Rekurzió

(Horváth Gyula és Szlávi Péter előadásai felhasználásával)

Rekurzió

Klasszikus példák

- Faktoriális

$$n! = \begin{cases} n*(n-1)! & ha \;\; n > 0 \\ 1 & ha \;\; n = 0 \end{cases}$$

- Fibonacci-számok

$$Fib(n) = \begin{cases} 0 & ha \;\; n = 0 \\ 1 & ha \;\; n = 1 \\ Fib(n-1) + Fib(n-2) & ha \;\; n > 1 \end{cases}$$

A rekurzió lényege: önhivatkozás

Rekurzió

Klasszikus példák

- Binomiális számok:

$$B(n,k)=\begin{cases}1 & ha\ \ k=0\\ B(n-1,k)+B(n-1,k-1) & ha\ \ 0<k<n\\ 1 & ha\ \ k=n\end{cases}$$

$$B(n,k)=\begin{cases}1 & ha\ \ k=0\\ B(n,k-1)*\dfrac{n-k+1}{k} & ha\ \ 0<k\le n\end{cases}$$

- McCarthy-féle 91-es függvény:

$$M(n)=\begin{cases}n-10 & ha\quad n>100\\ M(M(n+11)) & ha\quad n\le 100\end{cases}$$

Értéke 91 lesz, minden 100-nál kisebb n-re.

Rekurzió

Klasszikus példák

- Ackermann-függvény:

$$A(n,m)=\begin{cases} m+1 & ha\ \ n=0 \\ A(n-1,1) & ha\ \ n>0\ \ és\ \ m=0 \\ A(n-1,A(n,m-1)) & ha\ \ n>0\ \ és\ \ m>0 \end{cases}$$

Elvadultságáról:

- A(0,m)=m+1
- A(1,m)=m+2
- A(2,m)=2*m+3
- $A(3,m)=2^{m+3}-3$
- $A(4,m)=2^{\alpha}-3$, ahol az α a 2-t m+2-ször tartalmazza kitevőként.

$$A(4,m)=2^{2^{\cdots^{2}}}-3$$

Rekurzív specifikáció és algoritmus

Faktoriális:

$$n! = \begin{cases} n*(n-1)! & ha \ \ n>0 \\ 1 & ha \ \ n=0 \end{cases}$$

```
Fakt(n):
   Ha n>0 akkor Fakt:=n*Fakt(n-1)
   különben Fakt:=1
Eljárás vége.
```

Rekurzív specifikáció és algoritmus

Fibonacci számok:

$$Fib(n) = \begin{cases} 0 & ha\ \ n = 0 \\ 1 & ha\ \ n = 1 \\ Fib(n-1) + Fib(n-2) & ha\ \ n > 1 \end{cases}$$

```
Fib(n):
   Ha n=0 akkor Fib:=0
   különben ha n=1 akkor Fib:=1
   különben Fib:=Fib(n-1)+Fib(n-2)
Eljárás vége.
```

Lame számok: Lame(n)=Lame(n-1)+Lame(n-3)

Q számok: Q(n)=Q(n-Q(n-1))+Q(n-Q(n-2))

Rekurzív specifikáció és algoritmus

Binomiális számok:

$$B(n,k) = \begin{cases} 1 & ha\ \ k = 0 \\ B(n-1,k) + B(n-1,k-1) & ha\ \ 0 < k < n \\ 1 & ha\ \ k = n \end{cases}$$

```
Bin(n,k):
  Ha k=0 vagy k=n akkor Bin:=1
  különben Bin:=Bin(n-1,k)+Bin(n-1,k-1)
Eljárás vége.
```

$$B(n,k) = \begin{cases} 1 & ha\ \ k = 0 \\ B(n,k-1) * \dfrac{n-k+1}{k} & ha\ \ 0 < k \le n \end{cases}$$

```
Bin(n,k):
  Ha k=0 akkor Bin:=1
  különben Bin:=Bin(n,k-1)*(n-k+1)/k
Eljárás vége.
```

Rekurzív specifikáció és algoritmus

McCarthy 91-függvény:

$$M(n) = \begin{cases} n-10 & ha \quad n > 100 \\ M(M(n+11)) & ha \quad n \leq 100 \end{cases}$$

```
M(n):
  Ha n>100 akkor M:=n-10 különben M:=M(M(n+11))
Eljárás vége.
```

```
M(n):
  Ha n>100 akkor M:=n-10
        különben x:=M(n+11); M:=M(x)
Eljárás vége.
```

Tehát a dupla rekurzió algoritmikus szinten nem okoz semmilyen gondot!

Rekurzív specifikáció és algoritmus

Ackermann függvény:

$$A(n,m) = \begin{cases} m+1 & ha\ \ n=0 \\ A(n-1,1) & ha\ \ n>0\ \ és\ \ m=0 \\ A(n-1,A(n,m-1)) & ha\ \ n>0\ \ és\ \ m>0 \end{cases}$$

```
Ack(n,m):
   Ha n=0 akkor Ack:=m+1
   különben ha m=0 akkor Ack:=Ack(n-1,1)
   különben Ack:=Ack(n-1,Ack(n,m-1))
Eljárás vége.
```

Tehát a dupla rekurzió algoritmikus szinten nem okoz semmilyen gondot!

Közvetett rekurzió

Feladat

Döntsük el egy számról, hogy páros-e, ha nincs maradék-számítás műveletünk!

Megoldás

```
Páros(n):
  Ha n=0 akkor Páros:=igaz
  különben ha n=1 akkor Páros:=hamis
       különben Páros:=Páratlan(n-1)
Függvény vége.
```

Közvetett rekurzió

Feladat

Döntsük el egy számról, hogy páros-e, ha nincs maradék-számítás műveletünk!

Megoldás

```
Páratlan(n):
  Ha n=0 akkor Páratlan:=hamis
  különben ha n=1 akkor Páratlan:=igaz
       különben Páratlan:=Páros(n-1)
Függvény vége.
```

Közvetett rekurzió

Feladat

Döntsük el egy számról, hogy páros-e, ha nincs maradék-számítás műveletünk!

A két – közvetetten – rekurzív eljárás összevonható:

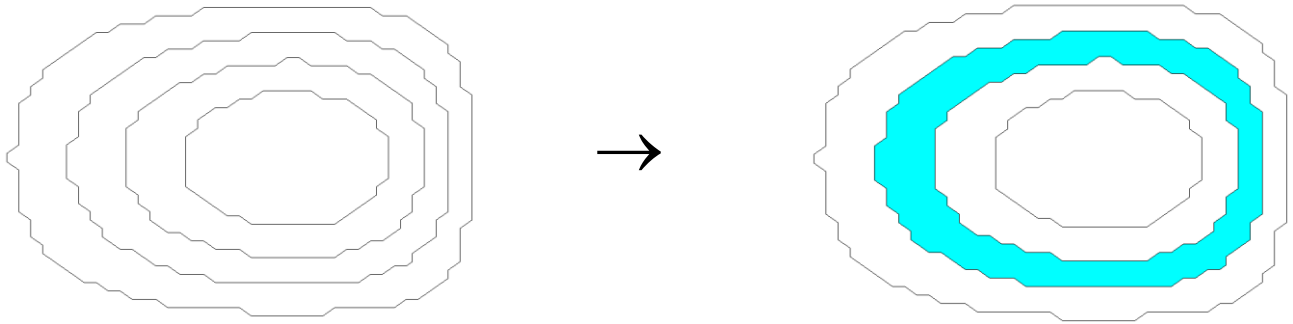
Megoldás

```
Páros(n):
   Ha n=0 akkor Páros:=igaz
   különben ha n=1 akkor Páros:=hamis
   különben Páros:=Páros(n-2)
Függvény vége.
```

Rekurzív eljárás

Feladat:

Egy kép egy adott (fehér színű) tartományát egy (A,B) belső pontjából kiindulva fessük be világoskékre!



