

Рекурзија

- Једна од централних идеја рачунарства
- Метода која решавање проблема своди на решавање проблема мање димензије

- Рекурзивна функција (неформално) је функција која у својој дефиницији има позив те исте функције (директно или индиректно)

- Структура – две компоненте:
  - услов рекурзивног позива (и сам рекурзивни позив) и
  - услов изласка из рекурзије (и излазак из рекурзије)

- Примерена рекурзивно формулисаним проблемима

Рекурзија: пример

- Формула за рачунање факторијела природног броја (или 0):
  - Итеративна

$$n! = \begin{cases} 1 & \text{ako je } n = 0, \\ 1 * 2 * \cdots * (n-1) * n & \text{ako je } n > 0 \end{cases}$$

  - Рекурзивна

$$n! = \begin{cases} 1, \text{ako je } n = 0, \\ n * (n-1)!, \text{ako je } n > 0 \end{cases}$$

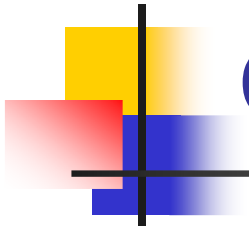
Факторијел: итеративна функција

```
int fakti(int n)
{
   int i;
   int r;
   r=1;
   for(i=1; i<=n; i++) r=r*i;
   return r;
}
```

Факторијел: рекурзивна функција

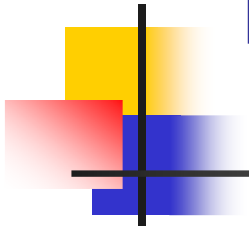
```
int faktr(int n)
{
   int w;
   if(n==0) w=1;
   else w=n*faktr(n-1);
   return w;
}
```

Факторијел: рекурзивна функција



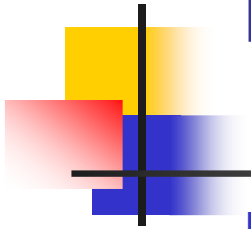
- Позив функције креира примерак сваке унутрашње (локалне) променљиве
- Престаје да постоји по завршетку тог активирања (позива) функције
- Пример: извршавање позива faktr(5)

Извршавање позива faktr(5)



```
faktr(5)
w
w=5*faktr(4)
        w
        w=4*faktr(3)
                w
                w=3*faktr(2)
                        w
                        w=2*faktr(1)
                                w
                                w=1*faktr(0)
                                        w
                                        w=1
                                        return 1
                                return 1
                        return 2
                return 6
        return 24
return 120
```

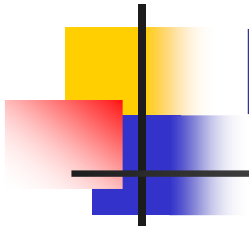
Рекурзија: стек



- Рекурзијом се не штеди меморија
- Простор за памћење података при рекурзивним позивима функције
- Стек – простор за локалне променљиве и параметре свих функција
- Меморија се на стеку додељује по LIFO принципу (Last In First Out)
- Простор се додељује и ослобађа на “врху” стека, динамички
  - Променљиве из main()
  - Променљиве и параметри функција позваних из main,
  - Променљиве и параметри функција позваних из тих функција итд.
  - По повратку из функције ослобађа се простор за њене податке на стеку

Рекурзија: стек за faktr(5)





Рекурзија: својства

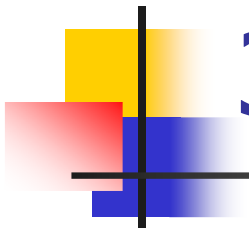
- Не повећава брзину извршавања
- Запис је компактнији
- Лакши за писање и разумевање
- Погодан у обради рекурзивно дефинисаних структура података (нпр. листе, стабла)

Рекурзија: примери

ЗБИР

$$\sum_{i=1}^{n} a_i = 0,\ n=0$$

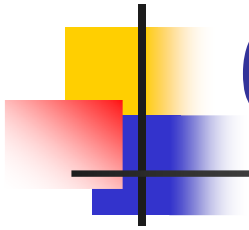
$$\sum_{i=1}^{n} a_i = \sum_{i=1}^{n-1} a_i + a_n\ ,\ n>0$$



