Programozás 2

Rekurzió; egyszerű feladatok megoldása rekurzióval; közvetett rekurzió, a Hanoi torony megoldása rekurzióval

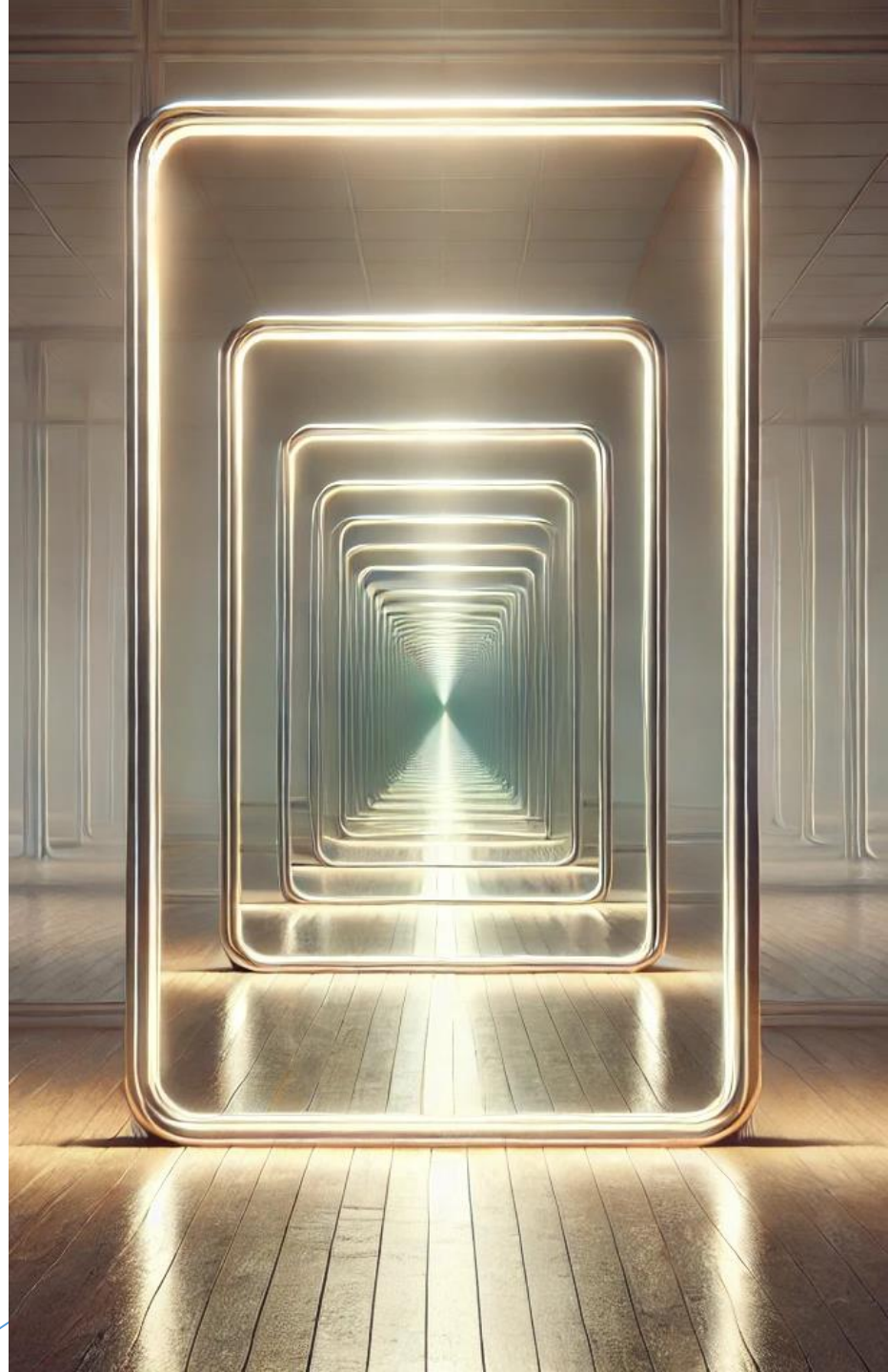
Rekurzió

A rekurzió egy olyan programozási technika, amelyben egy függvény saját magát hívja meg egy kisebb vagy egyszerűbb problémára.

