Programozás
10. előadás

Tartalom

- Programtranszformációk
- Segédösszegek számítása
- Rekurzió
- Rekurzió és iteráció
- Programozási tételek rekurzívan

Programtranszformációk

Programtranszformáció: Az algoritmus ekvivalens átalakítása, melynek célja

- hatékonyabbra írás
- egyszerűsítés
- megvalósíthatóság

Programtranszformációk

Egyszerűsítés, hatékonyabbra írás:

Az origótól legmesszebb levő pont

Változó
i:**Egész**

<table>
<tr><td colspan="3">Max:=1; MaxÉrt:=gyök(p[1].x²+p[1].y²)</td></tr>
<tr><td colspan="3">i=2..N</td></tr>
<tr><td rowspan="3"></td><td colspan="2">I   gyök(p[i].x²+p[i].y²)>MaxÉrt   N</td></tr>
<tr><td>Max:=i</td><td rowspan="2">—</td></tr>
<tr><td>MaxÉrt:=gyök(p[i].x²+p[i].y²)</td></tr>
</table>

A négyzetgyök monoton függvény, emiatt a maximum meghatározásához nem szükséges.

Programtranszformációk

Egyszerűsítés, hatékonyabbra írás:

Az origótól legmesszebb levő pont

Változó
i:**Egész**

Max:=1; MaxÉrt:=p[1].x^2+p[1].y^2
i=2..N

I	p[i].x^2+p[i].y^2>MaxÉrt	N
Max:=i		—
MaxÉrt:=p[i].x^2+p[i].y^2		

Itt még ugyanazt a képletet többször is kiszámítjuk.

Programtranszformációk

Többszörös kiszámítás elkerülése:

Az origótól legmesszebb levő pont

Változó
i:**Egész**
táv:**Valós**

$Max:=1;\ MaxÉrt:=p[1].x^2+p[1].y^2$

$i=2..N$

- $táv:=p[i].x^2+p[i].y^2$
- $táv>MaxÉrt$
  - I: $Max:=i$; $MaxÉrt:=táv$
  - N: —

Programtranszformációk

Párhuzamos értékadás kifejtése:

a,b,c:=f(x),g(x),h(x)

Egymás utáni kiszámításra bontható, ha az összefüggés körmentes:

a:=f(x); b:=g(x); c:=h(x)

Programtranszformációk

Párhuzamos értékadás kifejtése:

```
a,b,c:=b,c,a
```

segédváltozóval egymás utáni kiszámításra bontható, ha az összefüggés kört tartalmaz:

```
segéd:=a; a:=b; b:=c; c:=segéd
```

Programtranszformációk

Ciklusok összevonása:

Azonos lépésszámú ciklusok összevonhatóak, ha függetlenek egymástól.

Változó
i:**Egész**

S:=0
i=1..N
S:=S+X[i]
P:=1
i=1..N
P:=P*X[i]

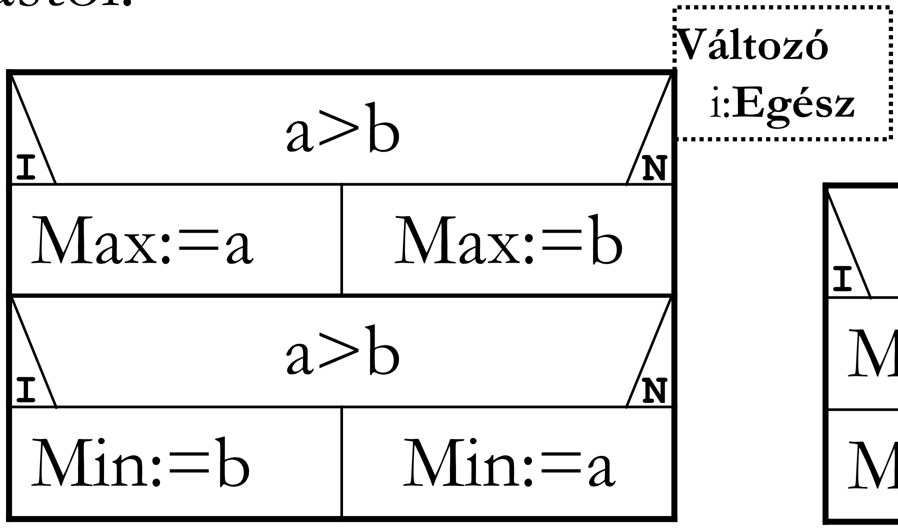
Változó
i:**Egész**

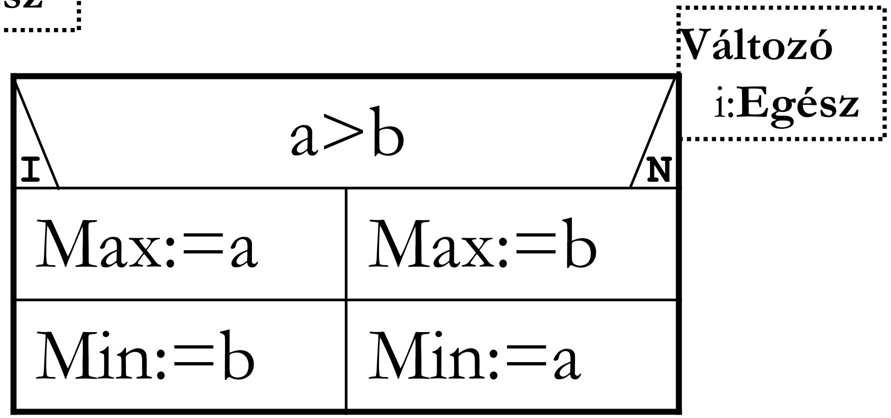
S:=0; P:=1
i=1..N
S:=S+X[i]; P:=P*X[i]