Festendők a „belső pontok”, azaz

Belső(i,j)=(i=A és j=B) vagy
Fehér(i,j) és (Belső(i–1,j) vagy Belső(i+1,j) vagy
Belső(i,j–1) vagy Belső(i,j+1))

Rekurzív eljárás

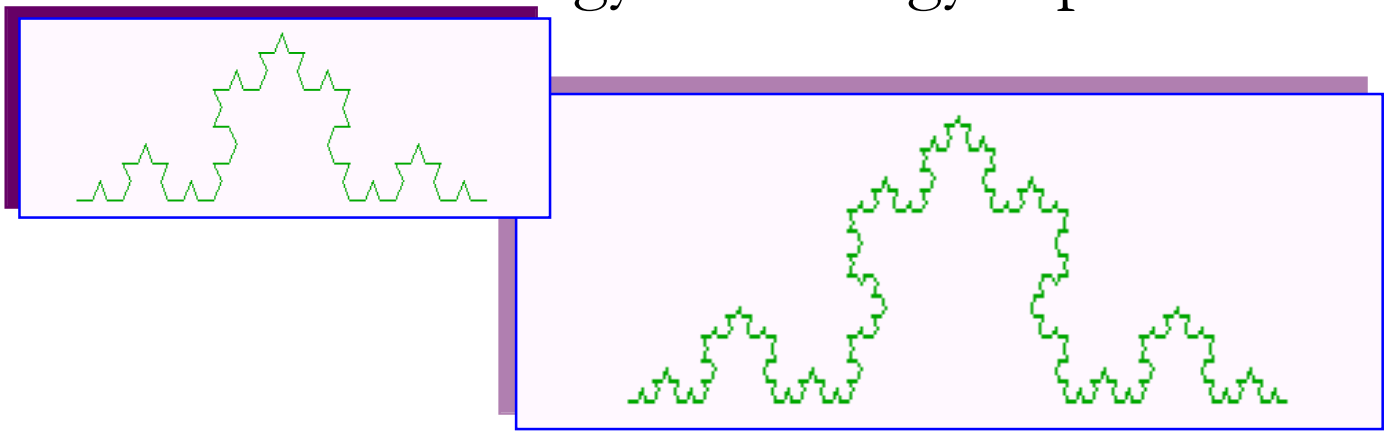
Rekurzív festés pontonként:

```
RekPont(x,y):
   Pont(x,y)
   Ha Üres(x-1,y) akkor RekPont(x-1,y)
   Ha Üres(x,y-1) akkor RekPont(x,y-1)
   Ha Üres(x+1,y) akkor RekPont(x+1,y)
   Ha Üres(x,y+1) akkor RekPont(x,y+1)
Eljárás vége.
```

Rekurzió Logo nyelven

Koch fraktál:

- Vegyünk egy egységnyi szakaszt!
- Vágjuk ki a középső harmadát!
- Illesszük be a kivágott részt egy egyenlő oldalú háromszög oldalaiként!
- Alkalmazzuk ugyanezt az így kapott 4 szakaszra!

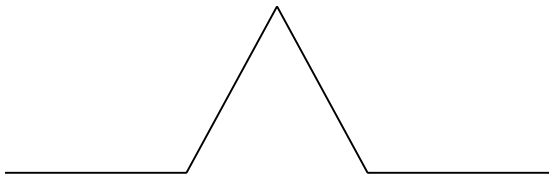


Rekurzió Logo nyelven

Koch fraktál, Logo megoldás:

```
eljárás koch :n :h
  Ha :n=0 [előre :h]
  [koch :n-1 :h/3 balra 60
   koch :n-1 :h/3 jobbra 120
   koch :n-1 :h/3 balra 60
   koch :n-1 :h/3]
vége
```


koch 1 100



koch 2 100

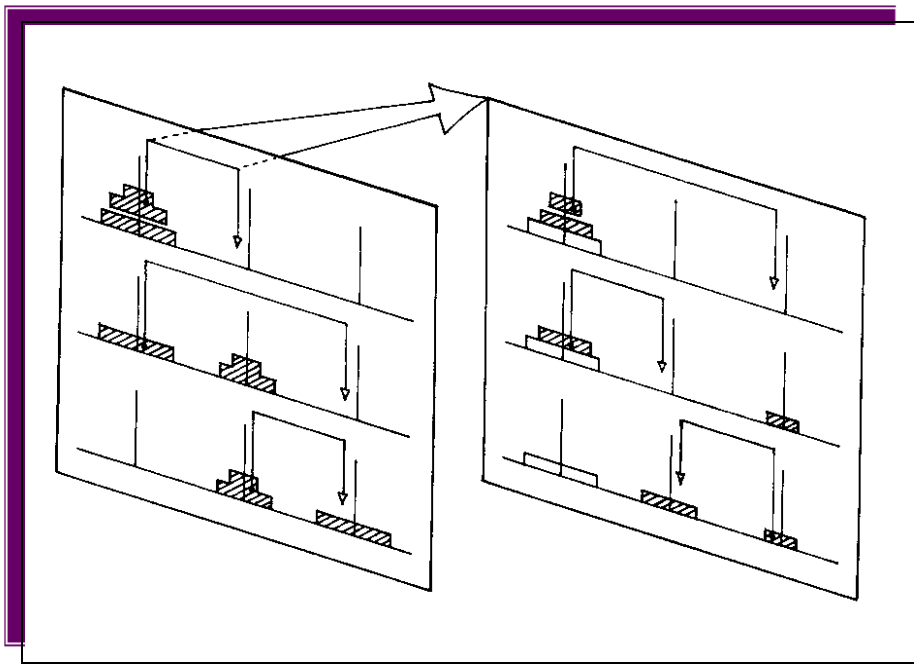
Rekurzív eljárás

Hanoi tornyai:

Adott 3 rudacska. Az elsőn egyre csökkenő sugarú korongok vannak. Az a feladat, hogy tegyük át a harmadik rudacskára a korongokat egyenként úgy, hogy az átpakolás közben és természetesen a végén is minden egyes korongon csak nála kisebb lehet. Az átpakoláshoz lehet segítségül felhasználni a középső rudacs-kát.

Rekurzív eljárás

Hanoi tornyai:



Rekurzív eljárás

Hanoi tornyai:

```
Hanoi(n,A,B,C):
  Ha n>0 akkor Hanoi(n-1,A,C,B)
               Ki: N,A,B
               Hanoi(n-1,C,B,A)
  Elágazás vége
Eljárás vége.
```

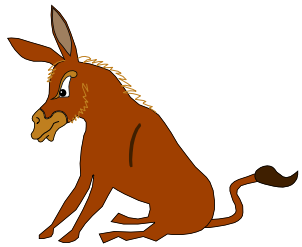
```
Hanoi(n,A,B,C):
  Ha n>1 akkor Hanoi(n-1,A,C,B)
               Ki: n,A,B
               Hanoi(n-1,C,B,A)
       különben Ki: n,A,B
Eljárás vége.
```

A megvalósítás problémái

Problémák

„Állatorvosi ló (-: csacsi :-)” – a faktoriális függvény

```
Fakt(n):
  Ha n=0 akkor  f:=1
         különben f:=n*Fakt(n-1)
 Fakt:=f
Függvény vége.
```



Tételezzük föl, hogy a „környezet” képes az eljárások *többszörös* (értsd: egymásba ágyazott) *hívás*ának lekezelésére.
(*Visszatérési cím a verembe; aktuális és formális paraméterek összekapcsolása.*)

A megvalósítás problémái

Összefoglalva a problémákat:

1. A *bemenő paraméter* problematikája.

Hogyan kerül a bemenő érték ugyanazzal a kóddal megvalósított eljáráshoz, azaz a rekurzívan újból a hívotthoz?

