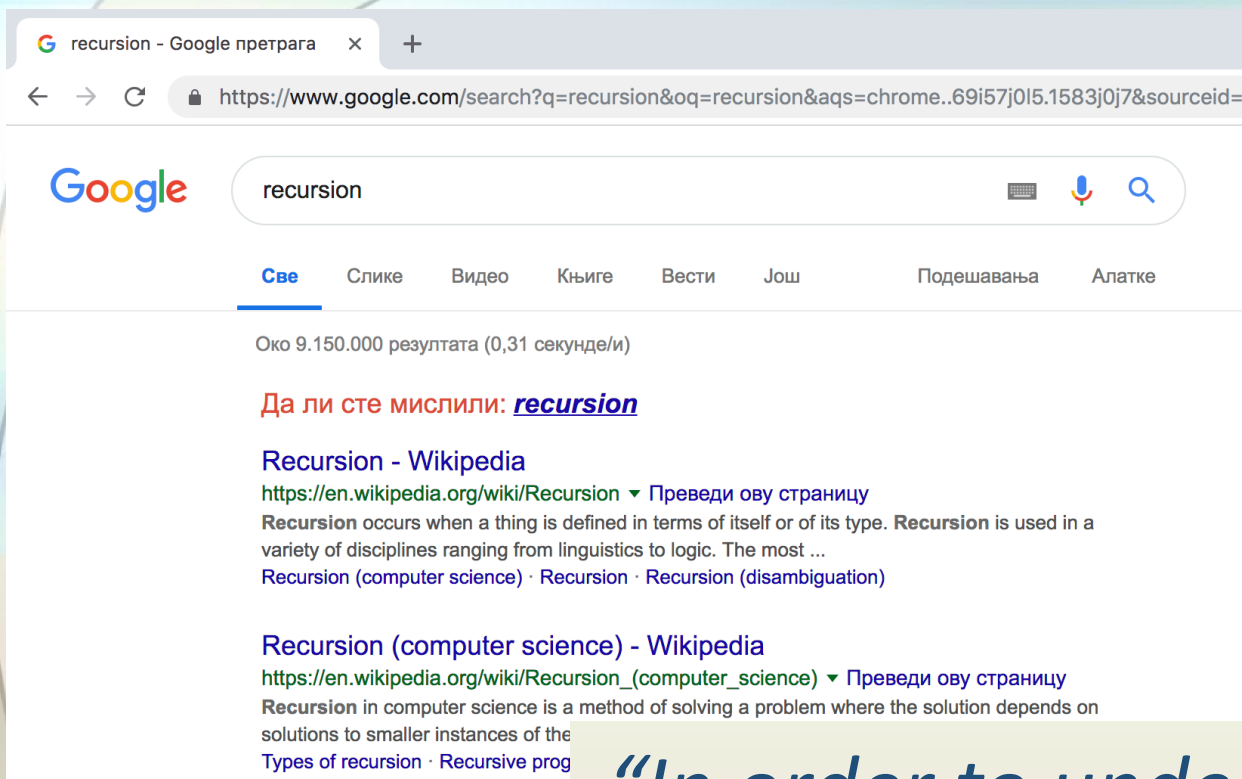


# Algoritmi i strukture podataka

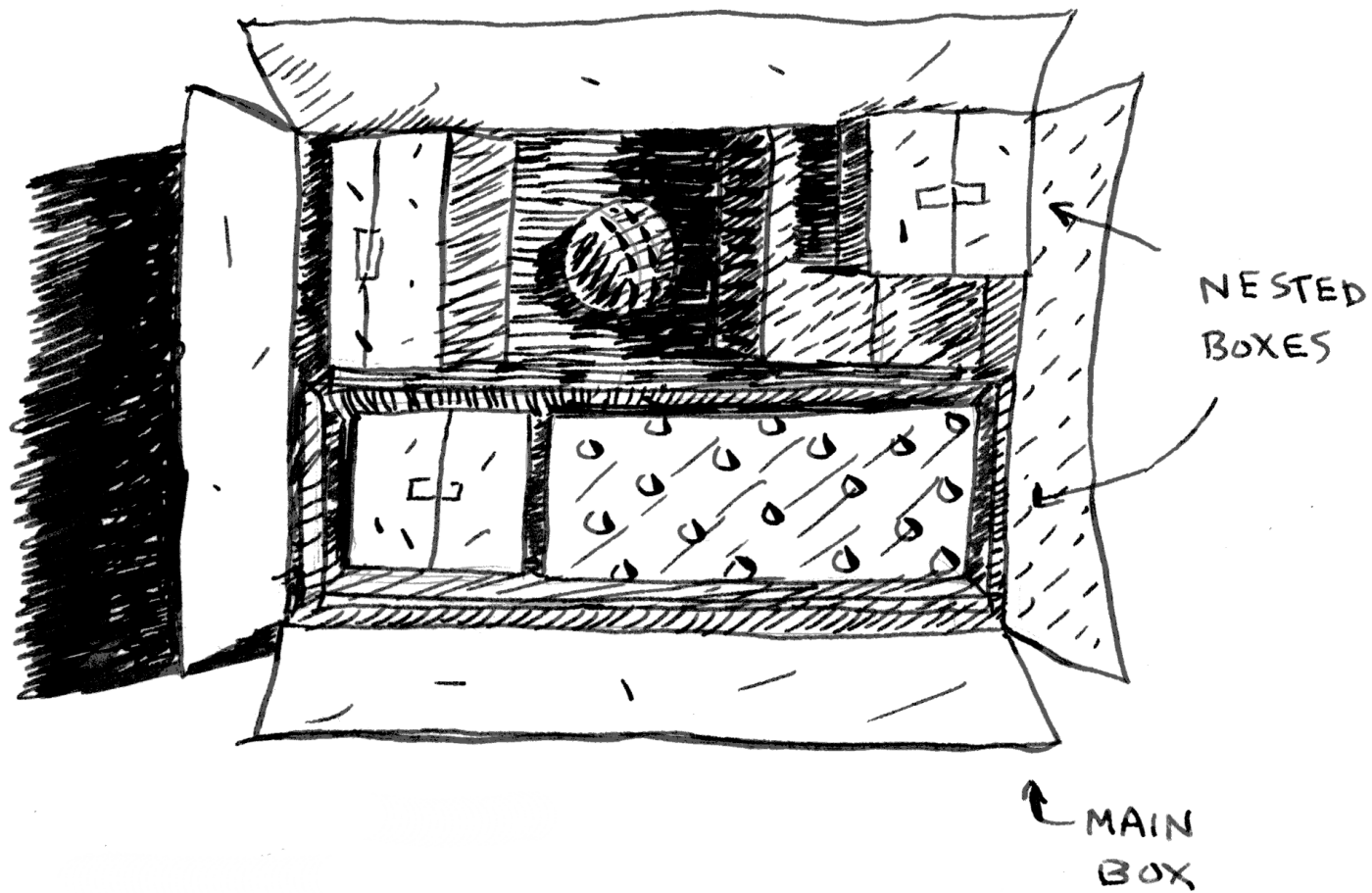
02 Rekurzija

Katedra za informatiku, Fakultet tehničkih nauka, Novi Sad

2021



*“In order to understand recursion,  
one must first understand recursion.”*



Preuzeto sa: <https://medium.freecodecamp.org/how-recursion-works-explained-with-flowcharts-and-a-video-de61f40cb7f9>

# Definicija

- Funkcija poziva samu sebe
- Rekurzivna funkcija se sastoji od:
  - Osnovnih tj. baznih slučajeva
    - Najčešće se surećemo sa jednim osnovnim slučajem, ali ih može biti i više
    - Nema rekurzivnih poziva
    - Moraju postojati
    - Osiguravaju završetak postupka
  - Rekurzivnih poziva
    - Osnovni cilj: napredovanje ka osnovnom slučaju

# Tipovi

- Linearna
  - 1 rekurzivni poziv
  - Repna rekurzija
    - Poslednji korak je rekurzivni poziv
- Binarna
  - 2 rekurzivna poziva
- Višestruka

# Rekurzija - poređenje

Dobre strane	Loše strane
Kraće	Teža za razumevanje kod nekih algoritama
Jednostavnije	Veće zauzeće memorije (stack overflow)
Čitljivije	Sporije od iterativnih postupaka (zbog mnogobrojnih poziva funkcije)
Elegantnije	Prekidanje postupka u toku izvršavanja
Neki algoritmi i strukture podataka se rekurzivno definišu pa ih je lakše rekurzivno implementirati	Zavisnost od broja poziva

# Primer (1/7)

- Zadatak: pronaći sumu elemenata liste rekurzivnim putem
- Način rešavanja:
- 1) Odredimo kada je rešavanje ovog problema trivijalno - to će biti naš osnovni slučaj
  - Sumu liste je trivijalno izračunati ako je broj elemenata 1
  - U tom slučaju, suma odgovara tom jednom elementu
  - U kodu bismo pisali:

```
if len(our_list) == 1:  
    return our_list[0]
```
  - Napomena: Često se može desiti da se bazni slučaj preskoči i tada rekurzija postaje beskonačna. Da bismo to rešili, bazni slučaj moramo nekad proširiti i na nevalidne slučajeve (znak == zameniti sa <=).