Egy rekurzív függvény általában két részből áll:

- **Alapeset (base case)** – amikor a rekurzió megáll, és nem hívódik meg újra.
- **Rekurzív eset (recursive case)** – amikor a függvény önmagát hívja meg egy kisebb részproblémára.

Egyszerű példa: Gondoljunk egy tükör elé állított tükörre – a kép újra és újra megismétlődik, egyre kisebb méretben (itt azonban nincs alapeset, végtelenszer megismétlődik).



Feladat: Készítsünk függvényt a faktoriális kiszámítására!

Faktoriális definíciója: n! = n * (n-1) * (n-2) * ... * 1

Megoldás iterációval (ciklussal), rekurzió használata nélkül:

```
int faktorialis_iterativ(int n) {
    int f = 1;
    for (int i=n; i>=1; i--) {
        f = f * i;
    }
    return f;
}
```

Az algoritmus időbonyolultsága: O(n)

Faktoriális definíciója: $n! = n * (n-1) * (n-2) * \ldots * 1$

Rekurzióval: $n! = n * (n-1)!$

Alapeset: $0! = 1$

Megoldás rekurzió használatával:

```
int faktorialis_rekurziv(int n) {
    if (n == 0) {
        return 1;
    }
    return n * faktorialis_rekurziv(n - 1);
}
```

```
int faktorialis_rekurziv(int n) {
    if (n == 0) {
        return 1;
    }
    return n * faktorialis_rekurziv(n - 1);
}
```

Pl. n=5-re:

faktorialis_rekurziv(5) = 5 * faktorialis_rekurziv(4) = 5 * 24 = 120
faktorialis_rekurziv(4) = 4 * faktorialis_rekurziv(3) = 4 * 6 = 24
faktorialis_rekurziv(3) = 3 * faktorialis_rekurziv(2) = 3 * 2 = 6
faktorialis_rekurziv(2) = 2 * faktorialis_rekurziv(1) = 2 * 1 = 2
faktorialis_rekurziv(1) = 1 * faktorialis_rekurziv(0) = 1 * 1 = 1
faktorialis_rekurziv(0) = 1

Az algoritmus időbonyolultsága: O(n)

Rekurzió vs. iteráció: mikor érdemes melyiket használni?

	Rekurzió	Iteráció (ciklus)
Memóriahasználat:	Magasabb (minden hívás új stack frame-et igényel)	Alacsonyabb (ugyanaz a változó módosul)
Egyszerűség:	Egyes problémák esetén olvashatóbb, elegánsabb	Gyakran könnyebben optimalizálható
Sebesség:	Lassabb (rekurzív hívások és stack-kezelés miatt)	Gyorsabb és hatékonyabb
Használati esetek:	Hierarchikus, osztható problémák (pl. fa bejárása, Fibonacci, Hanoi torony)	Egyszerű ismétlődések (pl. tömb végigjárása, számlálások)

- Ha a probléma természetesen részekre bontható, **rekurzió** lehet előnyös.
- Ha a probléma egyszerű ciklussal megoldható, **iteráció** a hatékonyabb.

Feladat: Készítsünk függvényt az n. Fibonacci szám meghatározására!

