ELTE | IK

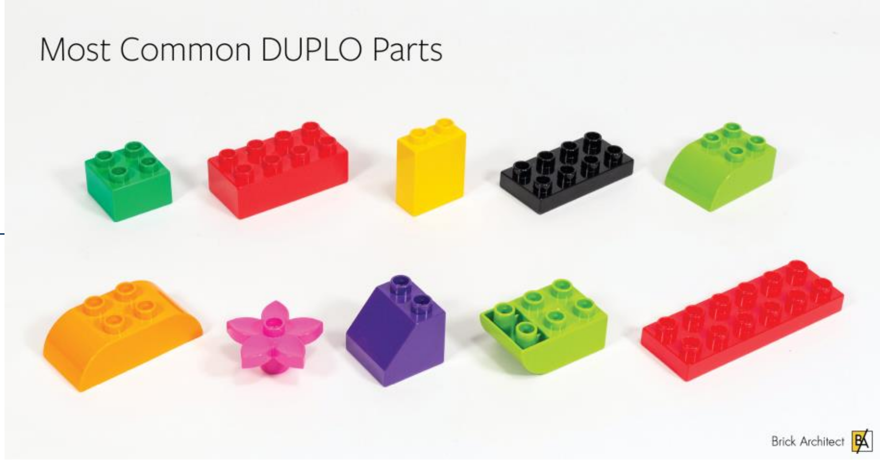
PROGRAMOZÁS

12. előadás

Horváth Győző, Horváth Gyula, Szlávi Péter

Programozási minták

1. Összegzés
2. Megszámolás
3. Maximumkiválasztás
   a. Minimumkiválasztás
4. Feltételes maximumkeresés
5. Keresés
6. Eldöntés
   a. Mind eldöntés
7. Kiválasztás
8. Másolás
9. Kiválogatás



Rekurzió

Klasszikus példák rekurzióra

- 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

Faktoriális:

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

```
Be: n∈N
Ki: f∈N
Ef: -
Uf: f=n!
```

Rekurzív specifikáció

Faktoriális:

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

```
Be: n∈N
Ki: f∈N
Fv: fakt:N->N, fakt(n)={n*fakt(n-1), ha n>0;
                        1          , ha n=0}
Ef: -
Uf: f=fakt(n)
```

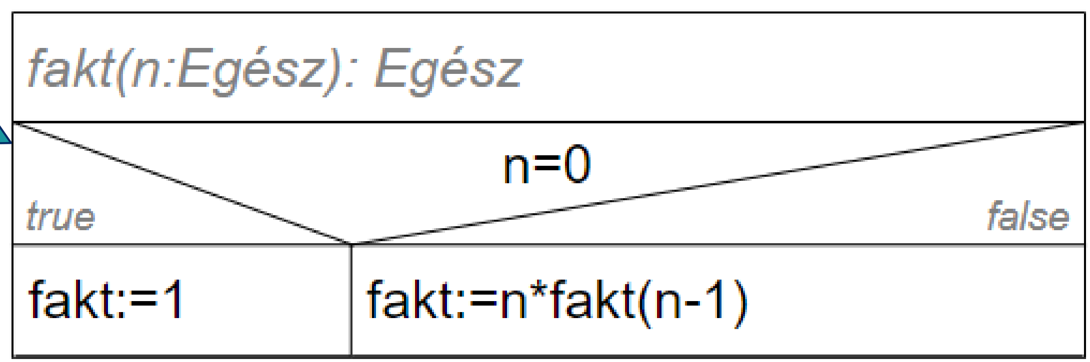
Rekurzív specifikáció és algoritmus

Faktoriális:

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

```
Be: n∈N
Ki: f∈N
Fv: fakt:N->N, fakt(n)={n*fakt(n-1), ha n>0;
                        1          , ha n=0}
Ef: -
Uf: f=fakt(n)
```

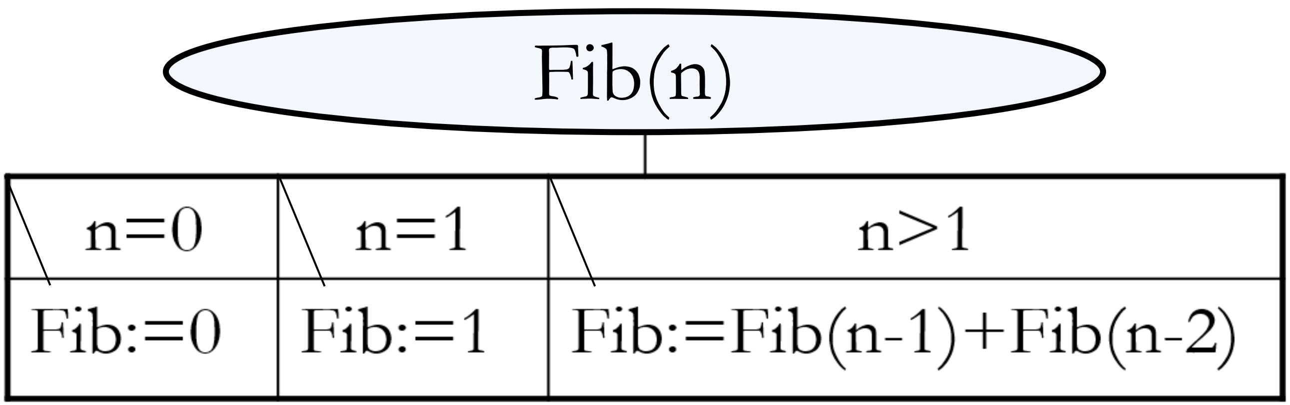
Itt egy 2-alternatívájú függvényt kell algoritmizálni, ami egy „2-irányú” elágazással történik.



Rekurzív specifikáció és algoritmus

Fibonacci-számok:

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

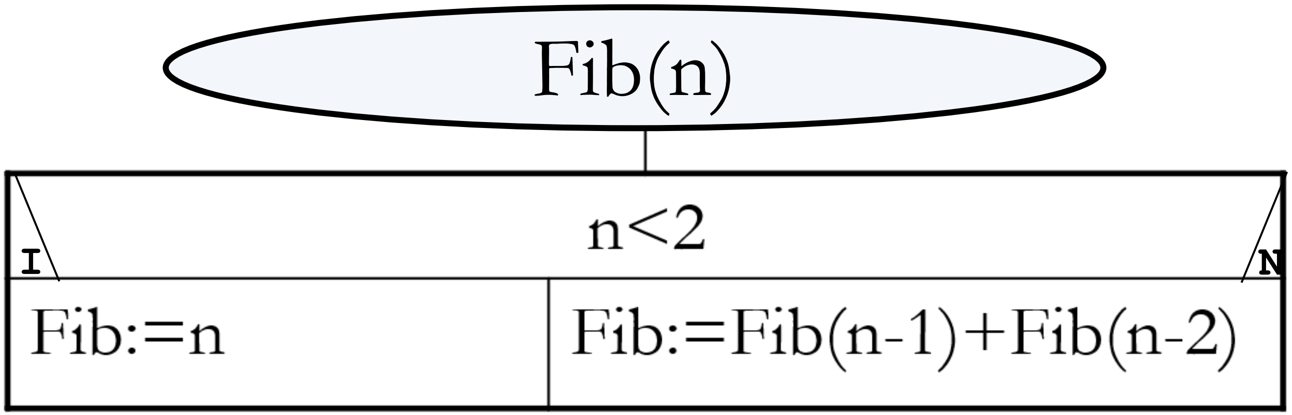


Háromirányú elágazás a megoldás

Rekurzív specifikáció és algoritmus

Fibonacci-számok:

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

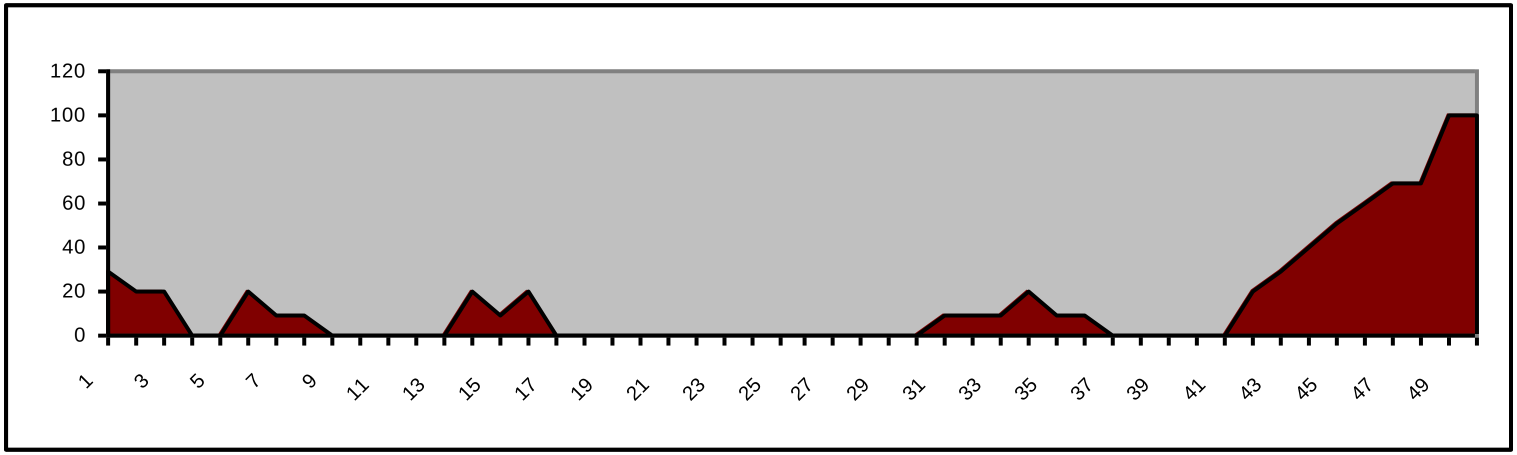


Kétirányú elágazássá alakított megoldás

Intervallumos példák

Feladat:

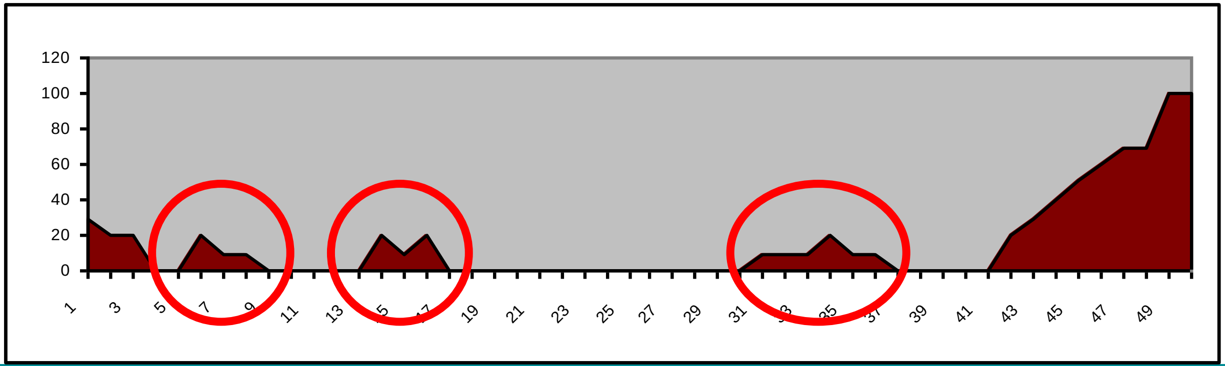
Egy repülőgéppel Európából Amerikába repültünk. Az út során bizonyos kilométerenként mértük a felszín tengerszint feletti magasságát (≥0). 0 magasságot ott mértünk, ahol tenger van, >0-t pedig ott, ahol szárazföld. Adjuk meg a legszélesebb szigetet!