Збир: рекурзивна функција

```
float sum(float a[], unsigned n) {
    if (n == 0)
        return 0.0f;
    else
    return sum(a, n-1) + a[n-1];
}
```

Штампање цифара целог броја



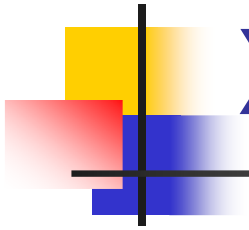
- Једно решење: итеративна функција која рачуна цифре као остатак при целобројном дељењу са 10, од цифре најмање до цифре највеће тежине

- Друго решење: рекурзивна функција – позива саму себе за штампање водећих цифара (цифара количника при дељењу са 10), а затим штампа последњу цифру

Штампање цифара целог броја: рекурзивно решење

```
/* printd: stampa cifre celog broja n */
void printd(int n)
{
    if(n<0) {
        putchar('-');
        n = -n;
    }
    if(n/10)
        printd(n/10);
    putchar(n%10 + '0');
}
```

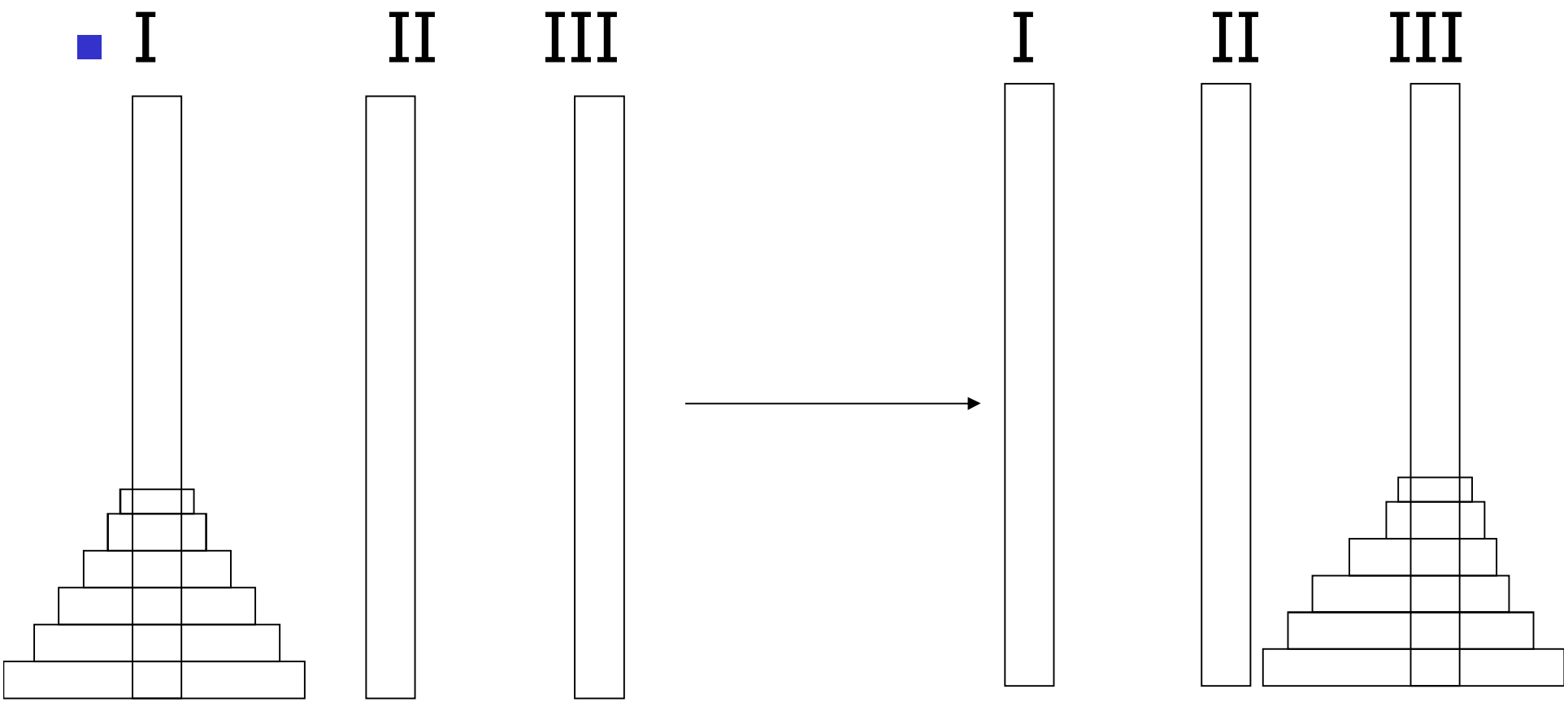
Ханојске куле



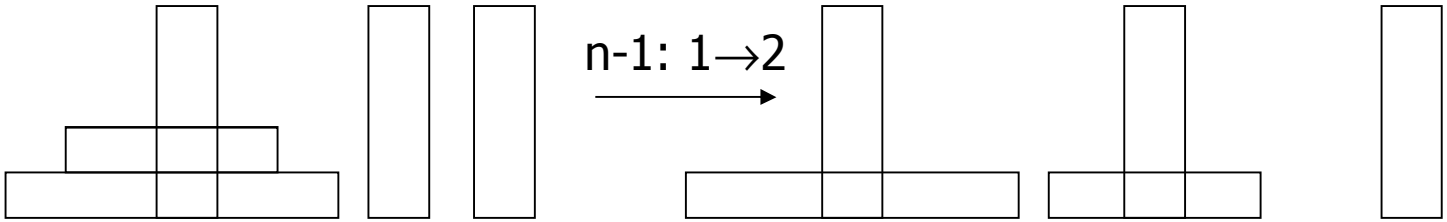
- Задатак: дата су три штапа, на првом *n дискова различитих пречника, поређаних тако да мањи диск лежи на већем. Треба преместити све дискове на трећи штап уистом поретку, тако што се премешта један по један диск, коришћењем сва три штапа*

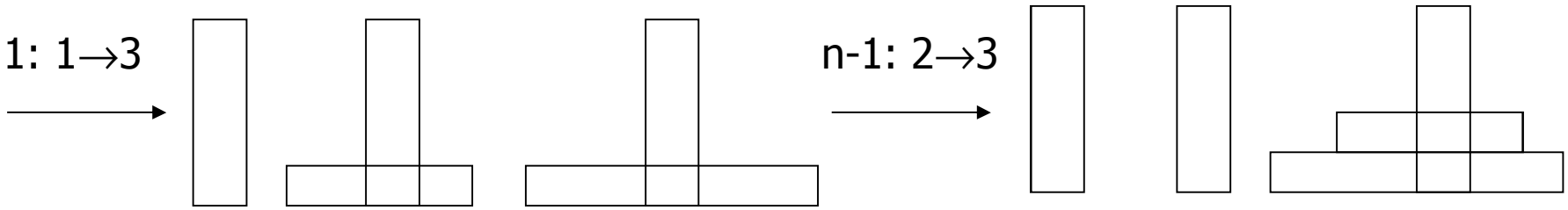
Ханојске куле

- Задатак: преместити *n* дискова као на слици, један по један, коришћењем сва три штапа; диск може да лежи само на већем диску



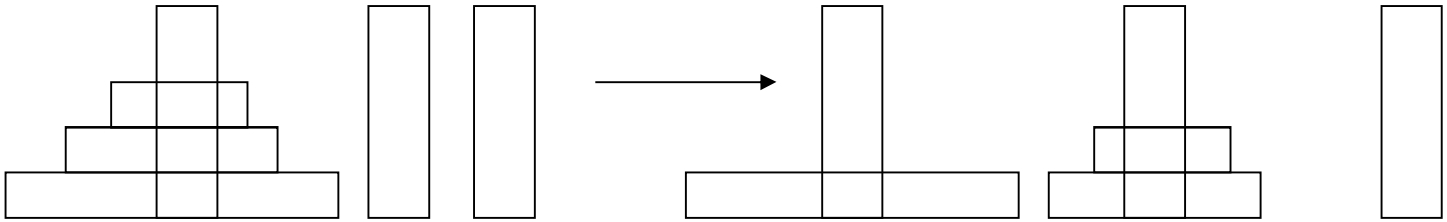
Ханојске куле: n=2:
1→3 (2) (са 1. на 3. помоћу 2.)