Fibonacci számok: 0, 1, 1, 2, 3, 5, 8, 13, 21, …

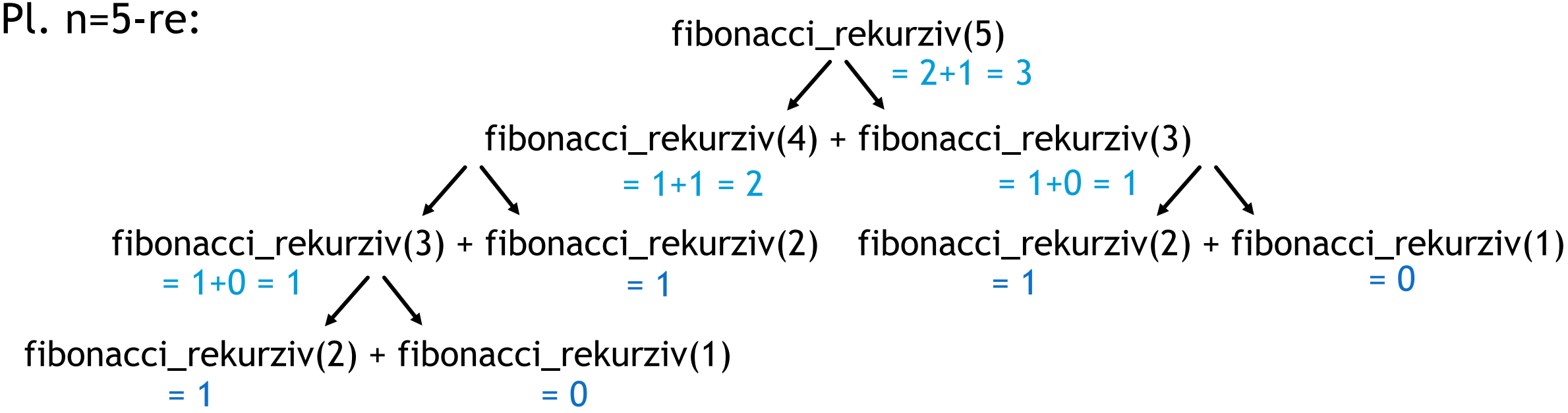
Első Fibonacci szám: F(1) = 0
Második Fibonacci szám: F(2) = 1
Harmadik számtól: F(n) = F(n-1) + F(n-2)

Megoldás rekurzióval:

```
int fibonacci_rekurziv(int n) {
    if (n == 1) return 0;
    if (n == 2) return 1;
    return fibonacci_rekurziv(n - 1) + fibonacci_rekurziv(n - 2);
}
```

```
int fibonacci_rekurziv(int n) {
    if (n == 1) return 0;
    if (n == 2) return 1;
    return fibonacci_rekurziv(n - 1) + fibonacci_rekurziv(n - 2);
}
```

Pl. n=5-re:



Az algoritmus időbonyolultsága: $O(2^n)$

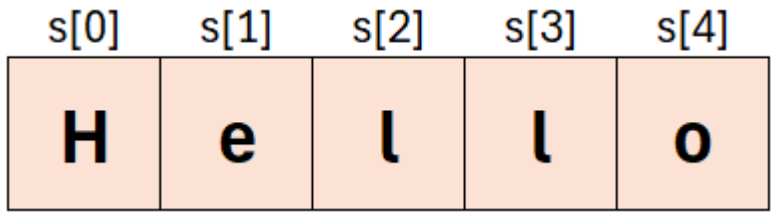
Megoldás iterációval (ciklussal), rekurzió használata nélkül:

```
int fibonacci_iterativ(int n) {
    if (n == 1) return 0;
    if (n == 2) return 1;
    int a = 0, b = 1, c;
    for (int i = 3; i <= n; i++) {
        c = a + b;
        a = b;
        b = c;
    }
    return c;
}
```

i	a	b	c
-	0	1	-
3	1	1	1
4	1	2	2
5	2	3	3

Időbonyolultság: O(n)

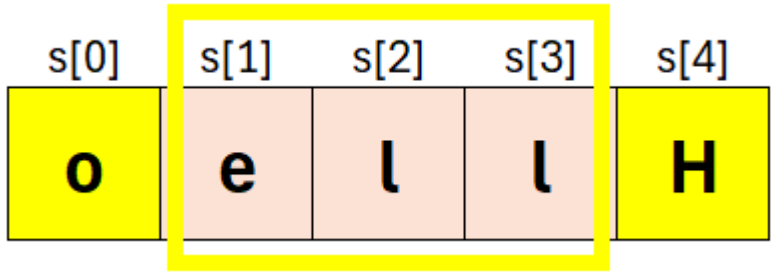
Feladat: Készítsünk egy rekurzív függvényt, amely tükröz egy karakterláncot (sztringet).



Pl. Bemenet: „Hello”

Kimenet: „olleH”

Az alapötlet az, hogy a sztring első és utolsó karakterét felcseréljük, majd a belső részen (első és utolsó karakter nélküli részsztringen) újra alkalmazzuk a folyamatot.