## Primer (2/7)

- 2) Zamislimo da već imamo implementiranu funkciju za određivanje sume.
- 3) Odredimo na koji način ćemo postići napredovanje prema osnovnom slučaju (tako što ćemo rekurzivnom pozivu proslediti vrednost manju od parametra, kraću listu i sl.)
- U našem primeru, ukoliko imamo funkciju koja sumira elemente liste, možemo joj proslediti sve elemente osim jednog (prvog ili poslednjeg). Suma konačne liste će odgovarati zbiru tog jednog elementa i sume ostatka liste.

```
rest_sum = find_sum(our_list[1:])  
return our_list[0] + rest_sum
```



## Primer (3/7)

- Ono što se ne vidi na prvi pogled je da se ovo parče koda može preformulisati da rekurzivni poziv postane poslednji poziv u funkciji

```
return our_list[0] + find_sum(our_list[1:])
```

- Sada je jasno da ovde imamo slučaj repne rekurzije => može se transformisati u iterativni postupak
- Drugi pristup: Umesto sabiranja prvog elementa sa preostalim, možemo sabrati sume dve polovine kolekcije

```
n = len(our_list)
return find_sum(our_list[:n//2]) + find_sum(our_list[n//2:])
```

- Gledamo da dva dela liste imaju otprilike jednak broj elemenata
- Važno je da neki element ne preskočimo

## Primer (4/7)

- Celo rešenje:

```
def find_sum(our_list):  
    n = len(our_list)  
    if n <= 1:  
        return our_list[0]  
    return our_list[0] + find_sum(our_list[1:])
```

- Ili:

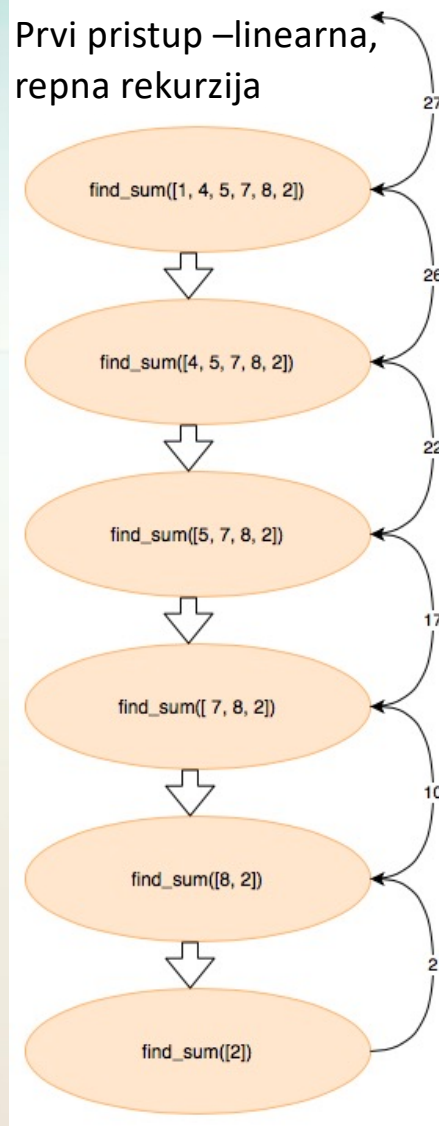
```
def find_sum(our_list):  
    n = len(our_list)  
    if n <= 1:  
        return our_list[0]  
    return find_sum(our_list[:n//2]) + find_sum(our_list[n//2:])
```

- Koji od dva pristupa je efikasniji?