Programtranszformációk

Elágazások összevonása:

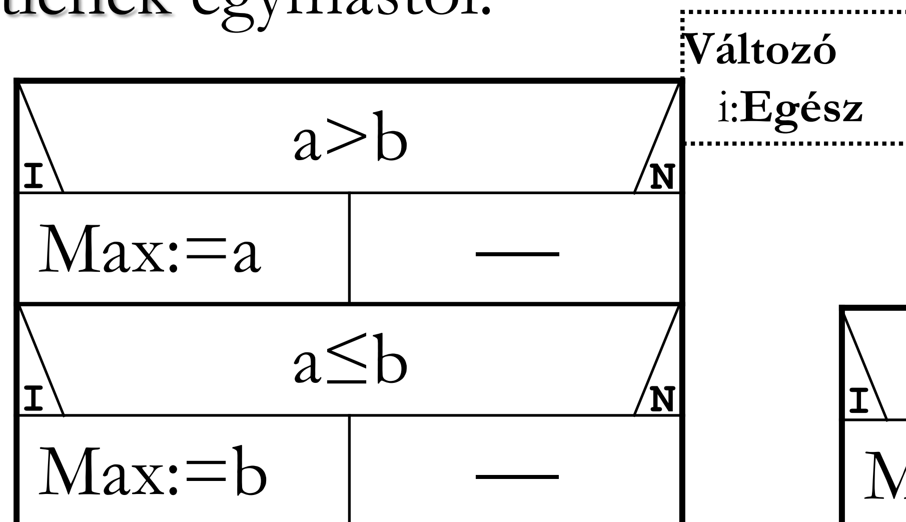
Azonos feltételű elágazások összevonhatóak, ha függetlenek egymástól.

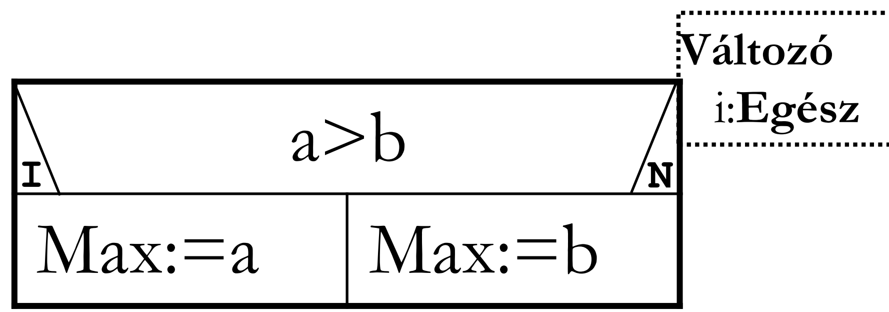




Elágazások összevonása:

Kizáró feltételű, teljes, egyágú elágazások is összevonhatók, ha függetlenek egymástól.