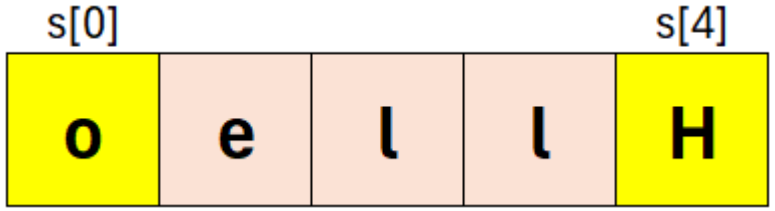


Alapeset: Ha a sztring üres vagy csak egy karakterből áll, nincs mit megfordítani, tehát visszatérünk.

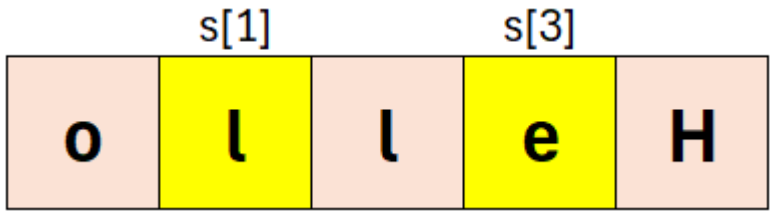
Rekurzív lépés: Cseréljük fel az első és az utolsó karaktert. Hívjuk meg a függvényt a sztring középső részére.

```c
#include <stdio.h>
#include <string.h>

void karakterlanc_tukrozes(char str[], int start, int end) {
    // alapeset
    if (start >= end) return;
    // karakterek csereje
    char temp = str[start];
    str[start] = str[end];
    str[end] = temp;
    // rekurziv fuggvenhivas
    karakterlanc_tukrozes(str, start+1, end-1);
}

int main() {
    char s[] = "Hello";
    int n = strlen(s);
    karakterlanc_tukrozes(s,0,n-1);
    printf("%s", s);
}
```

karakterlanc_tukrozes(s,0,4)



karakterlanc_tukrozes(s,1,3)



karakterlanc_tukrozes(s,2,2)

(start ≥ end) → visszatérés

Az algoritmus időbonyolultsága: O(n)

Közvetett rekurzió

A rekurzív függvény nem közvetlenül hívja meg önmagát, hanem két vagy több függvény hívja egymást.

Feladat: Készítsünk programot, amely eldönti egy természetes számról, hogy az páros-e, ha nincs maradékszámítás (%) műveletünk!

Közvetett rekurzió segítségével:

- Páros(n) függvény:

  Alapeset: ha n=0 → igaz
  ha n=1 → hamis

  Rekurzív lépés: Páros(n) = Páratlan(n-1)

  Az adott szám páros, ha a nála eggyel kisebb szám páratlan.

- Páratlan(n) függvény:

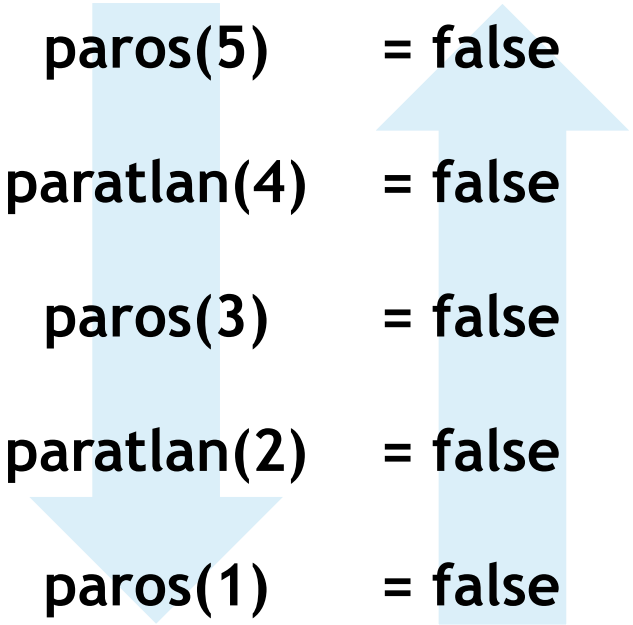
  Alapeset: ha n=0 → hamis
  ha n=1 → igaz

  Rekurzív lépés: Páratlan(n) = Páros(n-1)

