Praktikum iz programiranja 3

2023/24

Rekurzija

- U matematici ili informatici označava postupak ili funkciju koji u svojoj definiciji koriste sami sebe
- Rekurzivne definicije su prisutne i u matematici:

Definicija prirodnih brojeva:

- 1 je prirodan broj
- Ako je n prirodan broj, onda je to i n+1

Fibonačijev niz:

- 0. član niza je 1
- 1. član niza je 1
- Svaki n-ti član niza (n>1) je suma prethodna dva ($a_n = a_{n-1} + a_{n-2}$)

Rekurzija

- U matematičkom smislu rekurzija predstavlja definisanje problema uz pomoć samog tog problema.
- Drugi način definisanja rekurzije kaže da rekurzija predstavlja način definisanja problema preko pojednostavljene verzije istog tog problema.
- Jedan primer kojim se ovo može opisati je rešavanje problema pronalaska puta do kuće (problem ćemo označiti izrazom “pronađi put do kuće”). Rekurzijom bi se ovaj problem mogao opsati u tri koraka, na sledeći način:
  1) ako si kod kuće, ostani u mestu,
  2) inače, napravi jedan korak prema kući,
  3) pronađi put do kuće.
- Tačka pod brojem 3 u ovom opisu problema predstavlja poziv istog problema iz definicije, ali posle učinjene jedne jednostavne akcije, a to je jedan korak prema kući, problem je pojednostavljen.

Rekurzija

- Opisani postupak se može uopštiti, čime dobijamo generalni algoritam koji rešava probleme rekurzijom, i on se sastoji od sledeća tri koraka:

  1) trivajlni slučaj (kojim se prekida proces izračunavanja),
  2) izvršavanje jedne akcije koja nas vodi ka trivijalnom slučaju,
  3) rekurzivni poziv.

- Ovako opisani postupak rešavanja problema ima algoritamski oblik i za većinu problema se može implementirati.