2. A *függvényeljárások értékvisszaadása*.

Hogyan kerül a (rekurzívan) hívott függvény értéke a hívó eljárásban „felszínre”?

3. A *lokális változó*k *egyedisége*.

A megvalósítás problémái

A megoldás vázlata:

1. és 3. probléma – belépéskor *verme*lés, kilépéskor *verem*ből kivétel.

2. probléma (nem csak rekurziónál) – az érték *verem*be tétele közvetlenül a visszatérés előtt.

A függvényt *eljárás*sá alakítva:

```
Fakt(n):
  Verembe(n)
  Ha n=0 akkor f:=1
  különben Fakt(n-1); Veremből(f)
           n:=Veremtető; f:=n*f
  Veremből(n); Verembe(f)
Eljárás vége.
```

A megvalósítás problémái

Általános séma a rekurzió végrehajtására:

```
Rekurzív(paraméterek):
   …
   Rekurzív(paraméterek)
   …
Eljárás vége.
Rekurzív(paraméterek):
   …
   Verembe(bemenő paraméterek, lokális változók)
   Rekurzív(paraméterek)
   Veremből(bemenő paraméterek, lokális változók)
   …
Eljárás vége.
```

A megvalósítás problémái

Megjegyzés: Ha a bemenő paramétereknek és a lokális változóknak a fordítóprogram eleve a veremben foglal helyet, akkor nincs vermelési idő.

Rekurzív függvények esete:

```
Rekfgv(paraméterek):
   …
   Verembe(bemenő paraméterek, lokális változók)
   Rekfgv(paraméterek); Veremből(érték)
   Veremből(bemenő paraméterek, lokális változók)
   …
   Verembe(érték)
Eljárás vége.
```

Problémák a rekurzióval

Bajok a rekurzióval

Hely: nagyra dagadt veremméret.

Idő:a vermelés adminisztrációs többletterhe, a többszörösen ismétlődő hívások.

Pl. Fibonacci-számoknál:

r(N):=az N. Fibonacci-szám kiszámításához szükséges hívások száma

r(0):=1, r(1):=1, r(i):=r(i-1)+r(i-2)+1

Állítás:

a) r(i)=F(i+1)+F(i)+F(i-1)-1 i>1

*b) r(i)=2*F(i+1)-1, ahol F(i)=az i. Fibonacci-szám.*

Korlátos memóriájú függvények

Korlátos memóriájú függvények

Ha egy rekurzív függvény minden értéke valamely korábban kiszámolható értékből számolható, akkor némi memória-felhasználással elkészíthető a rekurzió mentes változat, amelyben az egyes függvényértékeknek megfeleltetünk egy F(N) vektort.

A függvény általános formája:

$$f(i)=\begin{cases} g(f(i-1), f(i-2),\ldots, f(i-K)) & ha \quad i \geq K \\ h(i) & ha \quad 0 \leq i < K \end{cases}$$

Korlátos memóriájú függvények

Korlátos memóriájú függvények

```
f(N):
   Ha N<K akkor f:=h(N)
   különben f:=g(f(N-1),...,f(N-K))
Függvény vége.
```

Az ennek megfelelő vektoros változat:

```
f(N):
   Ciklus I=0-tól K-1-ig
      F(I):=h(I)
   Ciklus vége
   Ciklus I=K-tól N-ig
      F(I):=g(F(I-1),...,F(I-K))
   Ciklus vége
   f:=F(N)
Függvény vége.
```

Korlátos memóriájú függvények

Korlátos memóriájú függvények

Ez így természetesen nem hatékony tárolás, hiszen a rekurzív formulából látszik, hogy minden értékhez csak az őt megelőző K értékre van szükség.

A hatékony megoldásban az alábbi ciklust kell átalakítani:

```
Ciklus I=K-tól N-ig
  F(I):=g(F(I-1),...,F(I-K))
Ciklus vége
```

Lehet pl. `F(I mod K):=g(F(K-1),...,F(0))`, ha a g() függvény kiszámítása nem függ a paraméter sorrendtől.

Korlátos memóriájú függvények

Példa: Fibonacci-számok

$$Fib(n) = \begin{cases} 0 & ha \;\; n = 0 \\ 1 & ha \;\; n = 1 \\ Fib(n-1) + Fib(n-2) & ha \;\; n > 1 \end{cases}$$

```
Fib(n):
  Ha n=0 akkor Fib:=0
  különben ha n=1 akkor Fib:=1
  különben Fib:=Fib(n-1)+Fib(n-2)
Eljárás vége.
```

```
Fib(n):
  F(0):=0; F(1):=1
  Ciklus i=2-től n-ig
    F(i):=F(i-1)+F(i-2)
  Ciklus vége
  Fib:=F(n)
Függvény vége.
```

Korlátos memóriájú függvények

Példa: Fibonacci-számok

$$Fib(n) = \begin{cases} 0 & ha \;\; n = 0 \\ 1 & ha \;\; n = 1 \\ Fib(n-1) + Fib(n-2) & ha \;\; n > 1 \end{cases}$$

Helytakarékos megoldás:

```
Fib(n):
  F(0):=0; F(1):=1
  Ciklus i=2-től n-ig
    F(i mod 2):=F(0)+F(1)
  Ciklus vége
  Fib:=F(n mod 2)
Függvény vége.
```

Rekurzió memorizálással

Megoldási ötlet: amit már kiszámoltunk egyszer, azt ne számoljuk újra! Tároljuk a már kiszámolt értékeket, s ha szükségünk van rájuk, használjuk fel őket!

A megoldásban F(i)≥0 jelenti, ha már kiszámoltuk az i-edik Fibonacci számot.

```
Fib(N):
   Ha F(N)<0 akkor ha N<2 akkor F(N):=N
                           különben F(N):=Fib(N-1)+Fib(N-2)
   Fib:=F(N)
Függvény vége.
```

Rekurzió memorizálással

Binomiális együtthatók számolása.

```
Bin(N,K):
  Ha N=0 vagy N=K akkor Bin:=1
  különben Bin:=Bin(N-1,K-1)+Bin(N-1,K)
Függvény vége.
```

Ugyanez memorizálással:

```
Bin(N,K):
  Ha B(N,K)<0 akkor
    Ha N=0 vagy N=K akkor B(N,K):=1
    különben B(N,K):=Bin(N-1,K-1)+Bin(N-1,K)
  Elágazás vége
  Bin:=B(N,K)
Függvény vége.
```

Oszd meg és uralkodj!

Oszd meg és uralkodj!

Több részfeladatra bontás, amelyek hasonlóan oldhatók meg, lépései:

- a triviális eset (amikor nincs rekurzív hívás)
- felosztás (megadjuk a részfeladatokat, amikre a feladat lebontható)
- uralkodás (rekurzívan megoldjuk az egyes részfeladatokat)
- összevonás (az egyes részfeladatok megoldásából előállítjuk az eredeti feladat megoldását)

Oszd meg és uralkodj!

Ezek alapján a következőképpen fogunk gondolkodni:

- Mi az általános feladat alakja? Mik a paraméterei? Ebből kapjuk meg a rekurzív eljárásunk specifikációját.
- Milyen paraméterértékekre kapjuk a konkrét feladatot? Ezekre fogjuk meghívni kezdetben az eljárást!
- Mi a leállás (triviális eset) feltétele? Hogyan oldható meg ilyenkor a feladat?
- Hogyan vezethető vissza a feladat hasonló, de egyszerűbb részfeladatokra? Hány részfeladatra vezethető vissza?

Oszd meg és uralkodj!

