Algoritmusok gyakorlat, 2020

3. hét, szeptember 21.

3. gyakorlat

Oszd-meg és uralkodj:

A feladatot ***több*** [diszjunkt] ***részfeladat***ra osztjuk, amelyek hasonlóak az eredeti feladathoz, de méretük kisebb, rekurzív módon megoldjuk a részfeladatokat, majd összevonjuk ezeket a megoldásokat, hogy az eredeti feladatra megoldást adjanak.

3. gyakorlat

- **Felosztás**: hogyan osztjuk a feladatot több részfeladatra.
- **Uralkodás**: a részfeladatokat rekurzív módon megoldjuk. Ha a részfeladatok mérete elég kicsi, akkor közvetlenül megoldja a részfeladatokat.
- **Összevonás**: a részfeladatok megoldásait összevonjuk az eredeti feladat megoldásává.

3. gyakorlat

A rekurzió olyan művelet, mely végrehajtáskor a saját maga által definiált műveletet, vagy műveletsort hajtja végre (általában a probléma egy kisebb példányára), ezáltal önmagát ismétli. Egy rekurzív algoritmusnak két része van:

- Alapeset: a megoldást már előre tudjuk vagy könnyen kiszámítható (ekkor nincs rekurzív hívás)
- Rekurzív eset: a megoldást úgy kapjuk, hogy a probléma más, általában kisebb példányainak megoldását használjuk fel.

3. gyakorlat

Faktoriális számítás: $n! = n \cdot (n - 1) \cdot \ldots \cdot 2 \cdot 1$

Rekurzív összefüggés: $n! = n \cdot (n - 1)!$

- Alapeset: $n = 1$
- Rekurzív eset: $n > 1$: $n \cdot (n - 1)!$

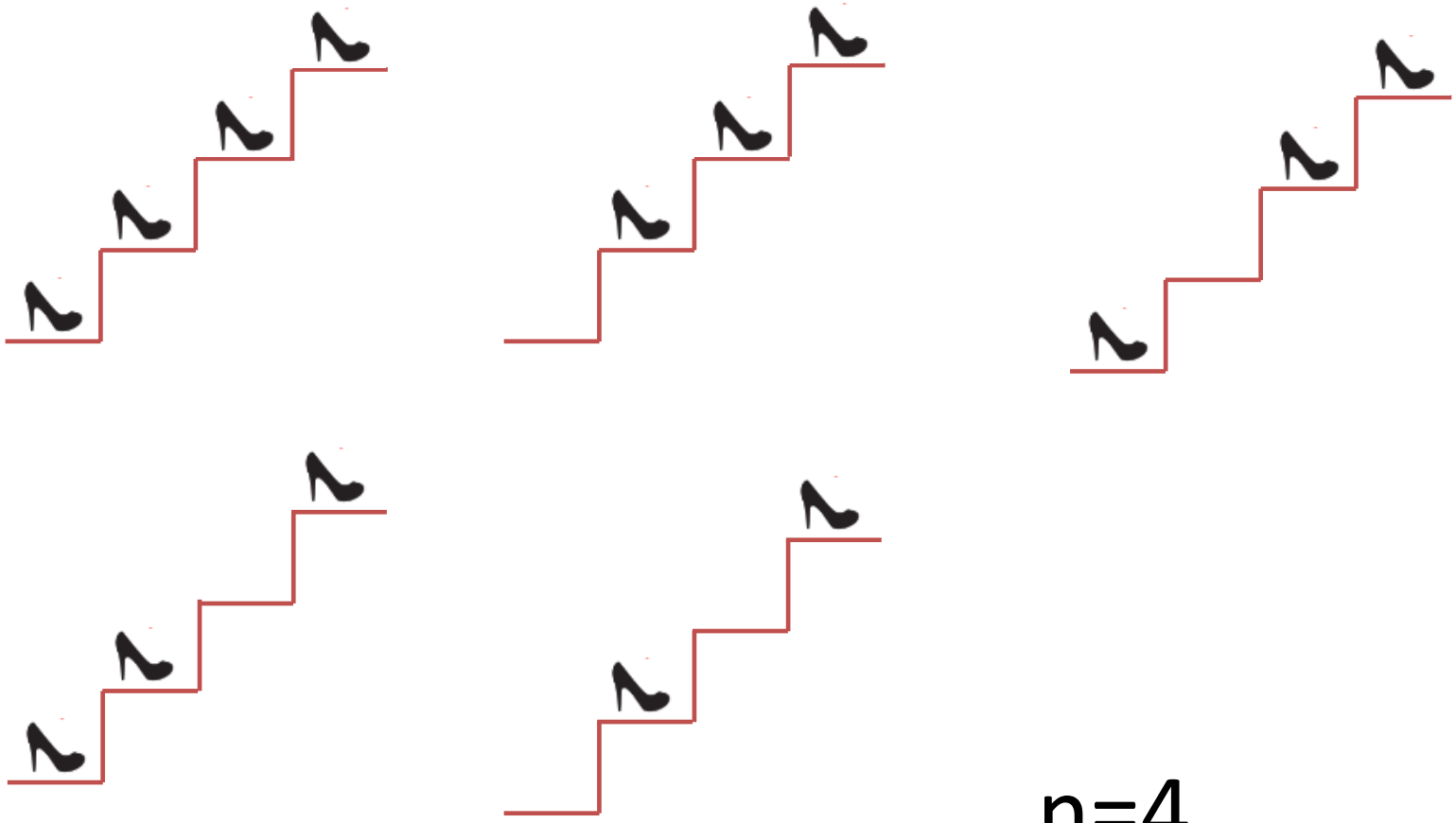
3. gyakorlat

```
public static long factorial(int n)
 {
      if (n > 1)
          return n * factorial(n - 1);
      else
          return 1;
 }
```

