Dinamikus programozás

Konzultáció

4. alkalom

Recap - rekurzió

- Rekurzió: egy eljárást rekurzívnak nevezünk, ha benne előfordul saját magának a hívása
  - Hogy elkerüljük a végtelen ciklust, definiálni kell egy **bázis esetet**
  - Egy feladat megoldásához mindig meg kell találnunk a **rekurzív összefüggést**

Pl.:

- Írj (rekurzív) programot amely kiszámítja a megadott szám faktoriálisát!
- Írj (rekurzív) programot amely kiszámítja az n-edik fibonacci számot!
- Írj (rekurzív) programot amely kiszámítja hányféleképpen mehetünk fel egy n lépcsőfokból álló lépcsőn, ha egyszerre csak 1 vagy 2 lépcsőfokot léphetünk!

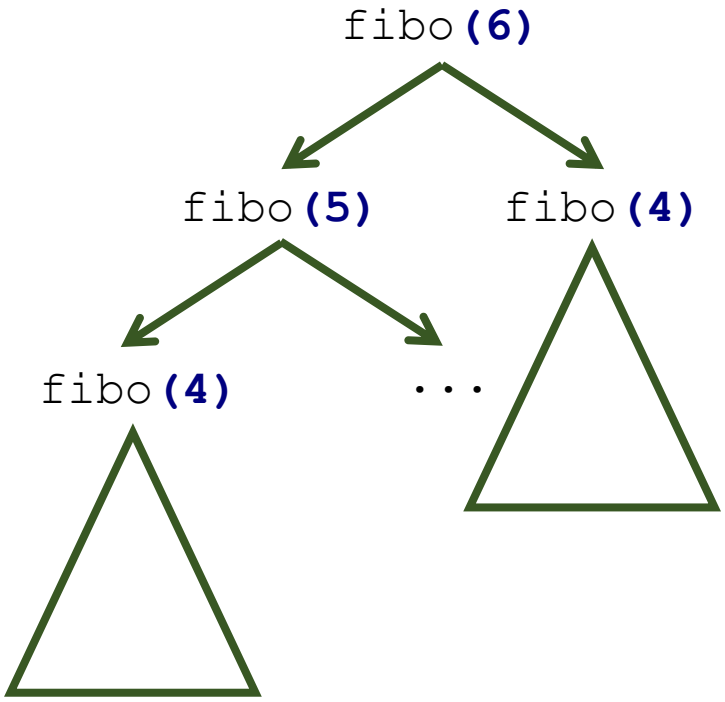
Rekurzió – sebesség I

- **Imperatív** programozási nyelvekben (mint amilyen a C/C++, Java, Python, …) a függvényhívás alapvetően egy költséges művelet (verem kezelés, változók átadása, …)
- **Funkcionális** programozási nyelvekben (mint amilyen a Haskell, Mercury, …) azonban nem is nagyon van más választás, ezért a függvényhívás alsóbb szinteken máshogy van megvalósítva, optimalizálva van. Például a faktoriálist kiszámító program Haskell-ben:

```
fakt :: Int -> Int
fakt n | n == 0 = 1
       | n > 0 = n * fakt(n - 1)
```

Rekurzió – sebesség II

- A másik gyengesége, hogy egy értéket többször is kiszámítunk, például a Fibonacci-s programban:



- Vegyük észre, a háromszöggel jelölt részekben ugyanazt számoljuk ki kétszer!

Rekurzió memorizálás

- Az alapötlet az, hogy ha már megvan egy értékünk, azt mentsük le, és ha kell, akkor onnan csak elővesszük

```
fibo(n) {
    if ( n == 0 ) {
        F[0] = 0 return F[0]
    }
    if ( n == 1 ) {
         F[1] = 1 return F[1]
    }
    if ( F[n] != -1 ) {
        return F[n];
    }
    else {
        F[n] =    fibo(n-1) +
                  fibo(n-2)
        return F[n];
    }
}
```

```
fibo(n) {
    if ( n == 0 ) {
        return 0
    }
    if ( n == 1 ) {
        return 1
    }
    else {
        return    fibo(n-1) +
                  fibo(n-2)
    }
}
```

Dinamikus programozás

- Az előzővel az értékek duplán kiszámolását már helyre tettük, már csak a függvényhívás lassúságán kellene gyorsítani.
- Itt pedig az az ötlet, hogy: „ha kell egy érték, azért nem saját magam fogom meghívni, hanem azt kiveszem egy tömbből”
- Ezt általában akkor érdemes használni, ha egy rekurzív algoritmus ismételten visszatér ugyanazokra a feladatokra, vagy a probléma egy optimális megoldása önmagán belül tárolja a további részfeladatok optimális megoldásait

Dinamikus programozás - Fibonacci

```
fibo(n) {
    F[0] = 0
    F[1] = 1
    for( i = 2 to n ) {
        F[i] =   F[i-1] +
                 F[i-2]
    }
    return T[n]
}
```

```
fibo(n) {
    if ( n == 0 ) {
        return 0
    }
    if ( n == 1 ) {
        return 1
    }
    else {
        return   fibo(n-1) +
                 fibo(n-2)
    }
}
```

- A ciklussal pedig a rekurzív hívásokat „szimuláljuk”

Feladatok - átírásos

- Írjuk át az eddigi rekurzív programokat dinamikus programozással!

1. Írj (dinamikusan) programot amely kiszámítja a megadott szám faktoriálisát!
2. Írj (dinamikusan) programot amely kiszámítja hányféleképpen mehetünk fel egy n lépcsőfokból álló lépcsőn, ha egyszerre csak 1 vagy 2 lépcsőfokot léphetünk!

+ 1. Hány nyúlpár születik n év múlva, ha feltesszük, hogy:

- először csak egy nyúlpár van
- minden nyúlpár amikor szaporodik egy újabb nyúlpárt hoz létre
- a nyulak a születésók után a következő 2 évben egyszer-egyszer szaporodnak

Feladat – gondolkozós I

- Írj programot, ami megkeresi egy tömbben a legnagyobb elemet!
- list = [2, 4, 9, 1] -> max(list) = 9

Megoldás - rekurzív

```
//n == length(list) - 1
max(list, n) {
   max1 = list[n]
   if ( n == 1 ) {
      max2 = list[n-1]
   }
   else {
      max2 = max(list, n-1)
   }
   if ( max1 < max2 ) {
      return max2;
   }
   return max1;
}
```

Megoldás - dinamikus

```
//n == length(list) - 1
max(list, n) {
   max1 = list[n]
   for( i = 0 to n-1 ) {
      max2 = list[i]
      if ( max1 < max2 ) {
         max1 = max2;
      }
   }
   return max1;
}
```

Feladat – gondolkozós II

- A sudoku egy logikai játék, melyben megadott szabályok szerint számjegyeket kell elhelyezni egy táblázatban.

Forrás: Wikipédia

	3							
			1	9	5			
		8					6	
8				6				
4			8					1
				2				
	6					2	8	
			4	1	9			5
							7	

→

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

- Írj programot, ami kitölti a sudoku dobozt!
  (megoldás következő alkalommal)