  Az adott szám páratlan, ha a nála eggyel kisebb szám páros.

```c
#include <stdio.h>
#include <stdbool.h>

bool paratlan(int n);

bool paros(int n) {
    if (n == 0) return true;
    if (n == 1) return false;
    return paratlan(n - 1);
}

bool paratlan(int n) {
    if (n == 0) return false;
    if (n == 1) return true;
    return paros(n - 1);
}

int main() {
    int n = 5;
    if (paros(n)) { printf("%d paros", n);    }
            else { printf("%d paratlan", n); }
}
```

Pl. n=5-re:



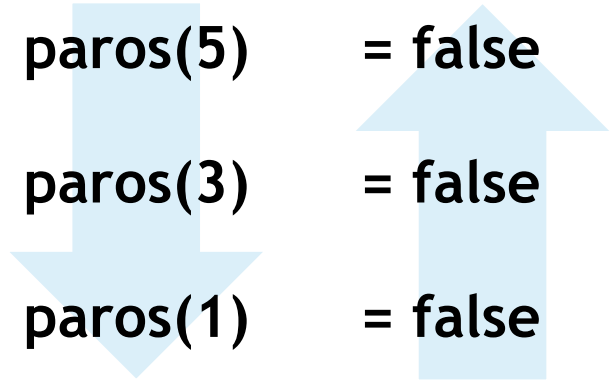
Időbonyolultság: O(n)

Közvetlen rekurzió segítségével:

Mivel a két függvény összevonható, így a feladat megoldható közvetlen rekurzióval is.

```c
#include <stdio.h>
#include <stdbool.h>

bool paros(int n) {
    if (n == 0) return true;
    if (n == 1) return false;
    return paros(n - 2);
}

int main() {
    int n = 5;
    if (paros(n)) { printf("%d paros", n);     }
            else { printf("%d paratlan", n); }
}
```

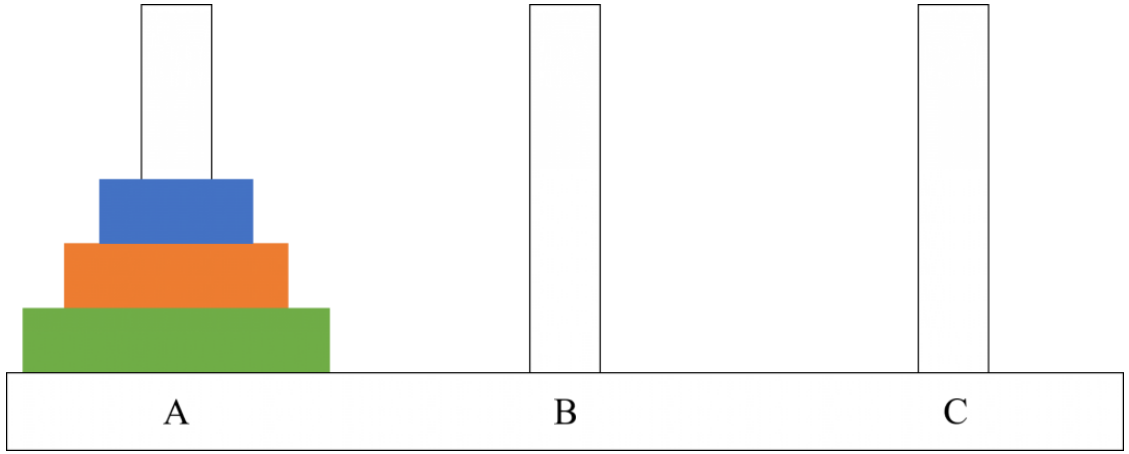
Pl. n=5-re:



Időbonyolultság: O(n)

Hanoi torony

Feladat: Rakjuk át az összes (n) korongot az első rúdról az utolsóra úgy, hogy minden lépésben csak egy korongot lehet áttenni, nagyobb korong nem tehető kisebb korongra, és ehhez összesen három rúd áll rendelkezésünkre.