Mi számít szigetnek?

1. Szélén kontinens

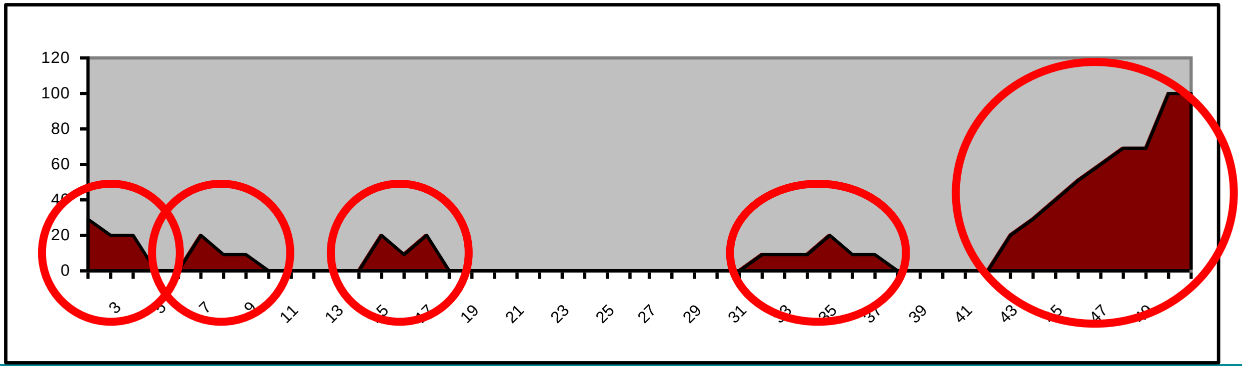
```
Fv: szigetkezdet:N->L,
    szigetkezdet(i)={hamis, ha i=1;
                     mag[i-1]=0 és mag[i]>0 egyébként}
Fv: szigetvég:N->L,
    szigetvég(i)=   {hamis, ha i=n;
                     mag[i+1]=0 és mag[i]>0 egyébként}
```



Mi számít szigetnek?

2. Szélén sziget

```
Fv: szigetkezdet:N->L,
    szigetkezdet(i)={mag[i]>0, ha i=1;
                     mag[i-1]=0 és mag[i]>0 egyébként}
Fv: szigetvég:N->L,
    szigetvég(i)=   {mag[i]>0, ha i=n;
                     mag[i+1]=0 és mag[i]>0 egyébként}
```



Programozási mintákkal

1. Válogassuk ki azokat a szigetkezdeteteket, amelyeknek van vége is! (összevonható)
2. Másolással határozzuk meg hozzájuk a szigetvégeket! (összevonható)
3. Másolással határozzuk meg a távolságokat! (összevonható)
4. Határozzuk meg a legnagyobb távolságot, ha van!

Specifikáció:

Be: n∈N, mag∈N[1..n]

Ki: van∈L, k∈N, v∈N

Fv: szigetkezdet, szigetvég...

Fv: keresvége:N->L x N,

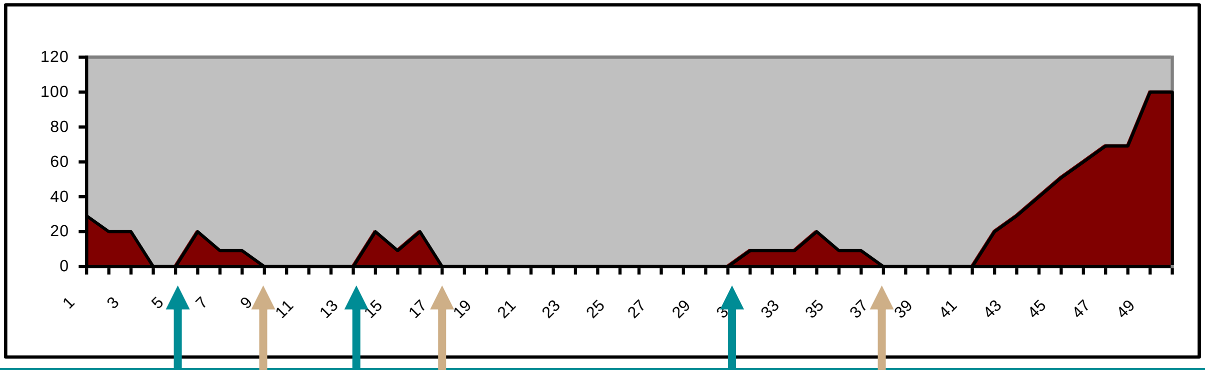
    keresvége(i)=**KERES**(j=i..n, szigetvég(j))

Fv: **táv**:N->N, táv(i)=keresvége(i).ind - i

Ef: -

Uf: (van,k,)=**MAX**(i=1..n, táv(i), szigetkezdet(i) és keresvége(i).van) és

    van -> v=keresvége(k).ind



Kis variáció…

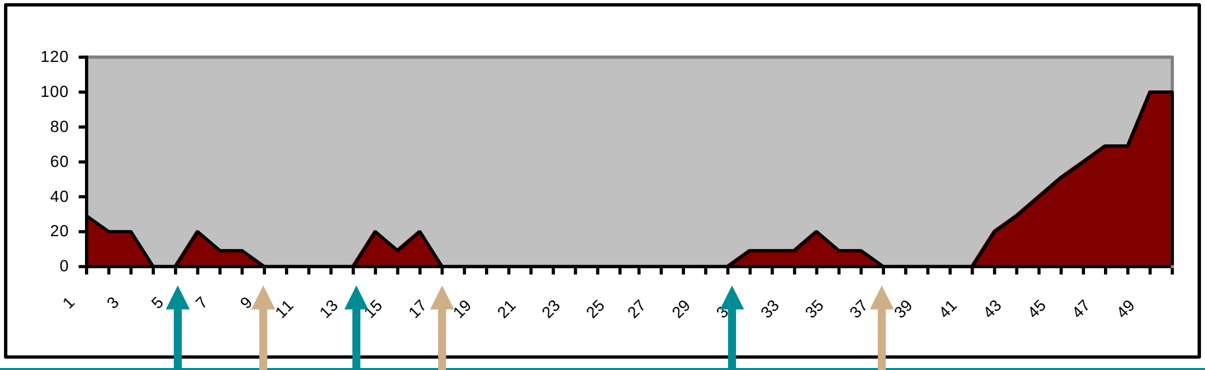
1. Válogassuk ki azokat a szigetkezdeteket, amelyeknek van vége is! (összevonható)
2. Másolással határozzuk meg hozzájuk a szigetvégeket! (összevonható)
3. Másolással határozzuk meg a távolságokat! (összevonható)
4. Határozzuk meg a legnagyobb távolságot, ha van!

Specifikáció:

```
Be: n∈N, mag∈N[1..n]
Ki: van∈L, k∈N, v∈N
Fv: szigetkezdet, szigetvég...
Fv: kereseleje:N->L x N,
    kereseleje(i)=KERES(j=-i..-1, szigetkezdet(-j))
Ef: -
Uf: (van,v,max)=MAX(i=1..n, i - -kereseleje(i).ind,
                            szigetvég(i) és kereseleje(i).van) és
    van -> k=v-max+1
```

Hátulról keresés!

Hátulról keresés miatt a negált értéket kell visszaadnunk!



Visszavezetés

```
Fv: kereseleje:N->L x N,
    kereseleje(i)=KERES(j=-i..-1, szigetkezdet(-j))
Uf: (van,v,max)=MAX(i=1..n, i - -kereseleje(i).ind,
                        szigetvég(i) és kereseleje(i).van) és
    van -> k=v-max+1
```

Feltételes maximumkeresés

```
maxind   ~  v
maxért   ~  max
e..u     ~  1..n
f(i)     ~  i+kereseleje(i).ind
T(i)     ~  szigetvég(i) és
            kereseleje(i).van
```

Keresés (kereseleje)

```
i     ~ j
e..u  ~ -i..-1
T(i)  ~ szigetkezdet(-j)
```

Algoritmus

Feltételes maximumkeresés

```
maxind  ~  v
maxért  ~  max
e..u    ~  1..n
f(i)    ~  i+kereseleje(i).ind
T(i)    ~  szigetvég(i) és
           kereseleje(i).van
```

Keresés (kereseleje)

```
i     ~ j
e..u  ~ -i..-1
T(i)  ~ szigetkezdet(-j)
```

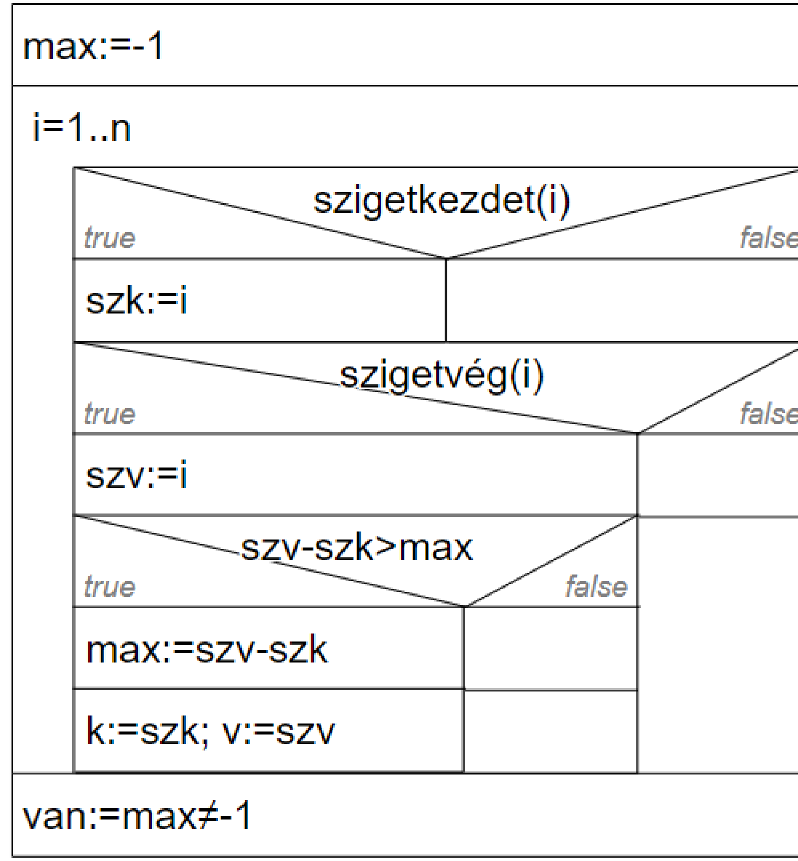
kereseleje(i:Egész):(Logikai,Egész)
Vált ind:Egész, van:Logikai

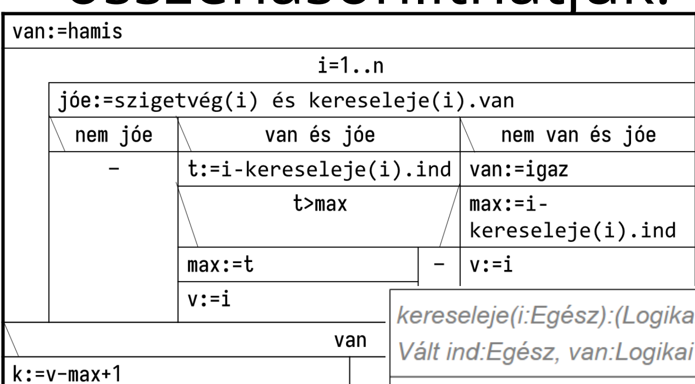
```
ind:=-i
ind<=-1 és nem szigetkezdet(-ind)
    ind:=ind+1
van:=ind<=-1
kereseleje:=(van,ind)
```