3. gyakorlat

Adjunk rekurzív algoritmust, ami meghatározza, hogy hányféleképpen mehetünk fel egy *n* lépcsőfokból álló lépcsőn, ha egyszerre csak 1 vagy 2 lépcsőfokot léphetünk!

3. gyakorlat



n=4

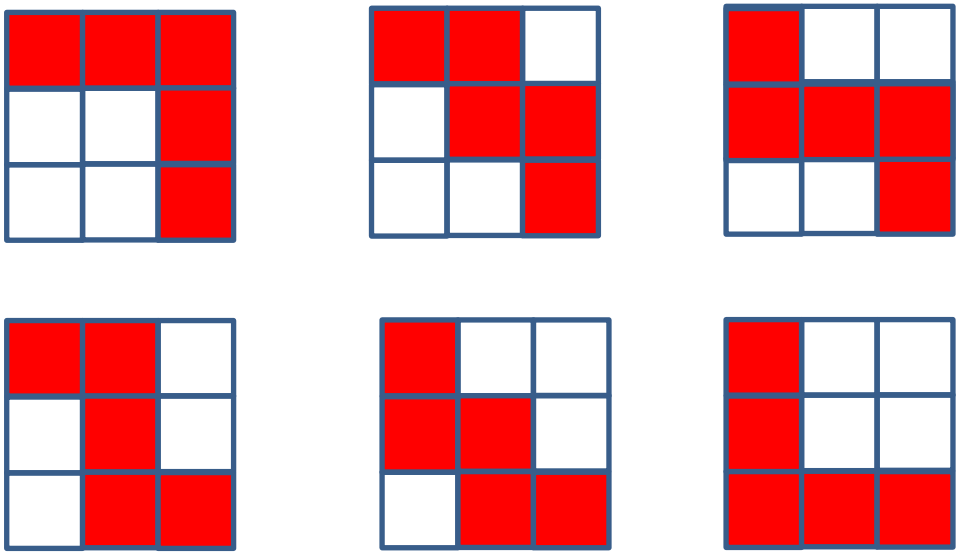
3. gyakorlat

```
public static int findStep(int n)
{
      if (n == 1 )
         return 1;
      else if (n == 2)
         return 2;
       else
         return findStep(n - 2) + findStep(n - 1);
}
```