- Melyek ilyenkor az általános feladat részfeladatainak a paraméterei? Ezekkel kell majd meghívni a rekurzív eljárást!
- Hogyan építhető fel a részfeladatok megoldásaiból az általános feladat megoldása?

Oszd meg és uralkodj!

Gyorsrendezés (quicksort):

- felbontás: $\boxed{X_1, \ldots, X_{k-1}}\, X_k \,\boxed{X_{k+1}, \ldots, X_n}$ szétválogatás

  ahol $\forall i,j\ (1 \le i < k;\ k < j \le n)$: $X_i \le X_j$

- uralkodás: mindkét részt ugyanazzal a módszerrel felbontjuk két részre, rekurzívan
- összevonás: automatikusan történik a helyben szétválogatás miatt
- triviális eset: $n \le 1$

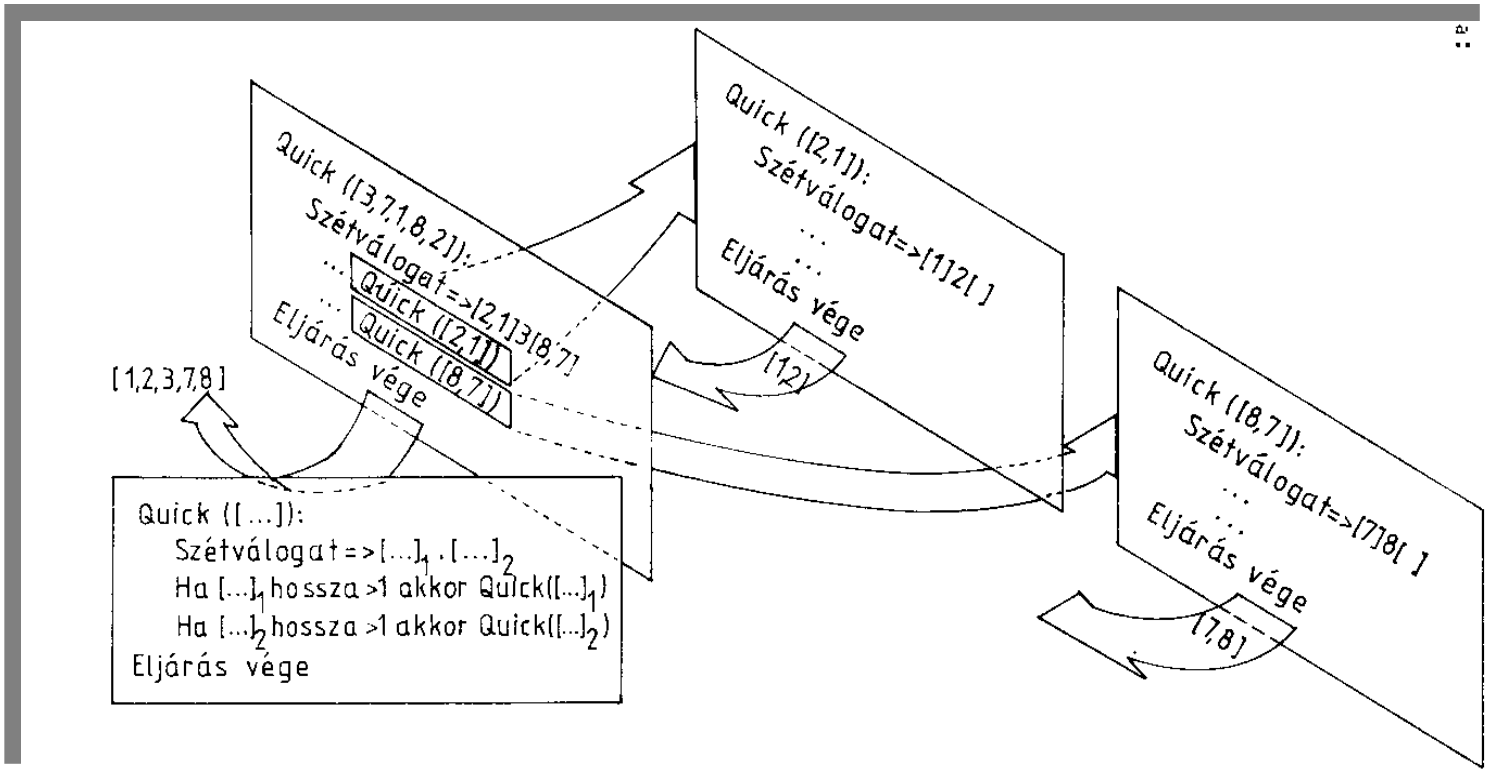
Oszd meg és uralkodj!

Gyorsrendezés (quicksort):

```
Quick(E,U):
   Szétválogatás(E,U,K)
   Ha E<K-1 akkor Quick(E,K-1)
   Ha k+1<U akkor Quick(K+1,U)
Eljárás vége.
```

Oszd meg és uralkodj!

Gyorsrendezés (quick sort):



Oszd meg és uralkodj!

Gyorsrendezés:

```
Quick(A,e,v):
  Szétválogat(A,e,v,k)
  Rendezés(A,e,k-1)
  Rendezés(A,k+1,v)
Eljárás vége.

Rendezés(A,e,v):
  Ha v-e>0 akkor Quick(A,e,v)
Eljárás vége.
```

Közvetlen rekurzió: Az A eljárás saját magát hívja.

Közvetett rekurzió: A hívja B-t, B hívja A-t.

Oszd meg és uralkodj!

Szétválogatás helyben:

```
Szétválogat(A,e,v,k):
  i:=e; j:=v; y:=A(e)
  Ciklus amíg i<j
    Ciklus amíg i<j és A(j)≥y
      j:=j-1
    Ciklus vége
    Ha i<j akkor A(i):=A(j); j:=j-1
            Ciklus amíg i<j és A(i)≤y
              i:=i+1
            Ciklus vége
            Ha i<j akkor A(j):=A(i); i:=i+1
  Ciklus vége
  A(i):=y; k:=i
Eljárás vége.
```

Oszd meg és uralkodj!

Összefésüléses rendezés (mergesort):

- felbontás: a sorozat két részsorozatra bontása (középen)

$$\boxed{X_1, \dots, X_k}\boxed{X_{k+1}, \dots, X_n}$$

- uralkodás: a két részsorozat rendezése (rekurzívan)
- összevonás: a két rendezett részsorozat összefésülése
- triviális eset: $n \leq 1$

Oszd meg és uralkodj!

Összefésüléses rendezés (mergesort):

```
Rendez(E,U):
   Ha E<U akkor K:=(E+U)/2
                Rendez(E,K); Rendez(K+1,U)
                Összefésül(E,K,U)
Eljárás vége*.
```

Oszd meg és uralkodj!

i-edik legkisebb kiválasztása:

- felbontás: $\boxed{X_1, \dots, X_{k-1}}\ X_k\ \boxed{X_{k+1}, \dots, X_n}$ szétválogatás (ahol $\forall i,j\ (1 \le i \le k;\ k \le j \le n)$: $X_i \le X_j$)
- uralkodás: i<K esetén az első, i>K esetén a második részben keresünk tovább, rekurzívan
- összevonás: automatikusan történik a helyben szétválogatás miatt
- triviális eset: i=k

Oszd meg és uralkodj!

i-edik legkisebb kiválasztása:

```
Kiválasztás(E,U,i,Y):
   Szétválogatás(E,U,K)
   Ha i=K akkor Y:=X(K)
   különben ha i<K akkor Kiválasztás(E,K-1,i,Y)
   különben Kiválasztás(K+1,U,i-K,Y)
Eljárás vége.
```

Oszd meg és uralkodj!