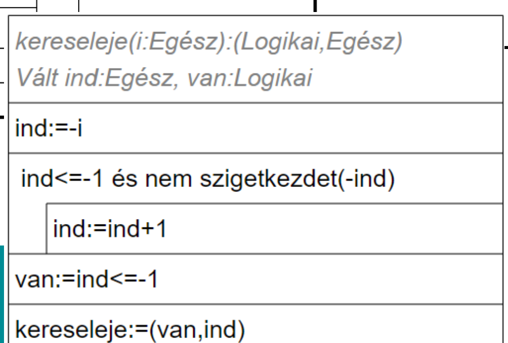
```
van:=hamis
i=1..n
    jóe:=szigetvég(i) és kereseleje(i).van
    nem jóe:
        -
    van és jóe:
        t:=i-kereseleje(i).ind
        t>max
            igen: max:=t
                  v:=i
            nem:  -
    nem van és jóe:
        van:=igaz
        max:=i-
        kereseleje(i).ind
        v:=i
van
    igen: k:=v-max+1
    nem:  -
```

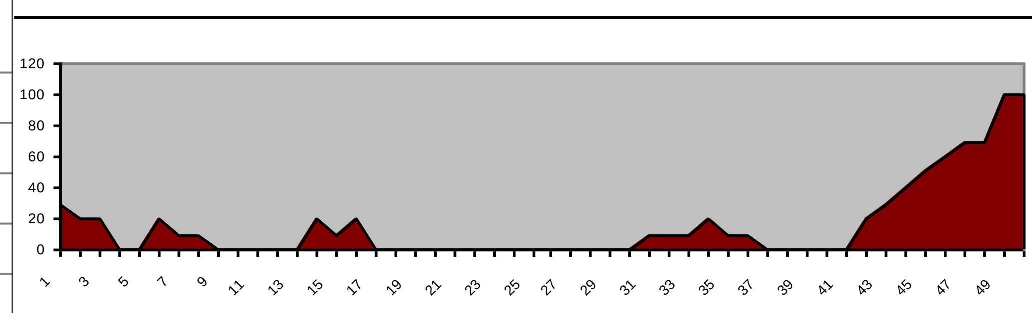
Algoritmikus gondolkodással

Ötlet: induljunk az elejétől, ha szigetkezdetet találunk, jegyezzük meg, ha véget, akkor számolhatjuk a sziget hosszát, és az eddigi maximummal összehasonlíthatjuk.





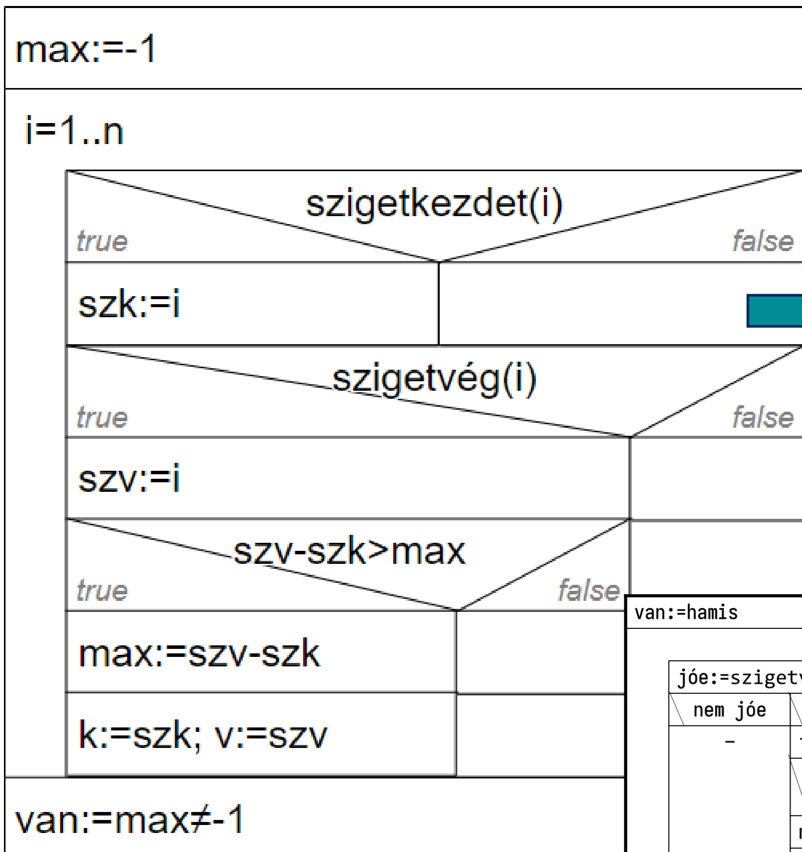


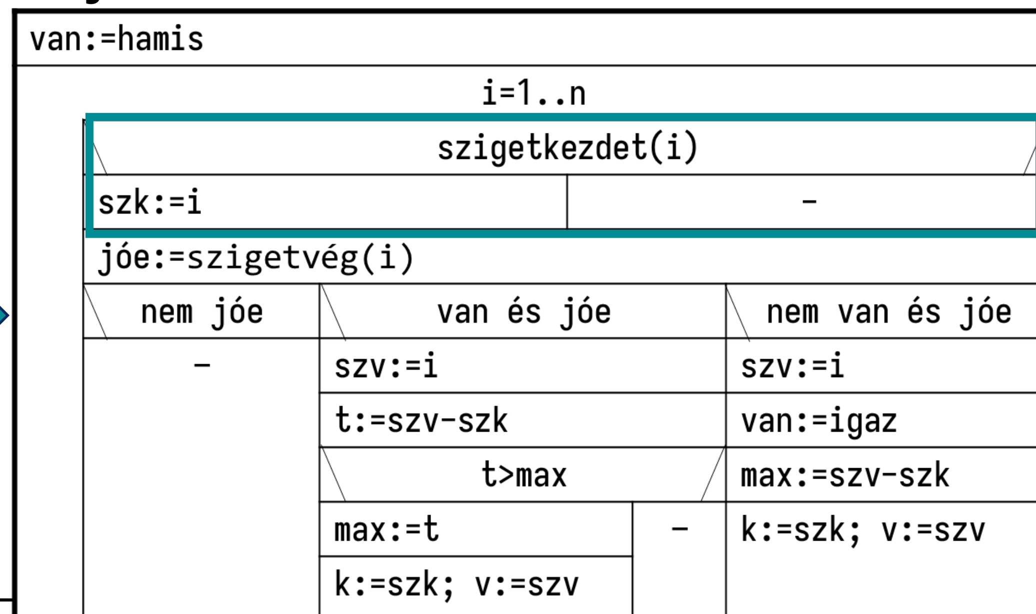


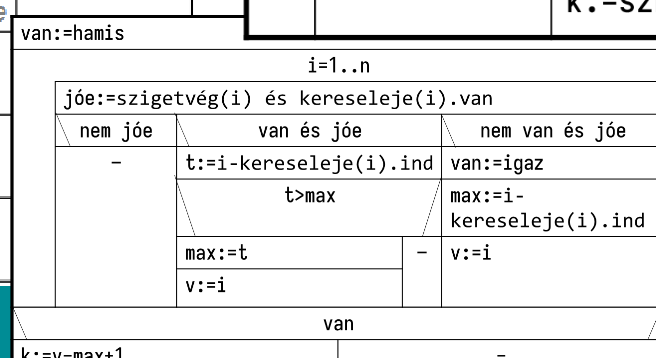
ELTE | IK

Programtranszformáció: közelítés

Mindkét megoldás feltételes maximumkeresés. Írjuk át a naív megoldásunkat struktúrájában ahhoz hasonlóvá!



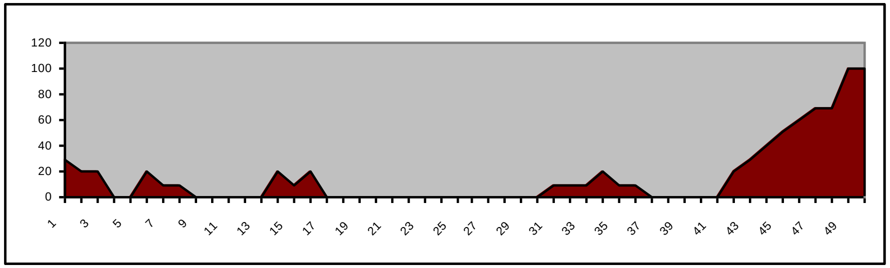




Eredeti

További közelítés: algoritmikus absztrakció – rekurzív függvény

Ötlet: Próbáljuk a kereseleje függvényt rekurzívan felírni! Ez a függvény minden pontban megmondja az adott ponthoz tartozó szigetkezdetet (ha van).



Specifikáció:

```
Fv: kereseleje:N->N,
    kereseleje(i)={i, ha szigetkezdet(i);
                   0, ha i < 1;
                   kereseleje(i-1) egyébként}
Uf: (van,v,max)=MAX(i=1..n, i-kereseleje(i),
                        szigetvég(i) és kereseleje(i)>0) és
    van -> k=v-max+1
```

Rekurzív függvény

Egyetlen feltételes maximumkeresés

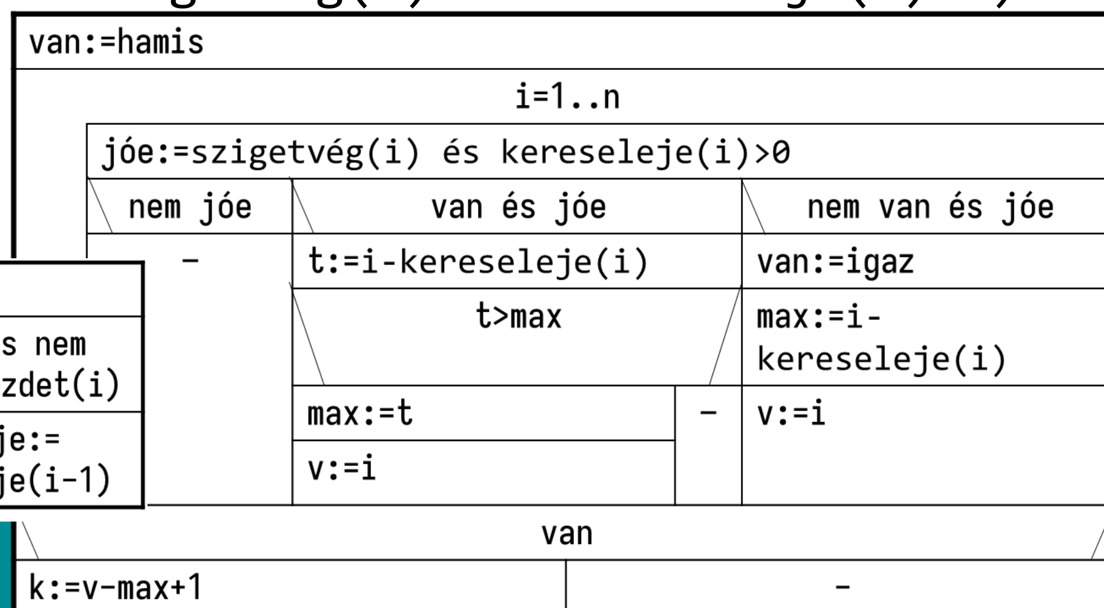
Visszavezetés

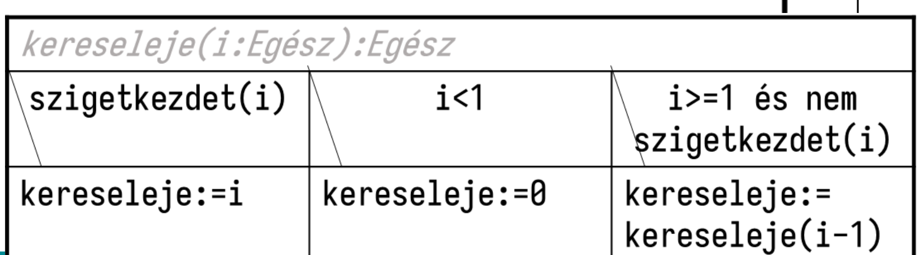
Feltételes maximumkeresés

maxind	~	v
maxért	~	max
e..u	~	1..n
f(i)	~	i-kereseleje(i)
T(i)	~	szigetvég(i) és kereseleje(i)>0