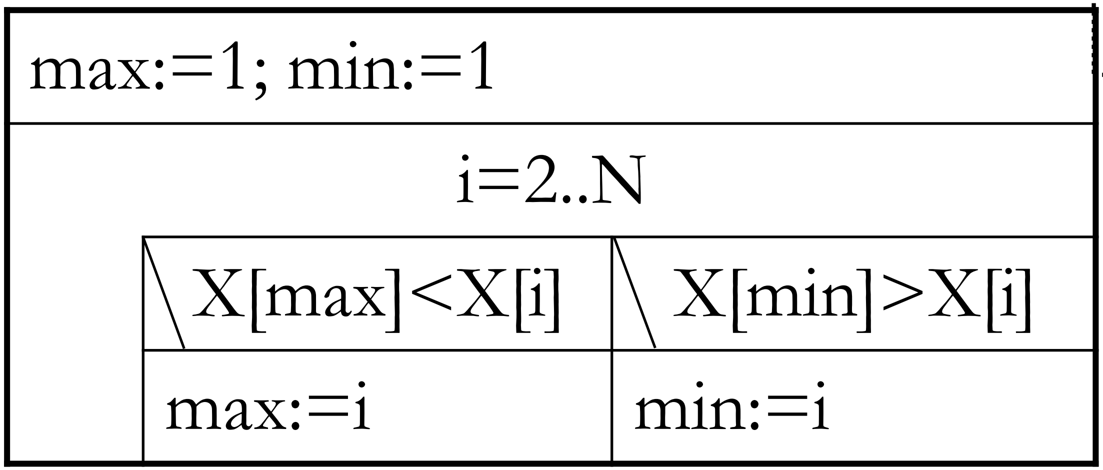


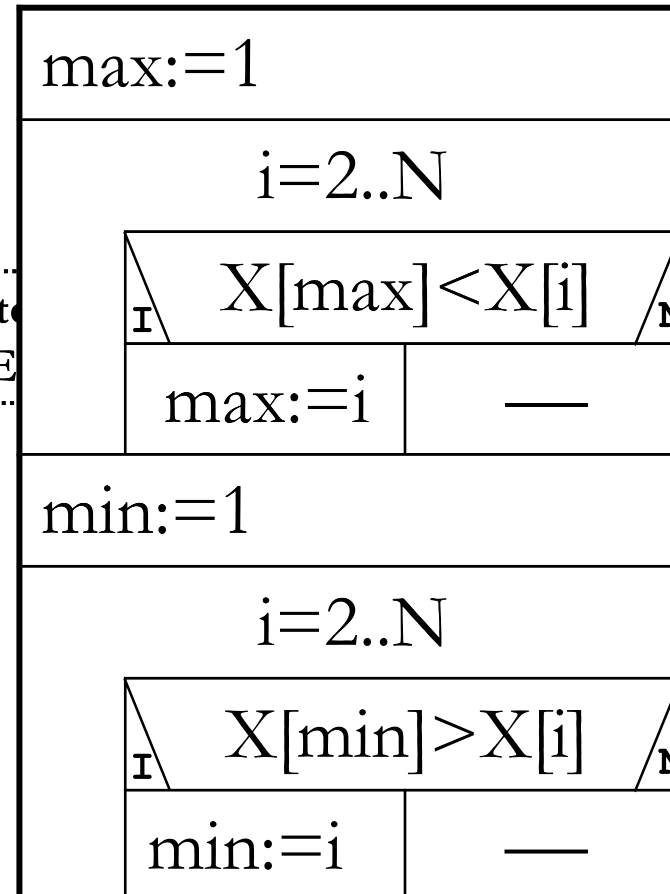


Programtranszformációk

Ciklusok és elágazások összevonása:

Azonos lépésszámú ciklusok, kizáró feltételű elágazások is összevonhatók, ha függetlenek egymástól.

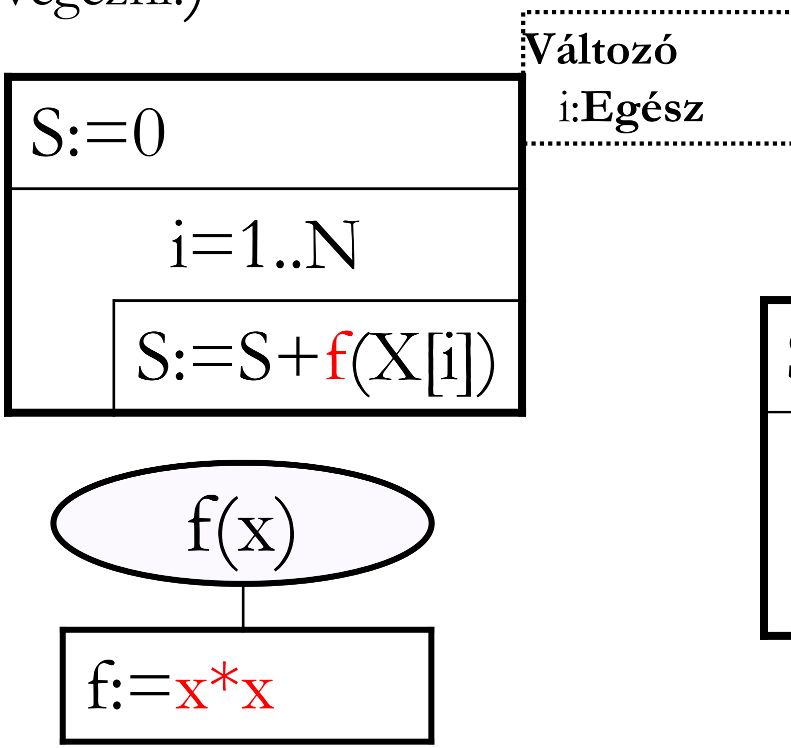


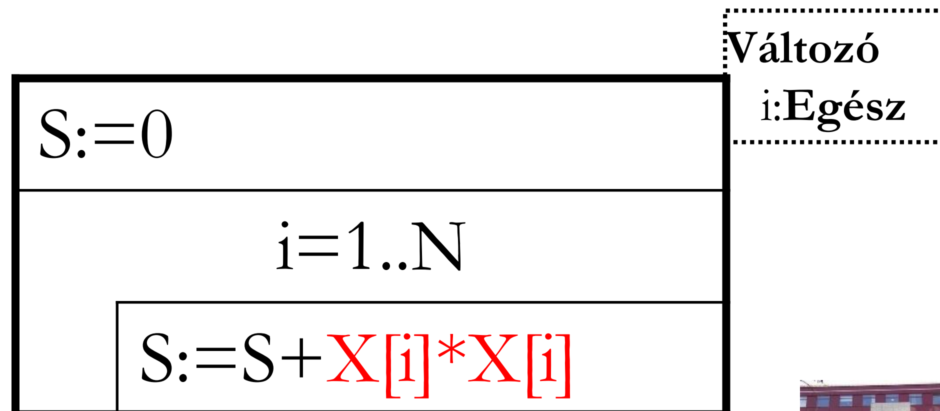


Programtranszformációk

Függvény behelyettesítése:

Függvényhívás helyére egy egyszerű függvény képlete (a függvény törzse) behelyettesíthető. (C++ fordítók ilyen optimalizálást el tudnak végezni.)





Programtranszformációk

Utasítás kiemelése ciklusból:

A ciklus magjából a ciklustól független utasítások kiemelhetők.

(C++ fordítók ilyen optimalizálást el tudnak végezni.)

Változó
i:**Egész**

S:=0

i=1..N

S:=S+h[i]*r^2*pi

Változó
i:**Egész**
rpi:**Valós**

S:=0; rpi:=r^2*pi

i=1..N

S:=S+h[i]*rpi

Programtranszformációk

Keresés, eldöntés → kiválasztás transzformáció:

A vizsgálandó sorozat végére helyezzünk egy T tulajdonságú elemet – biztosan találunk ilyet!