Párhuzamos maximum-minimum kiválasztás

- Egyszerre kell egy sorozat maximumát és minimumát is meghatározni
- A megoldás ötlete: 2 elem közül 1 hasonlítással eldönthetjük, hogy melyik a maximum és melyik a minimum.
- Ha kettőnél több elemünk van, akkor osszuk két részre a sorozatot, mindkét részben határozzuk meg a maximumot és a minimumot, majd ezekből adjuk meg a teljes sorozat maximumát és minimumát!

Oszd meg és uralkodj!

Párhuzamos maximum-minimum kiválasztás

- leállási feltétel: az éppen vizsgált sorozatnak legfeljebb 2 eleme van: a maximum és a minimum 1 hasonlítással meghatározható
- felbontás: a sorozat két részsorozatra bontása (középen)

  $\boxed{X_1, \ldots X_{k-1}, X_k}$ $\boxed{X_{k+1}, \ldots, X_n}$
- uralkodás: mindkét részsorozatra meghatározzuk a maximumot és a minimumot (rekurzívan)
- összevonás: a két maximum közül a nagyobb lesz a sorozat maximuma, a két minimum közül pedig a kisebb lesz a sorozat minimuma.

Oszd meg és uralkodj!

```
Maxmin(X,E,U,Max,Min):
  Ha U-E=0 akkor Max:=E; Min:=E
  különben ha U-E=1 akkor
     Ha X(E)≤X(U) akkor Max:=U; Min:=E
     különben Max:=E; Min:=U
  különben K:=(E+U)/2
           Maxmin(X,E,K,Max1,Min1)
           Maxmin(X,K+1,U,Max2,Min2)
           Ha X(Min1)≤X(Min2) akkor Min:=Min1
                              különben Min:=Min2
           Ha X(Max1)≥X(Max2) akkor Max:=Max1
                              különben Max:=Max2
  Elágazás vége
Eljárás vége.
```

Oszd meg és uralkodj!

N szám legnagyobb közös osztója

- A sorozatot bontsuk két részre; mindkét résznek határozzuk meg a legnagyobb közös osztóját, majd ezek legnagyobb közös osztója lesz a megoldás. Ehhez a legnagyobb közös osztó alábbi tulajdonságát használjuk ki:

$$\text{lnko}(X_1, \ldots X_k, X_{k+1}, \ldots, X_n) = \text{lnko}(\text{lnko}(X_1, \ldots X_k), \text{lnko}(X_{k+1}, \ldots, X_n))$$

Oszd meg és uralkodj!

N szám legnagyobb közös osztója

- leállási feltétel: az éppen vizsgált sorozatnak 1 eleme van: a legnagyobb közös osztó önmaga
- felbontás: a sorozat két részsorozatra bontása (középen)

$$\boxed{X_1, \ldots X_{k-1}, X_k}\boxed{X_{k+1}, \ldots, X_n}$$

- uralkodás: mindkét részsorozatra meghatározzuk a legnagyobb közös osztót (rekurzívan)
- összevonás: a két legnagyobb közös osztónak vesszük a legnagyobb közös osztóját.

Oszd meg és uralkodj!

N szám legnagyobb közös osztója

Gyorsítási lehetőségek:

- ha az első rész legnagyobb közös osztója 1, akkor a második részt már ki sem kell számolni, az eredmény 1;
- ha a második rész legnagyobb közös osztója 1, akkor a két rész legnagyobb közös osztója is biztosan 1.

Oszd meg és uralkodj!

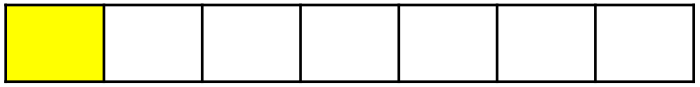
```
Legnagyobb(X,E,U,L):
  Ha U-E=0 akkor L:=X(E)
  különben
     K:=(E+U)/2; L:=1
     Legnagyobb(X,E,K,L1)
     Ha L1>1 akkor Legnagyobb(X,K+1,U,L2)
                   Ha L2>1 akkor L:=lnko(L1,L2)
                        különben L:=1
          különben L:=1
  Elágazások vége
Eljárás vége.
```

Közvetett rekurzió - járdakövezés

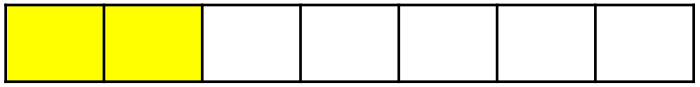
Feladat

Számítsuk ki, hogy hányféleképpen lehet egy n egység méretű járdát kikövezni 1x1, 1x2 és 1x3 méretű lapokkal!

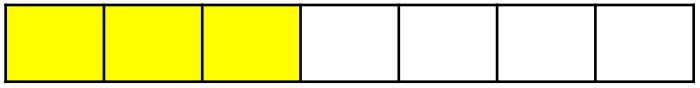
Az első helyre tehetünk 1x1-es lapot:



Az első helyre tehetünk 1x2-es lapot:



Az első helyre tehetünk 1x3-as lapot:



Az első esetben n-1, a másodikban n-2-t, a harmadikban pedig n-3 cellát kell még lefednünk. Azaz az n cella lefedéseinek Lefed(n) száma Lefed(n-1)+Lefed(n-2)+Lefed(n-3).

Közvetett rekurzió - járdakövezés

Megoldás

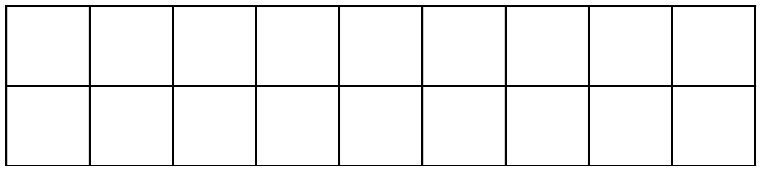
```
Lefed(N):
  Elágazás
    N=1 esetén Lefed:=1
    N=2 esetén Lefed:=2
    N=3 esetén Lefed:=4
    egyéb esetben Lefed:=Lefed(N-1)+
                         Lefed(N-2)+Lefed(N-3)
  Elágazás vége
Függvény vége.
```

Sokszoros hívás esetén vagy memorizálás, vagy ciklusos megoldás kell!

Közvetett rekurzió - járdakövezés

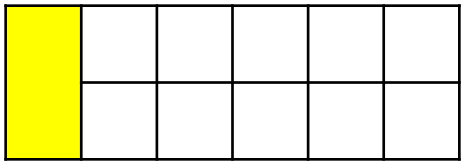
Feladat

Számítsuk ki, hogy hányféleképpen lehet egy 2xn egység méretű járdát kikövezni 1x2 és 1x3 méretű lapokkal!

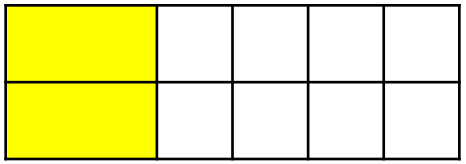


Közvetett rekurzió - járdakövezés

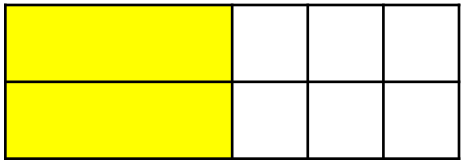
Az első oszlop egyféleképpen fedhető le:



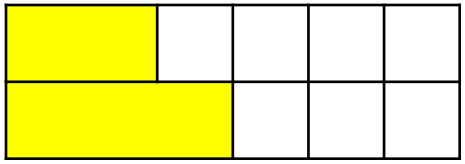
Az első két oszlop további elrendezéssel újra egyféleképpen fedhető le:



Az első három oszlop újra egyféleképpen:



Sajnos ez is előfordulhat:



Közvetett rekurzió - járdakövezés

Jelölje A(n) a megoldás értékét 2xn egység méretű járda esetén!