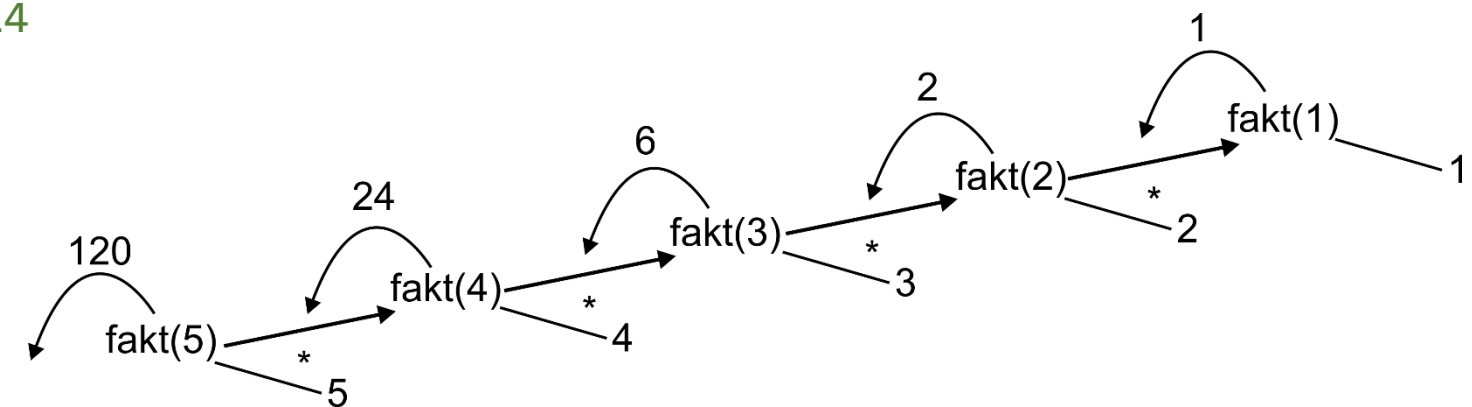
Rekurzija - Faktorijel

- $0! = 1 \quad n! = n \cdot (n-1)!$

```
def fakt(n):
    if n<2:
        return 1
    else:
        return n*fakt(n-1)
print(fakt(5))
```

```
fakt(5) = 5 · fakt(4)
        = 5 · (4 · fakt(3))              // (jer je fakt(4) = 4 · fakt(3))
        = 5 · (4 · (3 · fakt(2)))        // (jer je fakt(3) = 3 · fakt(2))
        = 5 · (4 · (3 · (2 · fakt(1)))) // (jer je fakt(2) = 2 · fakt(1))
        = 5 · (4 · (3 · (2 · 1)))        // (jer je fakt(1) = 1)

        = 5 · (4 · (3 · (2 · 1)))        // množenje se sada vrši redom
        = 5 · (4 · (3 · 2))              // kojim se f-je završavaju, tj. unazad
        = 5 · (4 · 6)
        = 5 · 24
        = 120
```



Rekurzija

- Svako rešenje rekurizvnog problema se sastoji od dva glavna dela:
  - **Bazni slučaj(evi)** - rešenja trivijalnih instanci problema
  - **Rekurzivni slučajevi** - zavisnost rešenja problema od rešenja manjih instanci

Definicija Rekurzija predstavlja metod u kome rešenje polaznog problema nalazimo koristeći rešenja podproblema iste strukture - jednostavnije instance istog problema.

Rekurzija – često postavljana pitanja

- Funckija nije završila sa radom i ja ne mogu ponovo da je pozovem?
- Ukoliko funkcija pozove samu sebe, dobiću beskonačnu petlju?
- Kako da odredim koji su mi bazni slučajevi potrebni ili mi možda neki nedostaje?

Rekurzija - Fibonači

Fibonačijev niz:
1. član niza je 0
2. član niza je 1
Svaki n-ti član niza (n>2) je suma prethodna dva
($a_n = a_{n-1} + a_{n-2}$)

```
def fibonacci (n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacci(n-1) + fibonacci(n-2)

print(fibonacci(5))
```

```
5
```

Verižni razlomci

- Verižni razlomak je izraz oblika:

$$\alpha = a_0 + \cfrac{b_0}{a_1 + \cfrac{b_1}{a_2 + \cfrac{b_2}{a_3 + \ddots}}}$$

gde su a_i, b_i proizvoljni izrazi.

Jednostavni verižni razlomak je verižni razlomak kojem su svi b_i jednaki 1.

Verižni razlomci

- Jedan od verižnih razlomaka se vezuje za vrednost broja pi:

$$\frac{4}{\pi} = 1 + \cfrac{1}{3 + \cfrac{4}{5 + \cfrac{9}{7 + \cfrac{16}{9 + \cfrac{25}{11 + \cfrac{36}{13 + \cdots}}}}}}$$

- Uočavamo dva niza:
  - Sabirci: 1, 3, 5, 7, 9, 11,...
  - Deljenici: 1, 4, 9, 16, 25, 36,...
- Rekurzija se ne može širiti u beskonačnost

Verižni razlomci

```
import math
def verizni(n):
    if n>200:
        return 2*n-1
    else:
        return (2*n-1)+n*n/verizni(n+1)


print(verizni(1))
print(4/math.pi)
```

```
1.2732395447351628
1.2732395447351628
```

String u broj, broj u string

- Napisati rekurzivnu funkciju koja ceo broj pretvara u niz karaktera.

```
def itoa(i):
    if i<10:
        return chr(ord('0')+i)

    return itoa(i//10) + chr(ord('0')+(i % 10))

n = int(input())
print(type(n))
k = itoa(n)
print(k)
print(type(k))
```

```
123456
<class 'int'>
123456
<class 'str'>
```

String u broj, broj u string

- Napisati rekurzivnu funkciju koja niz karaktera (string koji od znakova sigurno sadrži samo cifre pa to nije potrebno proveravati) pretvara u broj.

```
def atoi(s):
    if s=='':
        return 0

    return 10*atoi(s[0:len(s)-1]) + ord(s[len(s)-1])- ord('0')

n = input()
print(type(n))
k = atoi(n)
print(k)
print(type(k))
```

```
123
<class 'str'>
123
<class 'int'>
```