## Primer (5/7)

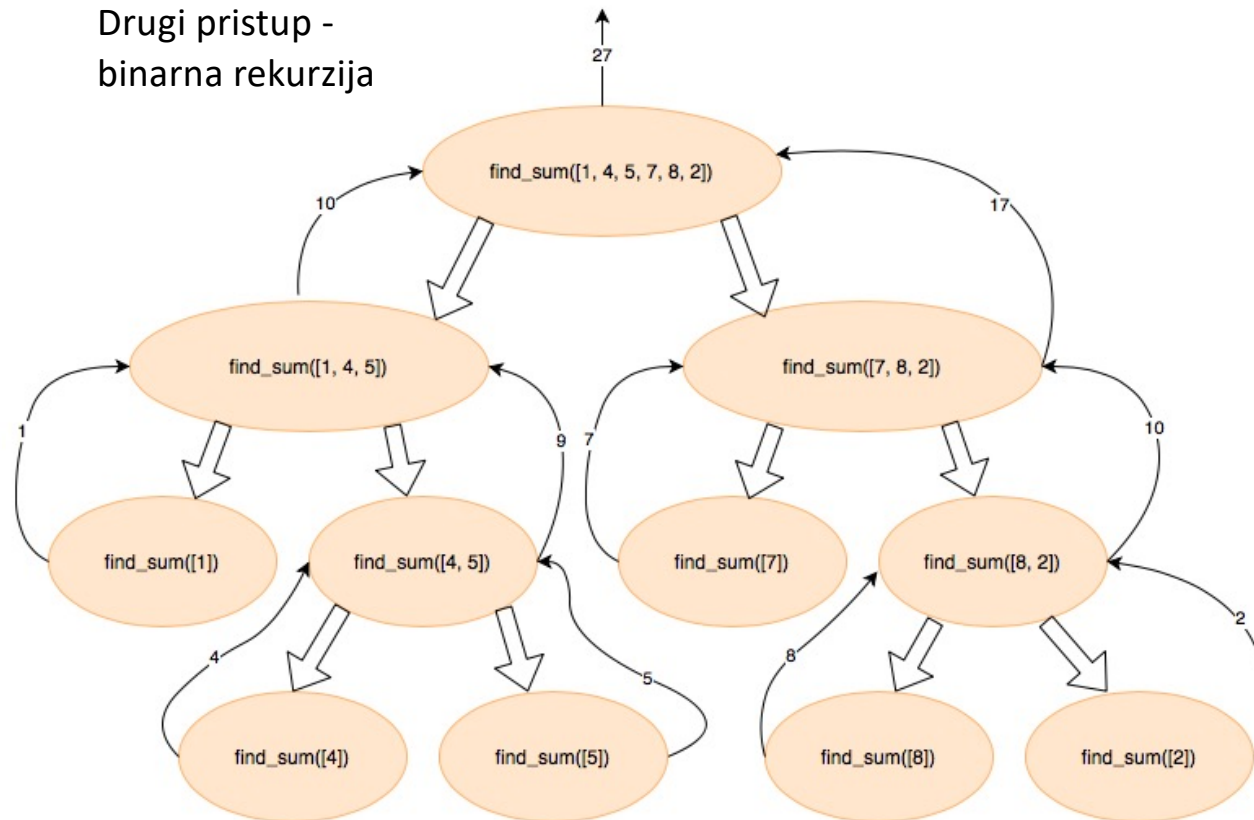
- Poređenje dva pristupa za listu [1, 4, 5, 7, 8, 2]
- Bele strelice reprezentuju rekurzivne pozive
- Crne strelice reprezentuju povratne vrednosti

Prvi pristup –linearna, repna rekurzija



## Primer (6/7)

Drugi pristup -  
binarna rekurzija



# Primer (7/7)

- Na osnovu skica odrediti:
  - Koji od dva pristupa brže dolazi do rešenja?
  - Koja je maksimalna dužina liste za koju su pojedinačni pristupi primenljivi?

# Saveti za zadatke

- Pokušajte da izbegnete upotrebu globalnih varijabli

# Zadatak 1

- Naći sumu prvih  $n$  prirodnih brojeva rekursivnim putem.



## Zadatak 2

- Naći maksimalni element niza rekursivnim putem.

## Zadatak 3

- Izračunati  $n$ -ti Fibonačijev broj pomoću:
  1. Linearne
  2. Binarne rekurzije.

*\* Uporedite vreme izvršavanja za obe implementacije.*

## Zadatak 4

- Napisati rekurzivnu funkciju za replikaciju. Funkcija ima dva parametra – prvi je broj koji treba umnožiti, a drugi broj ponavljanja.
- Primer poziva: `replicate(5,6)`
- Rezultat: `[5, 5, 5, 5, 5, 5]`

## Zadatak 5

- Prikazati apsolutne putanje svih datoteka na zadatoj putanji u direktorijumskom stablu.
- *Napomena:* koristiti moduo **os**

## Zadatak 6

- Napisati program za konverziju pozitivnog celog broja zapisanog u dekadnom sistemu u string u zadatoj brojnoj osnovi.
- *Napomena:* brojne osnove mogu imati vrednost od 2 do 16

## Zadatak 7

- *Tower of Hanoi*
- Data je platforma sa tri stuba označena sa a, b i c. Na stubu a nalaze se n diskova složenih po veličini od najmanjeg ka najvećem. Napisati algoritam za prebacivanje diskova sa stuba a na stub b.
- *Napomene:*
  - Prebacuje se jedan disk po potezu.
  - Diskovi uvek moraju biti sortirani od najmanjeg ka najvećem.