Specifikáció:

```
Fv: kereseleje:N->N,
    kereseleje(i)={i, ha szigetkezdet(i);
                   0, ha i < 1;
                   kereseleje(i-1) egyébként}
Uf: (van,v,max)=MAX(i=1..n, i-kereseleje(i),
                        szigetvég(i) és kereseleje(i)>0) és
    van -> k=v-max+1
```





Rekurzió átírása

kereseleje(i:Egész):Egész

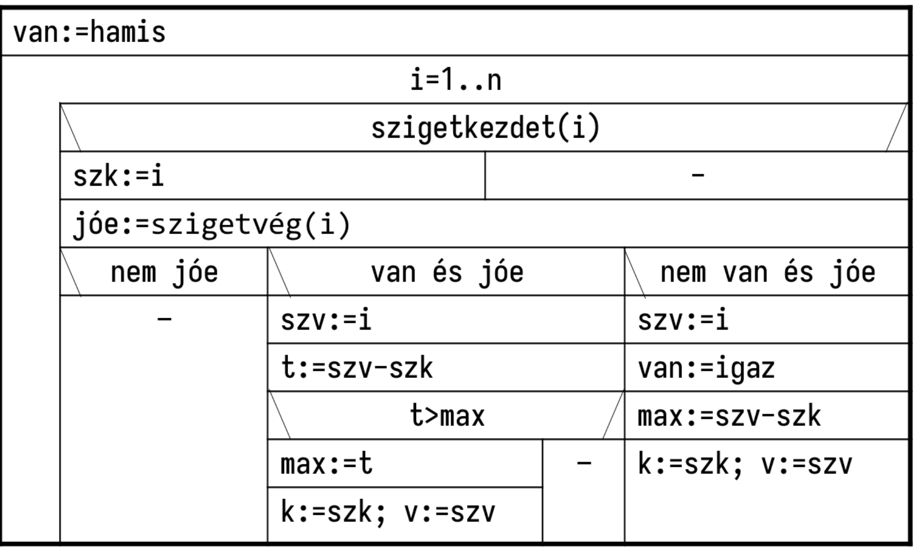
szigetkezdet(i)	i<1	i>=1 és nem szigetkezdet(i)
kereseleje:=i	kereseleje:=0	kereseleje:= kereseleje(i-1)

```
van:=hamis; szk:=0
i=1..n
    szigetkezdet(i)
        szk:=i              | szk:=szk
    jóe:=szigetvég(i) és szk>0
    nem jóe | van és jóe            | nem van és jóe
    -       | t:=i-szk              | van:=igaz
            | t>max                 | max:=i-szk
            |   max:=t   | -        | v:=i
            |   v:=i     |          |
van
    k:=v-max+1              | -
```

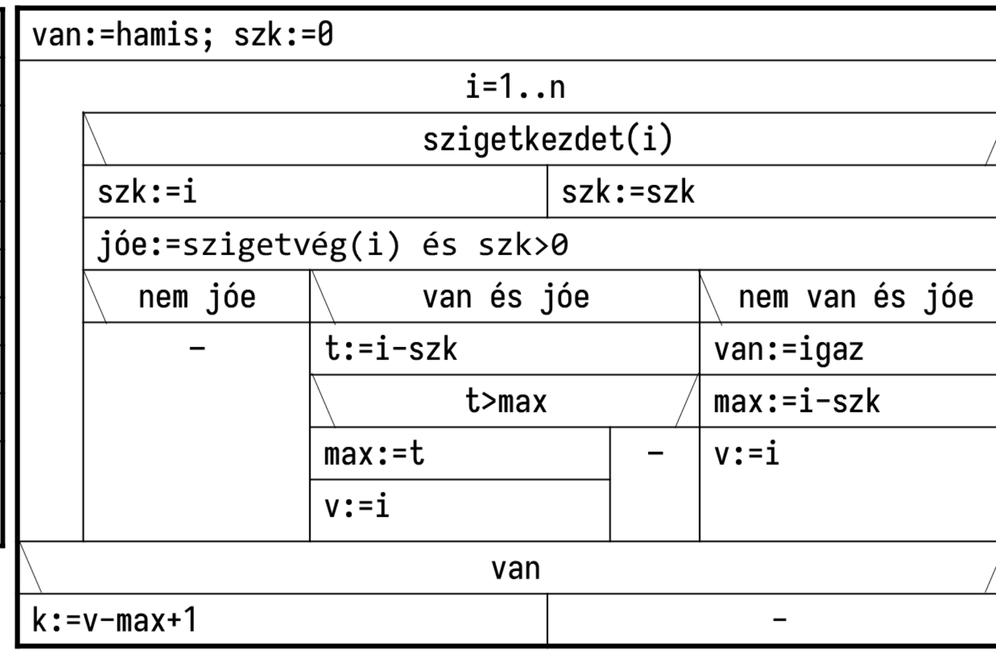
szk=kereseleje(i-1)

Összehasonlítás

Algoritmikus gondolkodással

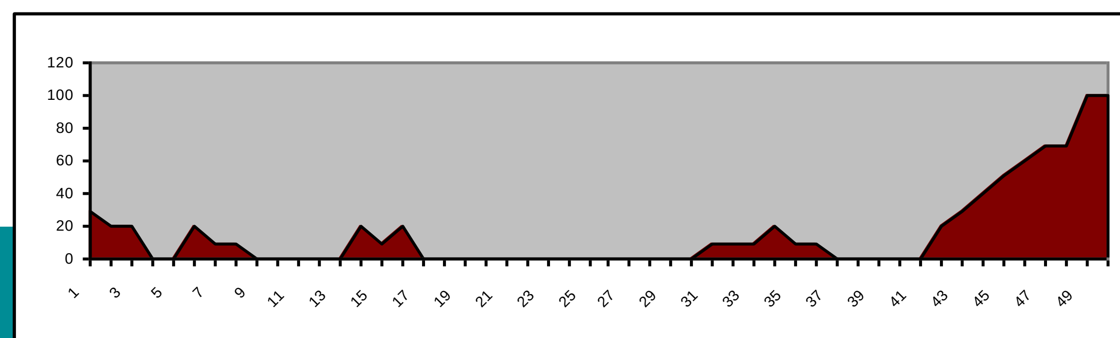


Rekurzív függvénnyel

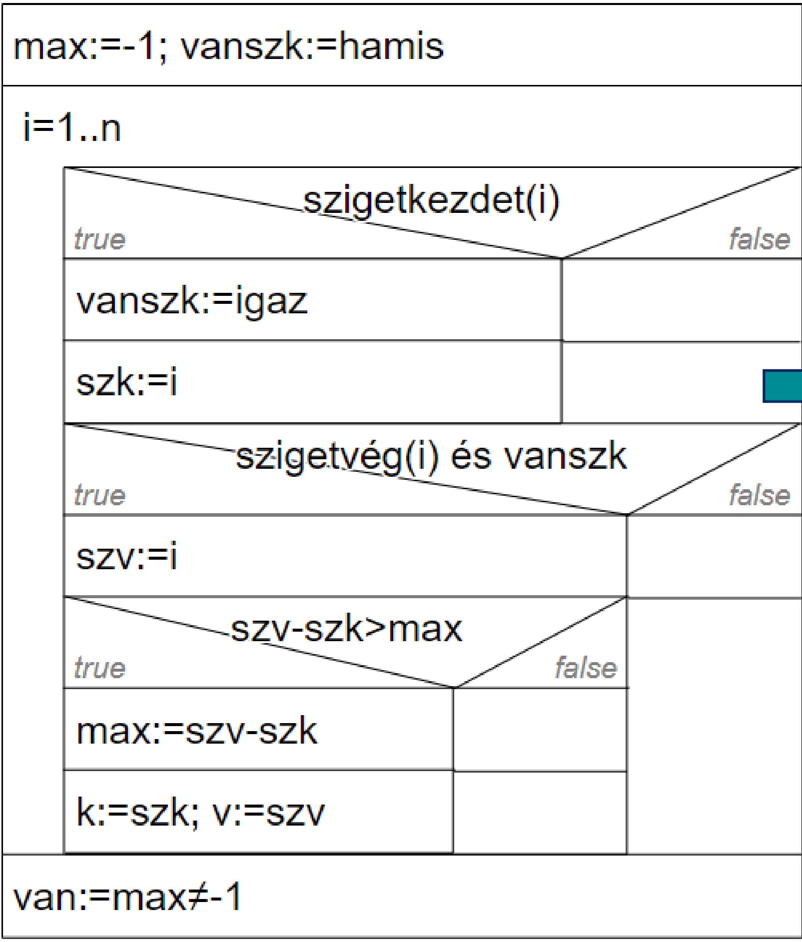


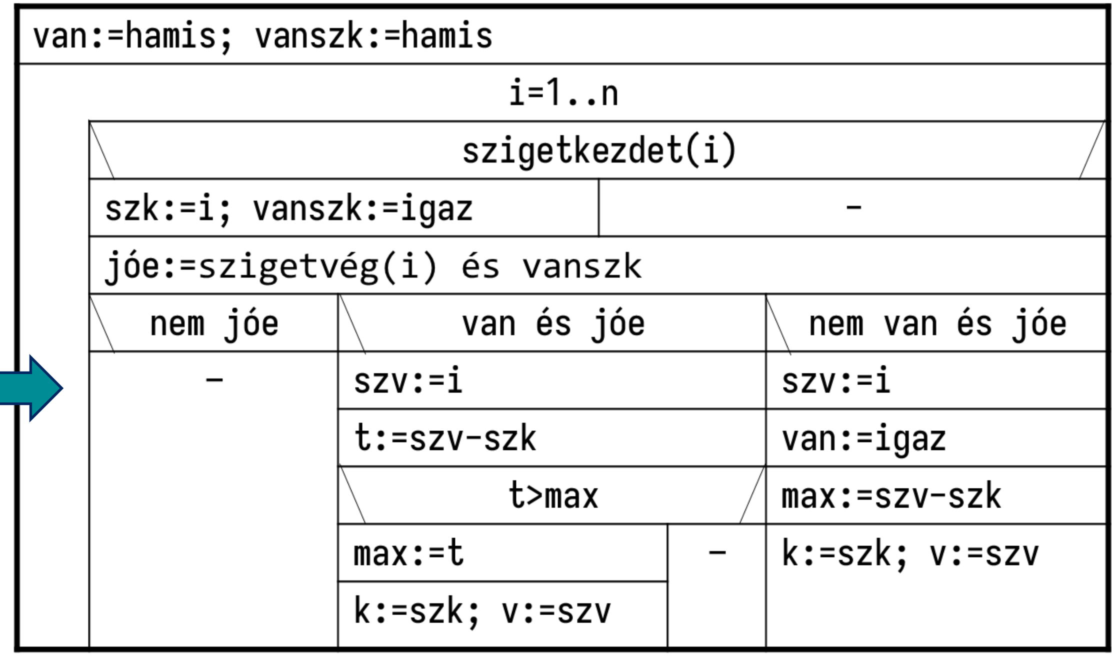
Funkcionálisan ugyanaz. Sőt!

Kiderült, hogy a bal oldali ROSSZ!