Változó
i:**Egész**

i:=1
i≤N és T(X[i])
i:=i+1
Van:=i≤N

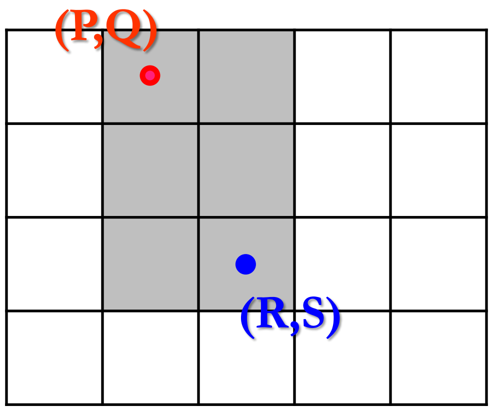
Változó
i:**Egész**

i:=1; X[N+1]:=Telem
T(X[i])
i:=i+1
Van:=i≤N

Segédösszegek

Egy földműves egy téglalap alakú területet szeretne vásárolni egy **N×M**-es téglalap alakú földterületen. Tudja minden megvásárolható földdarabról, hogy azt megművelve mennyi lenne a haszna vagy vesztesége.

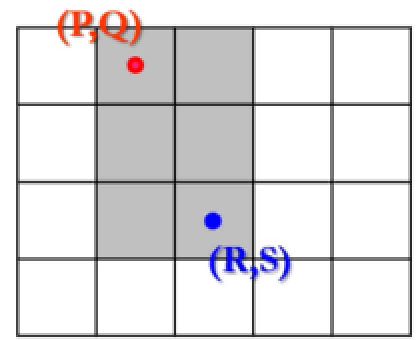
Add meg azt a téglalapot, amelyen a legnagyobb haszon érhető el!



Segédösszegek

- Bemenet: $N,M \in \mathbb{N}$, $T_{1..N,1..M} \in \mathbb{Z}^{N \times M}$
- Kimenet: $P,Q,R,S \in \mathbb{N}$
- Előfeltétel: –
- Utófeltétel: $1 \leq P \leq R \leq N$ és $1 \leq Q \leq S \leq M$ és

  $\forall i,j,k,l\ (1 \leq i \leq k \leq N, 1 \leq j \leq l \leq M)$: $érték(P,Q,R,S) \geq érték(i,j,k,l)$
- Definíció: $érték: \mathbb{N}^4 \rightarrow \mathbb{Z}$



$$érték(a,b,c,d) = \sum_{x=a}^{c} \sum_{y=b}^{d} T_{x,y}$$

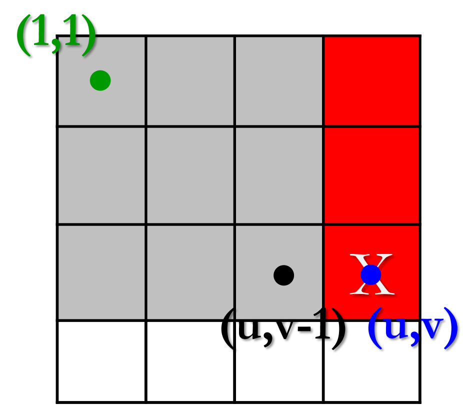
Most ciklust kellene írni i-re, j-re, k-ra, l-re, x-re és y-ra, azaz 6 ciklus lenne egymás belsejében. **Ez sok!**

Segédösszegek

- Az érték függvény definiálása:

Próbáljunk valami részcélt kitűzni:

számoljuk ki az (1,1) bal felső, (u,v) jobb alsó sarkú téglalapok értékét!



X= szürke téglalap értéke +
piros téglalap összege

X → E[u,v]

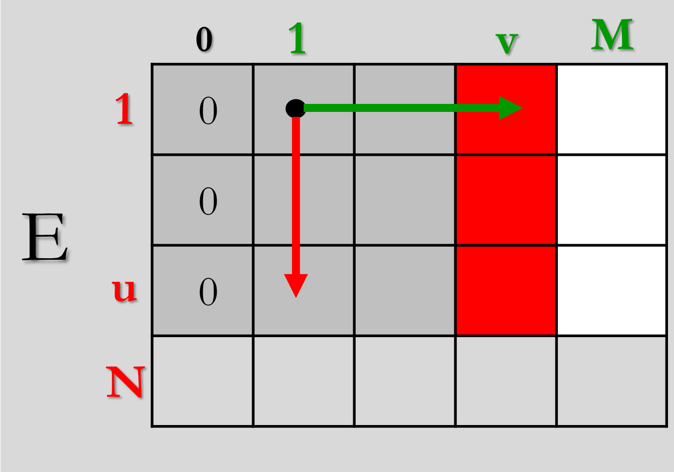
Segédösszegek

- Az érték függvény definiálása (folytatás):

  Az **E mátrix** kiszámítása (E[1..N,0]:=0):

```
v=1..M
    x:=0
    u=1..N
        x:=x+T[u,v]
        E[u,v]:=E[u,v-1]+x
...
```

Változó
u,v:**Egész**
x:**Egész**



Segédösszegek

- Az érték függvény definiálása (folytatás):

Definiáljuk E[u,v] segítségével az érték(i,j,u,v)-t!