Obrnut redosled cifara

- Napisati rekurzivnu funkciju kojom se cifre celog broja okreću u obrnutom redosledu.

```
def okreni_broj(n, reversed_n=0):
    if n == 0:
        return reversed_n
    else:
        zadnja_cifra = n % 10
        preostali_broj = n // 10
        return okreni_broj(preostali_broj, reversed_n * 10
+ zadnja_cifra)

# Primer poziva funkcije
n = 12345
print(okreni_broj(n))
```

Poređenje dva stringa

- Napraviti rekurzivnu f-ju koja poredi dva prosledjena string i daje meru udaljenosti.
- Udaljenost poredi vrednosti iz ASCII tabele odgovarajucih karaktera (odgovarajuci karakteri su npr. prvi karakteri oba stringa, drugi karakteri, treci isl.) prosledjenih stringova i ako je ASCII vrednost karaktera prvog stringa veca od ASCII vrednosti karaktera drugog stringa onda se udaljenost povecava za 1, ako su odgovarajuca dva karaktera ista onda i udaljenost ostaje ista,a ako je ASCII vrednost karaktera drugog stringa veca onda se udaljenost smanjuje za 1.
- Ako prvi string ima veci broj karaktera udaljenost se uvecava za razliku u broju karaktera, u suprotnom udaljenost se smanjuje za tu razliku.

Ulaz	Izlaz
123 123	0
123 124	-1
abc 111	3
1212 2121	0
1212 21211	-1
12121 2121	1

Poređenje dva stringa

```
def dststr(s1, s2):
    if s1=='' and s2=='':
        return 0

    if s1=='':
        return -1+dststr(s1, s2[1:len(s2)])
    elif s2=='':
        return 1+dststr(s1[1:len(s1)],s2)

    d = dststr(s1[1:len(s1)],s2[1:len(s2)])
    if s1[0]==s2[0]:
        return 0 + d
    elif s1[0]>s2[0]:
        return 1 + d
    else:
        return -1 + d
s1 = input()
s2 = input()
print(dststr(s1,s2))
```

Da li je string palindrom

- Napisati rekurzivnu funkciju koja ispituje da li je uneti string palindrom.

```
def da_li_je_palindrom(rec):
    if len(rec) == 1 or len(rec) == 0:
        return True
    else:
        if rec[0] == rec[-1]:
            return da_li_je_palindrom(rec[1:-1])
        else:
            return False

s = input()
s = s.lower()
print(da_li_je_palindrom(s))
```

Zbir parnih

- Napisati rekurzivnu funkciju koja u listi celih brojeva određuje zbir elemenata na parnim pozicijama.

```
def zbirP(L):
    if L==[]:
        return 0
    return L[0]+zbirP(L[2:len(L)])



L = []
n = int(input())
for i in range(n):
    L +=[int(input())]

print(zbirP(L))
```

```
5
1
2
3
4
5
9
>>>
```

Kombinacija bez ponavljanja

- Od n elemenata potrebno je izabrati k različitih elemenata (k ≤ n). Jedan element može samo jednom birati, dok je redosled proizvoljan.

- Broj različitih mogućih izbora:

$$C_n^k = \binom{n}{k} = \frac{n!}{k!\,(n-k)!}$$

- Od 5 elemenata {1,2,3,4,5} treba izabrati po dva elementa:

[5, 4] [5, 3] [5, 2] [5, 1] [4, 3] [4, 2] [4, 1] [3, 2] [3, 1] [2, 1]

Kombinacija bez ponavljanja

```
brk=[0]
def kombinacije_bez_ponavljanjem(L_pom,L,n,k):
    if k==0:
        brk[0] +=1
        print(L_pom[-1::-1])
    else:
        L_pom[k-1]=L[n-1]
        kombinacije_bez_ponavljanjem(L_pom,L,n-1,k-1)
        if n>k:
            kombinacije_bez_ponavljanjem(L_pom,L,n-1,k)

L=[1,2,3]
L_pom=[0,0]
k=2
n=3
kombinacije_bez_ponavljanjem(L_pom,L,n,k)
print(brk)
```

```
[3, 2]
[3, 1]
[2, 1]
[3]
>>>
```

Kombinacija sa ponavljanjem

- Od n elemenata potrebno je izabrati k elementa (k ≤ n). Jedan element se može više puta birati (najviše k puta), dok je redosled proizvoljan.