Jelölje B(n) a megoldás értékét 2xn egység méretű járda esetén, ha az egyik jobboldali sarok nincs befestve!

$$A(n)=\begin{cases}1 & \text{ha } n=1\\ 2 & \text{ha } n=2\\ 4 & \text{ha } n=3\\ A(n-1)+A(n-2)+A(n-3)+2*B(n-2) & \text{ha } n>3\end{cases}$$

$$B(n)=\begin{cases}0 & \text{ha } n=1\\ 0 & \text{ha } n=2\\ 1 & \text{ha } n=3\\ A(n-3)+B(n-1)+B(n-3) & \text{ha } n>3\end{cases}$$

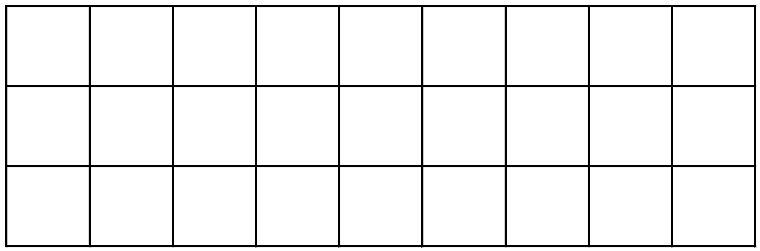
Közvetett rekurzió - járdakövezés

```
A(n):
   Ha n=1 akkor A:=1
   különben ha n=2 akkor A:=2
   különben ha n=3 akkor A:=4
   különben A:=A(n-1)+A(n-2)+A(n-3)+2*B(n-2)
Függvény vége.
B(n):
   Ha n<3 akkor B:=0
   különben ha n=3 akkor B:=1
   különben B:=A(n-3)+B(n-1)+B(n-3)
Függvény vége.
```

Közvetett rekurzió - járdakövezés

Feladat

Számítsuk ki, hogy hányféleképpen lehet egy 3xn egység méretű járdát kikövezni 1x2 méretű lapokkal!

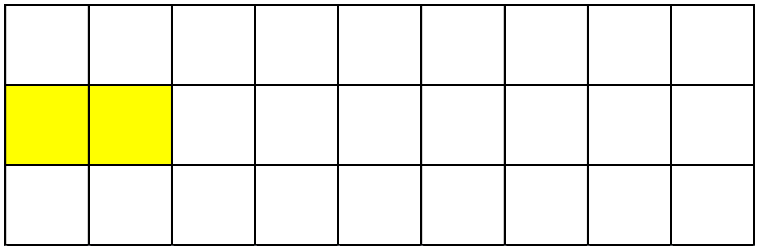


Megoldás

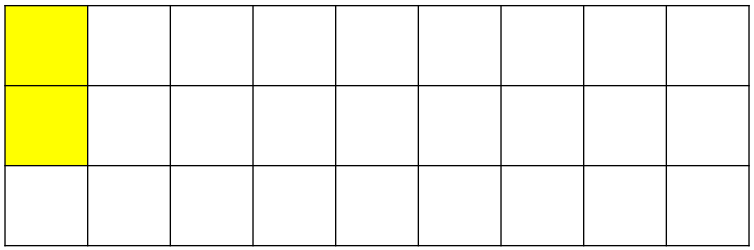
Biztos nincs megoldás, ha n páratlan szám!

Közvetett rekurzió - járdakövezés

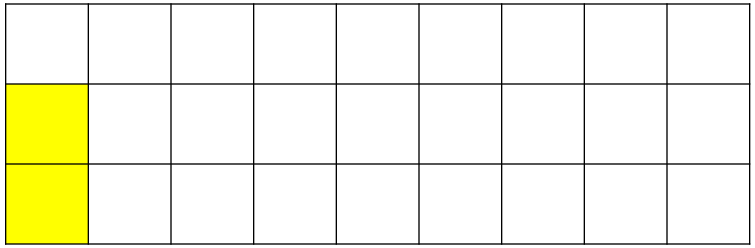
Az első oszlop középső négyzete háromféleképpen fedhető le.



1. eset



2. eset

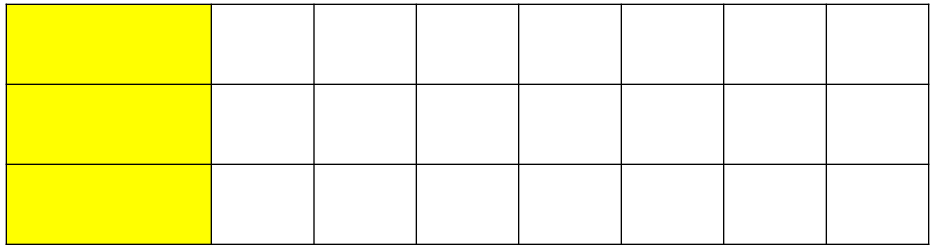


3. eset

Közvetett rekurzió - járdakövezés

Az egyes esetek csak az alábbi módon folytathatók:

Jelölje A(n) a megoldás értékét 3xn egység méretű járda esetén!

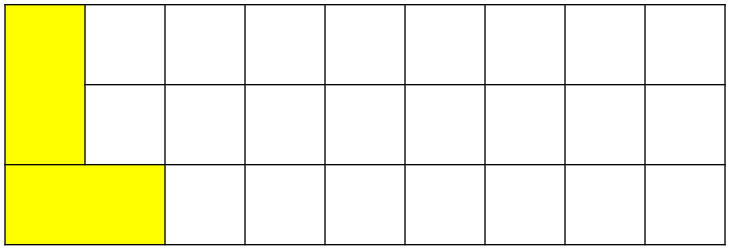


Az 1. eset csak így folytatható

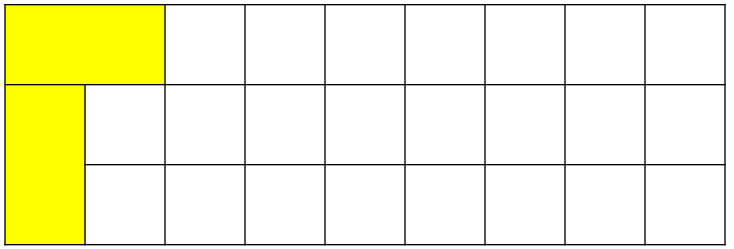
Közvetett rekurzió - járdakövezés

Jelölje B(n) azt, hogy hányféleképpen fedhető le egy 3xn egység méretű járda, amelynek a bal alsó sarka már le van fedve!

Szimmetria miatt a jobb felső sarok lefedettsége esetén is B(n)-féle lefedés van.



A 2. eset csak így folytatható



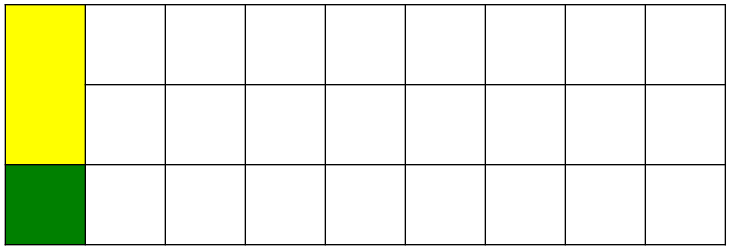
A 3. eset csak így folytatható

$$A(n)=\begin{cases}0 & ha \quad n=1\\ 3 & ha \quad n=2\\ A(n-2)+2*B(n-1) & ha \quad n>2\end{cases}$$

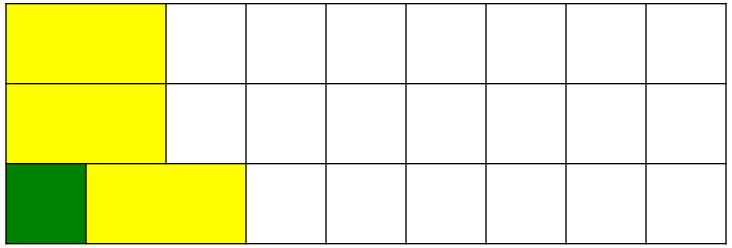
Közvetett rekurzió - járdakövezés

Az egyes esetek csak az alábbi módon folytathatók:

Jelölje B(n) azt, hogy hányféleképpen fedhető le egy 3xn egység méretű járda, amelynek a bal alsó sarka már le van fedve! B(n) páros n-re mindig 0 értékű!