érték(i,j,u,v)

érték:=E[u,v]-E[u,j-1]-E[i-1,v]+E[i-1,j-1]

	…	j-1	j	…	v	…
⋮						
i-1	…	$E_{i-1,j-1}$	$E_{i-1,j}$	…	$E_{i-1,v}$	…
i	…	$E_{i,j-1}$	$E_{i,j}$	…	$E_{i,v}$	
⋮				???		
u	…	$E_{u,j-1}$	$E_{u,j}$		$E_{u,v}$	
⋮						

A módszer neve: kumulatív összegzés.

Bemenet: $N,M \in \mathbb{N}$, $T_{1..N,1..M} \in \mathbb{Z}^{N \times M}$
Kimenet: $P,Q,R,S \in \mathbb{N}$
Előfeltétel: –
Utófeltétel: $1 \le P \le R \le N$ és $1 \le Q \le S \le M$ és
$\forall i,j,k,l\ (1 \le i \le k \le N, 1 \le j \le l \le M)$: érték(P,Q,R,S) $\ge$ érték(i,j,k,l)

Segédösszegek

- A maximális összegű téglalap kiválasztása:

Változó
i,j,k,l:**Egész**

```
P,Q,R,S:=1,1,1,1
Maxért:=T[1,1]
i=1..N
  j=1..M
    k=i..N
      l=j..M
        érték(i,j,k,l)>Maxért
        I: P:=i; Q:=j; R:=k; S:=l
           Maxért:=érték(i,j,k,l)
        N: —
```

A ciklusban számított érték konstans idővel határozható meg!

Rekurzió

Klasszikus példák:

- Faktoriális

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

- Fibonacci-számok

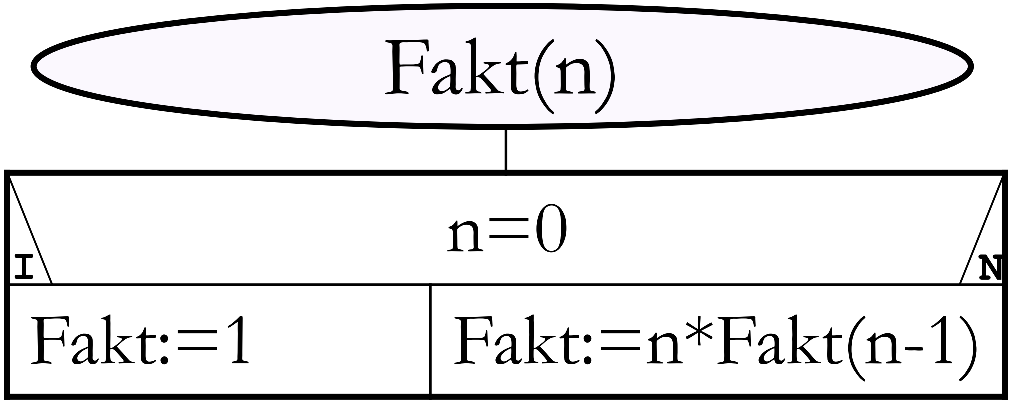
$$Fib(n) = \begin{cases} 0 & ha \;\; n = 0 \\ 1 & ha \;\; n = 1 \\ Fib(n-1) + Fib(n-2) & ha \;\; n > 1 \end{cases}$$

A rekurzió lényege: önhivatkozás

Rekurzív specifikáció és algoritmus

Faktoriális:

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



Itt egy 2-alternatívájú függvényt kell algoritmizálni, ami egy elágazással történik.

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)

n=0	n=1	n>1
Fib:=0	Fib:=1	Fib:=Fib(n-1)+Fib(n-2)

Háromirányú elágazás a megoldás.

Problémák a rekurzióval

Hely: nagyra dagadt veremméret.

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

Példa: **Fibonacci-számok esetén**

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

r(0):=1, r(1):=1, r(i):=r(i–1)+r(i–2)+1 (i>1)

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

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.

c) r(i)=$\Theta(c^i)$, azaz exponenciális műveletigényű.

Rekurzió és iteráció

Korlátos memóriájú függvények:

Ha egy rekurzív függvény minden értéke valamely korábban kiszámolható értékből számolható, akkor némi memória felhasználással elkészíthető a rekurziómentes változat, amelyben az egyes függvényértékeknek megfeleltetünk egy F(0..N) vektort.

A függvény általános formája:

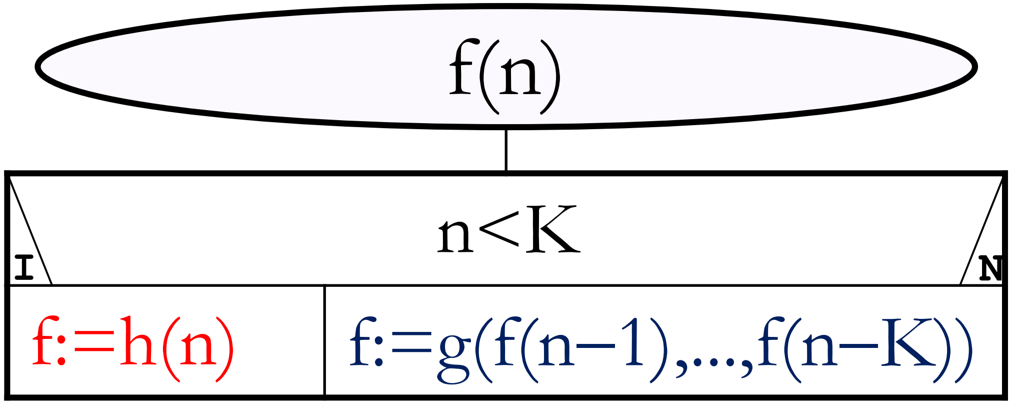
$$f(n) = \begin{cases} g\big(f(n-1), f(n-2), \ldots, f(n-K)\big) & \text{ha} \quad n \geq K \\ h(n) & \text{ha} \quad 0 \leq n < K \end{cases}$$