Újra: algoritmikus gondolkodással





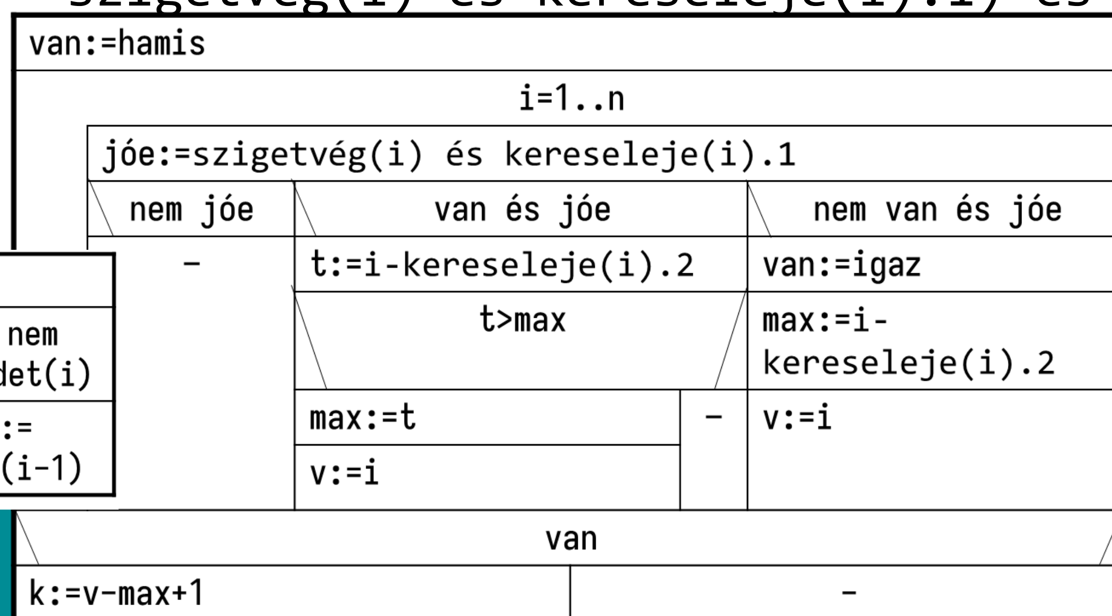
Újra: visszavezetés

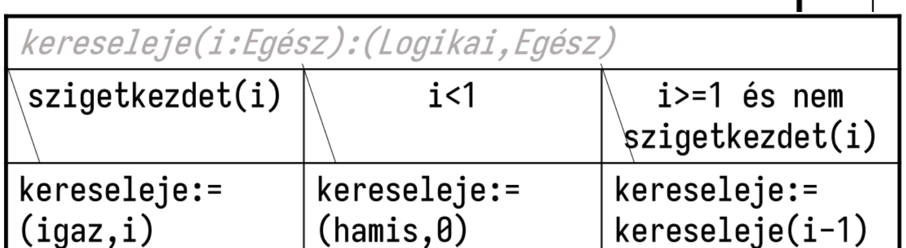
Feltételes maximumkeresés

```
maxind   ~  v
maxért   ~  max
e..u     ~  1..n
f(i)     ~  i-kereseleje(i).2
T(i)     ~  szigetvég(i) és
            kereseleje(i).1
```

Specifikáció:

```
Fv: kereseleje:N->L x N,
    kereseleje(i)={(igaz, i), ha szigetkezdet(i);
                   (hamis,0), ha i < 1;
                   kereseleje(i-1) egyébként}
Uf: (van,v,max)=MAX(i=1..n, i-kereseleje(i).2,
                    szigetvég(i) és kereseleje(i).1) és
    van -> k=v-max+1
```





Újra: rekurzió átírása

kereseleje(i:Egész):(Logikai,Egész)		
szigetkezdet(i)	i<1	i>=1 és nem szigetkezdet(i)
kereseleje:=(igaz,i)	kereseleje:=(hamis,0)	kereseleje:=kereseleje(i-1)

```
van:=hamis; vanszk:=hamis; szk:=0;
i=1..n                    (vanszk,szk)=kereseleje(i-1)
    szigetkezdet(i)
        vanszk:=igaz; szk:=i  |  vanszk:=vanszk; szk:=szk
    jóe:=szigetvég(i) és vanszk
    nem jóe | van és jóe        | nem van és jóe
    -       | t:=i-szk          | van:=igaz
            | t>max             | max:=i-szk
            |   max:=t    | -   | v:=i
            |   v:=i      |     |
van
    k:=v-max+1  |  -
```

Újra: rekurzió átírása

kereseleje(i:Egész):(Logikai,Egész)

szigetkezdet(i)	i<1	i>=1 és nem szigetkezdet(i)
kereseleje:=(igaz,i)	kereseleje:=(hamis,0)	kereseleje:=kereseleje(i-1)

van:=hamis; vanszk:=hamis; szk:=0;

i=1..n

(vanszk,szk)=kereseleje(i-1)

szigetkezdet(i)	
vanszk:=igaz; szk:=i	vanszk:=vanszk; szk:=szk

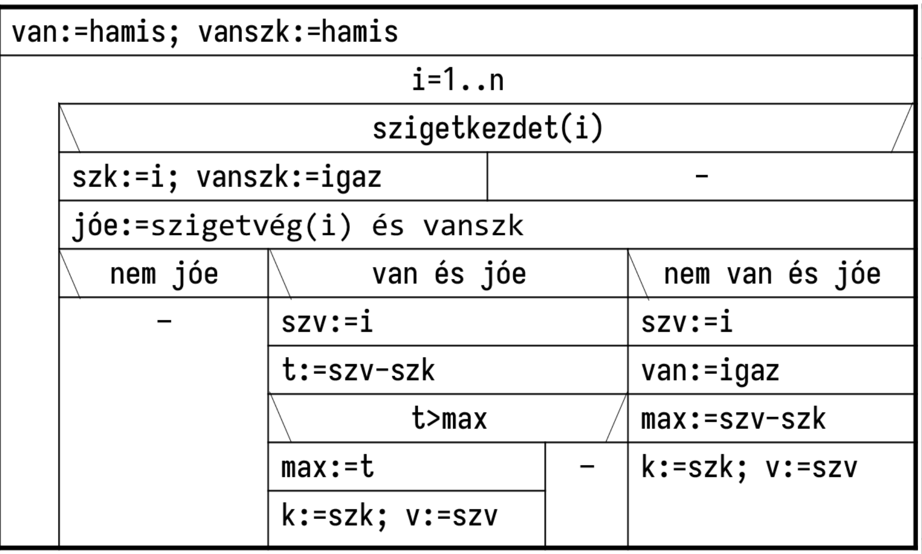
jóe:=szigetvég(i) és vanszk

nem jóe	van és jóe	nem van és jóe
–	t:=i-szk	van:=igaz
	t>max: max:=t; v:=i / –	max:=i-szk
		v:=i