A 2. eset csak így folytatható



Az 1. eset csak így folytatható

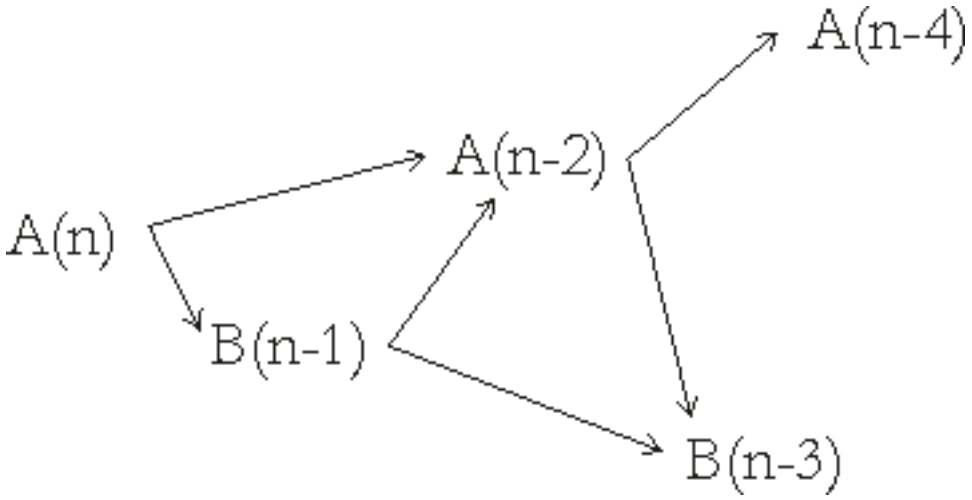
$$B(n)=\begin{cases} 1 & ha \quad n=1 \\ 0 & ha \quad n=2 \\ A(n-1)+B(n-2) & ha \quad n>2 \end{cases}$$

Közvetett rekurzió - járdakövezés

```
A(n):
   Ha n=1 akkor A:=0
   különben ha n=2 akkor A:=3
   különben A:=A(n-2)+2*B(n-1)
Függvény vége.
B(n):
   Ha n=1 akkor B:=1
   különben ha n=2 akkor B:=0
   különben B:=A(n-1)+B(n-2)
Függvény vége.
Kövezés(n):
   Ha páros(n) akkor Kövezés:=A(n)
   különben Kövezés:=0
Függvény vége.
```

Közvetett rekurzió - járdakövezés

Szükség van itt memorizálásra?



Igen, B(n-3)-hoz háromféle, A(n-4)-hez ötféle úton juthatunk el (B(n-3)-ból is számoljuk) – Fibonacci számszor!

Megjegyzés: Figyeljük meg, hogy csak minden második A(i) és B(i) értéket számoljuk ki!

Közvetett rekurzió - járdakövezés

```
A(n):
   Ha TA(n)<0 akkor
      Ha n=1 akkor TA(n):=0
      különben ha n=2 akkor TA(n):=3
      különben TA(n):=A(n-2)+2*B(n-1)
   Elágazás vége
   A:=TA(n)
Függvény vége.
B(n):
   Ha TB(n)<0
      Ha n=1 akkor TB(n):=1
      különben ha n=2 akkor TB(n):=0
      különben TB(n):=A(n-1)+B(n-2)
   Elágazás vége
   A:=TB(n)
Függvény vége.
```

Rekurzió és iteráció

Jobbrekurzió

A rekurzió problematikáját – mint láttuk – a lokális adatok, ill. paraméterek kezelése jelenti. Ha az eljárás ***utolsó*** utasításaként szerepel a rekurzív hívás, akkor ***nincs szükség*** a lokális adatok és paraméterek visszaállítására, azaz ***vermelésére***.

Ezt az esetet nevezzük jobbrekurziónak (vagy hosszabban jobboldali rekurziónak).

Rekurzió és iteráció

Jobbrekurzió alapesetei

```
R(x):
   S(x)
   Ha p(x) akkor R(x)
Eljárás vége.
```

```
R(x):
   Ciklus
      S(x)
   amíg p(x)
   Ciklus vége
Eljárás vége.
```

Azaz az átírás egy egyszerű hátultesztelős ciklus.

Rekurzió és iteráció

Jobbrekurzió alapesetei

```
R(x):
   Ha p(x) akkor T(x); R(x)
Eljárás vége.
```

```
R(x):
   Ciklus amíg p(x)
      T(x)
   Ciklus vége
Eljárás vége.
```

Azaz az átírás egy egyszerű elöltesztelős ciklus.

Rekurzió és iteráció

Jobbrekurzió alapesetei

```
R(x):
   S(x)
   Ha p(x) akkor T(x); R(x)
         különben U(x)
Eljárás vége.
```

```
R(x):
   S(x)
   Ciklus amíg p(x)
      T(x); S(x)
   Ciklus vége
   U(x)
Eljárás vége.
```

Rekurzió és iteráció

Jobbrekurzió példa

Egy szó kiírása:

```
Kiír(szó):
  Ha nem üres?(szó) akkor
    Ki: első(szó)
    Kiír(elsőutániak(szó))
Eljárás vége.
```

```
Kiír(szó):
  Ciklus amíg nem üres?(szó)
    Ki: első(szó)
    szó:=elsőutániak(szó)
  Ciklus vége
Eljárás vége.
```

Rekurzió és iteráció

Jobbrekurzió példa

Logaritmikus kiválasztás (az A vektor E. és V. eleme között az X elem biztosan megtalálható):

```
Kiválasztás(A,X,K,E,V):
  K:=(E+V) div 2
  Elágazás
    A(K)<X esetén Kiválasztás(A,X,K,K+1,V)
    A(K)>X esetén Kiválasztás(A,X,K,E,K-1)
  Elágazás vége
Eljárás vége.
```

Ez nem a formális jobbrekurzió változat, első lépésként tehát formális jobbrekurzióvá kell alakítani!

Rekurzió és iteráció

Jobbrekurzió példa

Logaritmikus kiválasztás (az A vektor E. és V. eleme között az X elem biztosan megtalálható):

```
Kiválasztás(A,X,K,E,V):
  K:=(E+V) div 2
  Elágazás
    A(K)<X esetén E:=K+1
    A(K)>X esetén V:=K-1
  Elágazás vége
  Ha A(K)≠X akkor Kiválasztás(A,X,K,E,V)
Eljárás vége.
```

Rekurzió és iteráció

Jobbrekurzió példa

Logaritmikus kiválasztás (az A vektor E. és V. eleme között az X elem biztosan megtalálható):

```
Kiválasztás(A,X,K,E,V):
  Ciklus
    K:=(E+V) div 2
    Elágazás
      A(K)<X esetén E:=K+1
      A(K)>X esetén V:=K-1
    Elágazás vége
  amíg A(K)≠X
  Ciklus vége
Eljárás vége.
```

Nem tisztán jobbrekurzió példa

Gyorsrendezés:

```
Quick(A,e,v):
  Szétválogat(A,e,v,k)
  Ha k-e>1 akkor Quick(A,e,k-1)
  Ha v-k>1 akkor Quick(A,k+1,v)
Eljárás vége.
```

```
Quick(A,e,v):
  Ciklus
    Szétválogat(A,e,v,k)
    Ha k-e>1 akkor Quick(A,e,k-1)
    e:=k+1
  amíg v-k>1
  Ciklus vége
Eljárás vége.
```

Rekurzió és iteráció

Balrekurzió

Ha az eljárás első utasításaként szerepel a rekurzív hívás, akkor a rekurzió lényegében az első nem rekurzívan számolható érték megkeresését szervezi. Majd a visszatérés(ek) után végzi el a transzformációt. Vagyis fordított sorrendű földolgozást végez.