3. gyakorlat

Adjunk rekurzív algoritmust, amely meghatározza, hogy hányféleképpen juthatunk el egy *k* sorból és *n* oszlopból álló sakktábla bal felső sarkából a jobb alsó sarkába, ha csak a jobbra vagy a lefelé szomszédos mezőre léphetünk!

3. gyakorlat



n=k=3

3. gyakorlat

```
static int numberOfPaths(int n, int k)
{
        if (n == 1 || k == 1)
                return 1;
         return numberOfPaths(n - 1, k) +
         numberOfPaths(n, k - 1);
}
```

3. gyakorlat

Hanoi tornyai probléma

Adott 3 rúd, az elsőn van n korong. Az első rúdról az utolsóra kell átrakni a korongokat úgy, hogy minden lépésben egy korongot lehet áttenni, nagyobb korong nem tehető kisebb korongra.

https://hu.wikipedia.org/wiki/Hanoi_tornyai

3. gyakorlat

Alapeset: n=1, a korongot áthelyezzük.
Rekurzív eset (n>1):
- a felső n-1 korongot félrerakjuk a szabályosan,
- a megmaradt 1 korongot áthelyezzük,
- a félrerakott n-1 korongot a helyükre tesszük.

A rudakat megszámozzuk: 1,2,3
Mi a segéd korong száma?
(1,2)->3, (1,3)->2, (2,3)->1
(i,j)->?

3. gyakorlat

```
static void towerOfHanoi(int n, int i, int j)
{
    if (n == 1)
    {
       System.out.println(i + " -> " + j);
       return;
    }
    towerOfHanoi(n-1,i, 6-(i+j));
    System.out.println(i + " -> " + j);
    towerOfHanoi(n-1, 6-(i+j),j);
 }
```

3. gyakorlat

Lépések száma:

$T(1)=1$

Ha $n>1$: $T(n)=2T(n-1)+1=2^2T(n-2)+2\cdot 1+1=\ldots$

$= 2^kT(n-k)+2^{k-1}+2^{k-2}+\ldots+1=\ldots$

$= 2^{n-1}T(1)+2^{n-2}+2^{n-3}+\ldots+2^0=\ldots$

$= 2^{n-1}+2^{n-2}+2^{n-3}+\ldots+1=2^n-1.$

Futási idő: $O(2^n)$

3. gyakorlat

Hilbert görbe

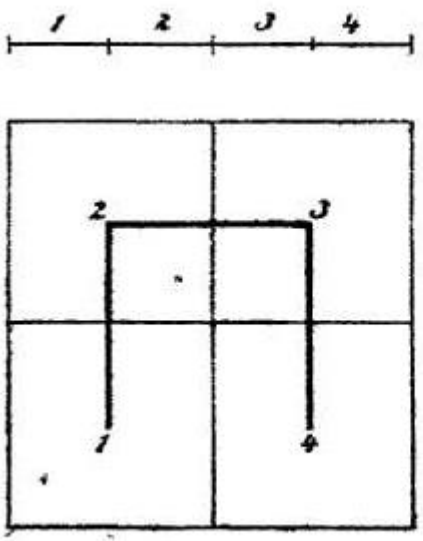


Fig. 1.