Rekurzió és iteráció

Korlátos memóriájú függvények:

Rekurzív változat:

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



Rekurzió és iteráció

Korlátos memóriájú függvények:

Iteratív (ciklusos) változat:

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

f(n)

i=0..K−1

F[i]:=h(i)

i=K..n

F[i]:=g(F[i−1],...,F[i−K])

F:=F[n]

Változó

F:**Tömb**[0..n:TH]

Rekurzió és iteráció

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 értékadást kell átalakítani:

```
F[i]:=g(F[i-1],...,F[i-K])
```

Lehet pl. így, ha a `g()` függvény kiszámítása nem függ a paraméterek sorrendjétől :

```
F[i mod K]:=g(F[0],...,F[K-1]).
```

Ekkor elegendő: F[0..K]

Rekurzió és iteráció

Példa: Fibonacci-számok$_{iteratív}$

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

Fib(n)

F[0]:=0; F[1]:=1

i=2..n

F[i]:=F[i–1]+F[i–2]

Fib:=F[n]

Változó
i:**Egész**
F:Tömb[0..n:Egész]

Rekurzió és iteráció

Példa: Fibonacci-számok$_{iteratív}$

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

Helytakarékos megoldás (K=2):

Fib(n)

F[0]:=0; F[1]:=1

i=2..n

F[i mod 2]:=F[0]+F[1]

Fib:=F[n mod 2]

Változó
i:**Egész**
F:Tömb[0..1:Egész]

Rekurzió memorizálással

Többszörös hívás elkerülése:

Amit már kiszámoltunk egyszer, azt ne számoljuk újra! Tároljuk a már kiszámolt értékeket, és ha újra szükségünk van rájuk, használjuk fel őket!

Példa: **Fibonacci-számok esetén**

A megoldásban F[i]≥0 jelentse, ha már kiszámoltuk az i-edik Fibonacci-számot.

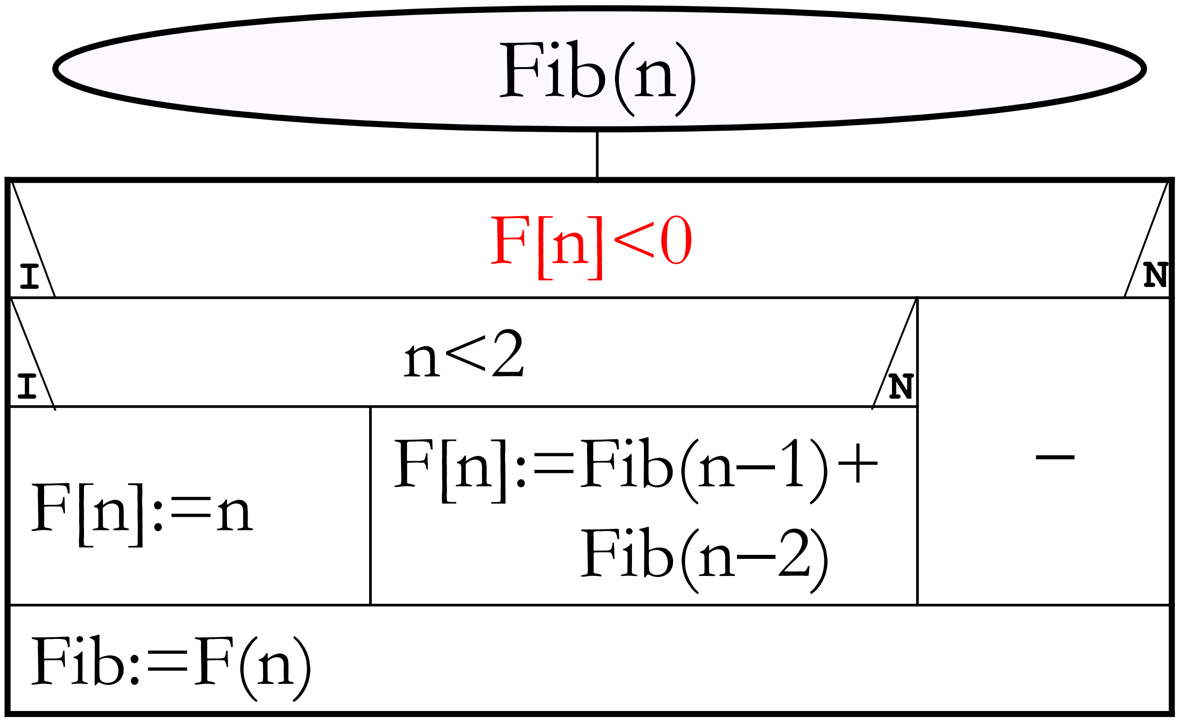
Változó
i:**Egész**
F:**Tömb**[…]

i=0..N
F[i]:=-1
F[N]:=Fib(N)

2018. 11. 21.