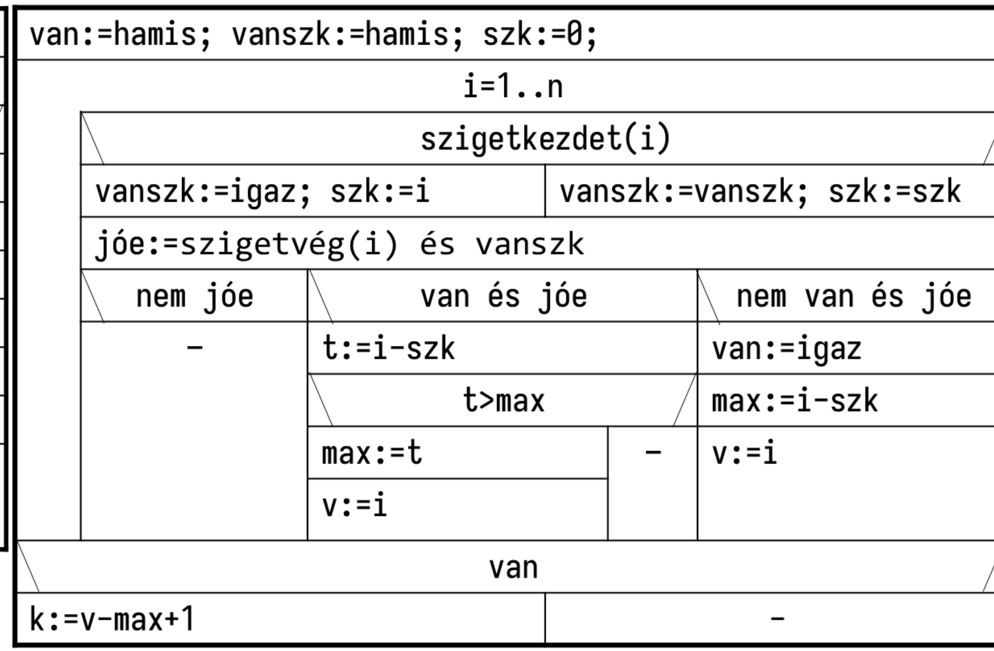
van	
k:=v-max+1	–

Újra: összehasonlítás

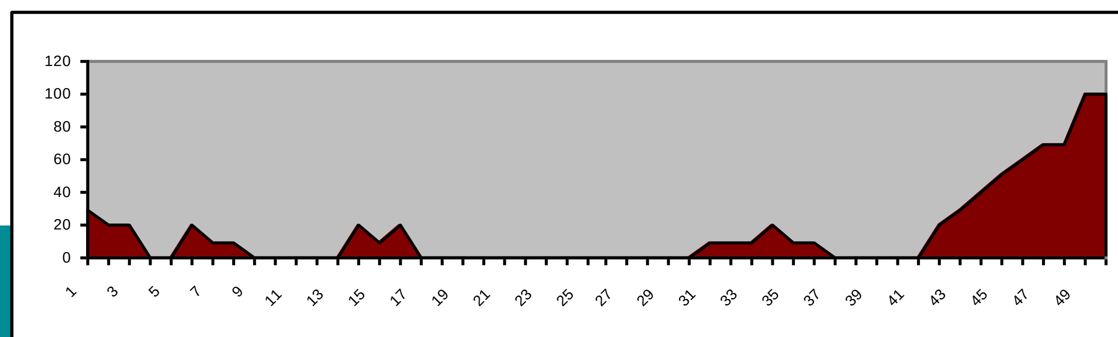
Algoritmikus gondolkodással



Rekurzív függvénnyel

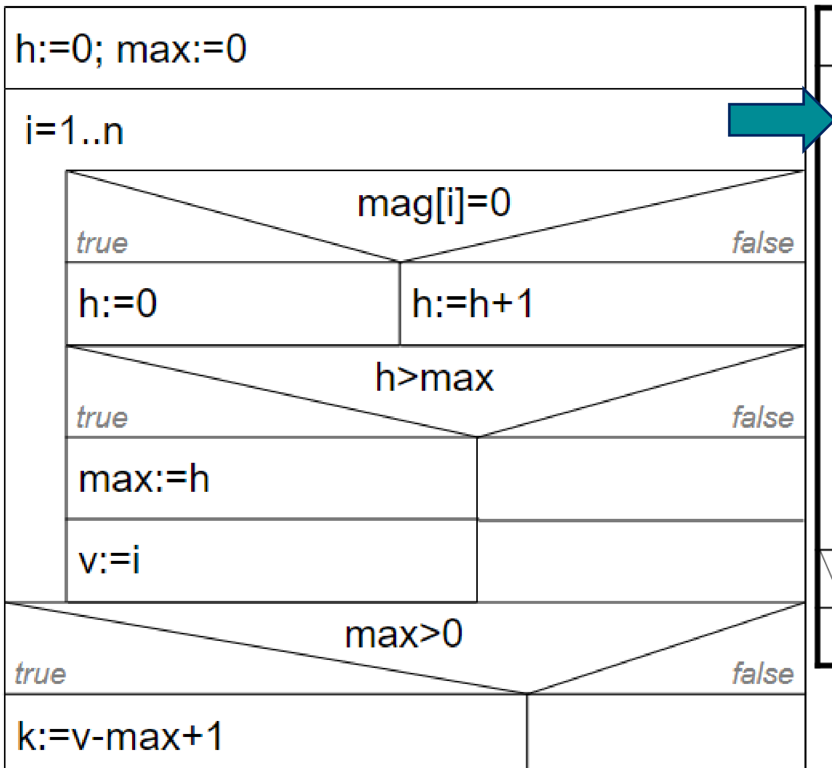


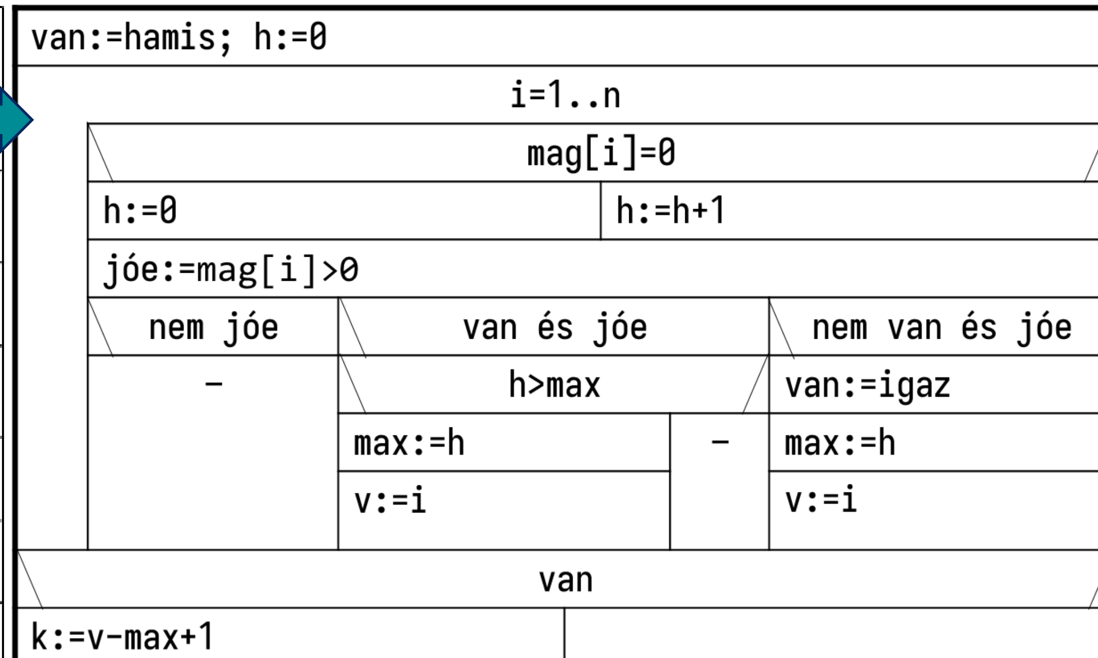
Funkcionálisan ugyanaz!

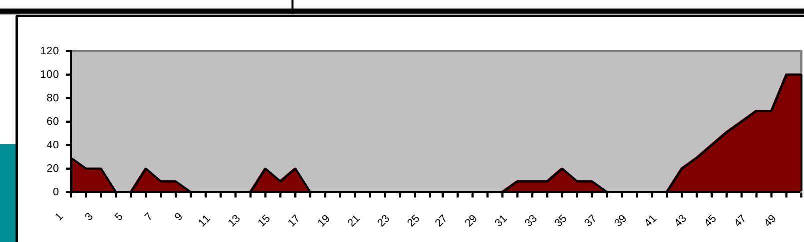


Másképp: algoritmikus gondolkodással

Ötlet: ha szárazföld fölött vagyunk, akkor növeljünk egy változót, és ennek a maximuma kell!







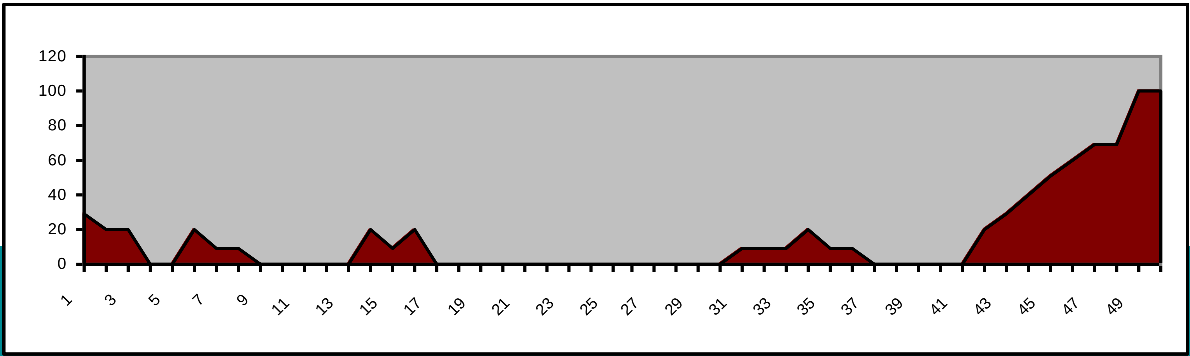
Másképp: rekurzív függvény

Ötlet: vezessünk be egy függvényt, amely minden pontban megmondja, hogy mekkora a távolság a szigetkezdet óta (tengernél 0). Hol veszi fel ez a legnagyobb értékét?

Specifikáció:

```
Fv: hossz:N->N, hossz(i)={0, ha i<1;
                          0, ha mag[i]=0;
                          táv(i-1)+1 egyébként}
Uf: (van,v,max)=MAX(i=1..n, hossz(i), mag[i]>0) és
    van -> k=v-max+1
```

Rekurzív függvény



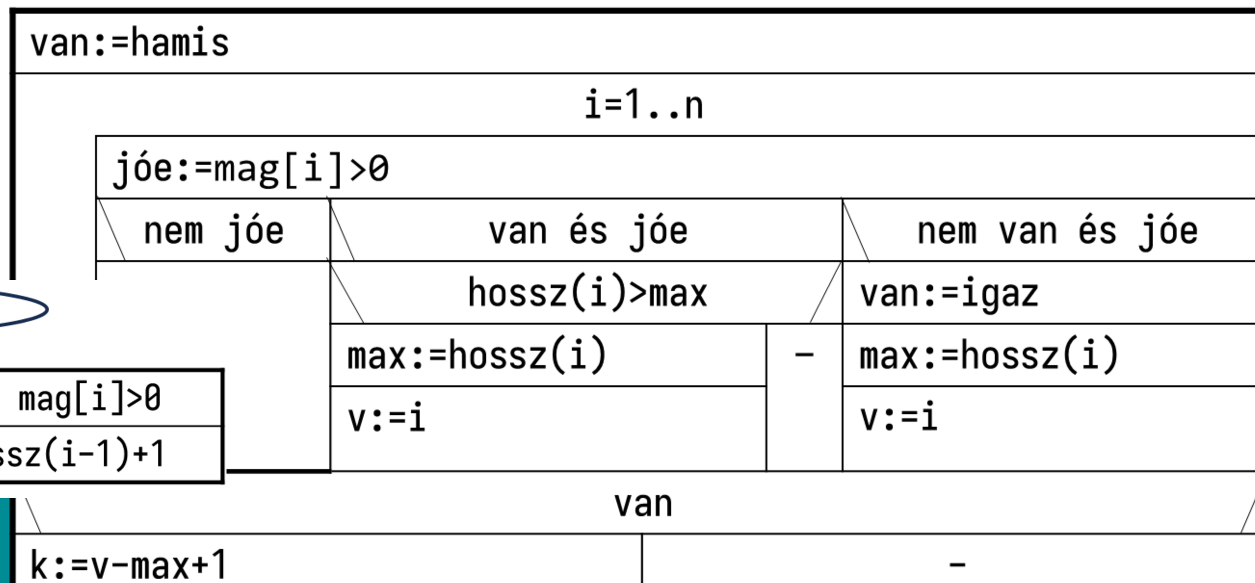
Másképp: visszavezetés

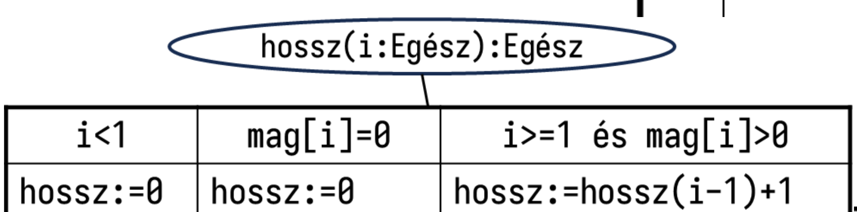
Feltételes maximumkeresés

```
maxind  ~  v
maxért  ~  max
e..u    ~  1..n
f(i)    ~  hossz(i)
T(i)    ~  mag[i]>0
```

Specifikáció:

```
Fv: hossz:N->N, hossz(i)={0, ha i<1;
                          0, ha mag[i]=0;
                          táv(i-1)+1 egyébként}
Uf: (van,v,max)=MAX(i=1..n, hossz(i), mag[i]>0) és
    van -> k=v-max+1
```