Játék: https://www.mathsisfun.com/games/towerofhanoi.html

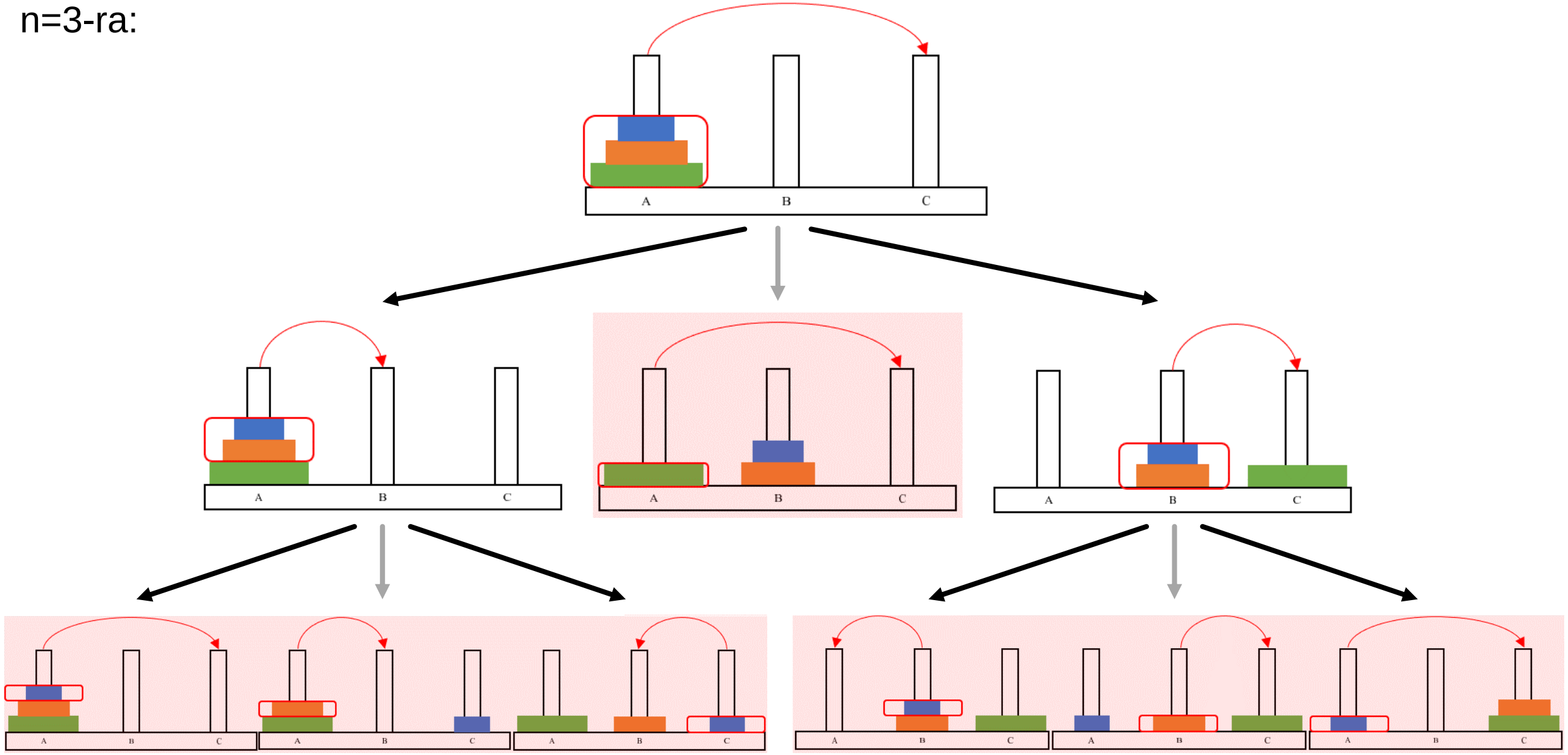
Megoldás rekurzióval:

Rekurzív lépés: Ha az átrakandó korongok száma több 0 (van mit átrakni):

- rakjunk át eggyel kevesebb korongot a segédrúdra (rekurzív hívás),
- rakjuk át az alsó korongot a helyére,
- a segédrúdról tegyük át a korongokat a helyére (rekurzív hívás).