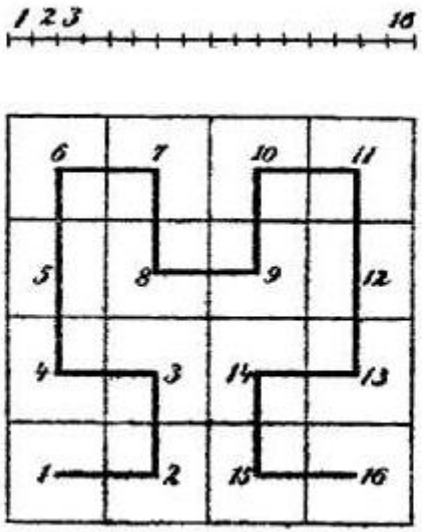


Fig. 2.

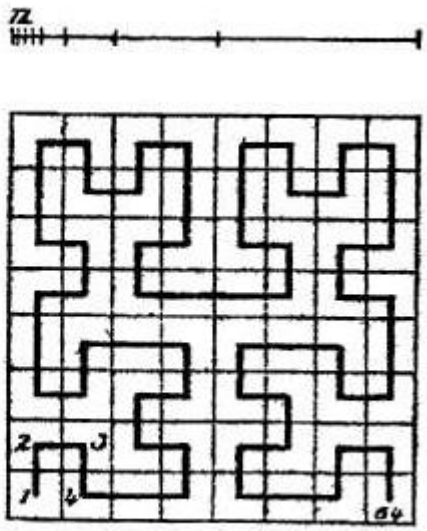


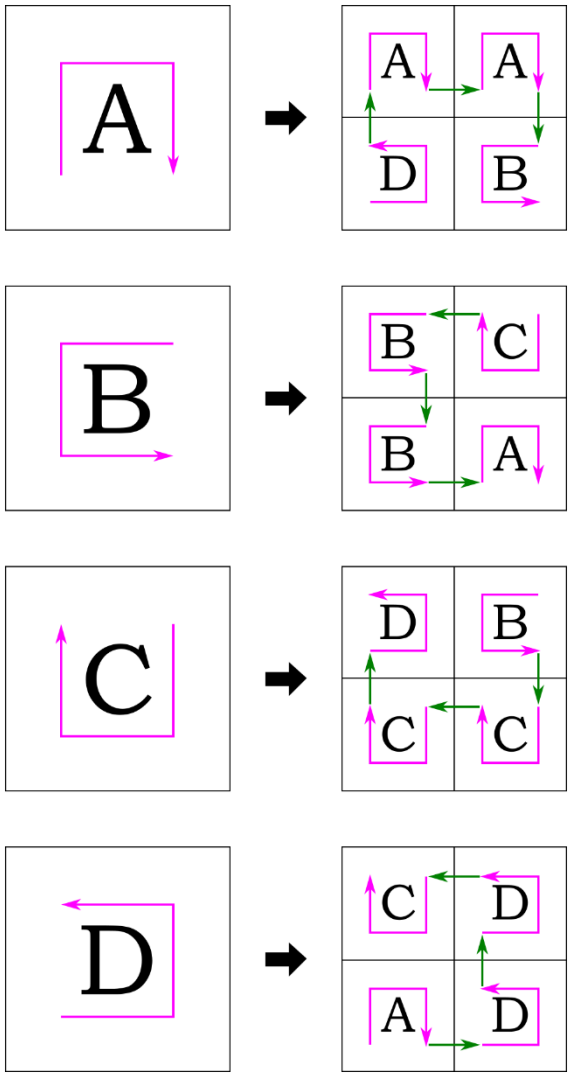
Fig. 3.

Forrás: D. Hilbert: Über die stetige Abbildung einer Linie auf ein Flächenstück. Mathematische Annalen 38 (1891), 459–460.