- Broj različitih mogućih izbora:

$$\bar{C}_n^k = \binom{n+k-1}{k}$$

- Od 4 elemenata {1,2,3,4} treba izabrati po dva elementa:

[4, 4] [4, 3] [4, 2] [4, 1] [3, 3] [3, 2] [3, 1] [2, 2] [2, 1] [1, 1]

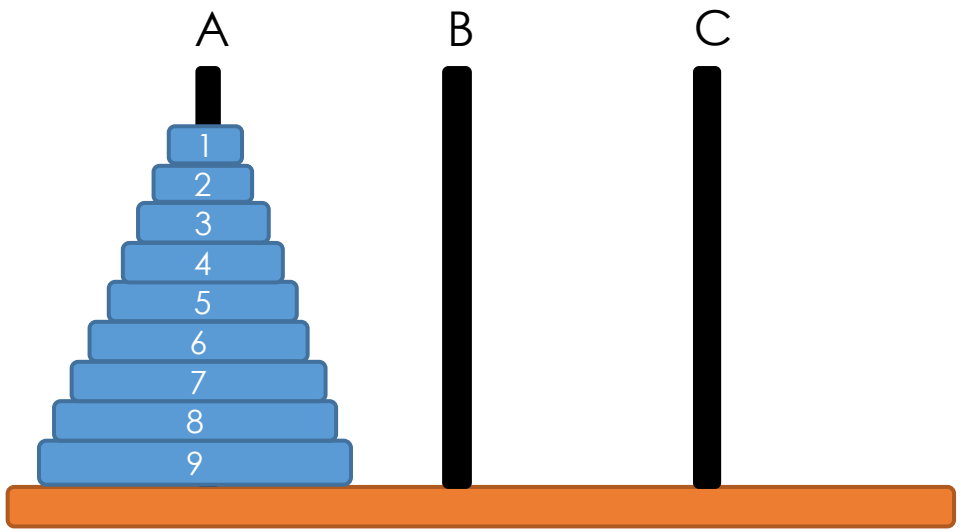
Kombinacija sa ponavljanjem

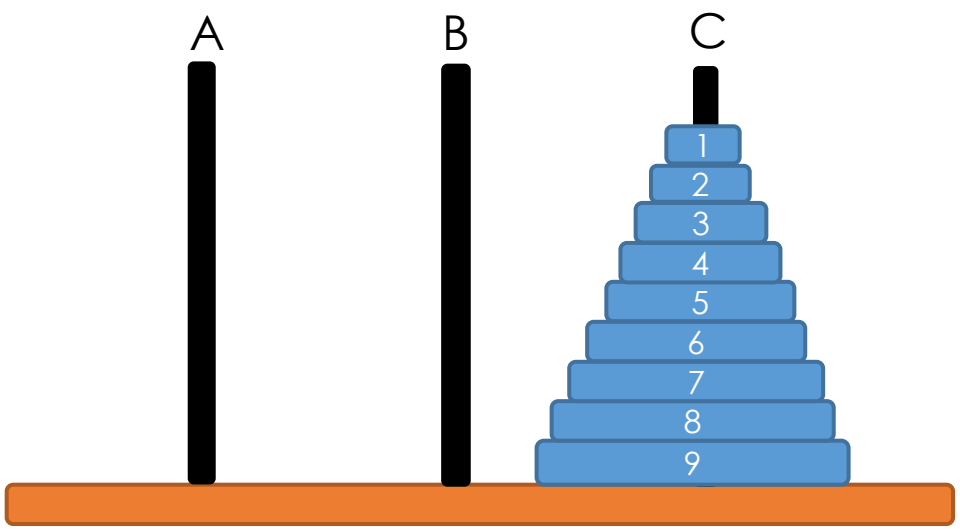
```
brk=[0]
def kombinacije_sa_ponavljanjem(L_pom,L,n,k):
    if k==0:
        brk[0] +=1
        print(L_pom[-1::-1])
    else:
        L_pom[k-1]=L[n-1]
        kombinacije_sa_ponavljanjem(L_pom,L,n,k-1)
        if n>1:
            kombinacije_sa_ponavljanjem(L_pom,L,n-1,k)
L=[1,2,3]
L_pom=[0,0]
k=2
n=3
kombinacije_sa_ponavljanjem(L_pom,L,n,k)
print(brk)
```

```
[3, 3]
[3, 2]
[3, 1]
[2, 2]
[2, 1]
[1, 1]
[6]
>>>
```