Rekurzió memorizálással

Algoritmus (folytatás):



Közvetett rekurzió

Feladat:

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

Megoldás:

Páros(n)

n=0	n=1	
Páros:=Igaz	Páros:=Hamis	Páros:=Páratlan(n−1)

Páratlan(n)

n=0	n=1	
Páratlan:= Hamis	Páratlan:= Igaz	Páratlan:=Páros(n−1)

Közvetlen rekurzió

Feladat:

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

A két – közvetetten – rekurzív eljárás most összevonható:

Megoldás:

Páros(n)

n=0	n=1	
Páros:=Igaz	Páros:=Hamis	Páros:=Páros(n−2)

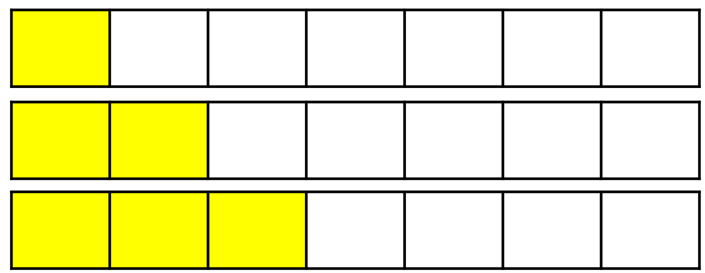
Közvetlen rekurzió – járdakövezés

Feladat:

Számítsuk ki, hogy hányféleképpen lehet egy n egység méretű járdát kikövezni 1×1, 1×2 és 1×3 méretű lapokkal!

Az első helyre tehetünk:

- 1×1-es lapot:
- 1×2-es lapot:
- 1×3-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 száma:

Lefed(n)=Lefed(n–1)+Lefed(n–2)+Lefed(n–3).

Közvetlen rekurzió – járdakövezés

Megoldás:

Lefed(n)

n=0	n=1	n=2	
Lefed:=1	Lefed:=1	Lefed:=2	Lefed:=Lefed(n–1) +Lefed(n–2) +Lefed(n–3)

Sokszoros hívás kiküszöbölése:

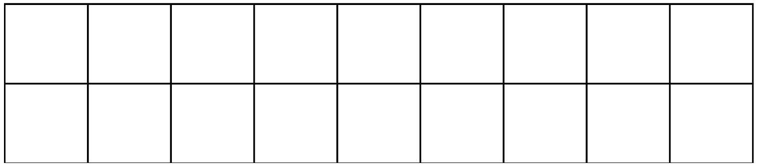
- vagy **memorizálás**sal,
- vagy **iteratív** (ciklusos) implementálással!

2018. 11. 21.

Közvetett rekurzió – járdakövezés

Feladat:

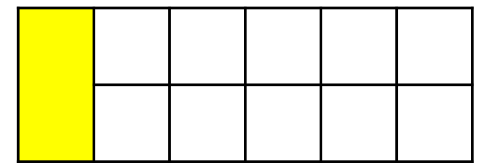
Számítsuk ki, hogy hányféleképpen lehet egy 2×n egység méretű járdát kikövezni 1×2 és 1×3 méretű lapokkal!



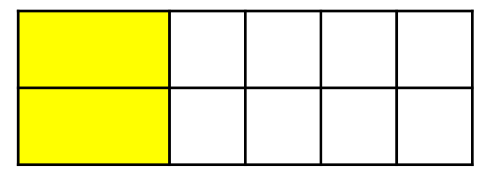
Közvetett rekurzió – járdakövezés

Megoldás:

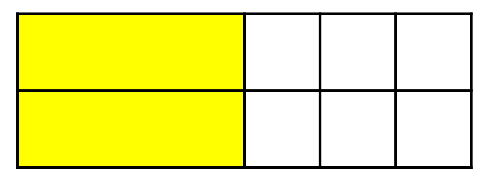
Az első oszlop egyféleképpen fedhető le:



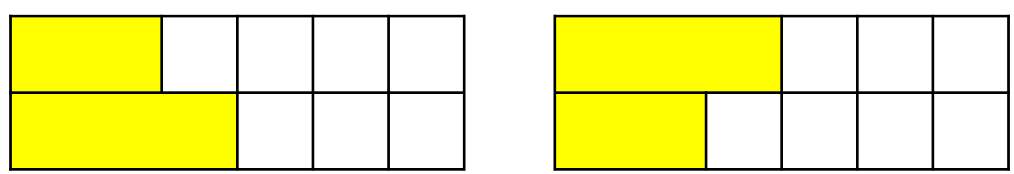
Az első két oszlop további elrendezéssel újra egyféleképpen fedhető le:



Az első három oszlop … újra egyféleképpen:



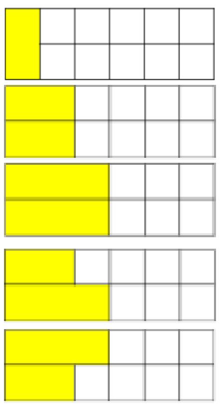
Sajnos ez is előfordulhat:



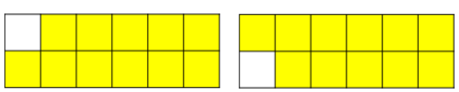
2018. 11. 21.

Közvetett rekurzió – járdakövezés

Jelölje A(n) a megoldás értékét 2×n egység méretű járda esetén! Jelölje B(n) a megoldás értékét 2×n egység méretű járda esetén, ha az egyik baloldali sarok nincs lefedve!