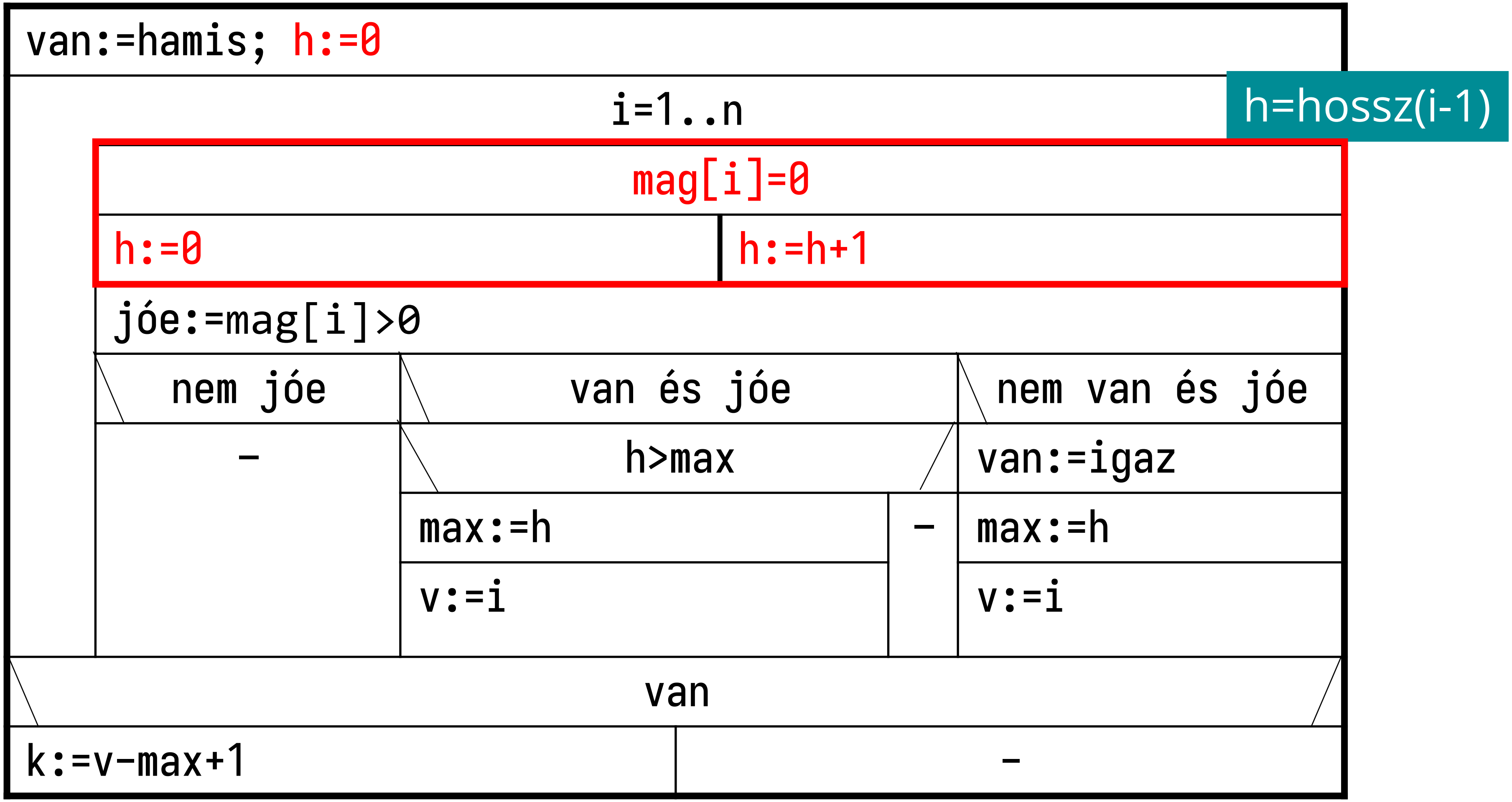




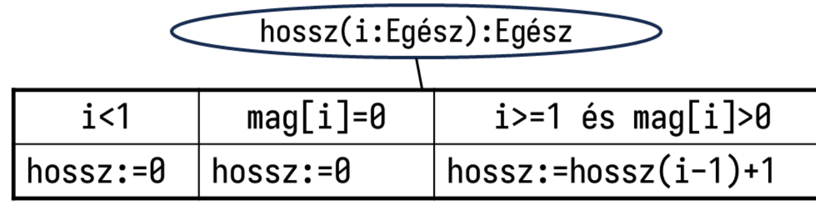
Másképp: rekurzió átírása

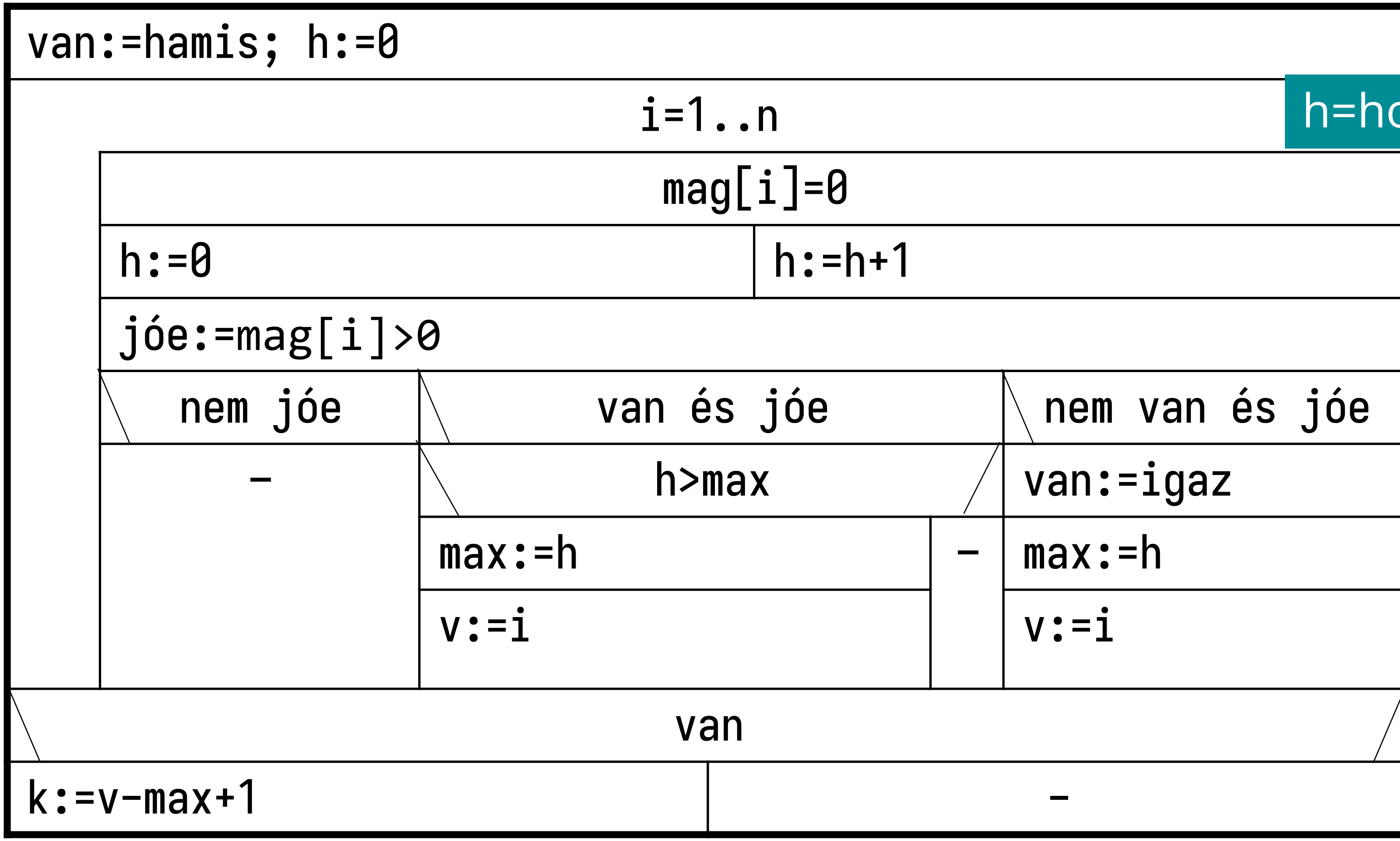
hossz(i:Egész):Egész

i<1	mag[i]=0	i>=1 és mag[i]>0
hossz:=0	hossz:=0	hossz:=hossz(i-1)+1



Másképp: rekurzió átírása

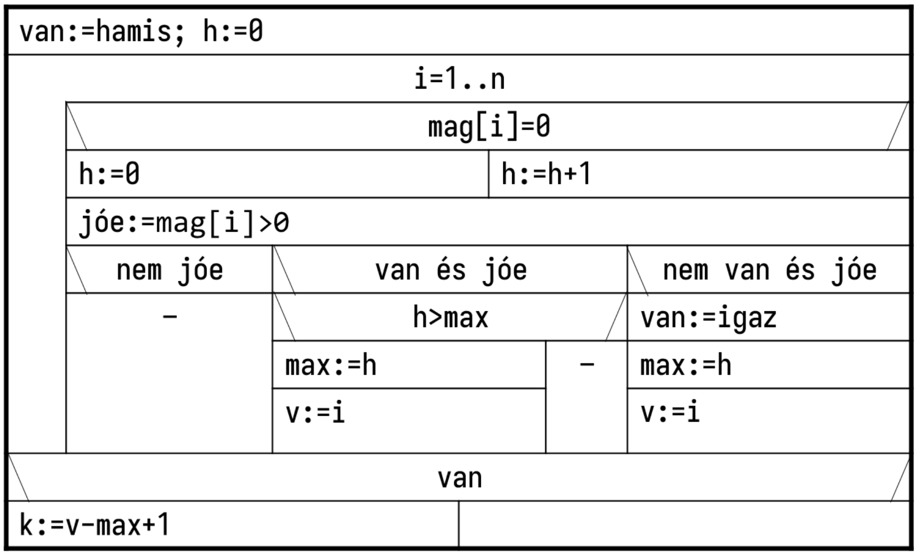




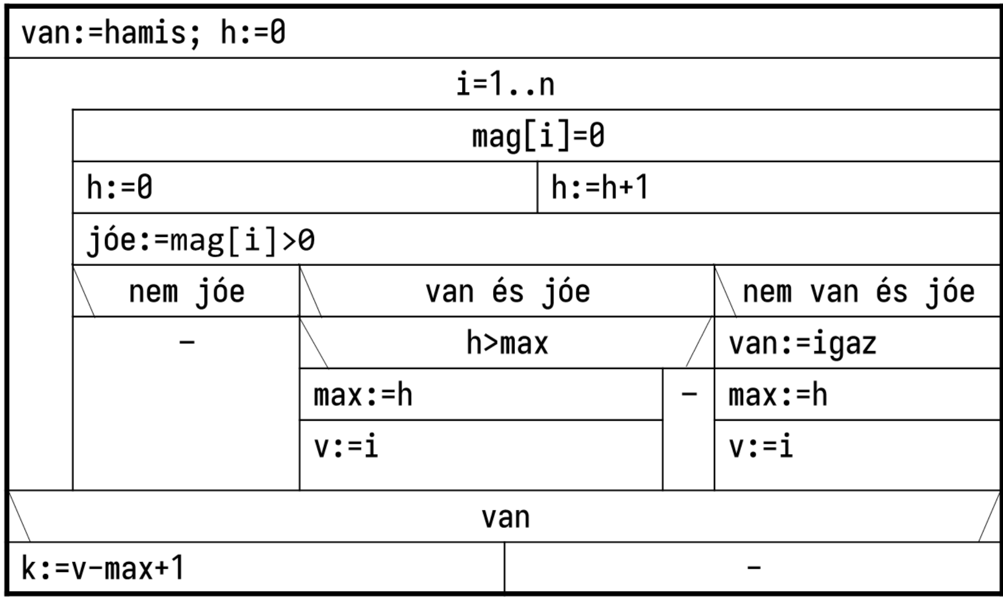
h=hossz(i-1)

Újra: összehasonlítás

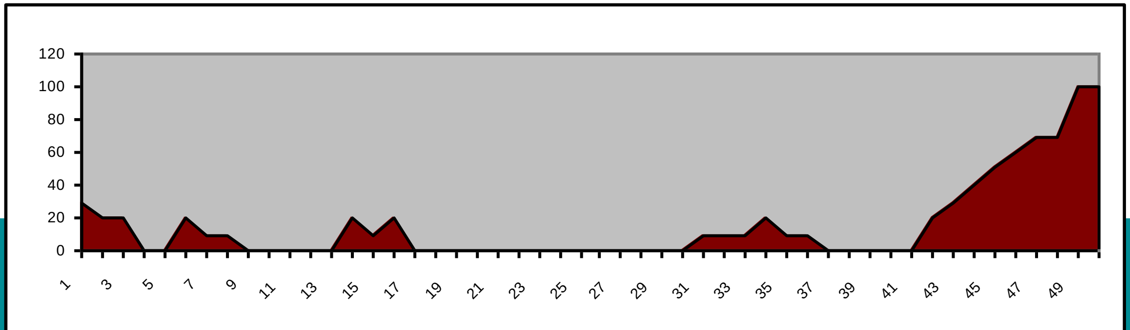
Algoritmikus gondolkodással



Rekurzív függvénnyel



Ez ugyanaz!!!



Rekurzív megoldás

Mindenhol, ahol „közben” megjegyzünk, gyűjtögetünk, és az előzővel ki tudjuk fejezni rekurzívan, ld. pl. az általánosított összegzést

Halmazok

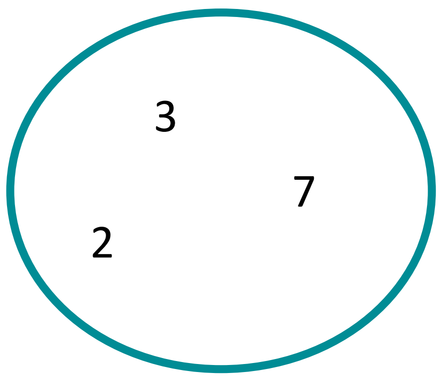
(dinamikus) tömbben

Halmazok

- Minden elem egyszer szerepel benne
- Ábrázolás
  - (Dinamikus) tömbben
  - Logikai vektorban

1	2	3
3	7	2

1	2	3	4	5	6	7	8	9	10
h	i	i	h	h	h	i	h	h	h



HalmazE

1	2	3
3	7	2

HalmazE: Minden elem egyszer szerepel-e?