3. gyakorlat

Hilbert görbe rekurzív definíciója.

https://en.wikipedia.org/wiki/Hilbert_curve

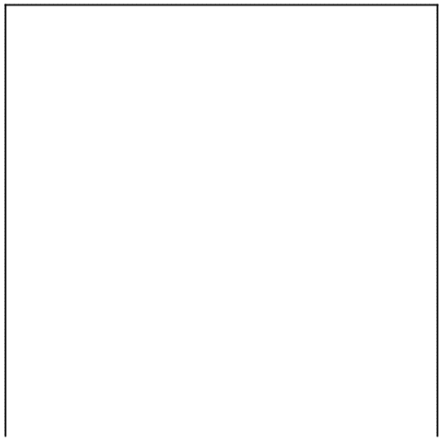


3. gyakorlat

```
public static void A(int n)
  {
    if (n > 0) {
        D(n-1); forward(1,0,1); //Y := Y + H; LINE;
        A(n-1); forward(1,1,0);  //X := X + H; LINE;
        A(n-1); forward(1,0,-1);  //Y := Y - H; LINE;
        B(n-1);
    }
  };

  public static void B(int n)
  {
    if (n > 0) {
        C(n-1); forward(1,-1,0); //X := X - H; LINE;
        B(n-1); forward(1,0,-1);  //Y := Y - H; LINE;
        B(n-1); forward(1,1,0);  //X := X + H; LINE;
        A(n-1);
    }
  };
```

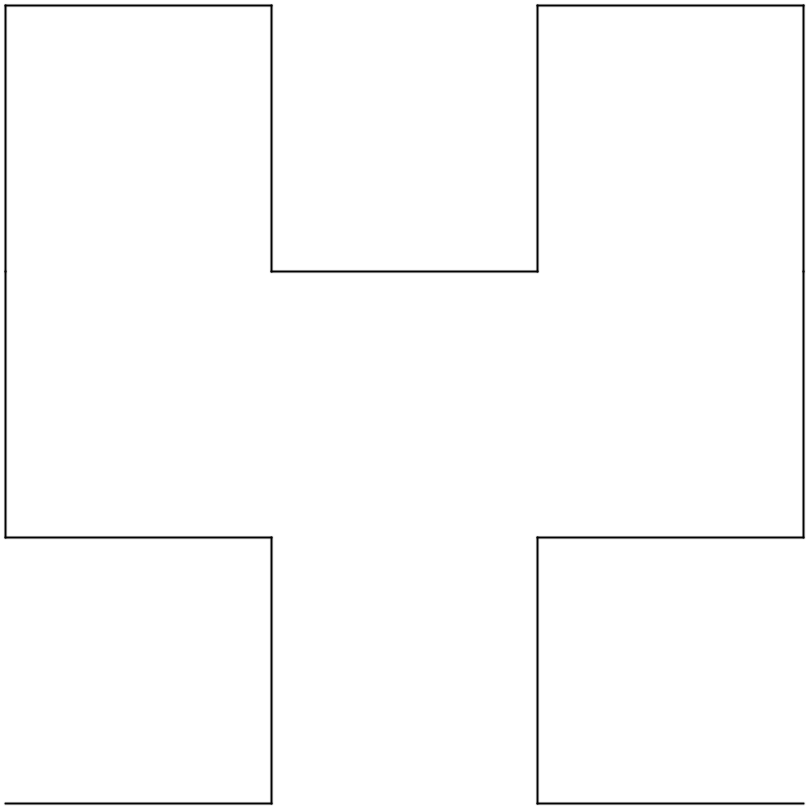
```
public static void C(int n)
  {
    if (n > 0) {
        B(n-1); forward(1,0,-1); //Y := Y - H; LINE;
        C(n-1); forward(1,-1,0);  //X := X - H; LINE;
        C(n-1); forward(1,0,1);  //Y := Y + H; LINE;
        D(n-1);
    }
  };

  public static void D(int n)
  {
    if (n > 0) {
        A(n-1); forward(1,1,0); //X := X + H; LINE;
        D(n-1); forward(1,0,1);  //Y := Y + H; LINE;
        D(n-1); forward(1,-1,0);  //X := X - H; LINE;
        C(n-1);
    }
  };
```