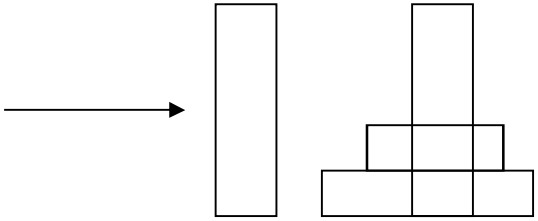


Ханојске куле: n=3: 1→3 (2)

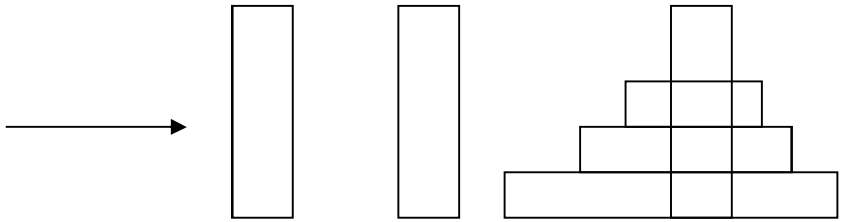
n-1: 1→ 2(3)



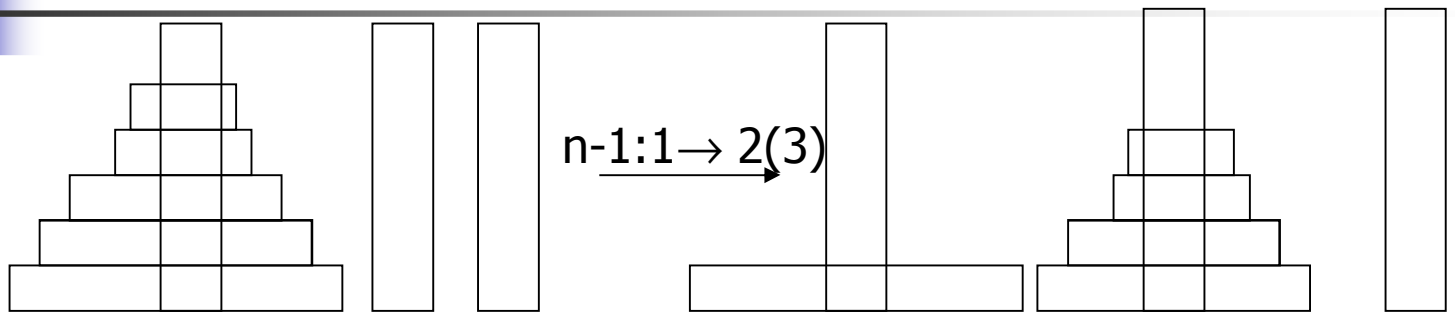
1: 1→3

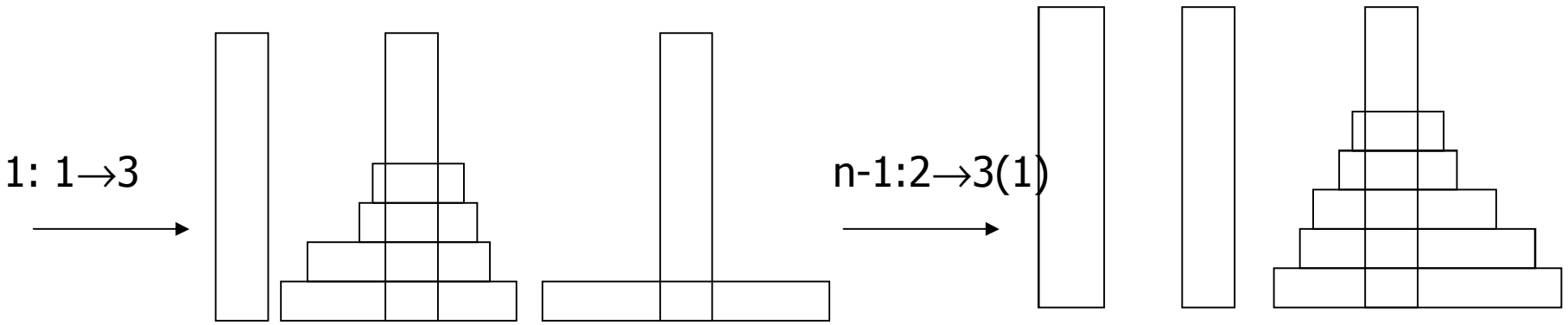


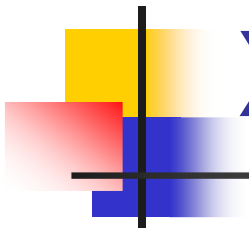
n-1: 2 → 3(1)