Általános sémája:

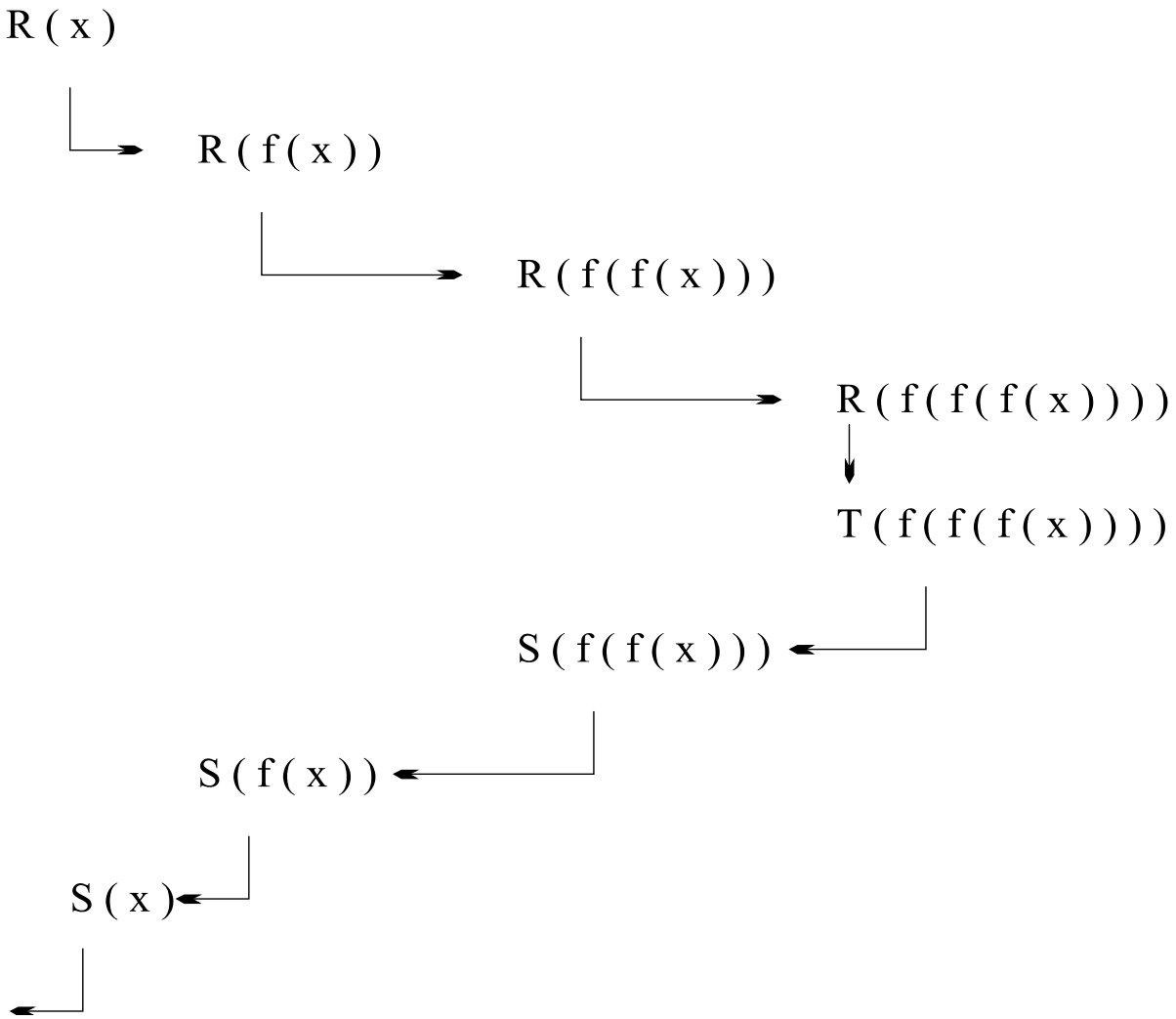
```
R(x,y):
   Ha p(x,y) akkor R(f(x),y); S(x,y)
             különben T(x,y)
Eljárás vége.
```

Rekurzió és iteráció

Balrekurzió – példák a feldolgozandóra

- a sorozat elemei egy soros állomány rekordjai,
- a sorozat elemeit láncolt ábrázolással tároljuk és az aktuális elemtől csak a következőt lehet elérni, az előzőt nem,
- a sorozat elemeit egy sor- vagy veremstruktúrában tároljuk,
- a sorozat elemeit mindig az előző elemből számítjuk, s a sorozat előre meg nem állapítható tagjától visszafelé kell kiírni az elemeket,
- nyelvünk csak olyan szövegkezelő függvényeket ismer, amelyek a szöveg első karakterét tudják megadni, illetve az első elhagyásával keletkezett részt.

Rekurzió és iteráció



Rekurzió és iteráció

Balrekurzió általános átírása

Ha feltehetjük, hogy az f függvénynek létezik inverze:

```
R(x,y):
  N:=0
  Ciklus amíg p(x,y)
    x:=f(x); N:=N+1
  Ciklus vége
  T(x,y)
  Ciklus I=1-től N-ig
    x:=f^-1(x); S(x,y)
  Ciklus vége
Eljárás vége.
```

Rekurzió és iteráció

Balrekurzió általános átírása

Ha az f függvénynek nem létezik inverze:

```
R(x,y):
  N:=0
  Ciklus amíg p(x,y)
    Verembe(x); x:=f(x); N:=N+1
  Ciklus vége
  T(x,y)
  Ciklus i=1-től N-ig
    Veremből(x); S(x,y)
  Ciklus vége
Eljárás vége.
```

Rekurzió és iteráció

Balrekurzió példa

Egy szó „tükrözve” kiírása, azaz betűi sorrendje megfordítása:

```
Tükröz(szó):
  Ha nem üres?(szó) akkor Tükröz(elsőutániak(szó))
                          Ki: első(szó)
Eljárás vége.
```

Rekurzió és iteráció

Balrekurzió példa

Egy szó „tükrözve” kiírása, azaz betűi sorrendje megfordítása:

```
Tükröz(szó):
  N:=0
  Ciklus amíg nem üres?(szó)
    Verembe(első(szó)); N:=N+1
    szó:=elsőutániak(szó)
  Ciklus vége
  Ciklus i=1-től N-ig
    Veremből(B); Ki: B
  Ciklus vége
Eljárás vége.
```

Rekurzió és iteráció

Balrekurzió példa

M-nél kisebb 2-hatványok visszafelé:

```
Hatványok(K,M):
  Ha K≤M akkor Hatványok(2*K,M); Ki: K
Eljárás vége.
```

```
Hatványok(K,M):
  N:=0
  Ciklus amíg K≤M
    K:=2*K; N:=N+1
  Ciklus vége
  Ciklus I=1-től N-ig
    K:=K/2; Ki: K
  Ciklus vége
Eljárás vége.
```

Rekurzió és iteráció

További példák

Legnagyobb közös osztó

Hatványozás

Körmentes labirintus (balra, egyenesen vagy jobbra léphet):

```
function success = find_way_out( maze, room )
  for every door in the room
    new_room = go_through_door( maze, door )
    if  ( find_way_out ( maze, new_room ) ) take that door.
```

Rekurzió előadás vége