3. gyakorlat



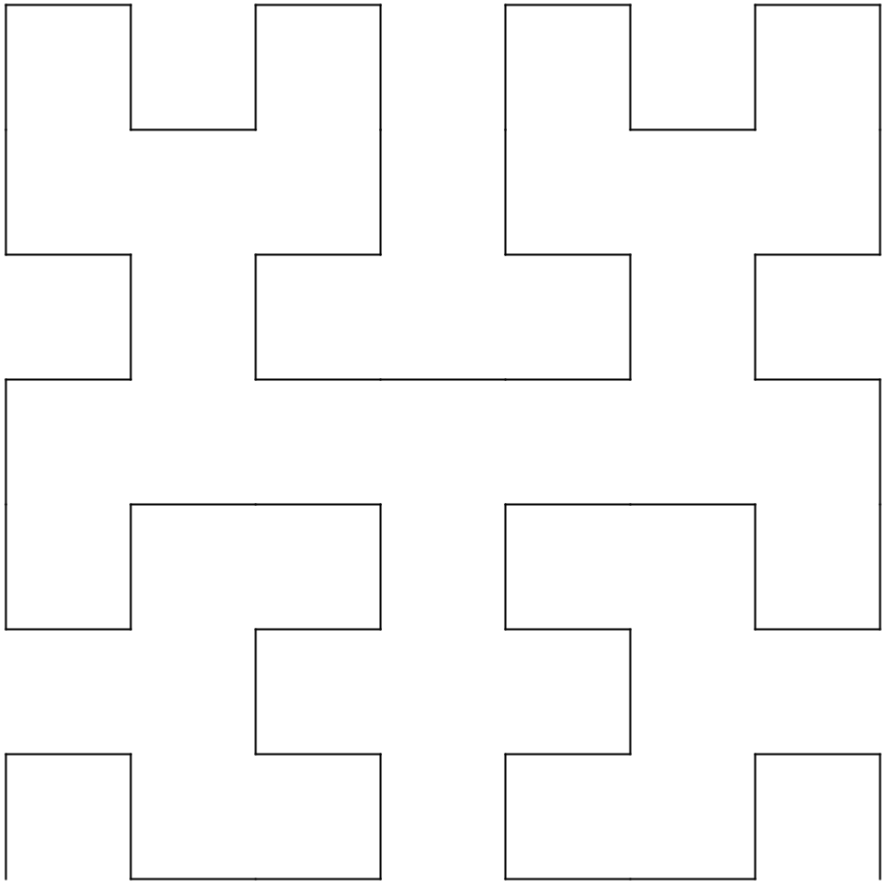
n=1

3. gyakorlat



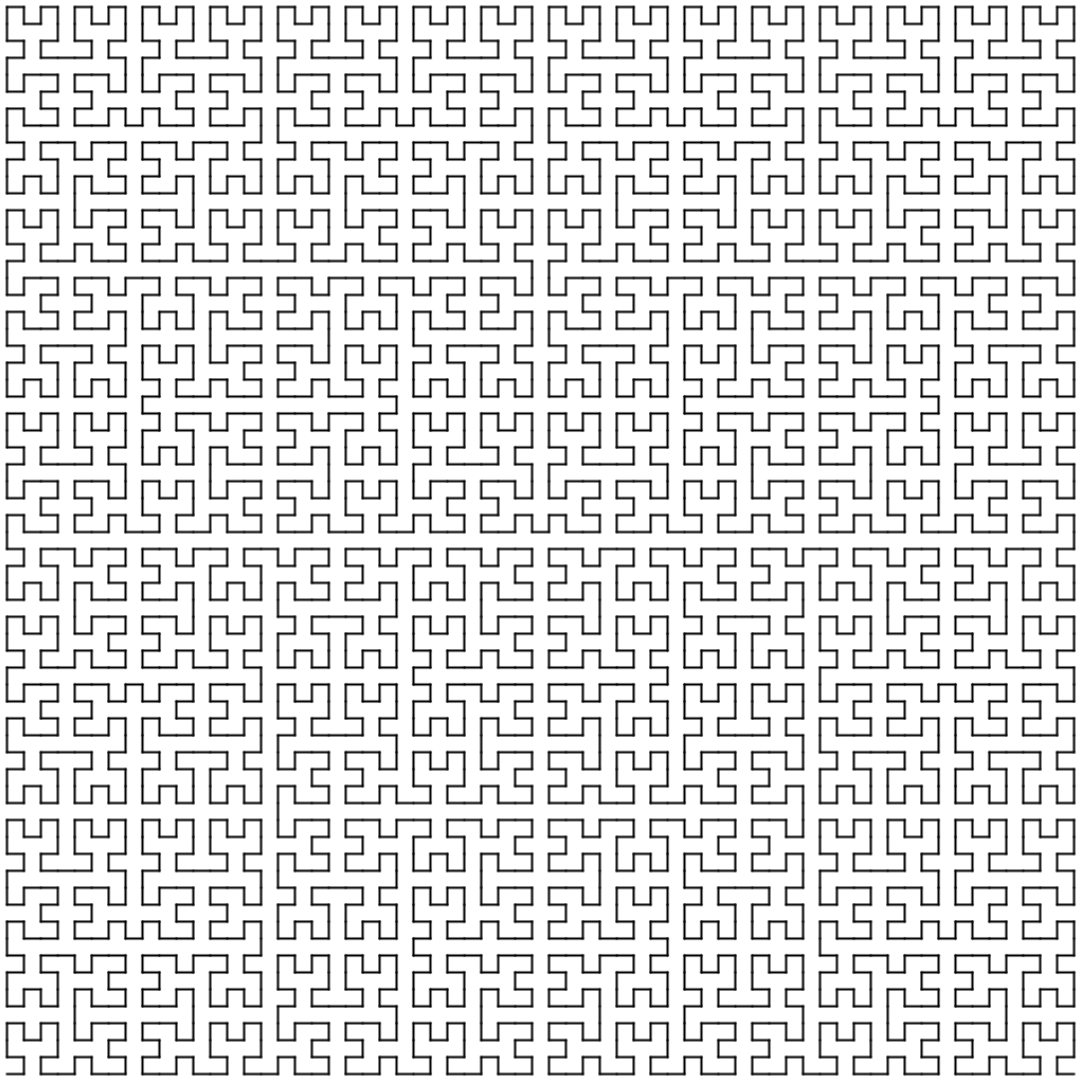
n=2

3. gyakorlat



n=3

3. gyakorlat



n=6

3. gyakorlat

$T(1)=3$

Ha $n>1$: $T(n)=4T(n-1)+3=4^2T(n-2)+4\cdot 3+3=\ldots$

$= 4^kT(n-k)+4^{k-1}\cdot 3+4^{k-2}\cdot 3+\ldots+4^0\cdot 3=\ldots$

$= 4^{n-1}T(1)+4^{n-2}\cdot 3+4^{n-3}\cdot 3+\ldots+4^0\cdot 3=\ldots$

$= (4^{n-1}+4^{n-2}+4^{n-3}+\ldots+1)\cdot 3=4^n-1.$

Futási idő: $O(4^n)$

Mivel egy szakasz hossza $1/2^{n-1}$, a görbe teljes hossza $2^{n+1} - 1/2^{n-1}$.

3. gyakorlat

Adjunk meg egy rekurzív algoritmust, amely meghatározza az *n* szám azon partícióinak számát, amelyben a partícióban szereplő számok páronként különböznek!

Házi feladat

3. gyakorlat

- Magyarázatok, megoldások:
- Gelle Kitti: Algoritmusok és adatszerkezetek I. gyakorlati jegyzet.
- http://www.inf.u-szeged.hu/~kgelle/sites/default/files/upload/alga-gyak-03.pdf