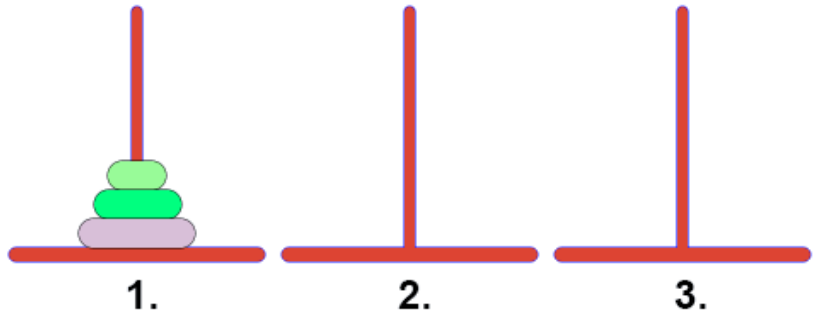
Alapeset: Ha az átrakandó korongok száma 0 (nincs mit átrakni): visszalépés.

n=3-ra:



Az algoritmus időbonyolultsága: $O(2^n)$

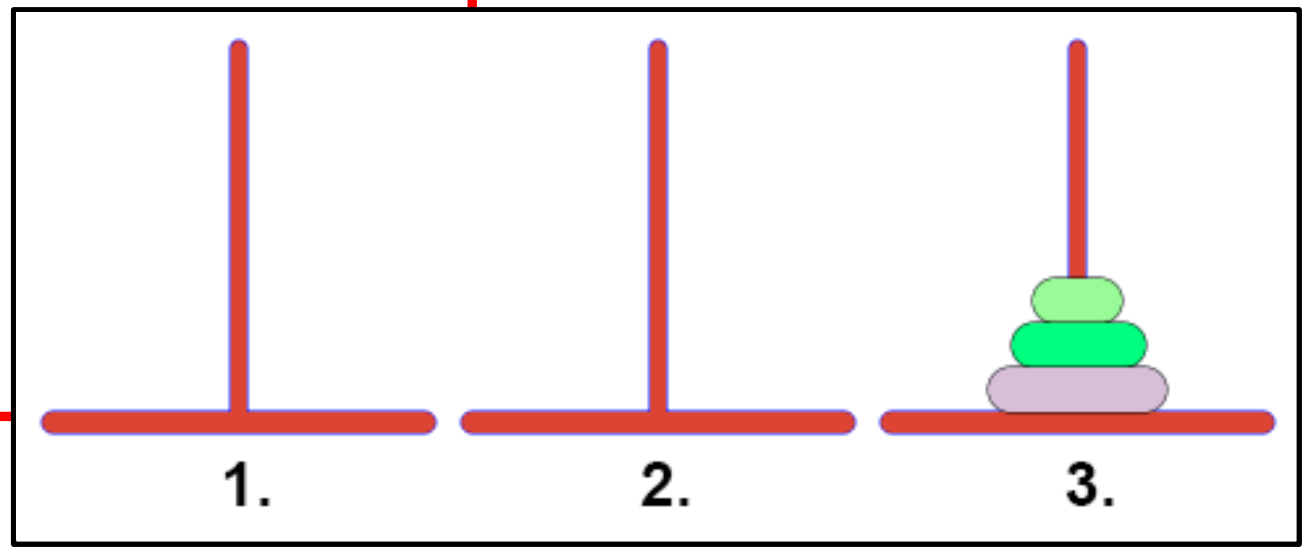
```c
#include <stdio.h>

int N = 3;
int rudak[3] = { N, 0, 0 };
int lepesszam = 0;

// honnan, hova = melyik rudrol (1,2,3) melyik rudra (1,2,3)
void atrak(int honnan, int hova) {
    rudak[honnan-1]--;
    rudak[hova-1]++;
    lepesszam++;
    printf("%d. lepes: %d rudrol %d rudra, korongok: %d %d %d\n",
        lepesszam, honnan, hova, rudak[0], rudak[1], rudak[2]);
}

// n = korongok szama
// honnan, hova = melyik rudrol (1,2,3) melyik rudra (1,2,3)
// tmp = segedrud (meghatarozhato honnan, hova ertekeibol) (1,2,3)
void hanoi(int n, int honnan, int hova) {
    if (n==0) return;
    int tmp = 6 - honnan - hova;
    hanoi(n - 1, honnan, tmp);      // n-1 korong a segedrudra
    atrak(honnan, hova);            // egy korong atrakasa
    hanoi(n - 1, tmp, hova);        // n-1 korong a segedrudrol
}

int main() {
    hanoi(rudak[0], 1, 3);
}
```