Ханојске куле: n>3 (1→3 (2))







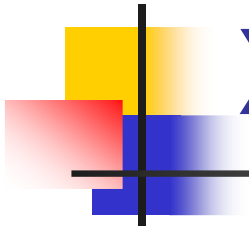
Ханојске куле: програм

```
#include <stdio.h>
/* prebacuje n diskova sa i-tog na j-ti štap */
void prebaci (int n, int i, int j);
void main()
{
   int broj; /*broj diskova */
   printf(”Unesite broj diskova”);
   scanf(”%d”, &broj);
   prebaci(broj, 1, 3);
   return;
}
```

Ханојске куле: функција пребацивања

```
void prebaci(int n, int i, int j)
{
    int k;
    /* odrediti pomoćni štap k */
        switch(i+j){
            case 3: k=3; break;
            case 4: k=2; break;
            case 5: k=1; break;
        }
        if(n==1)
            printf("Prebaceno sa %d na %d", i, j);
        else {
            prebaci(n-1, i, k);
            prebaci(1,i,j);
            prebaci(n-1, k,j);
        }
        return;
}
```

Ханојске куле: сложеност



- Број премештања n дискова је 2^n-1,
- Сложеност алгоритма је експоненцијална (O(2^n))

Бинарно претраживање

- Итеративна функција:

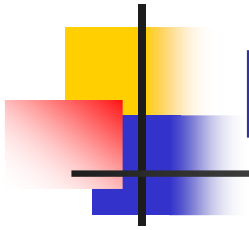
```
int binsearch(int x, int b[], int m)
{
    int l=0, d=m-1, s;
    while(l<=d){
        s=(l+d)/2;
        if(x<b[s]) d=s-1;
        else if(x>b[s]) l=s+1;
        else retrurn s;
    }
    return-1;
}
```

Бинарно претраживање

- Рекурзивна функција:

```
int binsearchr(int x, int b[], int l, int d)
{
    int s;
    if(l>d) return -1;
    else{
            s=(l+d)/2;
            if(x==b[s]) return s;
            else if(x<b[s]) d=s-1;
            else l=s+1;
            return binsearchr(x,b,l,d);
    }
}
```

Елиминација репне рекурзије



- Репна рекурзија:
  - Само један рекурзивни позив
  - Последњи исказ функције
  - Може да се замени GOTO исказом скока на почетак функције

Елиминација репне рекурзије: бинарно претраживање 2

```
int binsearchr(int x, int b[], int l, int d)
{
    int s, bp;
    if(l>d) return -1;
    else{
        s=(l+d)/2;
        if(x==b[s]) return s;
        else {
            if(x<b[s]) d=s-1;
            else l=s+1;
            bp=binsearchr(x,b,l,d);
            return bp;
        }
    }
}
```

Елиминација репне рекурзије: бинарно претраживање – корак 2

```
int binsearchr(int x, int b[], int l, int d)
{
        int s;
label:  if(l>d) return -1;
        else{
                s=(l+d)/2;
                if(x==b[s]) return s;
                else {
                        if(x<b[s]) d=s-1;
                        else l=s+1;
                        goto label;
                }
        }
}
```

Елиминација репне рекурзије: бинарно претраживање – корак 3

```
int binsearchr(int x, int b[], int l, int d)
{
        int s;
label:  if(d>=l)
        {
                s=(l+d)/2;
                if(x!=b[s])
                {
                        if(x<b[s]) d=s-1;
                        else l=s+1;
                        goto label;
                }
                else return s;
        }
        else return -1;
}
```

Елиминација репне рекурзије: елиминација goto

```
int binsearchr(int x, int b[], int l, int d)
{
        int s;
        while (d>=l)
        {
                s=(l+d)/2;
                if(x!=b[s])
                {
                        if(x<b[s]) d=s-1;
                        else l=s+1;
                }
                else return s;
        }
        return -1;
}
```