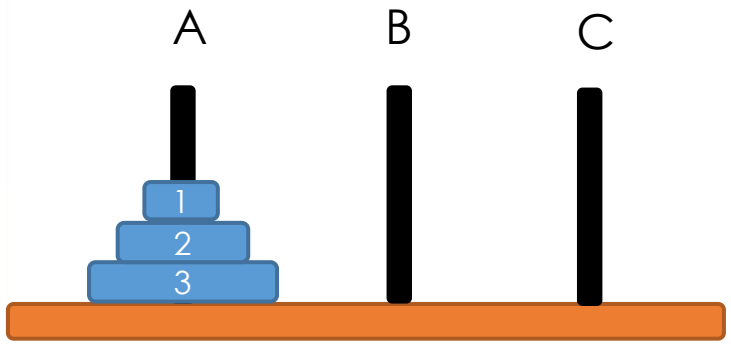
Hanojske kule

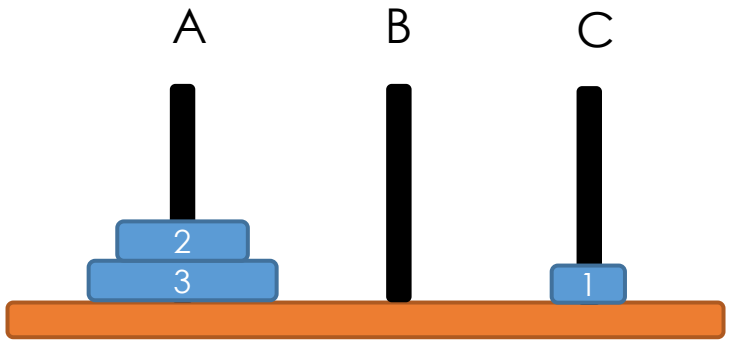
- Data su tri štapa i na jednom od njih 9 diskova, različitih veličina, poređani po veličini. Potrebno je prebaciti diskove na drugi štap. Pravila prebacivanja su sledeća:
  - U jednom koraku može se prebaciti disk sa vrha štapa na vrh drugog štapa.
  - U svakom trenutku diskovi na štapovima moraju biti poređani po veličini.