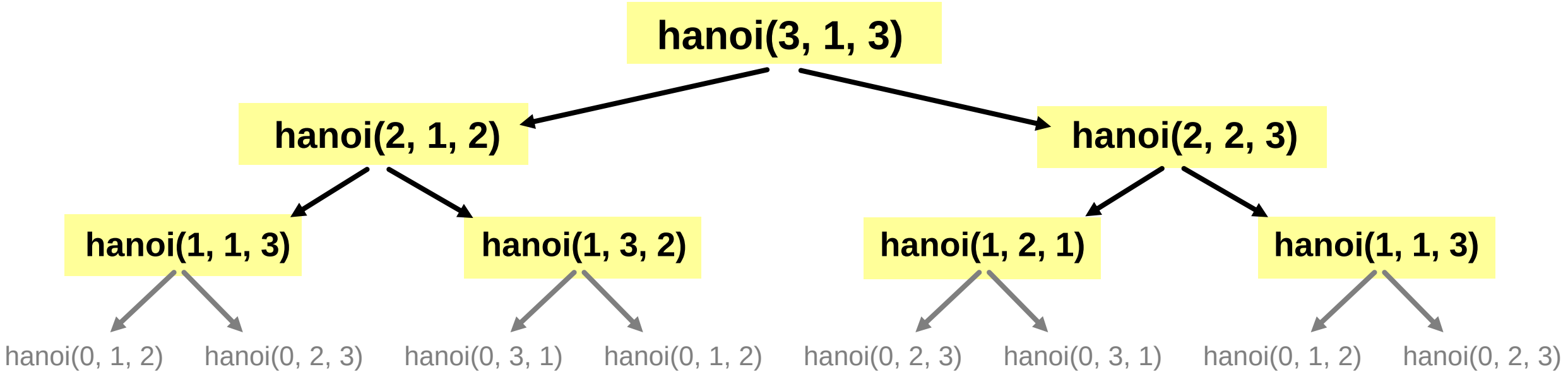
← függvény egy darab korong átrakására

← rekurzív függvény n darab korong átrakására

```
void hanoi(int n, int honnan, int hova) {
    if (n==0) return;
    int tmp = 6-honnan-hova;
    hanoi(n-1, honnan, tmp);
    atrak(honnan, hova);
    hanoi(n-1, tmp, hova);
}
```



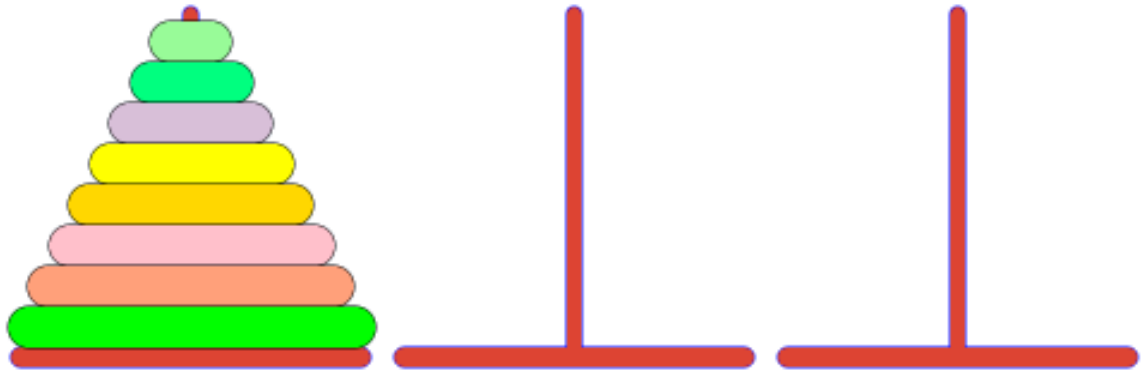
Rekurzív függvényhívások n=3-ra:



Legkevesebb mennyi lépéssel oldható meg a feladat, ha n korongunk van?

Lépések száma:

n = 1	1 lépés
n = 2	3 lépés (1+1+1)
n = 3	7 lépés (3+1+3)
n = 4	15 lépés (7+1+7)
n = 5	31 lépés (15+1+15)
n = 6	63 lépés (31+1+31)
n = 7	127 lépés (63+1+63)
n = 8	255 lépés (127+1+127)



Általánosítva: **n korong esetén a lépésszám 2^n-1**