Másképp: Minden elemre igaz, hogy előtte nem szerepel önmaga?

Specifikáció:

```
Be: n∈N, h∈Z[1..n]
Ki: halmaze∈L
Fv: nincs:N->L, nincs(i)=MIND(j=1..i-1, h[i]≠h[j])
Ef: -
Uf: halmaze=MIND(i=2..n, nincs(i))
```

ElemeE

1	2	3
3	7	2

ElemeE: Egy adott érték benne van-e a tömbben?

Specifikáció:

```
Be: n∈N, h∈Z[1..n], e∈Z
Ki: elemee∈L
Ef: -
Uf: elemee=VAN(i=1..n, h[i]=e)
```

HalmazÉpít

1	2	3
3	7	2

HalmazÉpít: Egy „sima” tömböt halmazzá alakít.

Másképp: Válogassuk az egyszer szereplő elemeket (akik előtt nincs önmaguk).

Specifikáció:

```
Be: n∈N, t∈Z[1..n]
Ki: db∈N, h∈Z[1..db]
Fv: nincs:N->L, nincs(i)=MIND(j=1..i-1, t[i]≠t[j])
Ef: -
Uf: (db,h)=KIVÁLOGAT(i=1..n, nincs(i), t[i])
```

Metszet

1	2	3
3	7	2

1	2	3	4	5
1	4	5	2	7

Metszet: Vegyük két halmaz azon elemeit, amelyek az egyikben és a másikban is benne vannak!

Specifikáció:

```
Be: n∈N, x∈Z[1..n], m∈N, y∈Z[1..m]
Ki: db∈N, metszet∈Z[1..db]
Ef: HalmazE(x) és HalmazE(y)
Uf: HalmazE(metszet) és ∀i∈[1..db]:(
        ElemeE(x, metszet[i]) és ElemeE(y, metszet[i]))
```

Metszet

1	2	3
3	7	2

1	2	3	4	5
1	4	5	2	7

Metszet: Vegyük két halmaz azon elemeit, amelyek az egyikben és a másikban is benne vannak!

Másképp: Válogassuk ki az egyik tömbből azokat az elemeket, amelyek a másikban is benne vannak!

Specifikáció:

```
Be: n∈N, x∈Z[1..n], m∈N, y∈Z[1..m]
Ki: db∈N, metszet∈Z[1..db]
Fv: benne:N->L, benne(i)=VAN(j=1..m, x[i]=y[j])
Ef: HalmazE(x) és HalmazE(y)
Uf: (db,metszet)=KIVÁLOGAT(i=1..n, benne(i), x[i])
```

Unió

1	2	3
3	7	2

1	2	3	4	5
1	4	5	2	7

Unió: Vegyük két halmaz azon elemeit, amelyek vagy az egyikben, vagy a másikban vannak!

Specifikáció:

```
Be: n∈N, x∈Z[1..n], m∈N, y∈Z[1..m]
Ki: db∈N, unió∈Z[1..db]
Ef: HalmazE(x) és HalmazE(y)
Uf: HalmazE(unió) és ∀i∈[1..db]:(
        ElemeE(x, unió[i]) vagy ElemeE(y, unió[i])
```

Unió

1	2	3
3	7	2

1	2	3	4	5
1	4	5	2	7

Másképp: Vegyük az egyik tömböt, majd válogassuk ki a másikból azokat az elemeket, amelyek az egyikből hiányoznak, és fűzzük a végére!

Specifikáció:

```
Be: n∈N, x∈Z[1..n], m∈N, y∈Z[1..m]
Ki: db∈N, unió∈Z[1..db]
Sa: deltadb∈N, delta∈Z[1..deltadb]
Fv: benne:N->L, benne(i)=VAN(j=1..n, y[i]=x[j])
Ef: HalmazE(x) és HalmazE(y)
Uf: (deltadb,delta)=KIVÁLOGAT(i=1..m, nem benne(i), y[i]) és
    db=n+deltadb és
    ∀i∈[1..n]:(unió[i]=x[i]) és
    ∀i∈[1..deltadb]:(unió[n+i]=delta[i])
```

Unió

1	2	3
3	7	2

1	2	3	4	5
1	4	5	2	7

Másképp: Vegyük az egyik tömböt, majd válogassuk ki a másikból azokat az elemeket, amelyek az egyikből hiányoznak, és fűzzük a végére!

Specifikáció:

```
Be: n∈N, x∈Z[1..n], m∈N, y∈Z[1..m]
Ki: db∈N, unió∈Z[1..db]
Sa: deltadb∈N, delta∈Z[1..deltadb]
Fv: benne:N->L, benne(i)=VAN(j=1..n, y[i]=x[j])
Fv: f:N->Z, f(i)={x[i], ha 1<=i<=n;
                  delta[i-n] egyébként}
Ef: -
Uf: (deltadb,delta)=KIVÁLOGAT(i=1..m, nem benne(i), y[i]) és
    db=n+deltadb és unió=MÁSOL(i=1..db, f(i))
```

Halmazba

1	2	3
3	7	2

Halmazba: Egy elem betevése a halmazba.

Másképp: Ha nincs a tömbben, tegyük a tömb végére.

Specifikáció:

```
Be: n∈N, h∈Z[1..n], e∈Z
Ki: n'∈N, h'∈Z[1..n']
Fv: eleme:Z->L, eleme(e)=VAN(i=1..n, h[i]=e)
Ef: HalmazE(h)
Uf: eleme(e) -> (n'=n és h'=h) és
    nem eleme(e) ->
          (n'=n+1 és ∀i∈[1..n]:(h[i]=h'[i]) és h'[n']=e)
```

Halmazba

1	2	3
3	7	2

Halmazba: Egy elem betevése a halmazba.

Másképp: Ha nincs a tömbben, tegyük a tömb végére.

Állapottér specifikáció:

```
A= (n:N, h:Z[1..n], e:Z)
Ef=(n=n' és h=h' és e=e')
Uf=(benne(e) -> Ef és
    nem benne(e) ->
             (n=n'+1 és ∀i∈[1..n]:(h[i]=h'[i]) és h[n]=e))

Fv: benne:Z->L, benne(e)=VAN(i=1..n, h[i]=e)
```

Halmazból

1	2	3
3	7	2

Halmazba: Egy elem kivétele a halmazból.

Másképp: Ha benne van a tömbben, a tömb utolsó elemét tegyük a helyére.

Specifikáció:

```
Be: n∈N, h∈Z[1..n], e∈Z
Ki: n'∈N, h'∈Z[1..n']
Sa: van∈L, ind∈N
Fv: eleme:Z->L x N, eleme(e)=KERES(i=1..n, h[i]=e)
Ef: -
Uf: (van,ind)=eleme(e) és
    nem van -> (n'=n és h'=h) és
    van     -> (n'=n-1 és
                ∀i∈[1..ind-1]:(h'[i]=h[i]) és
                h'[ind]=h[n] és
                ∀i∈[ind+1..n-1]:(h'[i]=h[i]))
```

Halmazok

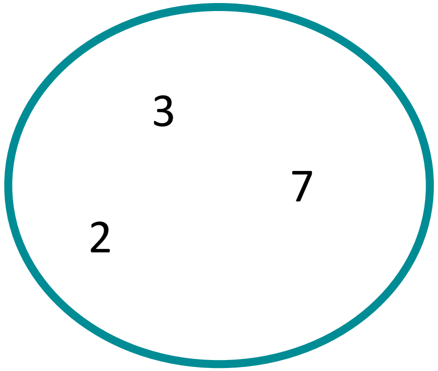
logikai vektor

Halmazok

- Minden elem egyszer szerepel benne
- Ábrázolás
  - (Dinamikus) tömbben
  - Logikai vektorban

1	2	3
3	7	2

1	2	3	4	5	6	7	8	9	10
h	i	i	h	h	h	i	h	h	h



ElemeE

1	2	3
3	7	2

1	2	3	4	5	6	7	8	9	10
h	i	i	h	h	h	i	h	h	h

ElemeE: Egy adott érték benne van-e a tömbben?

Specifikáció:

```
Be: n∈N, h∈L[1..n], e∈[1..n]
Ki: elemee∈L
Ef: -
Uf: elemee=h[e]
```

HalmazÉpít

1	2	3	4	5	6	7	8	9	10
h	i	i	h	h	h	i	h	h	h

HalmazÉpít: Egy „sima” tömböt halmazzá alakít.

Másképp: Azok az elemek igazak, amelyek szerepelnek a tömbben.

Specifikáció:

```
Be: tdb∈N, t∈E[1..tdb], n∈N, E=[1..n]
Ki: h∈L[1..n]
Fv: benne:N->L, benne(i)=VAN(j=1..tdb, t[j]=i)
Ef: -
Uf: h=MÁSOL(i=1..n, benne(i))
```

Metszet

Metszet: Vegyük két halmaz azon elemeit, amelyek az egyikben és a másikban is benne vannak!

Specifikáció:

```
Be: n∈N, x∈L[1..n], y∈L[1..n]
Ki: metszet∈L[1..n]
Ef: -
Uf: metszet=MÁSOL(i=1..n, x[i] és y[i])
```

1	2	3	4	5	6	7	8	9	10
h	h	i	h	h	h	i	h	i	h

1	2	3	4	5	6	7	8	9	10
h	i	i	h	h	h	i	h	h	h

Unió

Unió: Vegyük két halmaz azon elemeit, amelyek az egyikben vagy a másikban is benne vannak!

Specifikáció:

```
Be: n∈N, x∈L[1..n], y∈L[1..n]
Ki: unió∈L[1..n]
Ef: -
Uf: unió=MÁSOL(i=1..n, x[i] vagy y[i])
```

1	2	3	4	5	6	7	8	9	10
h	h	i	h	h	h	i	h	i	h

1	2	3	4	5	6	7	8	9	10
h	i	i	h	h	h	i	h	h	h

Halmazba

1	2	3	4	5	6	7	8	9	10
h	i	i	h	h	h	i	h	h	h

Halmazba: Az adott elemet állítsuk igazra, a többi változatlan!

Specifikáció:

```
Be: n∈N, h∈L[1..n], e∈[1..n]
Ki: h'∈L[1..n]
Fv: f:N->L, f(i)={h[i], ha i≠e;
                  igaz egyébként}
Ef: -
Uf: h'=MÁSOL(i=1..n, f(i))
Uf: ∀i∈[1..e-1]:(h'[i]=h[i]) és
    h'[e]=igaz és
    ∀i∈[e+1..n]:(h'[i]=h[i])
```

Halmazból

1	2	3	4	5	6	7	8	9	10
h	i	i	h	h	h	i	h	h	h

Halmazba: Az adott elemet állítsuk hamisra, a többi változatlan!

Specifikáció:

```
Be: n∈N, h∈L[1..n], e∈[1..n]
Ki: h'∈L[1..n]
Fv: f:N->L, f(i)={h[i], ha i≠e;
                  hamis egyébként}
Ef: -
Uf: h'=MÁSOL(i=1..n, f(i))
Uf: ∀i∈[1..e-1]:(h'[i]=h[i]) és
    h'[e]=hamis és
    ∀i∈[e+1..n]:(h'[i]=h[i])
```