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



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

Közvetett rekurzió – járdakövezés

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

A(n)

n=1	n=2	n=3	
A:=1	A:=2	A:=4	A:=A(n–1)+A(n–2)+A(n–3)+ 2*B(n-2)

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

B(n)

n<3	n=3	
B:=0	B:=1	B:=A(n–3)+B(n–2)+B(n–3)

Programozási tételek rekurzívan

Sorozatszámítás (összegzés):

A sorozatszámítás tétel egy egyszerű rekurziót tartalmazott, ahol minden kiszámolt érték az előző egyetlen értéktől függött:

$$F(X_{1..n}) := \begin{cases} F_0 \,, & , n = 0 \\ f(F(X_{1..n-1}), X_n) & , n > 0 \end{cases}$$

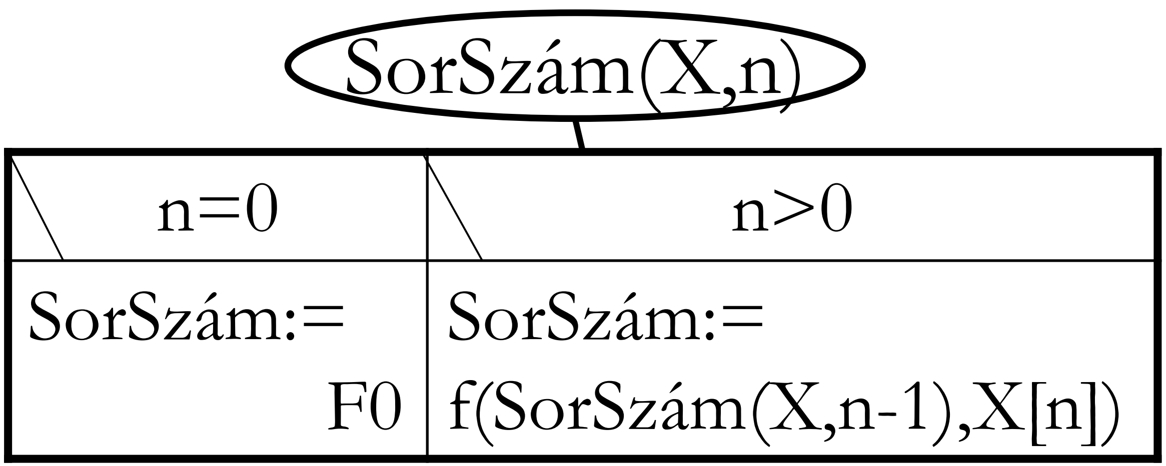
Programozási tételek rekurzívan

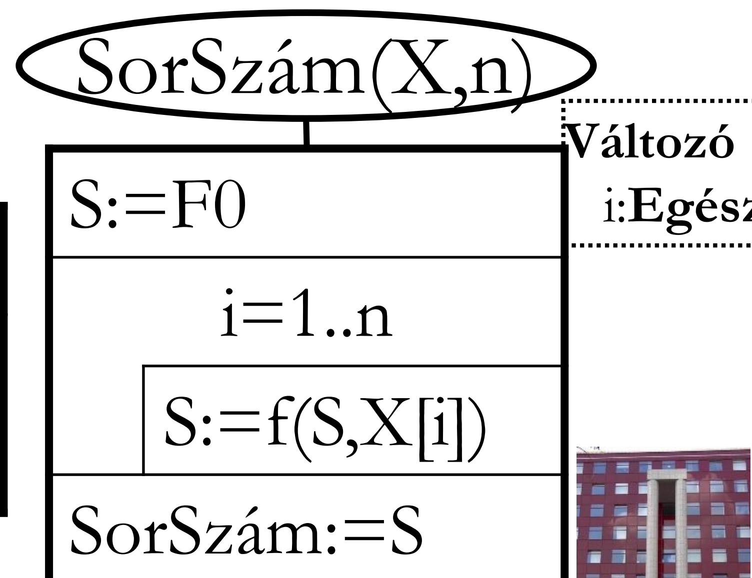
$$F(X_{1..n}) := \begin{cases} F_0\,, & n = 0 \\ f(F(X_{n-1}), X_n)\,, & n > 0 \end{cases}$$

Sorozatszámítás (összegzés):

A sorozatszámítás tétel egy egyszerű rekurziót tartalmazott, ahol minden kiszámolt érték az előző egyetlen értéktől függött:

$$F(X, n) := \begin{cases} F_0\,, & n = 0 \\ f(F(X, n-1), X_n)\,, & n > 0 \end{cases}$$





Programozási tételek rekurzívan

Maximum-kiválasztás:

A maximum-kiválasztás tétel rekurzívan ugyanezen az elven fogalmazható meg:

$$\text{Maximum}(X, n) := \begin{cases} X_1, & n = 1 \\ \max(\text{Maximum}(X, n-1), X_n), & n > 1 \end{cases}$$

Maximum(X,n)

n=1	n>1
Maximum:=x[1]	Maximum:= max(Maximum(X,n-1),X[n])

Programozási tételek rekurzívan

Keresés:

A keresés tétel is ugyanezen az elven fogalmazható meg rekurzívan, de már háromirányú elágazással:

$$\text{Keresés}(X,n) := \begin{cases} (\text{hamis}, -), & n = 0 \\ (\text{igaz}, n), & T(X_n) \\ \text{Keresés}(X, n-1) & \text{egyébként} \end{cases}$$

Keresés(X,n)

n=0	T(X[n])	
Keresés:= (hamis,-)	Keresés:= (igaz,n)	Keresés:= Keresés(X,n-1)

Tartalom