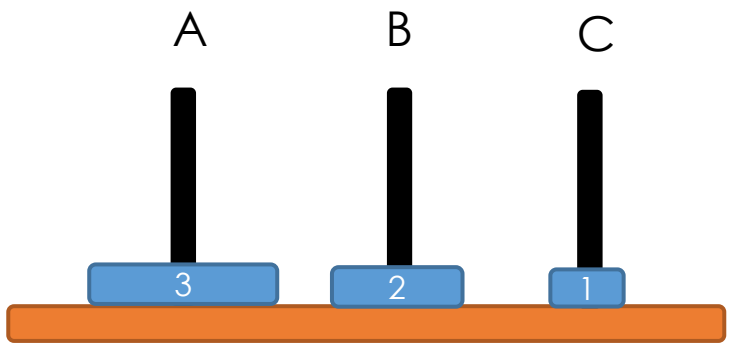


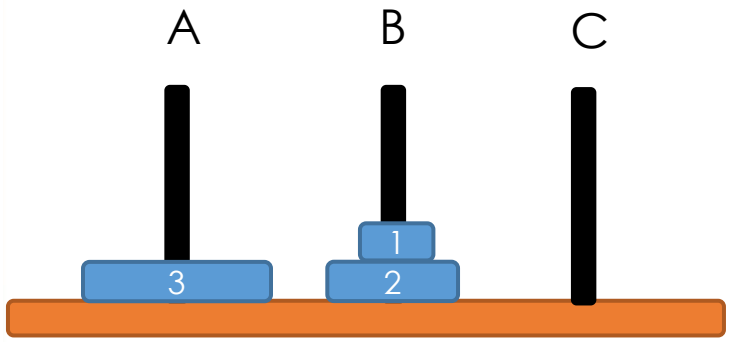


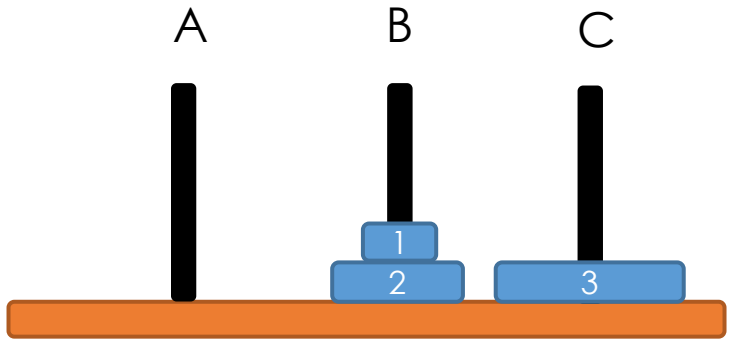
Hanojske kule

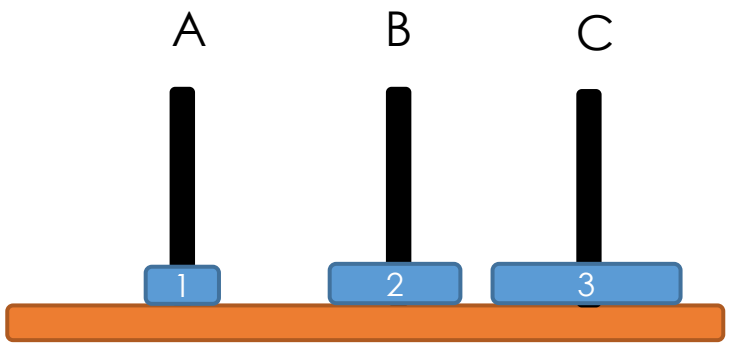


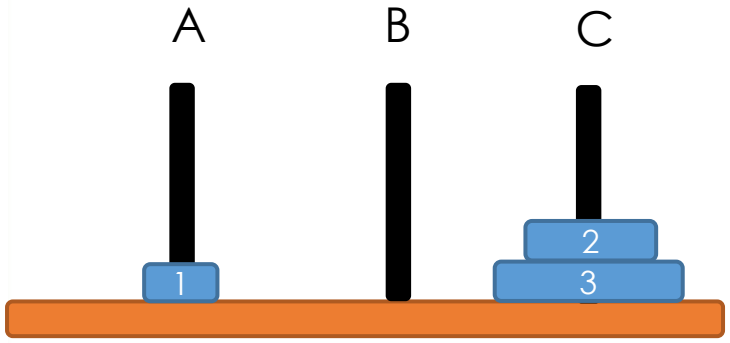


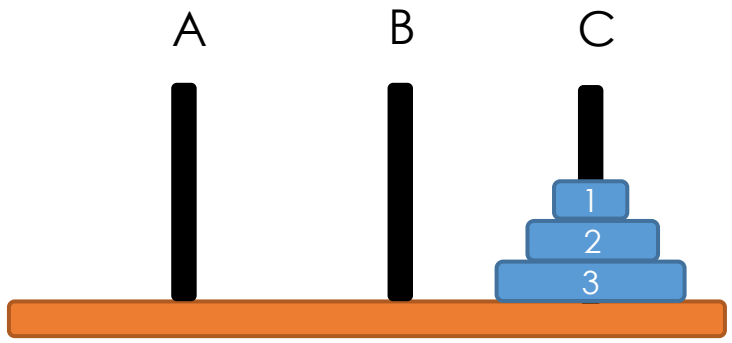








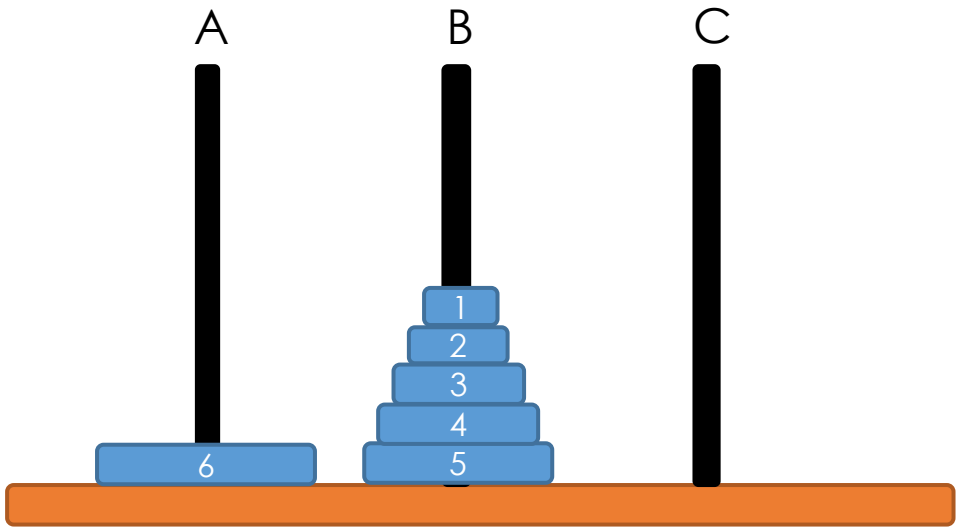




Hanojske kule

- Označimo štapove sa A, B i C a diskove sa $1, 2, \ldots, n$. Pretpostavimo da želimo da diskove premestimo sa štapa A na štap C.

- Da bi disk veličine n prebacili sa štapa A na štap C svi ostali diskovi moraju biti na štapu B.



Hanojske kule

- U prvom delu ćemo prebaciti diskove $1, 2, \ldots, n-1$ sa A na B, a zatim disk n sa A na C.
- Sada imamo problem da sa B treba prebaciti $n-1$ disk na C.
- Ovo je isti problem samo sa drugačijim oznakama štapova i sa jednim diskom manje (manja istanca istog problema).
- Poenta je da najveći disk koji je na dnu nekog štapa ne utiče na ostale diskove i njihovo prebacivanje.

Hanojske kule

```
#f-ja Kule za argumente uzima broj diskova (n) i nazive štapova
#(polazni, dolazni i pomocni)
def Kule(n, polazni, dolazni, pomocni):
#bazni slučaj. Ako postoji samo jedan disk, izvršiti prebacivanje
    if n == 1:
        print ("Pomeri sa " + polazni + " na " + dolazni)
    else:
#prvo se prebacuje n-1 diskova sa polaznog na pomoćni štap
        Kule(n-1, polazni, pomocni, dolazni)
#najveći disk se prebaci sa polaznog na dolazni štap
        Kule(1, polazni, dolazni, pomocni)
#n-1 diskova se prebacuje sa pomoćnog na dolazni štap
        Kule(n-1, pomocni, dolazni, polazni)

Kule(3, "A", "C", "B")
```

```
Pomeri sa A na C
Pomeri sa A na B
Pomeri sa C na B
Pomeri sa A na C
Pomeri sa B na A
Pomeri sa B na C
Pomeri sa A na C
```